

MAVLINK HUD FIELD MANUAL

OPERATOR'S HANDBOOK

VERSION 1.0GENERATED 2026

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

Table of Contents

Chapter 1: Build Guide

Chapter 2: Power Systems

Chapter 3: Flight Modes

Chapter 4: MAVLink Internals

Chapter 5: EKF Failsafes

Chapter 6: Control Architecture

Chapter 7: Custom Airframes

Chapter 8: Sensor Architecture

Chapter 9: Companion Computers

Chapter 10: ExpressLRS

Chapter 11: Navigation & Mission

Chapter 12: ChibiOS Integration

Chapter 13: Object Avoidance

Chapter 14: Advanced Tuning

Chapter 15: Log Analysis

Chapter 16: Build System

Chapter 17: Flashing & Cloning

Chapter 18: Remote ID

MISSION COMMAND REFERENCE

Chapter 19: Mission Scenarios

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





CHAPTER 1: BUILD GUIDE

FLIGHT CONTROLLER ARCHITECTURE: THE COMPUTE CORE

CRITICAL The Flight Controller (FC) is the central nervous system of the vehicle. In the ArduPilot ecosystem, the selection of the processor family dictates the architectural ceiling of the vehicle, influencing everything from loop rates to peripheral expandability. Unlike Betaflight, where an F7 is simply "faster," in ArduPilot, the processor architecture determines which *features* are physically capable of running.

1. PROCESSOR ARCHITECTURE AND THE STM32 HIERARCHY

The industry standard for flight control logic is the STMicroelectronics STM32 family. While marketing materials often emphasize clock speed (MHz), the critical constraints for ArduPilot integration are **Flash Memory**, **Floating Point Unit (FPU) Precision**, and **DMA Throughput**.

1.1 THE H743: THE MODERN STANDARD

- **Core:** Cortex-M7 @ 480 MHz.
- **Flash:** 2MB.
- **The "Double Precision" Advantage:**



The defining feature of the H743 is its hardware support for Double-Precision Floating Point math. ArduPilot's state estimator (EKF3) relies heavily on complex matrix operations to fuse GPS, Gyro, and Accelerometer data. On older processors (F4/F7), these calculations are often done in single-precision to save time, or emulated in software. The H7 can crunch these numbers natively in double-precision.

  - **Real World Impact:** This reduces "Integration Drift." Over a long autonomous mission (30+ minutes), minute rounding errors in position estimation accumulate. The H7 minimizes this, resulting in more accurate Return-to-Launch (RTL) performance and tighter loiter circles.
- **Memory Architecture:**
  The 2MB Flash is mandatory for the modern ArduPilot binary. The firmware size has grown significantly with the addition of

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

Lua Scripting, DDS (ROS2), and support for hundreds of sensors. 2MB ensures you never have to choose between "GPS Drivers" and "Scripting."

1.2 THE F765: THE FIXED-WING WORKHORSE

- **Core:** Cortex-M7 @ 216 MHz.
- **Flash:** 2MB.
- **Verdict:** This chip is the "Sweet Spot" for fixed-wing aircraft. While it runs at half the speed of the H7, planes do not require the 1kHz+ loop rates of racing drones. Crucially, it retains the **2MB Flash**. This means a Matek F765-Wing board can run the exact same complex Lua scripts and autonomous missions as a Cube Orange, just at a lower loop rate.

1.3 THE F722 TRAP: A STRUCTURAL DEAD-END

WARNING: Do not buy STM32F722 boards for ArduPilot.

- **The Marketing Lie:** The F722 is often sold as a "High Performance F7" upgrade to the F405.
- **The Fatal Flaw:** It has only **512KB** of Flash memory.
- **The ArduPilot Reality:** The standard ArduPilot Copter binary is now over 1MB. To make it fit on an F722, the developers must use a "Cut Down" build server definition.
  - **Lua Scripting:** COMPLETELY DISABLED.
  - **DDS/ROS2:** COMPLETELY DISABLED.
  - **Gimbal Drivers:** OFTEN REMOVED.
  - **Terrain Following:** OFTEN REMOVED.
- **The Result:** You buy a "high end" board, but you are locked out of the high-end software features. The F722 is excellent for Betaflight (racing), but it is a dead-end for Autonomy.

---

2. INERTIAL MEASUREMENT UNITS (IMUS): THE SENSING LAYER

The IMU provides the raw Angular Velocity (Gyro) and Linear Acceleration (Accel) data to the EKF. The industry is currently in a transition period between legacy sensors and next-gen MEMS.

2.1 MPU6000 (LEGACY)

For a decade, this was the king. It had a high noise floor, but it was "deaf" to high-frequency vibrations. It was easy to tune because it simply didn't hear the motor bearings screaming at 200Hz. It is now End-of-Life (EOL).

2.2 ICM-42688-P (THE HIGH-BANDWIDTH SUCCESSOR)

This is the sensor found on almost all modern H7 boards.

- **The Good:** Extremely low noise floor. It is precise.
- **The Bad:** Extremely high bandwidth. It hears *everything*.
- **The Implication:** If you bolt an ICM-42688 to a noisy frame without software filtering, your drone will fly to the moon.
- **The Fix:** You **MUST** configure the **Harmonic Notch Filter** in ArduPilot. This filter uses the RPM data from the ESCs to "notch out" the exact frequency of the motors from the Gyro data *before* it hits the PID loop. With the Notch Filter, the 42688 is superior. Without it, it is unflyable.

2.3 BMI270 (THE BUDGET TRAP)

Found on cheap flight controllers.

- **The Limitation:** It has a hardware low-pass filter baked into the silicon that limits the sampling rate to ~3.2kHz.
- **Phase Lag:** This internal filter creates a delay between the physical movement and the digital signal.
- **Tuning Impact:** Delay (Phase Lag) is the enemy of the D-Term. You cannot tune a BMI270 drone as "tightly" as an ICM drone. It will oscillate if you push the gains too high because the flight controller is reacting to old news.

---

3. VIBRATION ISOLATION: PHYSICS OF MOUNTING

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY



How you mount the Flight Controller is just as important as which one you buy.

3.1 HARD MOUNTING (THE CUBE PHILOSOPHY)

High-end controllers like the Cube Orange use **Internal Isolation**.

- **The Design:** The IMU sensor is mounted on a heavy brass block, which floats on foam inside the plastic case.
- **The Physics (F=ma):** By increasing the mass (m) of the sensor block, the inertia increases. It takes more force (F) to shake it. This acts as a mechanical low-pass filter that blocks high-frequency motor noise while allowing low-frequency movement (turning) to pass through instantly.
- **The Rule: NEVER** soft-mount a Cube. If you put a Cube on gummies, you create two spring systems in series (The gummy + The internal foam). This creates complex resonance amplification (Harmonics) that can destroy flight stability. Hard mount it with screws or stiff VHB tape.

3.2 SOFT MOUNTING (AIOS AND LIGHT BOARDS)

Lightweight "All-In-One" boards lack the mass for inertial damping.

- **The Requirement:** You **MUST** use rubber grommets ("gummies").
- **The Tuning:** The gummies act as the mechanical filter. If they are too hard, noise gets through. If they are too soft, the entire flight controller "wobbles" on the stack during a snap roll. The Gyro interprets this board-wobble as drone-movement, causing the FC to fight a ghost movement. This manifests as a "low frequency bounce" after a flip.

---

4. ARDUPILOT INTEGRATION: KEY PARAMETERS

How to tell the software about your hardware choices.

`SCHED_LOOP_RATE`

- **Definition:** How many times per second the main control loop runs.
- **H7 Recommendation: 800Hz** or **1000Hz**. The processor has the power to run the loop fast, minimizing latency for the ICM-42688 sensors.
- **F765/F405 Recommendation: 400Hz**. A safe standard. Going higher offers diminishing returns on slower processors and eats CPU cycles needed for logging/telemetry.

`INS_GYRO_FILTER`

- **Definition:** The cutoff frequency for the static Low-Pass Filter on the Gyro.
- **Guidance:**
  - **5" Racer (Stiff Frame):** 80Hz - 100Hz. (Less filtering, less delay).
  - **10" Cinelifter (Flexy Frame):** 40Hz - 60Hz. (More filtering needed to hide resonance).
  - **Warning:** Setting this too low (< 40Hz) introduces dangerous Phase Lag, causing slow oscillations.

`INS_HNTCH_ENABLE`

- **Action:** Set to **1** immediately on any build with an **ICM-42688** sensor.
- **Why:** Without the Harmonic Notch, the high-bandwidth sensor will be overwhelmed by motor noise. This parameter enables the dynamic filter that tracks motor RPM.

CONTROL LINK ARCHITECTURE: EXPRESSLRS AND MAVLINK

CRITICAL In the ArduPilot ecosystem, the radio link is not just for control sticks; it is a bidirectional data pipeline for MAVLink telemetry. This fundamentally changes the requirements compared to a racing drone. A system that is "solid" for a racer (who only needs 50Hz telemetry for battery voltage) may be completely unusable for an autonomous operator trying to upload a mission.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





1. MODULATION PHYSICS: LORA VS. FSK

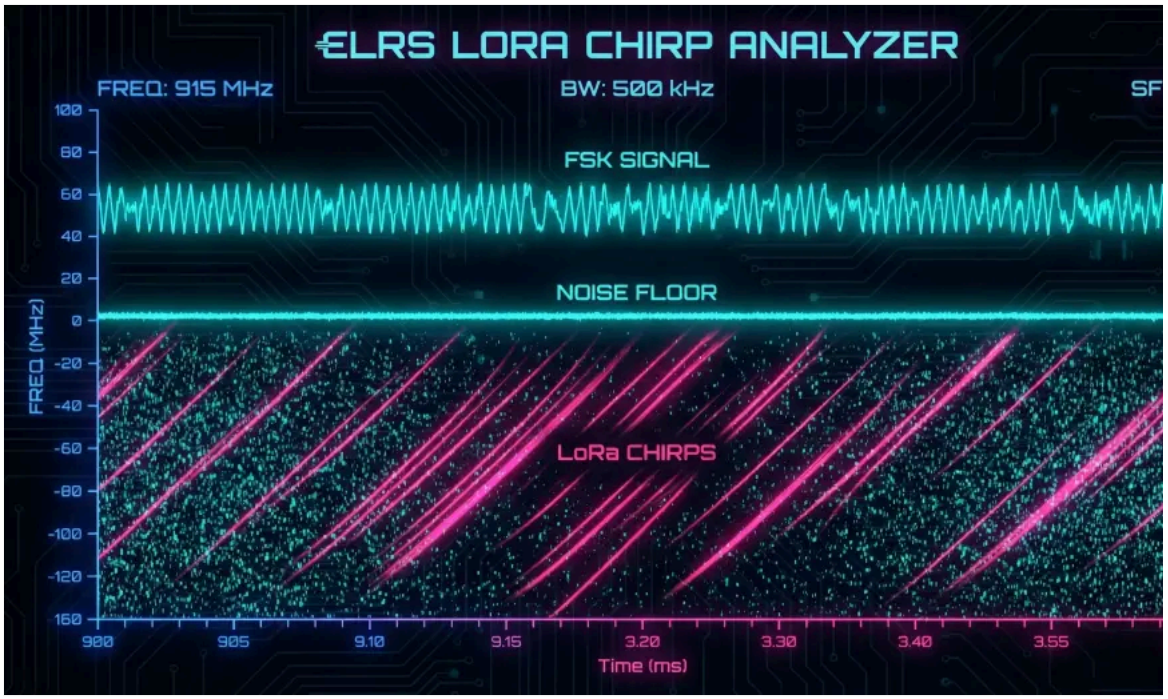
To understand why ExpressLRS (ELRS) has taken over the world, we must look at the modulation layer.

1.1 FSK (FREQUENCY SHIFT KEYING)

- **The Legacy:** Used by FrSky, Spektrum, and Futaba.
- **The Mechanism:** The radio shifts the frequency slightly up or down to represent a 1 or a 0.
- **The Limitation:** FSK requires a **Positive Signal-to-Noise Ratio (SNR)**. The signal must be louder than the background noise. If the signal drops below the noise floor, the receiver can no longer distinguish the frequency shift.
- **The Result:** Reliable, but range is limited by power output. 100mW gets you ~2km.

1.2 LORA (CHIRP SPREAD SPECTRUM)

- **The Modern Standard:** Used by ExpressLRS and Crossfire.
- **The Mechanism:** LoRa encodes data into "Chirps"—sweeping frequency variations that rise or fall over time.



- **The Coding Gain:** The receiver is not listening for volume; it is matching a *pattern*. It's like hearing a specific whistle in a crowded stadium. Even if the whistle is quieter than the crowd (Negative SNR), the human brain (or LoRa chip) can pick out the rising pitch.
- **The Sensitivity:** LoRa can demodulate signals down to **-20dB SNR**. It can hear a whisper in a hurricane.
- **The Result:** 100mW gets you 10km+.

---

2. THE BANDWIDTH BOTTLENECK: MAVLINK INTEGRATION

ArduPilot uses the MAVLink protocol to talk to the Ground Station (Mission Planner / QGC). MAVLink is "heavy." It sends parameter lists, mission waypoints, and status text.

2.1 THE 900MHZ TRAP

Many long-range pilots instinctively buy 900MHz ELRS hardware.

- **The Physics:** Lower frequency (900MHz) penetrates obstacles better than 2.4GHz.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

- **The Trade-off:** The 900MHz band is narrow. To fit within regulatory limits and maintain range, ELRS 900MHz typically runs at low packet rates (50Hz - 200Hz).
- **The Calculation:** At 50Hz packet rate, the available bandwidth for telemetry is often less than **50 bytes per second**.
- **The User Experience:** You connect Mission Planner. It says "Getting Params 1 of 800". You wait. And wait. It takes 5 minutes to download the parameters. The map updates once every 3 seconds. It is safe to fly, but agonizing to configure.

2.2 THE SOLUTION: 2.4GHZ HIGH-SPEED

- **The Physics:** 2.4GHz has massive spectral bandwidth. ELRS can run at 333Hz, 500Hz, or even 1000Hz (F1000).
- **The Calculation:** At F1000 mode (1000Hz), ELRS has enough "air time" to slot in over **4000 bits per second** of telemetry data while still maintaining ultra-low latency control.
- **The User Experience:** Parameters download in seconds. The Artificial Horizon moves smoothly.

2.3 GEMINI: THE BEST OF BOTH WORLDS

The latest "Gemini" hardware (e.g., Radiomaster Ranger Gemini) transmits on **two frequencies simultaneously**.

- **Diversity:** If interference jams 2.400GHz, the packet still gets through on 2.480GHz.
- **Reliability:** It offers the bandwidth of 2.4GHz with link reliability that rivals 900MHz.
- **Verdict:** For professional ArduPilot builds, **ELRS 2.4GHz Gemini** is the current gold standard.

---

3. CONFIGURATION FOR MAVLINK

Getting MAVLink to work over ELRS requires specific settings. It is not "Plug and Play."

3.1 TELEMETRY RATIO (LUA SCRIPT)

On your radio handset, run the ExpressLRS Lua script.

- **Standard/Race Setting:** `1:64`. This sends one telemetry packet for every 64 control packets. Good for VTX status, useless for MAVLink.
- **MAVLink Requirement:** Set this to `1:2`.
  - **What this does:** It tells the system to sacrifice control packet density to prioritize downlink data. Every other packet is a telemetry packet. This maximizes the bandwidth for Mission Planner.

3.2 FLIGHT CONTROLLER PARAMETERS

You must tell ArduPilot to speak the right language on the UART.

- `SERIALx_PROTOCOL = 23` (RCIN). This enables the specific CRSF-to-MAVLink translation.
- `SERIALx_OPTIONS = 0`.
- `SERIALx_BAUD = 460` (460,800). ELRS usually negotiates this, but setting it explicitly ensures stability.

3.3 RSSI AND LINK QUALITY

To see your signal strength in the OSD:

- `RSSI_TYPE = 3` (ReceiverProtocol). ArduPilot reads the RSSI value directly from the CRSF data stream.
- `RC_OPTIONS` → Bit 2 (**Suppress RC Failsafe**).
  - **Why:** ELRS has its own failsafe flag bit. ArduPilot should listen to that bit, rather than guessing failsafe based on channel values (like older PWM receivers).

GNSS & MAGNETOMETRY: STATE ESTIMATION SENSORS

CRITICAL An autonomous vehicle is a robot that makes decisions based on where it *thinks* it is. If the GPS drifts 5 meters, the drone moves 5 meters. If the Compass drifts 20 degrees, the drone spirals out of control. The sensors define the reality for the

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





computer.

1. GNSS GENERATIONS: THE MULTIPATH REVOLUTION

For years, the u-blox **Neo-M8N** was the industry standard. It is now obsolete. The shift to the **M10** engine is not just an upgrade; it is a generational leap in reliability.

1.1 U-BLOX M8 (LEGACY)

- **Architecture:** Could track 3 constellations simultaneously (usually GPS + GLONASS + Galileo).
- **Sensitivity:** Good for open fields.
- **The Failure Mode: Multipath Interference**. In urban environments (near buildings) or under tree canopies, GPS signals bounce off surfaces. The M8 struggles to distinguish the direct signal from the reflected "echo." Since the echo has traveled further, the timing is off, and the GPS solution jumps ("glitches").

1.2 U-BLOX M10 (THE STANDARD)

- **Architecture:** Tracks **4** constellations simultaneously (Adds **BeiDou**).
- **Satellite Count:** A typical M8 sees 12-16 satellites. An M10 in the same spot sees **30+**.
- **The Physics:** More satellites means better geometry (lower HDOP).
- **The Killer Feature:** The M10 chip has advanced RF front-ends and algorithms specifically designed to filter out Multipath signals.
- **Real World Result:** The drone holds position under a tree or next to a wall where an M8 would wander or lose lock entirely.

---

2. THE COMPASS PROBLEM: RESISTIVE VS. INDUCTIVE

The Magnetometer (Compass) is the most hated sensor in the drone world. It is prone to interference and calibration errors. However, **how the sensor works** determines whether it will work *for you*.

2.1 ANISOTROPIC MAGNETORESISTIVE (AMR)

- **Common Chips:** QMC5883L, IST8310.
- **The Physics:** The electrical resistance of a permalloy film changes when exposed to a magnetic field.
- **The Fatal Flaw:** Resistance also changes with **Temperature**.
- **The Scenario:** You calibrate the compass on your bench at 22°C. You fly on a sunny day. The board heats up to 45°C. The resistance changes. The sensor thinks the magnetic field has rotated.
- **The Consequence: Toilet Bowling.** The FC thinks the drone is facing North-East when it is facing North. It tries to correct, entering a growing spiral.

2.2 MAGNETO-INDUCTIVE

- **Common Chip:** PNI RM3100.
- **The Physics:** It uses a solenoid coil as the inductor in an oscillation circuit. The magnetic field changes the **inductance** of the core, which shifts the **frequency** of the oscillation.
- **The Advantage:** It measures **Time** (Frequency), not Resistance. Time does not drift with temperature.
- **Verdict:** The RM3100 is virtually immune to thermal drift. For any professional or expensive rig, this sensor is mandatory.

---

3. DEALING WITH INTERFERENCE (EMI)

The #1 cause of "Compass Variance" errors is the drone interfering with itself.

3.1 AMPERE'S LAW & TWISTED PAIRS

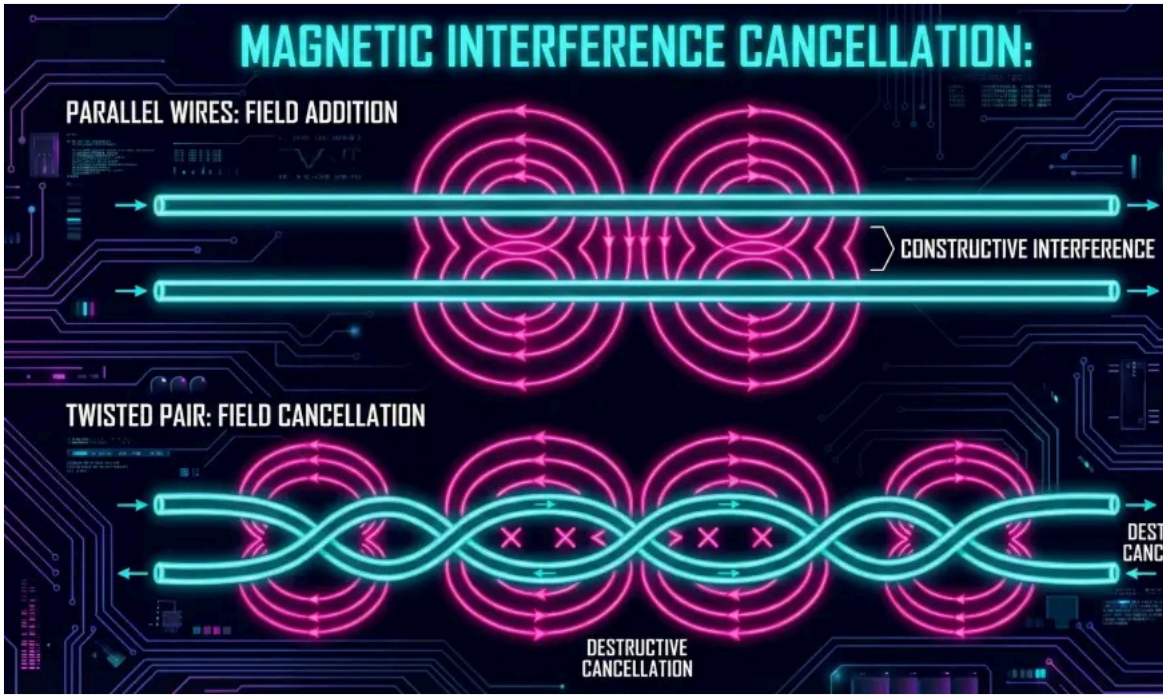
- **The Physics:** Any current flowing through a wire generates a magnetic field around it.
- **The Problem:** The battery wires carrying 50 Amps to the ESCs create a massive magnetic field that swamps the Earth's weak magnetic field.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- **The Fix: Twist** the Positive and Negative wires together tightly.



- **Why:** The current flows in opposite directions in the two wires. The magnetic field from the positive wire is effectively canceled by the equal-and-opposite field from the negative wire. This simple mechanical step reduces noise floor by 90\%.

3.2 THE INVERSE CUBE LAW

- **The Physics:** The strength of a magnetic dipole field drops off with the **cube** of the distance ($1/r^3$).
- **The Practicality:** Distance is your best filter.
  - Mounting the compass flat on the top plate (1cm from PDB) = **100\% Interference**.
  - Mounting the compass on a 5cm mast = **0.8\% Interference**.
- **Rule:** Always use a GPS mast. If you are building a small drone where a mast isn't possible, use a rear-mounted TPU holder to get the GPS/Compass as far from the power wires as possible.

---

4. ARDUPILOT INTEGRATION: KEY PARAMETERS

`GPS_GNSS_MODE`

- **Configuring Constellations:** This bitmask tells the u-blox chip which satellites to listen to.
- **Recommendation:** Set to **65** (GPS + SBAS + Galileo + BeiDou + GLONASS). Use all available constellations for maximum redundancy.

`COMPASS_ORIENT`

- **The #1 Setup Error:** If your compass arrow points forward but the chip is mounted upside-down (common in GPS pucks), you must tell ArduPilot.
- **The Check:** If the HUD moves correctly in Pitch/Roll but Yaw drifts or spins, your `COMPASS_ORIENT` is wrong.
- **Common Setting:** `0` (None) if arrow is forward/up. `8` (Roll 180) if arrow is forward/down.

`EKF3_CHECK_SCALE`

- **The "Compass Variance" Trigger:** ArduPilot's EKF monitors the "Innovation" (Error) between the GPS heading and the Compass heading.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

- **The Threshold:** If the error exceeds this scale (default 100-150\%), the EKF declares the compass "Bad" and triggers a failsafe.
- **Diagnosis:** If you get "Compass Variance" warnings in flight, **do not increase this limit**. It means your compass is physically mounted too close to power wires (Mag Interference). Fix the build, not the parameter.

FLIGHT CONTROLLER BOARD ARCHITECTURE: LAYOUT & POWER

CRITICAL The physical layout of the flight controller determines its electrical noise environment and thermal performance. Choosing between an All-In-One (AIO) and a Stack system is not just about size; it's about physics.

1. FORM FACTORS: AIO VS. STACK

1.1 THE AIO (ALL-IN-ONE) ARCHITECTURE

An AIO board integrates the Flight Controller (MCU/IMU) and the 4-in-1 ESCs onto a single PCB.

- **The Advantage:** Compactness and weight. Essential for sub-250g builds and "Whoop" class drones.
- **The Thermal Problem:**
  - **Physics:** ESC MOSFETs generate significant heat (switching losses). The STM32 processor and IMU are sensitive to heat.
  - **The Gradient:** On an AIO, the heat from the ESCs soaks directly into the IMU. This causes gyro bias drift during the flight as the board heats up.
- **The Electrical Noise Problem:**
  - **Physics:** High-current switching (40kHz-96kHz) happens millimeters away from sensitive 3.3V logic lines.
  - **Consequence:** AIOs inherently suffer from higher noise floors. They require aggressive software filtering (lower `INS_GYRO_FILTER`) which adds latency.
- **Repairability:** Zero. If you blow one ESC FET, the entire FC is trash.

1.2 THE STACK ARCHITECTURE (FC + ESC)

The industry standard for 5" and larger drones. The FC and ESC are separate boards connected by a ribbon cable.

- **The Advantage: Isolation**.
  - **Thermal:** The ESC is usually on the bottom, receiving direct airflow. The FC is stacked above it with an air gap. This thermal break keeps the Gyro temperature stable.
  - **Electrical:** The high-current path is confined to the lower board. The upper board remains a "clean" environment for logic and sensors.
- **Maintenance:** Modular replacement.

---

2. POWER REGULATION: THE BEC (BATTERY ELIMINATION CIRCUIT)

The BEC is the heart of the FC's power distribution network. It steps down the main battery voltage (e.g., 25V for 6S) to usable logic voltages (5V, 9V, 3.3V).

2.1 LINEAR REGULATORS (LDO)

- **How they work:** They burn off excess voltage as heat.
- **Efficiency:** Terrible. Converting 25V to 5V is ~20\% efficient.
- **Usage:** Only used for the final clean 3.3V stage for the MCU and IMU. They are electrically "quiet" (no switching noise), providing the cleanest power for sensors.

2.2 SWITCHING REGULATORS (BUCK CONVERTERS)

- **How they work:** They rapidly switch the input voltage on and off, using an inductor to smooth the output.
- **Efficiency:** Excellent (>90\%). Used for the 5V (RC, GPS) and 9V (VTX) rails.

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- **The Danger: Switching Noise**. A cheap or poorly filtered switching regulator introduces high-frequency noise into the video feed (lines in FPV) and can interfere with sensitive GPS receivers.
- **Design Check:** High-quality FCs shield the inductor component. Look for a grey square block on the board—if it's shielded, good. If you see exposed copper coils, expect noise.

2.3 THE "9V RAIL" MYTH

Many modern digital VTXs (DJI O3, Walksnail) draw up to 2A.

- **The Trap:** Many FCs claim a "9V 2A" BEC. In reality, under load and heat, they often voltage-sag, causing the VTX to reboot in mid-air ("blackout").
- **The Fix:** For high-power builds, run the VTX directly from V-Bat (if supported) or use an external, dedicated BEC (PDB). Do not trust the onboard FC BEC for >15W loads unless verified.

---

3. MOUNTING PATTERNS

- **30.5mm x 30.5mm (M3):** The "Standard". Most robust. Large pads for soldering. Fits standard 4-in-1 ESCs. Best for 5" - 10" builds.
- **20mm x 20mm (M2/M3):** The "Mini". Now capable of handling high power (60A), but thermal mass is lower. Harder to solder. Good for lightweight 5" or 3-4" builds.
- **25.5mm x 25.5mm (Whoop):** Exclusively for AIO boards.

PROPULSION PHYSICS: MOTORS & PROPELLERS

CRITICAL The propulsion system is the physical interface between the flight controller's digital commands and the atmosphere. A mismatch here cannot be fixed by PID tuning. This guide deconstructs the electromechanical physics of the drivetrain to allow for calculated, not guessed, component selection.

1. THE ELECTROMECHANICAL CORE: BRUSHLESS MOTOR THEORY

Modern drones exclusively use **Permanent Magnet Synchronous Motors (PMSM)**, commonly referred to as Brushless DC (BLDC) motors. Unlike brushed motors which use mechanical commutators (creating friction and noise), BLDC motors shift the complexity of commutation to the Electronic Speed Controller (ESC).

1.1 INRUNNER VS. OUTRUNNER ARCHITECTURE

- **Inrunner:** The rotor spins inside the stator. Common in RC cars or ducted fans. High RPM, low torque.
- **Outrunner:** The rotor (the "bell" with magnets) spins around the outside of the stator.
  - **The Physics Advantage:** Torque (tau) is the product of Force (F) and Radius (r). By placing the magnets on the outer perimeter, outrunners maximize the lever arm (r) for a given motor volume. This allows them to generate the massive torque required to swing large propellers directly, without the weight and failure points of a reduction gearbox.

1.2 THE MOTOR CONSTANTS: BEYOND THE MARKETING

To predict performance, we must look beyond the box art and understand the three intrinsic motor constants.

THE VELOCITY CONSTANT (KV)

Defined as the theoretical RPM per Volt under no load (RPM ~= Kv * V).

- **The Mechanism:** Kv is determined by the **Back-EMF**. As the motor spins, it acts as a generator, creating a voltage that opposes the battery. The motor stops accelerating when the Back-EMF equals the battery voltage.
- **Winding Physics:** Fewer turns of thicker wire = Low Resistance = Low Back-EMF = **High Kv**.
- **Application:** High Kv motors are built for speed but require huge current to produce torque.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





THE TORQUE CONSTANT (KT)

This is the metric that matters for heavy lift and crisp handling. It describes the torque produced per Ampere of current (Nm/A).

- **The Inverse Law:** There is a rigid physical relationship between Speed and Torque. You cannot increase one without decreasing the other.

  ```
  Kt ~= 9.55 / Kv
  ```

- **Implication:** A 2700Kv motor has inherently low torque per amp. To spin a heavy prop, it must draw massive current, generating waste heat (I^2R losses). A 1700Kv motor produces the same torque with far less current, making it the correct choice for heavy propellers.

THE MOTOR CONSTANT (KM)

The "Efficiency Truth." Km defines the torque produced for a given amount of waste heat.

- **Significance:** Unlike Kv, which changes with winding turns, Km is constant for a specific stator size and magnet configuration. A larger stator or higher-grade magnets will increase Km, indicating a motor that can produce more torque with less waste heat.

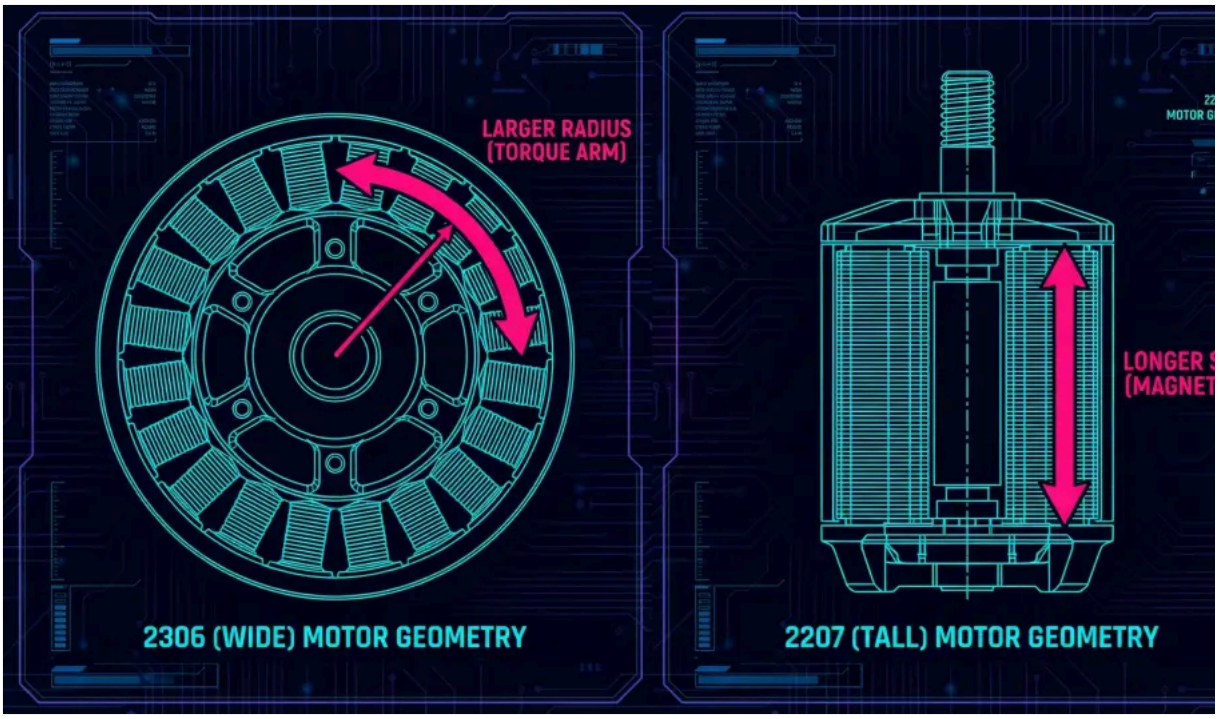
---

2. STATOR GEOMETRY & MAGNETIC FLUX

Motor size is standardized (e.g., **2306** = 23mm width, 6mm height). However, the **Aspect Ratio** of the stator fundamentally changes the power delivery feel.

2.1 WIDE STATOR (2306) VS. TALL STATOR (2207)

Both motors have roughly the same stator volume (~2500 mm^3), but they fly differently.



- **Wide Stator (2306):**
  - **Physics:** The magnetic interaction occurs further from the center axis. This larger radius increases the instantaneous torque leverage.
  - **Cooling:** Larger top/bottom surface area allows for better convective cooling from the prop wash.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

    - **Flight Feel:** "Smooth" and "Linear." Preferred by Freestyle pilots who want consistent power across the throttle range.

- **Tall Stator (2207):**
    - **Physics:** Increased surface area along the vertical axis (longer magnets).
    - **Performance:** Higher magnetic flux interaction area typically yields more "low-end grunt" or "snap."
    - **Flight Feel:** "Responsive" and "Aggressive." Preferred by Racers who need to change RPM instantly to corner at high speeds.

2.2 STATOR LAMINATIONS

The stator is not solid iron; it is a stack of thin steel sheets.

- **Eddy Currents:** Rapidly changing magnetic fields induce rogue currents in the iron core, creating heat.
- **Lamination Thickness:** Thinner laminations (0.15mm vs 0.2mm) reduce these eddy currents. High-end motors use thinner laminations to improve efficiency at high RPM.

---

3. PROPELLER AERODYNAMICS: THE AIRSCREW

The propeller converts rotational torque into aerodynamic thrust. This is a fluid dynamics problem governed by **Blade Element Momentum Theory**.

3.1 DIAMETER (D)

Thrust scales with the **fourth power** of diameter (Thrust is proportional to D^4).

- **The Scale Effect:** Increasing prop size from 5" to 6" doesn't just add "a little" thrust; it drastically increases the disk area and the air mass moved.
- **Efficiency:** Larger props are inherently more efficient (Grams/Watt) because they accelerate a large mass of air slowly (low Disc Loading), rather than a small mass of air quickly.

3.2 PITCH

The theoretical distance the prop moves forward in one revolution.

- **High Pitch (e.g., 5.1"):** High Angle of Attack (AoA). Moves more air per revolution but stalls at low RPM (inefficient hover). Requires massive motor torque.
- **Low Pitch (e.g., 3.0"):** Low AoA. Spins up instantly (responsive) but "revs out" at top speed (the drone hits a wall of air resistance).

3.3 MOMENT OF INERTIA (MOI)

Often ignored, this is the most critical factor for PID tuning.

- **Definition:** Resistance to change in rotation. `MoI = sum(mass * radius^2)`.
- **Heavy Tips:** A propeller with heavy blades or winglets has high MoI.
- **The Control Loop:** When the Flight Controller sends a command to "Stop Rolling," the motor must actively brake the propeller.
    - **High MoI:** The prop keeps spinning due to momentum. The drone "overshoots" the angle, then bounces back. The pilot feels "slop."
    - **Low MoI:** The prop stops instantly. The drone feels "locked in."
- **Recommendation:** Prioritize light-weight polycarbonate props over heavy glass-nylon blends for precision flight.

---

4. SYSTEM INTEGRATION: MATCHING MOTOR TO PROPELLER

The art of propulsion engineering lies in matching the motor's Torque-Speed curve to the propeller's Load curve.

4.1 ADVANCE RATIO (J)

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- **Static vs. Dynamic:** On the bench, the air entering the prop is still (V=0). In flight, the air is moving.
- **Advance Ratio (J):** The ratio of forward speed to tip speed.

  `J = Velocity / (RPM * Diameter)`
- **The Stall:** As the drone flies faster, the *effective* angle of attack of the propeller decreases.
  - **Over-propping:** If you put a high-pitch prop on a low-torque motor, the motor cannot reach the RPM needed to maintain a positive angle of attack at high speeds. The prop "unloads," and the drone stops accelerating.
  - **Under-propping:** A low-pitch prop on a high-RPM motor acts as a speed governor. The drone hits its top speed instantly and wastes battery trying to spin faster against air drag.

---

5. COMPREHENSIVE SELECTION GUIDE

5" FREESTYLE (THE STANDARD)

- **Goal:** Balance of power, response, and durability.
- **Stator:** 2306 or 2207.
- **Kv (6S):** 1750Kv - 1850Kv.
- **Prop:** 5×4.3×3 (Pitch 4.3). The "Goldilocks" prop.

5" RACING

- **Goal:** Maximum top end and cornering grip. Efficiency is irrelevant.
- **Stator:** 2207 (Tall) or 2208.
- **Kv (6S):** 1950Kv - 2100Kv.
- **Prop:** 5.1×5.1×3 (High Pitch). Requires high RPM to bite.

7" LONG RANGE / MOUNTAIN SURFING

- **Goal:** Efficiency and Reliability.
- **Stator:** 2806.5 or 2807. (Large stator needed for torque).
- **Kv (6S):** 1300Kv. (Low Kv for Torque).
- **Prop:** 7×3.5×2 (Bi-Blade). Low pitch and low blade count maximize Grams/Watt.

3" CINEWHOOP

- **Goal:** Lift heavy cameras (GoPro) in a small footprint.
- **Stator:** 1507 or 2004 (Wide/Flat).
- **Kv (6S):** 2800Kv - 3500Kv.
- **Prop:** 5-Blade (D76) or 3-Blade Ducted. High blade count needed to grab "dirty" air inside the ducts.

THE "PROP WASH" DIAGNOSTIC

If your drone oscillates when descending into its own wake (prop wash):

1. **Physics:** Your propeller's Moment of Inertia > Your Motor's Braking Torque.
2. **The Fix:**
   - **Software:** Increase D-Term (Active Braking) - Risky, heats motors.
   - **Hardware (Correct):** Switch to a lighter propeller or a lower pitch. Or upgrade to a larger stator size (e.g., 2207 to 2306).

---

6. ARDUPILOT INTEGRATION: KEY PARAMETERS

How to tell the software about your propulsion choices.

`MOT_THST_EXPO` (THRUST LINEARIZATION)

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- **The Problem:** Thrust is not linear. 50\% throttle does not equal 50\% thrust. (Thrust is proportional to RPM squared).
- **The Fix:** This parameter "flattens" the curve so the PID loop sees linear response.
- **Setting:**
  - **5" Prop:** ~0.65
  - **10" Prop:** ~0.60
  - **Large Props:** Lower values.
  - **Tuning:** If the drone oscillates at hover but is stable at high throttle, your Expo is wrong.

`MOT_SPIN_ARM` VS `MOT_SPIN_MIN`

- **SPIN_ARM:** The speed the motors spin when you arm the drone (on the ground). Set this low (e.g., 0.10) just to show the pilot it's live.
- **SPIN_MIN:** The "Air Mode" floor. ArduPilot will NEVER drop the motor speed below this value during flight, even at zero throttle.
  - **Why:** If a motor stops in the air, the prop can desync or stall. This keeps the prop "biting" the air for stability during dives.
  - **Setting:** usually 0.15 (15\%).

`MOT_BAT_VOLT_MAX` / `MOT_BAT_VOLT_MIN`

- **Voltage Scaling:** As the battery drains (Voltage drops), the motors lose top-end power.
- **The Feature:** ArduPilot automatically scales the PID outputs. As voltage drops, it pushes the motors harder to achieve the same thrust.
- **Requirement:** You must set these parameters correctly (e.g., 25.2V Max, 21.0V Min for 6S) for the scaling to work.

ELECTRONIC SPEED CONTROLLERS (ESCS): THE POWER TRANSLATORS

CRITICAL The ESC is the most hardworking computer on your aircraft. It must perform thousands of calculations per second to manage the violent collapse of magnetic fields, all while sitting millimeters away from high-current noise that would blind a lesser device. In the ArduPilot ecosystem, the ESC has evolved from a simple power switch into a sophisticated sensor that dictates the quality of your flight.

1. COMMUTATION PHYSICS: THE INVISIBLE DANCE

Brushless motors (BLDC) are fundamentally three-phase AC motors powered by a DC battery. The ESC's job is to translate that DC into a rotating magnetic field.

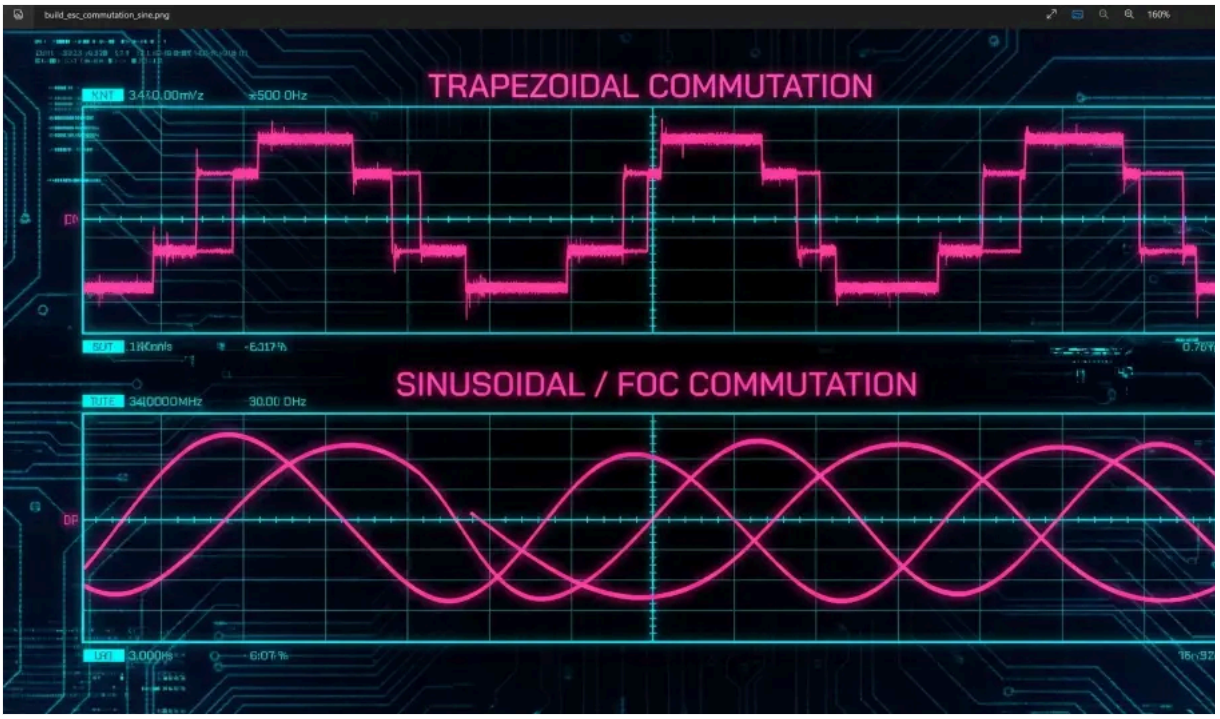
1.1 THE SIX-STEP DANCE (TRAPEZOIDAL)

Most standard ESCs (BLHeli_S, BLHeli_32) use **Trapezoidal Commutation**.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY







- **The Method:** At any given microsecond, the ESC energizes two of the three motor wires. One is "High" (Positive), one is "Low" (Negative), and the third is "Floating."
- **The "Listen" Phase:** The ESC uses the floating wire as a sensor. As the permanent magnets on the motor bell sweep past the coils, they induce a tiny voltage in that floating wire (Back-EMF).
- **The Trigger:** The ESC waits for this voltage to cross zero. This "Zero Crossing" tells the ESC exactly where the rotor is, allowing it to "fire" the next step in the sequence perfectly.
- **The Compromise:** Because this switching is digital (on/off), the current flow is "choppy." This creates **Torque Ripple**—microscopic vibrations that you can hear as a "whine" and your gyro can hear as "noise."

1.2 THE SILENT PATH (SINUSOIDAL / FOC)

Higher-end ESCs (like the APD F-Series or AM32 in Sine mode) use **Field Oriented Control (FOC)**.

- **The Method:** Instead of on/off switching, the ESC drives all three wires continuously with varying voltages to create a smooth Sine Wave.
- **The Benefit:** Operation is nearly silent. Torque is consistent throughout the rotation, reducing the vibration floor of the entire airframe. For high-end cinema rigs where every vibration ruins a shot, FOC is mandatory.

---

2. PROTOCOLS: THE DSHOT REVOLUTION

In the early days, we used **PWM (Pulse Width Modulation)**. It was a messy, analog-style system where the length of a pulse determined the speed. If electrical noise "stretched" a pulse, the motor sped up unexpectedly. You had to "calibrate" ESCs to teach them what 0\% and 100\% looked like.

2.1 THE DIGITAL CERTAINTY OF DSHOT

DShot (Digital Shot) changed everything. Instead of pulses, the Flight Controller sends a 16-bit digital packet.

- **The Packet:** `[11 bits Throttle] [1 bit Telemetry Request] [4 bits Checksum]`
- **Why it wins:** Because it is digital, it is immune to noise. The ESC either receives the exact number "1024" (50\% throttle) or the checksum fails and it ignores the packet. There is no "drifting" or "guessing." **Calibration is dead.**

2.2 BIDIRECTIONAL DSHOT: THE MAGIC BULLET

This is the single most important feature for an ArduPilot builder.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

- **The Loop:** Immediately after receiving a command, the ESC sends a packet back to the FC on the same wire, reporting the **exact RPM** of the motor.
- **The Result:** ArduPilot now knows exactly how fast every motor is spinning 400+ times per second.
- **The Harmonic Notch:** ArduPilot uses this data to center a digital "Notch Filter" on the motor frequency. If your motor is creating noise at 182Hz, the filter tracks it to 182Hz and deletes it. This allows you to run higher PID gains and get "locked-in" flight without the motors getting hot.

---

3. FIRMWARE ECOSYSTEM: CHOOSING YOUR BRAIN

Not all ESCs speak the same language. The firmware determines the feature set.

3.1 BLHELI_S (8-BIT)

- **Hardware:** Older BB21 or BB51 chips. Found on budget AIO boards and "Whoop" ESCs.
- **The Limitation:** The stock firmware is outdated and cannot handle Bidirectional DShot efficiently.
- **The Fix:** You **MUST** flash **Bluejay Firmware**. This is an open-source project that "unlocks" these 8-bit chips, enabling Bi-DShot and variable PWM frequencies (48kHz/96kHz) for smoother flight and longer battery life.

3.2 BLHELI_32 (32-BIT)

- **Hardware:** STM32L4 or F0 chips. The standard for high-performance builds from 2017-2024.
- **Status: Closed Source & Dead.** The developer ceased operations in 2024 due to geopolitical sanctions.
- **Legacy:** Existing ESCs work fine, but there will be no future updates or bug fixes.

3.3 AM32 (THE FUTURE)

- **Status:** Open Source.
- **Platform:** Runs on generic 32-bit MCUs (STM32, AT32).
- **Features:** It is the spiritual successor to BLHeli_32, offering advanced features like Sine Startup, Variable PWM, and S-Port telemetry. New hardware is increasingly shipping with AM32 pre-installed.

---

4. PRACTICAL BUILD STRATEGY: STACKS VS. INDIVIDUALS

4.1 THE 4-IN-1 ESC (THE "STACK")

- **Pros:** Clean wiring, compact, lighter.
- **Cons: The Chain Reaction.** If you blow one MOSFET on motor #3, you have to throw away the entire $80 board.
- **Thermal Trap:** All four ESCs are on one PCB. In a heavy-lift hover, they heat each other up.

4.2 INDIVIDUAL ESCS (ON THE ARMS)

- **Pros:** Thermal isolation (they are cooled by the prop wash). Repairability (replace one at a time).
- **Cons:** Messy wiring. Requires a Power Distribution Board (PDB).
- **Verdict:** Use Stacks for 5" freestyle and racing. Use Individual ESCs for 7"+ long-range or professional heavy-lift rigs.

---

5. THE CAPACITOR: THE INDUCTIVE SPIKE

CRITICAL: Every time the ESC switches off a phase, the magnetic field in the motor collapses. This generates a high-voltage spike (Inductive Kickback).

- **Active Braking:** When you lower the throttle, the ESC actively brakes the motor (Damped Light). This turns the motor into a generator, dumping massive energy back into the power wires.
- **The Danger:** This "Ripple Voltage" can spike to 35V on a 4S system or 50V+ on a 6S system.
  - **Result:** It blows the 5V regulator on your Flight Controller, or burns out the Video Transmitter.
- **The Fix:** You **MUST** solder a **Low-ESR Electrolytic Capacitor** (e.g., 1000uF 50V) directly to the ESC battery pads.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- **The Trap:** Do not solder it to the XT60 pigtail. The wire length adds inductance that prevents the capacitor from "catching" the spikes. It must be at the source of the noise: the ESC pads.

---

6. ARDUPILOT INTEGRATION: KEY PARAMETERS

`MOT_PWM_TYPE`

- **Selection:** Set to **6** (DShot600).
- **DShot300 vs DShot600:** DShot600 is faster, but if you have long signal wires or a noisy build, DShot300 is more robust against packet loss.

`SERVO_BLH_AUTO`

- **Passthrough:** Set to **1** (Enabled).
- **The Function:** This allows you to connect your Flight Controller to your PC via USB, and then use the **BLHeli Configurator** or **ESC-Configurator** (web) to flash your ESCs. The FC acts as a bridge. You do not need a separate programmer.

`SERVO_DSHOT_ESC` (POLE COUNT)

- **The Math:** The ESC reports *Electrical* RPM (eRPM). ArduPilot needs *Mechanical* RPM.
- **Formula:** `Mechanical_RPM = eRPM / (Pole_Count / 2)`
- **Configuration:** You must set the number of magnetic poles (count the magnets on the bell). Typically **14** for 5" motors. If this is wrong, your Harmonic Notch Filter will target the wrong frequency, and your drone may vibrate or fly away.

POWER SYSTEMS ENGINEERING: THE FOUNDATION OF FLIGHT

CRITICAL The power system is the most dangerous component of your aircraft. A failure here is rarely a "glitch"—it is usually a fire or a "dead stick" crash from 400 feet. In autonomous flight, where the computer (FC) depends on a stable voltage rail to think, the power system is the literal foundation of the mission.

1. BATTERY CHEMISTRY: ENERGY VS. POWER

Not all Lithium is created equal. The choice between LiPo and Li-Ion is a trade-off between **Burst Power** and **Total Endurance**.

1.1 LITHIUM POLYMER (LIPO)

The standard for performance flight.

- **The Structure:** Soft foil pouches stacked together. This maximizes surface area, allowing for extremely high discharge rates.
- **The IR (Internal Resistance):** Very low (< 3 milli-Ohms per cell).
- **The Personality:** LiPos can dump their entire energy payload in under 2 minutes. They provide the "punch" needed for aggressive maneuvers or heavy lifting.
- **The Danger:** If punctured or overcharged, the foil pouches can "vent with flame" (thermal runaway). Always store and charge in a fireproof container.

1.2 LITHIUM ION (LI-ION)

The standard for long-range cruising.

- **The Structure:** Cylindrical metal cans (18650 or 21700).
- **The Superpower:** Massive energy density. A 21700 pack can hold 2x the energy of a LiPo of the same weight.
- **The Limitation:** High Internal Resistance (> 15 milli-Ohms). If you try to pull 100 Amps from a Li-Ion pack, the voltage will "sag" so low that the Flight Controller will reboot, and the internal heat will likely damage the cells.
- **Use Case:** Missions requiring 30+ minutes of flight at steady, low throttle.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

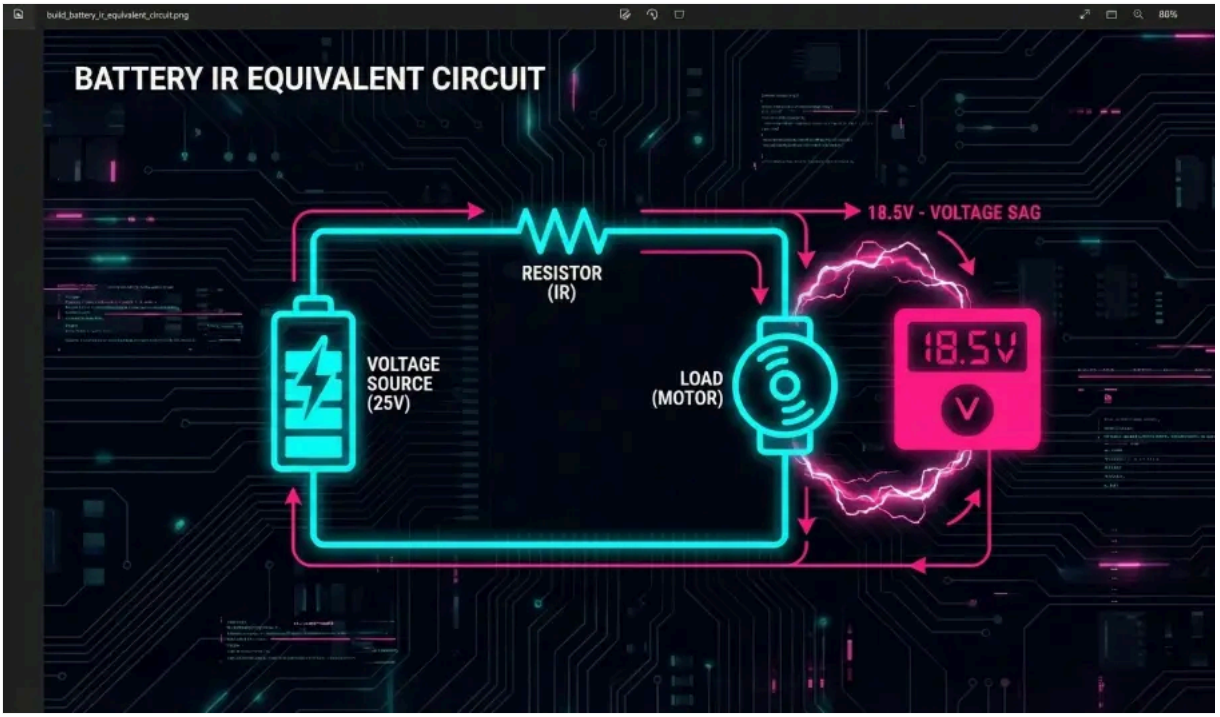


2. THE C-RATING LIE: HOW TO PROTECT YOUR WALLET

Most battery labels are pure marketing. A "150C" label is physically impossible; the 12AWG wires on the battery would melt into slag at that current.

2.1 THE PHYSICS OF INTERNAL RESISTANCE (IR)

The true health of a battery is measured in **milli-Ohms (mΩ)**, not "C-Rating."



- **New Battery:** 1-3 mΩ per cell.
- **Dying Battery:** 10+ mΩ per cell.
- **The Diagnostic:** Most modern chargers (like ISDT or SkyRC) can measure IR during charging. If one cell has a significantly higher IR than the others, that battery is a "fire in waiting." Retire it.

2.2 CALCULATING REAL-WORLD LIMITS

The true limit is determined by the **Internal Resistance (IR)** and the thermal mass.

- **Formula:** `Max Amps = sqrt(Watts_dissipated / IR)`
- **Rule of Thumb:**
  - "100C" label ≈ **35-40C** Real World.
  - "50C" label ≈ **20-25C** Real World.
  - Plan your build assuming **30C** max continuous draw.

---

3. IMPEDANCE MATCHING: CONNECTORS & THE "SPARK"

The connector is the smallest part of your drone, but it carries the most load. It is a resistor. If it gets hot after a flight, it is too small.

3.1 CONNECTOR HIERARCHY

- **XT30:** Rated for 30A. Use on 2-3" micro drones.
- **XT60:** Rated for 60A. The global standard for 5" drones.
- **XT90:** Rated for 90A. Mandatory for 7" and X-Class builds.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

- **AS150:** 7mm Anti-Spark bullets. Mandatory for 12S (50V) industrial rigs.

3.2 THE ANTI-SPARK SOLUTION

When you plug in a 6S (25V) battery, the capacitors on the ESCs charge instantly, creating a plasma arc (the "pop"). This arc erodes the gold plating on your connectors, increasing resistance.

- **The Fix:** Use **XT90-S** or **AS150** connectors. They have a built-in resistor that pre-charges the system before the final metal-to-metal contact is made. This preserves your connectors and your electronics.

---

4. VOLTAGE SAG PHYSICS

Voltage Sag is the difference between your battery's resting voltage and the voltage the FC sees under load.

Formula: `V_drop = Current * Resistance`

THE SCENARIO

- **Battery:** 6S LiPo (25.2V).
- **IR:** 0.02 Ohms (20mΩ).
- **Connector/Wire Resistance:** 0.003 Ohms.
- **Throttle Punch:** 100 Amps.

THE CALCULATION

`V_drop = 100 * (0.02 + 0.003) = 2.3 Volts`

- **Result:** The instant you hit full throttle, your voltage drops to **22.9V**.
- **Implication:** Your motors lose RPM (since RPM = Kv * Volts). The drone feels "soft."
- **Mitigation:**
    1. **Keep Warm:** IR increases dramatically when cold. Pre-heat packs in winter.
    2. **Parallel Packs:** Running two packs in parallel halves the total IR (`1/Rt = 1/R1 + 1/R2`), reducing sag by 50\%.

---

5. ARDUPILOT INTEGRATION: POWER MONITORING

The Flight Controller needs to know the voltage and current to calculate remaining flight time and failsafe conditions.

5.1 `BATT_MONITOR`

- **Setting:** Usually set to **4** (Analog Voltage and Current).
- **Calibration:** Do not trust the factory numbers. Measure your battery with a multimeter, then adjust `BATT_VOLT_MULT` in ArduPilot until the Mission Planner HUD matches your multimeter exactly.

5.2 `BATT_FS_LOW_VOLT` (THE LIFEBOAT)

- **Function:** This triggers a Return-to-Launch (RTL) or Land when the battery is low.
- **Rule:** Set this to **3.5V per cell** (21.0V for 6S).
- **Why?** If you set it lower, you might not have enough "reserve energy" to fight a headwind on the way home. It is better to land early than to drop from the sky.

5.3 `BATT_AMP_PERVLT`

- **Function:** Calibrates the current sensor.
- **Method:** Fly a battery, note how many mAh ArduPilot *thinks* it used, then look at how many mAh your charger puts back in. If the charger says 1000mAh and ArduPilot says 800mAh, you need to adjust this multiplier to ensure your "Fuel Gauge" is accurate.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





FIXED WING AIRFRAME PRINCIPLES

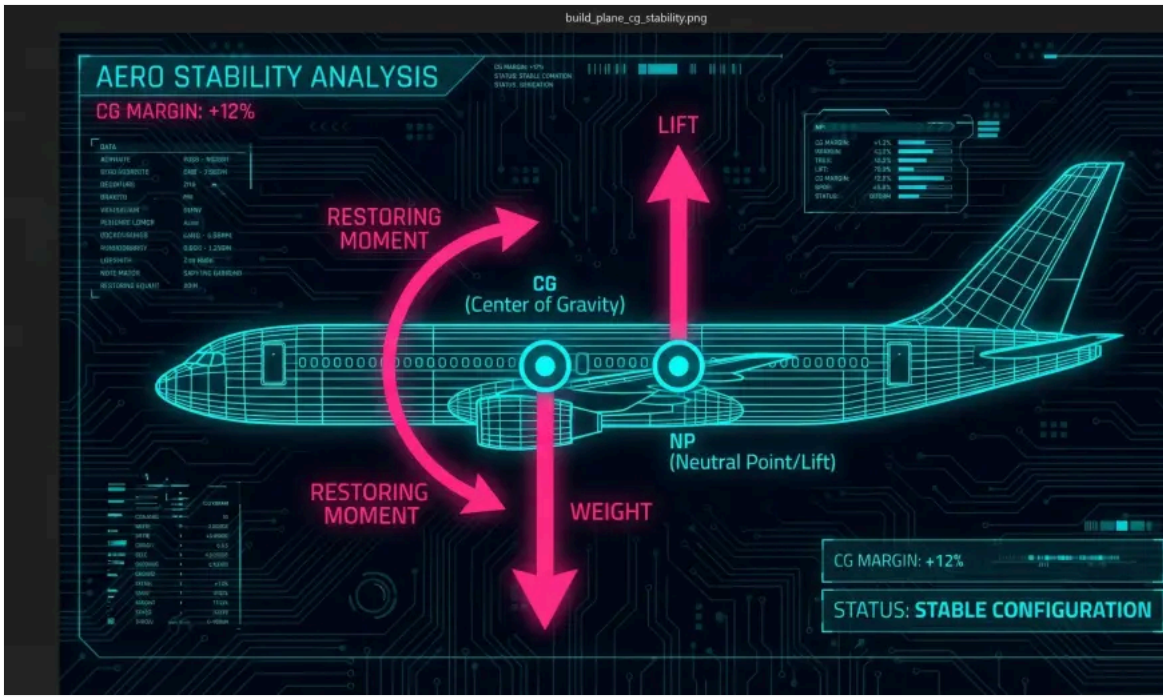
CRITICAL Unlike a multicopter, which is a brick that beats the air into submission with raw power, a fixed-wing aircraft is a refined instrument of aerodynamics. A multicopter relies on the flight controller to stay airborne; a plane relies on physics. If your airframe is aerodynamically unstable, no amount of PID tuning will save it.

1. THE PHYSICS OF STABILITY: THE "DART" ANALOGY

To understand why planes fly (or crash), you must understand **Longitudinal Stability**. Imagine throwing a dart backwards. It immediately flips around to fly heavy-end first. This is stability.

1.1 CENTER OF GRAVITY (CG) VS. NEUTRAL POINT (NP)

- **The Neutral Point (NP):** Think of this as the "Aerodynamic Center" of the entire aircraft—the point where all the lift forces effectively act.
- **The Center of Gravity (CG):** The point where the aircraft balances.
- **The Golden Rule:** The CG must **ALWAYS** be in front of the NP.



  - **Why?** When the CG is forward, the weight pulls the nose down. The horizontal stabilizer (tail) is rigged to create a downward force (Decalage) to balance this.
  - **The Stability Mechanism:** If a gust of wind pitches the nose up, the wing generates more lift. Because the CG is *forward* of the lift, this extra lift creates a lever arm that pushes the nose back down. The plane self-corrects.
- **The Tail-Heavy Crash:** If the CG moves behind the NP, the stability mechanism reverses. A gust pitches the nose up → Wing generates more lift → Lift is *in front* of the pivot → Nose pitches up *more*. This is a "Divergent" system. The plane will flip uncontrollable within seconds of takeoff.

1.2 STATIC MARGIN

The distance between the CG and the NP is called the **Static Margin**.

- **5\% to 15\% MAC:** This is the sweet spot.
  - **5\% (Agile):** The plane is twitchy and responsive. Great for fighters, harder for autopilots.
  - **15\% (Stable):** The plane wants to fly straight. It resists turning. Perfect for mapping drones.

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

- **ArduPilot Implication:** A highly stable plane (High Static Margin) requires more servo authority to turn. You may need to increase your `MIXING_GAIN` or mechanical throw to force the nose around.

---

2. AIRFOIL SELECTION: CHOOSING YOUR WING

The cross-section of your wing (the airfoil) dictates the personality of your drone.

2.1 FLAT BOTTOM (CLARK-Y)

- **Profile:** Flat on the bottom, curved on top.
- **Personality:** The "School Bus."
- **Pros:** Generates massive lift at low speeds. Very gentle stall characteristics (the nose just drops mushily). Easy to build and launch.
- **Cons:** High drag at high speeds. If you dive, it creates a massive pitching-up moment that the servos must fight.
- **Best For:** Mapping, Surveying, Long-Endurance Loitering.

2.2 SYMMETRICAL (NACA 00XX)

- **Profile:** Teardrop shape. Identical top and bottom.
- **Personality:** The "Acrobat."
- **Pros:** Zero pitching moment. It goes exactly where you point it. It flies upside down just as well as right side up.
- **Cons:** Zero lift at zero Angle of Attack (AoA). You must constantly hold "up elevator" (or trim) to keep it level.
- **Best For:** VTOL transitions, high-speed couriers.

2.3 REFLEX (FLYING WINGS)

- **Profile:** The trailing edge curves *up* slightly.
- **Physics:** A flying wing has no tail to push the tail down. To prevent it from tumbling forward, the airfoil itself must have a built-in "up elevator" curve at the back. This is called Reflex.
- **The Trap:** You cannot use standard airfoils on a flying wing. It will become unstable and tuck (dive) as speed increases.

---

3. MECHANICAL LINKAGE: THE "RESOLUTION" WAR

This is the single most common reason ArduPilot users fail to get a good tune.

3.1 THE GEOMETRY PROBLEM

Imagine you are driving a car, but the steering wheel is loose. You turn it 10 degrees before the tires move. This is **Slop**. Now imagine the steering is super sensitive: moving the wheel 1mm turns the tires 45 degrees. This is **Low Resolution**.

ArduPilot suffers from both.

- **The Mistake:** Users connect the pushrod to the **outermost hole** on the servo arm and the **innermost hole** on the control surface horn.
  - **Result:** A tiny servo movement creates a HUGE surface movement.
  - **The Math:** If your servo has 1000 steps of resolution, and that throw moves the aileron 90 degrees, you get ~0.1 degrees per step. But you only need 15 degrees of travel to fly! You are throwing away 80\% of your servo's precision.
  - **The Symptom:** The plane "hunts" or wobbles in level flight because the servo cannot move "just a little bit." It has to move "too much" or "not at all."

3.2 THE ENGINEERING FIX

- **Servo Arm:** Use the **Innermost** hole possible.
- **Control Horn:** Use the **Outermost** hole possible.
- **Result:** The servo has to move 60 degrees to move the surface 20 degrees. You have tripled your torque and tripled your resolution. ArduPilot can now make microscopic corrections for butter-smooth flight.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





4. ARDUPILOT INTEGRATION: THE SOFTWARE LAYER

Once the physics are sound, you must configure the software to respect them.

4.1 STALL SPEED VS. TRIM SPEED (`TRIM_ARSPD_CM`)

- **The Danger:** Users often set their cruise speed (`TRIM_ARSPD_CM`) too close to the stall speed to "save battery."
- **The Turn Trap:** When a plane banks 60 degrees, it must generate **2Gs** of lift to stay level (1 / cos(60)). This increases the stall speed by 41\% (sqrt(2)).
- **Scenario:** You cruise at 12m/s. Your stall speed is 10m/s. You enter a sharp turn. Your stall speed jumps to 14.1m/s. **You are now stalled.** The inner wing drops, and the plane enters a spin.
- **Rule:** Set `TRIM_ARSPD_CM` to at least **1.3x** your measured stall speed.

4.2 THE "HARD DECK" (`ARSPD_FBW_MIN`)

This parameter is the safety floor.

- **Function:** If airspeed drops below this value, ArduPilot will **pitch the nose down**, regardless of your altitude (even if you are 2 meters off the ground).
- **Why?** It is better to crash under control than to stall and spin. A controlled descent gives you a chance to recover energy; a stall is chaos.
- **Tuning:** Set this to roughly equal your level-flight stall speed. This gives the autopilot permission to dive to regain energy if the motor fails or the wind shears.

4.3 `PTCH2SRV_RLL` (THE COORDINATE TURN)

- **Physics:** When a plane rolls, it loses vertical lift. It needs "Up Elevator" to keep the nose up in the turn.
- **The Parameter:** `PTCH2SRV_RLL` tells the integrator how much elevator to add based on the bank angle.
- **Tuning:** If the nose drops every time you turn, increase this. If the nose pitches up and the plane climbs in turns, decrease it.

MULTICOPTER FRAME MECHANICS

CRITICAL The frame is not just a skeleton; it is a mechanical component of the control loop. If the frame flexes, the PID loop fails.

1. GEOMETRY & MIXING: THE DEADCAT PROBLEM

1.1 TRUE X

- **Geometry:** Motors are equidistant from the center. Symmetrical.
- **Physics:** Pitch and Roll authority are identical.
- **Verdict:** The perfect geometry for control theory. Used for Racing.

1.2 DEADCAT (DC)

- **The Design:** Front arms are swept back/out to keep props out of the camera view.
- **The Problem:** The **Center of Thrust** is no longer aligned with the **Center of Mass**.
- **The Consequence:**
  - **Coupling:** A pure Pitch command also creates a Yaw moment. A pure Roll command creates unequal motor loading.
  - **Motor Saturation:** In a hard yaw spin, two diagonal motors may hit 100\% throttle while the others are at 10\%. The drone runs out of overhead ("authority") and drifts.
- **The Fix:** ArduPilot's `AP_MotorsMatrix`.
  - You must select the correct frame type. ArduPilot calculates a custom **Mixing Matrix** that adjusts the motor output factors to account for the asymmetry.
  - **Note:** You must run **AutoTune**. The PID gains for Pitch and Roll will be significantly different on a Deadcat frame.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





2. MATERIAL SCIENCE: STIFFNESS VS. RESONANCE

Carbon Fiber is conductive, stiff, and light. But not all carbon is equal.

2.1 STIFFNESS (MODULUS)

- **Toray T700:** High modulus (Stiff).
- **The Physics:** We want the frame to be infinitely stiff.
  - **Why?** If the arm flexes, it acts as a "spring." When the motor accelerates, the arm bends back, *then* snaps forward.
  - **Phase Lag:** This bending creates a time delay between the motor command and the vehicle movement.
  - **Tuning:** Phase Lag limits your P-Gains. If you try to tune a flexible frame tightly, it will oscillate because the FC is reacting to delayed motion.

2.2 RESONANCE FREQUENCY

Every object has a "Resonant Frequency"—the note it hums when you flick it.

- **The Source:** Motors vibrate at the frequency of their rotation (e.g., 150Hz to 400Hz).
- **The Danger Zone:** If your frame's resonant frequency overlaps with your motor RPM frequency, the vibration amplifies 10x.
  - **Result:** The Gyro gets swamped with noise.
- **The Solution:** Stiffer frames (thicker arms, 5mm-6mm) push the resonant frequency **UP** (e.g., to 600Hz), moving it safely away from the primary motor frequencies.

---

3. VIBRATION MANAGEMENT

3.1 SOFT MOUNTING (THE STACK)

Since we can't eliminate all vibration, we must isolate the Brain.

- **Gummies:** Silicone grommets act as a mechanical Low-Pass Filter.
- **The Tuning Trap:**
  - **Too Hard:** Vibration gets through → Hot Motors.
  - **Too Soft:** The stack "wobbles" during flips. The Gyro sees this wobble as vehicle movement.
  - **Symptom:** "Bounce back" at the end of a flip. The FC fights the momentum of its own mounting gummies.

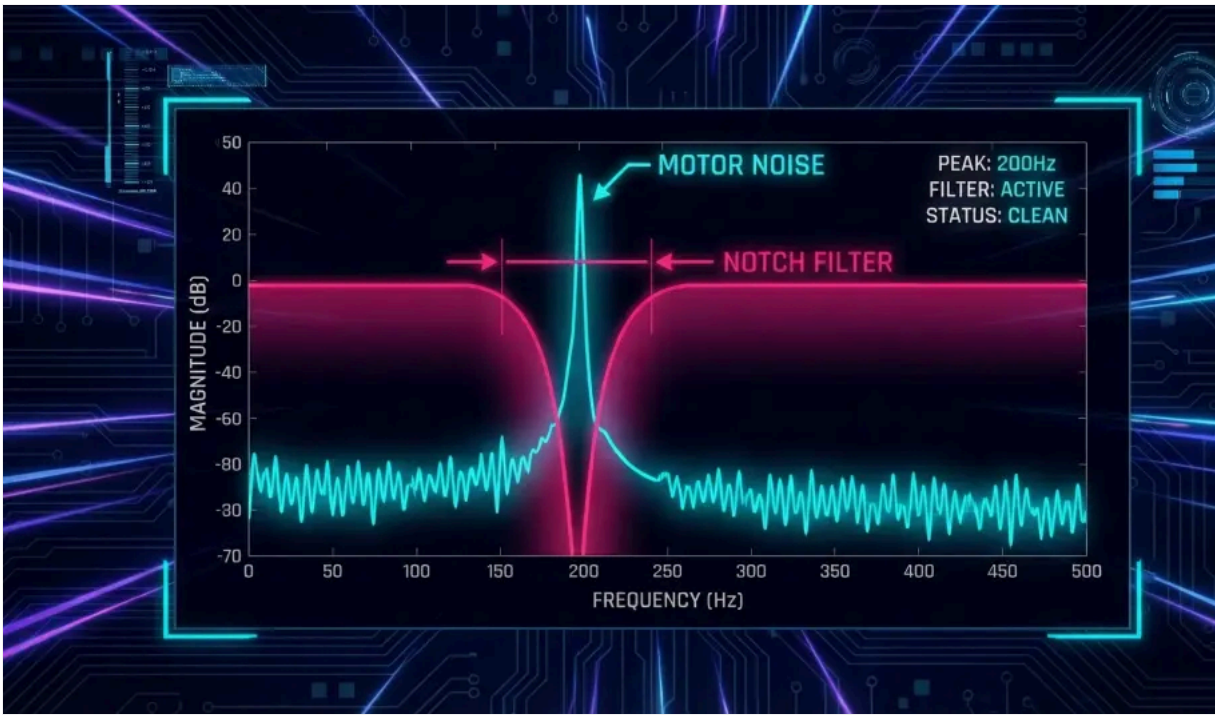
3.2 NOTCH FILTERING: THE SOFTWARE SOLUTION

ArduPilot's **Harmonic Notch Filter** is the magic bullet.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY







- **How it works:** It uses the RPM telemetry from the ESCs to determine *exactly* what frequency the motors are vibrating at in real-time.
- **The Action:** It places a digital "Notch" filter at that frequency on the Gyro data.
- **The Benefit:** It deletes the noise without adding the **Phase Lag** of a traditional Low-Pass filter.
- **Result:** You can fly a bent, noisy, resonant frame and it will still feel smooth, because the FC is blind to the vibration.

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

CHAPTER 2: POWER SYSTEMS

UAV POWER SYSTEMS: CHEMISTRY & PHYSICS

CRITICAL The battery is not just a fuel tank; it is a voltage source with internal resistance. Understanding this resistance is key to preventing mid-air failures (voltage sag) and fires.

1. BATTERY CHEMISTRY: THE ELECTRON EXCHANGE

1.1 LIPO (LITHIUM POLYMER)

The standard for high-performance flight.

- **Structure:** Uses a solid polymer electrolyte.
- **Voltage:** 3.7V Nominal / 4.20V Max.
- **Advantages:** Extremely low Internal Resistance (IR). Can dump massive current (100A+) instantly.
- **The "Puff":** When a LiPo is abused (over-discharged or overheated), the electrolyte decomposes into gas (mostly Oxygen and Hydrogen). This gas is trapped in the foil pouch, causing it to "puff." **A puffed LiPo is a fire hazard.** The internal structure is compromised, and the layers can short-circuit.

1.2 LIHV (HIGH VOLTAGE LIPO)

- **Voltage:** 3.8V Nominal / 4.35V Max.
- **The Trade-off:** By charging to a higher voltage, you get ~10\% more energy density. However, the chemical stress on the cathode is significantly higher.
- **Lifespan:** LiHV cells degrade faster than standard LiPo cells if consistently charged to 4.35V. They are for racing/performance, not longevity.

1.3 LI-ION (CYLINDRICAL CELLS)

Used for Long Range / Endurance. Common cells: **Sony VTC6** (18650), **Molicel P42A** (21700).

- **Density:** Superior Energy Density (Wh/kg). They hold more fuel for the weight.
- **Power:** Inferior Power Density (W/kg). They have higher Internal Resistance.
- **The Sag:** Because of high IR, they sag heavily under load.
  - *LiPo:* 100\% throttle → Voltage drops 1V.
  - *Li-Ion:* 100\% throttle → Voltage drops 4V.

---

2. THE C-RATING LIE VS. INTERNAL RESISTANCE (IR)

2.1 THE MARKETING LIE

Manufacturers label packs with "100C" or "150C."

- **The Math:** A 1300mAh (1.3Ah) battery at 100C claims to deliver $1.3 \times 100 = 130$ Amps continuous.
- **The Reality:** The XT60 connector melts at 60A. The 12AWG wire melts at 100A. The "C-Rating" is effectively a marketing number with no standardized physical basis.

2.2 THE REAL METRIC: INTERNAL RESISTANCE (IR)

IR is the impedance to electron flow inside the cell. It is measured in milliohms ($m\Omega$).

- **Ohm's Law:** $V_{drop} = I \times R$.
- **Example:** If you pull 50 Amps from a pack with $10m\Omega$ (0.01 Ω) total resistance:

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





$$V_{drop} = 50 \times 0.01 = 0.5 \text{ Volts lost as heat inside the pack.}$$

- **Heat:** $P = I^2R$. $50^2 \times 0.01 = 25$ Watts of heat generated inside the battery.
- **Rule of Thumb (per cell):**
  - **< 2** $m\Omega$**:** Race Grade.
  - **2 - 5** $m\Omega$**:** Good / Freestyle.
  - **> 10** $m\Omega$**:** Old / Degraded. Relegate to goggle battery or disposal.
- **Temperature Factor:** IR decreases as heat increases. This is why batteries perform better in summer than winter. Pre-heating LiPos to 30°C before a race increases performance.

---

3. POWER DISTRIBUTION ARCHITECTURE

3.1 CONNECTORS: THE BOTTLENECK

The connector is often the point of highest resistance in the circuit.

- **XT30:** Rated 30A continuous. Use for micro drones (< 250g).
- **XT60:** Rated 60A continuous. The industry standard. Realistically handles 100A bursts (3s).
- **XT90:** Rated 90A continuous. Use for X-Class or Cinelifters (10").
- **AS150:** Anti-Spark. Mandatory for high-voltage (12S+) setups to prevent the "pop" that erodes contacts.

3.2 CAPACITORS: THE ELECTRONIC SHOCK ABSORBER

Electronic Speed Controllers (ESCs) chop voltage thousands of times per second. This creates massive inductive voltage spikes ("Noise").

- **The Physics:** A capacitor resists changes in voltage. It acts as a reservoir, filling up during spikes and draining during dips.
- **Low ESR:** You must use **Low ESR** (Equivalent Series Resistance) capacitors (e.g., Panasonic FM/FR series). Standard capacitors cannot react fast enough to filter 96kHz switching noise.
- **Placement:** Must be soldered *at the source* of the noise—directly onto the ESC battery pads. The longer the wire between the ESC and the Cap, the less effective it is (due to wire inductance).

3.3 WIRE GAUGE

Voltage drop over wire length is significant at low voltages (DC).

- **Battery Leads:** 12AWG or 14AWG.
- **Motor Wires:** 18AWG or 20AWG.
- **Signal Wires:** 28AWG (silicone).

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





CHAPTER 3: FLIGHT MODES

ACRO MODE (COPTER)

EXECUTIVE SUMMARY

Acro (Acrobatic) mode is the purest form of flight control, providing direct "Rate" control over the vehicle's angular velocity. Unlike Stabilize or Loiter, the stick inputs command rotation speed (degrees per second), not an angle or position. It is the primary mode for FPV racing and freestyle flying.

THEORY & CONCEPTS

1. RATE CONTROL VS. ANGLE CONTROL

- **Angle Control (Stabilize):** If you push the stick 50\% Right, the drone tilts 22 degrees Right and **stops**. When you let go, it levels itself.
- **Rate Control (Acro):** If you push the stick 50\% Right, the drone starts rolling Right at 180 deg/s and **continues rolling** until you center the stick. When you let go, it stays at whatever angle it was at.
- *Analogy:* Angle control is like a steering wheel (turn 10 degrees, car turns 10 degrees). Rate control is like a joystick in a fighter jet (push right to roll, center to stop rolling).

2. BODY FRAME PHYSICS

In Acro mode, rotation is calculated in the **Body Frame**.

- **The Effect:** If you are pitched 90 degrees forward (nose down) and you apply Yaw, the drone spins around its own vertical axis, which looks like a "Roll" relative to the ground.
- *Why?* This is essential for 3D maneuvers like "Power Loops" and "Matty Flips," where the pilot needs the drone to rotate relative to its own cockpit, not the Earth's horizon.

HARDWARE DEPENDENCY MATRIX

Acro is the most robust flight mode because it requires the absolute minimum sensor set.

SENSOR	REQUIREMENT	CODE IMPLEMENTATION NOTES
Gyroscope	CRITICAL	The core rate controller relies entirely on gyro feedback to maintain the requested angular velocity.
Accelerometer	OPTIONAL	Only required if `ACRO_TRAINER` is enabled (for leveling) or for "AirMode" throttle boosting (which uses vertical acceleration estimates). Pure rate mode works without it.
GPS	**NONE**	Completely ignored.
Compass	**NONE**	Heading is maintained via gyro integration (drift is possible over long periods).
Barometer	OPTIONAL	Used for altitude stabilization if throttle is centered, but throttle is largely manual.

CONTROL ARCHITECTURE (ENGINEER'S VIEW)

Acro operates in the **Body Frame**. This is a crucial distinction from "Earth Frame" modes like Stabilize or Sport.

1. **Input Shaper**:
   - Pilot sticks (normalized -1 to 1) are converted to target **Angular Rates** (deg/s).
   - *Code Path:* `get_pilot_desired_angle_rates()`.
   - *Expo & Super Rates:* The input is shaped by `ACRO_RP_EXPO` and `ACRO_RP_RATE` (or `ATC_INPUT_TC`) to define the feel.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





2. **Acro Trainer (Virtual Flybar):**
    - If `ACRO_TRAINER` is enabled, the code calculates a "Leveling Rate" proportional to the current lean angle error.
    - This leveling rate is **mixed** with the pilot's request.
    - *Logic:* As you move the stick further, the pilot's rate request overrides the leveling request. At center stick, the leveling request dominates, bringing the vehicle flat.
3. **Rate Controller:**
    - The target rates are fed into the PID loop (`ATC_RAT_RLL_P/I/D`, etc.).
    - The PID controller drives the motors to match the gyro rate to the target rate.

PILOT INTERACTION

- **Stick Center:** The vehicle maintains its *current* attitude (it stays tilted). If `ACRO_TRAINER` is on, it slowly levels out.
- **Stick Deflection:** The vehicle rotates at a speed proportional to stick deflection. Full stick = `ACRO_RP_RATE` (e.g., 360 deg/s).
- **Throttle:** Manual control. However, if "AirMode" is active (via `ATC_THR_MIX_MAN` or `ACRO_OPTIONS`), the mixer will prioritize attitude control over throttle, potentially spooling up motors even at zero throttle to maintain stability during maneuvers.

FAILSAFE LOGIC

Acro is often the *target* of a failsafe, rather than having failsafes of its own.

- **GPS/Compass Loss:** No effect. Acro continues to function perfectly.
- **GCS Failsafe:** Standard throttle failsafe applies (usually RTL or Land).
- **Crash Detection:** If the crash check logic detects a crash (high accel, no rotation), it may disarm the vehicle to prevent damage.

KEY PARAMETERS

PARAMETER	DEFAULT	DESCRIPTION
`ACRO_RP_RATE`	360	Maximum rotation rate (deg/s) for Roll/Pitch at full stick.
`ACRO_Y_RATE`	202.5	Maximum rotation rate (deg/s) for Yaw.
`ACRO_TRAINER`	2	0=Disabled (Pure Rate), 1=Leveling, 2=Leveling & Limited (prevents flipping).
`ACRO_BAL_ROLL`	1.0	"Virtual Flybar" strength. Higher values make it level faster when sticks are released (if Trainer is active).
`ATC_THR_MIX_MAN`	0.1	Defines how much authority the attitude controller has over the throttle. Higher values (0.5+) create "AirMode" behavior for hang-time stability.

TUNING & TROUBLESHOOTING

SYMPTOM	PROBABLE CAUSE	CORRECTIVE ACTION
Bounce-back after Flip	D-Term too low or P-Term too high.	Increase `ATC_RAT_RLL_D` / `ATC_RAT_PIT_D` slightly to dampen the stop.
Sluggish Response	Low Rate or high Filtering.	Increase `ACRO_RP_RATE`. Check `INS_GYRO_FILTER` (ensure it's not too low, e.g., < 40Hz).
Drift when Level	Accelerometer trim or Gyro bias.	Perform Accel Calibration (`AHRS_TRIM_X/Y`). In pure Acro, some drift is expected over time.
Wobble on Descent	Prop wash.	Increase `ATC_THR_MIX_MAN` (AirMode) to keep PID loops active at low throttle.

SOURCE CODE REFERENCE

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- **Mode Logic:** `ardupilot/ArduCopter/mode_acro.cpp`
- **Rate & Trainer Logic:** `ModeAcro::get_pilot_desired_angle_rates()`
- **Control Loop:** `ModeAcro::run()`

PRACTICAL GUIDE: UNLOCKING FPV POTENTIAL

The defaults are safe for a DJI clone, but terrible for a 5" racer.

1. ENABLE TRUE ACRO

- **Set** `ACRO_TRAINER = 0`.
- The default (2) limits your roll angle and auto-levels the drone. This prevents flips and rolls. Disable it immediately for freestyle.

2. INCREASE THE RATES

- **Parameter:** `ACRO_RP_RATE`
- **Default:** 360 deg/s.
- **FPV Standard: 667** (approx Betaflight 1.0) or **900**.
- **Max:** 1080 deg/s.

3. ENABLE AIRMODE

- **Parameter:** `ATC_THR_MIX_MAN`
- **Default:** 0.1 (10\% authority).
- **Tuning:** Increase to **0.5** (50\%).
- **Why?** When you chop the throttle to zero to dive a building, the PID loop needs authority to keep the drone stable. Without this, the drone tumbles.

TRAINING WITH AR

Learning Acro mode can be disorienting because you have no self-leveling reference. Pilots often lose track of their horizon or altitude.

Using Supported AR Glasses with MAVLink HUD allows you to keep a **virtual artificial horizon** and **altitude tape** in your field of view while maintaining line-of-sight with the drone. This "Training Wheels" approach significantly reduces crash risk during the learning phase by giving you an objective reference for "Level".

AIRMODE (COPTER)

EXECUTIVE SUMMARY

AirMode is not a distinct flight mode but a specific **stabilization feature** that can be enabled within modes like Acro, Stabilize, and AutoTune. Its primary purpose is to maintain full authority of the PID controllers even at zero throttle. This allows the vehicle to maintain its attitude during free-fall, inverted hang-time, or aggressive descents where the throttle is cut to minimum.

THEORY & CONCEPTS

1. THE PID DECAY PROBLEM

In standard stabilization, when the throttle is at 0\%, the flight controller reduces the authority of the PID loops. This is to prevent the drone from "twitching" or flipping over while it is sitting on the ground idling.

- **The Downside:** If you are flying high and cut the throttle to 0\% to "free fall," the drone loses its ability to stay level. A gust of wind can tumble it.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

- **The Solution:** AirMode. It forces the PID loops to stay at 100\% authority regardless of throttle position.

2. AUTHORITY VS. THRUST

AirMode separates the concepts of **Rotation** (Torque) and **Altitude** (Thrust).

- Even if you want 0\% net thrust, AirMode will spin up Motor 1 and spin down Motor 4 to create a roll.
- *Result:* You gain "Hang Time" stability. You can perform complex acrobatic tricks in a vacuum of throttle without losing control of the aircraft's orientation.

HARDWARE DEPENDENCY MATRIX

AirMode is a pure software logic feature affecting the motor mixer.

SENSOR	REQUIREMENT	CODE IMPLEMENTATION NOTES
Gyroscope	CRITICAL	AirMode relies on gyro feedback to maintain attitude authority when motors are at idle.
Throttle	**INPUT**	AirMode modifies how the system reacts to low/zero pilot throttle input.

CONTROL ARCHITECTURE (ENGINEER'S VIEW)

To understand AirMode, you must understand the ArduPilot Mixer strategy.

1. **The "Minimum" Mix:** Normally, when the pilot drops the throttle to zero, the mixer switches to a "Minimum" state. In this state, the PID output is constrained to prevent the motors from spinning up significantly. This is a safety feature to prevent the drone from flipping over on the ground while idling.
2. **The "Manual" Mix (AirMode):** When AirMode is active, the mixer is forced to stay in the "Manual" state (`ATC_THR_MIX_MAN`) even at zero throttle.
    - *Mechanism:* The code sets `_throttle_rpy_mix_desired = _thr_mix_man` .
    - *Result:* The PID loops retain full authority. If the drone tilts uncommanded (e.g., wind gust in freefall), the motors will aggressively spin up to correct it, even if the throttle stick is at bottom.
    - *Code Path:* `update_throttle_mix()` logic overrides the standard spool-down logic.

ACTIVATION METHODS

AirMode is not selected like "Loiter" or "RTL". It is enabled via:

1. **Parameters:** Setting `ACRO_OPTIONS` bit 1 (Force AirMode) permanently enables it for Acro mode.
2. **RC Switch:** Assigning `RCx_OPTION` to `84` (AirMode) allows a toggle switch.
3. **Arming:** Assigning `RCx_OPTION` to `154` (ArmDisarmAirMode) enables it immediately upon arming.

FAILSAFE & SAFETY LOGIC

AirMode introduces a specific risk profile, particularly during landing.

1. **Landing Bounce:**
    - *Risk:* When the drone touches the ground, the impact creates a sudden gyro spike (shock).
    - *Reaction:* In standard modes, zero throttle dampens the reaction. In AirMode, the P-term sees this spike as an error and commands a motor surge to "correct" the attitude. This causes the drone to bounce or "freak out" on the ground.
    - *Mitigation:* ArduPilot increases the landing detector delay (`LAND_AIRMODE_DETECTOR_TRIGGER_SEC` , default 3s) to prevent false-positives, but this means you must be ready to disarm instantly upon touchdown.
2. **Safety Recommendation:** It is highly recommended to assign AirMode to a switch so you can turn it **OFF** just before landing to ensure a smooth touchdown.

KEY PARAMETERS

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





PARAMETER	DEFAULT	DESCRIPTION
`ATC_THR_MIX_MAN`	0.1	(0.1 - 0.9) Defines the ratio of throttle vs attitude authority. Higher values give the PID loop more "headroom" to spin up motors at zero throttle. Typical AirMode feel requires 0.5+.
`ACRO_OPTIONS`	0	Bitmask. Set bit 1 to "Force AirMode" in Acro.
`LAND_AIRMODE_DETECTOR_TRIGGER_SEC`	3.0	Time (seconds) the landing detector waits before declaring a landing when AirMode is active.

SOURCE CODE REFERENCE

- **Mixer Logic:** `Copter::update_throttle_mix()`
- **Attitude Control Interface:** `AC_AttitudeControl_Multi::set_throttle_mix_man()`

PRACTICAL GUIDE: THE "LANDING SWITCH"

The safest way to fly AirMode is to have a dedicated switch.

1. **Assign Switch:** Set `RC7_OPTION = 84` (AirMode).
2. **Takeoff:** Take off in Stabilize/Loiter with AirMode **OFF**.
3. **Flight:** Once airborne, flip the switch **ON**. Enjoy the hang time.
4. **Landing:**
    - Approach the ground.
    - When 1 meter high, flip AirMode **OFF**.
    - Land normally. The motors will settle peacefully.
    - *Why?* This prevents the "P-Term Explosion" where the drone bounces violently if it hits the ground while AirMode is active.

ALTITUDE HOLD (COPTER)

EXECUTIVE SUMMARY

Altitude Hold (AltHold) is a semi-autonomous flight mode that automatically maintains a consistent Z-axis altitude while allowing the pilot to manually control roll, pitch, and yaw. It acts as a "cruise control" for height, relieving the pilot of throttle management.

THEORY & CONCEPTS

1. Z-AXIS PHYSICS: THE GRAVITY NEUTRAL POINT

To hold altitude, the drone must produce exactly enough thrust to counter gravity.

- **The Problem:** Battery voltage drops during flight, and air density changes with height. "50\% Throttle" does not produce a constant amount of lift.
- **The Solution:** The Altitude Controller learns the **Hover Throttle** (`MOT_THST_HOVER`) dynamically. It continuously calculates the "Neutral Point" where the drone neither climbs nor falls.

2. BAROMETRIC DRIFT VS. INERTIAL CERTAINTY

Barometers are sensitive to noise. If you rely solely on a barometer, the drone will "bounce" as the air pressure fluctuates.

- **Fusion:** ArduPilot fuses the Barometer with the **Z-Accelerometer**.
- **The Logic:** The Accelerometer detects instant changes (e.g., "I just dropped 5cm!"), while the Barometer provides the long-term truth (e.g., "I am 10m high"). This fusion creates a smooth, lag-free altitude hold.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





HARDWARE DEPENDENCY MATRIX

Unlike Stabilize or Acro, this mode depends heavily on a Z-axis estimator.

SENSOR	REQUIREMENT	CODE IMPLEMENTATION NOTES
Barometer	REQUIRED	Primary source for altitude estimation. If the barometer drifts (e.g., due to light hitting it or air pressure changes), the drone will drift vertically.
Accelerometer	CRITICAL	Used for inertial navigation to fuse with barometer data for a low-latency Z-estimate.
Rangefinder	OPTIONAL	If installed and enabled (`RNGFND_GAIN` > 0), the EKF can use it for low-altitude precision, but the mode falls back to Baro seamlessly.
GPS	**NONE**	Not required. Ideal for indoor flight.

CONTROL ARCHITECTURE (ENGINEER'S VIEW)

AltHold is effectively two different modes running simultaneously:

1. **Horizontal (X/Y): Stabilize Mode**
   - Pilot Roll/Pitch sticks map directly to **Lean Angle** (e.g., 45 degrees).
   - When sticks are centered, the vehicle self-levels.
   - *Note:* It does *not* hold horizontal position; it will drift with the wind.
2. **Vertical (Z): Velocity Controller**
   - **Pilot Input:** The throttle stick commands a **Climb Rate** (cm/s), not a motor power level.
   - **The Cascade:**
     - *Input:* `get_pilot_desired_climb_rate()` converts stick position to target velocity.
     - *Velocity Loop:* Compares target vs actual climb rate. Error feeds into...
     - *Accel Loop:* Compares target vs actual Z-acceleration. Error feeds into...
     - *Motor Mixer:* Final throttle output.

PILOT INTERACTION & DEADZONE

The throttle behavior is defined by the **Deadzone** (`THR_DZ`).



Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

- **Center Stick (Deadzone):** When the throttle is within the deadzone (typically 40\%-60\%), the target climb rate is forced to **0**. The controller locks the current altitude.
- **Outside Deadzone:** The stick commands a linear climb/descent rate up to `PILOT_SPEED_UP` (default 250 cm/s) or `PILOT_SPEED_DN`.
- **Tactile Feel:** This creates a "sticky" feel at center, allowing you to let go of the throttle without the drone falling.

FAILSAFE & SAFETY LOGIC

- **Vibration Sensitivity:** Because the Z-controller relies heavily on the Accelerometer (Z-axis) to react fast, **high vibrations** can cause AltHold to fail.
  - *Symptom:* The drone shoots up (Skyrocket) when engaging AltHold.
  - *Cause:* Aliasing of vibration noise into the Z-accel signal makes the EKF think the drone is falling, so it applies max throttle.
- **Barometer Glitches:** Sudden pressure changes (opening a door, sunlight hitting the sensor) can cause jumpy altitude corrections.

KEY PARAMETERS

PARAMETER	DEFAULT	DESCRIPTION
`PILOT_SPEED_UP`	250	Max climb rate (cm/s) at full throttle.
`PILOT_SPEED_DN`	0	Max descent rate (cm/s) at zero throttle. 0 means it matches SPEED_UP.
`THR_DZ`	100	Size of the deadzone (in PWM/10). 100 = +/- 10\% around mid-stick.
`PSC_POSZ_P`	1.0	Position Z P-gain. Converts altitude error to target velocity.
`PSC_VELZ_P`	5.0	Velocity Z P-gain. Converts velocity error to target acceleration.

TUNING & TROUBLESHOOTING

SYMPTOM	PROBABLE CAUSE	CORRECTIVE ACTION
"Rocketing" Upwards	High Z-axis Vibrations.	Check prop balance, secure flight controller. Check `[VIBE](/field-manual/log-analysis/vibration-analysis.html).VibZ` in logs.
Yo-Yo (Oscillation)	`PSC_VELZ_P` too high.	Reduce Velocity P gain.
Drifts Down/Up slowly	Barometer light sensitivity or inadequate "Hover Throttle" learning.	Cover barometer with foam. Ensure `MOT_THST_HOVER` is learning (`MOT_HOVER_LEARN`).

SOURCE CODE REFERENCE

- **Mode Logic:** `ardupilot/ArduCopter/mode_althold.cpp`
- **Pilot Input Logic:** `Copter::get_pilot_desired_climb_rate()`
- **Loop Structure:** `ModeAltHold::run()` calls `pos_control->update_z_controller()`.

AUTO MODE (COPTER)

EXECUTIVE SUMMARY

Auto Mode is the primary autonomous flight mode, executing a pre-programmed mission plan uploaded to the flight controller. It navigates through a series of 3D waypoints using a sophisticated trajectory generation system (SCurves) to ensure smooth,

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

cinematic motion. It is the "brain" of commercial operations, handling everything from takeoff to landing without pilot input.

THEORY & CONCEPTS

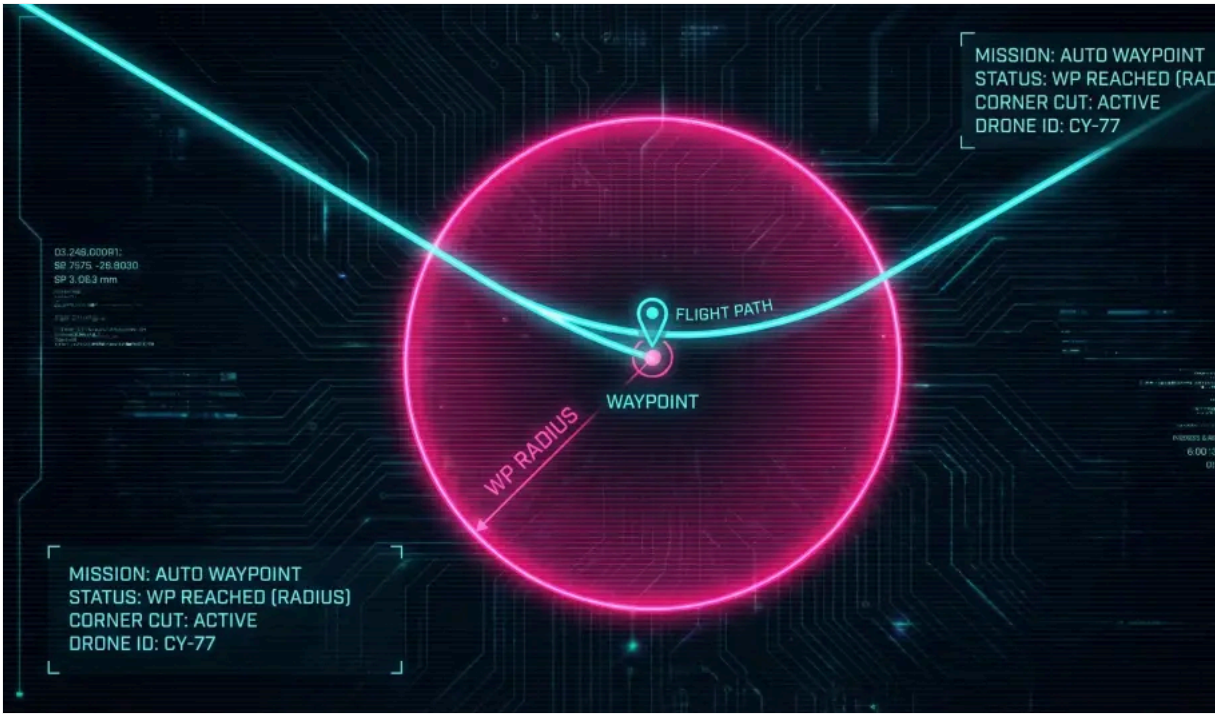
1. 3D WAYPOINT NAVIGATION

Waypoint navigation is more than just flying from A to B. It requires solving a 3D path problem.

- **The Problem:** Drones have inertia. They cannot turn 90 degrees instantly.
- **The Solver:** ArduPilot calculates a **Path Trajectory**. It "leads" the drone by placing a target point slightly ahead of the actual position.
- **SCurves:** By using S-Curve math, the drone accelerates and decelerates like a smooth cinematic camera rig rather than a jerky robot.

2. THE ACCEPTANCE RADIUS

How does the drone know it "Reached" a waypoint?



- **The Zone:** `WPNAV_RADIUS` defines a sphere around the point.
- **The Logic:** Once the drone enters this sphere, the navigator immediately begins "Blended Tracking" towards the *next* point. This allows the drone to carry its speed through the turn rather than coming to a full stop.

HARDWARE DEPENDENCY MATRIX

Auto mode relies entirely on the vehicle's ability to know *where* it is in the world.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

SENSOR	REQUIREMENT	CODE IMPLEMENTATION NOTES
GPS	CRITICAL	Requires a 3D Lock and a healthy EKF position estimate. Loss of GPS triggers a failsafe (usually Land or AltHold).
Compass	CRITICAL	Accurate heading is required for navigation. EKF compass variance errors will prevent mode engagement.
Barometer	REQUIRED	Primary source for relative altitude (AGL).
Rangefinder	OPTIONAL	Can be used for "Terrain Following" missions (`TERRAIN_ENABLE`), keeping the drone at a fixed height above ground.

CONTROL ARCHITECTURE (ENGINEER'S VIEW)

Auto Mode is unique because it is a **Meta-Mode** containing its own internal state machine.

1. **The Sub-Mode Machine:**
   - Unlike other modes, `ModeAuto::run()` acts as a dispatcher.
   - It switches between internal states: `TAKEOFF` , `WP` (Waypoint), `LAND` , `RTL` , `CIRCLE` , `LOITER` , `PAYLOAD_PLACE` .
   - *Code Path:* `ModeAuto::run()` .
2. **Trajectory Generator (SCurves):**
   - ArduPilot uses **SCurve** navigation (S-shaped velocity profiles).
   - Instead of flying straight lines with sharp stops, it calculates a jerk-limited path. This means the drone accelerates smoothly, cruises, and decelerates smoothly into corners.
   - *Benefit:* Cinematic video and reduced mechanical stress.
3. **Mission Command Handler:**
   - The `AP_Mission` library feeds commands (e.g., `MAV_CMD_NAV_WAYPOINT`) to the mode.
   - The mode interprets these commands and updates the `AC_WPNav` targets.

PILOT INTERACTION & OVERRIDES

Even in Auto, the pilot is not necessarily locked out. This is controlled by **Stick Mixing**.

- **Yaw Override:** By default, the pilot can manually yaw the vehicle while it flies the mission path. This is useful for pointing a camera. (Disable with `AUTO_OPTIONS` bit 2).
- **Speed Nudging:** In some configurations, the pitch stick can speed up or slow down the mission execution.
- **Landing Repositioning:** If `LAND_REPOSITION` is enabled, the pilot can use the roll/pitch sticks during the final landing phase to "nudge" the drone away from obstacles.

FAILSAFE LOGIC

- **GPS Failsafe:** If the EKF loses position confidence (GPS Glitch or Loss), the vehicle will immediately switch to **Land** (if `FS_EKF_ACTION` is 1) or **AltHold** (if 3). This is a critical safety behavior to understand: *the drone will stop navigating.*
- **GCS Failsafe:** If the mission continues beyond radio range, the drone continues *unless* `FS_GCS_ENABLE` is set to "RTL". For autonomous missions, GCS failsafe is often disabled.

KEY PARAMETERS

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





PARAMETER	DEFAULT	DESCRIPTION
`WPNAV_SPEED`	500	Default horizontal speed (cm/s) between waypoints.
`WPNAV_ACCEL`	100	Max acceleration (cm/s^2). Lower values make starts/stops smoother.
`AUTO_OPTIONS`	0	Bitmask options. Bit 2: Ignore Pilot Yaw (locks heading to mission).
`WPNAV_RADIUS`	200	(cm) The radius around a waypoint at which the drone considers the point "reached" and begins the turn to the next point.
`RTL_AUTOLAND`	0	Time (ms) to wait at the final waypoint.

TUNING & TROUBLESHOOTING

SYMPTOM	PROBABLE CAUSE	CORRECTIVE ACTION
"Stop-and-Go" at Waypoints	The vehicle stops at every point instead of flowing.	Increase `WPNAV_RADIUS`. If the turn radius is too tight for the speed, it *must* stop to turn.
Overshooting Corners	`WPNAV_ACCEL` too low or Speed too high.	Increase Accel or reduce Speed to allow the drone to corner tighter.
Jerky Yaw turns	`ATC_ACCEL_Y_MAX` too high.	Reduce Yaw Acceleration max to smooth out automated heading changes.

SOURCE CODE REFERENCE

- **Mode Logic:** `ardupilot/ArduCopter/mode_auto.cpp`
- **Waypoint Runner:** `ModeAuto::wp_run()`

PRACTICAL GUIDE: TUNING FOR CINEMATIC WAYPOINTS

Default ArduPilot tuning is "Safe" (robotic). For cinematic video, you need to soften the edges.

1. SMOOTH THE CORNERS

If the drone stops at every waypoint, your video looks robotic.

- **Parameter:** `WPNAV_RADIUS`
- **Default:** 200 (2m).
- **Tuning:** Increase to **300 - 500** (3-5m). This tells the drone to start cutting the corner earlier, maintaining momentum.

2. REMOVE THE "JERK"

"Jerk" is the change in acceleration. High jerk causes gimbal micro-shakes.

- **Parameter:** `WPNAV_JERK`
- **Default:** 1 ($1m/s^3$).
- **Tuning:** Reduce to **0.5**. This makes the drone ramp up its speed very slowly, like a train leaving a station.

3. VERTICAL SMOOTHNESS

The default vertical acceleration is quite aggressive ($100cm/s^2$).

- **Parameter:** `WPNAV_ACCEL_Z`
- **Default:** 100.
- **Tuning:** Reduce to **50 - 70**. This prevents the drone from "popping" up to altitude, making elevation changes feel like a gentle elevator.

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





BRAKE MODE (COPTER)

EXECUTIVE SUMMARY

Brake Mode is a dedicated safety mode designed to stop the vehicle as quickly as possible and hold its position. It is often assigned to a "Panic Switch" or triggered by the "Pause" button on GCS controllers (like the old 3DR Solo controller). Once engaged, it ignores all pilot inputs until switched out or a timeout occurs.

THEORY & CONCEPTS

1. KINETIC ENERGY DISSIPATION

Stopping a drone moving at 20 m/s requires dissipating a massive amount of kinetic energy.

- **The Problem:** If you just "level" the drone, it will coast for 50 meters due to momentum.
- **The Solution:** The Brake controller calculates the **Deceleration Vector**. It commands a negative pitch/roll to produce a horizontal thrust component that actively fights the drone's current velocity.

2. OPEN-LOOP VS. CLOSED-LOOP STOP

- **Manual (Open-Loop):** You tilt the drone back, but you have to guess when to level it.
- **Brake (Closed-Loop):** The EKF monitors the GPS velocity. As the speed reaches zero, the controller smoothly reduces the "back-angle" to exactly 0, preventing the drone from flying backwards after it stops.

HARDWARE DEPENDENCY MATRIX

Since Brake mode relies on stopping and holding a 3D position, it has similar requirements to Loiter.

SENSOR	REQUIREMENT	CODE IMPLEMENTATION NOTES
GPS	CRITICAL	Requires a valid position estimate to calculate velocity errors and hold position after stopping.
Compass	CRITICAL	Required for the inertial navigation system to function correctly.
Barometer	REQUIRED	Used for altitude hold (Z-axis stop).

CONTROL ARCHITECTURE (ENGINEER'S VIEW)

Brake Mode is essentially a simplified PosHold mode with hard-coded targets.

1. **Input Lockout:**
   - Unlike almost every other mode, `ModeBrake::run()` does **not** read pilot sticks.
   - Throttle, Roll, Pitch, and Yaw are completely ignored.
   - *Code Path:* `ModeBrake::run()` calls `pos_control->input_vel_accel_xy` with zero vectors.
2. **Deceleration Logic:**
   - The controller sets a Target Velocity of `(0,0,0)` cm/s.
   - The `AC_PosControl` library uses its internal PID loops to drive the vehicle to this velocity using `BRAKE_MODE_DECEL_RATE` (Default: $750 cm/s^2$).
   - This is typically much more aggressive than a standard Loiter stop.
3. **Position Hold:**
   - Once velocity reaches zero, the underlying `PosControl` logic automatically latches the current position and fights drift.

TIMEOUT BEHAVIOR

Brake Mode has a unique feature: it can automatically exit itself.

- **Trigger:** If `timeout_to_loiter_ms()` is called (usually by GCS logic), a timer starts.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- **Action:** When the timer expires, the vehicle attempts to switch to **Loiter** mode.
- **Fallback:** If Loiter fails (e.g., GPS glitch), it falls back to **AltHold** to at least maintain altitude.
- *Code Path:* `timeout_to_loiter_ms`.

FAILSAFE LOGIC

- **GPS Loss:** If GPS is lost while braking, the vehicle cannot determine its velocity or position. It will likely trigger a standard EKF Failsafe (switching to Land or AltHold), overriding the Brake mode.

KEY PARAMETERS

PARAMETER	DEFAULT	DESCRIPTION
`BRAKE_MODE_DECEL_RATE`	750	(cm/s^2) The deceleration aggression. Higher values stop faster but may cause pitch-up oscillations.
`BRAKE_MODE_SPEED_Z`	250	(cm/s) Max vertical speed allowed during the brake maneuver.

SOURCE CODE REFERENCE

- **Mode Logic:** `ardupilot/ArduCopter/mode_brake.cpp`

PRACTICAL GUIDE: THE PANIC SWITCH

You lose video. You lose orientation. The drone is flying towards a tree. You need to stop NOW.

CONFIGURATION

1. **RC Setup:** Assign a momentary switch (like the "Trainer" switch) on your radio to a spare channel (e.g., Channel 8).
2. **ArduPilot:** Set `RC8_OPTION = 17` (Brake).
    - *Note:* Do not assign this to a flight mode slot. An RC Option overrides *any* flight mode.

USAGE

- **Press and Hold:** The drone slams on the brakes. It ignores all stick inputs.
- **Regain Composure:** Take a breath. Look at the drone. Orient yourself.
- **Release:** The drone stays in Loiter (or whatever mode you were in, usually).
- **Warning:** The braking is aggressive (0.75G). Ensure your battery is secure, or it might eject through the front.

CIRCLE MODE (COPTER)

EXECUTIVE SUMMARY

Circle Mode is an autonomous flight mode that makes the vehicle orbit a specific point of interest (POI). It allows for smooth, cinematic orbital shots with the nose of the aircraft (and camera) locked onto the center. It supports dynamic pilot adjustments to the radius and speed during flight.

THEORY & CONCEPTS

1. CENTRIPETAL FORCE IN FLIGHT

To fly in a circle, a drone must constantly accelerate towards the center of that circle.

- **The Physics:** The drone's total thrust vector must provide two components:
    1. **Vertical:** To stay in the air (counter gravity).

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY



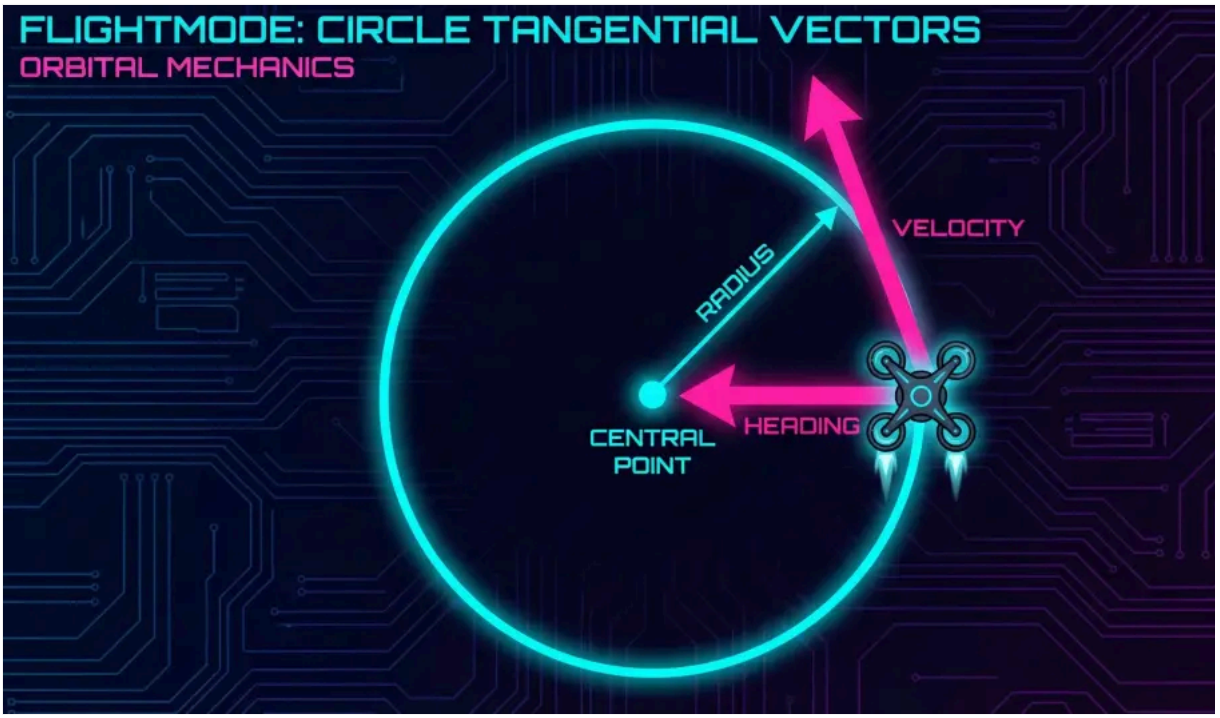


2. **Horizontal (Radial):** To provide the centripetal force needed for the turn.

- **The Result:** This is why a drone **Banks** into a turn. The steeper the bank, the more horizontal force is generated, allowing for a tighter or faster circle.

2. POI ORIENTATION

Unlike standard flight where the "forward" direction is where the nose points, in Circle mode, the "forward" direction is **Tangential** to the circle. ArduPilot automatically manages the Yaw so that the nose always points at the center, creating a "Satellite" perspective of the target.



HARDWARE DEPENDENCY MATRIX

Like Loiter, Circle relies on precise positioning to maintain its orbit.

SENSOR	REQUIREMENT	CODE IMPLEMENTATION NOTES
GPS	CRITICAL	Requires a valid position estimate to calculate the orbital path.
Compass	CRITICAL	Required for heading control (keeping the nose pointed at the center).
Barometer	REQUIRED	Used for altitude hold (Z-axis).

CONTROL ARCHITECTURE (ENGINEER'S VIEW)

Circle Mode delegates its navigation logic to the `AC_Circle` library.

1. **Center Point Calculation:**
   - By default, when you engage Circle mode, the "Center" is calculated as a point **one radius distance** in front of the vehicle. This means the drone will effectively fly "around" the object you were looking at.
   - *Option:* If `CIRCLE_OPTIONS` bit 2 is set (`INIT_AT_CENTER`), the current location *becomes* the center, and the drone flies out to the radius.
2. **Trajectory Generation:**
   - The `AC_Circle` library maintains an internal angular state (`_angle`).
   - It calculates a target position on the perimeter based on `_angle` and `_radius`.
   - It feeds a Velocity Target to the `PosControl` library to drive the vehicle along the tangent of the circle.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

3. **Yaw Control:**
    - The autopilot automatically calculates the bearing to the center point and locks the Yaw target to that bearing.
    - *Panorama Mode:* If Radius is 0, the vehicle stays in place and simply rotates (Panorama).

PILOT INTERACTION

If `CIRCLE_OPTIONS` bit 0 is enabled (Manual Control, default ON), the pilot can modify the orbit in real-time:

- **Pitch Stick:** Modifies the **Radius**.
    - Pitch Forward: Reduces radius (spirals in).
    - Pitch Back: Increases radius (spirals out).
- **Roll Stick:** Modifies the **Angular Rate** (Speed).
    - Roll Right: Increases clockwise speed.
    - Roll Left: Increases counter-clockwise speed (or slows down/reverses).
- **Throttle:** Standard Altitude Hold (Climb Rate control).

FAILSAFE LOGIC

- **GPS Loss:** The vehicle cannot maintain the orbit. It will likely trigger an EKF failsafe (Land or AltHold).
- **Crash Risk:** If the radius is reduced to zero while moving fast, the centripetal acceleration requirement drops, but the vehicle enters "Panorama" mode (spinning in place).

KEY PARAMETERS

PARAMETER	DEFAULT	DESCRIPTION
`CIRCLE_RADIUS`	1000	(cm) Default radius of the orbit (10m).
`CIRCLE_RATE`	20	(deg/s) Default angular speed. Positive = Clockwise, Negative = CCW.
`CIRCLE_OPTIONS`	1	Bitmask. Bit 0: Manual Control (Sticks adjust orbit). Bit 1: Face direction of travel. Bit 2: Init at Center.

SOURCE CODE REFERENCE

- **Mode Logic:** `ardupilot/ArduCopter/mode_circle.cpp`
- **Library Logic:** `AC_Circle::update()`

PRACTICAL GUIDE: DYNAMIC ORBITING

The beauty of Circle Mode is that it isn't static. You can "Fly" the orbit.

1. THE ENTRY

- Fly towards your subject (e.g., a tree).
- Stop about 10m away, facing the tree.
- Switch to **Circle Mode**.
- **Result:** The drone will calculate the center to be 10m directly in front of you (where the tree is) and start orbiting.

2. THE SPIRAL (ADVANCED SHOT)

You can create a cinematic "reveal" shot by spiraling out.

- While orbiting, **Pull Pitch Back** gently.
- **Effect:** The radius increases smoothly. The drone spirals away from the subject while keeping it in frame.

3. SPEED CONTROL

- **Push Roll Right:** Speed up (Orbital Velocity increases).
- **Push Roll Left:** Slow down, stop, or reverse direction.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- *Note:* Unlike Loiter, you aren't fighting the mode. You are adjusting the *parameters* of the mode in real-time.

DRIFT MODE (COPTER)

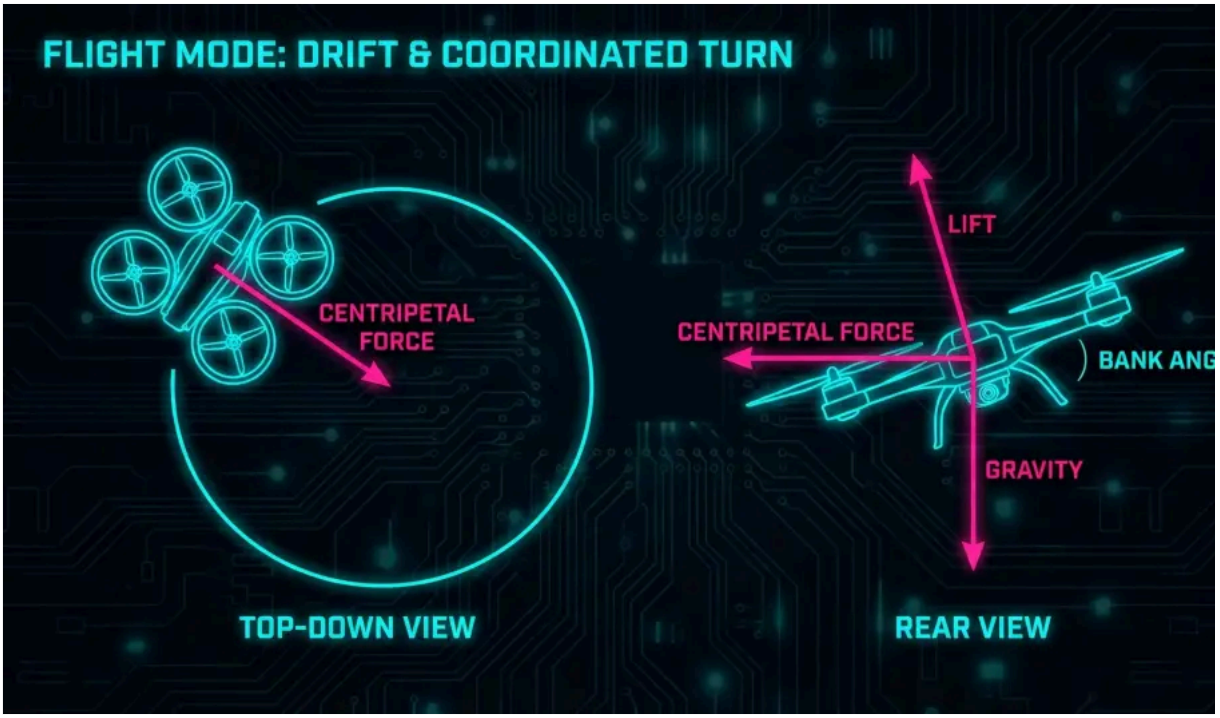
EXECUTIVE SUMMARY

Drift Mode completely reimagines the control scheme of a multicopter to mimic the behavior of a fixed-wing aircraft or a car. It provides "coordinated turns" where rolling the vehicle automatically induces a yaw rotation, allowing for smooth, banking turns without needing manual yaw input. It also features automatic braking when the sticks are released.

THEORY & CONCEPTS

1. COORDINATED VS. UNCOORDINATED FLIGHT

In traditional aviation, a "Coordinated" turn is one where the plane doesn't slip sideways through the air.



- **The Drone Problem:** Drones are inherently uncoordinated. They can fly sideways just as easily as forwards.
- **The Drift Mode Solution:** ArduPilot forces coordination by **Linking Yaw to Roll**. By automatically yawing the drone into the direction of the roll, it creates a bank-and-turn feel similar to a fixed-wing airplane.

2. RELATIVE MOMENTUM

Drift mode uses **Velocity Control** for its horizontal axes.

- **The Feel:** Because the Pitch stick controls speed, and the Roll stick controls turn rate, flying Drift mode feels like "Driving a car in 3D space." It removes the complexity of managing the drone's heading separately from its movement vector.

HARDWARE DEPENDENCY MATRIX

Drift Mode relies on velocity estimation to perform its coordinated turns and braking.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

SENSOR	REQUIREMENT	CODE IMPLEMENTATION NOTES
GPS	CRITICAL	Required for inertial velocity calculations. Without a position estimate, the braking and roll-assist logic cannot function.
Compass	CRITICAL	Required for heading and velocity frame transformation.
Barometer	REQUIRED	Used for altitude hold (Z-axis).

CONTROL ARCHITECTURE (ENGINEER'S VIEW)

Drift Mode uses a custom "remap" of the pilot inputs that exists nowhere else in the codebase.

1. **Stick Remapping:**
   - **Roll Stick (Right Stick X):** Commands **Yaw Rate** (Turn rate). It also commands a coordinated bank angle.
   - **Pitch Stick (Right Stick Y):** Commands **Pitch Angle** (Acceleration/Deceleration).
   - **Yaw Stick (Left Stick X):** Commands **Lateral Velocity** (Side-slip).
   - **Throttle Stick:** Standard Altitude Hold (Climb Rate).
2. **Coordinated Turn Logic:**
   - The controller calculates a target yaw rate based on the Roll Stick input.
   - *Equation:* `Yaw_Rate = Roll_Input * (1.0 - (Forward_Speed / 50m/s))`.
   - *Effect:* At low speeds, the yaw response is sharp. At high speeds, the yaw response is dampened to prevent spinning out, creating a smooth banking curve.
3. **Velocity Controller (Roll Axis):**
   - The roll axis runs a P-loop (`DRIFT_SPEEDGAIN`) to minimize lateral velocity.
   - This acts like the "keel" of a boat or the vertical stabilizer of a plane, fighting side-slip and forcing the drone to fly nose-first.

PILOT INTERACTION

- **Flying Forward:** Push Pitch stick forward.
- **Turning:** Push Roll stick Left/Right. The drone banks and turns. No rudder (Yaw stick) required.
- **Stopping:** Release the Pitch stick. The drone automatically pitches back to brake (using `DRIFT_SPEED` logic).

FAILSAFE LOGIC

- **GPS Loss:** The mode relies on `inertial_nav.get_velocity_neu_cms()`. If GPS fails, the velocity estimate becomes invalid. The behavior will likely degrade to a drift-prone Stabilize-like feel or trigger an EKF failsafe.

KEY PARAMETERS

PARAMETER	DEFAULT	DESCRIPTION
`DRIFT_SPEED`	500	Max speed (cm/s). Also used to calculate braking aggression.
`DRIFT_SPEEDGAIN`	14	Gain for the side-slip controller. Higher values make the "virtual keel" stronger, forcing nose-forward flight more aggressively.
`DRIFT_COORD`	0	(Boolean) If enabled, allows manual yaw input to override the coordinated turn logic.

SOURCE CODE REFERENCE

- **Mode Logic:** `ardupilot/ArduCopter/mode_drift.cpp`
- **Control Loop:** `ModeDrift::run()`

FLIP MODE (COPTER)

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





EXECUTIVE SUMMARY

Flip Mode is a pre-programmed aerobatic maneuver that makes the vehicle perform a 360-degree rotation on the Roll or Pitch axis. It is designed to be safe and reproducible, using a specific state machine to manage the rotation rate and throttle to minimize altitude loss.

THEORY & CONCEPTS

1. ROTATIONAL MOMENTUM

Performing a 360-degree flip requires overcoming the **Inertia** of the battery and frame.

- **The Rotation:** The drone must rotate at a rate high enough to complete the circle before it hits the ground.
- **The Math:** 400 deg/sec is chosen because it is fast enough to look "snappy" but slow enough that most standard motors have enough torque to stop the rotation at exactly 360 degrees.

2. THRUST MANAGEMENT IN INVERSION

Gravity is always pulling the drone DOWN.

1. **Stage 1:** Thrust is applied UP to gain momentum.
2. **Stage 2:** When inverted, the props point DOWN. If you don't cut the throttle, you will accelerate into the ground at $20m/s^2$.
3. **The Solution:** ArduPilot's flip logic performs an automated **Throttle Cut** during the inverted phase of the maneuver.

HARDWARE DEPENDENCY MATRIX

Like Acro, Flip mode relies on high-speed rate control.

SENSOR	REQUIREMENT	CODE IMPLEMENTATION NOTES
Gyroscope	CRITICAL	Required for the 400 deg/sec rate controller.
Accelerometer	CRITICAL	Required for the "Recovery" phase to return the vehicle to a level attitude after the flip completes.
GPS	**NONE**	Not used.
Barometer	**NONE**	Altitude control is open-loop (hardcoded throttle boosts) during the maneuver.

CONTROL ARCHITECTURE (ENGINEER'S VIEW)

Flip Mode is implemented as a rigid **State Machine** in `mode_flip.cpp`.

1. **Stage 1: Start (Nose Up)**
   - The vehicle rotates until it reaches 45 degrees.
   - **Throttle:** Boosted by `FLIP_THR_INC` (+20\%) to gain vertical momentum.
2. **Stage 2: Inversion (The Flip)**
   - The vehicle commands a hard rotation rate (`FLIP_ROTATION_RATE` = 400 deg/s).
   - **Throttle:** Reduced by `FLIP_THR_DEC` (-24\%) to prevent accelerating into the ground while inverted.
3. **Stage 3: Recovery (Leveling)**
   - Once the vehicle passes the inverted point and approaches level (< 45 deg error), it switches to the `Recover` state.
   - It uses the Earth-Frame Angle Controller (Stabilize) to lock level.
   - **Throttle:** Boosted again (`FLIP_THR_INC`) to catch the fall.
4. **Stage 4: Complete**
   - The mode exits and returns to the previous flight mode (usually Stabilize or AltHold) or stays in Stabilize.

PILOT INTERACTION

- **Trigger:** The mode is engaged via a switch.
- **Direction Selection:** Once engaged, the pilot moves the Roll or Pitch stick to "command" the direction of the flip.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- **Abort:** If the pilot moves the sticks significantly during the flip (> 40 degrees equivalent), the mode aborts and returns manual control to prevent fighting the pilot.

FAILSAFE LOGIC

- **Timeout:** If the flip takes longer than 2.5 seconds (`FLIP_TIMEOUT_MS`), the mode abandons the maneuver and switches to a Stabilize-like recovery to prevent an infinite tumbling state.

KEY PARAMETERS

There are **NO** user-configurable parameters for Flip Mode in the standard parameter list. The behavior is hardcoded in the source.

CONSTANT (HARDCODED)	VALUE	DESCRIPTION
`FLIP_ROTATION_RATE`	40000	(centi-deg/s) 400 degrees per second rotation target.
`FLIP_THR_INC`	0.20	20\% throttle boost during start/recovery.
`FLIP_THR_DEC`	0.24	24\% throttle cut during inversion.

SOURCE CODE REFERENCE

- **Mode Logic:** `ardupilot/ArduCopter/mode_flip.cpp`
- **State Machine:** `ModeFlip::run()`

FLOWHOLD MODE (COPTER)

EXECUTIVE SUMMARY

FlowHold is a horizontal position-control mode designed for GPS-denied environments. It utilizes an Optical Flow sensor to arrest drift and maintain position. Unlike Loiter, it does not target a specific geographic coordinate but rather continuously minimizes horizontal velocity error derived from visual data.

THEORY & CONCEPTS

1. WHAT IS OPTICAL FLOW?

For those new to ArduPilot, an Optical Flow sensor is essentially a downward-facing camera that tracks the texture of the ground. By analyzing how pixels move from frame to frame, it calculates the "flow rate."

- **Analogy:** Imagine looking down at the pavement through a straw while walking. The speed at which the cracks in the pavement pass through your view is "Optical Flow."

2. OPTICAL FLOW VS. INERTIAL PREDICTION

FlowHold is a **Velocity Estimator**.

- **The Problem:** IMU sensors (Accelerometers) are very noisy and drift within seconds.
- **The Flow Advantage:** Optical flow sensors provide a high-rate (50Hz+) measurement of movement over the ground.
- **The Fusion:** The EKF uses the flow data to constantly correct the "Drift" of the accelerometers. This allows the drone to remain stationary even when the GPS signal is blocked by a roof.

3. THE "FAKE MOTION" PROBLEM (GYROSCOPIC COMPENSATION)

A raw optical flow sensor cannot distinguish between two very different movements:

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





1. **Translation:** The drone moves forward. The ground appears to slide backward.
2. **Rotation:** The drone pitches forward (noses down) while hovering in place. The camera tilts, so the ground *also* appears to slide backward.

If the drone didn't know the difference, pitching forward to accelerate would make it think it was *already* moving fast, causing it to fight its own inputs.

HOW ARDUPILOT SOLVES IT

ArduPilot performs **Gyroscopic Compensation**. It subtracts the vehicle's angular rotation (measured by the Gyro) from the visual flow rate.

- **The Math:** `True_Flow = Sensor_Flow - Body_Rotation_Rate`
- **The Code:** Validated in `mode_flowhold.cpp`:

```
Vector2f raw_flow = copter.optflow.flowRate() - copter.optflow.bodyRate();
```

- **Why this matters for you:** If your `FLOW_FXSCALER` or `FLOW_FYSCALER` parameters are incorrect, the units won't match. Pitching the drone will create "ghost" velocity readings, causing instability or "toilet bowling" even if the PID gains are perfect.

HARDWARE DEPENDENCY MATRIX

The code explicitly checks for sensor health before entering or maintaining this mode.

SENSOR	REQUIREMENT	CODE IMPLEMENTATION NOTES
Optical Flow	CRITICAL	`copter.optflow.enabled()` and `healthy()` must be true. Failsafe triggers if quality drops below `FHLD_QUAL_MIN`.
GPS	**NONE**	`requires_GPS()` returns `false`. GPS is ignored for navigation.
Compass	OPTIONAL	Heading is maintained via the AHRS, but FlowHold itself operates in the body-frame relative to flow.
Barometer	REQUIRED	Primary source for Z-axis altitude hold unless a Rangefinder is active.
Rangefinder	**RECOMMENDED**	If absent, the vehicle uses a "Virtual Odometry" estimate which correlates IMU accel with Flow rate to guess height. This is the primary cause of instability. See `update_height_estimate()`.

CONTROL ARCHITECTURE (ENGINEER'S VIEW)

FlowHold operates on a unique control structure distinct from Loiter or PosHold.

1. **Input Shaper:** Pilot Roll/Pitch sticks are mapped directly to **Lean Angle** (not Velocity).
   - *Code Path:* `get_pilot_desired_lean_angles` called in `run()`.
2. **Flow-to-Angle Controller:**
   - The raw flow rate is filtered (`FHLD_FILT_HZ`) and constrained (`FHLD_FLOW_MAX`).
   - It is fed into a **PI Controller** (`FHLD_XY_P`, `FHLD_XY_I`).
   - **Output:** A correction lean angle that is added to the pilot's input.
   - *Code Path:* `flowhold_flow_to_angle()`.
3. **Z-Axis Controller:**
   - Uses standard `PosControl` Z-axis logic.
   - **Critical Detail:** It calculates altitude error based on the EKF origin. If relying on Barometer (indoor), pressure fluctuations will cause the Z-controller to command throttle changes, leading to "ceiling suction" or floor drops.

PILOT INTERACTION

- **Active Input:** When you move the sticks, you are in **Stabilize-like** control. You are fighting the flow controller's I-term if you push against the drift.

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- **Stick Release (Braking):** When sticks are centered, the `braking` flag is set.
  - *Logic:* The controller applies a "Brake Gain" derived from `FHLD_BRAKE_RATE`.
  - *Feel:* The vehicle pitches back aggressively to kill momentum, then locks position.

FAILSAFE LOGIC

The `mode_flowhold.cpp` `run()` function contains specific checks:

1. **Low Flow Quality:**
   - *Trigger:* Sensor quality drops below `FHLD_QUAL_MIN` (Default: 10).
   - *Behavior:* The flow correction term is zeroed out (see `run()` loop).
   - *Result:* The vehicle silently degrades to **AltHold**. It will drift with the wind. It does *not* land or switch modes automatically.
2. **Height Estimation Divergence:**
   - If no rangefinder is present, and the vehicle is static (no velocity change), the height estimator cannot update.
   - *Result:* The flow scaler may be incorrect, leading to oscillation (toilet bowling) or sluggishness.

KEY PARAMETERS

Parameter definitions start at line 11.

PARAMETER	DEFAULT	DESCRIPTION
`FHLD_QUAL_MIN`	10	Minimum flow quality (0-255). Below this, the vehicle stops holding position and acts like AltHold.
`FHLD_FLOW_MAX`	0.6	The maximum flow rate (approx m/s) the controller will attempt to correct. Lower values make it less aggressive.
`FHLD_XY_P`	3.0	Proportional gain for the position hold. Increase for tighter hold, decrease if oscillating.
`FHLD_XY_I`	0.05	Integral gain. Helps correct for steady-state drift (e.g., wind).
`FHLD_BRAKE_RATE`	8	(deg/s). How aggressively the vehicle pitches back to stop when sticks are released.
`FHLD_FILT_HZ`	5	Input filter for the flow sensor. Lower values smooth out noisy sensors but add latency.
`FLOW_FXSCALER`	0	**Critical for Theory:** Calibrates the sensor's X-axis field of view to match the gyro. If this is wrong, gyroscopic compensation fails.
`FLOW_FYSCALER`	0	**Critical for Theory:** Calibrates the sensor's Y-axis field of view to match the gyro.

TUNING & TROUBLESHOOTING

SYMPTOM	PROBABLE CAUSE	CORRECTIVE ACTION
"Toilet Bowling" (Swirling)	Compass interference OR poor height estimate.	If no Rangefinder: perform aggressive maneuvers to help the EKF learn height scale. If Rangefinder: Check alignment.
Drifting with Wind	I-Term saturation or too low.	Increase `FHLD_XY_I` to allow the controller to lean against steady air currents.
Oscillation (Jitter)	P-Gain too high for the surface texture.	Reduce `FHLD_XY_P`.
Surging when Pitching	Incorrect Gyro Compensation.	Tune `FLOW_FXSCALER` / `FLOW_FYSCALER`. If the drone accelerates when you just tilt it, the sensor thinks rotation is translation.
Ceiling Strike	Barometer drift in indoor environment.	Cover the barometer with open-cell foam. Use a Rangefinder (Lidar/Sonar) to override baro.

SOURCE CODE REFERENCE

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- **Mode Logic:** `ardupilot/ArduCopter/mode_flowhold.cpp`
- **Height Estimator:** `ModeFlowHold::update_height_estimate()`
- **State Machine:** `ModeFlowHold::run()`

FOLLOW MODE (COPTER)

EXECUTIVE SUMMARY

Follow Mode allows the copter to autonomously track and follow another MAVLink-enabled vehicle (Lead Vehicle) or a Ground Control Station (GCS). It maintains a specific relative offset (distance, altitude, angle) and matches the lead vehicle's velocity vector.

THEORY & CONCEPTS

1. RELATIVE REFERENCE FRAMES

In robotics, there are two ways to describe position:

1. **Earth-Fixed (Inertial):** Latitude and Longitude. (e.g., "Go to North 10m").
2. **Body-Fixed (Local):** Relative to the drone. (e.g., "Move 10m to my right").

- **The Follow Problem:** To follow a moving person, the drone must constantly calculate its position in the **Target's Body Frame**. If the person turns, the drone must rotate around them to maintain the same "Over the shoulder" angle.

2. TIME SYNCHRONIZATION (LATENCY)

Following is a "Chase" problem.

- **The Latency:** The leader sends its position over radio. By the time the follower receives it, the leader has already moved.
- **The Correction:** ArduPilot uses **Velocity Extrapolation**. It takes the leader's *Speed* and *Heading* to predict where they will be 200ms in the future, aiming the drone at the future point rather than the stale past point.

HARDWARE DEPENDENCY MATRIX

Both the follower (your drone) and the leader must have precise positioning.

SENSOR	REQUIREMENT	CODE IMPLEMENTATION NOTES
GPS	CRITICAL	Required on BOTH vehicles. The follower needs its own position to calculate the vector to the leader's reported position.
Compass	CRITICAL	Required for heading control and offset calculations.
Telemetry	CRITICAL	A MAVLink data link (SiK Radio, Wifi, ELRS bi-directional) is required to receive `GLOBAL_POSITION_INT` messages from the lead vehicle.

CONTROL ARCHITECTURE (ENGINEER'S VIEW)

Follow Mode is effectively a dynamic wrapper around **Guided Mode**.

1. **Target Estimation (`AP_Follow`):**
   - The vehicle listens for MAVLink messages (ID #33 `GLOBAL_POSITION_INT` or #144 `FOLLOW_TARGET`).
   - It filters this data to estimate the lead vehicle's Position and Velocity.
   - It applies the user-defined **Offsets** (`FOLL_OFS_X/Y/Z`) to calculate a "Virtual Target Point" where the follower *should* be.
2. **Velocity Controller:**
   - Instead of just flying to the waypoint (which would cause lag), the controller uses **Feed-Forward Velocity**.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

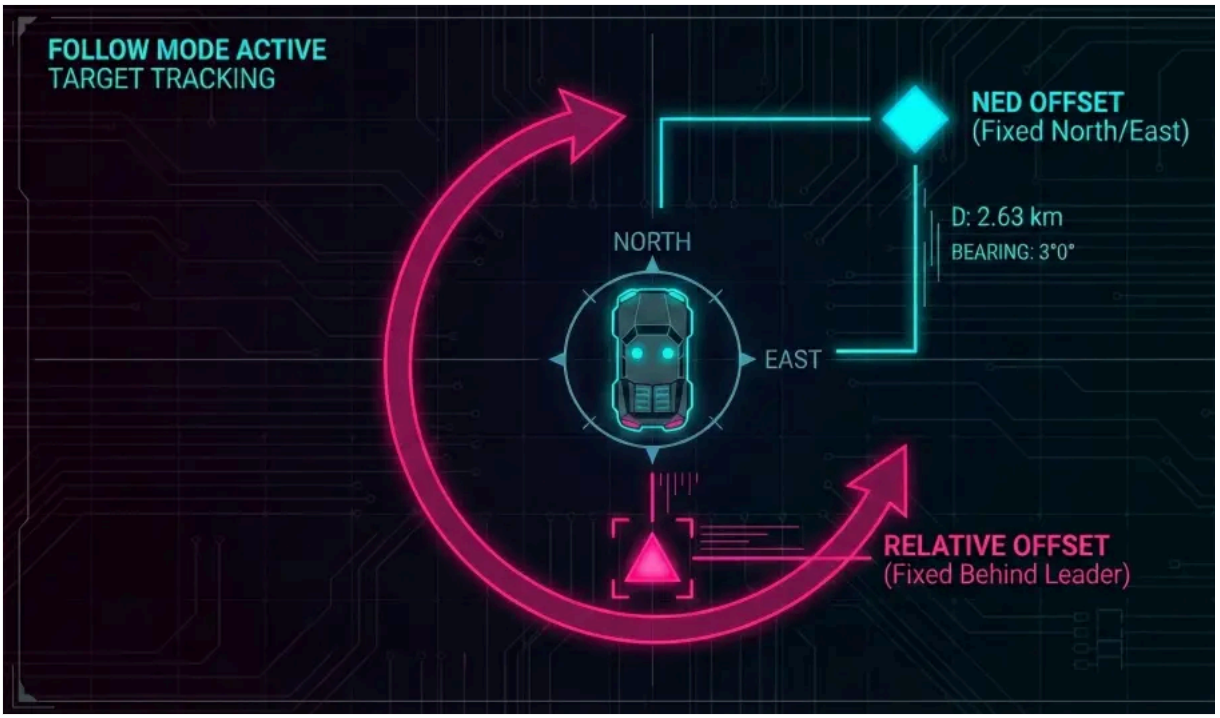




- `Target_Velocity = Lead_Velocity + Position_Error_Correction`.
- This allows the drone to match speed perfectly without lagging behind.
- The computed velocity is passed to the `ModeGuided` velocity controller logic.

OFFSETS & REFERENCE FRAMES

Understanding the Reference Frame (`FOLL_OFS_TYPE`) is critical for predictable behavior.



1. **NED Frame (Type 0):** Offsets are Earth-fixed (North/East).
   - *Example:* If Offset X = 10, the drone stays 10 meters North of the leader, regardless of which way the leader is flying.
2. **Relative Frame (Type 1):** Offsets are Body-fixed to the Leader.
   - *Example:* If Offset X = -10, the drone stays 10 meters *behind* the leader. If the leader turns 90 degrees, the drone orbits to stay behind it.

FAILSAFE LOGIC

- **Target Loss:** If the MAVLink stream is interrupted for more than 3 seconds (`AP_FOLLOW_TIMEOUT_MS`), the vehicle stops tracking.
  - *Behavior:* It sets the desired velocity to zero and effectively enters a **PosHold** state until the signal returns.
- **GPS Loss:** Standard EKF failsafes apply (Land or AltHold).

KEY PARAMETERS

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

PARAMETER	DEFAULT	DESCRIPTION
`FOLL_SYSID`	0	The MAVLink System ID of the lead vehicle to track. 0 = track the first vehicle seen.
`FOLL_OFS_X`	0	(meters) Distance offset (Forward/North).
`FOLL_OFS_Y`	0	(meters) Distance offset (Right/East).
`FOLL_OFS_Z`	0	(meters) Altitude offset (Down). Negative values = higher than leader.
`FOLL_OFS_TYPE`	1	0=NED (North/East), 1=Relative (Forward/Right).
`FOLL_POS_P`	0.1	Gain for position error correction. Higher values close the gap faster but may overshoot.

SOURCE CODE REFERENCE

- **Mode Logic:** `ardupilot/ArduCopter/mode_follow.cpp`
- **Message Handler:** `AP_Follow::handle_msg()`

GUIDED MODE (COPTER)

EXECUTIVE SUMMARY

Guided Mode is the primary interface for "Offboard Control." While Auto mode runs a pre-uploaded mission, Guided mode listens for real-time commands from a Ground Control Station (GCS) or Companion Computer. It can fly to a specific point, hold a velocity vector, or accept direct attitude targets, making it the mode of choice for swarm operations and dynamic path planning.

THEORY & CONCEPTS

1. OFFBOARD CONTROL HIERARCHY

In advanced drone systems, "Guided Mode" is the portal between the **Low-Level Autopilot** (ArduPilot) and the **High-Level Brain** (ROS, Python script, or GCS).

- **The Interface:** The High-Level brain makes decisions (e.g., "There is a person, fly to them"). It sends a high-level command to ArduPilot.
- **The Execution:** ArduPilot handles the difficult part: the PIDs, the Motor Mixer, and the EKF stabilization required to reach that point.

2. SLEW RATES

Real physical objects cannot change speed instantly.

- **The Concept:** If a companion computer asks for 20m/s instantly, the drone would tilt violently and crash.
- **The Solution:** `GUIDED_SLEW_SPD`. ArduPilot applies a low-pass filter to the incoming external commands, ensuring the drone accelerates at a safe, physically attainable rate.

HARDWARE DEPENDENCY MATRIX

Guided mode requires robust positioning to execute external commands.

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

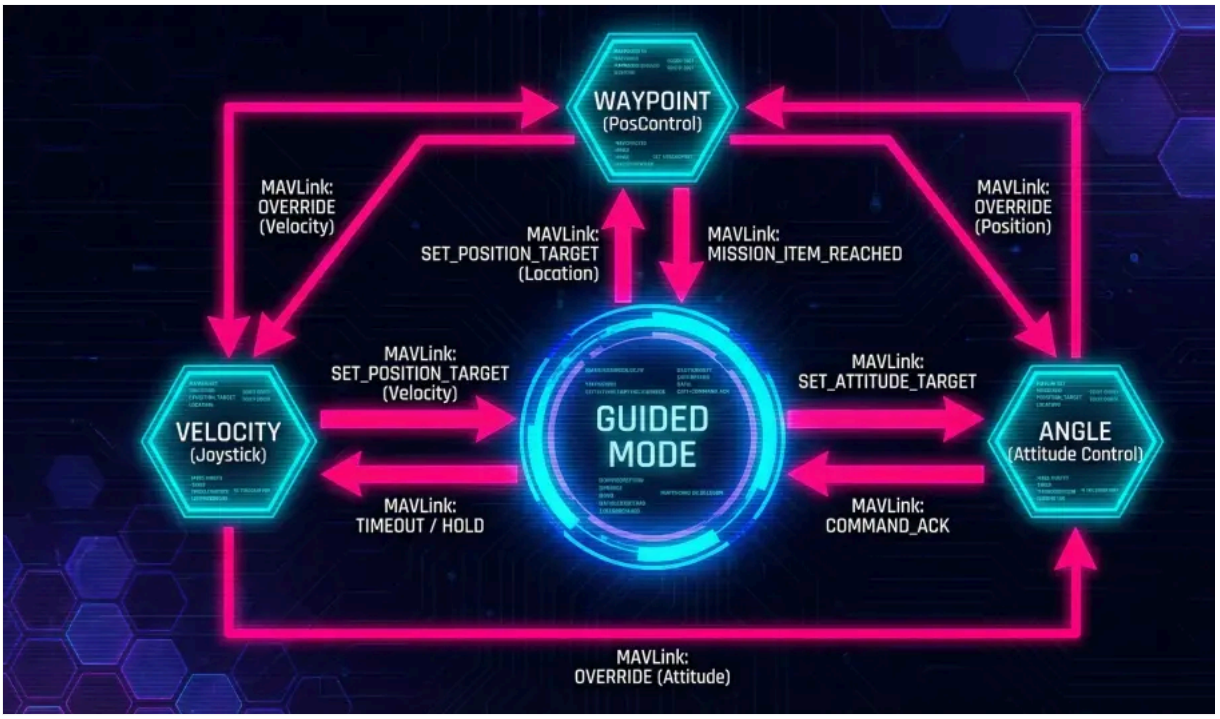




SENSOR	REQUIREMENT	CODE IMPLEMENTATION NOTES
GPS	CRITICAL	Required for all Position and Velocity sub-modes. Only the `Angle` sub-mode *might* run without it (if `GUIDED_NOGPS` is used, but standard Guided requires it).
Compass	CRITICAL	Required for heading control.
Telemetry	CRITICAL	A bi-directional link (MAVLink) is required to send commands.

CONTROL ARCHITECTURE (ENGINEER'S VIEW)

Guided Mode is a "Meta-Mode" with its own internal state machine, switching logic based on the *type* of MAVLink message received.



1. **Waypoint (WP) State:**
   - *Trigger:* `SET_POSITION_TARGET_GLOBAL_INT` (with Position only) or "Fly To" clicks on a map.
   - *Behavior:* Uses `AC_WPNav` (same as Auto) to generate a smooth path to the target.
2. **Velocity/Accel State:**
   - *Trigger:* `SET_POSITION_TARGET_LOCAL_NED` (with Velocity bitmask).
   - *Behavior:* Passes velocity vectors directly to the Position Controller. Used for "Joystick" control or Companion Computer path following.
3. **Angle (Attitude) State:**
   - *Trigger:* `SET_ATTITUDE_TARGET`.
   - *Behavior:* Bypasses the position controller and feeds lean angles directly to the Attitude Controller. Used for aggressive maneuvering or non-GPS fallback.

SAFETY & TIMEOUTS

Because Guided mode relies on an external stream of commands, it has a built-in "Deadman Switch."

- **Logic:** If the vehicle is in a high-frequency control state (Velocity, Accel, or Angle) and stops receiving MAVLink messages for `GUID_TIMEOUT` seconds (default 2.4s), it triggers a safety reaction.
- **Reaction:**
  - *Velocity/Accel:* The target velocity is set to zero (stops and holds position).

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

    - *Angle:* The vehicle attempts to level out.
- **Note:** The "Waypoint" state does *not* timeout; once sent, the drone will fly to the destination even if the link is lost.

KEY PARAMETERS

PARAMETER	DEFAULT	DESCRIPTION
`GUID_TIMEOUT`	2.4	(seconds) The timeout for streaming commands (Velocity/Attitude). Does not apply to "Go To" waypoints.
`GUID_OPTIONS`	0	Bitmask. Bit 2: Ignore Pilot Yaw. Bit 6: Use "Thrust" field as direct throttle instead of climb rate.
`WPNAV_SPEED`	500	Max speed for "Go To" commands.

SOURCE CODE REFERENCE

- **Mode Logic:** `ardupilot/ArduCopter/mode_guided.cpp`
- **State Dispatcher:** `ModeGuided::run()`
- **Command Interface:** `ModeGuided::set_destination()`

PRACTICAL GUIDE: COMPANION COMPUTER SAFETY

When writing code (Python/ROS) to drive Guided Mode, you are the pilot. You must implement your own safety checks.

1. THE HEARTBEAT (DEADMAN SWITCH)

If your script crashes, the drone will keep executing the last velocity command forever (until `GUID_TIMEOUT` kicks in).

- **Configure:** Set `GUID_TIMEOUT = 0.5` (500ms).
- **Logic:** Your script MUST send a command at least every 200ms (5Hz).
- **Result:** If your script freezes, the drone stops in 0.5 seconds. The default 2.4s is too long for indoor flight.

2. VELOCITY SMOOTHING

Your vision system might be noisy, sending "jittery" velocity targets.

- **Parameter:** `GUID_SLEW_SPD` (note: verify name in full param list, typically related to WPNAV or POS control filtering).
- **Alternative:** Use `WPNAV_ACCEL` to limit how fast the velocity controller can ramp up.
- **Best Practice:** Filter your velocity commands *before* sending them to ArduPilot. A simple low-pass filter in Python makes the drone fly 10x smoother.

HELI AUTOROTATE MODE (COPTER)

EXECUTIVE SUMMARY

This mode is **exclusive to Traditional Helicopters**. It automates the complex "Autorotation" maneuver required to land a helicopter safely after a motor failure. Instead of crashing, the helicopter uses the energy stored in the rotor blades and the airflow during descent (gliding) to maintain rotor RPM, flaring at the bottom to cushion the landing.

THEORY & CONCEPTS

1. THE PHYSICS OF AUTOROTATION

Autorotation is the process of extracting energy from the air to keep the rotor spinning.

- **The Inflow:** As the helicopter falls, air rushes UP through the rotor blades.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- **The Angle of Attack:** By reducing the collective pitch (blade angle), the upward air flow actually "pushes" the blades to spin faster (like a windmill).
- **The Flare:** Just before the ground, the pilot increases pitch. This converts the spinning rotor energy (Inertia) into Lift, stopping the descent for a soft landing.

2. ENERGY MANAGEMENT

Autorotation is an energy-management game.

- **Potential Energy (Altitude):** Your fuel.
- **Kinetic Energy (Airspeed):** Helps keep the rotor spinning.
- **Rotational Inertia (RPM):** Your "Battery" for the final landing.
- The autopilot's job is to trade Altitude for RPM and Airspeed, ensuring the RPM stays within the "Green Zone" until the very last second.

HARDWARE DEPENDENCY MATRIX

Requires specific setup for helicopters.

SENSOR	REQUIREMENT	CODE IMPLEMENTATION NOTES
RPM Sensor	CRITICAL	The controller needs precise Rotor Speed (Head Speed) feedback to manage the collective pitch during glide.
Airspeed	**RECOMMENDED**	Improves the glide efficiency significantly.
GPS	**RECOMMENDED**	Required for position-controlled glide and flare. Without it, the mode relies on inertial estimates which drift.
Rangefinder	CRITICAL	Essential for determining the exact moment to trigger the **Flare** and **Touchdown** phases.

CONTROL ARCHITECTURE (ENGINEER'S VIEW)

This mode implements a multi-stage State Machine in `mode_autorotate.cpp`.

1. **Entry Phase:**
   - Triggered when the motor interlock is disengaged (motor failure simulation) or Mode Switch is used.
   - The collective pitch is reduced immediately to preserve rotor RPM (`AROT_COL_ENTRY`).
2. **Steady-State Glide (SS_GLIDE):**
   - The controller manages two main variables:
     - **Head Speed:** Controlled via Collective Pitch. (Low RPM → Lower Collective to speed up).
     - **Forward Speed:** Controlled via Pitch Attitude.
   - Goal: Maintain efficient airspeed for maximum range or minimum descent rate.
3. **Flare Phase:**
   - Triggered by Altitude (Rangefinder).
   - The vehicle pitches up aggressively to convert forward airspeed into lift and rotor energy.
4. **Touchdown:**
   - The remaining rotor energy is used to cushion the final meter of descent.

FAILSAFE & BAILOUT

- **Bailout:** If the pilot re-engages the motor interlock (decides to abort the landing), the `Bailout` logic immediately ramps the motor back up to flight idle speed to recover powered flight.
- **Bad RPM:** If the rotor RPM drops too low during glide, the controller sacrifices altitude (lowers collective) to recover RPM, prioritizing control authority over flight time.

KEY PARAMETERS

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





PARAMETER	DEFAULT	DESCRIPTION
`AROT_ENABLE`	0	Master switch for the feature.
`AROT_COL_ENTRY`	-2	Collective pitch (degrees) used during entry phase.
`AROT_HS_TARG`	1500	Target Head Speed (RPM) during glide.
`AROT_FLARE_TIME`	3.0	Duration (seconds) of the flare maneuver.

SOURCE CODE REFERENCE

- **Mode Logic:** `ardupilot/ArduCopter/mode_autorotate.cpp`
- **State Machine:** `ModeAutorotate::run()`

LAND MODE (COPTER)

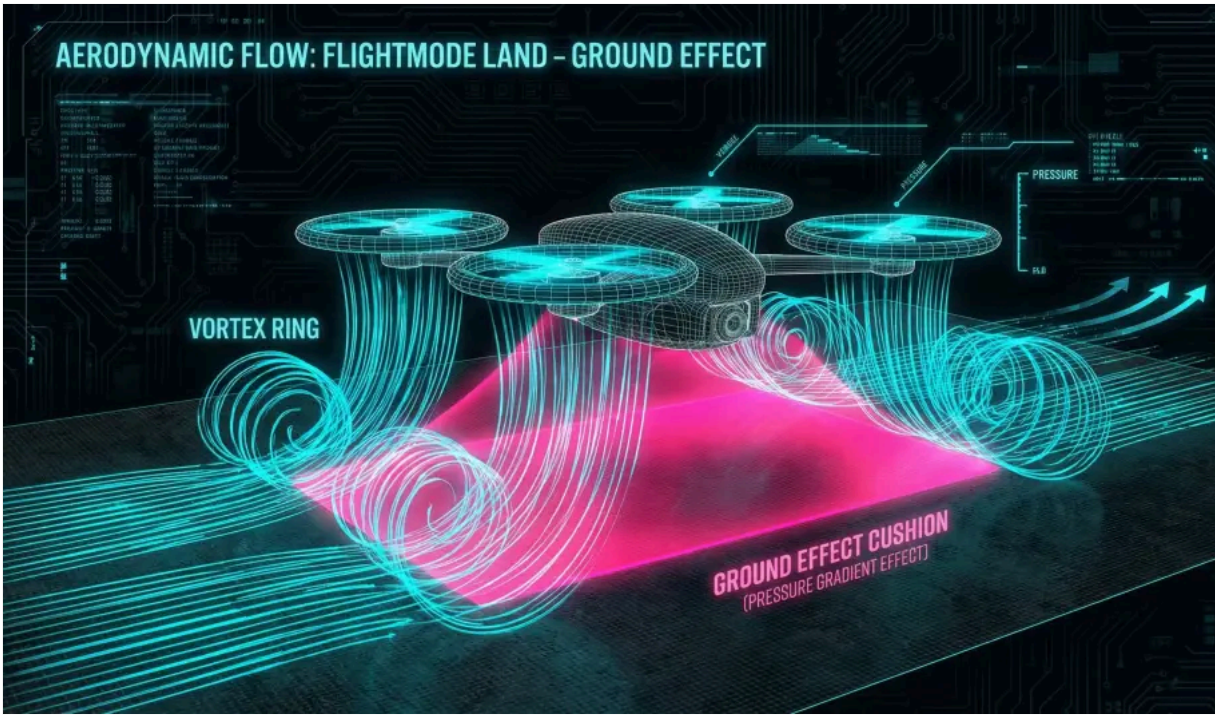
EXECUTIVE SUMMARY

Land Mode initiates a vertical descent to the ground. Its primary goal is to bring the vehicle down safely, detecting the ground to shut off motors automatically. It adapts its behavior based on sensor availability, offering a Position-Controlled descent if GPS is healthy, or a simple Level-Descent if not.

THEORY & CONCEPTS

1. GROUND EFFECT

When a drone gets close to the ground (within one rotor diameter), its own downwash creates a "cushion" of high-pressure air.



- **The Problem:** This cushion can make the drone "float" or bounce, confusing the Land Detector.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

- **The Solution:** The Land Mode uses a specific **Descent Rate Controller** that ignores the floatiness and forces the drone down at a constant `LAND_SPEED`.

2. BAROMETRIC GROUND SUCTION

A fast-descending drone creates a low-pressure zone above its propellers. If the barometer is mounted poorly, it might see this as an "Altitude Increase," causing the drone to speed up its descent. ArduPilot's EKF uses accelerometer data to reject these barometric "Lies" during the final landing phase.

HARDWARE DEPENDENCY MATRIX

Land mode is designed to work with whatever sensors are available, degrading gracefully.

SENSOR	REQUIREMENT	CODE IMPLEMENTATION NOTES
Barometer	REQUIRED	Primary source for altitude and descent rate control.
GPS	OPTIONAL	If available, the descent is position-locked (stops wind drift). If lost, the vehicle drifts with the wind while descending.
Rangefinder	RECOMMENDED	Drastically improves the "Slow Down" logic. Without it, the vehicle assumes Barometer altitude is accurate, which can be risky near the ground.
Compass	OPTIONAL	Required only for the GPS-assisted (Loiter-style) descent.

CONTROL ARCHITECTURE (ENGINEER'S VIEW)

Land Mode logic splits into two distinct paths in `ModeLand::run()`.

1. **GPS Run (`gps_run`):**
   - Acts like **Loiter Mode** but with a forced descent.
   - The horizontal controller actively fights wind to stay over the landing point.
2. **No-GPS Run (`nogps_run`):**
   - Acts like **Stabilize Mode** (self-leveling) but with a forced descent.
   - The vehicle *will* drift with the wind. The pilot must manually correct pitch/roll to avoid hitting obstacles.

VERTICAL DESCENT PROFILE

The descent is not a single speed; it is a multi-stage profile managed by `land_run_vertical_control`.

1. **Fast Descent:** The vehicle descends at `WPNAV_SPEED_DN` (default 150 cm/s).
2. **Slow Down:** When the vehicle thinks it is 10 meters (`LAND_ALT_LOW`) above the ground (or home), it slows to `LAND_SPEED` (default 50 cm/s).
   - *Critical Note:* If you take off from a hill and fly over a valley, the Barometer thinks you are high up. The drone may descend fast until it smashes into the ground because it didn't know the ground came up. **Rangefinders solve this.**

PILOT INTERACTION

- **Repositioning:** If `LAND_REPOSITION` is enabled (default), the pilot can use the Pitch/Roll sticks to move the drone horizontally during the descent.
  - In GPS mode, this feels like "nudging" the landing point.
  - In No-GPS mode, this feels like flying in Stabilize.
- **Throttle:** Pilot throttle input is ignored (unless repositioning logic overrides it in specific setups), as the Z-controller is fully automated.

FAILSAFE LOGIC

- **Landing Detector:** The motors will **not** disarm until the Landing Detector logic confirms touchdown. It looks for:
  1. Vertical Velocity near zero.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





2. Throttle command at minimum.
3. Rotation rates near zero.

- If the drone is bouncing (AirMode issue) or "floating" (Ground Effect), it may refuse to disarm.

KEY PARAMETERS

PARAMETER	DEFAULT	DESCRIPTION
`LAND_SPEED`	50	(cm/s) Final descent speed. Lower is safer for landing gear.
`LAND_ALT_LOW`	1000	(cm) Altitude at which to switch to `LAND_SPEED`.
`LAND_REPOSITION`	1	(0/1) Enable pilot stick input to move horizontally during landing.
`WPNAV_SPEED_DN`	150	(cm/s) Initial fast descent rate.

SOURCE CODE REFERENCE

- **Mode Logic:** `ardupilot/ArduCopter/mode_land.cpp`
- **GPS Path:** `ModeLand::gps_run()`

PRACTICAL GUIDE: TUNING THE LAND DETECTOR

Does your drone "bounce" on the ground and refuse to disarm? Or disarm in mid-air?

THE BOUNCE (REFUSAL TO DISARM)

- **Cause:** The drone touches down, but the "Ground Effect" turbulence makes the gyro noisy. The Land Detector thinks "I'm still moving!" and keeps the motors running.
- **Fix 1:** Lower `LAND_SPEED`. If you hit the ground gently (30 cm/s), there is less shock.
- **Fix 2:** Disable AirMode (`RCx_OPTION = 84`). AirMode is the #1 cause of landing bounces.

THE FALSE LANDING (MID-AIR DISARM)

- **Cause:** You are descending fast in a lightweight drone. The throttle is at minimum. The wind catches the drone, momentarily stopping its descent. The detector thinks "Zero throttle + Zero Velocity = Ground".
- **Fix:** Ensure `DISARM_DELAY` is > 0.
- **Fix:** Avoid zero-throttle dives in **Stabilize** mode if you have a sensitive land detector. Use **Acro** (which ignores land detection) for diving.

LOITER MODE (COPTER)

EXECUTIVE SUMMARY

Loiter Mode is the default GPS-assisted flight mode for ArduCopter. It automatically holds position, altitude, and heading when the sticks are centered. When the pilot moves the sticks, the vehicle accelerates up to a maximum speed. It is characterized by its "smooth" and "heavy" feel, as it resists rapid changes in direction to provide stable video.

THEORY & CONCEPTS

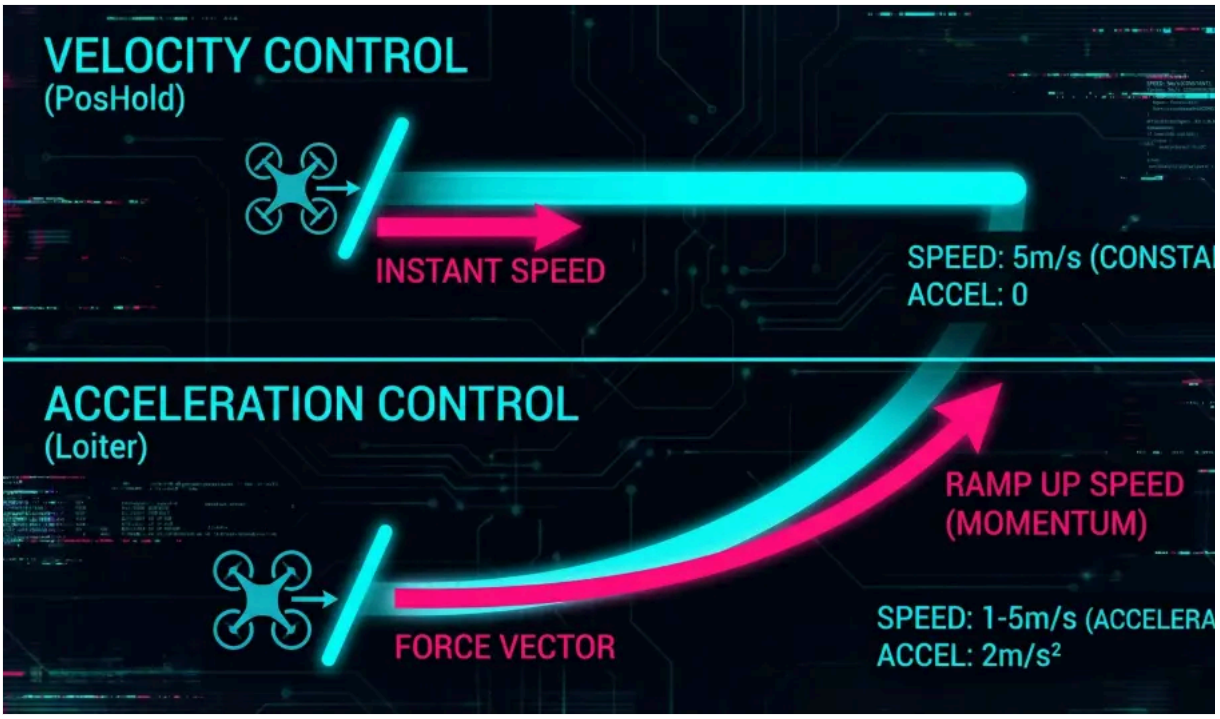
1. VELOCITY VS. POSITION CONTROL

In robotics, there are two ways to hold position:

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY







- **Position Lock:** "Stay at this GPS coordinate." (Stiff but can jitter).
- **Velocity Lock:** "Maintain 0 m/s speed." (Smooth but can drift).
- **The Loiter Hybrid:** ArduPilot Loiter is primarily a **Velocity Controller**. When you release the sticks, it calculates the braking velocity. Once stopped, it captures the current position as a "soft anchor" to prevent drift.

2. THE DRIFT FEEL

Loiter is designed for **Cinematic Stabilty**.

- **The Math:** By using an **Acceleration-based** input model, the drone feels heavy. It doesn't snap to a stop; it decelerates at a natural, smooth rate. This prevents the "Camera Shake" that occurs when a high-torque drone stops instantly.

HARDWARE DEPENDENCY MATRIX

Loiter is critically dependent on a high-quality position estimate.

SENSOR	REQUIREMENT	CODE IMPLEMENTATION NOTES
GPS	CRITICAL	Requires a 3D Lock and low HDOP. EKF variance issues will trigger a failsafe.
Compass	CRITICAL	Heading is required to translate "Forward" stick into "North/East" velocity. Compass interference causes "Toilet Bowling".
Barometer	REQUIRED	Used for altitude hold (Z-axis).

CONTROL ARCHITECTURE (ENGINEER'S VIEW)

Loiter uses the `AC_Loiter` library, which implements an **Acceleration Controller**.

1. **Stick to Acceleration:**
   - Pilot Stick Deflection is mapped to a target **Acceleration** (Lean Angle).
   - This differs from **PosHold**, which maps sticks to **Velocity** (Speed).
   - *Result:* The drone feels like it has "momentum." You push the stick, it leans and speeds up. You release, it leans back and slows down.
2. **Velocity Integrator:**
   - The requested acceleration is integrated into a target velocity.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

- This target velocity is fed into the Position Controller (`AC_PosControl`).

3. **Braking Logic:**
   - When sticks are centered, the controller enters a braking phase.
   - It calculates the distance required to stop based on `LOIT_BRK_ACCEL` and `LOIT_BRK_JERK` (smoothness).
   - It effectively generates a virtual waypoint at the stopping point and flies to it.

PILOT INTERACTION

- **Moving:** Push sticks to accelerate. The vehicle will speed up until it reaches `LOIT_SPEED`.
- **Stopping:** Release sticks. The vehicle will "coast" to a stop. The distance it coasts depends on how fast you were going and the Braking parameters.
- **Repositioning:** You can "nudge" the position at any time.

FAILSAFE LOGIC

- **GPS Loss:** If the EKF loses confidence in the position (GPS Glitch/Loss), the mode fails.
  - *Reaction:* It usually triggers an EKF Failsafe event, switching to **Land** or **AltHold** (which doesn't require GPS).
  - *Danger:* If it switches to AltHold, the wind will immediately blow it away. Be ready to take over.

KEY PARAMETERS

PARAMETER	DEFAULT	DESCRIPTION
`LOIT_SPEED`	1250	(cm/s) Max horizontal speed.
`LOIT_ACC_MAX`	500	(cm/s^2) Max acceleration. Higher = punchier.
`LOIT_BRK_ACCEL`	250	(cm/s^2) Deceleration rate when stopping. Higher = stops faster but jerks more.
`LOIT_BRK_JERK`	500	(cm/s^3) "Jerk" limit. How fast the braking acceleration ramps up. Lower = smoother start to braking.

SOURCE CODE REFERENCE

- **Mode Logic:** `ardupilot/ArduCopter/mode_loiter.cpp`
- **Controller Logic:** `AC_Loiter::update()`

PRACTICAL GUIDE: MAKING LOITER FEEL LIKE POSHOLD

Many FPV pilots hate Loiter because it feels "slushy" compared to the direct response of PosHold or Stabilize. You can tune Loiter to be much crisper.

THE PROBLEM: "THE DRIFT"

By default, when you let go of the sticks, Loiter gently decelerates over 2-3 meters. This is great for filming, but bad for precision flying.

THE FIX: AGGRESSIVE BRAKING

1. **Increase `LOIT_BRK_ACCEL`:**
   - Default: $250 cm/s^2$ (2.5 m/s).
   - Try: **500 - 800**. This makes the drone brake twice as hard.
2. **Increase `LOIT_BRK_JERK`:**
   - Default: $500 cm/s^3$.
   - Try: **1000 - 1500**. This allows the braking force to ramp up instantly, rather than easing in.
3. **Increase `LOIT_ACC_MAX`:**
   - Default: 500.
   - Try: **700 - 900**. This makes the drone accelerate faster when you *push* the stick.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





RESULT

The drone will now stop almost instantly when you release the sticks, similar to a DJI drone in "Sport" mode, while still maintaining the GPS position lock safety.

POSHOLD MODE (COPTER)

EXECUTIVE SUMMARY

PosHold (Position Hold) is a hybrid flight mode that attempts to offer the best of both worlds: the direct, responsive feel of **Stabilize** when the pilot is flying, and the GPS-locked position holding of **Loiter** when the pilot releases the sticks. It is often preferred by FPV pilots who want GPS safety without the "heavy" feeling of Loiter.

THEORY & CONCEPTS

1. DIRECT PILOT LINK

The key difference between PosHold and Loiter is the **Stick-to-Angle Mapping**.

- **Loiter:** Stick = Requested Velocity.
- **PosHold:** Stick = Requested Lean Angle.
- **The Difference:** In PosHold, if you tilt the stick 10 degrees, the drone tilts 10 degrees *immediately*. It feels "Crisp" like Stabilize mode, but it won't drift when you let go.

2. WIND CORRECTION MEMORY

Holding position in wind requires leaning into the wind.

- **The Problem:** If you are hovering in a 10 m/s wind, the drone might need a 5-degree lean just to stay still.
- **The Trick:** When you are in the "Hold" phase of PosHold, the EKF learns this **Wind Lean Angle**. When you start flying again, it adds that 5-degree lean as an offset to your stick input, so you don't have to fight the wind yourself.

HARDWARE DEPENDENCY MATRIX

Like Loiter, PosHold relies on GPS for the braking and holding phases.

SENSOR	REQUIREMENT	CODE IMPLEMENTATION NOTES
GPS	CRITICAL	Required for the "Hold" phase and braking logic. Without GPS, the mode cannot stop the vehicle automatically.
Compass	CRITICAL	Required for heading and wind compensation logic.
Barometer	REQUIRED	Used for altitude hold (Z-axis).

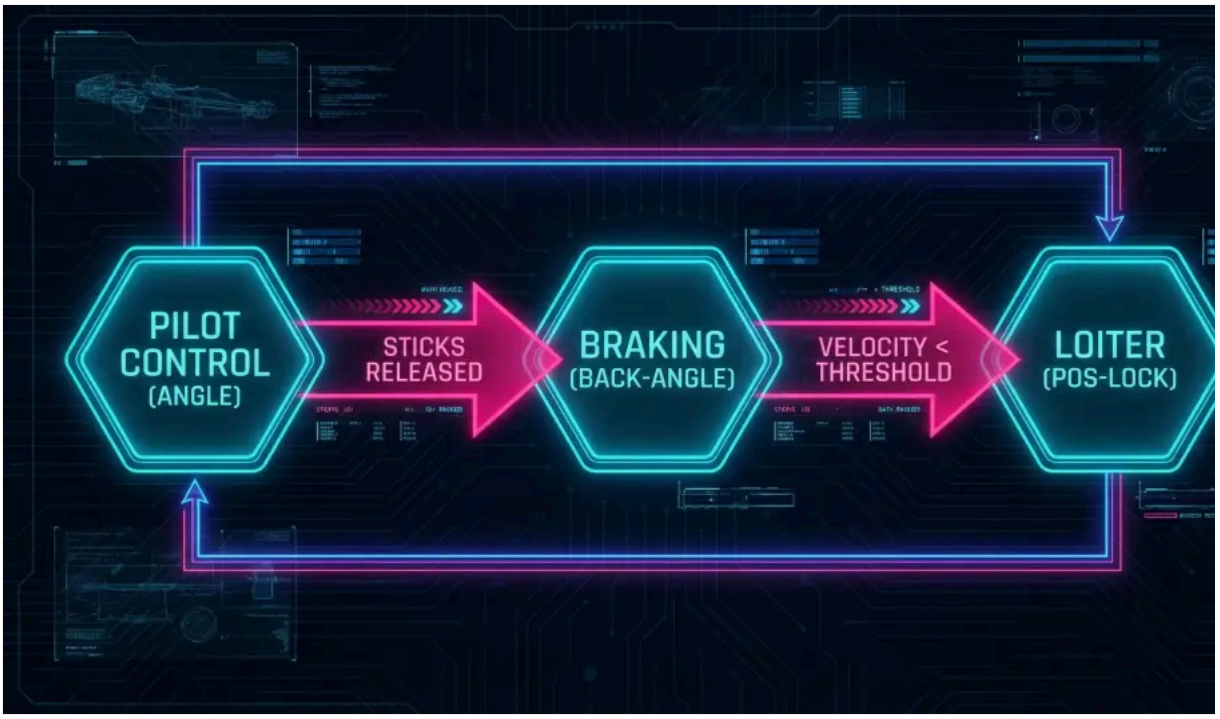
CONTROL ARCHITECTURE (ENGINEER'S VIEW)

PosHold implements a custom State Machine in `ModePosHold::run()` that switches behavior based on pilot input.

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY







1. **Phase 1: Pilot Override (Stick Input)**
    - **Behavior:** The pilot controls the **Lean Angle** directly (like Stabilize/AltHold).
    - *Difference from Loiter:* In Loiter, you command Acceleration/Velocity. In PosHold, you command the angle. This feels much more "connected."
    - *Wind Compensation:* The controller injects a "Wind Lean Angle" estimate (learned during the hold phase) so you don't have to constantly lean into the wind manually.
2. **Phase 2: Braking (Sticks Released)**
    - **Trigger:** When pilot sticks return to center.
    - **Behavior:** The controller calculates a "Back Angle" to arrest velocity.
    - *Math:* It measures the current velocity vector and leans the opposite way.
3. **Phase 3: Loiter (Stopped)**
    - **Trigger:** When velocity drops near zero.
    - **Behavior:** It engages the full **Loiter** controller to lock the 3D position and compensate for drift.

PILOT INTERACTION

- **Flying:** Feels crisp like Stabilize. The drone tilts exactly as much as you move the stick.
- **Stopping:** When you let go, the drone will pitch back aggressively to stop (Braking).
    - *Note:* The transition can sometimes feel "jerky" if the braking rate is high.
- **Throttle:** Standard Altitude Hold (Climb Rate control).

FAILSAFE LOGIC

- **GPS Loss:** If GPS fails, the vehicle cannot perform the Braking or Loiter phases.
    - *Reaction:* It typically triggers an EKF failsafe (Land/AltHold).
    - *Danger:* If it degrades to AltHold, the "Braking" behavior disappears. The drone will just drift with its momentum when you let go.

KEY PARAMETERS

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

PARAMETER	DEFAULT	DESCRIPTION
`PHLD_BRAKE_RATE`	8	(deg/s) Max angle rate for braking. Higher = harder stop.
`PHLD_BRAKE_ANGLE`	3000	(centi-deg) Max lean angle (30 deg) allowed during braking.
`PSC_POSXY_P`	1.0	Position controller P-gain (used during the Hold phase).

SOURCE CODE REFERENCE

- **Mode Logic:** `ardupilot/ArduCopter/mode_poshold.cpp`
- **Stick Smoothing:** `ModePosHold::update_pilot_lean_angle()`

PRACTICAL GUIDE: TUNING THE "BRAKE"

Many pilots find the default braking in PosHold too aggressive, causing the drone to pitch back violently when sticks are released.

THE GOAL

Make the braking transition smoother without sacrificing stopping power.

THE PARAMETERS

1. `PHLD_BRAKE_RATE` **(Default: 8 deg/s):** This controls how fast the drone pitches back to stop.
   - **To Smooth it Out:** Reduce this value to **4 - 6** deg/s. This makes the braking pitch-back slower and less "jerky," preventing the drone from violently bucking when you let go of the sticks.
   - **To Stop Faster:** Increase this value, but be aware that it may cause oscillation if the PID tune is not perfect.
2. `PHLD_BRAKE_ANGLE` **(Default: 3000 cdeg = 30 deg):** This limits the maximum lean angle the drone will use to stop.
   - **To Limit Speed:** If your drone brakes too hard from high speed, reduce this to **2000 (20 deg)**. This forces a longer stopping distance but keeps the airframe more stable.

TROUBLESHOOTING: "TOILET BOWLING"

If the drone stops but then starts swirling in a circle (toilet bowling) during the "Hold" phase, this is **not** a PID issue.

- **Cause:** Almost always Compass interference or Mag calibration.
- **Fix:** Re-calibrate compass and check `COMPASS_MOT_x` (Compass Motor Compensation).

For full parameter details, see the ArduPilot Wiki: PosHold Mode.

RTL MODE (COPTER)

EXECUTIVE SUMMARY

Return to Launch (RTL) is the primary failsafe mode. When engaged, it navigates the vehicle back to its home position (or the nearest Rally Point) at a safe altitude and automatically lands. It uses a predefined state machine to ensure obstacles are cleared before horizontal movement begins.

THEORY & CONCEPTS

1. BARRIER CLEARANCE

The #1 cause of RTL failure is the drone hitting an obstacle on the way home.

- **The Logic:** ArduPilot always **Climbs First**.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- **Safety:** It compares your current altitude to `RTL_ALT`. If you are lower, it climbs straight up before starting the horizontal trip. This ensures it clears trees, buildings, or fences.

2. THE CONE OF SILENCE

Why does the drone sometimes *not* climb when you trigger RTL?

- **The Problem:** If you are 2 meters from your landing spot, you don't want the drone to climb to 30m just to fly 2m sideways.
- **The Geometry:** ArduPilot builds a "Virtual Cone" above the landing point. If you are inside this cone, the RTL altitude is limited to your current height, forcing an immediate descent and land.

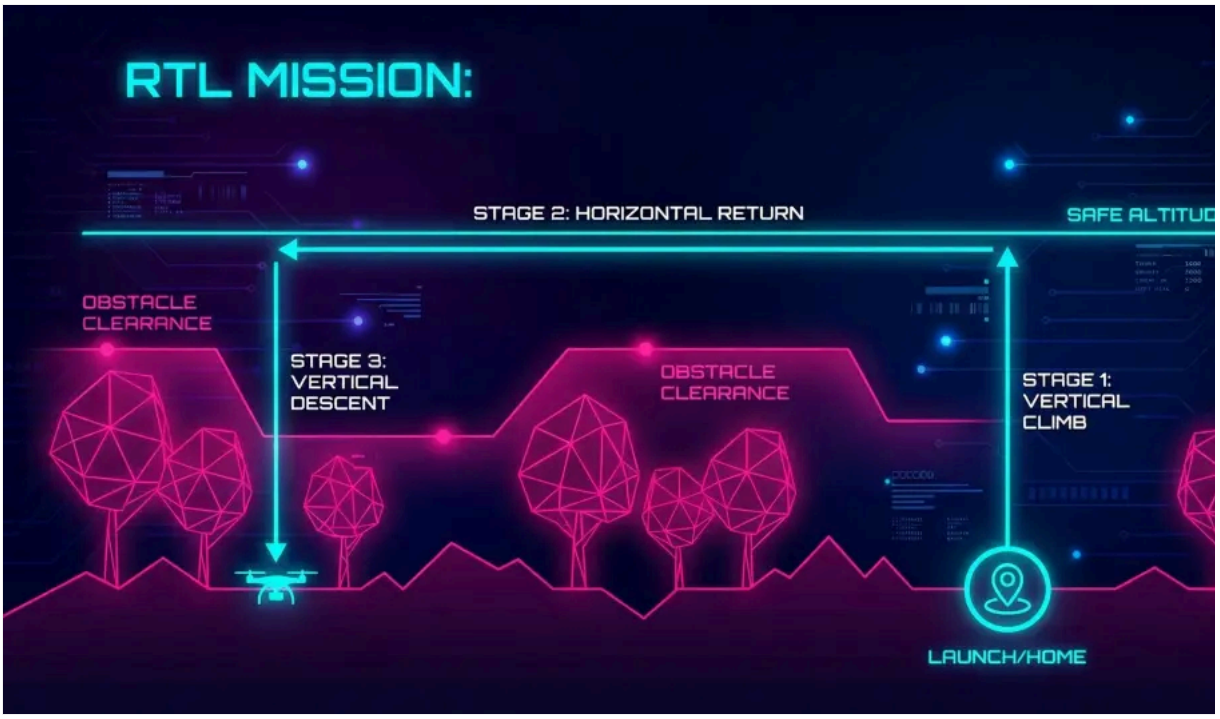
HARDWARE DEPENDENCY MATRIX

RTL is an autonomous navigation mode dependent on accurate positioning.

SENSOR	REQUIREMENT	CODE IMPLEMENTATION NOTES
GPS	CRITICAL	Requires a valid 3D position lock. If GPS is lost during flight, RTL cannot be engaged (the vehicle may land in place or drift).
Compass	CRITICAL	Required for heading control during the return path.
Barometer	REQUIRED	Primary source for altitude relative to Home.
Terrain Data	OPTIONAL	Required if `RTL_ALT_TYPE` is set to "Terrain" to climb over hills.

CONTROL ARCHITECTURE (ENGINEER'S VIEW)

RTL operates as a rigid State Machine defined in `ModeRTL::run()`.



1. **Stage 1: Initial Climb (`INITIAL_CLIMB`)**
    - The vehicle stops horizontal movement.
    - It climbs to `RTL_ALT`.
    - *Optimization:* If already above `RTL_ALT`, it maintains current altitude.

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

- *Cone Logic:* If close to home, `RTL_CONE_SLOPE` limits the climb to prevent shooting up just to come right back down.

2. **Stage 2: Return Home (`RETURN_HOME`)**
   - The vehicle flies a straight line to the Home coordinates.
   - Speed is defined by `WPNAV_SPEED`.
   - Nose orientation is controlled by `WP_YAW_BEHAVIOR` (usually points to home).
3. **Stage 3: Loiter at Home (`LOITER_AT_HOME`)**
   - Once over home, it pauses for `RTL_LOIT_TIME` (ms). This gives the pilot a chance to visually acquire the drone before landing.
4. **Stage 4: Final Descent / Landing (`FINAL_DESCENT` / `LAND`)**
   - It descends to `RTL_ALT_FINAL`.
   - If `RTL_ALT_FINAL` is 0 (default), it proceeds to land and disarm.

TERRAIN FOLLOWING

RTL can be configured to follow terrain using `RTL_ALT_TYPE`.

- **0 (Relative):** Altitude is relative to Home. (Simple geometry).
- **1 (Terrain):** Altitude is relative to the ground beneath the drone.
  - *Requirement:* Needs a Rangefinder OR a valid Terrain Database on the flight controller.
  - *Calculation:* `compute_return_target()` recalculates the target altitude dynamically.

FAILSAFE LOGIC

- **GPS Glitch:** If GPS variance becomes too high during the return leg, the vehicle may switch to **Land** mode immediately to prevent flyaways.
- **Rally Points:** If Rally Points are configured, RTL will choose the *closest* Rally Point instead of Home.

KEY PARAMETERS

PARAMETER	DEFAULT	DESCRIPTION
`RTL_ALT`	1500	(cm) Return altitude (15m).
`RTL_ALT_FINAL`	0	(cm) Altitude to hover at after returning. 0 = Land immediately.
`RTL_LOIT_TIME`	5000	(ms) Time to hover over home before landing.
`RTL_CONE_SLOPE`	0.5	Defines the slope of the "virtual cone" above home. High values force a steeper climb even when close to home.
`RTL_SPEED`	0	(cm/s) Speed for the return leg. 0 = Use `WPNAV_SPEED`.

SOURCE CODE REFERENCE

- **Mode Logic:** `ardupilot/ArduCopter/mode_rtl.cpp`
- **Target Calculation:** `ModeRTL::compute_return_target()`

PRACTICAL GUIDE: CONFIGURING RTL SAFETY

RTL is your insurance policy. Don't trust the defaults.

STEP 1: THE RETURN ALTITUDE

Set `RTL_ALT` to be higher than the tallest obstacle in your flight area (trees/buildings) plus a 10m buffer.

- **Example:** If trees are 20m high, set `RTL_ALT = 3000` (30m).
- **Warning:** Do not set this higher than 120m (400ft) to avoid violating airspace rules.

STEP 2: THE CONE OF SILENCE

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





If you fly in tight spaces (like a backyard), the default "Cone" might force the drone to climb into a tree canopy when you just wanted it to land.

- **The Fix:** Increase `RTL_CONE_SLOPE`.
- **Logic:** A steeper slope means the drone enters the cone sooner.
- **Value:** Try `3.0`. This effectively disables the cone for short distances, forcing the drone to climb to `RTL_ALT` almost immediately, which is safer if you have obstacles nearby but clear sky above.

STEP 3: THE "WAIT"

Always set `RTL_LOIT_TIME` to at least `5000` (5 seconds).

- **Why?** This gives you time to look up, spot the drone overhead, and verify it is descending on the correct spot. If it's drifting (bad GPS), you have 5 seconds to switch to Stabilize and save it.

SIMPLE AND SUPER SIMPLE MODES (COPTER)

EXECUTIVE SUMMARY

Simple and **Super Simple** are not distinct flight modes but rather **Coordinate Transforms** applied to the pilot's Roll and Pitch inputs. They change the "Frame of Reference" for the sticks.

THEORY & CONCEPTS

1. RELATIVE REFERENCE FRAMES

In physics, everything is relative to your "Frame of Reference."

- **Body Frame:** "Forward" is where the nose points.
- **Earth Frame:** "Forward" is North.
- **Pilot Frame (Simple):** "Forward" is the way the drone was pointing when you armed it.
- *Why this helps:* It removes the "Inverted Controls" problem. If the drone is flying towards you, "Right" on the stick still moves the drone to *your* right, even though the drone is rolling to its *left*.

2. SIMPLE MODE (ROTATION)

Locks the control frame to the heading at takeoff (or arming). "Forward" on the stick always flies away from the pilot (assuming the pilot is standing behind the drone at takeoff), regardless of which way the drone's nose is pointing.

3. SUPER SIMPLE MODE (VECTOR)

Locks the control frame to the vector between Home and the Vehicle. "Pull Back" on the stick always brings the drone towards Home.

HARDWARE DEPENDENCY MATRIX

These modes rely entirely on the heading estimation.

SENSOR	REQUIREMENT	CODE IMPLEMENTATION NOTES
Compass	CRITICAL	Accurate heading is required to perform the rotation of the stick inputs. If the compass is wrong, the "Simple" controls will be skewed (e.g., Forward flies diagonal).
GPS	**CRITICAL (Super Simple)**	Super Simple requires knowing the vehicle's position relative to Home. Without a 3D Lock, it behaves like standard Simple mode.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

CONTROL ARCHITECTURE (ENGINEER'S VIEW)

The logic acts as a "Input Shaper" that runs *before* the Flight Mode logic.

1. **Simple Mode (Rotation):**
   - *Reference:* `simple_cos_yaw`, `simple_sin_yaw` (Stored heading at arming).
   - *Operation:* The code rotates the Pilot's Roll/Pitch vector by the difference between the **Current Heading** and the **Reference Heading**.
   - *Effect:* If you yaw the drone 90 degrees right, the controller effectively yaw-rotates your stick input 90 degrees left to compensate.
   - *Code Path:* `Copter::update_simple_mode()`.
2. **Super Simple Mode (Vector):**
   - *Reference:* Home Position.
   - *Operation:* It calculates the bearing from Home to the Vehicle.
   - *Logic:* It treats that bearing as "Forward".
   - *Smart Zone:* If the vehicle flies within 10m of home, Super Simple automatically disables to prevent rapid control reversal (the "singularity" at the origin).

PILOT INTERACTION

- **Activation:** These are typically assigned to checkboxes on the Flight Mode switch setup (Channel 5) or separate auxiliary switches (Ch7/Ch8).
- **Reset:** The "Simple" reference heading can be reset in flight by toggling the switch or re-arming.
- **Disorientation:** While intended for beginners, these modes can be dangerous if the compass fails or if the pilot physically moves from their original takeoff position.

FAILSAFE LOGIC

- **GPS Loss:** Super Simple mode degrades to Simple Mode logic (using the last known bearing or simple heading).
- **Compass Variance:** If the EKF declares a Compass Variance error, Simple Mode may become unpredictable.

KEY PARAMETERS

PARAMETER	DEFAULT	DESCRIPTION
`SUPER_SIMPLE`	0	Can be used to force Super Simple behavior if `SIMPLE` mode is engaged.
`SIMPLE_MODE`	0	(Bitmask) Used to enable Simple mode on specific flight mode switch positions.

SOURCE CODE REFERENCE

- **Transformation Logic:** `Copter::update_simple_mode()`

SMART RTL MODE (COPTER)

EXECUTIVE SUMMARY

Smart RTL (Return to Launch) is a sophisticated variation of the standard RTL mode. Instead of flying a straight line home at a fixed altitude, it **backtracks** along the exact path the vehicle traveled. This is invaluable when flying around obstacles (like behind a building or under trees) where a straight-line return would result in a crash.

THEORY & CONCEPTS

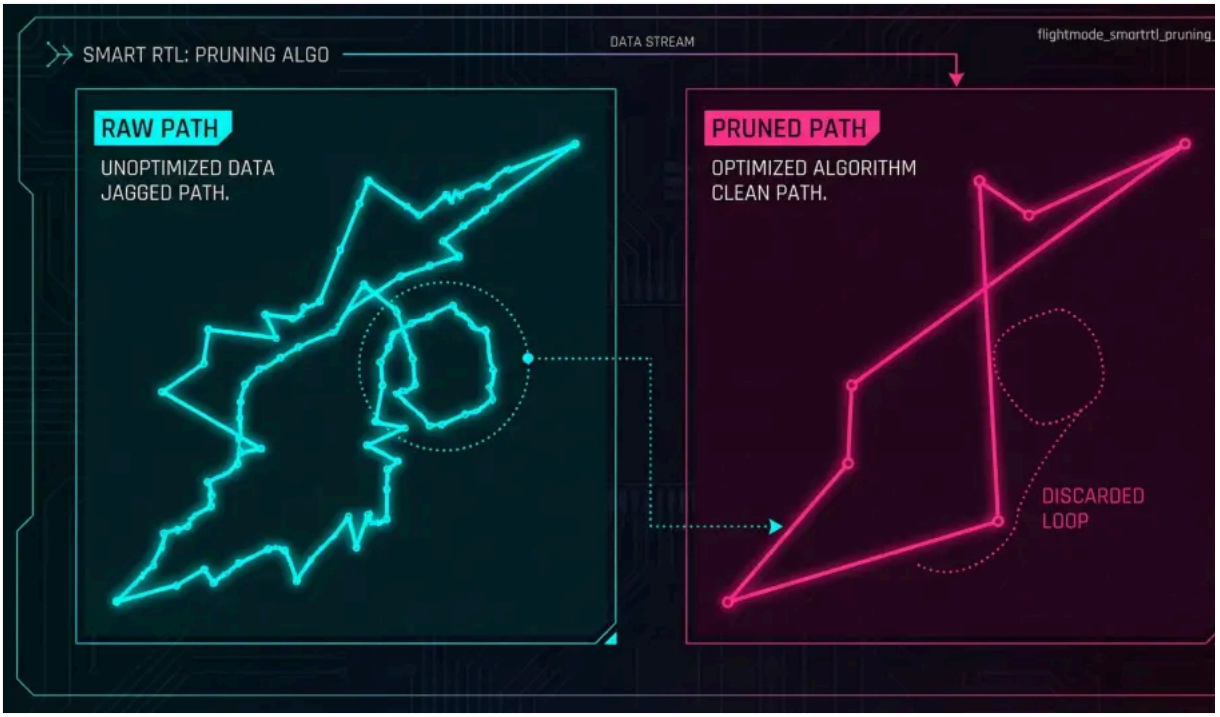
1. THE BREADCRUMB ALGORITHM

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





Smart RTL is a classic **Breadcrumb** algorithm.



- **The Problem:** GPS signals only tell you where you are, not where it's safe to fly.
- **The Logic:** "If I flew through here successfully 2 minutes ago, it's probably safe to fly through here again now."
- **Path Pruning:** To avoid wasting memory on every tiny movement, ArduPilot uses the **Douglas-Peucker Algorithm**. It analyzes the path and keeps only the "Corner" points, discarding redundant points on a straight line.

2. SAFEGUARD VS. BEELINE

Standard RTL is a "Beeline"—it takes the shortest distance. Smart RTL is a "Safeguard"—it takes the proven distance. ArduPilot will choose the proven distance whenever possible to maximize airframe safety.

HARDWARE DEPENDENCY MATRIX

Smart RTL is resource-intensive and relies on precise tracking.

SENSOR	REQUIREMENT	CODE IMPLEMENTATION NOTES
GPS	CRITICAL	Requires a 3D Lock to record the path points. If GPS is lost during flight, path recording stops.
Compass	CRITICAL	Required for heading control during the return.
Memory	CRITICAL	The flight controller allocates a specific buffer (`SRTL_POINTS`) to store the path.

CONTROL ARCHITECTURE (ENGINEER'S VIEW)

Smart RTL relies on the `AP_SmartRTL` library to record and optimize the path in real-time.

1. **Path Recording:**
   - As you fly, the controller records your position every `SRTL_ACCURACY` meters (default 2m).
   - It stores these as 3D vectors in a ring buffer.
2. **Path Optimization (Pruning):**
   - To save memory, a background task constantly "prunes" the path.
   - **Simplification:** It removes points that form a straight line (Ramer-Douglas-Peucker algorithm).

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

- **Loop Closing:** If you fly in a circle and cross your own path, it "cuts the loop," discarding the circle segments. This ensures the return path is always the shortest *safe* route back.

3. **Execution:**
   - When engaged, the vehicle pauses, climbs to `RTL_ALT` (optional), and then flies the recorded waypoints in reverse order.
   - Once it reaches the arming point, it performs a standard Land.

FAILSAFE LOGIC

- **Buffer Overflow:** If the buffer fills up (e.g., a very long, complex flight), the system stops recording new points.
  - *Warning:* Users should monitor the `SRTL` messages. If the buffer is full, Smart RTL might not bring you all the way back to the latest point, but it will still backtrack the recorded path.
  - *Hard Failsafe:* If `SRTL` cannot initialize (e.g., bad GPS at takeoff), the system automatically falls back to **Standard RTL** or **Land**.
- **GPS Loss:** If GPS is lost in flight, the path becomes invalid. Smart RTL will fail to engage, forcing a fallback to Land/AltHold.

KEY PARAMETERS

PARAMETER	DEFAULT	DESCRIPTION
`SRTL_ACCURACY`	2.0	(m) Min distance between recorded points.
`SRTL_POINTS`	300	Max number of points to record. 300 points @ 2m spacing = ~600m of unique path (loops don't count).
`RTL_ALT`	1500	(cm) Minimum return altitude.
`RTL_SPEED`	0	Return speed (0 = `WPNAV_SPEED`).

SOURCE CODE REFERENCE

- **Mode Logic:** `ardupilot/ArduCopter/mode_smart_rtl.cpp`
- **Library Logic:** `AP_SmartRTL`

PRACTICAL GUIDE: TESTING SMART RTL

Before trusting your drone to backtrack autonomously, verify the buffer logic.

1. THE "U" TEST

1. Take off in **Loiter**.
2. Fly 20m forward.
3. Fly 20m Right.
4. Fly 20m Back (forming a 'U' shape).
5. Engage **Smart RTL**.
6. **Observation:** The drone should fly Forward → Left → Backward (retracing the U).
7. **Failure:** If it flies diagonally straight to home, Smart RTL failed (or buffer was empty) and it fell back to Standard RTL.

2. THE LOOP CUT

1. Fly a complete circle (starting and ending at the same spot).
2. Fly 10m away.
3. Engage **Smart RTL**.
4. **Observation:** The drone should fly straight back to the circle start point, ignoring the loop. This confirms the pruning algorithm is saving memory.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





SPORT MODE (COPTER)

EXECUTIVE SUMMARY

Sport Mode offers a unique control scheme: **Earth-Frame Rate Control**. Like Acro, the sticks command angular rotation rates (deg/s) instead of angles. However, unlike Acro, these rates are relative to the **Earth's Horizon**, not the vehicle's body.

THEORY & CONCEPTS

1. EARTH FRAME RATES (RATE ISOLATION)

The defining feature of Sport Mode is that it **decouples** the drone's tilt from its rotation.

- **Acro (Body Frame):** If you are nose-down and you yaw, the drone rolls.
- **Sport (Earth Frame):** If you are nose-down and you yaw, the drone just spins its nose while staying nose-down. The autopilot handles the complex motor mixing to keep the pitch/roll constant relative to the *horizon*, not the drone.

2. THE ATTITUDE LEASH

Think of Sport mode as "Rate mode with a leash." You have the freedom to rotate as fast as you want, but the autopilot holds onto a leash (`ANGLE_MAX`). Once you hit that limit, the autopilot "yanks" the leash to prevent the drone from flipping over, even though you are in a rate-control mode.

HARDWARE DEPENDENCY MATRIX

Sport Mode is robust and requires minimal sensors.

SENSOR	REQUIREMENT	CODE IMPLEMENTATION NOTES
Gyroscope	CRITICAL	Required for the rate controller.
Accelerometer	CRITICAL	Required to estimate the Earth-Frame (Horizon). Without an accurate level estimate, the controller cannot decouple Body-Roll from Earth-Roll.
Compass	OPTIONAL	Helpful for maintaining heading drift, but not strictly required for the core stabilization.
GPS	**NONE**	Not used.

CONTROL ARCHITECTURE (ENGINEER'S VIEW)

Sport Mode sits between Stabilize and Acro.

1. **Stick to Earth-Rate:**
   - Pilot inputs are converted to target rates (deg/s).
   - The code uses `input_euler_rate_roll_pitch_yaw()` .
   - *Transformation:* The controller calculates the necessary **Body Rates** (p, q, r) to achieve the requested **Euler Rates** (dRoll/dt, dPitch/dt).
2. **Angle Limiting (The Leash):**
   - Unlike Acro, Sport mode respects `ANGLE_MAX` .
   - *Mechanism:* It continuously predicts where the vehicle will be. If the predicted angle exceeds `ANGLE_MAX` , it overrides the pilot's rate command to "push" the vehicle back. This feels like hitting a soft wall.
3. **Self-Leveling (Virtual Flybar):**
   - When sticks are centered, the vehicle slowly returns to level (controlled by `ACRO_BAL_ROLL` / `ACRO_BAL_PITCH`). This makes it safer than Acro for FPV beginners.

PILOT INTERACTION

- **Turning:** You can bank the aircraft and pull back on the pitch stick to turn, similar to a plane or Acro mode.
- **Protection:** You can hold full stick, and the drone will pitch/roll aggressively but never flip over (it stops at `ANGLE_MAX`).

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY



- **Throttle:** Standard AltHold logic (Climb Rate).

FAILSAFE LOGIC

- **Crash Check:** Standard crash detection.
- **Attitude Loss:** If the AHRS (Accelerometer) fails or the EKF tumbles, Sport mode becomes disoriented because "Down" is unknown.

KEY PARAMETERS

PARAMETER	DEFAULT	DESCRIPTION
`ACRO_RP_RATE`	360	(deg/s) Max rotation rate. Shared with Acro.
`ANGLE_MAX`	4500	(centi-deg) The hard limit for bank/pitch angles.
`ACRO_BAL_ROLL`	1.0	Strength of self-leveling when sticks are centered.

SOURCE CODE REFERENCE

- **Mode Logic:** `ardupilot/ArduCopter/mode_sport.cpp`
- **Rate Loop:** `ModeSport::run()`

STABILIZE MODE (COPTER)

EXECUTIVE SUMMARY

Stabilize Mode is the most fundamental flight mode for multicopters. It provides self-leveling attitude control on the Roll and Pitch axes, while giving the pilot direct manual control over Yaw Rate and Throttle. It is the fallback mode for almost all safety failsafes because it relies on the absolute minimum amount of sensor data.

THEORY & CONCEPTS

1. STABILITY VS. AGILITY

In control theory, **Stability** is the ability of a system to return to equilibrium (level). **Agility** is the ability to change state quickly.

- **The Stabilize Trade-off:** This mode is biased towards stability. The "Angle P" gain determines how hard the drone fights to return to level. If you set it too high, the drone will shake. If too low, it feels "soupy" and slow to react.

2. MANUAL THROTTLE LINEARIZATION

Human brains are good at controlling speed, but bad at controlling power.

- **The Problem:** As a battery drains, 50\% power produces less thrust.
- **The Stabilization:** Even in "Manual" throttle, ArduPilot applies **Voltage Compensation**. It automatically increases the motor power as the battery sags, so the drone's hover position stays consistent throughout the entire flight.

HARDWARE DEPENDENCY MATRIX

Stabilize requires only the core inertial sensors.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

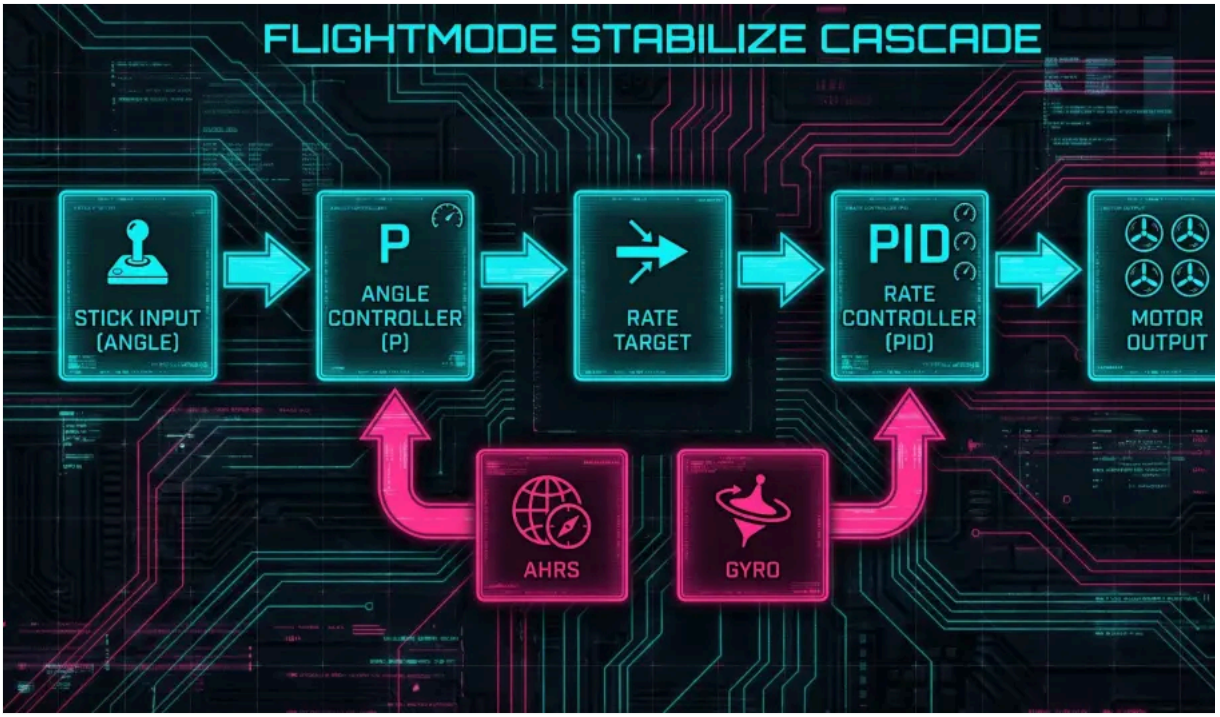




SENSOR	REQUIREMENT	CODE IMPLEMENTATION NOTES
Gyroscope	CRITICAL	Required for the Rate Controller (inner loop).
Accelerometer	CRITICAL	Required for the Angle Controller (outer loop) to determine "Level".
Compass	OPTIONAL	Heading drift may occur without it, but the mode remains flyable.
GPS	**NONE**	Not used.
Barometer	**NONE**	Not used. Throttle is manual.

CONTROL ARCHITECTURE (ENGINEER'S VIEW)

Stabilize implements a standard Cascade Controller:



1. **Angle Controller (Outer Loop):**
   - Input: Pilot Stick Angle (e.g., 30 deg).
   - Feedback: Current Estimated Angle (AHRS).
   - Output: Target Rate (deg/s).
   - *Gain:* `ATC_ANG_RLL_P` / `ATC_ANG_PIT_P`.
2. **Rate Controller (Inner Loop):**
   - Input: Target Rate (from Outer Loop) + Pilot Yaw Rate.
   - Feedback: Gyroscope Rate.
   - Output: Motor PWM.
   - *Gain:* `ATC_RAT_RLL_P/I/D`.

PILOT INTERACTION

- **Roll/Pitch:** Stick deflection maps to **Lean Angle**.
  - Center stick = 0 degrees (Level).
  - Full stick = `ANGLE_MAX` (default 45 degrees).
  - *Note:* The mapping is non-linear (tangent curve) to provide higher resolution near the center.
- **Yaw:** Stick deflection maps to **Yaw Rate** (deg/s).
- **Throttle:** Manual control.

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

- The output is normalized (0-1) and curve-fitted so that **Mid-Stick** corresponds to the **Hover Throttle** (`MOT_THST_HOVER`). This makes hovering easier than a pure linear map.

FAILSAFE LOGIC

- **Attitude Loss:** If the AHRS fails (E.g., high vibration confuses the accelerometer), Stabilize mode will fail to level the aircraft. This is often catastrophic.
- **Battery Failsafe:** Since Stabilize has no GPS, a Battery Failsafe action of "RTL" or "SmartRTL" will fail to engage. It typically downgrades to "Land".

KEY PARAMETERS

PARAMETER	DEFAULT	DESCRIPTION
`ANGLE_MAX`	4500	(centi-deg) Max lean angle at full stick.
`ATC_INPUT_TC`	0.15	Time constant for smoothing stick inputs. Higher = softer feel.
`MOT_THST_HOVER`	0.35	(0.0-1.0) The estimated throttle required to hover. The throttle curve centers around this value.

SOURCE CODE REFERENCE

- **Mode Logic:** `ardupilot/ArduCopter/mode_stabilize.cpp`
- **Control Loop:** `ModeStabilize::run()`

PRACTICAL GUIDE: TUNING STABILIZE

The default tuning is designed for large, slow drones. For small, agile quads, it feels terrible.

1. INCREASE THE ANGLE LIMIT

- **Parameter:** `ANGLE_MAX`
- **Default:** 4500 (45 degrees).
- **Tuning:** Increase to **5500** (55 deg) or **6000** (60 deg). This allows you to fly faster and fight strong winds.

2. TIGHTEN THE OUTER LOOP

If the drone feels "lazy" (slow to start tilting), increase the Angle P-Gain.

- **Parameter:** `ATC_ANG_RLL_P` / `ATC_ANG_PIT_P`
- **Default:** 4.5.
- **Tuning:** Try **6.0**. This makes the self-leveling reaction much snappier.

3. RATE VS. ANGLE

Don't confuse them.

- **Oscillates ONLY when you let go of the stick:** `ATC_ANG_RLL_P` is too high (Angle Loop).
- **Oscillates ALL the time:** `ATC_RAT_RLL_P` or `D` is too high (Rate Loop).

SYSTEM IDENTIFICATION MODE (COPTER)

EXECUTIVE SUMMARY

System Identification (SysID) is an advanced engineering mode designed to mathematically model the vehicle's physical response characteristics. It automatically injects a "Chirp" signal (frequency sweep) into a specific control axis while recording high-speed

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY



data logs. This data is used offline to generate Bode plots, identifying the vehicle's bandwidth, delay, and resonance frequencies for precise tuning.

THEORY & CONCEPTS

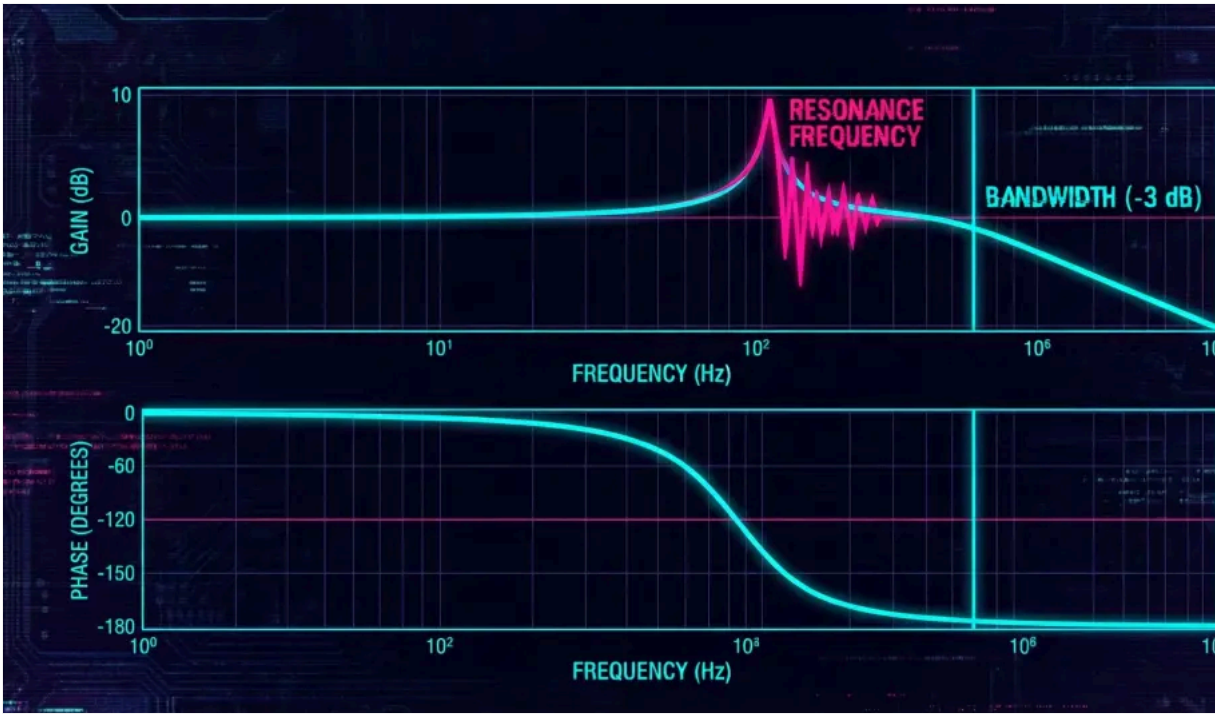
1. FREQUENCY DOMAIN ANALYSIS

Most flight modes operate in the **Time Domain** (How fast can I move *now*?). System ID operates in the **Frequency Domain** (How well can I move at *10 Hz*?).

- **The Concept:** Every drone has a "Resonant Frequency" where the frame or arms vibrate.
- **The Test:** By sweeping through frequencies, the EKF can find these resonant peaks and automatically configure **Notch Filters** to ignore them.

2. BODE PLOTS AND BANDWIDTH

A Bode plot shows the "Gain" and "Phase" of a system.



- **Gain:** If I command 10 degrees, do I actually get 10 degrees?
- **Phase:** If I command a move, how many milliseconds later does it happen?
- **Control Bandwidth:** This test reveals the "Speed limit" of your drone's brain. If the bandwidth is 10Hz, the drone cannot react to anything faster than 10 times per second.

HARDWARE DEPENDENCY MATRIX

This mode is for data gathering, not navigation.

SENSOR	REQUIREMENT	CODE IMPLEMENTATION NOTES
SD Card	CRITICAL	Requires high-speed logging. If logging fails, the test is useless.
Gyroscope	CRITICAL	The primary sensor being measured against the input chirp.
GPS	**CONTEXTUAL**	Required if identifying Position Controller loops (`SID_AXIS` > 13). Not required for Rate loops.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

CONTROL ARCHITECTURE (ENGINEER'S VIEW)

The mode operates as a signal injector overlaid on top of a standard flight mode.

1. **Signal Generation (`Chirp`):**
   - It generates a waveform that sweeps from `SID_F_START_HZ` to `SID_F_STOP_HZ` over `SID_T_REC` seconds.
   - The amplitude is defined by `SID_MAGNITUDE`.
2. **Injection Point:**
   - The chirp is added *on top* of the pilot's input.
   - *Example:* If `SID_AXIS` = 1 (Roll Angle), the controller commands `Target_Roll = Pilot_Roll + Chirp_Signal`.
   - *Benefit:* The pilot can still fly the drone to keep it level or away from walls while the jittery chirp test runs.
3. **Data Logging:**
   - It writes `SIDS` (Setup) and `SIDD` (Data) messages to the dataflash log at full loop rate (400Hz+).

SAFETY LOGIC

- **Mode Switch Abort:** Switching flight modes instantly stops the test.
- **Angle Limits:** If the vehicle leans past `ANGLE_MAX` (due to a violent chirp response), the test aborts.
- **Landing Detector:** If the vehicle lands, the test aborts.
- **Entry Checks:**
   - For Rate/Angle tests, you must enter from a **Manual Throttle** mode (Stabilize/AltHold) to ensure you have control authority.
   - For Position/Velocity tests, you must enter from **Loiter** to ensure the position controller is active.

KEY PARAMETERS

PARAMETER	DEFAULT	DESCRIPTION
`SID_AXIS`	0	0=Disabled. 1=Roll Angle, 2=Pitch Angle, 3=Yaw Angle... (See wiki for full list).
`SID_MAGNITUDE`	0	Amplitude of the injection. Start small! (e.g., 5-10 deg for Angle).
`SID_F_START_HZ`	0.5	Start frequency of the sweep.
`SID_F_STOP_HZ`	30	Stop frequency.
`SID_T_REC`	50	Duration of the sweep (seconds).

SOURCE CODE REFERENCE

- **Mode Logic:** `ardupilot/ArduCopter/mode_systemid.cpp`
- **Chirp Generator:** `Chirp::update()`

PRACTICAL GUIDE: THE FREQUENCY SWEEP

Stop guessing your notch filters. Measure them.

STEP 1: CONFIGURATION

1. **Axis:** Set `SID_AXIS = 1` (Roll Rate). This is the safest to test.
2. **Magnitude:** Set `SID_MAGNITUDE = 5` (5 deg/s). This is gentle.
3. **Frequency:** Start 0.5 Hz, Stop 40 Hz. Duration 60s.

STEP 2: THE FLIGHT

1. Take off in **AltHold**.
2. Hover at a safe height (5m).
3. Switch to **SystemID** mode.
4. **Hands Off:** Try not to touch the Roll stick. The drone will start to "shiver" (low frequency) and then buzz (high frequency).

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





5. **Wait:** Wait for the test to finish (60s) or until the drone stops buzzing.
6. Land and Disarm.

STEP 3: ANALYSIS

1. Upload the log to the ArduPilot WebTools SysID Plotter.
2. Look at the **Frequency Response**.
3. **The Peak:** If you see a massive spike at 150Hz, that is your frame resonance. Set `INS_HNTCH_FREQ = 150` to kill it.

THROW MODE (COPTER)

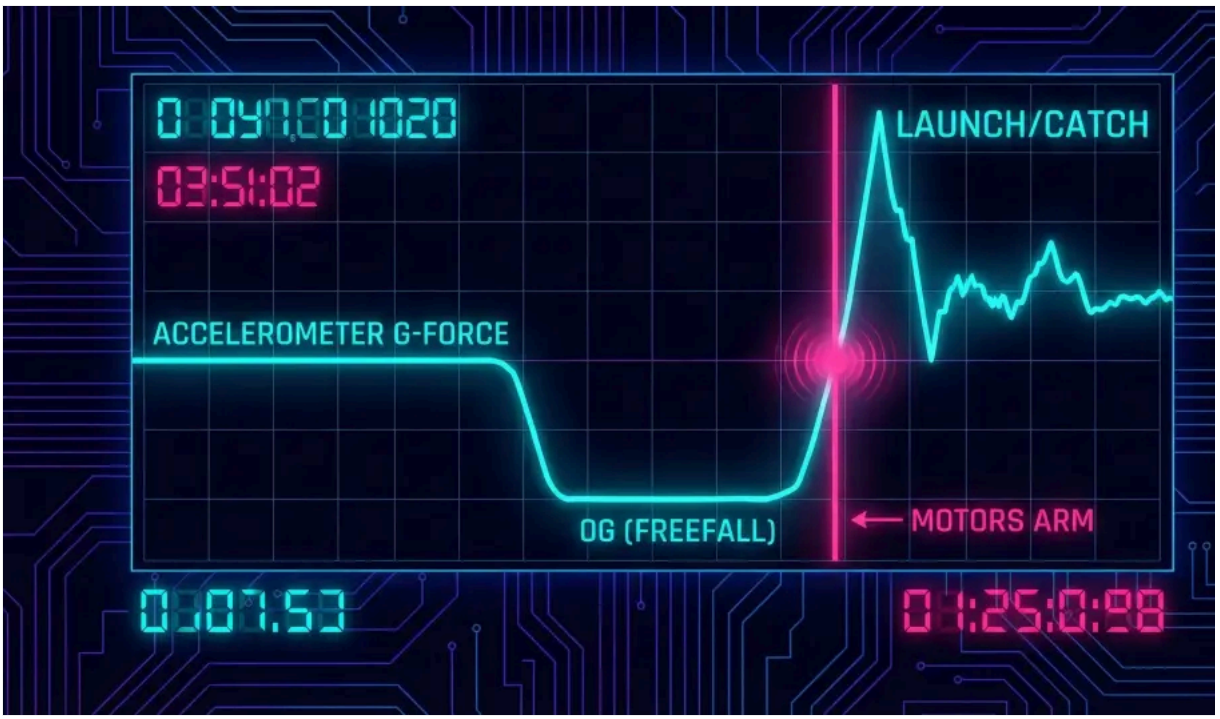
EXECUTIVE SUMMARY

Throw Mode is a specialized launch mode that allows a multicopter to be thrown into the air (or dropped from another vehicle) to start a flight. The vehicle detects the free-fall or high-velocity launch, automatically spools up the motors, stabilized itself, and then holds position. It is ideal for small racers or rugged drones where ground takeoffs are impossible (e.g., from a boat or tall grass).

THEORY & CONCEPTS

1. FREE-FALL PHYSICS

How does a drone know it was "thrown"?



- **The Weightless Point:** When you throw a drone up, there is a split second at the top of the arc where it is weightless.
- **The Signal:** The Accelerometer measures exactly **0G** on all axes.
- **The Logic:** ArduPilot's "Throw" detector looks for this 0G signature followed by a sudden "Catch" (as the drone stabilizes).

2. CENTRIPETAL COMPENSATION

Throwing a drone often involves a spinning motion.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

- **The Problem:** Centripetal force looks like gravity to an accelerometer.
- **The Solution:** ArduPilot uses the **Gyroscope** to calculate how much "Fake Gravity" is being generated by the spin and subtracts it from the accelerometer reading. This ensures the drone detects the release even if you throw it like a frisbee.

HARDWARE DEPENDENCY MATRIX

Throw Mode requires precise sensing to differentiate between "being held" and "being thrown".

SENSOR	REQUIREMENT	CODE IMPLEMENTATION NOTES
Accelerometer	CRITICAL	Used to detect free-fall (gravity → 0) and launch acceleration.
GPS	CRITICAL	Required for the final `Throw_PosHold` stage. Without GPS, the vehicle cannot stop its horizontal momentum after stabilization.
Barometer	REQUIRED	Used for altitude hold and vertical velocity checks.

CONTROL ARCHITECTURE (ENGINEER'S VIEW)

Throw Mode is a strict 5-Stage State Machine defined in `ModeThrow::run()`.

1. **Stage 1: Detecting (`Throw_Detecting`)**
   - **Motor State:** Motors are stopped or idling (`THROW_MOT_START`).
   - **Logic:** The code monitors vertical velocity and acceleration.
   - **Trigger:** It looks for a "Throw Event":
     - Vertical Velocity > 0.5 m/s.
     - High Total Speed (> 5 m/s) OR Freefall (Accel Z > -0.25g).
   - *Code Path:* `throw_detected()`.
2. **Stage 2: Spool Up (`Throw_Wait_Throttle_Unlimited`)**
   - Once detected, motors are commanded to spool up. The controller waits until the motors report they are ready for full throttle authority.
3. **Stage 3: Uprighting (`Throw_Uprighting`)**
   - The Attitude Controller fights to bring the vehicle level.
   - *Constraint:* It accepts any heading but demands 0 pitch/roll.
4. **Stage 4: Height Stabilize (`Throw_HgtStabilise`)**
   - The vehicle arrests its vertical momentum and climbs/descends to the target altitude (usually 3m above launch).
5. **Stage 5: Position Hold (`Throw_PosHold`)**
   - The vehicle brakes horizontally and enters a Loiter.
   - Once stable, it switches to `THROW_NEXTMODE` (default: Loiter).

SAFETY & INTERLOCKS

- **Arming:** You must arm the vehicle *before* throwing it.
- **Safety Switch:** Be careful with the physical safety switch on GPS units; you cannot press it once the drone is in the air.
- **Propeller Danger:** The motors *can* spin while you are holding it if `THROW_MOT_START` is set to 1. Set to 0 (Stopped) for safer hand-launching.

KEY PARAMETERS

PARAMETER	DEFAULT	DESCRIPTION
`THROW_TYPE`	0	0=Upward Throw, 1=Drop (e.g. from a plane).
`THROW_MOT_START`	0	0=Motors Stop, 1=Motors Idle. '0' is safer for hand throws.
`THROW_NEXTMODE`	5	The mode to switch to after stabilizing (5 = Loiter, 6 = RTL, etc.).
`THROW_ALT_MIN`	0	Min altitude change required to confirm throw.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





SOURCE CODE REFERENCE

- **Mode Logic:** `ardupilot/ArduCopter/mode_throw.cpp`
- **Detection Logic:** `ModeThrow::throw_detected()`

PRACTICAL GUIDE: THE "BOAT LAUNCH" TECHNIQUE

Launching a drone from a small, rocking boat is impossible in Stabilize mode (the gyro initializes wrong) or Loiter (it thinks it's moving). Throw Mode is the answer.

STEP 1: CONFIGURATION

- **Set** `THROW_MOT_START = 0`. This ensures the props **do not spin** when you arm. This is critical for hand safety.
- **Set** `THROW_TYPE = 0` (Upward).

STEP 2: THE PROCESS

1. Hold the drone firmly by the body.
2. Switch to **Throw Mode**.
3. **Arm the drone.** The motors will beep, but props will stay stationary. You are now "Live".
4. **The Throw:** Throw the drone **Up and Away** from the boat. Don't be gentle. It needs to detect >5 m/s acceleration or freefall.
5. **The Catch:** About 0.5s after release, the motors will roar to life. The drone will level itself and climb to 3m above the throw height.
6. **Takeover:** Once it is hovering stable, switch to Loiter and fly your mission.

TURTLE MODE (COPTER)

EXECUTIVE SUMMARY

Turtle Mode (officially "Crash Flip") is designed to recover a multicopter that has flipped upside down after a crash. Instead of walking to the vehicle, the pilot can use this mode to reverse the motors and "flip" the drone upright using its own thrust.

THEORY & CONCEPTS

1. REVERSE THRUST MECHANICS

Standard propellers and motors are optimized to spin in one direction.

- **The Airfoil:** Propeller blades have a specific shape (airfoil) that creates lift. When spun backwards, they are incredibly inefficient.
- **The Solution:** To flip a drone over, you don't need *efficient* lift, you just need *force*. ArduPilot commands the ESC to reverse the magnetic field of the motor, producing enough negative thrust to "kick" the drone off the ground.

2. GEOMETRY OF THE FLIP

Flipping an upside-down drone is a game of **Leverage**.

- The drone is a square. If it is resting on its "top," it is stable.
- By spinning only the two motors on one side (e.g., the Right side) in reverse, you create a torque around the Left side's props. This leverages the drone's own weight to pivot it over its edge.

HARDWARE DEPENDENCY MATRIX

This mode has strict hardware requirements related to the ESCs.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





SENSOR/HARDWARE	REQUIREMENT	CODE IMPLEMENTATION NOTES
DShot ESCs	CRITICAL	The ESCs must support DShot protocol (DShot300, DShot600, etc.) to accept the "Reverse Direction" command.
Gyroscope	OPTIONAL	Not strictly required for the flip itself (it's open-loop), but useful for detecting when upright.
GPS	**NONE**	Not used.

CONTROL ARCHITECTURE (ENGINEER'S VIEW)

Turtle Mode operates very differently from normal flight because it requires **reversing motor direction**.

1. **Entry Logic:**
   - When engaged, the mode checks if DShot is enabled. If not, it fails to initialize.
   - It sends a DShot command (`DSHOT_REVERSE`) to the ESCs.
2. **Motor Selection Logic:**
   - The code calculates which motors need to spin based on the pilot's stick input.
   - *Example:* If you push the Pitch Stick Forward, the "Rear" motors spin (in reverse) to lift the tail and flip the nose over.
   - *Math:* It projects the stick vector onto each motor's position vector to determine its contribution.
3. **Exit Logic:**
   - When you switch out of Turtle mode, the code sends `DSHOT_NORMAL` to restore normal rotation direction.

PILOT INTERACTION

- **Arming:** You must be DISARMED to enter Turtle Mode. Once in the mode, you must ARM the vehicle.
  - *Safety:* The motors will NOT spin immediately upon arming.
- **Flipping:**
  - Raise the throttle (safety interlock).
  - Push Pitch or Roll stick in the direction you want to flip.
  - The motors opposite that direction will spin up in reverse.
- **Disarming:** Once upright, DISARM immediately. Then switch back to a normal flight mode (Stabilize/Acro).

FAILSAFE LOGIC

- **DShot Failure:** If the ESCs do not acknowledge the reverse command (or support it), the motors might spin the wrong way, driving the drone into the ground instead of flipping it. Test this on the bench (props off) first!

KEY PARAMETERS

PARAMETER	DEFAULT	DESCRIPTION
`CRASH_FLIP_RATE`	0	(0-100) Controls the power of the flip. Higher values spin motors faster. Start low!
`FRAME_CLASS`	1	Must be a Multicopter (Quad, Hex, Octo). Not valid for Helis or Tricopters.

SOURCE CODE REFERENCE

- **Mode Logic:** `ardupilot/ArduCopter/mode_turtle.cpp`
- **Initialization:** `ModeTurtle::init()`

ZIGZAG MODE (COPTER)

EXECUTIVE SUMMARY

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





ZigZag Mode is a specialized semi-autonomous mode designed for agricultural spraying or lawn-mowing patterns. It allows a pilot to define two points (A and B) and then automatically fly back and forth between them, optionally stepping sideways by a fixed distance (`SID_DIST`) after each pass.

THEORY & CONCEPTS

1. SYSTEMATIC COVERAGE (THE LAWNMOWER PATTERN)

Systematic coverage is a common robotics problem.

- **The Problem:** How to ensure 100\% of a field is sprayed without gaps?
- **The Logic:** You define a "Baseline" (A to B). You then "Offset" that baseline by a fixed distance (e.g. 5m) for each pass.
- **The Math:** ArduPilot uses **Parallel Vector Projection**. It calculates the perpendicular vector to your path and shifts your target waypoint along that vector.

2. A-B REFERENCING

ZigZag simplifies the interface for the pilot. Instead of creating a complex mission with 100 waypoints, you only need to fly to two spots. The "A-B" logic allows the drone to generate the rest of the mission "on the fly" in the air, making it extremely efficient for field work where the boundaries might change slightly between flights.

HARDWARE DEPENDENCY MATRIX

ZigZag requires precise positioning to fly straight lines.

SENSOR	REQUIREMENT	CODE IMPLEMENTATION NOTES
GPS	CRITICAL	Required to define Points A and B and to navigate the line between them.
Compass	CRITICAL	Required for heading and grid alignment.
Barometer	REQUIRED	Used for altitude hold (Z-axis).

CONTROL ARCHITECTURE (ENGINEER'S VIEW)

ZigZag combines **Loiter** (for setup) and **WPNav** (for execution).

1. **Setup Phase (Loiter):**
   - The vehicle flies in a Loiter-like manual mode.
   - Pilot uses an Auxiliary Switch (`ZIGZAG_SaveWP`) to capture the current GPS coordinates as **Point A** or **Point B**.
2. **Execution Phase (WPNav):**
   - When triggered, the vehicle calculates a vector from A to B.
   - It uses the `AC_WPNav` library to fly a straight line to the destination.
   - *Sideways Step:* If configured, it calculates a perpendicular vector (`cross_product`) to the line AB and moves `ZIGZAG_SIDE_DIST` meters before starting the next pass.

PILOT INTERACTION

- **Switch 1 (Mode):** Select ZigZag Mode on the flight mode switch.
- **Switch 2 (Save):** A 3-position switch assigned to `ZIGZAG_SaveWP` (61).
  - **Low:** Save Point A.
  - **Mid:** Manual Control.
  - **High:** Save Point B.
- **Switch 3 (Auto) [Optional]:** A switch assigned to `ZIGZAG_Auto` (62) to toggle the automatic back-and-forth behavior.

FAILSAFE LOGIC

- **GPS Loss:** If GPS is lost, the mode cannot function. It will trigger a standard EKF Failsafe (Land/AltHold).
- **Boundary:** There are no hard geofences specific to ZigZag, but the standard Fence is active.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





KEY PARAMETERS

PARAMETER	DEFAULT	DESCRIPTION
`ZIGZAG_SIDE_DIST`	0	(m) Distance to step sideways after each pass. 0 = Manual switching only.
`ZIGZAG_DIRECTION`	0	0=Forward/Right, 1=Forward/Left... Defines the step direction.
`WPNAV_SPEED`	500	Speed of the auto-line flight.

SOURCE CODE REFERENCE

- **Mode Logic:** `ardupilot/ArduCopter/mode_zigzag.cpp`
- **Point Saving:** `ModeZigZag::save_or_move_to_destination()`

ACRO MODE (PLANE)

EXECUTIVE SUMMARY

ACRO Mode for fixed-wing aircraft provides a rate-stabilized flight experience. Unlike "Manual" mode, where the sticks directly move the servos, ACRO mode uses the gyroscope to maintain the commanded rotation rate. This makes the aircraft feel "locked in," smoothing out turbulence and wind gusts automatically.

THEORY & CONCEPTS

1. THE STABILITY VS. NEUTRALITY SPECTRUM

Flight modes exist on a spectrum of control authority.

- **Manual:** Neutral. The plane does exactly what the servos say. If wind hits a wing, the plane rolls.
- **Stabilize:** Stable. The autopilot constantly "pulls" the plane back to the horizon.
- **Acro:** Semi-Neutral. The autopilot uses the gyro to **lock** the current rotation rate. If you are in a 30-degree bank and a gust hits you, the autopilot moves the servos to *resist* that gust, keeping you at exactly 30 degrees.
- **The Benefit:** Acro is the most efficient way to fly because you don't fight the autopilot's desire to level out, which saves battery and allows for perfect cinematic curves.

HARDWARE DEPENDENCY MATRIX

ACRO mode relies on the rate gyros to function.

SENSOR	REQUIREMENT	CODE IMPLEMENTATION NOTES
Gyroscope	CRITICAL	The core of the Rate Controller. Without it, stabilization is impossible.
Airspeed	**RECOMMENDED**	While not strictly required, airspeed data helps scale the PID gains (scaling control surface deflection based on speed) for consistent handling.
GPS	**NONE**	Not used.

CONTROL ARCHITECTURE (ENGINEER'S VIEW)

Plane ACRO is significantly more complex than Copter ACRO because it supports three distinct sub-behaviors defined by `ACRO_LOCKING`.

1. **Rate Only (`ACRO_LOCKING` = 0):**
   - **Behavior:** Sticks command Rate (deg/s). Center stick command 0 deg/s.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- *Effect:* The plane stops rotating when you let go, but it does **not** hold the angle. It will slowly drift due to trim/gravity.

2. **Attitude Lock (`ACRO_LOCKING` = 1):**
   - **Behavior:** When sticks are centered, the controller "locks" the current Roll/Pitch angle.
   - *Effect:* If you bank 30 degrees and let go, the plane fights to hold exactly 30 degrees.
3. **Quaternion Lock (`ACRO_LOCKING` = 2):**
   - **Behavior:** Uses a full 3D Quaternion controller.
   - *Effect:* This allows "3D Heading Hold". You can point the nose straight up (Vertical Hover) and let go, and it will try to hold that vertical attitude. Standard Attitude Lock fails at 90 degrees pitch (Gimbal Lock); Quaternion does not.

PILOT INTERACTION

- **Sticks Active:** You are commanding the rotation speed of the aircraft.
- **Sticks Centered:**
  - *Rate Mode:* The plane stops rotating but may drift.
  - *Lock Mode:* The plane holds its current attitude "like it's on rails."
- **Difference from Manual:** In Manual, if a gust hits the wing, the plane rolls uncommanded. In ACRO, the servo will instantly move against the gust to keep the plane steady.

FAILSAFE LOGIC

- **Stall Prevention:** ACRO mode does **not** natively prevent stalls. It will happily try to hold a pitch angle that results in a stall. However, if `STALL_PREVENTION` is enabled, the controller may limit elevator output at low speeds (if an airspeed sensor is present).

KEY PARAMETERS

PARAMETER	DEFAULT	DESCRIPTION
`ACRO_ROLL_RATE`	180	(deg/s) Max roll rate at full stick.
`ACRO_PITCH_RATE`	180	(deg/s) Max pitch rate.
`ACRO_LOCKING`	0	0=Rate Only, 1=Attitude Lock, 2=Quaternion (3D) Lock.
`RLL_RATE_P`	...	Roll Rate PID gains (Main tuning knob for responsiveness).

SOURCE CODE REFERENCE

- **Mode Logic:** `ardupilot/ArduPlane/mode_acro.cpp`
- **Control Loop:** `ModeAcro::run()`

AUTOTUNE MODE (PLANE)

EXECUTIVE SUMMARY

AUTOTUNE Mode is a utility mode designed to automatically calibrate the Roll, Pitch, and Yaw PID controllers while you fly. It removes the guesswork from tuning a fixed-wing aircraft. The pilot flies the plane normally (it feels like **FBWA**), and the autopilot constantly analyzes the response to stick inputs, adjusting gains in real-time to achieve the desired response defined by `RLL2SRV_TCONST` and `PTCH2SRV_TCONST`.

THEORY & CONCEPTS

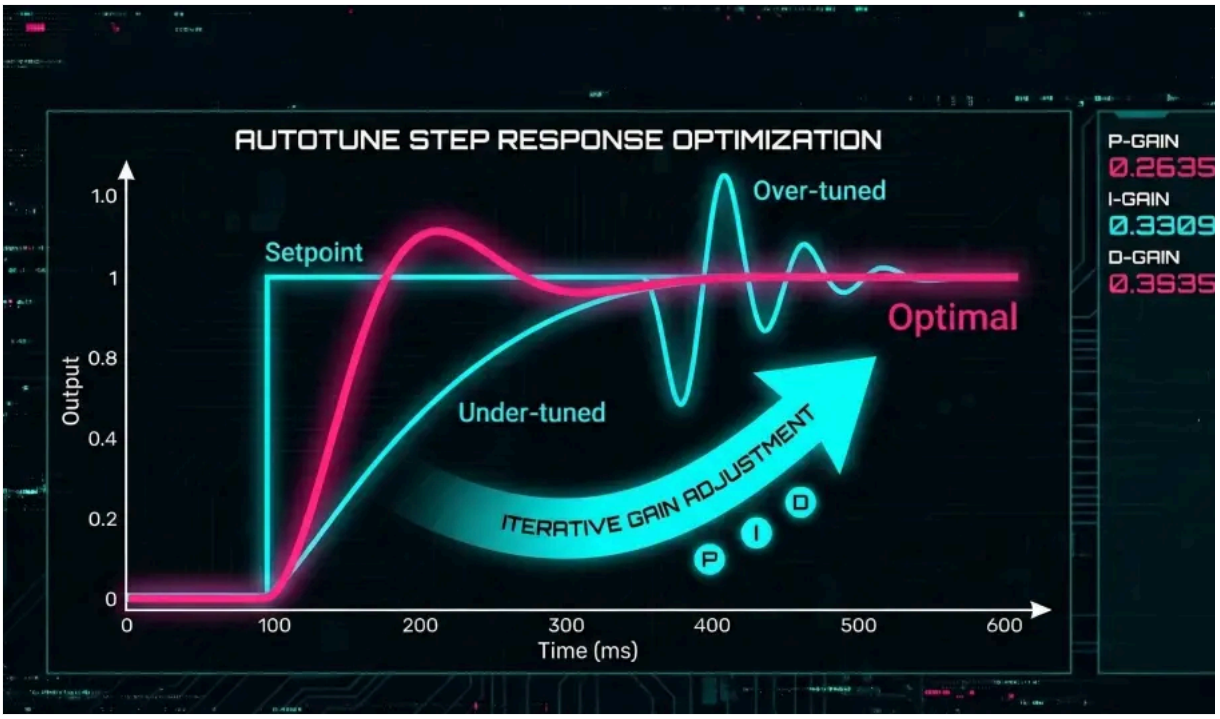
1. STEP-RESPONSE ANALYSIS

Autotune is a "Black Box" identification algorithm.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY







- **The Theory:** Every plane has a unique "Inertia" and "Control Authority." A heavy cargo plane needs more servo movement to roll than a small racer.
- **The Process:** When you hold the stick, the autopilot measures the **Step Response** (how long it takes for the plane to reach the target rate).
- **The Math:** It iteratively increases the P-Gain until it detects the slightest hint of oscillation, then backs off slightly. This gives you the "Tightest" possible tune without mechanical stress.

2. TIME CONSTANTS (TCONST)

Instead of asking you for "PID Gains," Autotune asks you for a **Response Goal** (`TCONST`).

- A `TCONST` of 0.5 means you want the plane to reach 63\% of its target angle within 0.5 seconds.
- *Why?* This is much more intuitive for a human. You describe how you want the plane to *feel*, and the computer does the math to find the gains.

HARDWARE DEPENDENCY MATRIX

The quality of the tune depends on the sensors used to measure the response.

SENSOR	REQUIREMENT	CODE IMPLEMENTATION NOTES
Gyroscope	CRITICAL	Measures the angular rate response. High noise levels will cause the tune to be conservative or fail.
Airspeed	**RECOMMENDED**	Highly recommended. Without airspeed, the tune is only valid for the speed at which you flew during the tune. With airspeed, the gains scale automatically across the flight envelope.
GPS	**NONE**	Not required.

CONTROL ARCHITECTURE (ENGINEER'S VIEW)

AUTOTUNE is built on top of **Fly-By-Wire A (FBWA)**.

1. **Flight Behavior:**
    - The pilot controls Bank Angle and Pitch Angle directly.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

- *Code Path:* `ModeAutoTune::update()` calls `plane.mode_fbwa.update()`. This confirms the flight characteristics are identical to FBWA.

2. **Tuning Logic:**
    - The tuning logic resides inside the PID controllers (`AP_RollController`, etc.).
    - **Trigger:** Tuning only happens when the pilot demands a **High Rotation Rate** (typically > 80\% stick deflection).
    - **Measurement:** The controller measures the delay and overshoot between the *Demanded Rate* and the *Actual Gyro Rate*.
    - **Adjustment:**
        - If response is sluggish: Increase P/D gains.
        - If response oscillates: Decrease P/D gains.
    - *Levels:* The tune iterates through levels (1 to roughly 6-8) until it finds the maximum safe gains.

PILOT INTERACTION

- **Engagement:** Switch to AUTOTUNE mode.
- **The Procedure:**
    1. Fly to a safe altitude.
    2. Roll hard left, then hard right. Repeat until the roll response feels crisp.
    3. Pitch up hard, then pitch down hard. Repeat.
    4. (Optional) Use rudder to tune Yaw.
- **Saving:**
    - To **SAVE** the new gains: Switch out of AUTOTUNE while the sticks are centered.
    - To **DISCARD** the gains: Switch out of AUTOTUNE while holding the sticks (not centered), OR simply reboot the board without switching modes.

FAILSAFE LOGIC

- **Bad Tune:** If the plane becomes unstable during the tune, simply switch back to **Stabilize** or **Manual**. The gains are not permanently saved until you disarm or switch modes safely.
- **Reversion:** ArduPilot keeps a copy of the original gains. If you toggle the switch correctly, it reverts instantly.

KEY PARAMETERS

PARAMETER	DEFAULT	DESCRIPTION
`AUTOTUNE_LEVEL`	6	Aggressiveness of the tune (6 is standard). Lower = softer tune.
`RLL2SRV_TCONST`	0.5	(s) Target Time Constant for Roll. Lower = snappier roll.
`PTCH2SRV_TCONST`	0.5	(s) Target Time Constant for Pitch.
`RLL2SRV_RMAX`	0	Max Roll Rate (deg/s). 0 = default (usually 60-75).

SOURCE CODE REFERENCE

- **Mode Wrapper:** `ardupilot/ArduPlane/mode_autotune.cpp`
- **Start Logic:** `Plane::autotune_start()`

AUTO MODE (PLANE)

EXECUTIVE SUMMARY

AUTO mode is the primary autonomous flight mode for fixed-wing aircraft. It executes a pre-planned mission uploaded to the flight controller. Unlike multicopters which can stop at waypoints, fixed-wing AUTO mode focuses on continuous path following, using banking turns and energy management to navigate efficiently.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





THEORY & CONCEPTS

1. TOTAL ENERGY CONTROL (TECS)

Autonomous fixed-wing flight is an energy-management problem.

- **The Physics:** Altitude is **Potential Energy**. Airspeed is **Kinetic Energy**.
- **The Controller (TECS):** Instead of controlling Pitch and Throttle separately, ArduPilot treats them as two sides of the same coin.
  - *To gain Energy:* Increase Throttle.
  - *To distribute Energy:* Use the Elevator to trade Speed for Height (or vice-versa).
- **The Benefit:** This prevents the "Phugoid Oscillation" (roller-coastering) where a plane over-climbs, loses speed, dives to gain speed, and repeats.

2. THE L1 NAVIGATION ALGORITHM

How do you make a plane fly a perfectly straight line in a crosswind?

- **The Concept:** L1 is a "non-linear" guidance law. It places a virtual point on the path ahead of the plane and calculates the lateral acceleration required to reach that point.
- **The Crabbing:** Because it follows the *ground track*, the plane will automatically "crab" (point its nose into the wind) to ensure the path over the ground is a straight line.

HARDWARE DEPENDENCY MATRIX

Fixed-wing autonomous flight has specific requirements for airspeed and attitude estimation.

SENSOR	REQUIREMENT	CODE IMPLEMENTATION NOTES
GPS	CRITICAL	Required for position and ground speed estimation.
Compass	CRITICAL	Required for heading. Unlike Copter, planes cannot simply "yaw" to fix heading; they must bank. Accurate heading is essential for L1 navigation.
Airspeed	**RECOMMENDED**	While synthetic airspeed (calculated from GPS/IMU) works, a physical pitot tube drastically improves **TECS** (Throttle/Pitch) performance, preventing stalls during climbs or turns.
Barometer	REQUIRED	Primary source for altitude control.

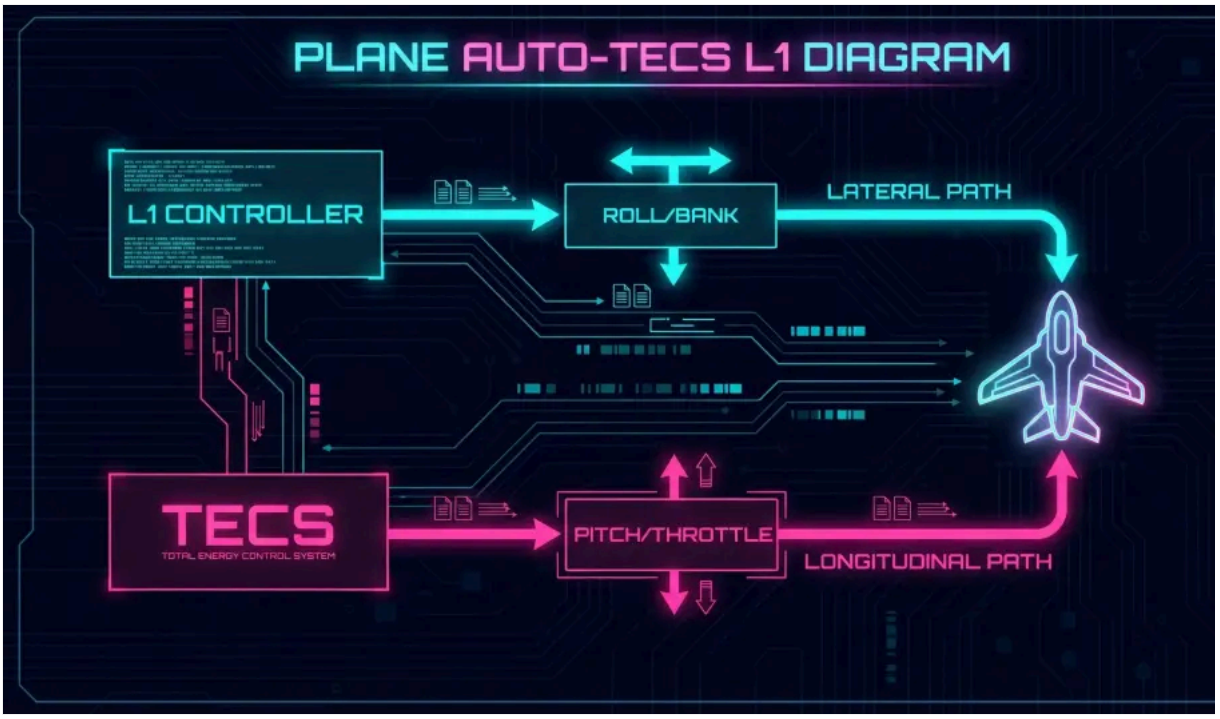
CONTROL ARCHITECTURE (ENGINEER'S VIEW)

Plane AUTO mode relies on two major controllers working in parallel:

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY







1. **Lateral Control (L1 Controller):**
   - **Goal:** Follow the line between two waypoints.
   - **Mechanism:** It calculates a "Lateral Acceleration" required to get back on the line. It converts this into a **Bank Angle**.
   - *Behavior:* It accounts for ground speed and turn radius. It will "cut corners" (turn early) at waypoints to maintain momentum.
   - *Reference:* `AP_L1_Control` library.
2. **Longitudinal Control (TECS):**
   - **Goal:** Manage Altitude and Airspeed.
   - **Mechanism:** Total Energy Control System (TECS). It trades Kinetic Energy (Speed) for Potential Energy (Altitude).
   - *Logic:*
     - To go UP and FAST: Increase Throttle.
     - To go DOWN and SLOW: Decrease Throttle + Pitch Up (drag).
     - To go UP and SLOW: Pitch Up (trade speed for height).

PILOT INTERACTION

In AUTO, the pilot is usually a passenger, but **Stick Mixing** allows intervention.

- **Stick Mixing (`STICK_MIXING` = 1):**
  - If you move the sticks, your input is **added** to the autopilot's command.
  - *Example:* If the plane is banking right to follow a waypoint, and you hold Left Stick, you can cancel out the bank or force it left.
  - *Danger:* You are fighting the autopilot. When you let go, it snaps back to the mission path.
- **Throttle:** Usually ignored (controlled by TECS), unless `THR_PASS_STAB` is enabled or specific parameters allow nudging.

FAILSAFE LOGIC

- **GPS Loss:** The L1 controller fails. The plane will likely switch to a non-GPS mode (like Stabilize or AltHold) or trigger a "Dead Reckoning" failsafe.
- **Stall Prevention:** Even in AUTO, the stall prevention logic is active. If airspeed drops too low, the controller will pitch down to regain speed, even if it means losing altitude (violating the waypoint altitude target).

KEY PARAMETERS

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

PARAMETER	DEFAULT	DESCRIPTION
NAVL1_PERIOD	20	(s) Aggressiveness of the L1 controller. Lower = tighter tracking (more weaving). Higher = smoother (sloppier).
TECS_CLMB_MAX	5	(m/s) Max climb rate allowed.
MIN_GNDSPD_CM	0	Minimum ground speed. Prevents the plane from stopping in high winds (flying backwards relative to ground).
STICK_MIXING	1	0=Disabled (Pilot locked out), 1=Enabled (Fly-Through).

SOURCE CODE REFERENCE

- **Mode Logic:** `ardupilot/ArduPlane/mode_auto.cpp`
- **Stick Mixing:** `Plane::stabilize_stick_mixing_fbw()`

CIRCLE MODE (PLANE)

EXECUTIVE SUMMARY

CIRCLE mode is a simple, non-GPS flight mode that forces the aircraft to fly in a circle at a fixed bank angle while maintaining its current altitude. Unlike **LOITER** mode, it does not attempt to hold a specific geographic position. It is often used as a failsafe behavior or a simple "park" mode when GPS is unreliable.

THEORY & CONCEPTS

1. GROUND TRACK VS. AIR TRACK

In fixed-wing flight, wind is a massive factor.

- **Air Track:** If you just hold a fixed bank angle (Manual), you will fly in a circle *relative to the air*. If there is a 20 m/s wind, your circle will drift 20 meters every second relative to the ground.
- **The Circle Mode (Dumb):** Plane CIRCLE mode is a "Fixed Bank" mode. It is useful because it doesn't require a GPS, but it is not a "Position Hold." It is a way to stay in a general area while descending or waiting for a signal.

2. CENTRIPETAL FORCE AND LIFT

When a plane banks, some of its lift is diverted to pull the plane into a turn (Centripetal Force).

- **The Problem:** Diversion of lift means there is less vertical lift to fight gravity.
- **The Autopilot:** ArduPilot's CIRCLE mode automatically increases the **Pitch Angle** and **Throttle** to compensate for this lost lift, ensuring the plane stays at its target altitude throughout the orbit.

HARDWARE DEPENDENCY MATRIX

This mode is designed to work with minimal sensors.

SENSOR	REQUIREMENT	CODE IMPLEMENTATION NOTES
GPS	**NONE**	Not used. The circle will drift with the wind.
Gyro/Accel	CRITICAL	Required for bank angle stabilization.
Barometer	REQUIRED	Required for altitude hold.

CONTROL ARCHITECTURE (ENGINEER'S VIEW)

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





The logic is extremely simple compared to Loiter.

1. **Bank Angle:**
   - The target Roll is set to `ROLL_LIMIT / 3`.
   - *Example:* If your `ROLL_LIMIT_DEG` is 45, the plane banks at 15 degrees.
   - *Direction:* Always clockwise by default (positive roll).
   - *Code Path:* `ModeCircle::update()` sets `nav_roll_cd`.
2. **Pitch/Throttle:**
   - Uses the standard Altitude Hold controller (`calc_nav_pitch` / `calc_throttle`).
   - It attempts to hold the altitude recorded when the mode was engaged.

COMPARISON: CIRCLE VS LOITER

- **CIRCLE:** "dumb" bank. Drifts with wind. No GPS required. Fixed radius (determined by airspeed and bank angle).
- **LOITER:** "smart" navigation. Holds 3D GPS coordinate. Adjusts bank angle to maintain a perfect circle over the ground regardless of wind.

FAILSAFE LOGIC

- **Long Range Failsafe:** CIRCLE is often used as the "Short Failsafe" action because it keeps the plane local without requiring a complex return path logic immediately.

KEY PARAMETERS

PARAMETER	DEFAULT	DESCRIPTION
`ROLL_LIMIT_DEG`	45	The bank angle in CIRCLE mode is derived from this (Limit / 3).

SOURCE CODE REFERENCE

- **Mode Logic:** `ardupilot/ArduPlane/mode_circle.cpp`

CRUISE MODE (PLANE)

EXECUTIVE SUMMARY

CRUISE Mode is the long-range pilot's best friend. It acts like **Fly-By-Wire B (FBWB)** for altitude and speed control but adds a **Ground Course Lock**. When you release the aileron stick, the plane locks onto its current compass heading and flies a perfectly straight line, correcting for wind drift.

THEORY & CONCEPTS

1. THE HEADING-HOLD LOGIC

CRUISE mode is essentially "Autopilot with a Compass."

- **The Problem:** In manual flight, you are constantly making small corrections to stay on path.
- **The Solution:** The Heading-Hold controller. When you let go of the aileron stick, the EKF captures the current **GPS Ground Course**. It then uses the L1 controller to build a virtual path along that course.
- **The Benefit:** This is the most efficient way to fly long distances. You can literally let go of the sticks for 10 miles and the plane will stay on its track.

2. RELATIVE AIRSPEED MANAGEMENT

In CRUISE, your throttle controls **Airspeed**, not motor power.

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- **Physics:** If you are flying into a 10 m/s headwind, your ground speed will be slow. If you turn and fly downwind, your ground speed will be fast.
- **Autopilot:** The plane manages the throttle to keep the *Airspeed* constant. This is safer because it ensures you always have enough lift over the wings to stay in the air, regardless of the wind.

HARDWARE DEPENDENCY MATRIX

Unlike FBWA, CRUISE is a navigation mode.

SENSOR	REQUIREMENT	CODE IMPLEMENTATION NOTES
GPS	CRITICAL	Required for Ground Course tracking. Without GPS, it cannot correct for wind drift or calculate the L1 trajectory.
Compass	CRITICAL	Required for heading.
Airspeed	**RECOMMENDED**	Improves the throttle/pitch management (TECS) significantly.

CONTROL ARCHITECTURE (ENGINEER'S VIEW)

CRUISE is a hybrid controller merging FBWB and L1 Navigation.

1. **Heading Lock Logic:**
   - *State:* When the pilot releases the Roll/Rudder sticks for > 0.5s.
   - *Action:* The code projects a **Virtual Waypoint** 1km ahead of the vehicle along its current Ground Course vector.
   - *Tracking:* It engages the **L1 Controller** to fly to that waypoint. This ensures the plane flies a "Track", not just a "Heading" (it crabs into the wind to fly straight over the ground).
   - *Code Path:* `ModeCruise::navigate()`.
2. **Altitude & Speed Logic (TECS):**
   - Identical to **FBWB**.
   - **Elevator Stick:** Commands Climb/Descent Rate (m/s).
   - **Throttle Stick:** Commands Target Airspeed (m/s).
   - *Mechanism:* The TECS controller manages pitch and throttle to achieve these targets.

PILOT INTERACTION

- **Turning:** Simply roll the plane with the aileron stick. The Heading Lock disengages, and the plane flies in "FBWA" roll mode (Angle control).
- **Locking:** Release the stick. After 0.5 seconds, the new heading is captured, and the lock re-engages.
- **Climbing/Descending:** Pull back or push forward. The plane climbs/descends at a controlled rate while maintaining speed and heading.

FAILSAFE LOGIC

- **GPS Loss:** If GPS is lost, the L1 controller fails. The mode effectively degrades to FBWB (holding attitude but drifting with wind).

KEY PARAMETERS

PARAMETER	DEFAULT	DESCRIPTION
`FBWB_CLIMB_RATE`	2.0	(m/s) Climb rate at full back stick.
`ARSPD_FBW_MIN`	9	(m/s) Minimum target airspeed (throttle stick down).
`ARSPD_FBW_MAX`	22	(m/s) Maximum target airspeed (throttle stick up).

SOURCE CODE REFERENCE

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- **Mode Logic:** `ardupilot/ArduPlane/mode_cruise.cpp`
- **Update Loop:** `ModeCruise::update()`

FLY BY WIRE A (PLANE)

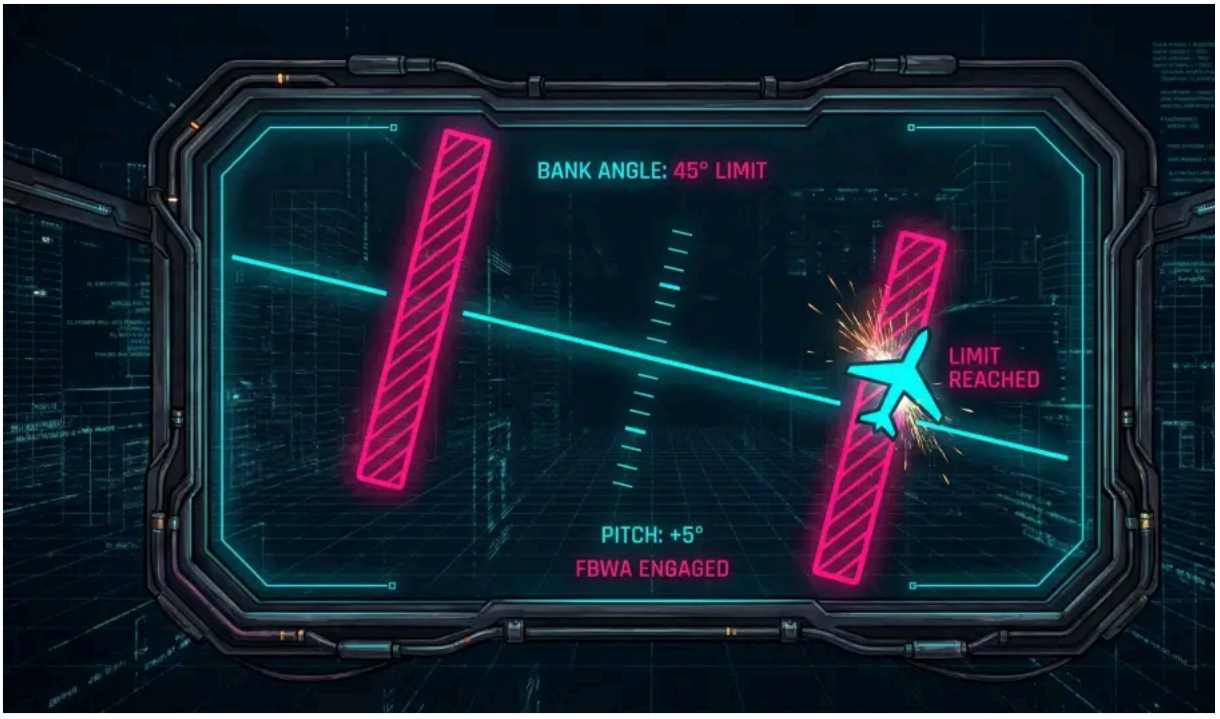
EXECUTIVE SUMMARY

Fly By Wire A (FBWA) is the most popular stabilized flight mode for fixed-wing aircraft. It provides "Angle Control" for Roll and Pitch, meaning the stick position corresponds directly to the aircraft's bank/pitch angle. It is heavily safety-limited, preventing the pilot from rolling inverted or stalling the aircraft (if configured).

THEORY & CONCEPTS

1. FLY-BY-WIRE ARCHITECTURE

In a standard plane, the cables move the surfaces. In a Fly-By-Wire system, the computer interprets your request and moves the surfaces for you.



- **Angle Limiting:** The autopilot knows the **Critical Bank Angle** where a plane will lose lift. It forbids you from exceeding that angle, making it impossible to "Death Spiral" the aircraft in a turn.
- **Self-Leveling:** It is a "Low Pass Filter" for human error. If you jerk the stick, the autopilot smoothens the move. If you let go, it returns to center.

2. STALL PREVENTION THEORY

A stall happens when the **Angle of Attack** (the angle between the wing and the oncoming air) is too high.

- **The Problem:** Most pilots pull "UP" on the elevator when they are scared. This increases the Angle of Attack and *causes* the stall.
- **The Solution:** ArduPilot's FBWA monitors the airspeed. If you are flying too slow, it **limits your elevator authority**. You can pull full back, and the plane will only pitch up a few degrees, forcing it to keep enough speed to stay in the air.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

HARDWARE DEPENDENCY MATRIX

FBWA requires accurate attitude estimation.

SENSOR	REQUIREMENT	CODE IMPLEMENTATION NOTES
Gyroscope	CRITICAL	Required for rate stabilization.
Accelerometer	CRITICAL	Required to determine "Level" and apply bank/pitch limits.
Airspeed	**RECOMMENDED**	Critical for **Stall Prevention**. Without airspeed, the controller cannot know if a high pitch angle is safe.
GPS	**NONE**	Not used.

CONTROL ARCHITECTURE (ENGINEER'S VIEW)

FBWA operates as an **Angle Controller** with hard limits.

1. **Angle Mapping:**
   - **Roll Stick:** Maps to Bank Angle (`ROLL_LIMIT`).
     - Full Left = `-ROLL_LIMIT` (e.g., -45 deg).
     - Center = 0 deg (Level).
   - **Pitch Stick:** Maps to Pitch Angle.
     - Full Back = `PITCH_LIMIT_MAX` (e.g., +20 deg).
     - Full Forward = `PITCH_LIMIT_MIN` (e.g., -25 deg).
   - *Self-Leveling:* Because center stick maps to 0 degrees, the plane automatically levels itself when you let go.
2. **Throttle Passthrough:**
   - Throttle is manual but constrained.
   - Range is clamped between `THR_MIN` and `THR_MAX`.
3. **Stall Prevention:**
   - If `STALL_PREVENTION` is enabled and an Airspeed sensor is present:
   - If airspeed drops below safe limits, the controller **overrides** the pilot's pitch input to lower the nose, maintaining flying speed. This happens even if you hold full back stick.

PILOT INTERACTION

- **Turning:** Hold the aileron stick to the side. The plane banks to `ROLL_LIMIT` and turns. To stop turning, center the stick.
  - *Note:* You may need to pull back on the elevator to maintain altitude in a steep turn, just like a real plane.
- **Climbing:** Pull back. The plane climbs at a fixed angle. It does *not* maintain a specific climb rate (m/s), just a specific nose-up angle.

FAILSAFE LOGIC

- **Attitude Loss:** If the AHRS fails (bad accel calibration or vibration), FBWA cannot determine level and may roll the plane upside down thinking it is flat.
- **Sensor Failure:** If the Airspeed sensor fails, Stall Prevention may misbehave.

KEY PARAMETERS

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





PARAMETER	DEFAULT	DESCRIPTION
`ROLL_LIMIT_DEG`	45	Max bank angle (degrees).
`PITCH_LIMIT_MAX`	20	Max pitch up angle (degrees).
`PITCH_LIMIT_MIN`	-25	Max pitch down angle (degrees).
`STALL_PREVENTION`	1	0=Disabled, 1=Enabled.
`RLL2SRV_P`	...	Roll Angle P-gain. Determines how aggressively it seeks the target angle.

SOURCE CODE REFERENCE

- **Mode Logic:** `ardupilot/ArduPlane/mode_fbwa.cpp`
- **Update Loop:** `ModeFBWA::update()`

FLY BY WIRE B (PLANE)

EXECUTIVE SUMMARY

Fly By Wire B (FBWB) is an advanced assisted flight mode that builds upon FBWA. While FBWA stabilizes attitude, FBWB stabilizes **Altitude** and **Airspeed**. It feels like flying a plane with cruise control and an autopilot-managed throttle.

THEORY & CONCEPTS

1. ENERGY CONSERVATION IN GLIDES

FBWB is an "Energy Management" mode.

- **The Physics:** Every plane has a "Best Glide" speed (`ARSPD_CRUISE`).
- **The Logic:** If you pull back on the elevator in FBWB, the plane climbs. To climb, it needs energy. The TECS controller will either increase the throttle (adding energy) or decrease the airspeed (trading kinetic for potential energy).
- **The Benefit:** You don't have to worry about the throttle. You just point the plane at the altitude you want, and the computer manages the "Engine" to get you there safely.

2. TERRAIN AWARENESS

Since FBWB is an altitude-locked mode, it is the safest mode for low-altitude cruising.

- If you fly over a hill, the plane will maintain its **Height Relative to Takeoff**.
- *Advanced:* If you have a Rangefinder or Terrain Database enabled, FBWB can automatically "Terrain Follow," keeping a constant distance from the dirt rather than a constant distance from the ocean.

HARDWARE DEPENDENCY MATRIX

FBWB relies on the energy management system (TECS).

SENSOR	REQUIREMENT	CODE IMPLEMENTATION NOTES
Airspeed	**RECOMMENDED**	While FBWB can fly with synthetic airspeed, a real Pitot tube provides the TECS controller with accurate energy data, preventing stalls and overspeeds much more reliably.
Barometer	`REQUIRED`	Required for altitude hold.
GPS	`REQUIRED`	Required for ground speed estimation (if no airspeed sensor) and position data.

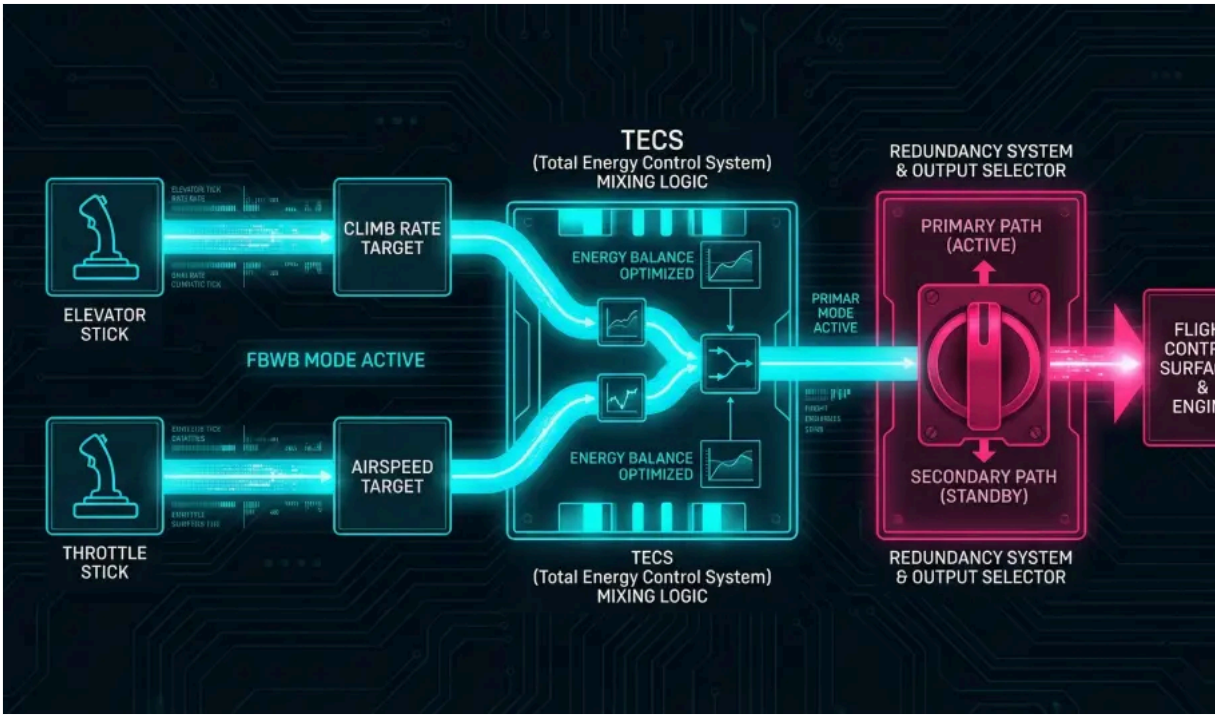
Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





CONTROL ARCHITECTURE (ENGINEER'S VIEW)

FBWB delegates pitch and throttle control to the **Total Energy Control System (TECS)**.



1. **Elevator Logic (Climb Rate):**
   - Unlike FBWA (where Elevator = Pitch Angle), in FBWB, **Elevator = Target Climb Rate**.
   - *Center Stick:* Maintain current altitude.
   - *Pull Back:* Climb at `FBWB_CLIMB_RATE` (m/s).
   - *Push Forward:* Descend.
   - *Code Path:* `Plane::update_fbwb_speed_height()`.
2. **Throttle Logic (Airspeed):**
   - The throttle stick does **not** control the motor directly. It sets the **Target Airspeed**.
   - *Stick Low:* Target `ARSPD_FBW_MIN`.
   - *Stick High:* Target `ARSPD_FBW_MAX`.
3. **TECS Mixer:**
   - The TECS controller calculates the necessary **Pitch Angle** and **Throttle \%** to achieve the requested Climb Rate and Airspeed simultaneously.
   - *Example:* If you demand a climb but low speed, TECS will pitch up and cut throttle. If you demand a climb and high speed, TECS will pitch up and floor the throttle.

PILOT INTERACTION

- **Cruising:** Get to your desired altitude, release the elevator stick. The plane locks that altitude.
- **Turning:** Use ailerons to turn (same as FBWA). The controller automatically adds back-pressure (pitch up) to maintain altitude in the turn.
- **Wind:** Because the throttle manages *Airspeed*, the plane will automatically throttle up when flying upwind and throttle down when flying downwind to maintain consistent lift.

FAILSAFE LOGIC

- **Stall Prevention:** TECS prioritizes speed over altitude. If you try to climb too steep and run out of power, TECS will nose-down to prevent a stall, even if it means losing altitude.

KEY PARAMETERS

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

PARAMETER	DEFAULT	DESCRIPTION
`FBWB_CLIMB_RATE`	2.0	(m/s) Climb rate at full stick deflection.
`ARSPD_FBW_MIN`	9	(m/s) Min target airspeed (Stick Low).
`ARSPD_FBW_MAX`	22	(m/s) Max target airspeed (Stick High).
`FBWB_ELEV_REV`	0	Reverse elevator response in FBWB (Up stick = Down).

SOURCE CODE REFERENCE

- **Mode Logic:** `ardupilot/ArduPlane/mode_fbwb.cpp`
- **Input Mapping:** `Plane::update_fbwb_speed_height()`

GUIDED MODE (PLANE)

EXECUTIVE SUMMARY

GUIDED Mode is designed for "Offboard Control". It allows a Ground Control Station (GCS) or Companion Computer to command the aircraft dynamically. The most common use case is "Click to Fly," where the user clicks a point on the map, and the plane flies to that location and loiters. It also supports advanced "Attitude Target" commands for researchers testing custom control loops.

THEORY & CONCEPTS

1. OFFBOARD STATE MACHINE

GUIDED mode is the entry point for **Robotic Automation**.

- **The Concept:** Standard modes are for "Flying." GUIDED mode is for "Instruction."
- **Targeting:** You don't tell the plane *how* to fly; you tell it *where* you want it to be. The L1 and TECS controllers take over to solve the math of the flight path.

2. HIGH-LATENCY SAFE LOGIC

When a companion computer controls a drone, there is always **Latency** (delay).

- **The Hazard:** If the computer crashes or the WiFi drops, the drone shouldn't keep flying straight.
- **The Guard:** Guided mode includes a "Slew" mechanism. It will only accept changes within a certain rate. If no new instructions arrive, the plane reverts to its default behavior (usually Loitering) rather than continuing its last commanded vector into a mountain.

HARDWARE DEPENDENCY MATRIX

Guided mode is a navigation mode.

SENSOR	REQUIREMENT	CODE IMPLEMENTATION NOTES
GPS	CRITICAL	Required for all Position-based commands.
Compass	CRITICAL	Required for heading.
Telemetry	CRITICAL	A link to the GCS/Companion Computer is required to receive commands.

CONTROL ARCHITECTURE (ENGINEER'S VIEW)

Plane Guided mode is simpler than Copter Guided mode. It generally behaves like **Loiter** but with a dynamic target.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





1. **Waypoint Logic:**
    - When you set a target location (Position + Altitude), the controller generates a path to it.
    - Once it reaches the target, it enters a **Loiter** (Orbit) state.
    - *Code Path:* `ModeGuided::navigate()` calls `plane.update_loiter()`.
2. **Attitude Target (Research):**
    - If enabled, external software can stream `SET_ATTITUDE_TARGET` messages (Quaternion + Thrust).
    - The Plane bypasses its high-level navigation and attempts to match the requested Roll/Pitch/Yaw.
    - *Warning:* No safety limits are applied here. You can stall or crash if the external computer commands it.
3. **Velocity Control:**
    - **Limitation:** Unlike Copter, standard fixed-wing Plane Guided mode does **not** generally support direct Velocity Vector control (e.g., "Fly North at 10m/s"). It is Position-centric.

PILOT INTERACTION

- **Stick Mixing:** By default, Stick Mixing is **disabled** in Guided mode (`STICK_MIXING` logic). The pilot is a passenger.
- **Takeover:** To abort, switch flight modes (e.g., to FBWA or Manual).

FAILSAFE LOGIC

- **Data Link Loss:** If the GCS link is lost, the plane will continue to Loiter at the last received target indefinitely (unless a separate GCS Failsafe is triggered to force RTL).

KEY PARAMETERS

PARAMETER	DEFAULT	DESCRIPTION
`GUIDED_SLEW_SPD`	0	Max speed change rate.
`GUIDED_SLEW_ALT`	0	Max altitude change rate.

SOURCE CODE REFERENCE

- **Mode Logic:** `ardupilot/ArduPlane/mode_guided.cpp`
- **Update Loop:** `ModeGuided::update()`

LOITER MODE (PLANE)

EXECUTIVE SUMMARY

LOITER Mode is the standard GPS-position-hold mode for fixed-wing aircraft. When engaged, the plane captures its current location and altitude, then orbits that point indefinitely. It uses the L1 Navigation Controller to maintain a precise ground track, automatically banking into the wind to ensure the circle remains perfectly round and centered relative to the ground.

THEORY & CONCEPTS

1. THE GEOMETRY OF THE LOITER

A fixed-wing plane cannot stop in the air. To "Hold Position," it must fly in a circle.

- **The Problem:** In a 10 m/s wind, a circular flight path relative to the *air* looks like a "Slinky" relative to the *ground*.
- **The Solution:** The L1 controller. It constantly varies the bank angle. It banks **steeper** on the downwind leg (to turn faster) and **shallower** on the upwind leg (to turn slower).
- **The Result:** A perfect circular ground track.

2. TANGENTIAL EXIT

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





When a plane leaves a Loiter to fly to a new waypoint, it doesn't just turn.

- **The Math:** ArduPilot calculates the **Tangent Point** on the circle that aligns with the next path.
- **The Benefit:** Smooth, efficient transitions without awkward "wiggles" as the plane leaves the orbit.

HARDWARE DEPENDENCY MATRIX

LOITER is a navigation mode.

SENSOR	REQUIREMENT	CODE IMPLEMENTATION NOTES
GPS	CRITICAL	Required to define the center point and track ground speed/course. Without GPS, the L1 controller cannot function.
Compass	CRITICAL	Required for heading.
Barometer	REQUIRED	Primary source for altitude control.
Airspeed	**RECOMMENDED**	Improves TECS performance (altitude/speed management) during the turn.

CONTROL ARCHITECTURE (ENGINEER'S VIEW)

LOITER mode combines L1 Lateral Navigation with TECS Longitudinal Control.

1. **Entry Logic:**
   - On engagement, the code captures the **Current Location** (Lat/Lon) and sets it as the "Loiter Center".
   - It captures the **Current Altitude** as the target altitude.
   - *Code Path:* `ModeLoiter:: enter()`.
2. **Lateral Control (L1 Orbit):**
   - The L1 Controller calculates the bank angle required to stay on the circle's perimeter (`WP_LOITER_RAD`).
   - *Wind Compensation:* If there is a crosswind, the plane will "crab" (yaw into the wind) and bank steeper on the downwind leg to prevent being blown off course.
   - *Code Path:* `ModeLoiter::navigate()` calls `plane.update_loiter()`.
3. **Longitudinal Control (TECS):**
   - The plane manages throttle and pitch to maintain the target altitude and airspeed (`TRIM_ARSPD_CM`).

PILOT INTERACTION

- **Default:** The pilot has no control. The plane flies the circle automatically.
- **Alt Control Option:** If `FLIGHT_OPTIONS` bit `ENABLE_LOITER_ALT_CONTROL` is set, the pilot can use the **Elevator** stick to change altitude (Climb/Descend) and the **Throttle** stick to change target Airspeed, similar to **FBWB**.
   - *Note:* You cannot "nudge" the circle's center horizontally in standard Plane Loiter.

FAILSAFE LOGIC

- **GPS Loss:** The L1 controller will fail. The plane may enter a "Dead Reckoning" mode or drift. It is safer to switch to **Stabilize** or **CIRCLE** (non-GPS) if GPS is lost.

KEY PARAMETERS

PARAMETER	DEFAULT	DESCRIPTION
`WP_LOITER_RAD`	60	(m) Radius of the orbit. Positive = Clockwise, Negative = Counter-Clockwise.
`MIN_GNDSPD_CM`	0	Minimum ground speed to maintain even in strong headwinds.

SOURCE CODE REFERENCE

- **Mode Logic:** `ardupilot/ArduPlane/mode_loiter.cpp`

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





MANUAL MODE (PLANE)

EXECUTIVE SUMMARY

MANUAL Mode provides a direct pass-through connection between the pilot's RC sticks and the aircraft's control surfaces. It bypasses all stabilization, navigation, and fly-by-wire limiters. It is the only mode that allows the pilot to override the autopilot completely, making it critical for recovering from sensor failures or bad tunes.

THEORY & CONCEPTS

1. DIRECT CONTROL SURFACE AUTHORITY

In Manual mode, you are the autopilot.

- **The Physics:** Every degree of stick movement corresponds to a degree of servo movement (or a specific PWM change).
- **The Aerodynamics:** You must manage the **Stability Derivative** of the plane. If you bank too much without adding elevator, the nose will drop (Spiral instability). If you pull up too much, the wings will stall.

2. THE FALLBACK STATE

In computer science, this is the **Fail-Safe** or **Limp Mode**. By design, MANUAL mode is the simplest possible logic path. By minimizing the number of code lines between the RC receiver and the Servos, ArduPilot ensures that even if the CPU is under heavy load or the EKF has crashed, you can still steer the aircraft.

HARDWARE DEPENDENCY MATRIX

This mode works even if the autopilot brain is partially failing.

SENSOR	REQUIREMENT	CODE IMPLEMENTATION NOTES
RC Receiver	CRITICAL	Required to read stick inputs.
Servo Output	CRITICAL	Required to drive the surfaces.
Gyro/Accel/GPS	**NONE**	Not used. If the AHRS tumbles or the EKF blows up, MANUAL mode will still fly perfectly fine (assuming the pilot is skilled).

CONTROL ARCHITECTURE (ENGINEER'S VIEW)

The logic is a simple assignment loop.

1. **Input Conditioning:**
   - Pilot inputs are read and scaled (-1 to +1).
   - **Expo:** ArduPlane applies specific exponential curves defined by `MAN_EXPO_ROLL`, `MAN_EXPO_PITCH`, and `MAN_EXPO_RUDDER`.
   - *Note:* This allows you to soften the feel of the manual plane without changing the settings on your transmitter.
2. **Output Mapping:**
   - The conditioned input is sent directly to the `SRV_Channels` library (`k_aileron`, `k_elevator`, `k_throttle`, `k_rudder`).
   - *Mixing:* The Servo Output library handles the physical mixing (e.g., V-Tail, Elevon, Flaperon) downstream of the mode logic. So even in MANUAL, your V-Tail mixer works correctly.
   - *Code Path:* `ModeManual::update()`.
3. **Controller Reset:**
   - The code explicitly calls `reset_controllers()` to ensure that when you switch *back* to a stabilized mode, the integrators (I-terms) are zeroed out, preventing a sudden "jump" or wind-up release.

PILOT INTERACTION

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- **Total Authority:** You can stall, spin, or over-stress the airframe. The autopilot will not stop you.
- **Trimming:** You must mechanically trim the aircraft or use the transmitter trims. (Note: Transmitter trims should be reset/zeroed before switching back to FBWA/stabilized modes to avoid confusing the controller).

FAILSAFE LOGIC

- **None:** MANUAL mode *is* the failsafe.

KEY PARAMETERS

PARAMETER	DEFAULT	DESCRIPTION
`MAN_EXPO_ROLL`	0	(0-100\%) Exponential curve for Roll stick.
`MAN_EXPO_PITCH`	0	(0-100\%) Exponential curve for Pitch stick.
`MAN_EXPO_RUDDER`	0	(0-100\%) Exponential curve for Rudder stick.

SOURCE CODE REFERENCE

- **Mode Logic:** `ardupilot/ArduPlane/mode_manual.cpp`
- **Pass-Through:** `ModeManual::run()`

RTL MODE (PLANE)

EXECUTIVE SUMMARY

Return to Launch (RTL) is the primary autonomous safety mode for fixed-wing aircraft. When triggered (by a switch or failsafe), the plane navigates back to the Home position (or the nearest Rally Point), climbs to a safe altitude, and loiters indefinitely until the pilot regains control or the battery fails.

THEORY & CONCEPTS

1. FAILSAFE LOGIC CHAIN

RTL is often the end of a **Decision Tree**.

- **Trigger 1:** Radio link lost (GCS Failsafe).
- **Trigger 2:** Low Battery.
- **Trigger 3:** Geofence Breach.
- **The Reaction:** ArduPilot chooses the safest way back. If "Smart RTL" has a valid path, it uses that (to avoid obstacles). If not, it uses standard RTL.

2. THE RETURN ALTITUDE PROFILE

Planes don't climb like elevators.

- **The Geometry:** To climb 100m, a plane must fly forward ~500m.
- **The Logic:** ArduPilot doesn't wait to reach the altitude before turning. It turns *towards* home and climbs *while* flying. This saves battery but requires the pilot to ensure `RTL_ALTITUDE` is high enough to clear local obstacles along the entire path.

HARDWARE DEPENDENCY MATRIX

RTL is an autonomous navigation mode.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





SENSOR	REQUIREMENT	CODE IMPLEMENTATION NOTES
GPS	CRITICAL	Required for position tracking and ground speed control. If GPS is lost, RTL cannot function (Mode Change Failure or degradation to Circle).
Compass	CRITICAL	Required for heading control.
Barometer	REQUIRED	Primary source for altitude.
Airspeed	**RECOMMENDED**	Prevents stall during the potentially aggressive climb-and-turn maneuver.

CONTROL ARCHITECTURE (ENGINEER'S VIEW)

RTL operates as a variation of **Loiter** but with a specific destination and altitude profile.

1. **Destination Selection:**
   - On entry, it calculates the "Best Return Location".
   - *Default:* Home Position (where it armed).
   - *Rally:* If Rally Points are configured (`RALLY_LIMIT_KM`), it selects the *closest* valid Rally Point instead of Home.
   - *Code Path:* `ModeRTL::_enter()`.
2. **Climb vs Turn:**
   - The plane attempts to turn towards home and climb simultaneously.
   - *Safety:* If `RTL_CLIMB_MIN` is set, the bank angle is limited until the plane has climbed at least that many meters. This prevents the "Death Spiral" where a low plane banks hard, loses more altitude, and crashes.
3. **Arrival:**
   - The plane flies a straight line to the destination.
   - Upon arrival, it enters a **Loiter** (Orbit) at `RTL_RADIUS`.
   - It maintains `RTL_ALTITUDE`.

PILOT INTERACTION

- **Locked Out:** Pilot stick input is ignored by default.
- **Override:** You must switch flight modes (e.g., to FBWA or Manual) to regain control. Nudging is generally not supported in RTL.

FAILSAFE LOGIC

- **RTL Autoland:** If `RTL_AUTOLAND` is enabled, the plane will *not* loiter indefinitely. Instead, it will traverse to the nearest `DO_LAND_START` mission item and execute an automatic landing sequence. This is critical for BVLOS operations to save the airframe if the link is never regained.

KEY PARAMETERS

PARAMETER	DEFAULT	DESCRIPTION
`RTL_ALTITUDE`	100	(m) Target altitude. -1 = Maintain current altitude.
`RTL_RADIUS`	0	(m) Loiter radius at home. 0 = Use `WP_LOITER_RAD`.
`RTL_CLIMB_MIN`	0	(m) Altitude to climb before allowing full banking.
`RTL_AUTOLAND`	0	0=Disabled (Loiter forever), 1=Fly Home then Land, 2=Go Straight to Landing Sequence.

SOURCE CODE REFERENCE

- **Mode Logic:** `ardupilot/ArduPlane/mode_rtl.cpp`
- **Navigation:** `ModeRTL::navigate()`

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





STABILIZE MODE (PLANE)

EXECUTIVE SUMMARY

STABILIZE Mode is the most basic assisted flight mode for fixed-wing aircraft. It automatically levels the wings and pitch when the pilot releases the sticks. However, unlike FBWA, it does **not** enforce bank or pitch angle limits. The pilot has full authority to roll the aircraft upside down, but the autopilot will constantly fight to return it to level flight.

THEORY & CONCEPTS

1. STABILITY DERIVATIVE MANAGEMENT

In aerospace engineering, **Stability Derivatives** describe how a plane naturally reacts to disturbances.

- **A "Stable" Plane:** If a gust hits the wing and rolls it, the plane's own shape (dihedral) will eventually roll it back.
- **The Problem:** Natural stability is slow.
- **The Stabilize Mode:** It uses the gyro to detect the gust *before* the plane rolls and applies immediate counter-servo movement. It effectively "increases" the natural stability of the aircraft.

2. MANUAL AUTHORITY OVER STABILITY

STABILIZE is a "Hybrid" mode.

- The autopilot works to keep the plane level (Stability).
- The pilot adds a direct servo offset (Authority).
- *Result:* You can loop the plane by holding full elevator. The autopilot fights you the whole time, but your stick "outvotes" the autopilot. This provides a safe way to fly aerobatics without the fear of the plane staying upside down when you let go.

HARDWARE DEPENDENCY MATRIX

Stabilize requires only the core inertial sensors.

SENSOR	REQUIREMENT	CODE IMPLEMENTATION NOTES
Gyroscope	CRITICAL	Required for rate stabilization.
Accelerometer	CRITICAL	Required to determine "Level".
GPS	**NONE**	Not used.
Airspeed	**RECOMMENDED**	Helps scale PID gains for better performance at different speeds, but not strictly required.

CONTROL ARCHITECTURE (ENGINEER'S VIEW)

Stabilize operates differently from FBWA. It uses **Zero-Targeting with Direct Mixing**.

1. **Targeting:**
   - The controller sets the **Target Roll** and **Target Pitch** to **0 degrees** (Level).
   - *Code Path:* `ModeStabilize::update()` sets `nav_roll_cd = 0`.
2. **Stick Mixing (Direct):**
   - The pilot's stick input is added *directly* to the servo output, bypassing the angle limiter logic used in FBWA.
   - *Effect:* If the plane is level and you hold full right aileron, the servo goes to full deflection. The plane will roll right. As it rolls, the autopilot sees an error (Target=0 vs Actual=Roll) and tries to counter-roll left.
   - *Result:* You are fighting the autopilot. The further you deflect the stick, the more you override the leveling command. This allows you to loop and roll, but the plane will snap back to level the moment you release the stick.

PILOT INTERACTION

- **Sticks Released:** The plane aggressively returns to level flight.
- **Sticks Held:** You can fly in any attitude, including inverted.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- **Throttle:** Manual control (pass-through).

COMPARISON: STABILIZE VS FBWA

- **FBWA:** Stick = Command Angle (e.g., 30 deg). You cannot roll past the limit.
- **STABILIZE:** Stick = Servo Deflection + Leveling Fight. You can roll past the limit.

FAILSAFE LOGIC

- **Attitude Loss:** If the AHRS fails, the plane may try to "level" to a wrong horizon.

KEY PARAMETERS

PARAMETER	DEFAULT	DESCRIPTION
`STAB_PITCH_DOWN`	2.0	(deg) Pitch down trim at low throttle to prevent stalling during glide.
`RLL2SRV_P`	...	Roll P-gain. Determines how aggressively it self-levels.

SOURCE CODE REFERENCE

- **Mode Logic:** `ardupilot/ArduPlane/mode_stabilize.cpp`
- **Control Loop:** `ModeStabilize::run()`

TAKEOFF MODE (PLANE)

EXECUTIVE SUMMARY

TAKEOFF Mode is a specialized autonomous mode designed to safely launch a fixed-wing aircraft. It handles the critical transition from "on the ground" (or in your hand) to "climbing away." It supports Hand Launches, Catapult/Bungee Launches, and Runway Takeoffs, managing throttle and attitude to prevent stalls and crashes during the most vulnerable phase of flight.

THEORY & CONCEPTS

1. THE CATAPULT CONSTRAINT

Launching a plane by hand or bungee is a violent event.

- **The Physics:** The plane goes from 0 m/s to 20 m/s in less than a second.
- **The Error:** GPS signals are too slow to track this (GPS updates at 10Hz, but the launch is over in 100ms).
- **The Solution:** The **Inertial Trigger**. ArduPilot uses the High-Rate IMU (400Hz) to detect the exact millisecond you release the bungee or throw the plane. It only starts the motor *after* it confirms the plane is safely away from your hand.

2. WING LEVELING ON CLIMB-OUT

Most crashes happen in the first 5 seconds after launch.

- **The Problem:** Torque from a large prop can roll a small plane into the ground.
- **The Logic:** During the "Level Altitude" phase, ArduPilot overrides all pilot turn commands. It uses 100\% of its roll authority to keep the wings flat until the plane is high enough to clear the ground.

HARDWARE DEPENDENCY MATRIX

Takeoff mode relies on sensors to detect the launch event.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

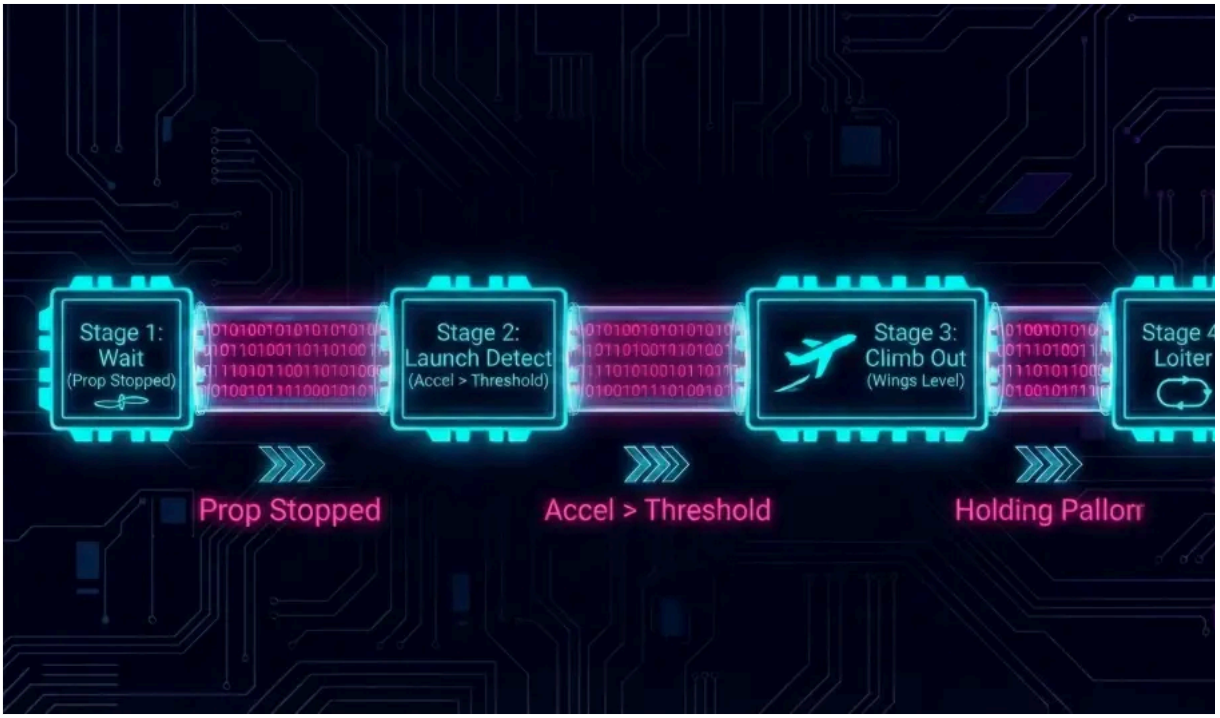




SENSOR	REQUIREMENT	CODE IMPLEMENTATION NOTES
Accelerometer	CRITICAL	Required to detect the launch acceleration (Hand/Catapult). `TKOFF_THR_MINACC`.
GPS	CRITICAL	Required to verify ground speed (`TKOFF_THR_MINSPD`) and track the climb path.
Airspeed	**RECOMMENDED**	Critical for Runway Takeoffs (`TKOFF_ROTATE_SPD`) to ensure the plane doesn't pitch up before it has flying speed.
Compass	CRITICAL	Required to maintain heading during the initial climb-out.

CONTROL ARCHITECTURE (ENGINEER'S VIEW)

The logic operates as a staged sequencer.



1. **Stage 1: Waiting (Throttle Suppressed)**
   - The pilot switches to TAKEOFF mode and arms the plane.
   - **The prop does NOT spin.** The controller holds the throttle at 0.
   - It waits for a "Launch Trigger":
     - **Accel:** Forward acceleration > `TKOFF_THR_MINACC` (e.g., the throw).
     - **Speed:** GPS Ground Speed > `TKOFF_THR_MINSPD` .
   - *Code Path:* `Plane::auto_takeoff_check()` .
2. **Stage 2: Launch Detected (Throttle Ramp)**
   - Once triggered, the throttle ramps up (`TKOFF_THR_SLEW`) to maximum (`TKOFF_THR_MAX`).
   - *Delay:* If `TKOFF_THR_DELAY` is set, the motor start is delayed to protect the thrower's hand.
3. **Stage 3: Initial Climb (Level)**
   - The plane holds the wings level (`LEVEL_ROLL_LIMIT`) and a specific pitch (`TKOFF_LVL_PITCH`) until it reaches `TKOFF_LVL_ALT` .
   - This ensures it gains speed before trying to turn.
4. **Stage 4: Loiter / Mission**
   - Once it reaches `TKOFF_ALT` or `TKOFF_DIST` , the mode completes.
   - *Behavior:* It typically enters a Loiter at the target altitude.

PILOT INTERACTION

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

- **Shake to Wake:** If `TKOFF_ACCEL_CNT` > 0, you may need to shake the plane back and forth to "arm" the launch detector before throwing.
- **Abort:** If the launch fails (e.g., bad throw), switch instantly to **MANUAL** or **FBWA** to kill the throttle or try to save it manually.

FAILSAFE LOGIC

- **Crash Detection:** If the plane detects a crash (high accel spike) immediately after launch, it disarms.
- **Timeout:** If the plane doesn't reach flying speed (`4 m/s`) within `TKOFF_TIMEOUT` seconds after trigger, it disarms to prevent the motor from burning out in the grass.

KEY PARAMETERS

PARAMETER	DEFAULT	DESCRIPTION
`TKOFF_ALT`	50	(m) Target altitude to reach before finishing takeoff.
`TKOFF_LVL_ALT`	20	(m) Altitude below which wings are held level (no turning).
`TKOFF_THR_MINACC`	0	(m/s^2) Acceleration threshold to trigger launch. Set to ~15 for hand launch. 0 = Trigger immediately (Runway).
`TKOFF_THR_DELAY`	0	(1/10s) Delay between throw detection and motor start. Essential for pusher props!
`TKOFF_ROTATE_SPD`	0	(m/s) For runway: speed at which to pitch up (rotate).

SOURCE CODE REFERENCE

- **Mode Logic:** `ardupilot/ArduPlane/mode_takeoff.cpp`
- **Launch Detector:** `Plane::auto_takeoff_check()`

THERMAL MODE (PLANE)

EXECUTIVE SUMMARY

THERMAL Mode (Soaring) allows a fixed-wing aircraft (typically a glider) to autonomously detect, track, and utilize updrafts (thermals) to gain altitude without using motor power. It uses an advanced estimation filter to center the aircraft in the lift, potentially extending flight times indefinitely.

THEORY & CONCEPTS

1. THE PHYSICS OF LIFT (THERMALS)

Thermals are rising columns of air heated by the ground.

- **The Problem:** You cannot "see" a thermal. You can only feel it as a vertical acceleration.
- **The Math:** By comparing the drone's energy state (Speed + Height) against the predicted drag, the EKF can mathematically "extract" the rising air's velocity.
- **The Strategy:** Once lift is detected, the plane turns into it. It "centers" the thermal by tightening the turn where the lift is strongest and widening where it is weakest.

2. CROSS-COUNTRY (XC) SOARING

Thermals drift with the wind.

- In `THERMAL` mode, the plane orbits a moving point in space. This is a 4D problem: Latitude, Longitude, Altitude, and Time.

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

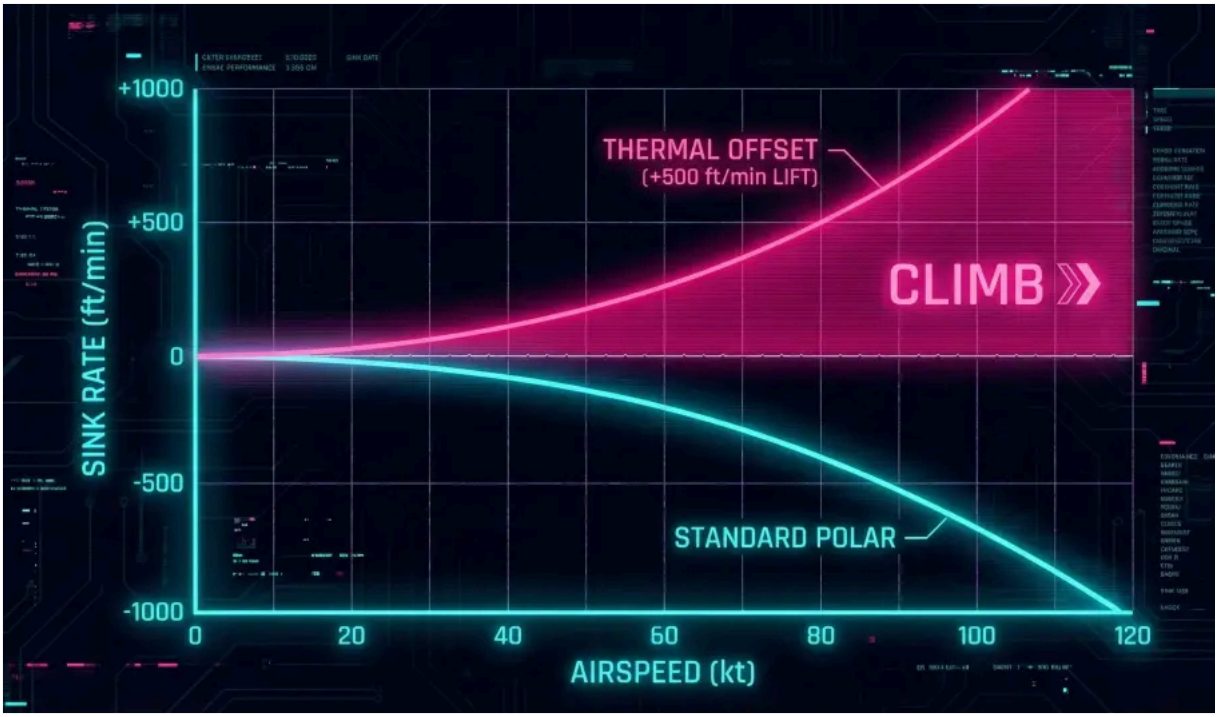




- ArduPilot's thermal EKF treats the rising air mass as a moving "Body," allowing the plane to "climb the elevator" while drifting downwind toward the mission goal.

3. DETECTION (NETTO VARIOMETER)

The `AP_Soaring` library continuously calculates the **Netto Climb Rate**.



- `Netto = Measured_Climb - Theoretical_Sink`.
- *Theoretical Sink:* Calculated from the aircraft's Polar Curve (`POLAR_CD0`, `POLAR_B`) and current airspeed/bank angle.
- If `Netto > SOAR_VSPEED`, a thermal is detected.

HARDWARE DEPENDENCY MATRIX

Effective soaring requires precise energy measurement.

SENSOR	REQUIREMENT	CODE IMPLEMENTATION NOTES
Barometer	CRITICAL	Primary source for vertical speed (Climb Rate).
Airspeed	CRITICAL	Required to calculate the aircraft's theoretical sink rate (Polar Curve). Without airspeed, the flight controller cannot distinguish between "climbing because I pulled up" (stick thermal) and "climbing because the air is rising" (true thermal).
GPS	CRITICAL	Required for position tracking and wind estimation.

CONTROL ARCHITECTURE (ENGINEER'S VIEW)

The Soaring feature acts as a high-level supervisor.

1. **Detection (Netto Variometer):**
    - The `AP_Soaring` library continuously calculates the **Netto Climb Rate**.
    - `Netto = Measured_Climb - Theoretical_Sink`.
    - *Theoretical Sink:* Calculated from the aircraft's Polar Curve (`POLAR_CD0`, `POLAR_B`) and current airspeed/bank angle.
    - If `Netto > SOAR_VSPEED`, a thermal is detected.
2. **Engagement:**

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

    - If `SOAR_ENABLE` is active, the autopilot can automatically pause the current mode (e.g., AUTO, CRUISE, FBWB) and switch to **THERMAL**.
3. **Tracking (The Thermal EKF):**
    - The controller uses an Extended Kalman Filter to estimate the **Center Position**, **Radius**, and **Strength** of the thermal.
    - It flies a Loiter path that constantly adjusts its center to match the estimated thermal core.
4. **Exit Strategy:**
    - The mode exits if:
        - Altitude > `SOAR_ALT_MAX`.
        - Thermal Strength < `SOAR_VSPEED`.
        - Distance drifted from flight path > `SOAR_MAX_DRIFT`.
    - *Code Path:* `ModeThermal::restore_mode()`.

PILOT INTERACTION

- **Automatic:** In most cases, the pilot does nothing. The plane thermals itself.
- **Manual Trigger:** You can manually switch to THERMAL mode if you suspect lift.
- **Suppression:** Toggle the `SOAR_ENABLE` switch low to force the plane back to its previous task.

KEY PARAMETERS

PARAMETER	DEFAULT	DESCRIPTION
`SOAR_ENABLE`	0	Master switch.
`SOAR_VSPEED`	0.7	(m/s) Minimum rising air speed to trigger thermal mode.
`SOAR_ALT_MAX`	350	(m) Ceiling. Stop thermalling above this.
`SOAR_ALT_MIN`	50	(m) Floor. Don't thermal below this.
`POLAR_CD0`	0.027	Drag coefficient (Zero lift). Critical for accurate detection.

SOURCE CODE REFERENCE

- **Mode Logic:** `ardupilot/ArduPlane/mode_thermal.cpp`
- **Update Loop:** `ModeThermal::update()`

TRAINING MODE (PLANE)

EXECUTIVE SUMMARY

TRAINING Mode is designed to teach a student pilot how to fly manually without the risk of losing control. It behaves exactly like **MANUAL** mode (direct stick-to-servo connection) as long as the aircraft remains within safe bank and pitch limits. If the student flies too close to the limit, the autopilot intervenes to gently "bounce" the plane back into the safe zone. Unlike FBWA, it does *not* self-level when you release the sticks.

HARDWARE DEPENDENCY MATRIX

Training mode requires full attitude estimation to know when to intervene.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





SENSOR	REQUIREMENT	CODE IMPLEMENTATION NOTES
Gyroscope	CRITICAL	Required for rate stabilization during intervention.
Accelerometer	CRITICAL	Required to determine bank/pitch angles for the limiter.
GPS	**NONE**	Not used.

CONTROL ARCHITECTURE (ENGINEER'S VIEW)

TRAINING mode implements a **Conditional Switch** logic.

1. **Safe Zone (Manual Pass-through):**
   - *Condition:* `Current_Roll < ROLL_LIMIT` AND `Current_Pitch < PITCH_LIMIT`.
   - *Behavior:* The pilot's inputs are passed directly to the servos (with Manual Expo applied). The autopilot does nothing.
   - *Feel:* The plane flies naturally. It will maintain a bank if you leave it there. It will stall if you fly too slow.
2. **Danger Zone (Intervention):**
   - *Condition:* The aircraft exceeds the angle limits.
   - *Behavior:* The controller overrides the manual pass-through and engages the **Stabilize** controller.
   - *Target:* It sets a target angle equal to the limit (e.g., 45 degrees).
   - *Effect:* If the pilot tries to roll past 45 degrees, the servo fights them to hold 45 degrees. If they release the stick, the plane stays at 45 degrees (or whatever limit was hit).
   - *Code Path:* `ModeTraining::update()`.

PILOT INTERACTION

- **Learning:** This mode is superior to FBWA for learning aerodynamics because you must use the elevator to maintain altitude in a turn (coordinated flight). In FBWA, the autopilot hides this physics requirement.
- **Limits:** You cannot roll upside down or loop.
- **Self-Leveling: NO.** If you bank 20 degrees and let go, the plane stays at 20 degrees (unlike Stabilize/FBWA).

FAILSAFE LOGIC

- **Sensor Failure:** If the AHRS drifts or fails, the "Safe Zone" might actually be upside down. Switch to MANUAL if the plane fights you incorrectly.

KEY PARAMETERS

PARAMETER	DEFAULT	DESCRIPTION
`ROLL_LIMIT_DEG`	45	Max bank angle before intervention.
`PITCH_LIMIT_MAX`	20	Max pitch up.
`PITCH_LIMIT_MIN`	-25	Max pitch down.
`MAN_EXPO_ROLL`	0	Expo applied during the manual phase.

SOURCE CODE REFERENCE

- **Mode Logic:** `ardupilot/ArduPlane/mode_training.cpp`
- **Control Loop:** `ModeTraining::run()`

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





CHAPTER 4: MAVLINK INTERNALS

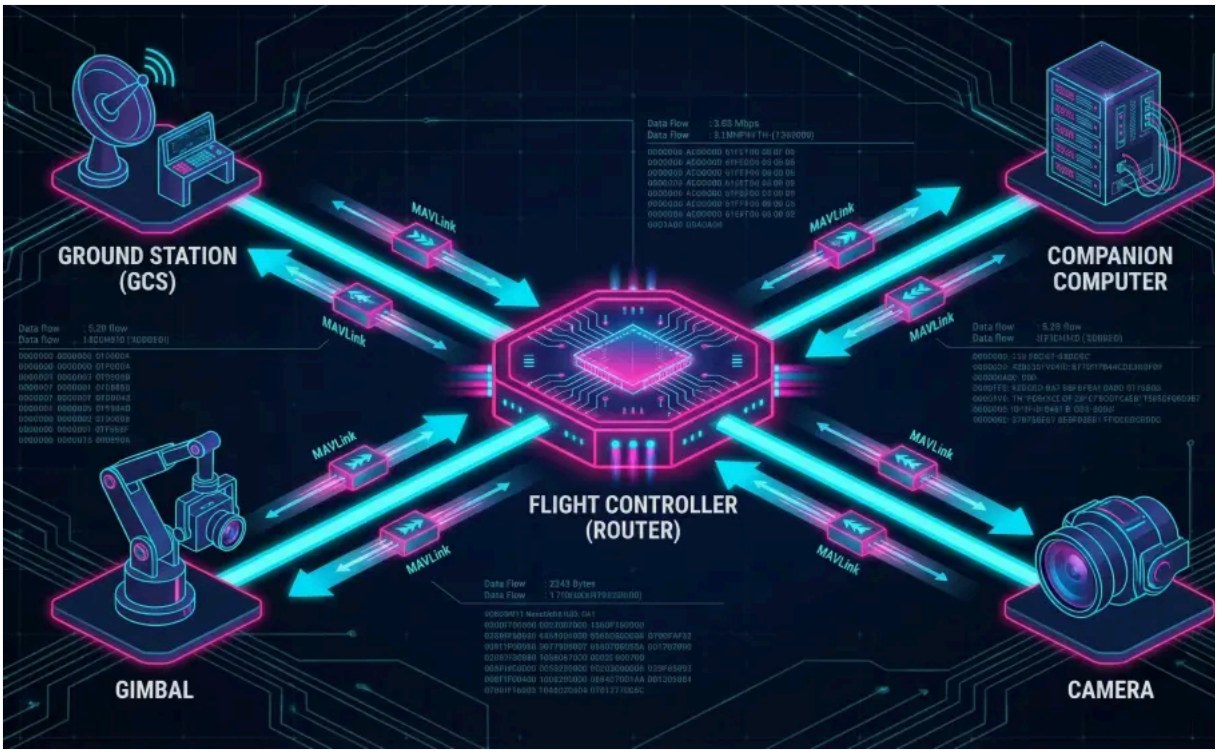
MAVLINK ROUTING

EXECUTIVE SUMMARY

ArduPilot does not just consume MAVLink data; it acts as a **Packet Switch** (Router). If you connect a telemetry radio to `SERIAL1` and an OSD to `SERIAL2`, ArduPilot can transparently forward messages between them. This system allows a ground station to communicate with a peripheral (like a gimbal or smart battery) *through* the flight controller without a direct connection.

THEORY & CONCEPTS

1. THE NETWORK TOPOLOGY PROBLEM



Early drones were simple: Ground Station ↔ Drone. A single serial wire.
Modern drones are networks: Ground Station ↔ Drone ↔ Gimbal ↔ Camera ↔ Companion Computer.

- **The Challenge:** How does the Ground Station tell the Camera to "Take Picture" if the Camera is plugged into the Drone, not the Ground Station?
- **The Solution:** MAVLink Routing. Every device has a unique address (System ID, Component ID). The Flight Controller acts as the "Hub," forwarding mail to the correct recipient.

2. BROADCAST VS. TARGETED

- **Broadcast (Target=0):** "To Whom It May Concern". Examples: `ATTITUDE`, `GPS_RAW`. Everyone needs to see these. ArduPilot copies them to all ports.
- **Targeted:** "To System 1, Component 154". Examples: `COMMAND_LONG` (Move Gimbal). ArduPilot looks up "154" in its phonebook (Routing Table) and sends it *only* to the correct port. This saves bandwidth.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

ARCHITECTURE (THE ENGINEER'S VIEW)

The core logic resides in the `MAVLink_routing` class.

1. THE ROUTING TABLE

The system maintains a dynamic routing table of known devices on the network.

- **Capacity:** It tracks up to `MAVLINK_MAX_ROUTES` (default: 5) external devices.
- **Learning:** The table is populated automatically via `learn_route()`.
  - Every time a valid MAVLink packet arrives, the code notes the `Source System ID` (sysid), `Source Component ID` (compid), and the `Channel` (Serial Port) it came from.
  - *Effect:* If you plug a gimbal into Serial 2, as soon as it sends a HEARTBEAT, ArduPilot "learns" that `SysID:1, CompID:154` lives on `Chan:2`.

2. PACKET FORWARDING LOGIC

When a packet arrives, `check_and_forward()` decides its fate:

1. **Targeted Messages:** (Packet has a specific `Target_SysID` and `Target_CompID`).
   - The router looks up the target in the table.
   - If found: It forwards the packet *only* to the mapped channel.
   - If not found: It broadcasts the packet to all channels (except the source).
2. **Broadcast Messages:** (Target = 0).
   - The packet is forwarded to *all* active channels (except the source).
3. **Local Processing:**
   - Regardless of forwarding, the packet is also passed to the internal flight controller logic for processing.

PRIVATE CHANNELS (`MAV_OPT_PRIVATE`)

You can mark a specific serial port as "Private" using the `SERIALx_OPTIONS` parameter.

- **Behavior:**
  - Packets *from* a private channel are **NOT** forwarded to other channels (unless specifically whitelisted, like GoPro).
  - Packets *to* a private channel are restricted.
- **Use Case:** Connect a high-bandwidth internal peripheral (like a companion computer sending vision data) without flooding your low-bandwidth telemetry radio link.

COMMON ISSUES & TROUBLESHOOTING

"I CAN'T SEE MY GIMBAL PARAMETERS"

- **Cause:** The routing table might be full, or the Gimbal hasn't sent a HEARTBEAT yet.
- **Fix:** Ensure the peripheral sends heartbeats. Check if you have too many MAVLink peripherals (more than 5).

"TELEMETRY LAG"

- **Cause:** Broadcast loops. If you have two telemetry radios connected to the same network (e.g., WiFi + RF) without proper configuration, packets can loop endlessly.
- **Fix:** ArduPilot has loop detection (it won't forward a packet back to its source), but complex topologies can still cause storms. Use distinct System IDs.

"GCS SEES THE DRONE BUT NOT THE GPS"

- **Cause:** `SERIALx_PROTOCOL` mismatch.
- **Fix:** Ensure both ports use MAVLink 1 or MAVLink 2. Routing between protocol versions works, but features like Signing might break.

SOURCE CODE REFERENCE

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- **Routing Logic:** `libraries/GCS_MAVLink/MAVLink_routing.cpp`
- **Table Definition:** `libraries/GCS_MAVLink/MAVLink_routing.h`

PRACTICAL GUIDE: ISOLATING TRAFFIC WITH PRIVATE CHANNELS

If you connect a Companion Computer (RPi) sending high-speed MAVLink (Vision, Obstacle Avoidance) to Serial 2, ArduPilot's default behavior is to forward **ALL** of that traffic to your Telemetry Radio on Serial 1. This will instantly choke your 57600 baud radio link.

THE FIX: `SERIALN_OPTIONS`

1. Identify the port connecting to the high-bandwidth device (e.g., `SERIAL2`).
2. Set `SERIAL2_OPTIONS` = **1024** (Bit 10: "Don't forward MAVLink to/from").
    - *Note:* If you are already using other options (like Swap), add 1024 to the existing value.
3. **Result:**
    - ArduPilot **consumes** the Vision data from Serial 2.
    - ArduPilot **does NOT forward** it to Serial 1 (Telemetry).
    - Your radio link remains clean and low-latency.

For more details, see the ArduPilot Wiki: Serial Options.

MAVLINK SCHEDULER & STREAM RATES

EXECUTIVE SUMMARY

You set `SR1_POSITION` to 50Hz, but your telemetry log shows only 10Hz. Why? The answer lies in the **MAVLink Scheduler**. ArduPilot uses a sophisticated "Bucket Scheduler" with strict prioritization and back-pressure flow control. It doesn't just blast data; it carefully manages the serial link bandwidth to ensure critical messages (like Heartbeats and Command ACKs) never get dropped in favor of bulk data.

THEORY & CONCEPTS

1. THE "TOKEN BUCKET" METAPHOR

Imagine a bucket with a hole in the bottom.

- **Tokens:** Bandwidth (Bytes).
- **Refill:** The serial link speed fills the bucket at a constant rate (e.g., 5760 bytes/sec).
- **Drain:** Sending a message removes tokens.
- **The Logic:** If the bucket is empty (no bandwidth), you can't send. You have to wait.
- **Prioritization:** Who gets to take tokens first? Heartbeats get first dibs. Status Text gets second. The "bulk" data (Attitude, GPS) fights for whatever scraps are left.

2. JITTER AND DETERMINISM

MAVLink is **Best Effort**, not Deterministic.

- *CAN Bus:* Guaranteed timing.
- *MAVLink:* "I'll send it when I can."
- *Result:* A 10Hz stream might arrive at intervals of 80ms, 120ms, 90ms, 110ms. Averaged out, it's 10Hz, but instantaneous timing is jittery. Do not rely on MAVLink for real-time control loops (use DroneCAN instead).

ARCHITECTURE (THE ENGINEER'S VIEW)

The scheduling logic is contained in `GCS_MAVLINK::update_send()`. This function is called every main loop cycle but runs for a maximum of **5 milliseconds** (or less if the scheduler budget is tight).

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





1. THE PRIORITY TIERS

Messages are processed in three distinct tiers. A lower tier only runs if the higher tier has nothing to send.

- **Tier 1: Deferred Messages (Critical):**
  - *Examples:* `HEARTBEAT`, `SYS_STATUS`, `POWER_STATUS`.
  - *Behavior:* These are hard-coded or highly specific. They are checked every loop. If their interval has passed, they are sent immediately.
- **Tier 2: Pushed Messages (Events):**
  - *Examples:* `COMMAND_ACK`, `STATUSTEXT` (Error messages), `PARAM_VALUE` (in response to a read).
  - *Behavior:* When an event happens (e.g., you switch flight modes), a message ID is "pushed" onto a queue. The scheduler drains this queue as fast as possible.
- **Tier 3: Streamed Messages (Bulk Data):**
  - *Examples:* `ATTITUDE`, `GLOBAL_POSITION_INT`, `VFR_HUD`.
  - *Behavior:* These are the "Background Noise" of MAVLink. They are grouped into **Buckets**.

2. THE BUCKET SYSTEM

ArduPilot handles hundreds of possible MAVLink messages. Checking a timer for *every single one* every loop would waste CPU. Instead, it uses **Stream Buckets**.

- **Initialization:** At startup (or when you change `SRx_` params), the code calculates the target interval (e.g., 50Hz = 20ms).
- **Assignment:** Each message ID is assigned to one of ~10 Buckets based on its rate.
  - *Bucket 0:* High Speed (e.g., 50Hz+).
  - *Bucket 9:* Very Slow (e.g., 0.1Hz).
- **Execution:** The scheduler processes **one bucket per loop**.
  - *Loop 1:* Check Bucket 0.
  - *Loop 2:* Check Bucket 1.
  - ...
  - *Effect:* This distributes the CPU load.

3. FLOW CONTROL (BACK PRESSURE)

This is the most common reason for missing data.

- **The Check:** Before sending *any* byte, the code calls `HAVE_PAYLOAD_SPACE`.
- **The Hardware:** It queries the UART driver: "Do you have room in your TX buffer?"
- **The Drop:** If the buffer is full (because the radio hasn't sent the previous bits yet), the scheduler **aborts** that message. It does *not* retry immediately. It moves on.
- *Result:* If your baud rate is too low for the requested stream rates, packets are silently dropped to prevent the flight controller from freezing while waiting for the radio.

COMMON ISSUES & TROUBLESHOOTING

"MY STREAM RATE IS FLUCTUATING"

- **Cause:** Bandwidth saturation. The "Back Pressure" mechanism is dropping packets randomly when the buffer fills.
- **Fix:** Reduce `SRx_` params or increase Baud Rate.

"I REQUESTED 50HZ BUT GOT 10HZ"

- **Cause 1:** `SRx_` params are "Requests", not guarantees. If the loop rate is slow or the buckets are full, you get what you get.
- **Cause 2: Link Throttling**. Some radios (like ELRS) have small buffers. ArduPilot fills them instantly, then pauses.

"MY OSD FLICKERS"

- **Cause:** OSDs often listen to specific streams. If high-priority traffic (like a parameter download) floods the link, the "Tier 3" stream data gets delayed.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





SOURCE CODE REFERENCE

- **Scheduler Loop:** `GCS_MAVLINK::update_send()`
- **Bucket Initialization:** `initialise_message_intervals_from_streamrates()`

PRACTICAL GUIDE: DEBUGGING MISSING TELEMETRY

You asked for `ATTITUDE` at 20Hz, but you are only getting 5Hz. Who is lying?

STEP 1: CHECK THE STATS

ArduPilot tracks dropped packets.

1. Connect via Mission Planner.
2. Go to the **Status** tab.
3. Look for `stream_slowdown`.
    - **Value:** This is a percentage (0-100) of how often the scheduler had to wait for the UART buffer.
    - **Interpretation:** If `stream_slowdown` > 5\%, your baud rate is too low for the amount of data you are trying to push.

STEP 2: CALCULATE THE COST

Bytes per second = `Sum(PacketSize * Rate)`.

- `ATTITUDE` (28 bytes) * 20 Hz = 560 B/s.
- `GPS_RAW_INT` (30 bytes) * 5 Hz = 150 B/s.
- **Total:** 710 B/s.
- **Capacity (57600 baud):** ~5,700 B/s. (You are safe).
- **Capacity (ELRS Airport):** Variable, but often < 1000 B/s at low packet rates.

THE FIX

1. **Reduce Rates:** Set unimportant streams (`SRx_RC_CHAN`, `SRx_RAW_SENS`) to **0**.
2. **Increase Baud:** Move from 57600 to 115200 or 460800 if hardware permits.
3. **Use `MAV_CMD_SET_MESSAGE_INTERVAL`:** This bypasses the clumsy `SRx_` groups and lets you request *only* `ATTITUDE` without also getting `AHRS2`, `AHRS3`, etc.

MAVLINK SERIALIZATION ARCHITECTURE

1. THE "WIRE" REALITY: WHY MAVLINK IS NOT PROTOBUF

MAVLink is designed for a specific hostile environment: **Low-Bandwidth, High-Latency, Unreliable Serial Links** (e.g., 57600 baud telemetry radios). In this domain, every byte costs airtime.

Unlike Protobuf, JSON, or XML, which prioritize schema flexibility and forward compatibility via Tag-Length-Value (TLV) structures, MAVLink prioritizes **Determinism** and **Overhead Efficiency**.

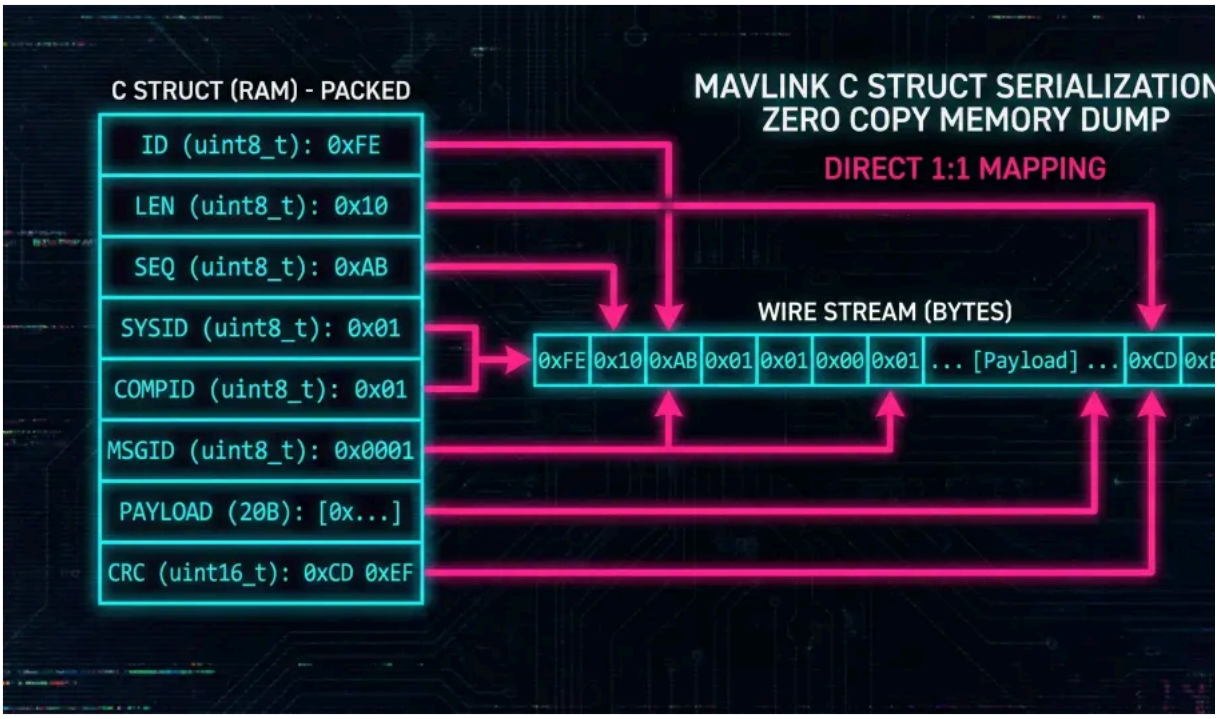
CORE PHILOSOPHY: C-STRUCT PACKING

MAVLink serialization is effectively "C-structs over the wire."

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY







- **Mechanism:** The message definition (XML) is compiled into a C struct.
- **Serialization:** The memory content of that struct is sent directly (Little Endian).
- **Parsing:** The receiver casts the buffer back to the struct (or copies it).
- **Cost:** Zero parsing overhead (CPU) and Zero serialization overhead (Bandwidth) beyond the header.

---

2. ANATOMY OF A PACKET (V2.0)

A MAVLink v2 packet is a rigid byte stream (defined in `mavlink_types.h`). It does not describe *what* it is carrying; it assumes the receiver has the same dictionary (XML definitions).

BYTE	FIELD	DESCRIPTION
0	`STX`	Magic Marker (`0xFD` for v2).
1	`LEN`	Payload Length (0-255).
2	`INC_FLAGS`	Incompatibility Flags (e.g., Signing).
3	`CMP_FLAGS`	Compatibility Flags.
4	`SEQ`	Packet Sequence (0-255) for loss detection.
5	`SYSID`	System ID (Sender, e.g., Drone).
6	`COMPID`	Component ID (Sender, e.g., Autopilot vs. Camera).
7-9	`MSGID`	Message ID (24-bit). The "Topic".
10...	`PAYLOAD`	The serialized data (up to 255 bytes).
N-2	`CRC`	Checksum (CRC16-MCRF4XX) of Header + Payload + **Extra Byte**.

THE "EXTRA BYTE" (CRC_EXTRA)

MAVLink's unique integrity check.

- **Problem:** If Sender V1 sends struct `{float A}` and Receiver V2 expects `{float A, float B}`, a blind cast is dangerous.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

- **Solution:** The CRC calculation includes a hidden "Seed" byte derived from the hash of the XML message definition.
- **Result:** If the Sender and Receiver have different XML definitions for `MSGID X`, the CRC check fails silently. **Schema enforcement is done by the checksum, not the parser.**

---

3. COMPARATIVE ANALYSIS: MAVLINK VS. PROTOBUF

Why doesn't ArduPilot just use Protobuf?

FEATURE	MAVLINK	PROTOBUF
Encoding	**Packed Binary (Fixed).** A float is always 4 bytes at a known offset.	**Varint + TLV.** A float is a Tag + WireType + 4 bytes.
Parsing	**O(1) / Zero-Copy.** Cast buffer to struct. Extremely fast on 8-bit MCUs.	**Recursive Descent.** Requires CPU cycles to decode tags and reconstruct objects.
Bandwidth	**Minimal.** Header (10 bytes) + Data. No field tags.	**Moderate.** Every field carries a tag overhead. Varints save space for small ints but cost for large ones.
Schema	**Rigid.** Changing a field changes the `CRC_EXTRA`, breaking the link.	**Flexible.** Unknown fields are ignored. Great for microservices, bad for safety-critical control loops.
Safety	**Deterministic.** We know exactly how many bytes `ATTITUDE` takes.	**Variable.** Message size varies with content (e.g., zeros are compressed in proto3). Harder to schedule on fixed time-slots.

Verdict: Protobuf is superior for *Inter-Process Communication (IPC)* on a Companion Computer. MAVLink is superior for *Telemetry and Command & Control (C2)* over a radio link.

---

4. ARDUPILOT IMPLEMENTATION

The rubber meets the road in the `GCS_MAVLink` library.

- `GCS_MAVLink.cpp`: The central hub. It manages the routing of messages between UARTs (Serial ports) and the internal Data Bus.
- **Fast Channel:** High-rate streams (Attitude, VFR_HUD) are often pushed into ring buffers to be drained by the UART DMA.
- **Lazy Serialization:** ArduPilot often checks `comm_get_txspace()` before attempting to serialize. If the radio buffer is full, the packet is dropped immediately to prevent latency buildup (Backpressure).

FIELD SORTING (WIRE OPTIMIZATION)

MAVLink generators (like `pymavlink`) reorder fields in the struct to ensure **Natural Alignment**.

- All `uint64_t` / `double` (8 bytes) come first.
- Then `uint32_t` / `float` (4 bytes).
- Then `uint16_t` (2 bytes).
- Then `uint8_t` (1 byte).

Why? To prevent the C compiler from adding "Padding Bytes" between fields to align memory access. This ensures the "Wire Format" is perfectly dense, with zero wasted bits.

MAVLINK FLOW CONTROL & THROTTLING

EXECUTIVE SUMMARY

ArduPilot uses a "Pull-Based" flow control system. Instead of blindly pushing messages into a buffer and hoping for the best, the scheduler explicitly queries the hardware: *"Do you have space for 40 bytes?"*. If the answer is No, the message is skipped for this

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





cycle. This prevents internal buffer overflows but can result in "gappy" telemetry if the link bandwidth is saturated.

THEORY & CONCEPTS

1. BACKPRESSURE (THE CLOGGED PIPE)

Imagine a funnel. You can pour water in faster than it drains out, but only until the funnel is full.



- **The Funnel:** The TX Buffer (RAM).
- **The Drain:** The Serial Baud Rate (Radio Speed).
- **The Pour:** MAVLink Scheduler.
- **Backpressure:** When the funnel is full, the water (data) spills over. ArduPilot checks the funnel level *before* pouring. If it's full, it holds the cup.

2. CTS/RTS (HARDWARE FLOW CONTROL)

- **CTS (Clear To Send):** A wire from the Radio to the Flight Controller.
  - *Low:* "I'm ready!"
  - *High:* "Stop! My buffer is full!"
- **Mechanism:** If the radio is transmitting a slow packet over the air, it pulls CTS High. The Flight Controller's UART driver sees this and stops pushing bits out of its internal buffer.

ARCHITECTURE (THE ENGINEER'S VIEW)

1. THE GATEKEEPER: `COMM_GET_TXSPACE`

Before any message is generated, the macro `HAVE_PAYLOAD_SPACE(chan, id)` is called.

- It calls `comm_get_txspace(chan)`.
- It queries the HAL UART Driver for `write_space()`.
- *Result:* This returns the number of bytes free in the hardware ring buffer.

2. THE DECISION LOGIC

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

Inside `do_try_send_message()`:

1. **Check Space:** Calculate the packet size. If `Free_Space < Packet_Size`, **ABORT**.
2. **Abort Behavior:**
    - The function returns `false`.
    - The `last_sent_ms` timestamp is **NOT** updated.
    - *Consequence:* The scheduler moves on to the next task. In the next main loop (e.g., 2.5ms later), it will see this message is still "overdue" and try again.
    - *Effect:* Messages simply "slide" in time until bandwidth opens up. They are not dropped in the traditional networking sense (packet loss), but they are delayed (latency).

3. HARDWARE FLOW CONTROL (RTS/CTS)

If your telemetry radio supports hardware flow control (RTS/CTS pins):

- **Radio Buffer Full:** The radio pulls the CTS pin high.
- **UART Driver:** The HAL driver detects this and stops shifting bytes out of its ring buffer.
- **Ring Buffer Fills:** The internal buffer fills up.
- **Back Pressure:** `comm_get_txspace` starts returning 0.
- **Scheduler Pauses:** ArduPilot stops generating MAVLink packets entirely until the radio clears its throat.

COMMON ISSUES & TROUBLESHOOTING

"TELEMETRY STOPS WHEN I DOWNLOAD PARAMETERS"

- **Cause:** Parameter packets are large and sent as fast as possible. They fill the buffer instantly.
- **Mechanism:** The scheduler sees `txspace == 0` for lower-priority telemetry messages (like Attitude) and skips them.
- **Fix:** Use `MAV_CMD_SET_MESSAGE_INTERVAL` to increase priority of critical streams, or use a faster link (USB/Wifi).

"ELRS / CROSSFIRE TELEMETRY IS SLOW"

- **Cause:** These links have extremely small buffers (often just a few packets).
- **Mechanism:** The link throttles heavily. ArduPilot's 50Hz streams saturate the link immediately, causing massive "sliding" delays.
- **Fix:** Reduce `SRx_` rates to match the *actual* throughput of the link (e.g., 2Hz or 4Hz).

SOURCE CODE REFERENCE

- **Space Check:** `comm_get_txspace()`
- **Send Loop:** `GCS_MAVLINK::do_try_send_message()`

ON-DEMAND STREAMING (MAVLINK)

EXECUTIVE SUMMARY

Modern Ground Control Stations (like QGroundControl) and custom apps (like MAVLink HUD) don't rely on the static `SRx_` parameters. Instead, they use the **On-Demand Streaming** protocol (`MAV_CMD_SET_MESSAGE_INTERVAL`). This allows the GCS to request exactly the messages it needs, at the exact rates it needs, dynamically at runtime.

THEORY & CONCEPTS

1. THE COMMAND PATTERN

In computing, the **Command Pattern** is used to encapsulate a request as an object. `MAV_CMD_SET_MESSAGE_INTERVAL` is a perfect example. Instead of changing a global state (a parameter), the GCS sends a discrete instruction to the system's "mouth" (the

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





MAVLink instance) to change its behavior. This is faster and more reliable than parameter syncing for real-time adjustments.

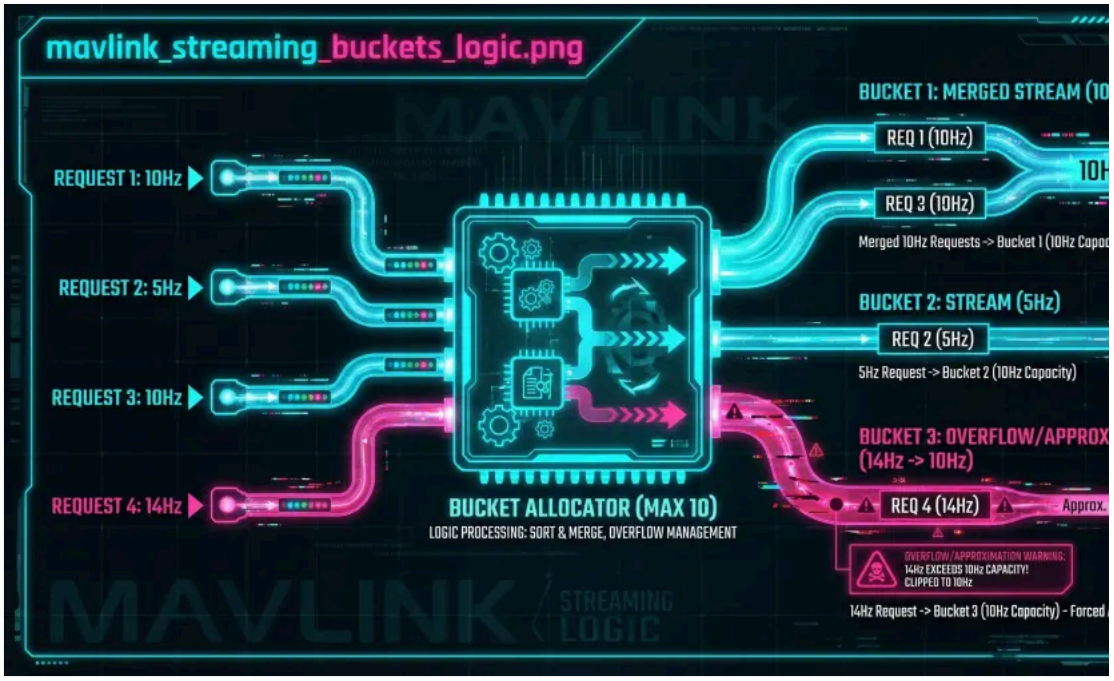
2. SPACE VS. TIME COMPLEXITY

Why use "Buckets"? If ArduPilot tracked 255 separate timers for 255 possible messages, the CPU would spend significant time just checking if it's "time to send" each one. By grouping them into 10 buckets, the scheduler only checks 10 timers. This is an optimization that prioritizes **CPU Efficiency** (saving time) over **Granular Precision** (saving space).

THE MECHANISM (ENGINEER'S VIEW)

When ArduPilot receives Command #511 (`MAV_CMD_SET_MESSAGE_INTERVAL`), it executes the following logic in `set_ap_message_interval()`:

1. **Request Handling:**
    - Input: `Message_ID` (e.g., 30 for Attitude) and `Interval` (us).
    - Conversion: Interval is converted to milliseconds. `0` or `-1` means "Stop Streaming".
2. **Bucket Assignment:**
    - ArduPilot does **not** store a unique timer for every message ID. It uses a **Bucket System** to save RAM.
    - It maintains **10 Stream Buckets**. Each bucket has a single `interval_ms`.



    - *Algorithm:*
        - **Search:** Does a bucket already exist with this exact interval? → Assign to it.
        - **Create:** Is there an empty bucket? → Create new bucket with this interval and Assign.
        - **Fallback (The "Closest" Match):** If all 10 buckets are used, the code finds the bucket with the *closest* interval and forces the message into it.
        - *Result:* You might request 45Hz, but if the closest bucket is 10Hz, you get 10Hz.

CRITICAL LIMITATIONS

- **The "10 Bucket" Limit:** You can only have 10 *unique* stream rates active simultaneously per serial port.
    - *Bad:* Requesting 10Hz, 11Hz, 12Hz... (consumes 3 buckets).
    - *Good:* Requesting 10Hz, 10Hz, 10Hz... (consumes 1 bucket).
- **Conflict:** This system overrides the `SRx_` parameters. If you manually request a message, it is removed from its old bucket and moved to the new one.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

TROUBLESHOOTING

"I ASKED FOR 20HZ BUT I'M GETTING 4HZ"

- **Cause:** You have exhausted the 10 buckets with random rate requests. ArduPilot forced your message into an existing bucket that happened to be slow (4Hz).
- **Fix:** Align your requests. Ask for standard rates (e.g., 10, 20, 50) rather than arbitrary calculations (e.g., 14.5Hz).

SOURCE CODE REFERENCE

- **Logic:** `GCS_MAVLINK::set_ap_message_interval()`
- **Command Handler:** `handle_command_set_message_interval()`

PRACTICAL GUIDE: CORRECTLY REQUESTING HIGH-SPEED TELEMETRY

Don't just spam requests. Respect the bucket limit to get the performance you want.

THE GOLDEN RULE

Always align your request rates. Use a small set of standard frequencies (e.g., 1Hz, 10Hz, 50Hz) for *all* your message types.

DO NOT DO THIS (THE "BUCKET BUSTER")

- Request Attitude at 30Hz
- Request GPS at 5Hz
- Request Battery at 1Hz
- Request VFR_HUD at 4Hz
- ...
- **Result:** You will quickly burn all 10 buckets. ArduPilot will then start merging your requests into "close enough" buckets, giving you unpredictable rates (e.g., your 30Hz Attitude might drop to 20Hz if that bucket was already full).

DO THIS (THE "ALIGNED STRATEGY")

- **Fast Group (20Hz):** Attitude, VFR_HUD
- **Medium Group (5Hz):** GPS, Global Position
- **Slow Group (1Hz):** Battery, SysStatus
- **Result:** You only use **3 buckets**. You have 7 left for other dynamic needs. Your rates will be exact and stable.

HOW TO VERIFY

Monitor the `Res` (Result) field in the `COMMAND_ACK` you receive.

- `0` (ACCEPTED): Success.
- `4` (FAILED): Command rejected (rare for this cmd).
- **Note:** ArduPilot silently modifies the rate if buckets are full, so `ACCEPTED` doesn't guarantee the *exact* rate if you violated the bucket rule. You must monitor the actual stream rate to confirm.

For more details, see the ArduPilot Wiki: Requesting Data.

THE PARAMETER PROTOCOL

EXECUTIVE SUMMARY

Downloading parameters is often the slowest part of connecting a Ground Control Station (GCS). ArduPilot manages over 1000 parameters. Dumping them all at once would choke the link and kill telemetry. The **Parameter Protocol** uses a cooperative state

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





machine to trickle these parameters down the pipe without interrupting critical flight data.

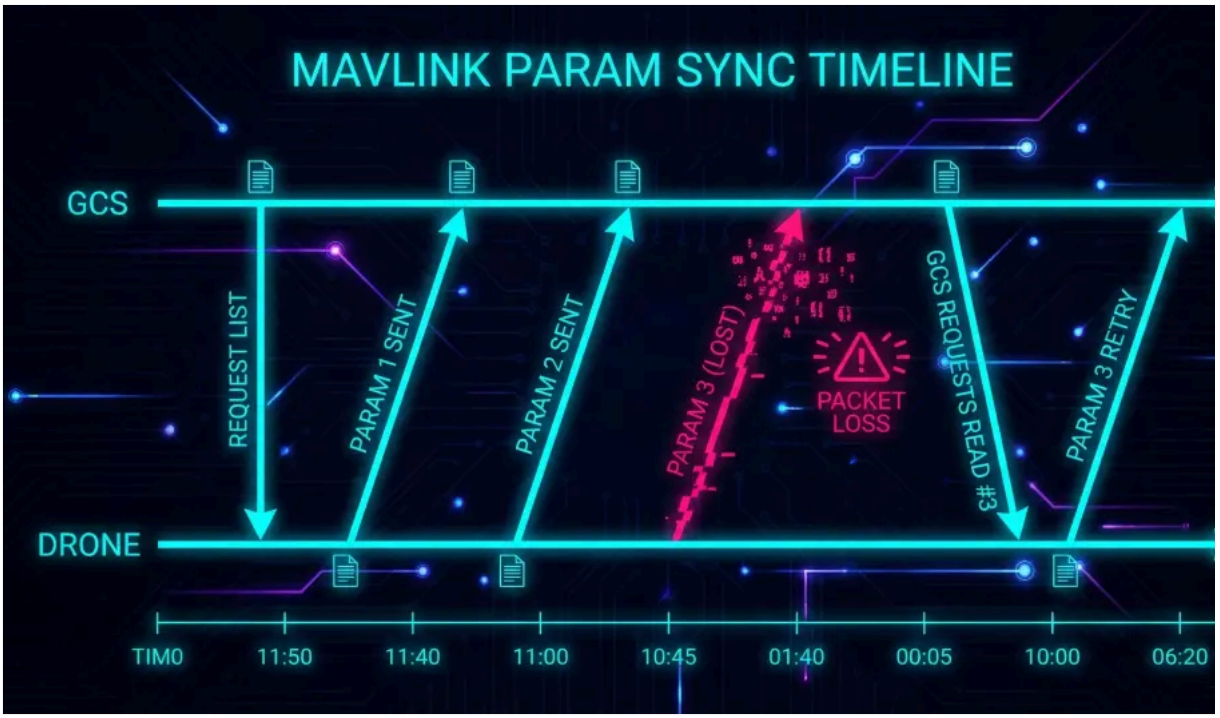
THEORY & CONCEPTS

1. RELIABLE TRANSPORT OVER UNRELIABLE LINKS

MAVLink usually runs over UDP (WiFi) or Serial (Radio), both of which are "Lossy." The Parameter Protocol implements its own **Reliability Layer**. It doesn't use standard TCP-style "sliding windows"; instead, it uses a **Request-Response** model. The Drone is the "Server" and the GCS is the "Client."

2. STATE SYNCING

The hardest problem in distributed systems is state synchronization. With 1000+ parameters, ArduPilot and the GCS must agree on the "Source of Truth." This is why Bulk Downloads take time: the system is ensuring that every single bit of the flight configuration is identical on both sides of the radio link.



ARCHITECTURE (THE ENGINEER'S VIEW)

The logic resides in `GCS_MAVLINK::queued_param_send()`.

1. THE ASYNC QUEUE (FAST LANE)

When you change a parameter (`PARAM_SET`) or request a single value (`PARAM_REQUEST_READ`), the response is queued in a dedicated high-priority buffer (`param_replies`).

- **Processing:** `send_parameter_async_replies()` checks this buffer first.
- **Priority:** These messages override the bulk download list. This ensures that if you toggle a switch, the GCS updates immediately, even if it's still downloading the full list in the background.

2. THE BULK ITERATOR (SLOW LANE)

When a GCS requests the full list (`PARAM_REQUEST_LIST`), ArduPilot enters an iterator mode.

- **State:** It stores a token (`_queued_parameter_token`) pointing to the current position in the parameter table.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

- **Step:** In each main loop cycle, it sends *one* parameter if bandwidth allows.
- **Throttling:** It calculates the available bandwidth and limits parameter traffic to roughly **33\%** (`1/3`). This guarantees that 66\% of the link remains free for Attitude, GPS, and Heartbeats.

3. RE-TRANSMISSION HANDLING

ArduPilot does *not* track which parameters the GCS has received. It blindly iterates through the list.

- **Packet Loss:** If a `PARAM_VALUE` packet is dropped, the GCS will notice a gap in the index (e.g., received #5, then #7).
- **Recovery:** The GCS must send `PARAM_REQUEST_READ` for the missing index (#6). ArduPilot treats this as a "Fast Lane" async request and replies immediately.

COMMON ISSUES & TROUBLESHOOTING

"PARAMETER DOWNLOAD GETS STUCK"

- **Cause:** High packet loss. The GCS is spending all its time requesting missing packets, saturating the "Fast Lane" and pausing the bulk download.
- **Fix:** Reduce telemetry rates (`SRx_`) to free up bandwidth, or improve radio signal.

"OSD UPDATES ARE SLOW WHILE CONNECTING"

- **Cause:** Even with 33\% throttling, the parameter download is a heavy load. The OSD stream might be pushed to lower-priority buckets.

SOURCE CODE REFERENCE

- **Bulk Sender:** `GCS_MAVLINK::queued_param_send()`
- **Async Reply:** `GCS_MAVLINK::send_parameter_async_replies()`

THE MISSION ITEM PROTOCOL

EXECUTIVE SUMMARY

Uploading a mission (Waypoints) is a critical operation that requires 100\% data integrity. Unlike telemetry (where dropping a packet is fine), dropping a Waypoint could crash the drone. The **Mission Item Protocol** is a transactional state machine that ensures every single point is received and verified before the mission is accepted. It handles Missions, Geofences, and Rally Points using a unified codebase.

THEORY & CONCEPTS

1. TRANSACTIONAL INTEGRITY

In database theory, a transaction must be **Atomic**. It either succeeds completely or fails completely. Mission uploads follow this rule. If you lose connection after uploading 49 of 50 waypoints, ArduPilot will reject the entire mission. It never attempts to fly a partial mission.

2. THE "HANDSHAKE" DESIGN

Why does ArduPilot *request* the items rather than the GCS *pushing* them? This is a "Pull" design that prevents the drone's memory from being flooded. The drone controls the rate of the transaction, ensuring it has enough time to write each waypoint to the permanent EEPROM/Flash storage before requesting the next one.

ARCHITECTURE (THE ENGINEER'S VIEW)

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





The logic is implemented in the `MissionItemProtocol` class.

1. UNIFIED DESIGN

ArduPilot uses polymorphism to handle different data types. The base class manages the MAVLink handshake, while subclasses handle the storage:

- `MissionItemProtocol_Waypoints`: Writes to `AP_Mission` (EEPROM).
- `MissionItemProtocol_Fence`: Writes to `AC_Fence` (RAM/Storage).
- `MissionItemProtocol_Rally`: Writes to `AP_Rally` (Storage).

2. THE UPLOAD TRANSACTION (GCS TO DRONE)

The upload follows a strict "Pull" model to ensure reliability.



1. **Initiation:** The GCS sends `MISSION_COUNT` (e.g., "I have 50 waypoints").
2. **Allocation:** ArduPilot checks if it has memory for 50 items. If yes, it requests the first one.
3. **The Loop:**
    - ArduPilot sends `MISSION_REQUEST_INT` for Item #0.
    - GCS sends `MISSION_ITEM_INT` #0.
    - ArduPilot verifies #0, stores it, and sends `MISSION_REQUEST_INT` for Item #1.
    - *Reliability:* If ArduPilot doesn't receive Item #0 within 1 second, it re-sends the request. This repeats until the item arrives or the operation times out.
4. **Completion:** Once the last item is received, ArduPilot sends `MISSION_ACK` (Accepted).

3. THE DOWNLOAD TRANSACTION (DRONE TO GCS)

Downloading is a "Push" model but still acknowledged.

1. GCS sends `MISSION_REQUEST_LIST`.
2. ArduPilot sends `MISSION_COUNT`.
3. GCS requests items one by one (`MISSION_REQUEST_INT`).
4. ArduPilot replies with the item.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

COMMON ISSUES & TROUBLESHOOTING

"MISSION UPLOAD FAILED (TIMEOUT)"

- **Cause:** Extremely high packet loss. The "Request → Item" loop failed so many times that the 8-second global timeout triggered.
- **Fix:** Check radio range. The protocol handles occasional drops, but not a dead link.

"MISSION REJECTED"

- **Cause:** You tried to upload more waypoints than the board supports (e.g., >700 on some boards), or the `MISSION_COUNT` packet was corrupted.

SOURCE CODE REFERENCE

- **Protocol Logic:** `MissionItemProtocol.cpp`
- **Count Handler:** `handle_mission_count()`

PRACTICAL GUIDE: MISSION UPLOAD SAFETY

A corrupt mission can cause a flyaway. Follow these rules to ensure safety.

1. VERIFY THE ACK

Never assume a mission uploaded just because the progress bar finished.

- **Look for:** `MISSION_ACK` (Result: 0).
- **Meaning:** ArduPilot has verified the count, stored all points, and closed the transaction. The mission is safe to fly.

2. THE "READBACK" RULE

In professional operations, we trust but verify.

1. Upload the Mission.
2. **Download** the Mission from the drone (Readback).
3. Compare the downloaded path with your planned path.
4. *Why?* This catches rare EEPROM corruption or logic errors where ArduPilot might have "snapped" a waypoint to a nearby grid or truncated a float value.

3. CHECK "TOTAL DISTANCE"

After upload, Mission Planner displays "Total Distance".

- **Sanity Check:** If you planned a 2km flight but the total distance is 12,000km, one of your waypoints is at (0,0) [Null Island]. **Do not fly.**

MAVLINK FTP

EXECUTIVE SUMMARY

MAVLink FTP (File Transfer Protocol) allows a Ground Control Station to interact with the flight controller's onboard filesystem (SD Card or Flash). It is used to download **Dataflash Logs**, upload **Terrain Data**, and manage **Lua Scripts**. Unlike standard telemetry, it is a request-response protocol designed for large binary blobs.

THEORY & CONCEPTS

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





1. ENCAPSULATION VS. NATIVE PROTOCOL

Most flight controllers use a simple SD card. You could just use a native SD protocol, but that requires a dedicated hardware wire. MAVLink FTP **encapsulates** filesystem commands (LS, RM, CP, READ) inside MAVLink packets. This allows you to manage files using the same radio link you use for telemetry, saving weight and complexity.

2. MULTI-THREADING IN REAL-TIME SYSTEMS

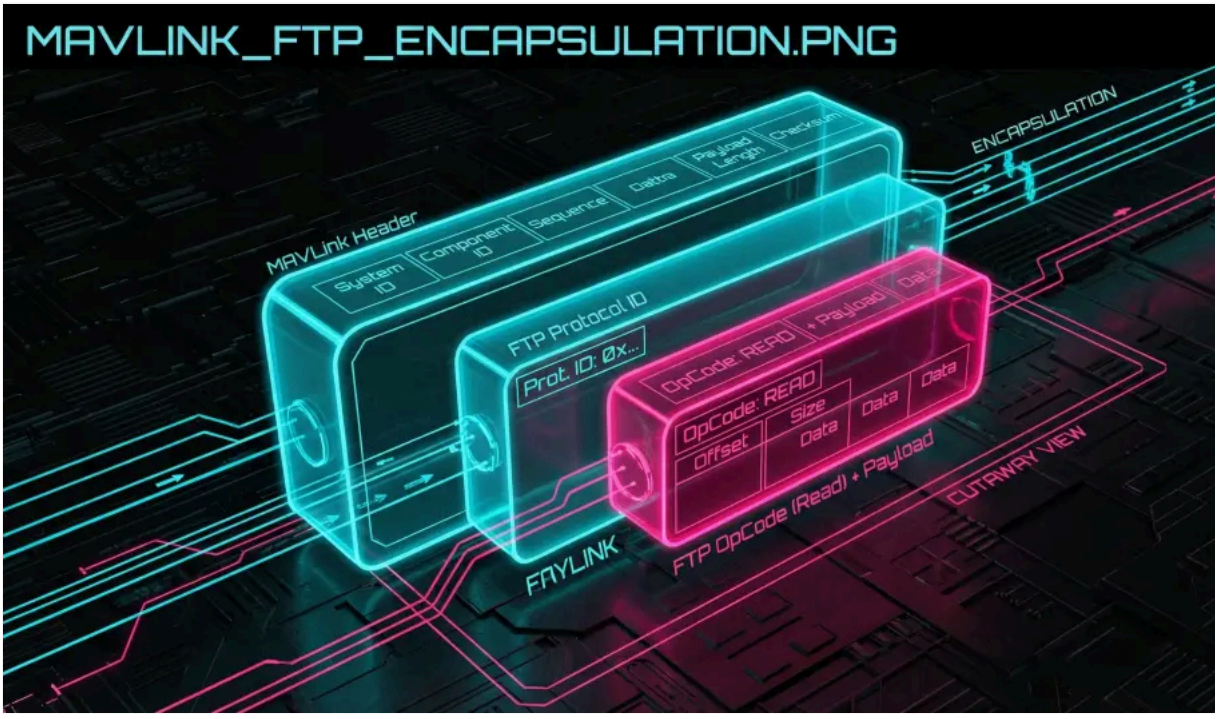
Filesystem access is a "Blocking" operation. If the main flight control loop tried to read a file from the SD card, the drone would freeze for 50ms and likely crash. This is why ArduPilot uses a **Producer-Consumer** model: the MAVLink thread produces a request, and a low-priority background thread consumes it and performs the slow work.

ARCHITECTURE (THE ENGINEER'S VIEW)

The core logic resides in `GCS_FTP.cpp`.

1. ENCAPSULATION

File operations are not sent as distinct MAVLink messages (like `READ_FILE`). Instead, they are encapsulated inside a generic `FILE_TRANSFER_PROTOCOL` message (ID #110).



- **Payload:** The payload is a raw byte array containing a custom "OpCode" header (Open, Read, Write, Terminate) and data.
- **Decoupling:** Because SD Card access is blocking and slow (10ms-100ms latency), ArduPilot **cannot** access the file in the main thread.
  - *Solution:* It queues the request into a ring buffer.
  - *Worker:* A separate **IO Thread** (`ftp_worker`) picks up the request, performs the blocking SD card operation, and queues the result back to the main thread.

2. BURST READ (THE SPEED HACK)

Standard FTP requires a "Ack" for every packet. This is too slow for 100MB log files over a radio link.

- **Mechanism:** `BurstReadFile` (OpCode 15).

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

- **Behavior:** The GCS requests "Read 100 chunks". ArduPilot enters a loop and blasts 100 packets back-to-back without waiting for an Ack between them.
- **Throttling:** It calculates a `burst_delay_ms` based on the link's estimated bandwidth to prevent flooding the buffer, but it pushes the link to its limit.

3. FILESYSTEM ABSTRACTION

The FTP server uses `AP_Filesystem` (`AP::FS()`). This means it works identically whether the storage is a physical SD Card (FATFS), an on-chip Flash chip (LittleFS), or a Posix file (SITL).

COMMON ISSUES & TROUBLESHOOTING

"LOG DOWNLOAD FAILED"

- **Cause:** Packet loss during Burst Read. Since there are no per-packet Acks, if a packet drops, the file is corrupted or the GCS detects a missing offset and aborts.
- **Fix:** Use a wired USB connection (reliable) instead of a telemetry radio (lossy).

"SLOW TRANSFER"

- **Cause:** Protocol overhead. Every 251-byte payload is wrapped in a 280-byte MAVLink packet. The effective throughput is often only 60\% of the baud rate.

SOURCE CODE REFERENCE

- **Protocol Logic:** `GCS_MAVLINK::handle_file_transfer_protocol()`
- **Worker Thread:** `GCS_MAVLINK::ftp_worker()`

PRACTICAL GUIDE: SPEEDING UP LOG DOWNLOADS

Trying to download a 500MB Dataflash log over a 57kbps telemetry radio will take days. MAVLink FTP is powerful, but physics still applies.

THE USB RULE

- **Always** use USB for log downloads if possible.
- **Speed:** USB achieves ~200-500 KB/s.
- **Radio:** SiK Radio achieves ~2 KB/s.
- **WiFi:** ESP8266 WiFi achieves ~300 KB/s (often faster than USB due to driver efficiency).

PROTOCOL TWEAKS (MISSION PLANNER)

If you *must* use a radio link (e.g., drone is on a roof):

1. Go to **Config/Tuning** → **Planner**.
2. Uncheck **"MAVLink FTP"**.
3. *Wait, what?* No, keep it checked. This guide is checking if you are paying attention. MAVLink FTP is *faster* than the legacy method.
4. Increase **"Attitude"** stream rate in the scheduler. ArduPilot prioritizes `ATTITUDE` packets. If you spam FTP bursts, you might choke the link.
5. **Real Tip:** Just pull the SD card. It's 100x faster.

MAVLINK2 SIGNING & SECURITY

EXECUTIVE SUMMARY

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





Standard MAVLink is unencrypted and unauthenticated. Anyone with a radio on the same frequency can inject commands (e.g., "Disarm"). **MAVLink2 Signing** adds an authentication layer. Every packet includes a cryptographic signature (SHA-256) generated using a secret key. If the signature is missing or invalid, the autopilot rejects the packet.

THEORY & CONCEPTS

1. CRYPTOGRAPHIC SIGNATURES (HMAC)

MAVLink2 signing uses a concept called **HMAC** (Hash-based Message Authentication Code). Instead of encrypting the data (hiding it), it signs the data (proving it's real).

- **The Key:** Both the Drone and GCS have a secret 32-byte key.
- **The Signature:** We hash the packet with the key. Only someone with the key can produce that specific hash.
- **Verification:** If the signature doesn't match the packet content, ArduPilot knows the packet was either corrupted by noise or injected by a malicious user.

2. REPLAY ATTACKS

A "Replay Attack" is when an attacker records a valid command (like "Disarm") and plays it back later. Even with a cryptographic signature, the packet *is* valid.

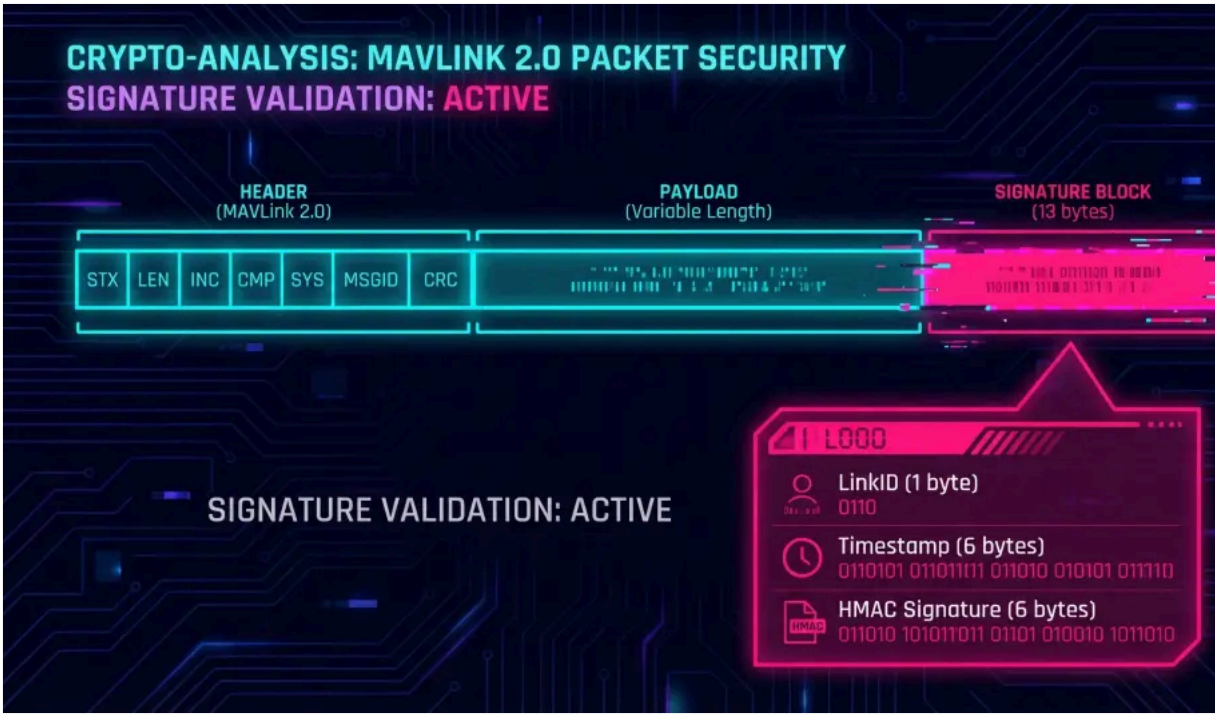
- **The Defense:** Timestamps. Every signed packet has a unique timestamp. If the timestamp is older than the last one received, ArduPilot rejects it as a replay.

ARCHITECTURE (THE ENGINEER'S VIEW)

The logic is split between the generated MAVLink headers (`mavlink_helpers.h`) and ArduPilot's key manager (`GCS_Signing.cpp`).

1. THE SIGNATURE BLOCK

MAVLink2 adds a 13-byte footer to signed packets:



- **Link ID (1 byte):** Identifies the communication channel (e.g., Radio 1, Wifi).

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

- **Timestamp (6 bytes):** 48-bit counter (microseconds since 2015). **Critical for Replay Protection.**
- **Signature (6 bytes):** The first 48 bits of the SHA-256 hash.

2. THE MECHANICS

1. **Signing (Sender):**
   - `Hash = SHA256(SecretKey + PacketContent + Timestamp)`.
   - The packet is transmitted with the timestamp and truncated hash.
2. **Verification (Receiver):**
   - Receiver calculates `ExpectedHash` using its local copy of `SecretKey`.
   - If `Hash == ExpectedHash`, the packet is authentic.
3. **Replay Protection:**
   - The receiver tracks the *Last Timestamp* seen for each Link ID.
   - If a new packet arrives with `Timestamp ⩽ LastTimestamp`, it is rejected as a **Replay Attack** (someone recorded an old "Disarm" command and is playing it back).

3. KEY MANAGEMENT

The 32-byte Secret Key is the root of trust.

- **Storage:** Keys are stored in the Flight Controller's persistent storage (FRAM/Flash).
- **Setup:** The key is usually generated by the Ground Control Station (GCS) and sent to the drone via the `SETUP_SIGNING` message.
   - *Constraint:* This message is only accepted when the vehicle is **Disarmed**.
- **Code Path:** `GCS_MAVLINK::handle_setup_signing()`.

COMMON ISSUES & TROUBLESHOOTING

"BAD SIGNATURE" / "LINK REJECTED"

- **Cause:** Key mismatch between GCS and Drone.
- **Fix:** Re-run the "Setup Signing" wizard in Mission Planner/QGC to sync a new key.

"REPLAY ERROR" AFTER REBOOT

- **Cause:** The GCS clock drifted or reset.
- **Fix:** The GCS usually handles this by negotiating a new timestamp offset, but sometimes a full restart of both sides is required.

SOURCE CODE REFERENCE

- **Key Logic:** `GCS_MAVLINK::load_signing_key()`
- **Message Handler:** `GCS_MAVLINK::handle_setup_signing()`

PRACTICAL GUIDE: ENABLING SIGNING

Security is useless if it's too hard to use. Here is the happy path.

STEP 1: GENERATE THE KEY (MISSION PLANNER)

1. Connect to the drone via USB (for reliability).
2. Press **Ctrl+F** (Temp Screen).
3. Click **"Mavlink Signing"**.
4. Click **"Setup Signing"**.
5. Mission Planner will generate a random 32-byte key and upload it to the drone.
6. **Important:** It will also save this key to your PC's `MissionPlanner.xml`.

STEP 2: SHARING THE KEY

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





If you want to use QGroundControl on your tablet, it needs the key too.

1. In the Mavlink Signing screen, click **"Show Key"**.
2. Copy the 64-character hex string.
3. In QGroundControl: **Application Settings** → **MAVLink** → **Signing Key**.
4. Paste the key.

STEP 3: VERIFICATION

1. Connect via Telemetry Radio.
2. Look at the console. You should see `GCS: Signing Active` or similar.
3. If you try to connect with a GCS that *doesn't* have the key, the drone will simply ignore it. The GCS will time out waiting for parameters.

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

CHAPTER 5: EKF FAILSAFES

THE EKF CORE: FUSION ARCHITECTURE

EXECUTIVE SUMMARY

The **Extended Kalman Filter (EKF)** is the mathematical brain of the autopilot. Its job is to calculate a single, trusted "Truth" (Attitude, Position, Velocity) by fusing data from noisy, contradictory sensors. It doesn't just average them; it builds a physics-based model of the vehicle and constantly corrects it based on sensor feedback.

THEORY & CONCEPTS

1. THE STATE OBSERVER

In control theory, an **Observer** is a mathematical model that guesses the state of a system (like a drone) based on its inputs (motor commands) and outputs (sensor readings). The EKF is a "Stochastic" observer, meaning it doesn't just guess where the drone is, it also calculates **how sure it is** about that guess.

2. SENSOR FUSION: THE "CONTRADICTORY" PROBLEM

- **The Gyro** is fast but drifts.
- **The GPS** is slow but absolute.
- **The Math:** The Kalman filter weight (Kalman Gain) acts like a dynamic volume knob. When the drone is moving fast, it turns up the volume on the Gyro. When the drone is hovering, it turns up the volume on the GPS. This "Optimal Weighting" is the secret to ArduPilot's legendary stability.

THE STATE VECTOR (THE "BRAIN")

ArduPilot's EKF3 tracks **24 Internal States**. It isn't just tracking "Where am I?", but "How broken are my sensors?".

1. **Attitude (4):** Quaternion defining rotation (Body to Earth).
2. **Velocity (3):** North, East, Down velocity.
3. **Position (3):** North, East, Down position.
4. **Gyro Bias (3):** The constant error drift of the gyroscopes.
5. **Accel Bias (3):** The constant error of the accelerometers.
6. **Earth Mag (3):** The magnetic field vector of the Earth (Declination/Dip).
7. **Body Mag (3):** The magnetic interference of the vehicle itself.
8. **Wind (2):** North, East wind velocity (estimated by comparing Airspeed vs Groundspeed).

CONTROL ARCHITECTURE (ENGINEER'S VIEW)

The EKF runs in a strict **Prediction-Correction** cycle.

1. PREDICTION (THE STRAPDOWN)

- **Trigger:** New IMU data (Gyro/Accel).
- **Rate:** Fast (~400Hz or IMU rate).
- **Logic:** "If I was at location X and moving at velocity V, and I accelerated by A for 2.5ms, where am I now?"
- *Code Path:* `UpdateStrapdownEquationsNED()`.
- *Result:* The EKF propagates the state forward in time. This provides the low-latency estimate used for stabilization loops.

2. CORRECTION (THE FUSION)

- **Trigger:** Slow sensor data arrives (GPS, Baro, Mag).
- **Rate:** Variable (GPS @ 5-10Hz, Baro @ 50Hz).

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- **Logic:** "My Prediction says I'm at X. The GPS says I'm at Y. The difference is the **Innovation**."
- **Update:** The EKF calculates how much to trust the GPS vs its own Prediction (based on covariance). It then "nudges" the states (Position, Velocity, and Biases) to reduce the error.
- *Code Path:* `UpdateFilter()`.

WHY IS THIS BETTER THAN COMPLEMENTARY FILTERS?

A simple filter blends Gyro + Accel. The EKF blends Gyro + Accel + GPS + Mag + Airspeed + Baro + Rangefinder + Optical Flow. Crucially, it estimates **Sensor Biases**. If the EKF realizes the GPS is consistently 2m to the right of the prediction, it might realize the Compass is wrong (causing the prediction to drift), or it might realize the GPS has a glitch.

SOURCE CODE REFERENCE

- **Core Loop:** `NavEKF3_core::UpdateFilter()`
- **State Definition:** `AP_NavEKF3_core.h`

EKF LANE SWITCHING & REDUNDANCY

EXECUTIVE SUMMARY

Modern flight controllers (like the Cube Orange) have multiple redundant IMUs (Accelerometers/Gyros). ArduPilot leverages this by running **Multiple EKF Instances** (Lanes) in parallel. Each Lane uses a different IMU. The autopilot constantly monitors the "Health" of each Lane and automatically switches to the best one if the primary sensor fails or produces inconsistent data.

THEORY & CONCEPTS

1. VOTING SYSTEMS & FAIL-OPERATIONAL LOGIC

In high-reliability engineering, **Redundancy** is not enough; you need **Isolation**. If you have two sensors and they disagree, you can't fly safely (which one is right?). By using **Three Lanes**, ArduPilot implements a "Majority Vote" system. If one Lane (one IMU) fails, the other two still agree, allowing the system to be "Fail-Operational"—it continues to fly safely even after a hardware failure.

2. COMMON MODE FAILURES

Lane switching protects against **Sensor Failure** (stuck gyro), but not against **External Failure**. For example, if you fly through a massive magnetic field, all three compasses might fail in the same way. This is a "Common Mode Failure." The EKF errorScore tracks the *consistency* of each lane; if all lanes have high error scores, the system knows the problem is external to the flight controller.

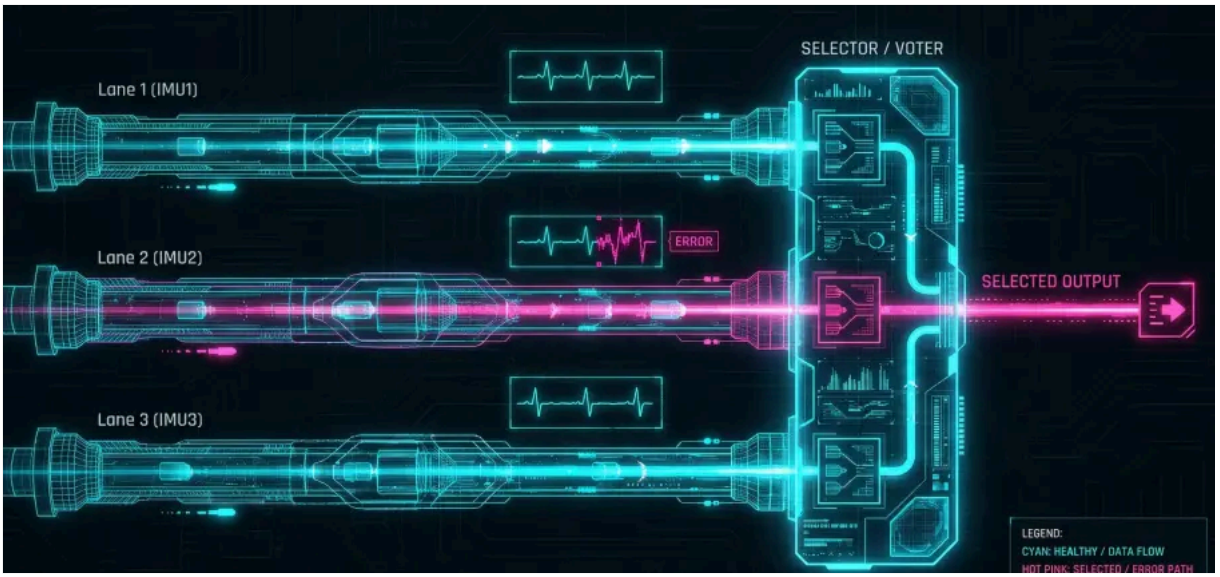
ARCHITECTURE (THE ENGINEER'S VIEW)

The logic is managed by the `AP_NavEKF3` frontend.

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY







1. PARALLEL LANES

- **Configuration:** `EK3_IMU_MASK` determines which IMUs are used.
- **Execution:** Up to 3 independent EKF cores run simultaneously (`NavEKF3_core`). They all fuse the same GPS/Baro data but use different IMU data.
- **Independence:** Because they rely on different physical sensors, a mechanical failure (stuck gyro) or aliasing glitch in IMU1 will corrupt Lane 1 but NOT Lane 2.

2. THE ERROR SCORE

Each core calculates a normalized **Error Score** (`0.0` to `1.0`+).

- **Metric:** It is derived from the **Test Ratios** of the sensor innovations (Velocity, Position, Height, Mag, Airspeed).
- *Formula (simplified):* `Score = MAX(Vel_Innovation, Pos_Innovation, Hgt_Innovation)`.
- **Meaning:**
  - `0.0 - 0.5`: Healthy.
  - `> 1.0`: The EKF is rejecting sensor data (inconsistent).
  - *Code Path:* `NavEKF3_core::errorScore()`.

3. THE SWITCHING LOGIC

The frontend compares the Error Scores of all active lanes.

- **The Threshold:** A switch occurs if the Active Lane's score exceeds the threshold (`1.0`) AND another lane has a significantly better score.
- **The Hysteresis:** To prevent rapid toggling, the alternative lane must be better by a margin (`BETTER_THRESH`).
- **The Failsafe Hook:** Before triggering an "EKF Failsafe" (Land/RTL), the vehicle calls `checkLaneSwitch()`. If a healthy lane exists, it switches lanes *instead* of declaring a failsafe.

COMMON ISSUES & TROUBLESHOOTING

"LANE SWITCH" MESSAGE IN LOG

- **Cause:** One IMU disagreed with the GPS/Baro significantly more than the others.
- **Analysis:** Plot `XKF1.Err`, `XKF2.Err`, `XKF3.Err`. If `XKF1` spikes but `XKF2` stays low, IMU1 likely suffered vibration aliasing or a hardware fault.

"UNHEALTHY AHRS" / "GYROS INCONSISTENT"

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

- **Cause:** The gyros on startup matched poorly. ArduPilot refuses to arm if the redundant sensors disagree.

SOURCE CODE REFERENCE

- **Lane Manager:** `NavEKF3::checkLaneSwitch()`
- **Scoring:** `NavEKF3_core::errorScore()`

PRACTICAL GUIDE: ANALYZING EKF HEALTH

The "EKF Lane Switch" message often scares pilots. Here is how to verify if it was a real hardware failure or just a glitch.

THE FORENSIC METHOD

1. Open the `.bin` log in Mission Planner.
2. Search for **XKF1**, **XKF2**, and **XKF3** (these correspond to Lane 1, 2, and 3).
3. Plot `Err` **(Error Score)** for all three lanes on the same graph.
4. **Interpret:**
    - **Scenario A (One Bad Apple):** Lane 1 spikes to `1.5` while Lane 2 and 3 stay at `0.1`.
        - *Conclusion:* IMU1 is faulty or suffering vibration aliasing. The system worked correctly by switching to Lane 2.
    - **Scenario B (The Global Crisis):** All three lanes spike to `1.0` simultaneously.
        - *Conclusion:* This is NOT a sensor fault. It is an external consistency issue (e.g., Compass interference, GPS glitch, or bad vibration affecting *all* IMUs). Switching lanes won't help here.

HOW TO FORCE A LANE SWITCH (BENCH TEST)

You can verify the redundancy system on the bench (Props OFF!).

1. Connect via USB and monitor the "Messages" tab.
2. Take a strong magnet.
3. Move it close to the flight controller (disturbing the internal mag or causing gyro bias).
4. You will likely see **"EKF3 Lane Switch 1"** messages as the EKF detects the disturbance on one sensor (due to physical placement) before the others, or as it rejects the magnetic anomaly on the active lane and tries another.
5. *Note:* This confirms the voting logic is active.

For more details, see the ArduPilot Wiki: EKF3 Handling.

EKF VARIANCE & INNOVATIONS

EXECUTIVE SUMMARY

When the GCS screams **"EKF Variance"** or **"Velocity Variance"**, it means the autopilot is confused. Specifically, it means the sensor data (GPS/Compass) disagrees with the internal physics model (Inertial Prediction) by a margin that exceeds the allowed safety threshold (Gate).

THEORY & CONCEPTS

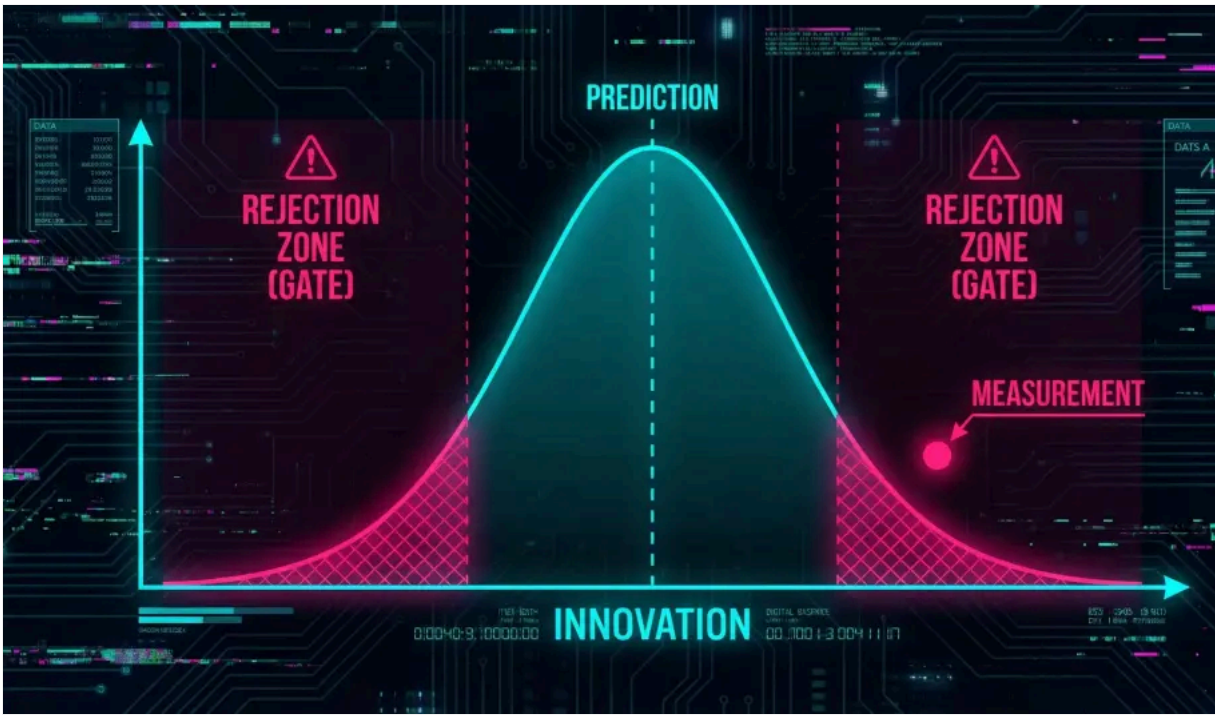
1. PROBABILITY DENSITY FUNCTIONS (THE BELL CURVE)

Every sensor measurement is a random variable. The EKF assumes that sensor noise follows a **Gaussian Distribution** (a Bell Curve).

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY







- **Innovation:** This is the distance from the center of the bell curve.
- **Gate (e.g., 5 Sigma):** This defines the "cutoff" at the edges of the curve. If a measurement falls outside this gate, there is a 99.99\% chance it is a glitch, not real movement.

2. THE P MATRIX (COVARIANCE)

The EKF maintains a massive 24×24 matrix called the **Covariance Matrix (P)**. It stores the "Confidence" for every state.

- **Hovering:** Confidence in Position is high.
- **GPS Glitch:** Confidence drops. The EKF "blooms" its uncertainty, waiting for the GPS to stabilize before trusting it again.

CORE CONCEPTS (THE ENGINEER'S VIEW)

1. INNOVATION (THE ERROR)

Innovation is the difference between "What I predicted" and "What I measured".

- *Formula:* `Innovation = Measurement - Prediction`
- *Example:* The EKF predicts the drone is at `X=10m`. The GPS reports `X=12m`. The Innovation is `2m`.

2. VARIANCE (THE UNCERTAINTY)

Variance describes how much we trust a value.

- *Measurement Variance:* How noisy is the GPS? (Reported by GPS HDOP).
- *State Variance:* How sure is the EKF of its own position? (Grows over time if no GPS).

3. THE "TEST RATIO" (THE TRIGGER)

To decide if a sensor is glitching or if the drone is actually moving, the EKF calculates a **Test Ratio**.

- *Formula:* `Ratio = Innovation^2 / (Variance * Gate^2)`
- *Interpretation:*
  - `Ratio < 1.0`: The measurement is consistent. Fuse it!
  - `Ratio > 1.0`: The measurement is wildly unlikely. **Reject it!** (This triggers the "Variance" alert).

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

THE INNOVATION GATES

You can tune how strict the EKF is using the **Gate Parameters**.

- `EK3_VEL_I_GATE` : Velocity innovation gate (default 500 = 5 sigma).
- `EK3_POS_I_GATE` : Position innovation gate.
- `EK3_HGT_I_GATE` : Height innovation gate.

TUNING TRADE-OFF:

- **Lower Gate (e.g., 300):** The EKF is stricter. It rejects bad GPS data faster (good for safety), but might falsely reject good GPS data during aggressive maneuvers (bad for reliability).
- **Higher Gate (e.g., 800):** The EKF is permissive. It accepts sloppy GPS data, but if the GPS glitches hard, the drone might "run away" to follow the glitch.

TROUBLESHOOTING "VARIANCE" ERRORS

1. **Compass Variance:** The Compass points North, but the GPS says we are moving East. Innovation spikes.
   - *Fix:* Calibrate Compass or check for interference.
2. **Velocity Variance:** The GPS says we stopped, but the Accelerometer says we are still braking.
   - *Fix:* Check vibration levels (`VIBE` log). High vibration makes the Accelerometer "lie," causing the prediction to drift.

SOURCE CODE REFERENCE

- **Calculation Logic:** `NavEKF3_core::CalculateVelInnovationsAndVariances()`

GPS GLITCH PROTECTION

EXECUTIVE SUMMARY

GPS is imperfect. Multipath errors (signals bouncing off buildings) can cause the GPS reported position to jump 10 meters instantly. If the flight controller trusted this blindly, the drone would violently roll to "correct" its position, crashing into the obstacle. **Glitch Protection** allows the EKF to reject these sudden jumps and transition to inertial flight (Dead Reckoning) until the GPS stabilizes.

THEORY & CONCEPTS

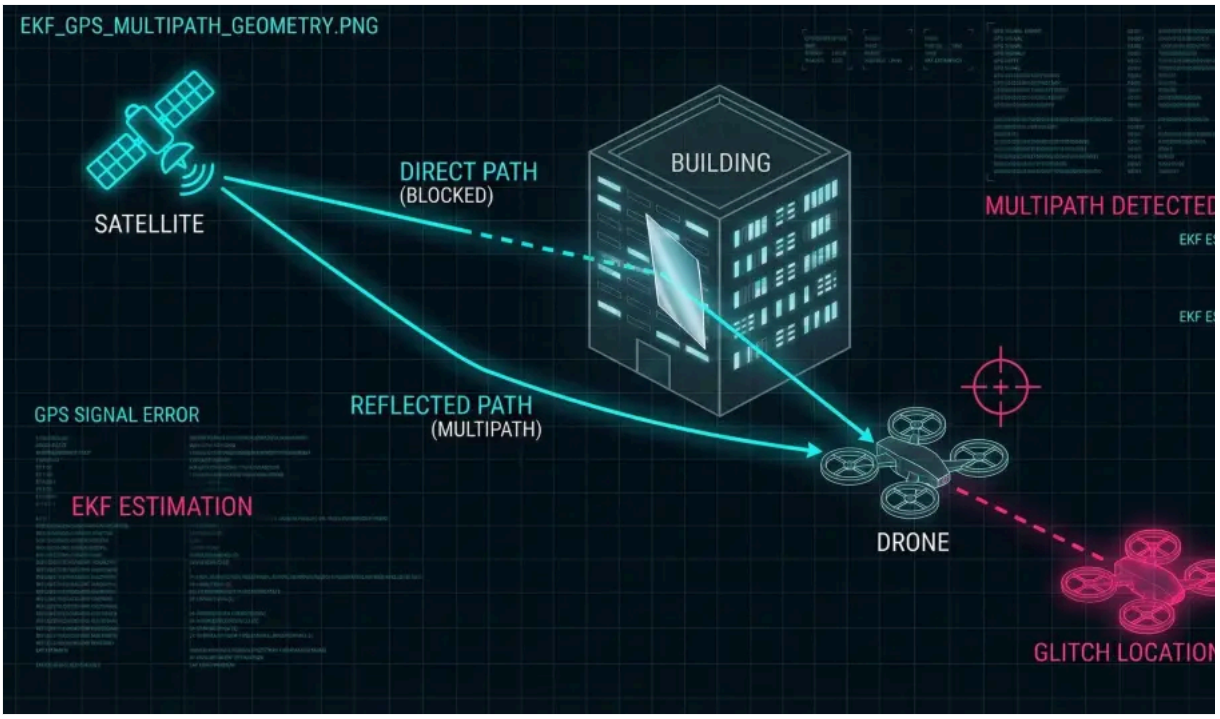
1. MULTIPATH INTERFERENCE

The biggest cause of GPS glitches is **Multipath**. This happens when the satellite signal bounces off a building or the ground before reaching the antenna.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY







- **The Physics:** Radio waves travel at the speed of light. If the signal bounces, it travels a longer path.
- **The Error:** The GPS receiver thinks the satellite is further away than it is. This shifts the calculated position.
- **The Glitch:** As you fly past the building, the bounce angle changes, causing the position to "jump."

2. TIME-OF-FLIGHT VS. INERTIAL CERTAINTY

The EKF knows exactly how much force the drone is applying (via IMU). If the GPS says the drone moved 10m East, but the IMU says we only accelerated by 0.1G, the EKF knows the GPS is lying. It trusts its **Inertial Certainty** over the "Glitched" GPS time-of-flight.

DETECTION ARCHITECTURE (ENGINEER'S VIEW)

The EKF3 detection logic in `SelectVelPosFusion()` is a two-layer defense.

1. THE INNOVATION CHECK (IMMEDIATE DEFENSE)

- **Concept:** "Physics says you can't teleport."
- **Mechanism:** The EKF compares the GPS Measurement against the IMU-based Prediction.
- **Trigger:** If the difference (Innovation) exceeds the gate (`EK3_POS_I_GATE`, default 5 sigma), the measurement is flagged as invalid.
- **Result:** The GPS data is **completely ignored** for that frame. The EKF relies 100\% on the IMU (Dead Reckoning).

2. THE GLITCH RADIUS (LONG-TERM DEFENSE)

What if the GPS drifts slowly (sub-gate) but persistently? Or what if the IMU drifts?

- **Parameter:** `EK3_GLITCH_RAD` (default 25m).
- **Concept:** "I might be wrong, but I'm not *that* wrong."
- **Mechanism:** If the estimated uncertainty (Variance) of the EKF's position grows larger than `EK3_GLITCH_RAD`, the EKF admits defeat.
- **Result:** It performs a **Position Reset**, snapping its internal state to the GPS coordinate. This prevents the "Runaway Drone" scenario where the EKF thinks it is right and the GPS is wrong forever.

DEAD RECKONING

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

When a Glitch is detected, the drone enters **Dead Reckoning**.

- **Behavior:** It uses the Accelerometers to calculate velocity and position.
- **Duration:** High-quality IMUs (like in the Cube) can maintain position accuracy for 10-30 seconds. Cheap IMUs drift faster.
- **Wind:** Without GPS, the drone cannot estimate wind. It assumes the last known wind vector is constant.

KEY PARAMETERS

PARAMETER	DEFAULT	DESCRIPTION
`EK3_POS_I_GATE`	500	(5 sigma) The threshold for rejecting a position jump. Lower = Stricter (safer against glitches, but risks rejecting good data during 10G turns).
`EK3_GLITCH_RAD`	25	(m) The maximum "Trust Bubble" the EKF will build around itself. If the GPS is outside this bubble for too long, the EKF resets.

SOURCE CODE REFERENCE

- **Fusion Logic:** `NavEKF3_core::SelectVelPosFusion()`

PRACTICAL GUIDE: THE ALTHOLD RESCUE

If you see "EKF Variance" or "GPS Glitch" on your HUD, your drone is confused.

THE PROBLEM

- **Mode:** Loiter (GPS).
- **Situation:** The drone starts leaning 20 degrees sideways, but the HUD says it is perfectly flat and stationary.
- **Diagnosis:** The GPS has drifted (multipath). The EKF thinks the *world* moved, so it is leaning to "hold position" against a phantom movement.

THE FIX

1. **Switch to AltHold:** Immediately.
    - *Why?* AltHold ignores GPS. It uses the Barometer (Z) and the Gyro (Pitch/Roll).
    - *Result:* The drone will level out instantly. It will drift with the wind, but it will stop fighting the ghost GPS data.
2. **Wait:** Let the GPS stabilize (usually takes 10-30 seconds).
3. **Check HUD:** When the "EKF" status turns Red to White, you can switch back to Loiter.

COMPASS FUSION & MAGNETIC INTERFERENCE

EXECUTIVE SUMMARY

Magnetometers (Compasses) are the most fragile sensors on a drone. They are easily corrupted by power lines, metal bridges, or even the drone's own motors. The EKF3 Compass Fusion system is designed to tolerate significant magnetic interference by estimating the **Magnetic Field Vector** (Earth + Body) rather than just trusting the raw compass heading.

THEORY & CONCEPTS

1. THE EARTH AS A BAR MAGNET (WMM)

The Earth's magnetic field is not uniform.

- **WMM (World Magnetic Model):** ArduPilot includes a compressed version of the NOAA/NGA magnetic model.

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- **The Check:** The EKF uses this model to know exactly what the magnetic field *should* look like at your current GPS location (Inclination, Declination, and Field Strength). If the compass reports something wildly different, ArduPilot knows you are near a metal object or that the sensor is broken.

2. DIP ANGLE (MAGNETIC INCLINATION)

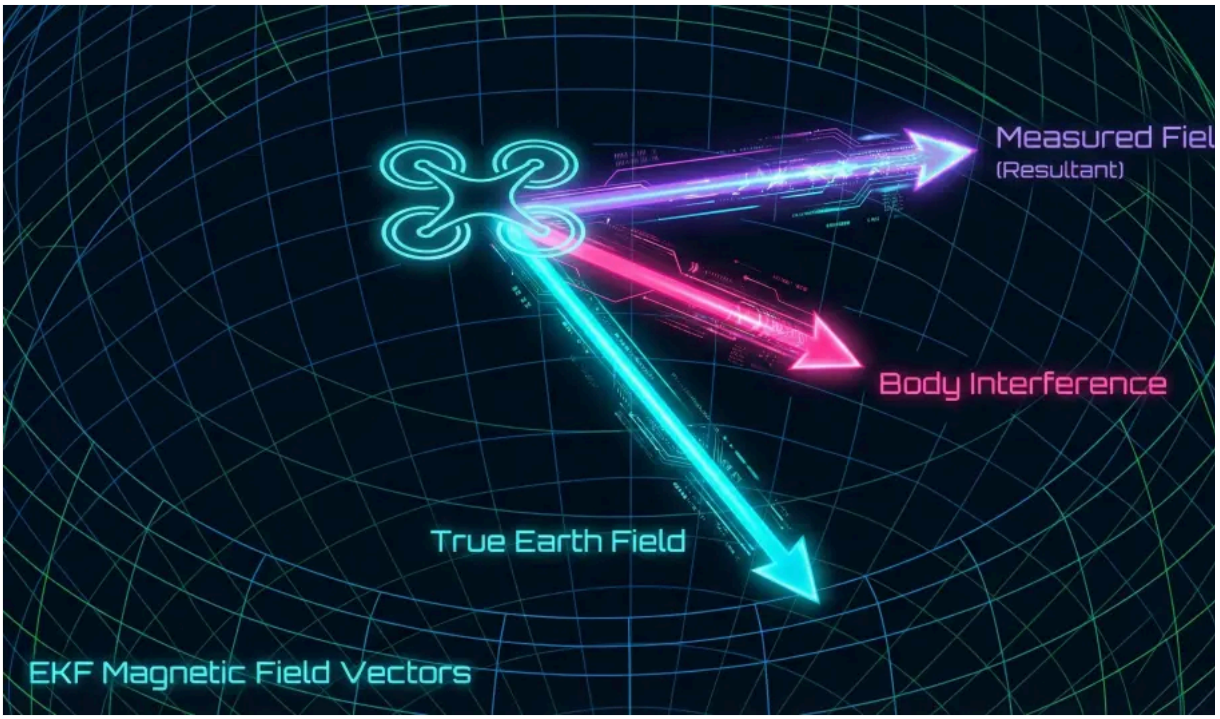
Near the Equator, the magnetic field is horizontal. Near the Poles, it points straight into the ground.

- **The Issue:** If your drone is not level, a vertical magnetic field can look like a horizontal one, causing a heading error.
- **The Fusion:** The EKF uses the IMU (Accelerometers) to know "Down" and then projects the magnetic vector into the horizontal plane to find "North."

CORE CONCEPTS (THE ENGINEER'S VIEW)

1. 3-AXIS FIELD ESTIMATION

Simple autopilots just use `atan2(MagY, MagX)` to get heading. ArduPilot's EKF is smarter.
It estimates 6 states related to magnetism:



- **Earth Field (3):** The true North/Down vector of the Earth at your location.
- **Body Field (3):** The interference attached to the drone (e.g., a magnetized screw next to the sensor).
- *Result:* The EKF learns to subtract the Body Field from the measurement to find the true Earth Field.

2. MAG ANOMALY DETECTION

The EKF compares the **Measured Field Length** against the **Expected Earth Field** (from the World Magnetic Model lookup table).

- **Takeoff Check:** If the field strength changes significantly as you throttle up (motor interference) or as you lift off (ground interference like rebar), the EKF flags a **Mag Anomaly**.
- **Response:** It stops fusing compass data temporarily and relies on Gyros (dead reckoning) until the field stabilizes.

3. EKF-GSF (COMPASS-LESS YAW)

What if the compass fails completely mid-flight?

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

- **Mechanism:** The **Emergency Yaw Reset**.
- **Algorithm:** The EKF runs a bank of parallel estimators (Gaussian Sum Filter) that try to guess the Yaw based on **Velocity**. If you fly forward, the GPS velocity vector *must* be the direction of travel (for a multicopter/plane in coordinated flight).
- **Result:** If the compass innovation grows too large (bad compass), the EKF resets its Yaw to match the GPS track. This saves the drone from the dreaded "Toilet Bowl" crash.

TROUBLESHOOTING MAG ISSUES

- **"Compass Variance":** The EKF has rejected the compass data.
  - *Cause:* Metal in the ground, or bad calibration.
- **"Mag Anomaly":** Sudden change in field strength.
  - *Cause:* Flying near a power pylon or bridge.
- **Toilet Bowling:** The drone circles while trying to hover.
  - *Cause:* The EKF is accepting bad compass data. The "Heading" is wrong, so when the drone tries to stop "North" drift, it actually accelerates "East", creating a spiral.

4. INDOOR FLIGHT & THE TOILET BOWL EFFECT

Indoor environments are "Mag-Toxic" due to rebar in concrete and electrical infrastructure. Without a clear view of the sky, standard GPS cannot correct compass-induced drift.

- **Deep Dive:** For a technical analysis of why this happens specifically in stadium and warehouse environments, see the **Toilet Bowl Effect Guide** on the MAVLink GPS site.

KEY PARAMETERS

PARAMETER	DEFAULT	DESCRIPTION
`EK3_MAG_CAL`	3	(0-6) Determines when the EKF is allowed to learn magnetometer offsets in-flight.
`EK3_MAG_I_GATE`	300	(3 sigma) Strictness of the innovation check.
`COMPASS_LEARN`	0	Legacy method. Prefer EKF learning (`EK3_MAG_CAL`).

SOURCE CODE REFERENCE

- **Fusion Logic:** `NavEKF3_core::FuseMagnetometer()`
- **Yaw Fusion:** `NavEKF3_core::fuseEulerYaw()`

PRACTICAL GUIDE: SOLVING TOILET BOWLING (COMPASS MOTOR COMPENSATION)

If your drone holds position well at idle but starts spiraling ("Toilet Bowling") as you fly faster or punch the throttle, your power distribution is generating a magnetic field that blinds the compass.

THE FIX: MAGFIT (MOTOR COMPENSATION)

You need to tell ArduPilot how much interference your motors generate so it can subtract it.

1. **Fly a "MagFit" Figure-8:**
   - Find a calm day.
   - Take off and fly aggressive fast circles or figure-8s. You need to use high throttle and high current.
   - Fly for about 1 minute.
   - Land and Disarm.
2. **Analyze the Log:**
   - Download the `.bin` log.
   - In Mission Planner, press **Ctrl+F** → **Mavlink Inspector** (wait, no) → **MagFit**.
   - Load your log.
   - The tool will calculate the interference curve.
3. **Apply the Parameters:**

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- If MagFit finds a correlation, it will suggest new values for `COMPASS_MOT_x` (x/y/z offsets).
- Accept the changes.
- Set `COMPASS_MOTCT = 2` (Use Current) if you have a current sensor, or `1` (Use Throttle) if you don't.

WHY NOT JUST MOVE THE COMPASS?

Moving the compass/GPS 5cm higher on a mast is often more effective than any software calibration. Magnetic fields decay with the **cube of the distance** ($1/r^3$). A small move makes a huge difference.

WIND ESTIMATION & DRAG FUSION

EXECUTIVE SUMMARY

Knowing the wind speed is critical for accurate navigation, especially for Return-to-Launch calculations ("Do I have enough battery to fight this headwind?"). While Planes use Pitot tubes, Multicopters typically don't. ArduPilot solves this using **Drag Fusion**: estimating wind by analyzing how much the drone has to lean to hold position or move.

THEORY & CONCEPTS

1. AERODYNAMIC DRAG MODELING

Every object moving through air creates a drag force.

- **Form Drag:** The force required to push the air out of the way. It increases with the **square of speed**.
- **Momentum Drag:** The force required to ingest air into the propellers and accelerate it. It increases **linearly with speed**.
- **The Physics:** By measuring the "Lean Angle" required to remain stationary, ArduPilot effectively turns the entire drone into a giant **Anemometer** (wind sensor).

2. RELATIVE AIRSPEED

Inertial sensors (IMUs) measure acceleration relative to the ground. But drag is a function of acceleration relative to the **Air**.
The EKF uses the difference between its **Inertial Velocity State** and its **Wind Velocity State** to calculate the **Relative Airspeed**, which is the variable used in the drag math.

ARCHITECTURE (THE ENGINEER'S VIEW)

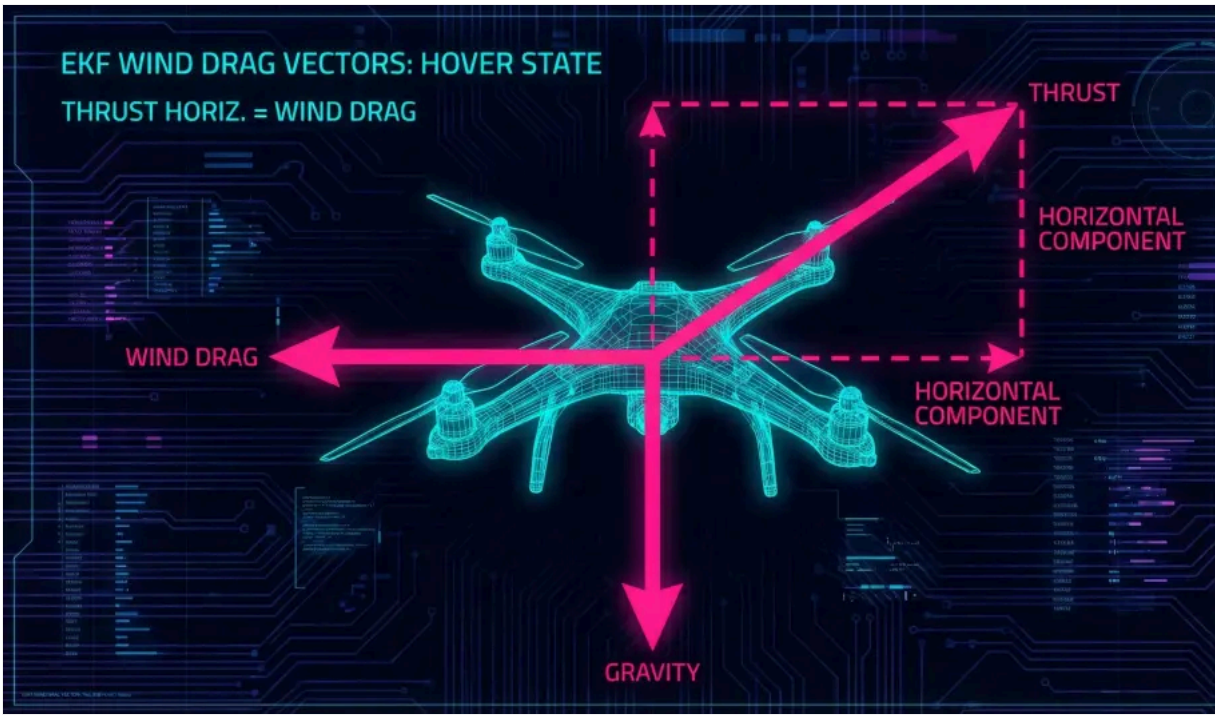
1. THE PHYSICS MODEL

A multicopter is a "Bluff Body" (a brick flying through air).

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY







- **Drag Force:** Air resistance pushes against the body.
- **Lean Angle:** To hold position against wind, the drone must tilt into the wind.
- **The Logic:** If the drone is tilting 5 degrees North but the GPS says Velocity is 0, the EKF infers a strong Wind from the North.

2. DRAG FUSION (`FUSEDRAGFORCES`)

The EKF uses a drag model to predict the acceleration the drone *should* feel based on its airspeed.

- **Inputs:**
  - IMU Accelerometers (Measured Force).
  - Vehicle Velocity (from GPS/Optical Flow).
  - Wind Velocity (State Estimate).
- **Correction:** If the measured acceleration doesn't match the prediction, the EKF updates its **Wind Velocity Estimate** (States 22 & 23) to resolve the discrepancy.
- *Code Path:* `NavEKF3_core::FuseDragForces()`.

3. AIRSPEED FUSION (`FUSEAIRSPEED`)

For vehicles with a Pitot tube (Planes), the logic is simpler.

- **Measurement:** True Airspeed (TAS).
- **Calculation:** `Wind_Vector = Ground_Velocity_Vector - Air_Velocity_Vector`.
- *Code Path:* `NavEKF3_core::FuseAirspeed()`.

WHY THIS MATTERS

- **Dead Reckoning:** If GPS fails, the drone must fly blind using accelerometers. If the EKF has a good Wind Estimate, it can compensate for wind drift even without GPS, lasting much longer before crashing.
- **Landing Accuracy:** Knowing the wind allows the drone to fight drift more aggressively during precision landings.

KEY PARAMETERS

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

PARAMETER	DEFAULT	DESCRIPTION
`EK3_DRAG_BCOEF_X`	0	Ballistic Coefficient (Bluff Body Drag). Determines how much the body "catches the wind". Needs tuning for accurate wind estimation on copters.
`EK3_DRAG_MCOEF`	0	Momentum Drag Coefficient. Account for the "intake momentum" of air going into the propellers.
`EK3_ENABLE`	1	Must be enabled for any of this to work.

SOURCE CODE REFERENCE

- **Fusion Logic:** `AP_NavEKF3_AirDataFusion.cpp`

OPTICAL FLOW FUSION

EXECUTIVE SUMMARY

Optical Flow is the primary method for non-GPS navigation (FlowHold) and precision landing. The sensor reports the "angular velocity" of the ground texture. The EKF fuses this with altitude data to estimate the vehicle's horizontal velocity.

THEORY & CONCEPTS

1. VISUAL ODOMETRY

Visual Odometry is the process of estimating movement by analyzing changes in camera images.

- **The Math:** By tracking the position of a feature (like a rock or a rug pattern) across multiple frames, the sensor calculates a 2D vector of movement in pixels.
- **The Problem:** The sensor doesn't know how big a pixel is in "meters."
- **The Scaler:** This is why Altitude is required. One pixel of movement at 1 meter height represents a much smaller physical distance than one pixel at 10 meters height.

2. RELATIVE VS. ABSOLUTE

Optical flow provides **Relative Position**. It knows you moved 1 meter, but it doesn't know *where* you are on the planet. This is why Flow-only flight (no GPS) will always drift slowly over time (Dead Reckoning error accumulation).

CORE CONCEPTS (THE ENGINEER'S VIEW)

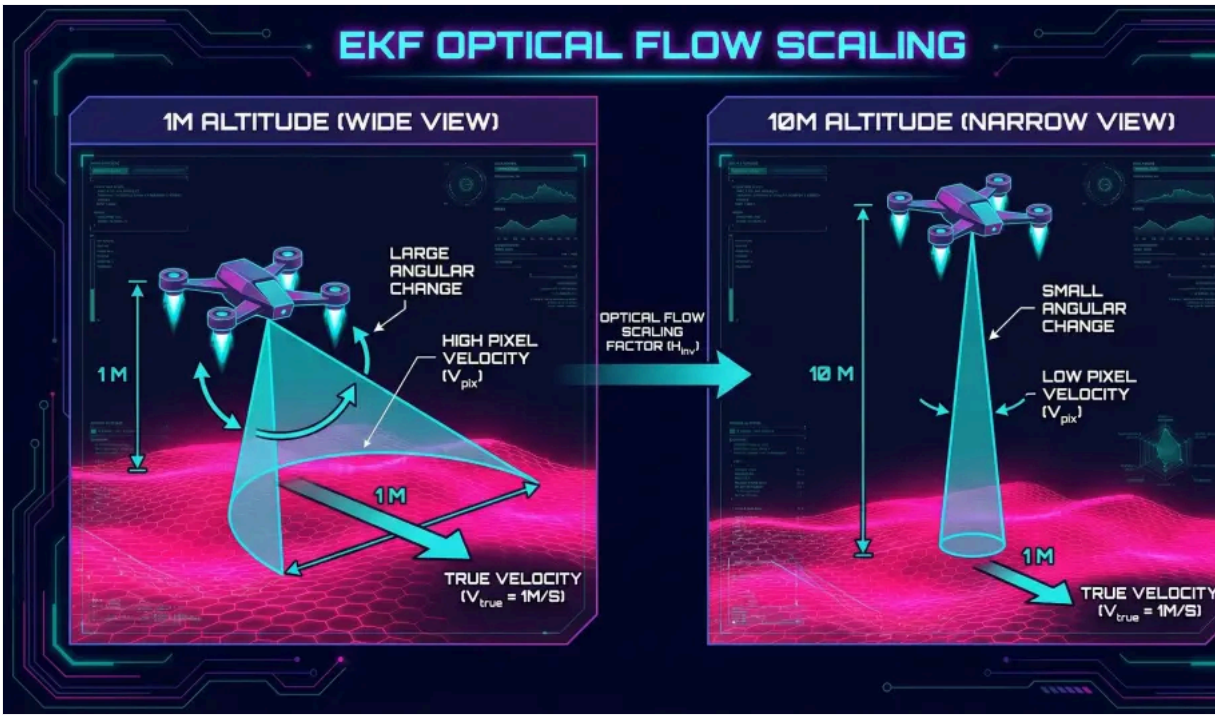
1. THE SCALING PROBLEM

The flow sensor sees "pixels per second" (Angular Rate).

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY







- **Formula:** `Velocity = Angular_Rate * Distance_To_Ground`.
- **The Issue:** If you don't know the distance (Altitude), you can't know the speed.
  - 10 rad/s at 1m height = 10 m/s.
  - 10 rad/s at 10m height = 100 m/s.
- **Correction:** The EKF must have a reliable `terrainState` (Height AGL).

2. TERRAIN STATE ESTIMATION (`ESTIMATETERRAINOFFSET`)

The EKF tracks a specific state for "Terrain Height" (relative to the EKF origin).

- **Source:** Rangefinder (Lidar/Sonar) is the primary source.
- **Logic:** `Height_AGL = EKF_Alt - Terrain_Alt`.
- **Estimation:** As the drone flies, the EKF updates the Terrain Altitude.
  - *Flat Ground:* The Terrain Alt stays constant.
  - *Sloped Ground:* If `EK3_TERR_GRAD` is set > 0, the EKF expects the terrain to change as the drone moves, increasing the uncertainty of the terrain state.

3. FUSION LOGIC (`FUSEOPTFLOW`)

- **Prediction:** The EKF predicts what the flow *should* be based on its current estimated Velocity and Height.
  - `Predicted_Flow = EKF_Velocity / EKF_Height_AGL`.
- **Correction:** It compares this to the *Measured Flow* (corrected for body rotation).
- **Update:** The difference (Innovation) is used to correct the **Velocity** states.

WHY RANGEFINDERS ARE CRITICAL

While the EKF *can* estimate scale without a rangefinder (using GPS or IMU integration), it is extremely fragile.

- **Without Lidar:** The EKF is guessing the scale factor. If it guesses wrong, a small drift becomes a massive acceleration command ("Runaway").
- **With Lidar:** The scale factor is mathematically locked. The velocity estimate becomes rock solid.

KEY PARAMETERS

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

PARAMETER	DEFAULT	DESCRIPTION
`EK3_FLOW_USE`	1	0=Disabled, 1=Use for Velocity, 2=Use for Position Holding.
`EK3_TERR_GRAD`	0.1	(m/m) Expected terrain slope. 0.1 = 10\% slope. Higher values make the EKF adapt faster to hills but maybe noisier.
`FLOW_POS_X/Y/Z`	0	Position of the sensor relative to COG. Critical for compensating for lever-arm effects during rotation.

SOURCE CODE REFERENCE

- **Terrain Estimator:** `NavEKF3_core::EstimateTerrainOffset()`
- **Flow Fusion:** `NavEKF3_core::FuseOptFlow()`

TERRAIN ESTIMATION (EKF)

EXECUTIVE SUMMARY

For precision landing and optical flow, the drone must know its height above the ground (AGL), not just its height above sea level (ASL). The EKF maintains a dedicated **Terrain State** that tracks the ground level relative to the Home Altitude. This allows the EKF to seamlessly fuse Rangefinder data with Barometer data.

THEORY & CONCEPTS

1. LASER TIME-OF-FLIGHT (LIDAR)

Most modern rangefinders use **Lidar**.

- **The Physics:** The sensor sends a pulse of light and measures the time it takes to return. Since light travels at a constant speed, the distance is calculated with millimeter precision.
- **The Difference:** Unlike a Barometer (which measures air pressure weight), Lidar measures physical distance.
- **Terrain Persistence:** ArduPilot doesn't just use the current Lidar reading; it maintains a "Terrain State." This means if you fly over a hole, the EKF knows the ground just "dropped" but the drone's altitude (relative to takeoff) is still the same.

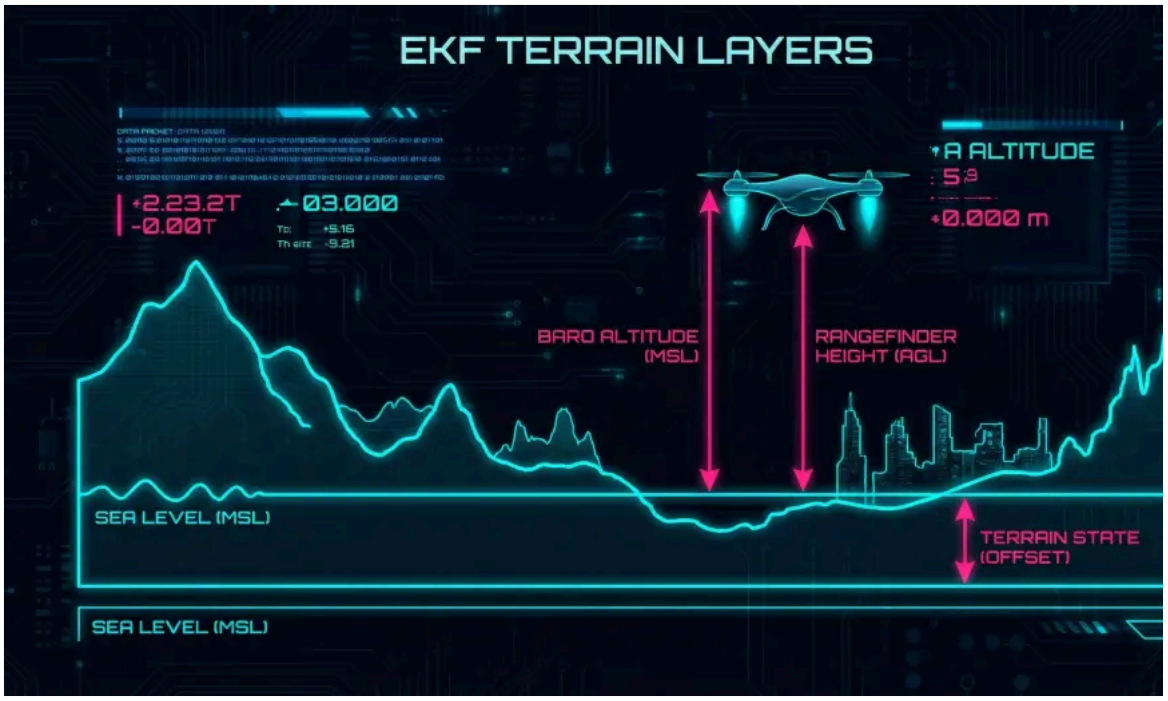
2. ABOVE GROUND LEVEL (AGL) VS. MEAN SEA LEVEL (MSL)

- **MSL:** Your height relative to the ocean (used by planes).
- **AGL:** Your height relative to the dirt below you (used by drones for landing).

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY







- *The EKF Role:* The EKF tracks MSL as its primary state and AGL as a "Terrain Offset" state. This allows it to fly a mission at a fixed MSL while simultaneously knowing its AGL for safety.

ARCHITECTURE (THE ENGINEER'S VIEW)

1. THE TERRAIN STATE (`TERRAINSTATE`)

The EKF tracks a single variable: **Terrain Vertical Position** (in NED frame).

- *Definition:* If the drone is at 10m Altitude (Baro) and the Rangefinder says "2m", the Terrain State is 8m.
- *Code Path:* `NavEKF3_core::EstimateTerrainOffset()`.

2. RANGEFINDER FUSION

When a Rangefinder is active:

1. **Prediction:** `Predicted_Range = Pos_Z - Terrain_State`.
2. **Measurement:** The Lidar reports a distance.
3. **Correction:** The difference (Innovation) is used to update the `terrainState`.
    - *Effect:* The EKF "moves the ground" to match the Lidar reading.

3. OPTICAL FLOW FUSION

Surprisingly, **Optical Flow** can also update the Terrain State.

- *Logic:* If the flow sensor sees the ground moving "too fast" for the current estimated height, it implies the ground is closer than thought.
- *Prerequisite:* Requires GPS velocity to be trusted.

HEIGHT SOURCE SWITCHING

The EKF must decide whether to trust the Barometer or the Rangefinder for its primary Z-axis control. This logic resides in `selectHeightForFusion()`.

- **Switching to Rangefinder:**
    - If `Range < RNG_USE_HGT` (percentage) AND speed is low (`RNG_USE_SPD`).
    - The EKF resets its Z-Axis origin to match the Rangefinder.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

- **Switching to Barometer:**
  - If the drone flies too high (out of range) or too fast.
  - It seamlessly blends back to Baro/GPS altitude.

KEY PARAMETERS

PARAMETER	DEFAULT	DESCRIPTION
`EK3_RNG_USE_HGT`	-1	(\%) Max rangefinder distance (as \% of max range) to use for primary altitude.
`EK3_RNG_USE_SPD`	2	(m/s) Max ground speed to use rangefinder. High speeds over rough terrain cause sensor lag issues.
`EK3_TERR_GRAD`	0.1	Expected terrain slope.

SOURCE CODE REFERENCE

- **Estimator Logic:** `AP_NavEKF3_OptFlowFusion.cpp`
- **Source Selection:** `AP_NavEKF3_PosVelFusion.cpp`

VIBRATION FAILSAFE & CLIPPING

EXECUTIVE SUMMARY

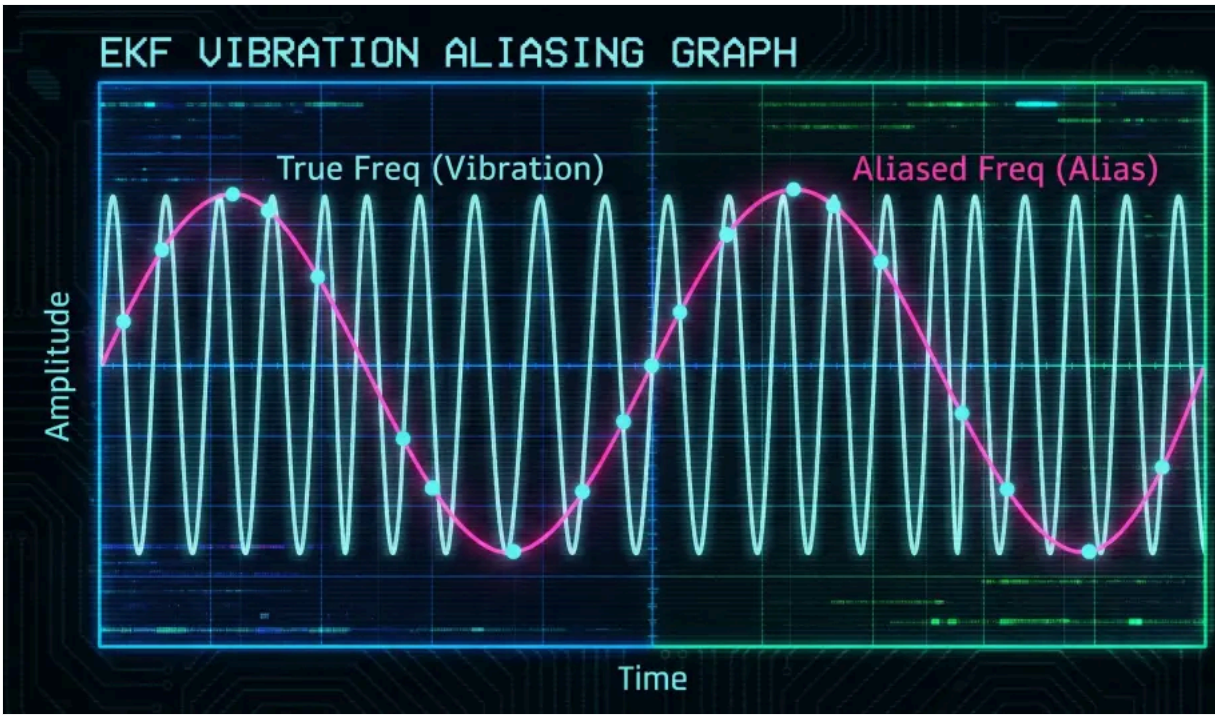
Vibration is the #1 enemy of the EKF. If the frame vibrates too much, the accelerometers produce "Aliased" data—high-frequency shaking that the EKF interprets as a constant acceleration. This causes the drone to "Rocket" into the sky or drift uncontrollably. The **Vibration Failsafe** monitors sensor health and triggers emergency measures (usually switching to `AltHold` or `Land`) if the vibration becomes dangerous.

THEORY & CONCEPTS

1. SIGNAL ALIASING & THE NYQUIST LIMIT

In digital signal processing, you must sample a signal at twice its frequency to "see" it.

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY



- **The Vibration:** A motor might vibrate at 200 Hz.
- **The Problem:** If the flight controller only samples the Accelerometer at 100 Hz, it cannot see the 200 Hz wave. Instead, that wave "aliases" into a lower frequency (like 5 Hz), making the drone think it is bobbing up and down.
- **The Solution:** ArduPilot samples at very high rates (up to 8 kHz) and uses **Hardware Low-Pass Filters** to kill the high-frequency vibration before it can alias into the EKF.

2. CLIPPING & NON-LINEAR ERRORS

Accelerometers are not infinite. They have a physical limit (clipping point).

- **The Error:** If a sensor clips at 16G, and a vibration spike hits 20G, the sensor only reports 16G.
- **The Math:** This "lost 4G" of data is a non-linear error. It can't be filtered out. The EKF sees a massive "Net Acceleration" in one direction that isn't real. This is why mechanical damping (foam/rubber) is superior to digital filtering for high-vibration frames.

ARCHITECTURE (THE ENGINEER'S VIEW)

1. CLIPPING (SENSOR SATURATION)

- **Mechanism:** Accelerometers have a physical range (e.g., +/- 16G). If a vibration spike exceeds this, the sensor reports "16G".
- **The Error:**
  - Real motion: +20G up, -20G down (Net: 0).
  - Measured: +16G up, -16G down (Net: 0).
  - *Result:* If the vibration is asymmetric, or if the clipping happens frequently, the integration (`Velocity = Sum(Accel * dt)`) accumulates a massive error.
- **Metric:** `[VIBE](/field-manual/log-analysis/vibration-analysis.html).Clip0/1/2` counts the number of times the sensor hit the limit. **Any** clipping is bad.

2. THE VIBRATION CHECK (`CHECK_VIBRATION`)

ArduCopter runs a dedicated check in `ekf_check.cpp`.

- **Trigger:** It monitors the EKF's **Vertical Velocity Innovation**.
- **Logic:**

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

    - If the EKF thinks we are climbing fast (`Innov > 0`) AND the Variance is high (`vel_variance > 1.0`), it assumes vibration aliasing.
    - *Why?* Because vibration usually aliases into the Z-axis (up/down) most severely.
- **Action:**
    - Sets `vibration_check.high_vibes = true` .
    - This forces the vehicle out of GPS modes (Loiter/Auto) and into **AltHold** (or Land).

WHY "ROCKETING"?

If vibration aliases as "Downward Acceleration" (the drone thinks it is falling), the Z-controller responds by applying Max Throttle to "stop the fall".

- **Result:** The drone shoots into the sky.
- **Fix:** Improving isolation dampers or balancing props is the *only* fix. PID tuning cannot fix mechanical vibration.

KEY PARAMETERS

PARAMETER	DEFAULT	DESCRIPTION
`INS_ACCEL_FILTER`	20	(Hz) Low-pass filter for accel. Lowering this can mask vibration but adds latency.
`EK3_HGT_I_GATE`	500	Height Innovation Gate. High vibration pushes the innovation past this gate, triggering the failsafe.

SOURCE CODE REFERENCE

- **Failsafe Logic:** `Copter::check_vibration()`
- **Clipping Count:** `AP_InertialSensor::get_accel_clip_count()`

PRACTICAL GUIDE: DIAGNOSING VIBRATION

If your drone switched to AltHold automatically or flew away, you likely have a vibe problem.

STEP 1: CHECK FOR CLIPPING

1. Open the log in Mission Planner.
2. Search for **VIBE** message.
3. Look at `Clip0` , `Clip1` , `Clip2` .
4. **The Rule:** These should be **Zero** during flight.
    - *Crash:* Clipping during a crash is normal.
    - *Flight:* If `Clip` increases while hovering, your autopilot is loose or your props are horribly unbalanced. **Do not fly.**

STEP 2: CHECK VIBE LEVELS

1. Plot `VIBE.VibeX` , `VibeY` , `VibeZ` .
2. **Safe Zone:** Below $15 m/s^2$.
3. **Danger Zone:** Above $30 m/s^2$.
4. **Critical:** Above $60 m/s^2$. The EKF will fail here.

THE HARDWARE FIX

- **Soft Mounting:** Put the flight controller on gel pads (Kyosho Zeal) or rubber bobbins.
- **Cable Discipline:** Ensure wires plugging into the FC are loose. A tight wire transmits frame vibration directly into the sensor, bypassing the soft mount.
- **Prop Balance:** Balance your props. Tape is cheaper than a new drone.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





THE CRASH CHECKER

EXECUTIVE SUMMARY

The **Crash Check** is a software safety mechanism designed to detect if the vehicle is out of control and impacting the ground. If a crash is confirmed, it **immediately disarms the motors** to prevent damage to the propellers, ESCs, and motors, and to reduce the risk of injury.

THEORY & CONCEPTS

1. DISARM SAFETY PROTOCOL

In the early days of drones, a crash would often lead to a "Fireball" or "Meltdown." This is because the flight controller, sensing it wasn't level, would spin the blocked motors to 100\% to try and fix the attitude.

- **The Problem:** High current into a stalled motor = Fire.
- **The Logic:** The Crash Checker acts as a **Supervisor**. It has the power to override the Attitude Controller and the Mixer. It is the "Kill Switch" of the operating system.

2. TIME-AVERAGED CONFIDENCE

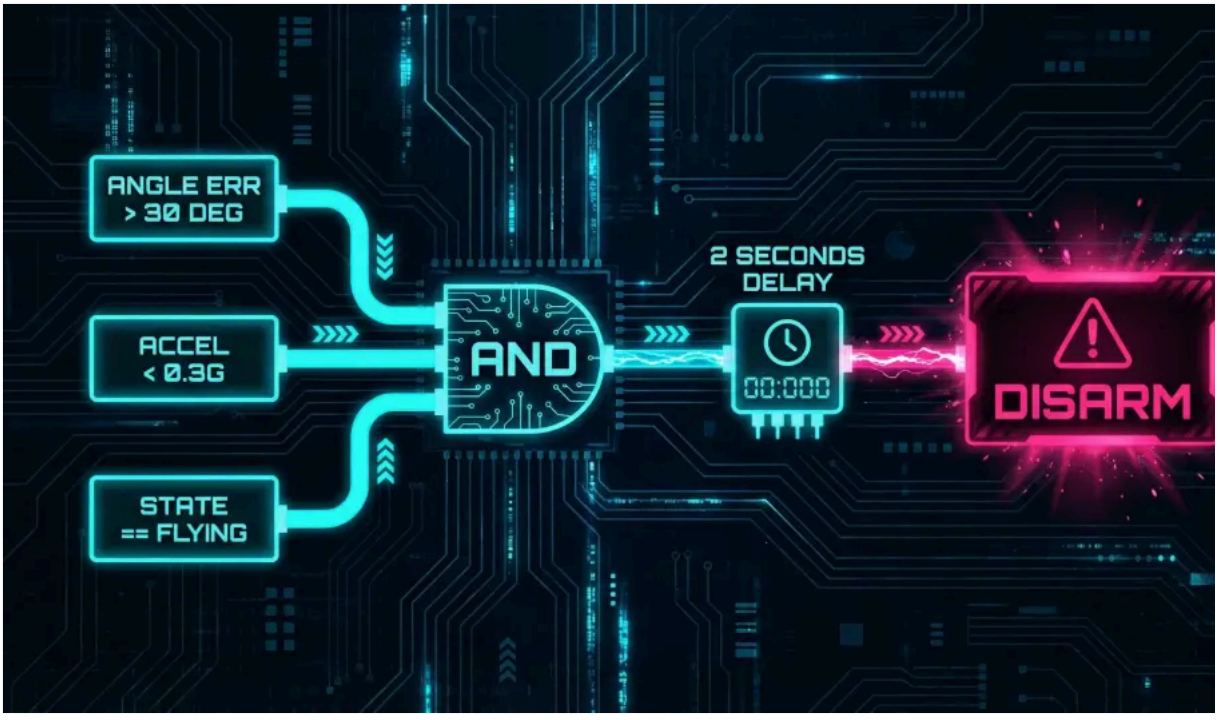
Why does ArduPilot wait for 2 seconds before disarming?

- **The Hazard:** Momentary impacts or "bumps" in the air shouldn't kill the power.
- **The Math:** It uses a **Confidence Window**. The crash criteria must be met consistently. This prevents "False Disarms" during aggressive flight or hard landings.

DETECTION ARCHITECTURE (THE ENGINEER'S VIEW)

The logic runs at 400Hz (or main loop rate) inside `Copter::crash_check()`.

It requires **ALL** of the following conditions to be true for **2 continuous seconds**:



1. **Massive Attitude Error:**
    - The difference between *Desired Angle* and *Actual Angle* exceeds 30 degrees (`CRASH_CHECK_ANGLE_DEVIATION_DEG`).

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

- *Meaning:* The autopilot is commanding "Level", but the drone is tilted > 30 degrees. It has lost control authority.

2. **IMU Confirmation:**
   - The vehicle is accelerating at < $3m/s^2$ (0.3G).
   - *Meaning:* It isn't pulling a high-G maneuver; it's likely tumbling or stationary on its side.
3. **Not Landed:**
   - The Land Detector says we are still "Flying".

WHY DISARM?

If the drone is upside down on the ground, the Attitude Controller will try to "flip it over" by applying Max Throttle to the low side.

- **Result:** The props are blocked by the ground. The motors stall. The ESCs overheat and catch fire.
- **Crash Check:** Recognizes this futile struggle and kills the power.

TROUBLESHOOTING FALSE POSITIVES

- **"Crash Disarm" on Landing:**
  - *Scenario:* You land hard and bounce. The drone tilts 40 degrees. You jam the throttle down.
  - *Problem:* The Land Detector takes ~1 second to decide "Landed". In that 1 second, the Crash Checker sees a large angle error and Disarms.
  - *Fix:* Improve landing gear or land smoother.
- **Acro Mode:** Crash check is often disabled or relaxed in Acro to allow for tumbling maneuvers.

KEY PARAMETERS

PARAMETER	DEFAULT	DESCRIPTION
`FS_CRASH_CHECK`	1	0=Disabled, 1=Disarm, 2=Disarm & Baro Check.

SOURCE CODE REFERENCE

- **Logic:** `Copter::crash_check()`

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





CHAPTER 6: CONTROL ARCHITECTURE

THE CONTROL STACK: FROM STICK TO PROP

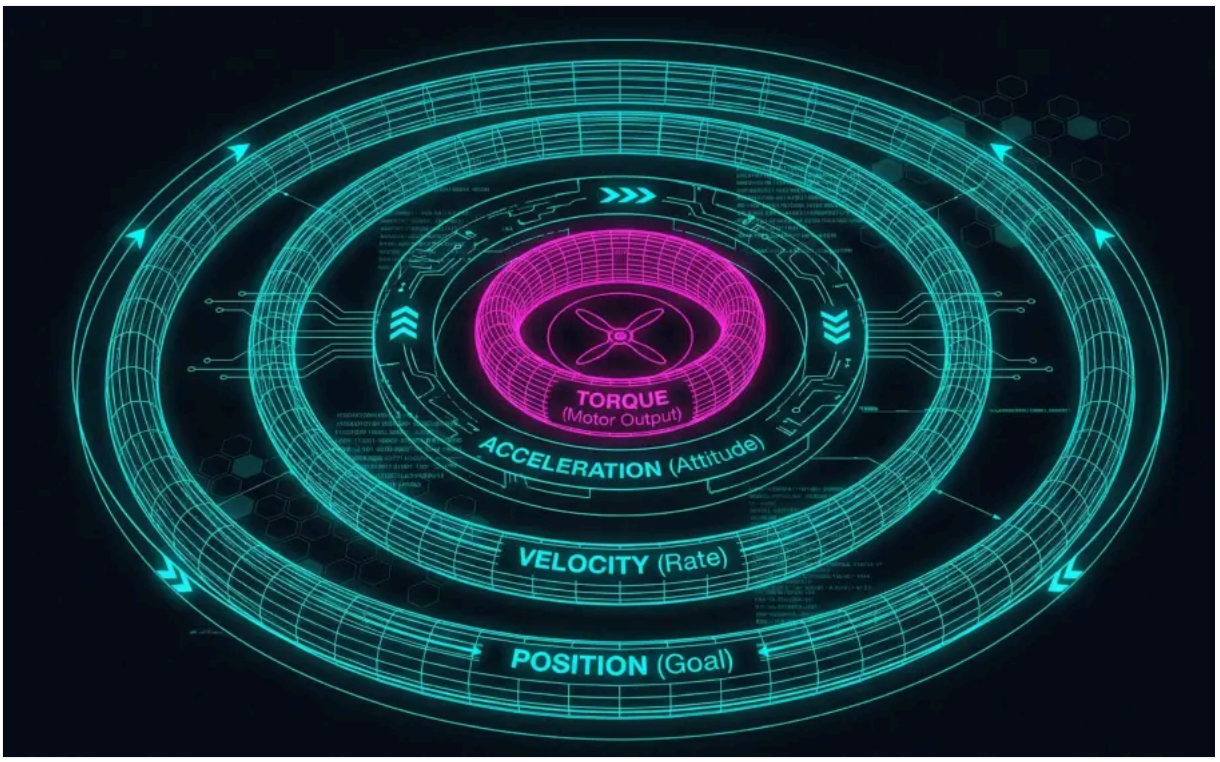
EXECUTIVE SUMMARY

When you move the sticks on your radio, you aren't moving the servos directly. You are feeding a request into the top of a complex "Cascade Controller." The pilot (or autopilot) requests a **Position** or **Angle**. The controller converts this into a **Rate**. The Rate controller converts this into **Motor Output**.

THEORY & CONCEPTS

1. THE HIERARCHY OF CONSTRAINTS

In robotics, you cannot change position without first changing velocity, and you cannot change velocity without first changing acceleration.



- **Mode Layer:** Defines the **Goal** (e.g. "Stay at 10m").
- **Position Layer:** Defines the **Velocity** needed to reach the goal.
- **Attitude Layer:** Defines the **Acceleration** (Lean Angle) needed to reach the velocity.
- **Rate Layer:** Defines the **Torque** needed to reach the angle.

2. TIME CONSTANTS & LOOP RATES

The stack is organized by frequency.

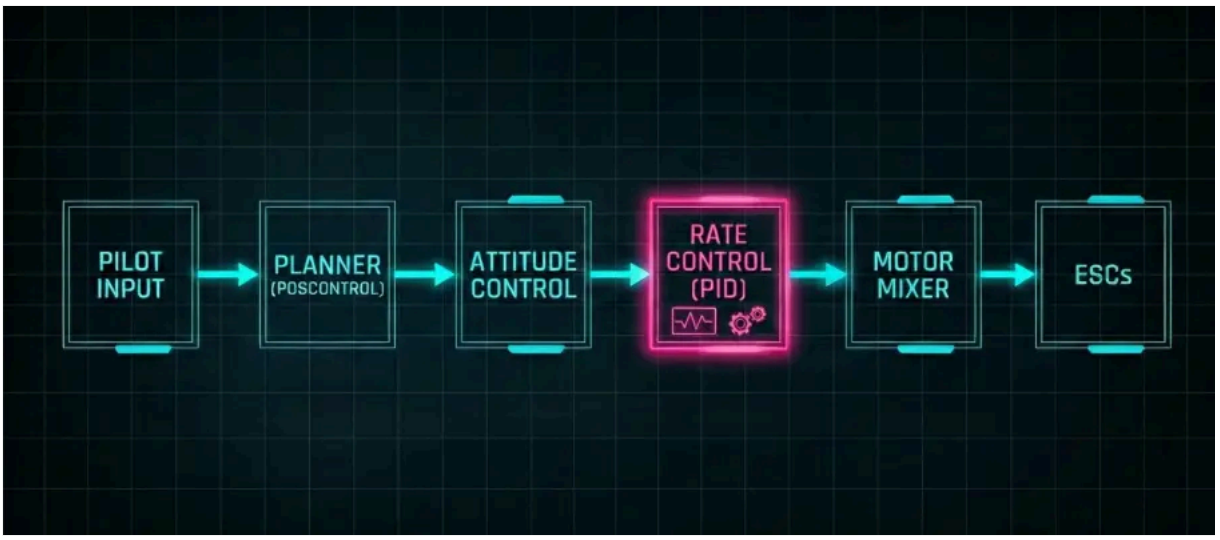
- **Outer Loops (Position):** Slow and stable. If they run slightly late, the drone drifts a few centimeters.
- **Inner Loops (Rate):** Fast and critical. If they run late, the drone oscillates or flips.
- *Why this matters:* This is why ArduPilot prioritizes the **Rate Controller** as a `FAST_TASK` in the scheduler.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

THE MAIN LOOP (400HZ)

The heartbeat of ArduCopter is the `fast_loop`.

- **Frequency:** Typically 400Hz (every 2.5 milliseconds).
- **Execution:** Every cycle, the scheduler runs the following critical tasks in order:



1. THE HIGH-LEVEL PLANNER (`UPDATE_FLIGHT_MODE`)

- **Input:** Pilot Sticks, Waypoints, Fence.
- **Logic:** "Where do I want to be?"
- **Output:** Target Lean Angles (Roll/Pitch) & Climb Rate.
- *Libraries:* `AC_WPNav`, `AC_PosControl`.

2. THE ATTITUDE CONTROLLER (`RUN_RATE_CONTROLLER`)

- **Input:** Target Lean Angles.
- **Logic:** "How fast do I need to rotate to get to that angle?"
- **Output:** Target Rotation Rates (Deg/s).
- *Libraries:* `AC_AttitudeControl`.

3. THE RATE CONTROLLER (INNER LOOP)

- **Input:** Target Rates vs Gyro Rates.
- **Logic:** "How much power do the motors need to achieve this rotation speed?"
- **Output:** Normalized Motor Throttle (0.0 to 1.0).
- *Technique:* PID + FeedForward.

4. THE MIXER (`MOTORS_OUTPUT`)

- **Input:** Roll, Pitch, Yaw, Throttle (0-1).
- **Logic:** "Which motors need to spin to create this torque?"
- **Output:** PWM / DShot pulses to ESCs.
- *Libraries:* `AP_Motors`.

DATA TYPES

- **Position/Velocity:** `Vector3f` (North, East, Down).
- **Attitude:** `Quaternion` (Rotation from Earth to Body frame).
- **Rates:** `Vector3f` (Radians/second).

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

SOURCE CODE REFERENCE

- **Scheduler Table:** `Copter::scheduler_tasks[]`
- **Loop Runner:** `AP_Vehicle::loop()`

AC_WPNAV - THE NAVIGATOR

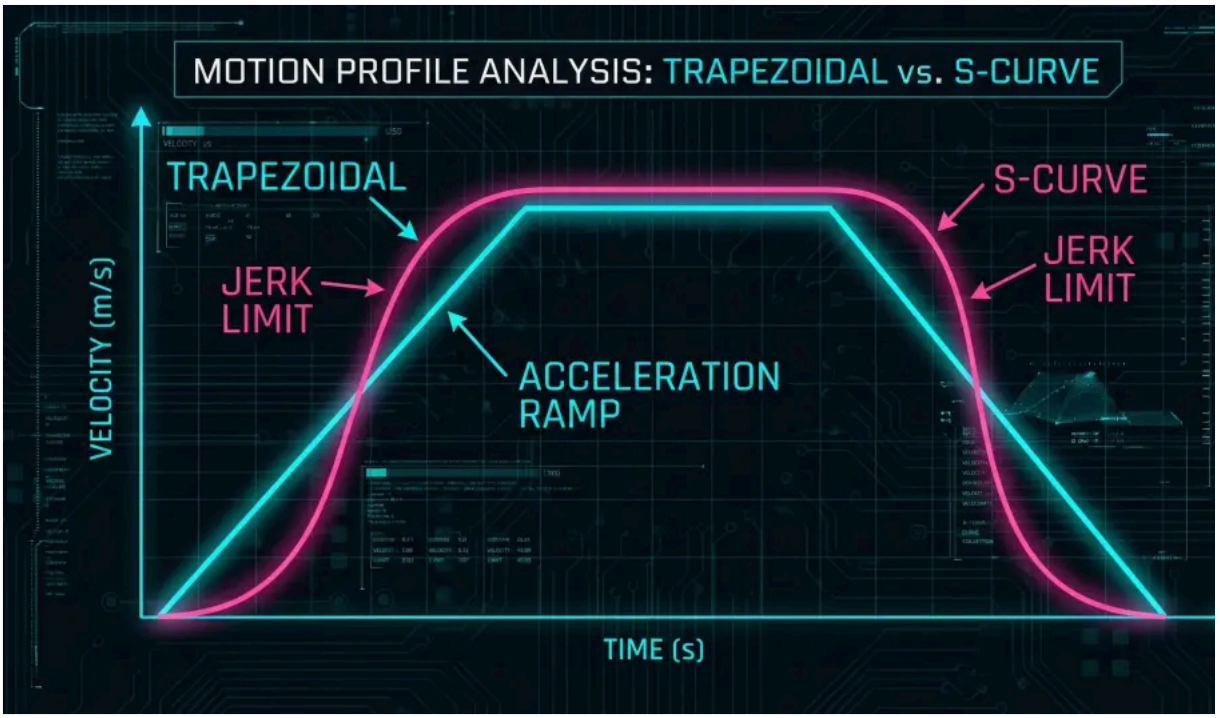
EXECUTIVE SUMMARY

The Waypoint Navigator (`AC_WPNav`) is responsible for generating the "Perfect Path" for the drone to fly. It does **not** fly the drone directly. Instead, it calculates a moving "Target Point" (Position, Velocity, Acceleration) that slides along the line between waypoints. The Position Controller then tries to chase this point.

THEORY & CONCEPTS

1. S-CURVE VS. LINEAR VELOCITY

Most hobbyist drones use a trapezoidal velocity profile: they accelerate at a constant rate, cruise, and stop.



- **The Problem:** The transition from "Stationary" to "Accelerating" is a sudden step. This causes "jerk" which makes the gimbal twitch and stresses the motors.
- **The S-Curve:** ArduPilot uses a 3rd-order S-Curve. It ramps the *acceleration* itself.
- **The Result:** The movement looks like it has "Natural Weight" (Physics-based) rather than robotic commands.

2. PATH BLENDING

When navigating a series of waypoints, the aircraft doesn't stop at each point.

- **Look-ahead:** The navigator analyzes the *next* waypoint while flying the current one.
- **Blending:** It calculates a circular arc that connects the two lines. The radius of this arc is limited by the `WPNAV_ACCEL` parameter.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

- *Why?* Maintaining momentum is more efficient and provides smoother footage than a series of stop-and-turn maneuvers.

ARCHITECTURE (THE ENGINEER'S VIEW)

1. S-CURVES (CINEMATIC SMOOTHNESS)

Early autopilots flew straight lines and stopped at every waypoint. ArduCopter uses **S-Curves** (Sigmoid velocity profiles).

- **Jerk Limited:** The path is calculated to limit "Jerk" (Change in Acceleration). This eliminates the robotic "twitching" seen in cheaper drones.
- **Mechanism:**
  - It calculates the distance to the next waypoint.
  - It generates a velocity profile that ramps up smoothly, cruises, and ramps down smoothly.
  - *Reference:* `SCurve` class.

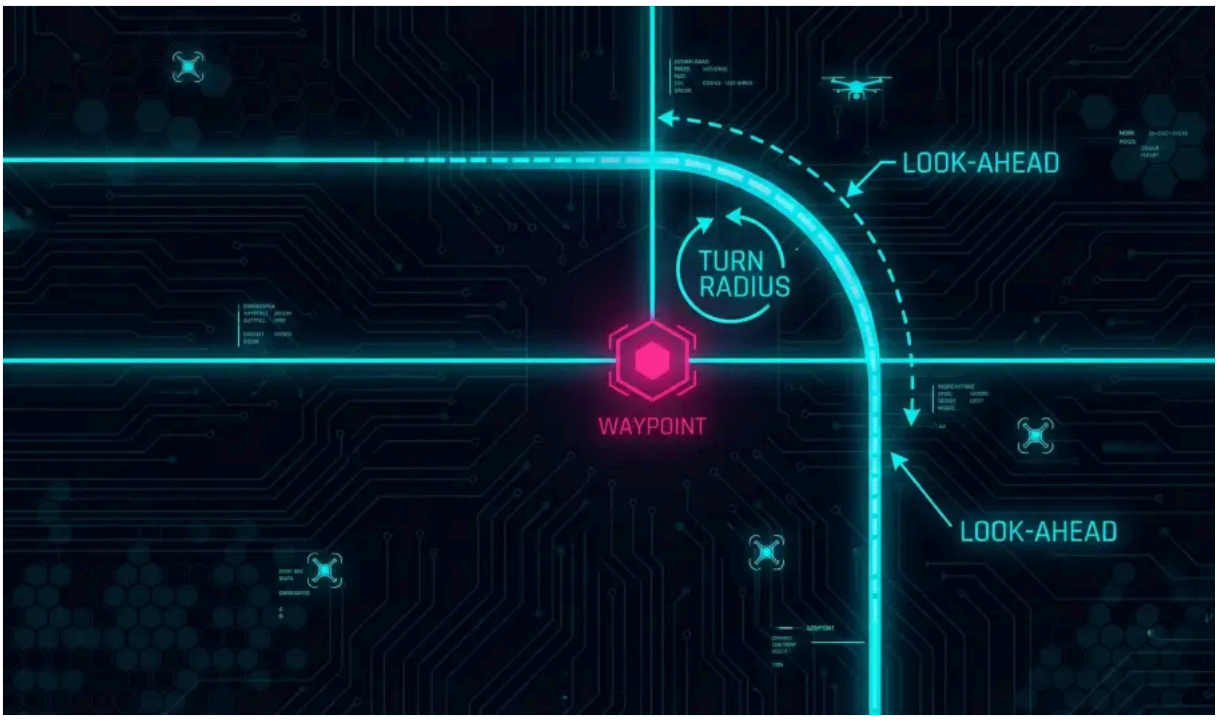
2. SPLINES (HERMITE CURVES)

If you use "Spline Waypoints" in your mission, `AC_WPNav` switches to a **Cubic Hermite Spline** solver.

- **Logic:** It defines a curved path that passes through the waypoint with a specific velocity vector (usually pointing to the *next* waypoint).
- **Result:** The drone flows through the point like a race car hitting an apex, rather than stopping and pivoting.

3. CORNERING (THE LEASH)

When flying straight lines (standard Auto), the drone doesn't stop at the corner.



- **Turn Radius:** It calculates a "Turn Radius" based on speed and `WPNAV_ACCEL`.
- **Cutting the Corner:** As it approaches the waypoint, `AC_WPNav` begins "blending" the current line into the next line, creating a rounded corner.
- **The "Target":** The computed target position moves along this rounded path.

KEY PARAMETERS

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

PARAMETER	DEFAULT	DESCRIPTION
`WPNAV_SPEED`	500	Max speed (cm/s).
`WPNAV_ACCEL`	100	Max acceleration. Lower = smoother starts/stops.
`WPNAV_RADIUS`	200	(cm) The radius at which the waypoint is considered "reached" and the turn begins.

SOURCE CODE REFERENCE

- **Update Loop:** `AC_WPNav::update_wpnav()`
- **Path Calculation:** `AC_WPNav::advance_wp_target_along_track()`

PRACTICAL GUIDE: TUNING THE NAVIGATOR

The "Stop and Go" behavior of ArduPilot Auto mode frustrates many new users. Here is how to make it flow.

THE PHYSICS OF THE TURN

ArduPilot will **NOT** violate your acceleration limit.

- **Scenario:** You are flying at 15 m/s (`WPNAV_SPEED = 1500`).
- **Constraint:** You limited acceleration to $1m/s^2$ (`WPNAV_ACCEL = 100`).
- **Physics:** To turn 90 degrees at 15 m/s requires huge centripetal acceleration ($a = v^2/r$).
- **The Result:** If `WPNAV_ACCEL` is low, the drone *must* calculate a massive turn radius (hundreds of meters) to stay within the limit. Since `WPNAV_RADIUS` (the trigger distance) is usually small (2m), the drone realizes "I can't make this turn without slowing down." So it brakes to a near-stop, turns, and accelerates again.

THE FIX

1. **Increase Acceleration:** Set `WPNAV_ACCEL = 300` ($3m/s^2$) or higher. This allows tighter high-speed turns.
2. **Increase Radius:** Set `WPNAV_RADIUS = 500` (5m). This gives the drone permission to start the turn earlier.

AC_POSCONTROL - THE XYZ CONTROLLER

EXECUTIVE SUMMARY

The Position Controller (`AC_PosControl`) is the "Middle Manager" of the flight stack. It takes orders from the Navigator ("Go to [10, 20, 5]") and converts them into orders for the Attitude Controller ("Lean Forward 5 degrees"). It handles the physics of moving a mass through space by cascading Position, Velocity, and Acceleration loops.

THEORY & CONCEPTS

1. THE INVERTED PENDULUM

A multicopter is essentially an **Inverted Pendulum**.

- **The Physics:** To move forward, you must first tilt the thrust vector.
- **The Lag:** There is a physical delay between "I want to move" and "The drone is tilted and producing horizontal force."
- **The Solution:** The Position Controller uses **FeedForward Acceleration**. It doesn't just wait for the drone to drift; it *commands* the tilt required to achieve the target velocity immediately.

2. P-PID CASCADE TUNING

The "Cascade" allows for granular control.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY



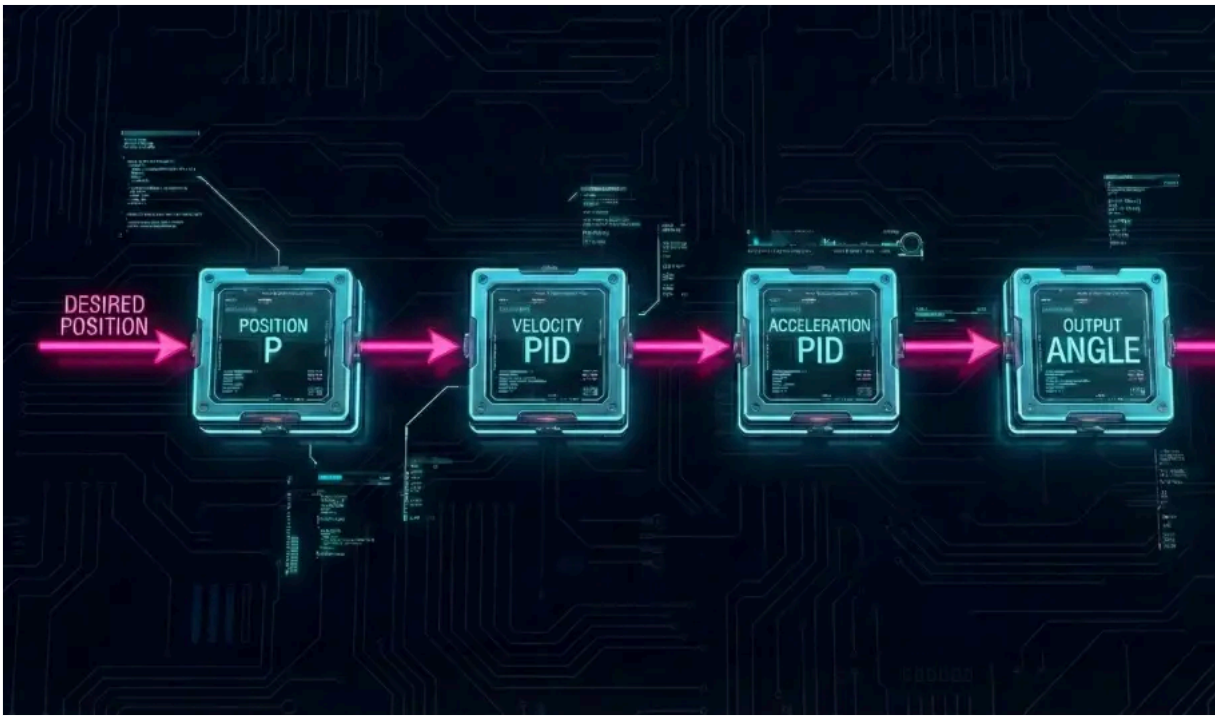


- **Outer P-Gain:** Handles the "Snap" to the position.
- **Inner PID-Gain:** Handles the "Damping" of the motion.
- *Key Logic:* If you have "Overshoot" (drone flies past the point and comes back), your **Velocity P** is too high relative to your **Position P**.

ARCHITECTURE (THE ENGINEER'S VIEW)

1. THE HORIZONTAL CASCADE (XY)

To move the drone, we must lean it.



1. **Position Loop:**
   - `Position_Error = Target_Pos - Current_Pos`
   - `Target_Velocity = Position_Error * P_Gain`
2. **Velocity Loop:**
   - `Velocity_Error = Target_Velocity - Current_Velocity`
   - `Target_Accel = PID(Velocity_Error)`
3. **Acceleration-to-Angle:**
   - To accelerate forward at `1g`, you must lean forward 45 degrees.
   - *Formula:* `Lean_Angle = atan(Target_Accel / Gravity)`
   - *Result:* Target Roll and Pitch angles are sent to `AC_AttitudeControl`.

2. THE VERTICAL CASCADE (Z)

To change altitude, we must change motor thrust.

1. **Position Loop:** `Altitude_Error` → `Target_Climb_Rate`.
2. **Velocity Loop:** `Climb_Rate_Error` → `Target_Z_Accel`.
3. **Acceleration Loop:** `Accel_Error` → `Throttle_Output (0-1)`.

WHY THIS MATTERS FOR TUNING

- **"Toilet Bowling":** If the XY Velocity PID is too aggressive (or compass is bad), the drone overshoots the target velocity, leans back too hard, stops, leans forward again, and starts circling the target.

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

- **"Yo-Yo" Altitude:** If the Z Velocity P-Gain is too high, the drone oscillates up and down because it reacts too violently to small climb rate errors.

KEY PARAMETERS

PARAMETER	DEFAULT	DESCRIPTION
`PSC_POSXY_P`	1.0	Position P-Gain. Converts distance error to speed.
`PSC_VELXY_P`	2.0	Velocity P-Gain. Converts speed error to acceleration.
`PSC_POSZ_P`	1.0	Altitude P-Gain.
`PSC_VELZ_P`	5.0	Climb Rate P-Gain.

SOURCE CODE REFERENCE

- **XY Controller:** `AC_PosControl::update_xy_controller()`
- **Z Controller:** `AC_PosControl::update_z_controller()`

PRACTICAL GUIDE: TUNING FOR WIND HOLD

If your drone gets pushed around by wind in Loiter mode, or oscillates (twitches) when hovering in wind, you need to tune the Velocity Loop.

1. THE TWITCH (HIGH P)

- **Symptom:** Drone hovers generally well but makes rapid, jerky twitching corrections in wind.
- **Diagnosis:** `PSC_VELXY_P` is too high. The controller is reacting to every tiny gust with a hard lean.
- **Fix:** Reduce `PSC_VELXY_P` from 2.0 to **1.5**.

2. THE DRIFT (LOW I)

- **Symptom:** Drone holds position in calm air, but in steady wind, it drifts 2-3 meters downwind and stays there.
- **Diagnosis:** Low `PSC_VELXY_I`. The P-term isn't strong enough to hold against constant wind pressure.
- **Fix:** Increase `PSC_VELXY_I`. The Integral term will accumulate the error and slowly lean the drone further into the wind until the drift stops.

3. THE OVERSHOOT (BALANCE)

- **Symptom:** You let go of the stick, and the drone flies *past* the stopping point, brakes hard, and flies back.
- **Fix:** Reduce `PSC_POSXY_P` (the "Approach Speed") or Increase `PSC_VELXY_D` (the "Braking Damping").

AC_ATTITUDECONTROL - THE ANGLE CONTROLLER

EXECUTIVE SUMMARY

The Attitude Controller is responsible for determining **how fast the vehicle should rotate** to achieve a desired **Orientation** (Roll, Pitch, Yaw). It sits between the Position Controller (which demands angles) and the Rate Controller (which demands motor torque). It uses sophisticated "Input Shaping" to ensure the vehicle rotates smoothly without overshooting or exceeding physical acceleration limits.

THEORY & CONCEPTS

1. ANGULAR ACCELERATION & CENTRIPETAL FORCE

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





When a drone rotates, it has to overcome the **Inertia** of the frame and the **Gyroscopic Effect** of the spinning propellers.

- **The Constraint:** Every vehicle has a physical limit on how fast it can rotate (`ATC_ACCEL_R_MAX`).
- **The Math:** If you command a 90 deg/s roll instantly, the Attitude Controller won't just ask for 90 deg/s. It will ramp the request from 0 to 90 using a trajectory that respects the frame's acceleration limit.

2. QUATERNION VS. EULER ANGLES

- **Euler Angles (Roll, Pitch, Yaw):** Easy for humans to understand, but they suffer from **Gimbal Lock** at 90 degrees.
- **Quaternions:** A 4-dimensional representation of rotation. They are mathematically superior, have no Gimbal Lock, and are used internally by ArduPilot for all calculations.
- *Conversion:* ArduPilot takes your Euler inputs (sticks) and converts them to a **Target Quaternion**. The error between the *Current Quaternion* and *Target Quaternion* is what drives the rotation rate.

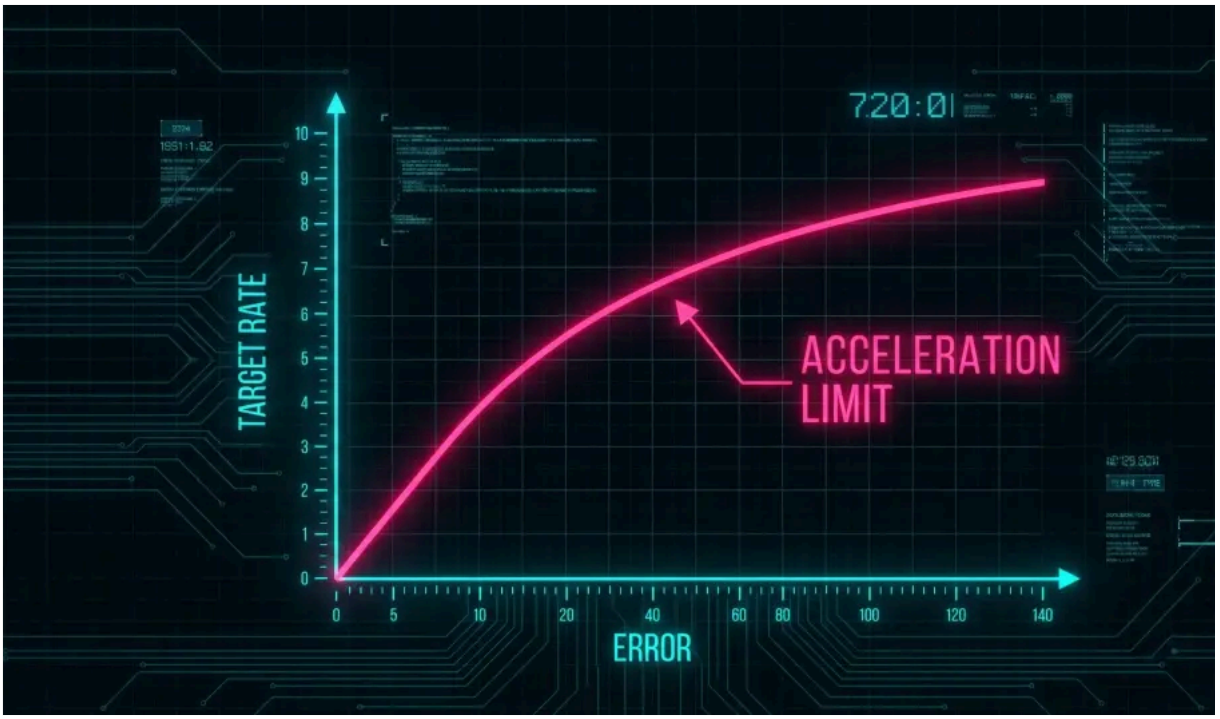
ARCHITECTURE (THE ENGINEER'S VIEW)

1. INPUT PROCESSING (`INPUT_EULER_ANGLE_...`)

- **Source:** Comes from Pilot Stick (Stabilize mode) or PosControl (Loiter/Auto mode).
- **Format:** Target Euler Angles (e.g., Roll 30 degrees, Pitch -10 degrees).
- **Yaw:** Usually passed as a Rate (deg/s), not an Angle, unless in a specific Heading Hold mode.

2. INPUT SHAPING (THE SQUARE ROOT CONTROLLER)

A standard PID controller is bad at handling large angle errors (step inputs). If you command a 45-degree roll instantly, a P-Controller would demand infinite rotation speed.
ArduPilot uses a **Square Root Controller** for the angle loop.



- **Logic:** It calculates the maximum velocity the vehicle can reach *and still stop in time* given its acceleration limit (`ATC_ACCEL_R_MAX`).
- **Result:**
  - Small Error: Behaves like a linear P-Controller.
  - Large Error: Behaves like a constant-acceleration trajectory generator.
- **Benefit:** Crisp response to small stick movements, but smooth, non-overshooting response to full-stick "bangs".

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

3. OUTPUT GENERATION

- **Target:** `_ang_vel_target` (Vector3f).
- **Type:** Body-Frame Angular Velocity (Radians/second).
- **Handoff:** These targets are passed down to the **Rate Controller** (`AC_AttitudeControl_Multi`).

KEY PARAMETERS

PARAMETER	DEFAULT	DESCRIPTION
`ATC_ANG_RLL_P`	4.5	Roll Angle P-Gain. Higher = Snappier response, Lower = Softer.
`ATC_ACCEL_R_MAX`	110000	(centi-deg/s/s) Max rotational acceleration.
`ATC_INPUT_TC`	0.15	Time Constant. Filters the pilot's stick input to make it feel "heavier" or "lighter".

SOURCE CODE REFERENCE

- **Input Function:** `AC_AttitudeControl::input_euler_angle_roll_pitch_euler_rate_yaw()`
- **Shaper Logic:** `AC_AttitudeControl::input_shaping_angle()`

PRACTICAL GUIDE: TUNING FOR SHARPNESS

If your drone feels "Laggy" in Stabilize mode (it follows your sticks but feels slow to start moving), you need to tune the Angle Controller.

STEP 1: THE SNAP (`ATC_ANG_RLL_P`)

This parameter controls how aggressively the drone tries to close the gap between your stick angle and the actual angle.

- **Default:** 4.5
- **Race Drones:** Increase to **6.0 - 8.0**. It will feel much more connected.
- **Cinematic:** Decrease to **3.5**. It will feel smoother.
- **Warning:** If you go too high (>10), the drone will oscillate (wobble) as it tries to level out.

STEP 2: THE ACCELERATION (`ATC_ACCEL_R_MAX`)

This limits how fast the rotation speed can change.

- **Default:** 110000 (1100 deg/s^2).
- **Optimization:** Set this to `0` (Disabled). This lets the drone accelerate as fast as the motors physically allow. This is standard for 5" freestyle quads.
- **Cinematography:** Keep it at default or lower (50000) to smooth out jerky stick inputs.

STEP 3: THE FEEL (`ATC_INPUT_TC`)

This is an input filter.

- **Default:** 0.15 (s).
- **Crisp:** Reduce to **0.10** or **0.08**. This removes the "Robot" feel and makes the drone track your stick instantly.

AC_ATTITUDECONTROL - THE RATE LOOPS

EXECUTIVE SUMMARY

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY



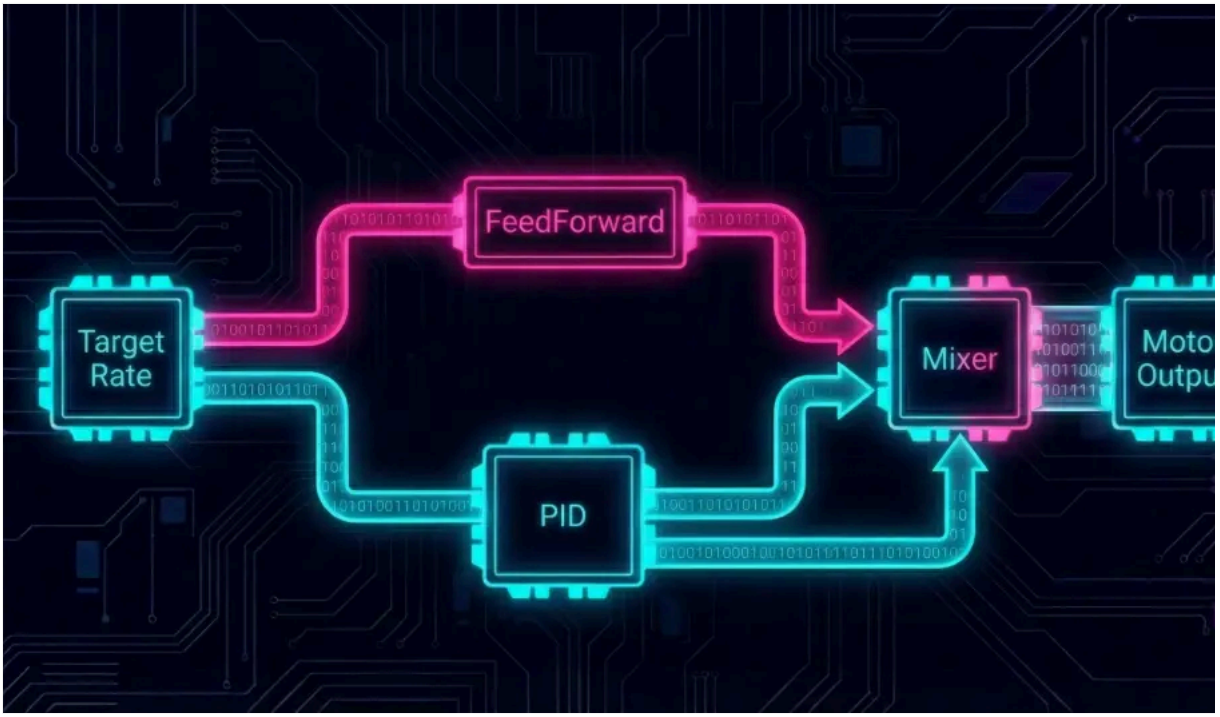


The Rate Controller is the "Inner Loop" of the flight stack. It runs at the full loop rate (400Hz+) and is responsible for stabilizing the vehicle against external disturbances (wind, turbulence) and executing the body-frame rotation commands requested by the Angle Controller. It converts **Target Rates** (deg/s) into **Motor Thrust** requests.

THEORY & CONCEPTS

1. FEEDFORWARD: ANTICIPATING PHYSICS

In a pure PID loop, the controller only reacts to an **Error**. (e.g. "I'm not at 100 deg/s, so I'll add power").



- **The Problem:** Waiting for an error causes **Latency**.
- **The Solution:** FeedForward (FF). The controller says: "I know the pilot wants 100 deg/s. Based on previous tuning, I know that needs exactly 40\% motor power. I'll add that 40\% *instantly*."
- *Result:* The drone feels "Snappy" and "Direct."

2. THE PID TUNING ORDER

The rate loop is the foundation. If it's not tuned correctly, nothing else works.

1. **FF First:** Adjust FF until the drone tracks the stick rates reasonably well.
2. **P Next:** Increase P until it responds crisply to gusts.
3. **D Last:** Increase D to dampen the "bounce" when you stop a roll.

ARCHITECTURE (THE ENGINEER'S VIEW)

1. THE PID LOOP

For each axis (Roll, Pitch, Yaw), the controller runs a separate PID loop.

- **Input:** `Error = Target_Rate - Gyro_Rate`.
- **P-Term:** Proportional response. High P = Responsive but oscillates.
- **D-Term:** Damping. Responds to the *change* in error. Stops the rotation from overshooting.
- **I-Term:** Integral. Corrects steady-state errors (e.g., a twisted frame or CG imbalance).
- **FeedForward (FF):** The "Secret Sauce".

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

- *Logic:* "I know I want to rotate at 100 deg/s. I know my frame needs 30\% power to do that. I'll just apply 30\% power *now* without waiting for an error to develop."
- *Result:* Zero-latency response.

2. FILTERING

Before the Gyro data hits the D-Term, it must be filtered to remove vibration noise.

- **Low Pass Filters:** `FLTD`, `FLTE`.
- **Notch Filters:** Targeted removal of propeller frequencies (Harmonic Notch).

3. OUTPUT HAND-OFF

The Rate Controller calculates a normalized value (-1.0 to +1.0) representing "Torque".

- It calls `motors.set_roll(output)`, `motors.set_pitch(output)`, etc.
- It does **not** decide which motors spin. That is the job of the **Mixer**.

KEY PARAMETERS

PARAMETER	DEFAULT	DESCRIPTION
`ATC_RAT_RLL_P`	...	Roll Rate P-Gain.
`ATC_RAT_RLL_D`	...	Roll Rate D-Gain (Damping).
`ATC_RAT_RLL_I`	...	Roll Rate I-Gain.
`ATC_RAT_RLL_FF`	...	Roll Rate FeedForward. Tuning this first is critical for crisp handling.
`ATC_RAT_RLL_FLTT`	20	Target filtering Hz.

SOURCE CODE REFERENCE

- **Controller Loop:** `AC_AttitudeControl_Multi::rate_controller_run()`

PRACTICAL GUIDE: TUNING THE D-TERM KICK

The D-Term (Derivative) is the "Shock Absorber" of your PID loop. Getting it right is the difference between a robot and a fluid flying machine.

THE SYMPTOM: "BOUNCE BACK"

- **Behavior:** You do a sharp roll and release the stick. The drone stops, but then "bounces" back slightly in the opposite direction.
- **Diagnosis: Low D-Term**. The P-term stopped the motion, but there was nothing to dampen the inertia, so it overshot and had to correct.
- **Fix:** Increase `ATC_RAT_RLL_D` and `ATC_RAT_PIT_D` in small increments (0.001 steps).

THE SYMPTOM: HOT MOTORS & SCREECHING

- **Behavior:** The drone flies fine, but the motors come down hot, or you hear a high-pitched "Trilling" sound during flight.
- **Diagnosis: High D-Term**. The D-term amplifies noise. If it is too high, it reacts to invisible micro-vibrations by frantically pulsing the motors.
- **Fix:** Decrease D-Term immediately. If D is low but motors are still hot, check your **Filters** (`INS_GYRO_FILTER` or Harmonic Notch).

THE PRO TIP: TUNE D *BEFORE* P

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





Modern theory (Betaflight/ArduPilot) suggests maximizing D first (until motors get warm) to provide maximum damping, then raising P to get sharpness. This provides the most "locked-in" feel.

AP_MOTORS - THE MATRIX & MIXER

EXECUTIVE SUMMARY

The Motor Mixer (`AP_MotorsMatrix`) is the bottom layer of the control stack. It takes the normalized Torque (Roll/Pitch/Yaw) and Thrust (Throttle) commands from the Rate Controller and translates them into physical PWM signals for each motor. It handles the geometry of the frame (Quad, Hex, Octo) and manages "Motor Saturation" to prioritize stability over altitude.

THEORY & CONCEPTS

1. TORQUE BALANCING

A quadcopter rotates by changing the torque balance of its motors.

- **Pitch/Roll:** Done by changing the relative **Thrust** (Vertical Force) between opposite motors.
- **Yaw:** Done by changing the relative **Torque** (Clockwise vs Counter-Clockwise rotation) of the motors.
- **The Matrix:** This is a mathematical map of which motor does what. If a frame is perfectly symmetrical, the factors are simple (0.5, 1.0). If the frame is asymmetric (like a Deadcat), the factors must be calculated to prevent "Coupling" (e.g., rolling causing a slight pitch).

2. MOTOR SATURATION & PRIORITY

What if you are at 100\% throttle and you want to roll?

- **Priority 1: Attitude.** The mixer will actually *reduce* the throttle of some motors below 100\% to create the torque needed for the roll.
- *Result:* The drone might lose altitude, but it will not crash. Stability is always prioritized over altitude.

ARCHITECTURE (THE ENGINEER'S VIEW)

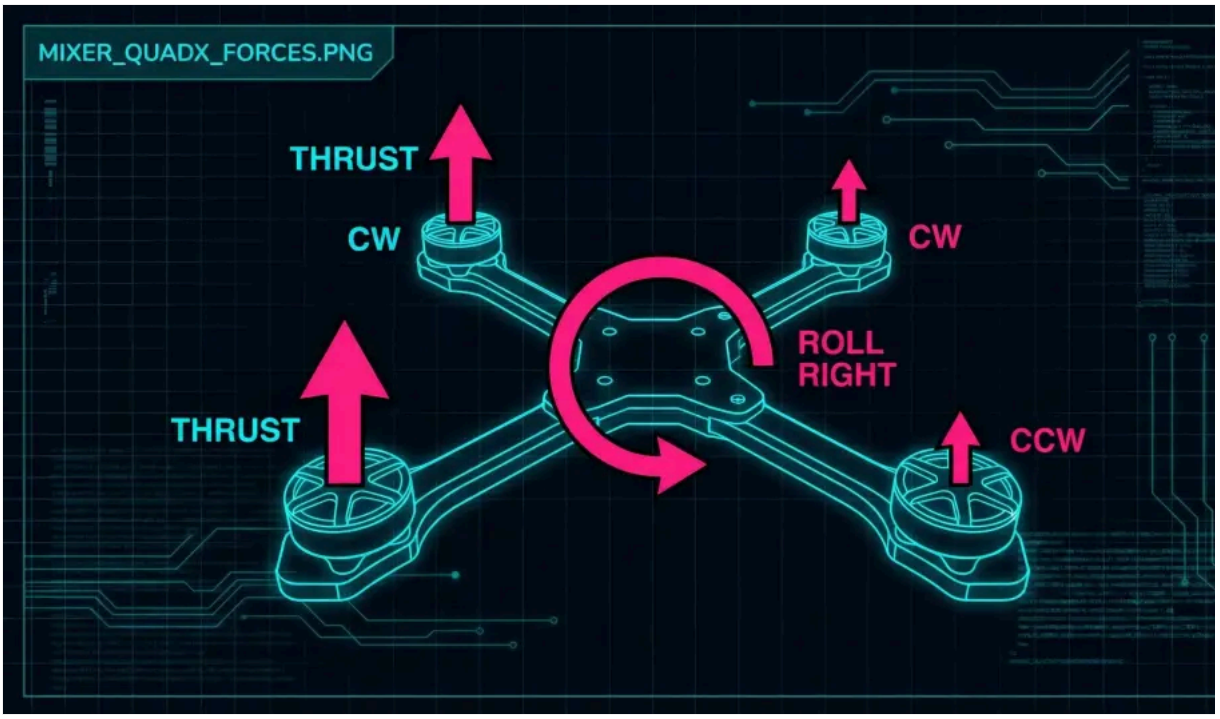
1. THE MIXING MATRIX

Each motor has a set of "Factors" based on its physical location.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY







- **Formula:**

```
Motor_Output = (Roll_In * Roll_Factor) +
               (Pitch_In * Pitch_Factor) +
               (Yaw_In * Yaw_Factor) +
               Throttle_In
```

- *Example (Quad X, Motor 1 - Front Right):*
  - Roll Factor: -0.5 (Spin down to roll right).
  - Pitch Factor: -0.5 (Spin down to pitch forward).
  - Yaw Factor: -1.0 (Spin down to yaw CW).

2. THE STABILITY PATCH (HANDLING SATURATION)

What happens if the drone is at 90\% Throttle, and the Stability Controller demands +30\% Roll?

- *Math:* 90\% + 30\% = 120\%.
- *Reality:* Motors max out at 100\%. If we just clip it, the drone won't roll enough, and it will crash.
- **The Solution:** The mixer automatically **lowers the average throttle** to make room for the roll command.
- *Result:* The drone climbs slower (or descends), but it *always* maintains the commanded attitude. **Attitude is king.**

3. THRUST LINEARIZATION

ESC/Motor combos are not linear. 50\% PWM does not equal 50\% Thrust (usually it's only ~25\% Thrust).

- **Problem:** If the autopilot assumes linear thrust, PIDs will be too weak at low throttle and too aggressive at high throttle (oscillations).
- **Correction:** `MOT_THST_EXPO` (default 0.65).
- **Logic:** The mixer applies an inverse quadratic curve to linearized the output *before* sending it to the ESC.

KEY PARAMETERS

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

PARAMETER	DEFAULT	DESCRIPTION
`MOT_SPIN_ARM`	0.10	(10\%) Motor speed when armed (Zero throttle). Keeps props spinning for air-mode authority.
`MOT_SPIN_MIN`	0.15	(15\%) Minimum speed during flight. Prevents motor desyncs.
`MOT_THST_EXPO`	0.65	Thrust curve linearization.
`MOT_PWM_TYPE`	0	0=Normal, 4=DShot150, 6=DShot600.

SOURCE CODE REFERENCE

- **Mixing Logic:** `AP_MotorsMatrix::output_armed_stabilizing()`

PRACTICAL GUIDE: VERIFYING MIXER ORDER & DIRECTION

Getting the motors plugged in wrong or spinning the wrong way is the #1 cause of "Instant Flip on Takeoff."

THE "MOTOR TEST" (SOFTWARE ORDER)

This verifies that "Motor 1" in code matches "Motor 1" on the frame.

1. **Remove Props.**
2. Connect Mission Planner → **Setup** → **Optional Hardware** → **Motor Test**.
3. Click **Test Motor A** (which is Motor 1).
   - **Expectation:** The **Front-Right** motor should spin.
   - *If Front-Left spins:* You have a wiring swap. Check your servo output map (`SERVO1_FUNCTION` , etc.).
4. Proceed through B, C, D in clockwise order (usually A=FR, B=BR, C=BL, D=FL for Quad X).

THE "MOTOSPIN" (HARDWARE DIRECTION)

This verifies the physical spin direction without needing a piece of paper.

1. **Props OFF.**
2. Enable DShot (`MOT_PWM_TYPE` = 4, 5, or 6).
3. Use the **"MotoSpin"** feature (if supported by your GCS/ESC Passthrough) OR simply hold a piece of paper against the motor bell during the "Motor Test."
   - *Paper pushes away:* Correct.
   - *Paper gets sucked in:* Wrong direction.
4. **Fixing Direction:**
   - **DShot:** Use **Bi-directional DShot Passthrough** to reverse the motor in the BLHeli/Bluejay settings. You do *not* need to swap wires.
   - **PWM:** Swap any two motor wires.

For full frame diagrams, see the ArduPilot Wiki: Connect ESCs and Motors.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





CHAPTER 7: CUSTOM AIRFRAMES

FRAME CLASS VS TYPE ARCHITECTURE

EXECUTIVE SUMMARY

When you configure a drone in Mission Planner, you select a **Frame Class** (e.g., Quad, Hexa) and a **Frame Type** (e.g., X, V, Plus). These are not just GUI labels; they map directly to C++ Enums that drive the initialization logic of the Motor Mixer. Understanding this mapping is the first step to creating a custom airframe.

THEORY & CONCEPTS

1. THE GEOMETRY OF CONTROL

A multicopter maneuvers by varying the thrust of its motors relative to its **Center of Gravity (CoG)**.

- **Frame Class:** Defines the "Complexity" of the mixer. A Quad has 4 motors to balance; an Octo has 8. More motors provide redundancy but require a more complex matrix.
- **Frame Type:** Defines the "Orthogonality." In a "Plus" frame, one motor handles Pitch exclusively. In an "X" frame, all four motors contribute to both Pitch and Roll.

2. SYMMETRY AND STABILITY

The EKF assumes the vehicle is physically symmetrical. If you mount a motor slightly further from the CoG than its partner, the drone will have a "biased" feel. ArduPilot compensates for this during initialization by normalizing the mixer factors, ensuring that a "10\% Roll" command results in the same angular acceleration whether you are rolling left or right.

ARCHITECTURE (THE ENGINEER'S VIEW)

The logic resides in `AP_MotorsMatrix::setup_motors()`.

1. FRAME CLASS (THE "HOW MANY")

Defined in `AP_Motors_Class.h`.

- **Purpose:** Determines the number of motors and the base structure.
- **Examples:** `MOTOR_FRAME_QUAD` (4), `MOTOR_FRAME_HEXA` (6), `MOTOR_FRAME_OCTA` (8).
- **Logic:** The code switches on `frame_class` to call the appropriate setup function (e.g., `setup_quad_matrix`).

2. FRAME TYPE (THE "WHERE")

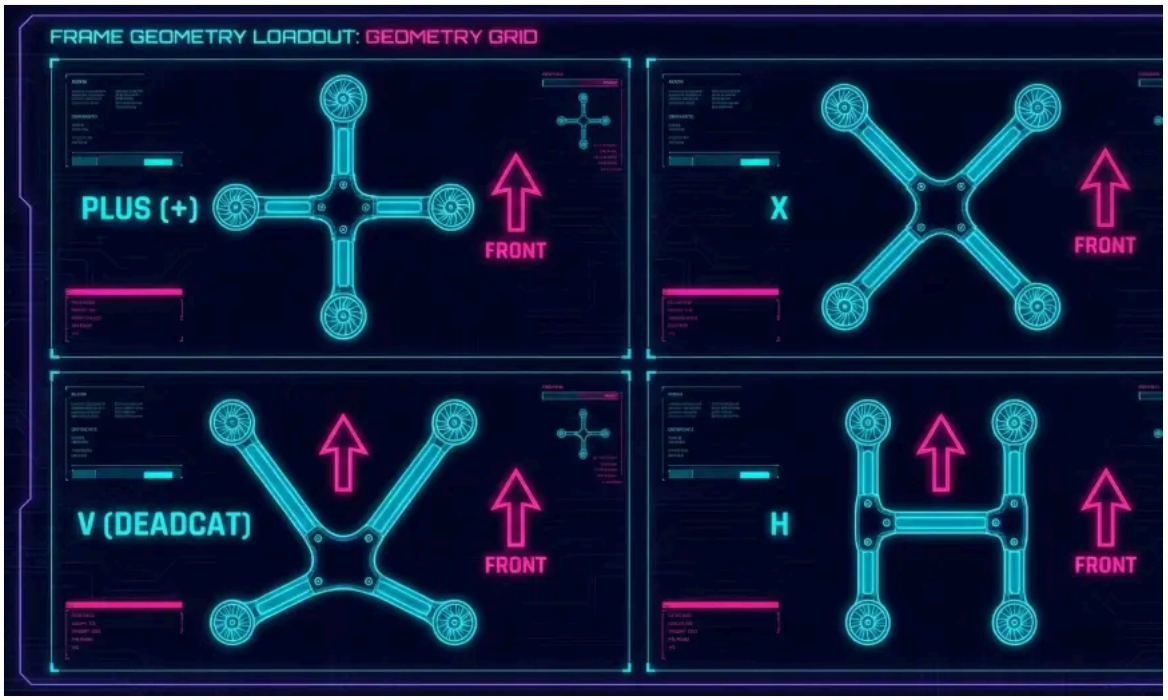
Defined in `AP_Motors_Class.h`.

- **Purpose:** Determines the geometry (angles) of the motors.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY







- **Examples:**
  - `MOTOR_FRAME_TYPE_X`: Standard X configuration.
  - `MOTOR_FRAME_TYPE_PLUS`: Motors at 0, 90, 180, 270 degrees.
  - `MOTOR_FRAME_TYPE_V`: Deadcat or V-tail geometry.
- **Logic:** Inside `setup_quad_matrix`, the code switches on `frame_type` to define the specific motor angles.

3. THE SETUP SEQUENCE

1. **Clear:** Existing motor definitions are wiped.
2. **Define:** The code iterates through the motors for the selected Class/Type.
3. **Add:** It calls `add_motor()` for each one, passing the physical angle (degrees) and Yaw Factor (CW/CCW).
4. **Normalize:** It calls `normalise_rpy_factors()` to ensure the total authority on each axis sums to reasonable values (preventing one axis from overpowering the others).

WHY IS THIS HARD-CODED?

You might wonder why ArduPilot doesn't just let you type in angles in the parameters.

- **Safety:** Hard-coding ensures that a parameter reset doesn't scramble your mixer and flip your drone.
- **Efficiency:** Pre-calculating the factors (Sin/Cos) saves CPU cycles during the fast loop.
- **Modern Solution:** For custom geometries without recompiling, ArduPilot now offers **Lua Scripting Mixers** (see Section 3).

KEY PARAMETERS

PARAMETER	DEFAULT	DESCRIPTION
`FRAME_CLASS`	1	1=Quad, 2=Hexa, 3=Octa, etc.
`FRAME_TYPE`	1	0=Plus, 1=X, 2=V, 3=H, etc.

SOURCE CODE REFERENCE

- **Setup Logic:** `AP_MotorsMatrix::setup_motors()`
- **Quad Logic:** `AP_MotorsMatrix::setup_quad_matrix()`

PRACTICAL GUIDE: SETTING UP A DEADCAT

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

A "Deadcat" frame has the front arms wider apart to clear the camera. If you configure it as a standard "X", the rear motors will overwork during pitch maneuvers.

THE SETUP

1. **Class:** `FRAME_CLASS = 1` (Quad).
2. **Type:** `FRAME_TYPE = 2` (V-Tail / Deadcat).
    - *Note:* In older versions, this was called "V". It applies to any frame where the front motors are wider than the rear motors.
3. **Reboot:** You **must** reboot the flight controller for mixer changes to take effect.

VALIDATION

1. **Remove Props.**
2. Arm the drone.
3. **Pitch Forward:** The rear motors should spin up.
4. **Pitch Back:** The front motors should spin up.
5. **Yaw Left:** The Front-Right and Rear-Left motors should spin up (CCW torque).
    - *If Yaw is wrong:* You likely have a motor spin direction issue, not a frame type issue.

DEFINING CUSTOM MIXERS IN C++

EXECUTIVE SUMMARY

For standard frames (X, H), ArduPilot automatically calculates the motor mixer based on the frame angle (e.g., 45 degrees). However, for asymmetric frames, V-Tails, or custom actuator layouts, you must define the mixer manually. This is done using the `add_motor` or `add_motor_raw` primitives in the C++ source code.

THEORY & CONCEPTS

1. TORQUE VS. THRUST

Every motor produces two forces:

1. **Thrust:** Pushes the drone UP (Vertical force).
2. **Torque:** Tries to rotate the drone body in the opposite direction of the propeller (Newton's 3rd Law).

- *Yaw Logic:* To Yaw Right, the drone slows down the CW-spinning motors and speeds up the CCW-spinning motors. The net thrust stays the same, but the net torque changes, rotating the drone.

2. COUPLING AND CROSS-TALK

A "Bad" custom mixer causes **Coupling**.

- *Scenario:* You command a Roll. Because your motor factors are slightly wrong, the drone rolls *and* pitches forward.
- *Fix:* High-fidelity custom mixers (like the Deadcat) must calculate factors based on the exact X/Y distance from the CoG, ensuring the Pitch and Roll vectors are mathematically orthogonal.

THE PRIMITIVE: `ADD_MOTOR_RAW`

This is the lowest-level way to define a motor. You explicitly tell the mixer how much each motor contributes to Roll, Pitch, and Yaw.

FUNCTION SIGNATURE

```
void add_motor_raw(int8_t motor_num, float roll_fac, float pitch_fac, float yaw_fac, uint8_t testing_order);
```


Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





UNDERSTANDING THE FACTORS (-1.0 TO 1.0)

- **Roll Factor:**
  - `-1.0`: Motor is on the Right. Spinning it up rolls the vehicle Left.
  - `+1.0`: Motor is on the Left. Spinning it up rolls the vehicle Right.
  - *Note:* Yes, it seems inverted. A positive roll command (Right) requires the Left motors to spin up.
- **Pitch Factor:**
  - `-1.0`: Motor is in Front. Spinning it up pitches the nose Up.
  - `+1.0`: Motor is in Back. Spinning it up pitches the nose Down.
- **Yaw Factor:**
  - `-1.0`: Motor torque spins the vehicle CW. To Yaw Right (CW), we spin this motor.
  - `+1.0`: Motor torque spins the vehicle CCW.

THE HELPER: `ADD_MOTOR` (ANGLE-BASED)

If your frame is symmetrical but just has weird angles (e.g., a Deadcat), you can let ArduPilot calculate the Sin/Cos for you.

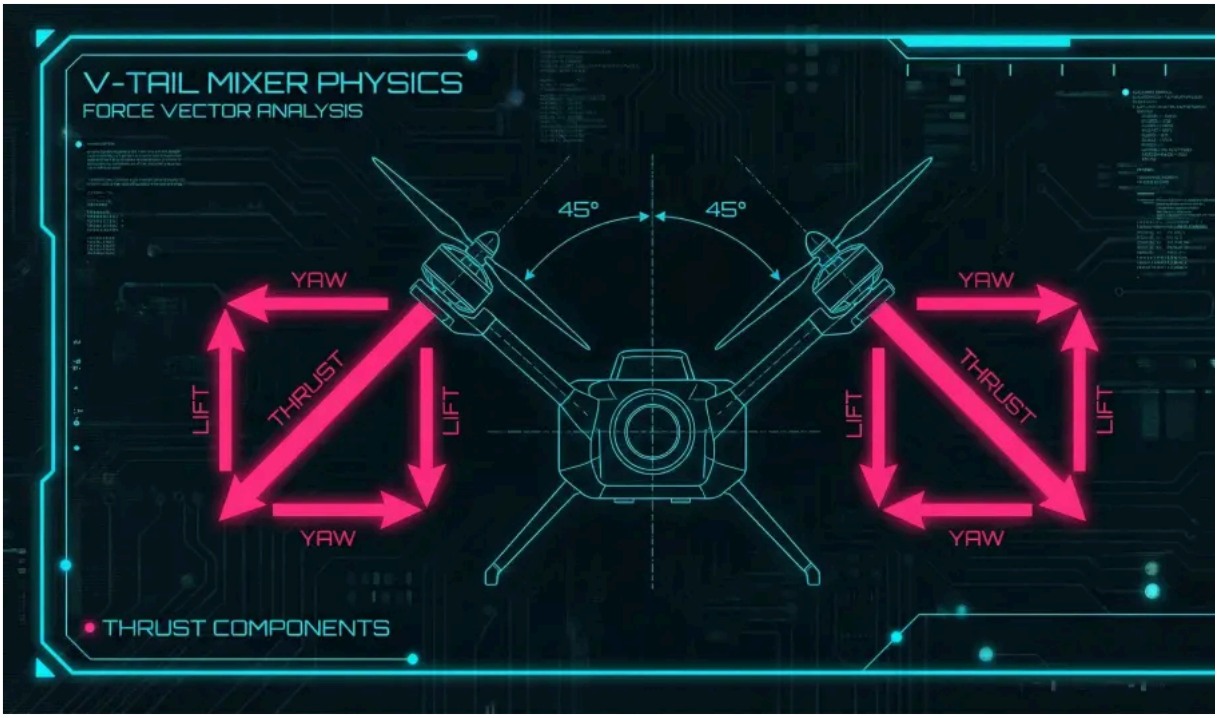
FUNCTION SIGNATURE

```
void add_motor(int8_t motor_num, float angle_degrees, float yaw_factor, uint8_t testing_order);
```

- **Angle:** 0 = Front. 90 = Right. 180 = Back. -90 = Left.

CASE STUDY: THE V-TAIL

A V-Tail quadcopter is complex because the rear motors contribute to both Pitch and Yaw, but not much Roll.



- **Code Path:** See `MOTOR_FRAME_TYPE_VTAIL`.
- **Logic:**
  - Front Motors: High Roll Factor, Zero Yaw Factor.
  - Rear Motors: Low Roll Factor, High Pitch Factor, High Yaw Factor.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

SOURCE CODE REFERENCE

- **Function Definitions:** `AP_MotorsMatrix.cpp`

LUA SCRIPTING MIXERS

EXECUTIVE SUMMARY

For years, creating a custom airframe meant editing C++ code, recompiling, and flashing custom firmware. Now, with ArduPilot's **Lua Scripting Engine**, you can define completely custom motor mixers—including 6-DOF Omnicopters—using a text file on the SD card.

THEORY & CONCEPTS

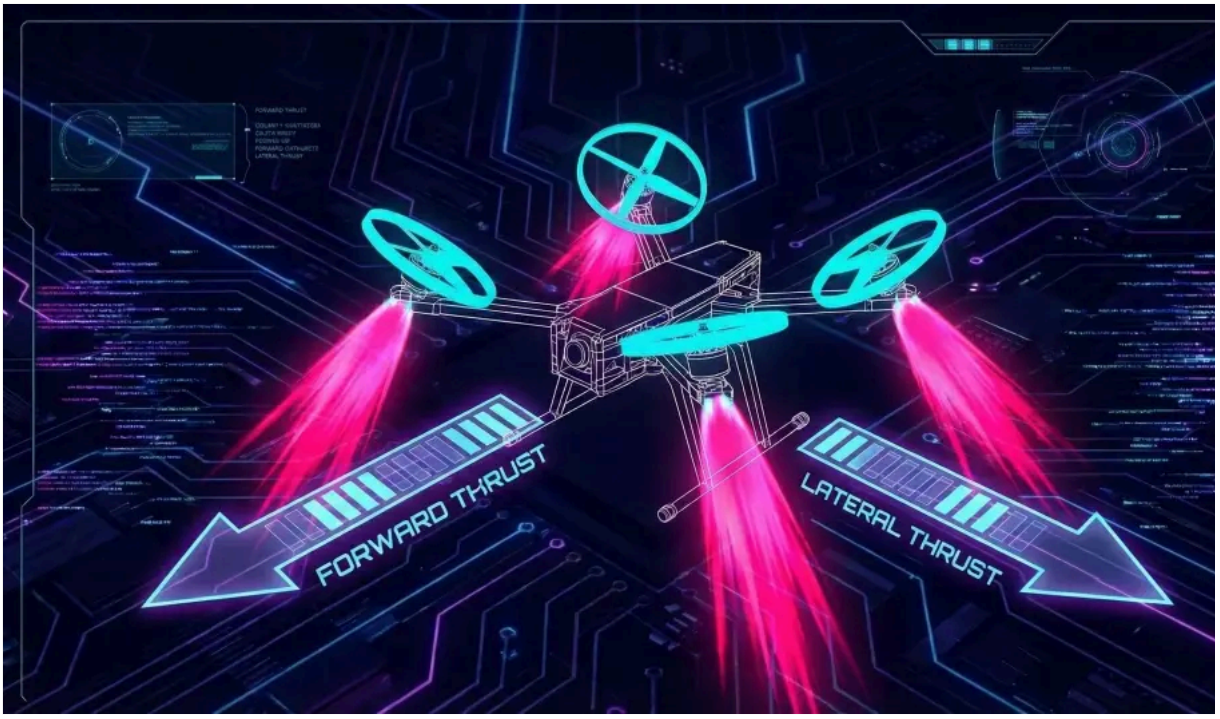
1. THE VIRTUAL SANDBOX

Lua scripting in ArduPilot runs in a restricted, protected memory area (the sandbox).

- **The Advantage:** You can experiment with complex 6-DOF mixers (like Tilt-Rotors or OMNI-frames) without the risk of a C++ segmentation fault crashing the entire flight controller.
- **The Power:** ArduPilot exposes the **Motor Matrix API** directly to Lua. This allows you to write scripts that change how the drone flies based on external factors (like a servo position or a distance sensor).

2. 6-DOF MECHANICS (OMNI-DIRECTIONAL)

Standard drones are "Under-Actuated" (they must tilt to move). A 6-DOF frame is "Fully-Actuated."



- **The Physics:** By mounting motors at specific angles, the drone can produce horizontal thrust *without* tilting.
- **The Controller:** Using the `Motors_6DoF` class in Lua, you gain access to the `Forward` and `Lateral` control channels, allowing the drone to "Stafe" and "Slide" while keeping the camera perfectly level.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

ARCHITECTURE (THE ENGINEER'S VIEW)

The system uses the `AP_Scripting` library to expose C++ Motor classes to the Lua environment.

1. THE SCRIPTING BACKENDS

There are two primary C++ classes available to scripts:

1. `Motors_dynamic` (`AP_MotorsMatrix_Scripting_Dynamic`):
   - Allows defining standard (3/4-DOF) multicopters.
   - Supports changing factors at runtime (e.g., for a variable-pitch prop or tilting arm).
2. `Motors_6DoF` (`AP_MotorsMatrix_6DoF_Scripting`):
   - Allows defining **6-Degree-of-Freedom** vehicles.
   - Inputs: Roll, Pitch, Yaw, Throttle, **Forward**, **Lateral**.
   - *Use Case:* Omnicopters, Coaxial-Tilt-Rotors.

2. HOW TO WRITE A MIXER SCRIPT

A Lua mixer script runs once at startup to configure the frame.

```
-- Select the Scripting Frame Class (16)
param:set('FRAME_CLASS', 16)

-- Define Motor 1 (Front Right)
-- motor_num, roll, pitch, yaw, throttle, forward, right, reversible, testing_order
motors:add_motor(0, -0.5, -0.5, -1.0, 1.0, 0, 0, false, 1)
```

- **No Recompile:** Just save this as `mixer.lua` in the `APM/scripts` folder on the SD card.
- **Initialization:** When the board boots, `FRAME_CLASS=16` tells the C++ code to look for definitions from the Lua engine.

ADVANCED CAPABILITIES

- **Reversible Motors:** The `reversible` flag tells the DShot backend that this motor can spin backward (3D flight).
- **Lateral Thrust:** You can map a motor to push the drone sideways (Lateral) without rolling. This enables "Strafing" in Loiter mode.

KEY PARAMETERS

PARAMETER	DEFAULT	DESCRIPTION
`FRAME_CLASS`	1	Set to **16** (Scripting Matrix) or **17** (Scripting 6DOF) to enable Lua mixers.
`SCR_ENABLE`	0	Set to 1 to enable the Lua engine (requires 2MB flash board).

SOURCE CODE REFERENCE

- **Dynamic Loader:** `AP_MotorsMatrix_Scripting_Dynamic::add_motor()`

PRACTICAL GUIDE: YOUR FIRST CUSTOM MIXER (QUAD X)

This example shows how to replicate the standard "Quad X" frame using Lua. Use this as a template for your weird custom builds.

STEP 1: THE SCRIPT (`QUAD_X.LUA`)

Create a file named `quad_x.lua` in your `APM/scripts/` directory with this content:

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





```
-- Define a standard Quad X layout
-- The signs determine the direction of torque/thrust
-- Roll/Pitch factors are usually 0.5 or 0.707 depending on normalization

local M1 = 0 -- Front Right (CCW)
local M2 = 1 -- Rear Left (CCW)
local M3 = 2 -- Front Left (CW)
local M4 = 3 -- Rear Right (CW)

-- motors:add_motor(motor_num, roll, pitch, yaw, throttle, forward, right, reversible, testing_order)

-- Motor 1: Front Right (Roll Right -, Pitch Forward -)
motors:add_motor(M1, -0.5, -0.5, -1.0, 1.0, 0, 0, false, 1)

-- Motor 2: Rear Left (Roll Left +, Pitch Back +)
motors:add_motor(M2,  0.5,  0.5, -1.0, 1.0, 0, 0, false, 2)

-- Motor 3: Front Left (Roll Left +, Pitch Forward -)
motors:add_motor(M3,  0.5, -0.5,  1.0, 1.0, 0, 0, false, 3)

-- Motor 4: Rear Right (Roll Right -, Pitch Back +)
motors:add_motor(M4, -0.5,  0.5,  1.0, 1.0, 0, 0, false, 4)

-- Log that we are done
gcs:send_text(6, "Lua: Custom Quad X Mixer Loaded")
```

STEP 2: CONFIGURATION

1. Set `SCR_ENABLE = 1` and Reboot.
2. Set `FRAME_CLASS = 16` (Scripting Matrix).
3. Reboot.
4. Check the "Messages" tab. You should see "Lua: Custom Quad X Mixer Loaded".
5. **Critical:** Perform a "Motor Test" (Propellers OFF!) to verify your mapping is correct.

THE OUTPUT MAP: SERVON_FUNCTION

EXECUTIVE SUMMARY

In ArduPilot, "Motor 1" is not physically tied to "Pin 1". Instead, there is a layer of abstraction called **SRV_Channels**. The Motor Mixer outputs to a logical "Function" (`k_motor1`), and you map that Function to any physical pin (`SERVO5_FUNCTION`) using parameters.

THEORY & CONCEPTS

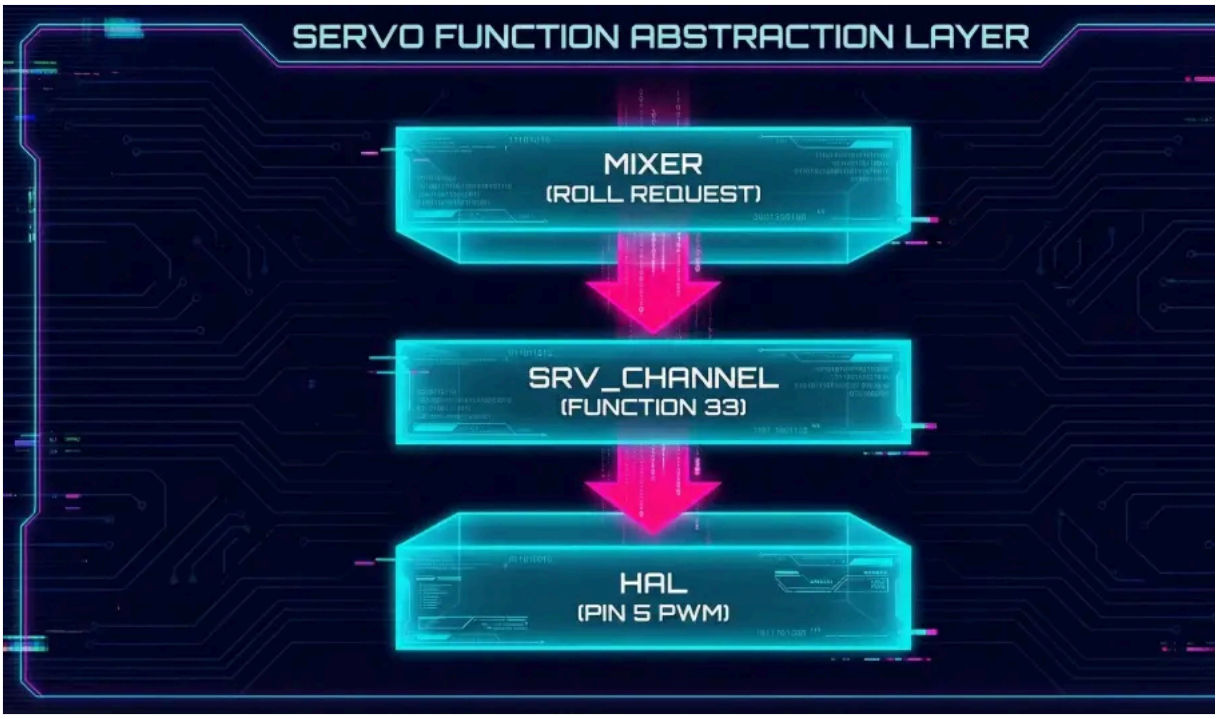
1. HARDWARE ABSTRACTION LAYERS (HAL)

A Hardware Abstraction Layer is a software bridge between the generic "Autopilot Brain" and the specific "Physical Pins" of a chip.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY







- **The Logic:** ArduCopter doesn't know what an STM32 or an H7 chip is. It just knows "Motor 1".
- **The Bridge:** `SRV_Channels` acts as the dispatcher. This allows the same ArduCopter firmware to run on hundreds of different boards with different pinouts.

2. FUNCTIONAL MAPPING VS. PIN MAPPING

- **Legacy Systems:** You had to plug Motor 1 into Pin 1. If Pin 1 broke, the board was useless.
- **ArduPilot:** Every pin is universal. If Pin 1 is dead, you can move the wire to Pin 8 and set `SERVO8_FUNCTION = 33`. The "Brain" never needs to know the hardware changed.

ARCHITECTURE (THE ENGINEER'S VIEW)

1. THE FUNCTION ENUM

ArduPilot defines a list of ~150 possible output functions in `Aux_servo_function_t`.

- **33:** Motor 1
- **34:** Motor 2
- **35:** Motor 3
- **36:** Motor 4
- ...
- **1:** Manual (Passthrough)

2. THE CONVERSION PIPELINE (`CALC_PWM`)

When the Mixer says "Motor 1 at 50\%", here is what happens:

1. **Normalization:** The mixer outputs a float `0.5`.
2. **Scaling:** `SRV_Channel::calc_pwm()` takes this float.
3. **Endpoint Lookup:** It looks up the `SERVOx_MIN` and `SERVOx_MAX` parameters for the assigned pin.
4. **Math:**

```
PWM = MIN + (Scaled_Value * (MAX - MIN))
```


Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

- *Example:* `1000 + (0.5 * (2000 - 1000)) = 1500us`.

5. **Trim:** If the function uses trim (like a servo surface), it centers around `SERVOx_TRIM`.

3. SAFETY INTERLOCKS

The `SRV_Channels` library also handles the **Safety Switch**.

- If the Safety Switch is active, `SRV_Channels` forces the output to `0` (or `disarmed_pwm`), regardless of what the Mixer requests.

DEBUGGING TIPS

- **"My Motor 1 is on Pin 5":** This is valid. Set `SERVO5_FUNCTION = 33` (Motor 1).
- **"My Servo moves backwards":** Set `SERVOx_REVERSED = 1`. This flips the math logic (1.0 input becomes MIN pwm).

SOURCE CODE REFERENCE

- **Conversion Logic:** `SRV_Channel::calc_pwm()`
- **Function List:** `SRV_Channel.h`

PRACTICAL GUIDE: CONFIGURING A GRIPPER (SERVO)

Let's say you have a 4-motor quadcopter (Channels 1-4) and you plug a Servo Gripper into Pin 5. You want to control it with Channel 9 on your transmitter.

STEP 1: PHYSICAL CONNECTION

- Plug the Servo Signal wire into Pin 5 (MAIN OUT 5).
- **Warning:** Most Autopilots do **NOT** provide 5V power on the servo rail. You must use a separate BEC to power the servo's Red/Black wires.

STEP 2: FUNCTION MAPPING

Tell ArduPilot what Pin 5 *is*.

- **Parameter:** `SERVO5_FUNCTION`
- **Value: 59** (`RCIN9`).
- **Meaning:** "Whatever PWM value comes in from Radio Channel 9, send it directly to Output Pin 5."

STEP 3: ENDPOINTS

Servos burn out if you drive them too far.

- **Open the Gripper:** Toggle your switch. If it buzzes, reduce `SERVO5_MAX` (e.g., from 1900 to 1800).
- **Close the Gripper:** Toggle switch back. If it buzzes, increase `SERVO5_MIN` (e.g., from 1100 to 1200).

STEP 4: REVERSING

If the switch is backwards (Up = Closed, Down = Open):

- **Parameter:** `SERVO5_REVERSED`
- **Value: 1**.

DEBUGGING MIXERS: MOTOR TEST & LOGGING

EXECUTIVE SUMMARY

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





After defining a custom frame, the most terrifying moment is the first arming. Will it flip? To verify your mixer *without* flying, you use the **Motor Test** tool in Mission Planner (or QGC). This tool sends a command to spin specific motors in a specific order, allowing you to verify that "Motor A" in the software corresponds to "Front Right" on your frame.

THEORY & CONCEPTS

1. PRE-FLIGHT CONFIDENCE & SAFETY

In aerospace, every system must be verified before "First Motion." The Motor Test is a **System Identification** step.

- **Verification:** You are proving that the Software Mapping, the ESC Protocol (DShot), and the Physical Wiring are all in alignment.
- **Torque Check:** By spinning just one motor, you verify its rotation direction. If Motor 1 is CCW but spins CW, the drone will flip instantly on arming. Motor Test is the only safe way to catch this before takeoff.

2. THE COMMAND/CONTROL LOOP BYPASS

The Motor Test is an **Open-Loop** command.

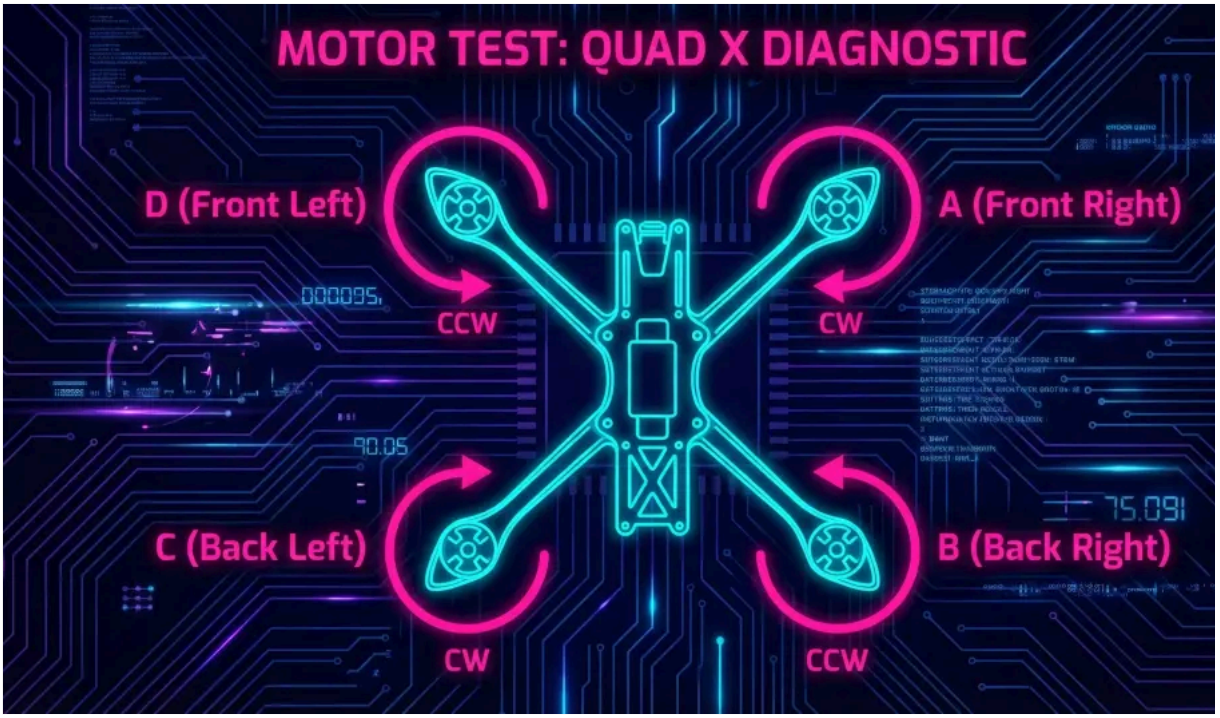
- It bypasses the EKF, the PIDs, and the IMU.
- *Why?* If the drone is on your workbench and you move it, the PIDs would normally try to spin the motors to "level" it. Motor Test disables this, sending a fixed PWM value directly to the motor. This is why it only works while the vehicle is **Disarmed**.

ARCHITECTURE (THE ENGINEER'S VIEW)

The logic operates via the MAVLink command `MAV_CMD_DO_MOTOR_TEST`.

1. THE TEST SEQUENCE

The test does **not** simply say "Spin Motor 1". It says "Spin the 1st motor in the sequence".



- **The Sequence:** Defined in the `setup_motors` function by the `testing_order` parameter.
- **Standard:** Clockwise starting from Front Right (A, B, C, D...).
- **Custom:** You define this order when calling `add_motor`.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

- *Code Path:* `handle_MAV_CMD_DO_MOTOR_TEST()`.

2. EXECUTION LOGIC

1. **Safety Check:** The vehicle must be Disarmed (usually). The Safety Switch must be active.
2. **Timeout:** The spin runs for a set duration (e.g., 2 seconds) and then auto-stops.
3. **Passthrough:** The code bypasses the entire Attitude Controller. It injects a PWM value directly into the `SRV_Channels` output buffer.

3. VERIFYING WITH LOGS (`RCOUT`)

If the physical motor doesn't spin, checking the logs tells you if it's a **Software** or **Hardware** issue.

- **Log:** `RCOUT`.
- **Channels:** `C1`, `C2`, `C3`, etc.
- **Test:** Run the Motor Test.
- **Analysis:**
  - *RCOUT shows 1100pwm:* The software is working. The issue is your wiring, ESC, or BEC.
  - *RCOUT shows 1000pwm (Min):* The software isn't trying to spin it. Check your `SERVOx_FUNCTION` mapping.

COMMON ISSUES & TROUBLESHOOTING

- **"Command Rejected":** You are likely Armed. You must Disarm to run the test.
- **"Motor spins wrong direction":** This is an ESC setting (swap wires or use DShot Reverse), NOT a mixer setting.
- **"Wrong motor spins":** Your `SERVOx_FUNCTION` mapping is wrong, or your `testing_order` in the C++ definition is confusing.

SOURCE CODE REFERENCE

- **Command Handler:** `GCS_MAVLINK_Copter::handle_MAV_CMD_DO_MOTOR_TEST()`

DSHOT & ESC TELEMETRY BACKEND

EXECUTIVE SUMMARY

DShot is a digital protocol for controlling ESCs. Unlike PWM (which varies pulse width), DShot sends a packet of data (Thrust + Telemetry Request + Checksum). This eliminates calibration issues and allows for **Bi-Directional Communication**: the ESC can report its RPM back to the flight controller on the *same wire* immediately after receiving a command.

THEORY & CONCEPTS

1. DIGITAL COMMUNICATION VS. ANALOG PWM

- **Analog (PWM):** The Flight Controller sends a pulse. The length of the pulse (1ms to 2ms) determines the speed.
  - *Problem:* Small voltage drops in the wires can change the pulse length. The ESC and FC must be calibrated to agree on what "1.5ms" looks like.
- **Digital (DShot):** The Flight Controller sends a digital number (e.g. `1024`).
  - *Benefit:* There is no calibration. `1024` always means exactly `1024`. It includes a **Checksum** (CRC), so if electrical noise corrupts the bit, the ESC ignores the packet rather than spinning up to full throttle.

2. GCR ENCODING

DShot uses **GCR (Group Code Recording)** encoding. This ensures there are never too many "zeros" or "ones" in a row.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- *Why?* High-speed serial links need frequent transitions (edge triggers) to keep the clocks on the FC and ESC synchronized. GCR guarantees these edges, allowing for reliable data transfer at up to 1.2 Mbit/s over simple wires.

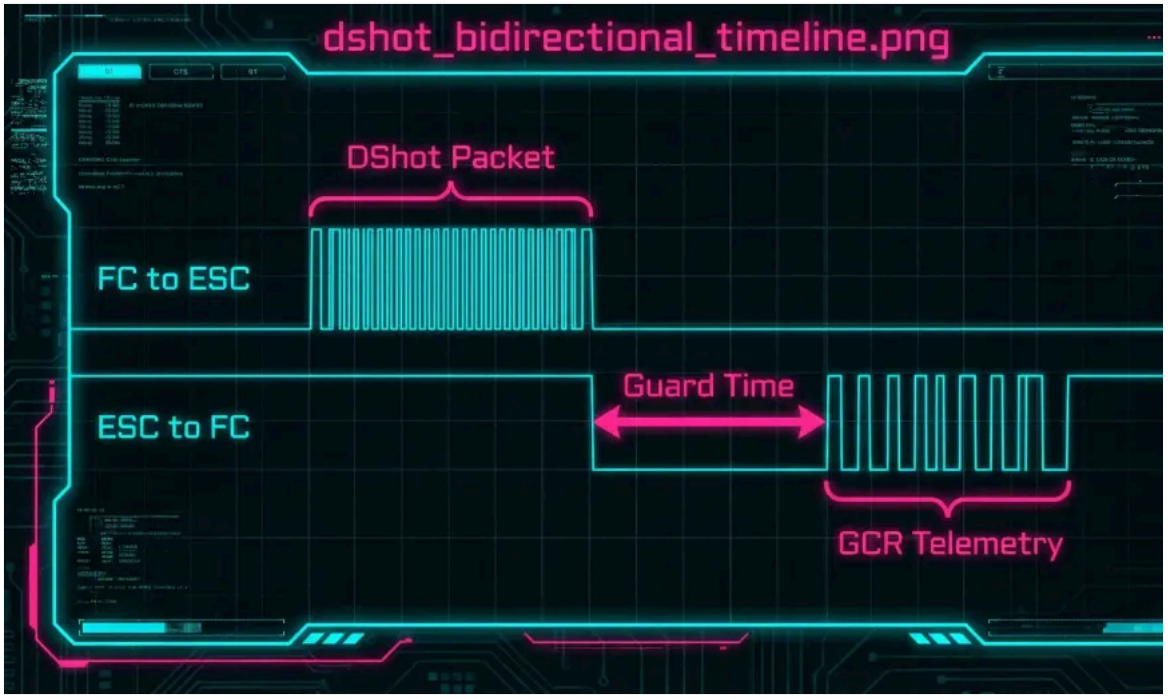
ARCHITECTURE (THE ENGINEER'S VIEW)

1. DSHOT TRANSMISSION

- **Protocol:** `MOT_PWM_TYPE` selects the speed (DShot150, 300, 600, 1200).
- **DMA (Direct Memory Access):** The CPU does not toggle the pins manually (too slow/jittery).
  - It creates a buffer of "High" and "Low" bit patterns in RAM.
  - It triggers a DMA transfer to the Timer hardware.
  - The Timer pushes the waveform out to the pin perfectly.
  - *Code Path:* `RCOutput::set_dshot_rate()`.

2. BI-DIRECTIONAL TELEMETRY (RPM)

- **The Problem:** We want RPM data for the Harmonic Notch Filter, but we don't want extra wires.
- **The Solution:**



1. Flight Controller sends DShot command (e.g., 20us).
2. Flight Controller flips the pin to **Input Mode**.
3. ESC waits 20us, then sends a GCR-encoded pulse stream representing RPM.
4. Flight Controller captures this stream using **Input Capture DMA**.
5. Flight Controller decodes it and feeds it to the EKF / Notch Filter.
   - *Code Path:* `bdshot_decode_telemetry_packet()`.

HARDWARE CONSTRAINTS

- **Timer Groups:** DShot channels are grouped by hardware timers. All channels on a single timer *must* run at the same DShot rate.
- **DMA Conflict:** Bi-Directional DShot requires a DMA channel for *both* TX and RX. Some flight controllers run out of DMA channels, forcing you to disable Bi-Dir on some outputs.

KEY PARAMETERS

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

PARAMETER	DEFAULT	DESCRIPTION
`MOT_PWM_TYPE`	0	4=DShot150, 5=DShot300, 6=DShot600.
`SERVO_BLH_AUTO`	1	Enable automatic ESC mapping for BLHeli passthrough.
`SERVO_DSHOT_ESC`	0	Which DShot type the ESCs support (for capability checks).

SOURCE CODE REFERENCE

- **Telemetry Decoder:** `RCOutput::bdshot_decode_telemetry_packet()`

PRACTICAL GUIDE: ENABLING DSHOT & RPM

Forget "DShot Calibration." DShot doesn't need calibration. But it does need configuration.

STEP 1: SET PROTOCOL

1. **Parameter:** `MOT_PWM_TYPE = 6` (DShot600).
2. **Reboot:** You MUST reboot the flight controller to switch the timer hardware from PWM mode to DShot mode.

STEP 2: ENABLE BI-DIRECTIONAL TELEMETRY

1. **Parameter:** `SERVO_BLH_AUTO = 1`. This enables the BLHeli Passthrough/Telemetry backend.
2. **Parameter:** `SERVO_BLH_TRATE = 10`. Sets the telemetry update rate (10Hz). *Note:* This is for the slow "Temperature/Volts" telemetry, not the fast RPM telemetry.
3. **Parameter:** `SERVO_BLH_BDMASK`. Set the bitmask for your motors.
    - *Example:* For a Quad on outputs 1-4, value is **15** (binary 1111).
4. **Reboot.**

STEP 3: VERIFY

1. Connect battery.
2. Go to Mission Planner → **Status**.
3. Look for `esc1_rpm`, `esc2_rpm`.
4. Spin the motors (props off). If you see RPM values changing, you are ready for Harmonic Notch tuning.

FRAME PHYSICS: GEOMETRY, RESONANCE, AND MATERIALS

CRITICAL The physical frame is not just a holder for components; it is a structural element that defines the control authority and vibration characteristics of the vehicle. A "floppy" frame cannot be tuned perfectly, no matter how good the software is.

1. FRAME GEOMETRY: THE THRUST MAP

The layout of the motors defines how the PID loop interacts with the physical world.

1.1 TRUE X

- **Layout:** Motors are equidistant from the center and form a perfect square.
- **Physics:** Roll authority equals Pitch authority.
- **Pros:** Perfectly symmetrical handling. Easy to tune. The "Gold Standard" for freestyle and racing.
- **Cons:** Propellers are often visible in the FPV camera view.

1.2 WIDE X (SQUASHED X)

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- **Layout:** Motors are wider apart laterally than longitudinally.
- **Physics:** Roll inertia is higher than Pitch inertia. Roll authority is slightly lower.
- **Pros:** Moves props out of the camera view.
- **Cons:** Requires asymmetric PID tuning (Pitch P often needs to be higher than Roll P). Separation of "Clean" vs "Dirty" air is different for front/rear props.

1.3 DEADCAT (DC)

- **Layout:** Front arms are swept back to clear the camera view. Rear arms are standard. The motors are *not* equidistant from the Center of Gravity (CG).
- **The Physics Problem:** The front motors are often closer to the CG (or further) than the rear.
- **Torque Imbalance:** In a fast yaw spin or snap roll, the lever arms are different. The FC must work harder to mix the motors correctly.
- **Tuning:** Deadcat frames often have "yaw washout" or strange dipping behavior in hard turns because the thrust vector center doesn't match the mass center.

---

2. MATERIAL SCIENCE: STIFFNESS IS SPEED

PID loops react to error. If the frame bends, the motor moves, but the Gyro (mounted in the center) doesn't feel it immediately. This delay allows the error to grow.

2.1 CARBON FIBER GRADES

- **Toray T300:** Standard "Hobby Grade." Good cost/performance ratio.
- **Toray T700:** High Tensile Strength. Used in premium frames. It is stiffer and less prone to delamination on impact.
- **Weave:**
  - **3K Twill:** The standard "checkerboard" look. Good strength in both X and Y directions.
  - **Unidirectional:** All fibers run parallel. Extremely stiff in one direction, very weak in the other. Used in arm cores ("sandwiched" carbon) to maximize arm stiffness against upward thrust.

2.2 RESONANCE AND HARMONICS

Every object has a natural frequency at which it vibrates.

- **The Goal:** Push the frame's resonant frequency **above** the operating frequency of the motors.
- **Stiff Frame:** Resonates at > 150Hz.
- **Floppy Frame:** Resonates at < 80Hz.
- **The Problem:** If the frame resonates at 100Hz, and your motors are spinning at 100Hz (mid-throttle), the vibration amplifies violently. The Gyro sees this noise and tries to correct it, causing the motors to twitch, creating *more* noise. This is a feedback loop (D-Term Oscillation).

---

3. VIBRATION MANAGEMENT: THE CLEAN GYRO

3.1 SOFTWARE FILTERING: THE NOTCH

ArduPilot uses a **Dynamic Harmonic Notch Filter**.

- **Function:** It tracks the throttle position (or ESC telemetry RPM) to predict the motor frequency.
- **Action:** It places a narrow "Notch" filter exactly at that frequency to erase the noise before it reaches the PID controller.
- **Result:** You can fly a noisy, bent, or beat-up frame smoothly because the computer ignores the specific frequency of the vibration.

3.2 MECHANICAL ISOLATION

- **Stack Isolation:** "Gummies" (silicone grommets) decouple the Flight Controller mass from the frame.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- **Cable Management:** A loose capacitor or battery wire tapping against the Gyro chip creates massive spikes that no filter can fix. **Tape down all wires.**
- **The "Mass-Spring-Damper" Model:**
  - **Mass:** The Flight Controller.
  - **Spring:** The Gummies.
  - **Damper:** The internal friction of the rubber.
  - **Tuning:** If the gummies are too soft, the FC "bounces" (low frequency oscillation). If too hard, they transmit motor noise (high frequency).

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

CHAPTER 8: SENSOR ARCHITECTURE

GPS INTEGRATION

EXECUTIVE SUMMARY

Global Navigation Satellite Systems (GNSS), commonly referred to as GPS, provide the absolute position reference for autonomous vehicles. While a standard phone GPS is accurate to within 5 meters, modern drone operations often require centimeter-level precision. ArduPilot supports advanced configurations including **Dual GPS Blending** for reliability and **Real-Time Kinematic (RTK)** for precision. It can even use **Moving Baseline** technology to calculate the drone's heading (Yaw) without a magnetic compass.

THEORY & CONCEPTS

1. STANDARD VS. RTK GPS: THE PHYSICS OF PRECISION

To understand why RTK is necessary, we must understand how standard GPS works.

- **Time of Flight (Standard GPS):** A satellite sends a signal stamped with the time. The receiver calculates how long the signal took to arrive (speed of light). Because the atmosphere (Ionosphere/Troposphere) slows the signal down unpredictably, this timing is always slightly "fuzzy."
  - *Result:* Accuracy is limited to **~2-5 meters**.
- **Carrier Phase (RTK):** Instead of just timing the "pings" (Code Phase), an RTK receiver counts the actual **radio waves** (Carrier Phase) of the 1.5GHz signal. Since the wavelength is only ~19cm, counting waves allows for millimeter-level measurement.
  - *The Catch:* The receiver can count waves, but it doesn't know the *total* number of waves (Integer Ambiguity). It needs a **Base Station** at a known location to tell it "I see 5,000,002 waves."
  - *Result:* By subtracting the Base Station's error from the Rover's measurement, atmospheric errors cancel out. Accuracy improves to **~1-2 centimeters**.

2. MOVING BASELINE (COMPASS-LESS YAW)

Magnetic Compasses are the Achilles' heel of drones. They are easily confused by power lines, bridges, and even the drone's own motors. Moving Baseline GPS replaces the compass entirely.

- **The Setup:** You install two GPS antennas on the drone (e.g., one on the nose, one on the tail).
- **The Logic:** One GPS acts as a "Moving Base," sending corrections to the other "Rover."
- **The Magic:** Because the two antennas are so close (e.g., 30cm) and share the exact same atmospheric conditions, the relative vector between them can be calculated with extreme precision (< 1cm).
- **The Result:** If you know the nose is exactly 30cm in front of the tail, and you measure the vector between them, you can calculate the **Heading (Yaw)** using simple trigonometry. This heading is immune to magnetic interference.

3. DILUTION OF PRECISION (DOP)

You will often see "HDOP" (Horizontal Dilution of Precision) in the logs. This is a multiplier of error based on satellite geometry.

- **Bad Geometry:** All satellites are clustered in one part of the sky. The triangulation triangles are skinny. Small timing errors create huge position errors.
- **Good Geometry:** Satellites are spread out (North, South, East, West, Zenith). The triangles are wide.
- **Why Dual GPS Helps:** Two antennas have different views of the sky. If one is blocked by the drone's body (e.g., in a bank), the other might still have a clear view, maintaining a lower HDOP.

ARCHITECTURE (THE ENGINEER'S VIEW)

1. AUTO-SWITCHING (REDUNDANCY)

This is the default mode (`GPS_AUTO_SWITCH = 1`). The goal is to always use the *single best* sensor.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

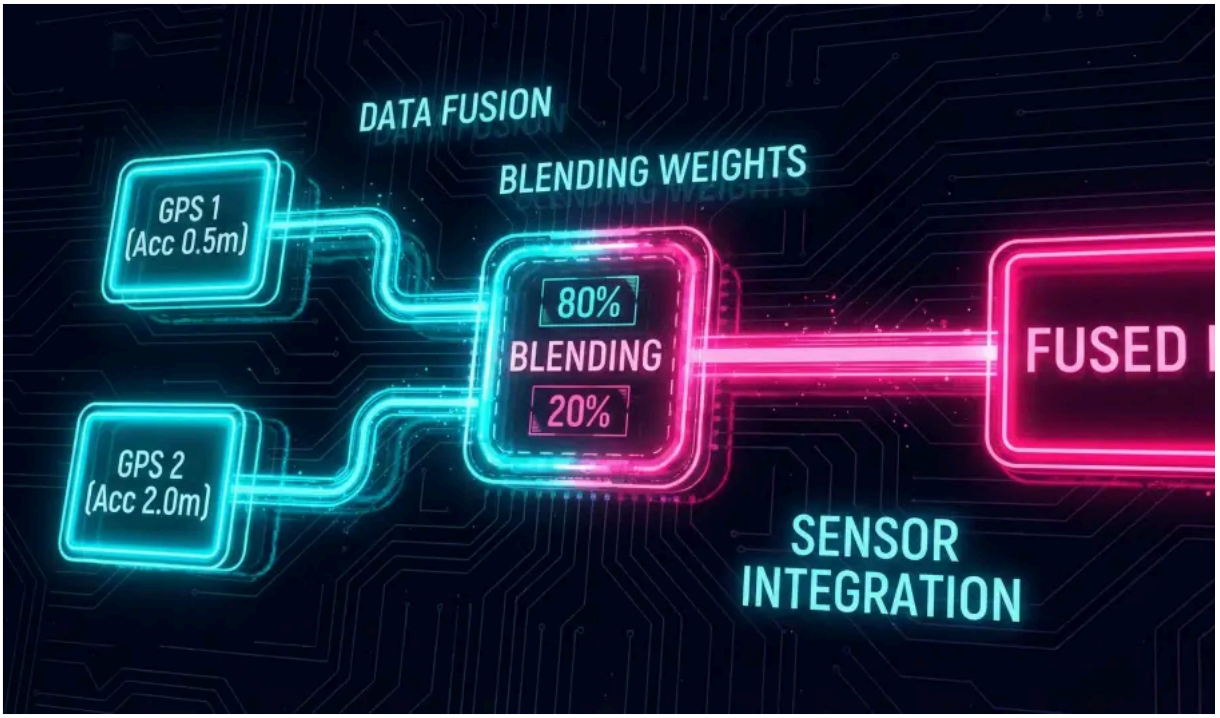




- **Logic:**
    1. **Fix Status:** Priority 1. An RTK Fixed unit beats an RTK Float unit, which beats a 3D Lock unit.
    2. **Satellite Count:** Priority 2. If Fix Status is equal, the unit with *more satellites* wins.
- **Hysteresis:** To prevent rapid toggling (which confuses the EKF), ArduPilot requires the new GPS to be "better" for a set time (e.g., 5 seconds for a 2-satellite lead).
- *Code Path:* `AP_GPS::update_primary()`.

2. GPS BLENDING (ACCURACY)

If enabled (`GPS_AUTO_SWITCH = 2`), ArduPilot creates a **Virtual GPS** instance.



- **Logic:** It calculates a weighted average of all healthy GPS units.
- **Weighting:** The weight is based on the reported **Accuracy** (Speed Accuracy, Horizontal Accuracy, Vertical Accuracy).
    - *Effect:* If GPS1 reports 0.5m accuracy and GPS2 reports 1.0m, GPS1 contributes 66\% to the blended solution.
- **Benefits:** Smoother flight tracks and glitch resistance. If one GPS wanders, the other pulls the average back.

3. MOVING BASELINE (GPS YAW)

This technique replaces the Magnetic Compass.

- **Hardware:** Two GPS antennas on the drone (Rover and Base).
- **Mechanism:** The "Base" sends RTCM corrections to the "Rover" over a serial wire. The "Rover" calculates the precise vector (baseline) to the "Base" with centimeter accuracy.
- **Result:** Since the antennas are fixed to the frame, this vector reveals the **Heading** of the drone.
- **Advantage:** Immune to magnetic interference (power lines, metal bridges).

4. VIRTUAL GPS & INDOOR POSITIONING

For professional indoor operations (stadiums, warehouses) where satellite signals are blocked, standard hardware GNSS is ineffective.

- **Solution:** Using an Android device as a "Virtual GPS" via the **MAVLink GPS** project.
- **Logic:** This system uses Android's **Fused Location Provider** (WiFi RTT, Bluetooth, and cellular anchors) to generate a high-precision `GPS_INPUT` message.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

- **Integration:** ArduPilot treats the phone as a standard GPS backend (`GPS_TYPE = 14`).
- **Best For:** GPS-denied environments where high-precision local infrastructure (WiFi APs) is available. See the **Venue Optimization Guide** for setup details.

KEY PARAMETERS

PARAMETER	DEFAULT	DESCRIPTION
`GPS_AUTO_SWITCH`	1	0=Disabled, 1=Use Best, 2=Blend, 4=Use Primary (if 3D fix).
`GPS_BLEND_MASK`	5	Which metrics to use for weighting (1=Horiz Pos, 2=Vert Pos, 4=Speed).
`GPS_POS_X/Y/Z`	0	Antenna position offsets. Critical for Moving Baseline calculations.
`GPS_TYPE`	1	1=Auto, 17=UBlox, 18=NovaTel. Setting specific types speeds up boot time.

SOURCE CODE REFERENCE

- **Switching Logic:** `AP_GPS::update_primary()`
- **Blending Logic:** `AP_GPS_Blended::calc_state()`

PRACTICAL GUIDE: CONFIGURING DUAL GPS

Why buy two GPS units? Because redundancy prevents flyaways.

OPTION A: USE BEST (REDUNDANCY)

- **Parameter:** `GPS_AUTO_SWITCH = 1`
- **Behavior:** ArduPilot uses GPS1. If GPS1 loses 3D lock or its HDOP spikes, it switches to GPS2.
- **Best For:** Systems with one "Good" GPS (RTK) and one "Cheap" GPS (Backup). You don't want to blend the cheap data into the good data.

OPTION B: BLENDING (ACCURACY)

- **Parameter:** `GPS_AUTO_SWITCH = 2`
- **Parameter:** `GPS_BLEND_MASK = 5` (Horizontal + Vertical Position)
- **Behavior:** ArduPilot computes a weighted average.
- **Best For:** Two identical GPS units (e.g., dual Here4).
- **Warning:** Blending requires both units to report accurate error metrics. If a cheap clone GPS reports "0.1m accuracy" when it's actually drifting 5m, it will corrupt the blended solution. **Only blend identical, high-quality units.**

RANGEFINDERS: TILT COMPENSATION

EXECUTIVE SUMMARY

Rangefinders (Lidar, Sonar, Radar) provide the critical "Height Above Ground" (AGL) metric needed for automated landing and terrain following. While a Barometer tells you how high you are above your takeoff point, a Rangefinder tells you how close you are to crashing into the hill you are flying over. ArduPilot applies sophisticated trigonometry to convert the raw "slant range" from these sensors into a useful vertical altitude.

THEORY & CONCEPTS

1. SENSOR PHYSICS: LIGHT VS. SOUND VS. RADIO

Not all rangefinders are created equal.

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

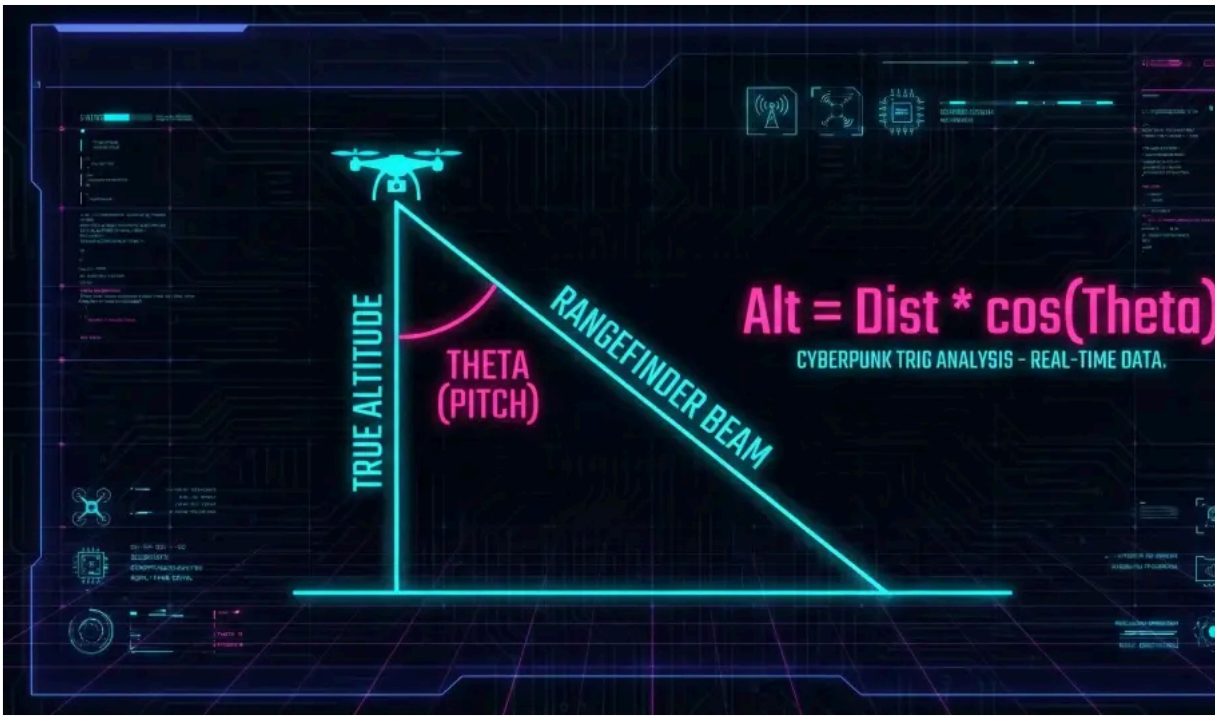




- **Lidar (Light Detection and Ranging):** Uses pulsed laser light.
  - *Pros:* Extremely fast update rates (500Hz+), very precise (mm).
  - *Cons:* Fails on **Black Surfaces** (absorbs light), **Mirrors/Water** (reflects light away), and in **Bright Sunlight** (saturation). It is also blinded by fog/dust.
- **Sonar (Ultrasonic):** Uses sound waves.
  - *Pros:* Works on glass/water/transparent surfaces. Immune to light conditions.
  - *Cons:* Very slow (speed of sound is slow). Wide "cone" picks up grass/obstacles easily. Noisy in prop-wash (air turbulence).
- **Radar (Radio Detection):** Uses mmWave radio.
  - *Pros:* Sees through dust, fog, and light rain. Robust against surface type.
  - *Cons:* Larger, heavier, more expensive.

2. THE "SLANT RANGE" PROBLEM

Rangefinders are "dumb." They measure the distance to the first thing they hit. If you pitch the drone forward 45 degrees, the sensor points forward, not down.



- **The Error:** The measured distance becomes the **Hypotenuse** of the triangle.
- **The Result:** If you don't correct for this, pitching forward makes the drone think it is "climbing" (distance increasing), so the autopilot cuts the throttle to compensate, causing a crash.

3. SURFACE TEXTURE & DIVERGENCE

- **Beam Divergence:** A laser beam spreads out over distance. At 50m, the "dot" might be 1m wide.
- **The "Edge" Problem:** If the beam hits the edge of a building, half the light returns from the roof (5m away) and half from the ground (20m away). The sensor averages this to 12.5m—a ghost measurement that exists nowhere.
- **Filtering:** ArduPilot uses a "Glitch Filter" to reject sudden jumps that are physically impossible for the drone to fly.

ARCHITECTURE (THE ENGINEER'S VIEW)

1. THE RAW MEASUREMENT

The `AP_RangeFinder` library reports the raw distance returned by the hardware driver.

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

- *Input:* `Distance_Raw`.
- *Correction:* None (at this layer).

2. THE TRIGONOMETRY CORRECTION

The correction logic resides in `AP_SurfaceDistance::update()` (or inside the EKF).

- **Formula:** `Altitude = Distance_Raw * cos(Roll) * cos(Pitch)`
- **Mechanism:** It accesses the Attitude Heading Reference System (AHRS) to get the current rotation matrix. It projects the laser vector onto the vertical Z-axis.

3. THE "CONE EFFECT" (GEOMETRIC ERROR)

Even with perfect math, physics gets in the way.

- **Scenario:** You bank 30 degrees right over a slope.
- **The Problem:** The laser is now hitting the ground *upslope* or *downslope* relative to the drone. The math assumes the ground is flat. This results in "Phantom Altitude Jumps" when maneuvering over rough terrain.

KEY PARAMETERS

PARAMETER	DEFAULT	DESCRIPTION
`RNGFND1_TYPE`	0	Driver selection (e.g., LightWare, MaxBotix, Benewake).
`RNGFND1_ORIENT`	25	(Down). Defines which way the sensor points. If set to `0` (Forward), the math changes completely (Obstacle Avoidance).
`RNGFND_GAIN`	0.5	Used by AltHold to blend Lidar vs Baro. Higher gain trusts Lidar more.

SOURCE CODE REFERENCE

- **Math Logic:** `AP_SurfaceDistance::update()`

OPTICAL FLOW: SENSOR DRIVERS & QUALITY

EXECUTIVE SUMMARY

Optical Flow sensors (like the PX4Flow or HereFlow) give drones the ability to hold position without GPS. They work like an optical mouse, tracking the movement of texture on the ground. However, they are highly sensitive to lighting, focus, and vibration. ArduPilot's driver layer is responsible for normalizing this complex data into a format the EKF can consume.

THEORY & CONCEPTS

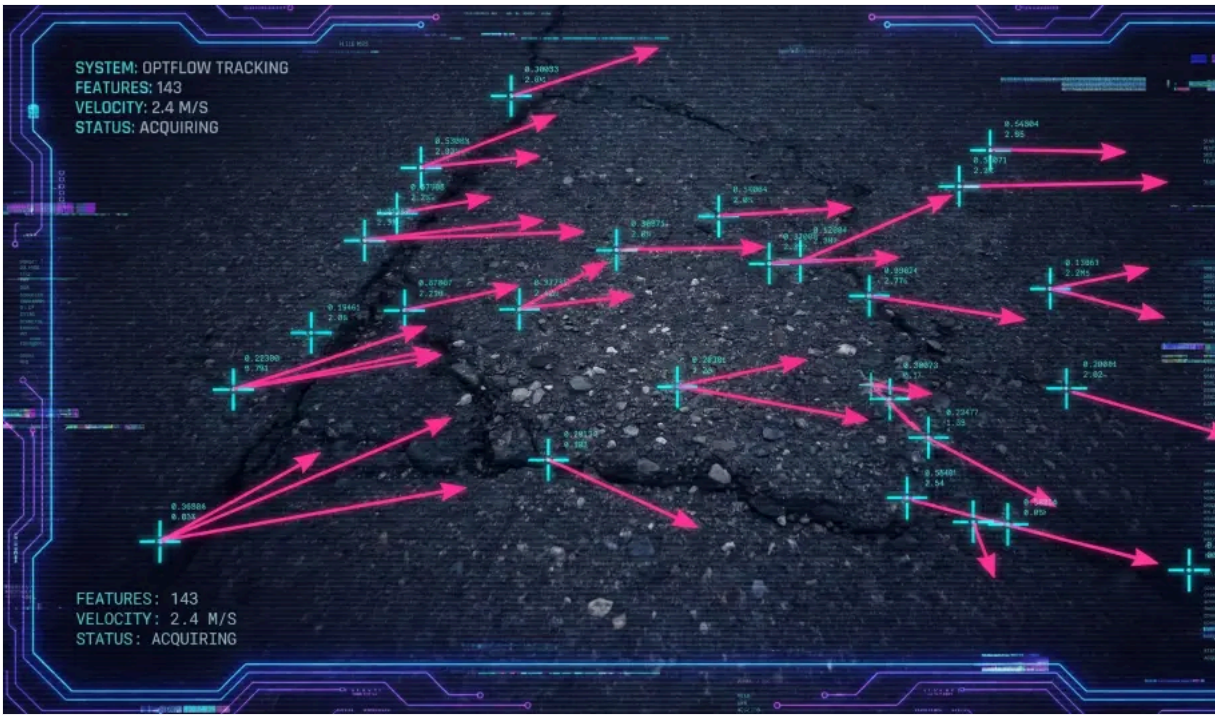
1. HOW IT WORKS (THE LUCAS-KANADE METHOD)

Most flow sensors use a variation of the Lucas-Kanade algorithm.

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY







- **Frame Comparison:** The sensor takes two photos in rapid succession.
- **Feature Tracking:** It identifies "corners" or high-contrast edges in Frame 1 and finds them in Frame 2.
- **Vector Calculation:** The distance these features moved (in pixels) divided by the time (dt) gives the **Flow Rate**.

2. THE "APERTURE PROBLEM" (TEXTURE FAILURE)

If you fly over a featureless surface (like a white tile floor or calm water), the sensor sees nothing.

- **The Result:** The algorithm cannot find matching features. It reports zero flow or chaotic noise.
- **ArduPilot's Defense:** The `Quality` metric. If the sensor can't find features, it drops the Quality score. The EKF sees this and refuses to use the data, triggering a failsafe (Land/AltHold) rather than flying away.

3. THE "BLUR" PROBLEM (FOCUS & VIBRATION)

- **Focus:** If the lens is out of focus, edges become gradients. The feature tracker fails. A sharp focus at the flying height is critical.
- **Motion Blur:** If the drone vibrates or rotates too fast, the image smears. The features disappear. This is why good lighting (fast shutter speed) and vibration isolation are mandatory for FlowHold.
- **Light Flicker:** Indoor lights flicker at 50Hz or 60Hz. This can beat-frequency with the camera shutter, causing "rolling bands" in the image that confuse the flow algorithm.

ARCHITECTURE (THE ENGINEER'S VIEW)

1. DATA STRUCTURE

The driver returns a `OpticalFlow_state` struct:

- `flowRate` (Vector2f): The total angular motion seen by the camera (rad/s).
- `bodyRate` (Vector2f): The angular motion caused by the drone's rotation (from a gyro).
- `quality` (uint8_t): Confidence score (0-255).

2. GYRO COMPENSATION (THE "FAKE MOTION" FIX)

- **Integrated Gyro (PX4Flow/HereFlow):** These sensors have a built-in rate gyro. They report their *own* rotation as `bodyRate`. This is ideal because the gyro and camera are physically coupled and time-synchronized on the sensor board.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

- **No Gyro (CXOF):** Cheaper sensors only report flow. The driver "fakes" the `bodyRate` by copying the Flight Controller's main Gyro (`AP::[ahrs](/field-manual/mavlink-interface/ahrs.html)().get_gyro()`).
  - *Risk:* If the main Gyro suffers from vibration aliasing, it corrupts the Flow compensation too.
  - *Latency:* There is a time delay between the camera capturing the frame and the FC reading the Gyro, leading to imperfect compensation during rapid maneuvers.

3. SURFACE QUALITY

Different drivers scale "Quality" differently to match the ArduPilot 0-255 standard.

- **CXOF:** Raw values (64-78) are expanded to fit 0-255.
- **PX4Flow:** Native 0-255 output.
- *Threshold:* `FLOW_MIN_QUAL` tells the EKF when to stop trusting the sensor.

KEY PARAMETERS

PARAMETER	DEFAULT	DESCRIPTION
`FLOW_TYPE`	1	1=PX4Flow, 4=CXOF, 6=UAVCAN (HereFlow).
`FLOW_POS_X/Y/Z`	0	Sensor offset. Critical for lever-arm compensation.
`FLOW_MIN_QUAL`	50	Minimum quality to fuse data. Increase this if the drone twitches in low light.

SOURCE CODE REFERENCE

- **Frontend Logic:** `AP_OpticalFlow::update()`

PRACTICAL GUIDE: FOCUSING THE LENS

The #1 reason Optical Flow fails is a blurry lens.

THE PROBLEM

Most sensors ship focused for "Infinity" (Outdoor). If you fly indoors at 1.5m, the floor is blurry.

THE PROCEDURE

1. **Connect USB:** Power the drone. Connect to Mission Planner.
2. **Open OptFlow View:** Press Ctrl+F → "Mavlink Inspector" → Look for `OPTICAL_FLOW` message.
3. **Monitor Quality:** Look at the `quality` field.
4. **The Box Test:** Put a textured object (patterned box, rug) on the floor 1.5m away (or your target flight height).
5. **Rotate Lens:** Manually twist the sensor lens. Watch the `quality` score.
   - *Goal:* Maximize the score (usually > 200).
   - *Verify:* Move the box slightly. The values should change rapidly.
6. **Lock it:** Use a tiny drop of hot glue or the set screw to lock the lens focus.

AIRSPEED SENSORS (PITOT)

EXECUTIVE SUMMARY

For fixed-wing aircraft, Airspeed is life. While GPS gives "Ground Speed", only a Pitot Tube measures "Airspeed" (the speed of air over the wings). ArduPilot uses this to manage energy (TECS) and prevent stalls.

ARCHITECTURE (THE ENGINEER'S VIEW)

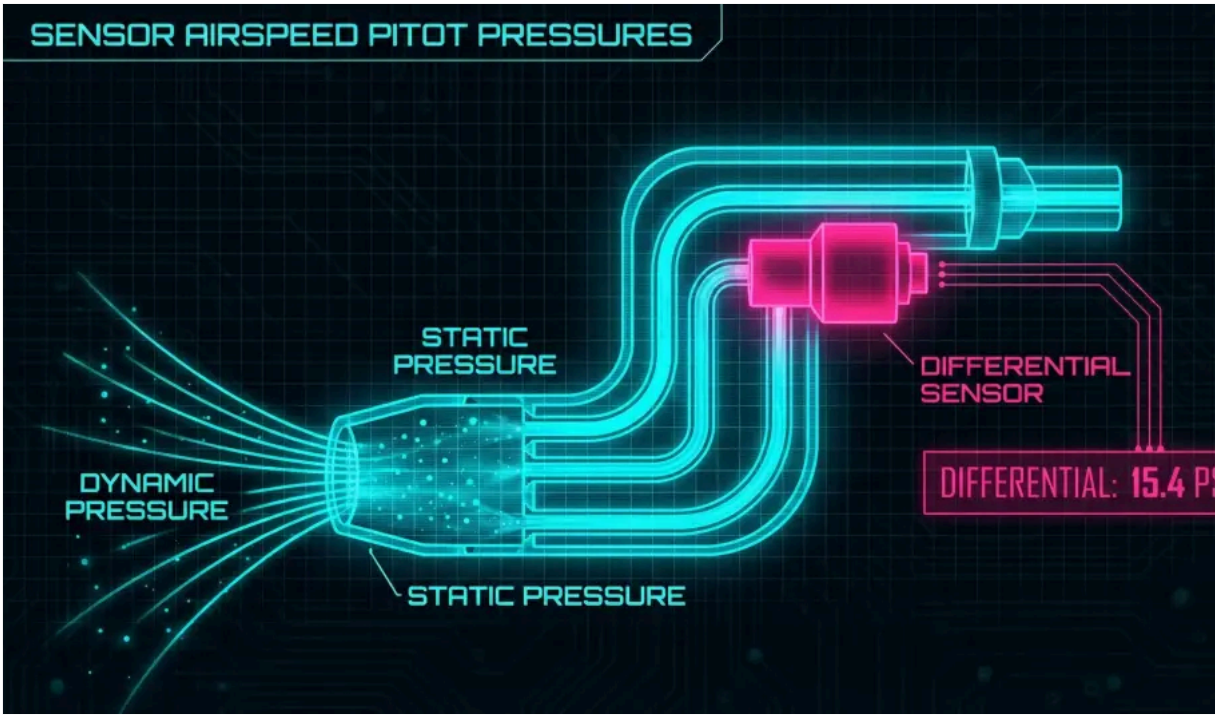
Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





1. THE PHYSICS (`ARSPD_RATIO`)

A Pitot tube measures Differential Pressure (Dynamic Pressure).



- **Formula:** `Pressure = 0.5 * Density * Velocity^2`.
- **ArduPilot:** `Airspeed = sqrt(Pressure) * ARSPD_RATIO`.
- *Calibration:* The `ARSPD_RATIO` accounts for tube length, sensor sensitivity, and air density. If this is wrong, the airspeed reading is wrong.

2. AUTO CALIBRATION (`ARSPD_AUTOCAL`)

ArduPilot can learn the `ARSPD_RATIO` in flight.

- **Logic:** It uses a 3-state Kalman Filter (Wind North, Wind East, Ratio).
- **Maneuver:** By flying in a circle, the ground speed changes (Headwind vs Tailwind), but the True Airspeed should stay relatively constant.
- **Solver:** The filter adjusts the `RATIO` until the calculated Wind Vector is consistent throughout the turn.
- *Code Path:* `Airspeed_Calibration::update()`.

3. FAILURE DETECTION (THE CLOG CHECK)

A clogged pitot tube is dangerous.

- **Scenario:** Tube blocked → Airspeed = 0 → Autopilot thinks "Stall!" → Pitches down → Crashes.
- **Detection:** The EKF monitors the consistency between GPS Speed and Airspeed.
  - If `GroundSpeed > 5` but `Airspeed < 2`, the sensor is flagged "Unhealthy".
  - *Action:* The plane switches to "Synthetic Airspeed" (Dead Reckoning using GroundSpeed - EstimatedWind).

KEY PARAMETERS

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

PARAMETER	DEFAULT	DESCRIPTION
`ARSPD_USE`	1	1=Use for flight control. 0=Log only.
`ARSPD_AUTOCAL`	0	0=Disabled, 1=Enable In-Flight Learning.
`ARSPD_RATIO`	2.0	Calibration factor. Typical range 1.5 - 3.0.

SOURCE CODE REFERENCE

- **Calibration Logic:** `Airspeed_Calibration::update()`

PRACTICAL GUIDE: THE LOITER CALIBRATION

Never trust the default `ARSPD_RATIO`. Calibrate it on the first flight.

STEP 1: CONFIGURATION

- Set `ARSPD_AUTOCAL = 1`.
- Take off in **FBWA** or **Manual**.

STEP 2: THE MANEUVER

1. Climb to a safe altitude.
2. Switch to **Loiter** mode.
3. Let the plane fly 5-6 complete circles.
    - *Why?* This exposes the plane to the wind from all angles. The EKF compares Ground Speed (GPS) vs Airspeed (Pitot) to solve for Wind Speed and Sensor Error.
4. Watch the GCS messages. You might see "Airspeed Calibration: Ratio 2.1".

STEP 3: SAVE AND DISABLE

1. Land.
2. Set `ARSPD_AUTOCAL = 0`.
3. **Check `ARSPD_RATIO`:** If it is between 1.5 and 3.0, it's good. If it's 6.0, check for leaks in your tubing.

BAROMETER PHYSICS

EXECUTIVE SUMMARY

The Barometer is the primary source of relative altitude for most drones. It measures atmospheric pressure. However, pressure changes with Weather, Temperature, and Propeller Wash. ArduPilot includes sophisticated compensation logic to prevent these factors from causing altitude glitches.

THEORY & CONCEPTS

1. HOW A MEMS BAROMETER WORKS

Modern drones use MEMS (Micro-Electro-Mechanical Systems) sensors.

- **The Mechanism:** Inside the chip, there is a microscopic silicone diaphragm with a vacuum on one side. As air pressure pushes on the diaphragm, it bends.
- **The Measurement:** Piezo-resistive strain gauges on the diaphragm change resistance as it bends. This resistance change is converted to a digital Pressure value.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- **The Problem (Temperature):** Silicone expands and contracts with heat. This expansion stresses the strain gauges *just like pressure does*. A hot sensor looks like a "High Pressure" sensor (low altitude).
- **The Solution:** Every sample includes a Temperature reading. The driver applies a polynomial curve (calibration coefficients stored in the sensor's ROM) to subtract the thermal expansion effect.

2. WEATHER VS. ALTITUDE (QNH)

- **Weather:** A low-pressure storm front moves in. The barometer drops. The drone thinks it climbed 50 meters.
- **Compensation:** ArduPilot does *not* track weather. It assumes "Pressure Change = Altitude Change".
- **Drift:** This is why Baro Altitude drifts over long flights (20+ minutes). GPS Altitude (though noisier) is absolute and does not drift with weather. The EKF fuses both to get the best of both worlds (Baro for short-term precision, GPS for long-term stability).

ARCHITECTURE (THE ENGINEER'S VIEW)

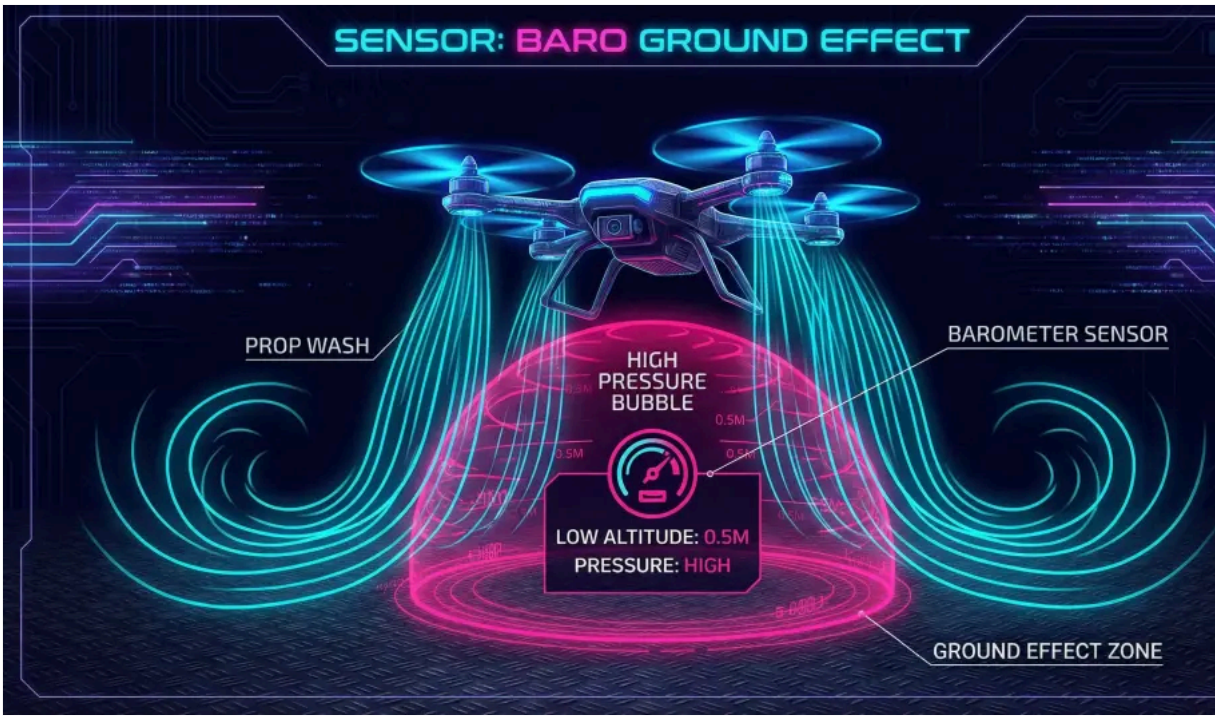
1. THE HYPSOMETRIC EQUATION

Converting Pressure to Altitude is not linear.

- **Formula:** `Altitude = 44330 * (1 - (Pressure / Ground_Pressure)^(1/5.255))`
- **Ground Pressure:** The EKF captures the "Ground Pressure" at arming. All flight altitudes are relative to this zero point.
- **Temperature:** Air density changes with heat. `AP_Baro` tracks `_ground_temperature` to scale the altitude calculation accurately.

2. GROUND STATIC EFFECT (GSE)

This is the most common issue on takeoff.



- **The Physics:** When a multicopter takes off, the propellers push a massive column of air downwards. This creates a high-pressure zone under the drone.
- **The Sensor:** The barometer sees High Pressure → Thinks "Below Sea Level".
- **The Result:** The drone thinks it is underground. When it climbs out of the ground effect (1-2m), the pressure normalizes. The EKF sees a sudden "Jump" in altitude.

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

- **Mitigation:** The EKF waits for the drone to climb a few meters before fully trusting the Barometer (relying on Accelerometer Z integration during the transition).

3. LIGHT SENSITIVITY

Many barometers (like the MS5611) are sensitive to UV light.

- **Scenario:** Sunlight hits the sensor through the case.
- **Result:** The silicon heats up instantly, causing a pressure spike. The drone "jumps" 1-2 meters.
- **Fix:** Open-cell foam covers the sensor to block light and wind while letting air pass.

KEY PARAMETERS

PARAMETER	DEFAULT	DESCRIPTION
`GND_TEMP`	0	(deg C) User override for ground temperature. 0 = Auto-detect.
`BARO_PROBE_EXT`	0	Probing mask for external barometers.
`EK3_GND_EFF_DZ`	4.0	(m) EKF Ground Effect Deadzone.

SOURCE CODE REFERENCE

- **Calibration:** `AP_Baro::update_calibration()`

PRACTICAL GUIDE: THE FOAM FIX

If your drone "twitches" vertically when sunlight hits it or when you fly fast, your barometer is exposed.

1. THE SYMPTOM

- **Sun:** You rotate the drone. As the sun hits the side of the flight controller, the altitude spikes 3-5 meters.
- **Speed:** You fly forward fast. The air pressure drops (Bernoulli's principle) inside the frame. The drone thinks it climbed, so it cuts throttle and descends.

2. THE SOLUTION

1. **Open the Frame:** Expose the Flight Controller (Cube, Pixhawk, etc.).
2. **Locate the Baro:** Look for a small metal can with a hole (MS5611) or a tiny black chip with a hole (BMP280).
3. **Apply Foam:** Glue a piece of **Open Cell Foam** (dark gray packing foam) over the sensor.
    - *Critical:* The foam must be breathable (air passes through) but opaque (blocks light).
    - *Test:* Shine a flashlight on it. If the altitude changes on the HUD, add more foam.

MAGNETOMETER MANAGEMENT

EXECUTIVE SUMMARY

The Magnetometer (Compass) is critical for heading, but also the most problematic sensor due to electromagnetic interference. ArduPilot supports multiple compasses, automatic prioritization, and complex orientation handling.

THEORY & CONCEPTS

1. DECLINATION: TRUE NORTH VS. MAGNETIC NORTH

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

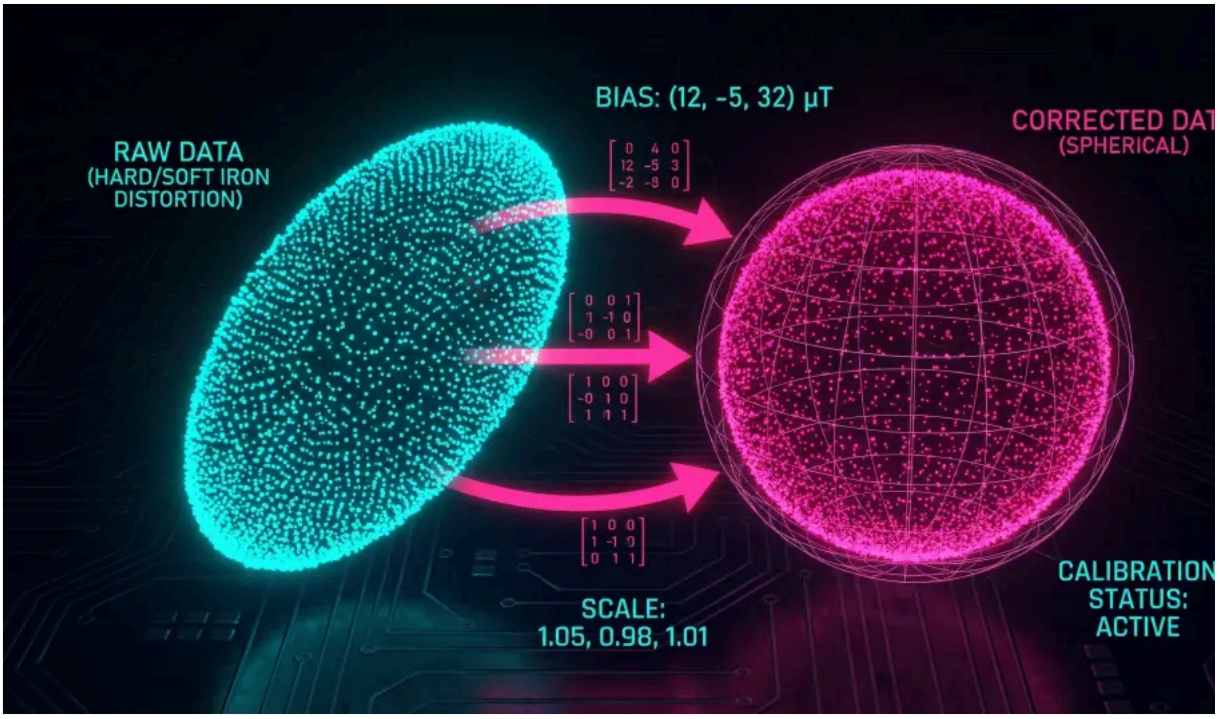




- **The Problem:** GPS navigation works on **True North** (Geographic Pole). Compasses point to **Magnetic North** (which wanders around Canada/Siberia).
- **The Correction:** ArduPilot uses a lookup table (World Magnetic Model) to find the "Declination" (offset) for your GPS location.
- *Why it matters:* If you are in New Zealand, the difference is ~20 degrees. If you don't correct for this, your drone will fly sideways (crab) when trying to fly a straight line.

2. HARD IRON VS. SOFT IRON CALIBRATION

- **Hard Iron (Offsets):** A permanent magnet attached to the frame (e.g., a screw, a speaker, or the PDB).
  - *Effect:* It adds a constant bias to the field.
  - *Calibration:* Calculates an X/Y/Z offset to subtract this bias.
- **Soft Iron (Scaling):** Ferrous metal (steel/iron) that bends the magnetic field lines.
  - *Effect:* It stretches the "Magnetic Sphere" into an ellipsoid.
  - *Calibration:* Calculates a scaling matrix (diagonals) to squish the ellipsoid back into a sphere.



3. MOTOR INTERFERENCE (EMF)

- **The Physics:** Current flowing through a wire creates a magnetic field (Ampere's Law).
- **The Effect:** When you throttle up, the ESC wires generate a field that twists the compass heading.
- **Compassmot:** A calibration routine that measures the magnetic shift *per Amp of current*. ArduPilot subtracts this dynamic offset in real-time.

ARCHITECTURE (THE ENGINEER'S VIEW)

1. INTERNAL VS EXTERNAL

- **Internal:** Soldered to the Flight Controller.
- **Pros:** Convenient.
- **Cons:** Very close to high-current power wires (ESCs). **Extremely noisy.**
- **Handling:** Rotated automatically by `AHRS_ORIENTATION`.
- **External:** Mounted on the GPS mast.
- **Pros:** Far from interference.
- **Cons:** Requires manual orientation.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

- **Handling:** Rotated manually by `COMPASS_ORIENT`. It **ignores** `AHRS_ORIENTATION`.

2. PRIORITY LOGIC

ArduPilot maintains a priority list (`COMPASS_PRIO1_ID`, etc.).

- **Selection:** The EKF consumes data from the **First Healthy Compass** in the priority list.
- **Recommendation:** Always set your External GPS Compass as `PRIO1_ID` and disable the Internals (`COMPASS_USE2 = 0`, `COMPASS_USE3 = 0`). This prevents the EKF from falling back to a noisy internal sensor if the external one glitches momentarily.

3. ORIENTATION LOGIC

The math is specific:

1. **Read Raw:** `(X, Y, Z)` from sensor.
2. **Apply Orientation:** Apply `COMPASS_ORIENT` rotation (e.g., Yaw 180).
3. **Result:** The vector must align with the Vehicle's Body Frame (X=Forward, Y=Right, Z=Down).

- *Common Error:* If you mount a GPS 180 deg rotated, you must set `COMPASS_ORIENT` to `Yaw180`. If you set it to `None`, the compass will point backwards, causing "Toilet Bowling".

KEY PARAMETERS

PARAMETER	DEFAULT	DESCRIPTION
`COMPASS_USE`	1	Enable/Disable specific compasses. Disable internals!
`COMPASS_ORIENT`	0	Rotation of the external sensor relative to the frame.
`COMPASS_LEARN`	0	Legacy "In-Flight Learning". Prefer EKF learning (`EK3_MAG_CAL`) instead.

SOURCE CODE REFERENCE

- **Read Loop:** `Compass::read()`

PRACTICAL GUIDE: MANAGING PRIORITIES

The default behavior (using all compasses) is often dangerous because internal compasses are noisy.

STEP 1: IDENTIFY

1. Go to **Setup** -> **Mandatory Hardware** -> **Compass**.
2. Look at the IDs.
    - **External:** Usually Compass 1 (on the GPS mast).
    - **Internal:** Usually Compass 2 & 3 (inside the Cube/Pixhawk).

STEP 2: REORDER (PRIORITY)

1. Ensure the **External Compass** is at the top of the list (`COMPASS_PRIO1_ID`).
2. If Compass 1 is Internal, use the "Up/Down" buttons to swap priorities.

STEP 3: DISABLE INTERNALS

1. Uncheck "Use this compass" for Compass 2 and 3.
2. **Why?** If the external compass fails, falling back to a noisy internal compass (which might be reporting "North is East" due to battery current) is often *worse* than having no compass at all.
3. **Exception:** If you have dual external GPS units, use both (`PRIO1` and `PRIO2`).

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





RPM SENSORS: THE HARMONIC NOTCH

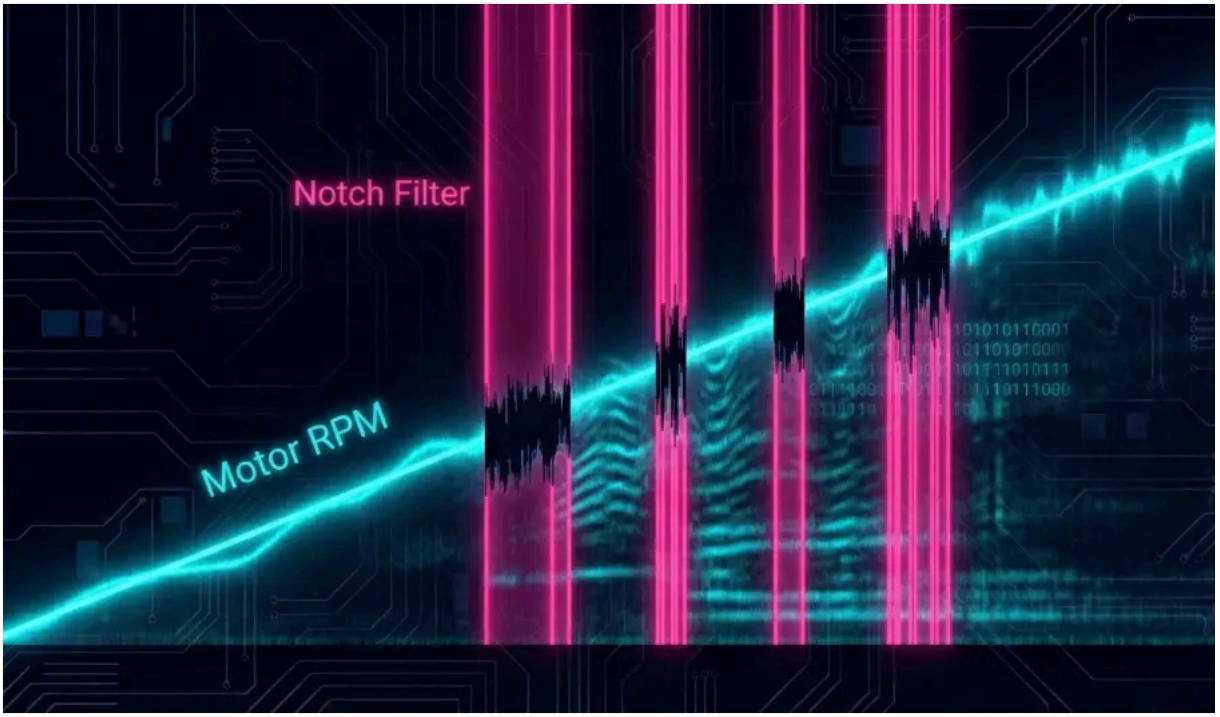
EXECUTIVE SUMMARY

Vibration is the enemy of stability. Propellers create massive vibration spikes at specific frequencies related to their RPM. A static Low-Pass Filter is blunt—it cuts noise but adds latency (lag), making the drone feel sluggish. The **Dynamic Harmonic Notch Filter** is a surgical tool: it tracks the RPM of the motors in real-time and deletes *only* the propeller noise, leaving the pilot's control inputs untouched.

THEORY & CONCEPTS

1. VIBRATION PHYSICS (THE PROPELLER)

A spinning propeller is an unbalanced mass.



- **Fundamental Frequency (1x):** `RPM / 60`. If props spin at 6000 RPM, the noise is at 100 Hz.
- **Harmonics (2x, 3x):** Blades flex, motors cog. This creates echoes at 200 Hz, 300 Hz, etc.
- **The Target:** We want to erase these specific spikes from the gyro data.

2. SIGNAL ALIASING (THE "WAGON WHEEL" EFFECT)

If your IMU samples at 1kHz, but the vibration is at 900Hz, the vibration will "fold back" (alias) and look like 100Hz noise.

- **Why Notch Filters Matter:** By removing the noise *before* it aliases (using high-rate sampling and filtering), we prevent the EKF from seeing ghost motion.

3. NOTCH VS. LOW PASS

- **Low Pass:** Like a heavy blanket. Muffles everything above 50Hz.
  - *Pro:* Simple.
  - *Con:* Delays the signal. The drone reacts late to gusts.
- **Notch:** Like a laser scalpel. Removes 100Hz +/- 5Hz.

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

- *Pro:* Zero latency for other frequencies.
- *Con:* Needs to know *exactly* where the noise is. This is why RPM sensors are critical.

ARCHITECTURE (THE ENGINEER'S VIEW)

1. THE SOURCE (`AP_RPM`)

- **ESC Telemetry:** Bi-directional DShot reports the RPM of every motor hundreds of times per second.
- **Hall Sensor:** A magnet on the bell triggers a GPIO pin interrupt.
- *Normalization:* `AP_RPM` converts these disparate sources into a unified `RPM` float value.

2. THE FREQUENCY CALCULATION (`AP_VEHICLE`)

The vehicle logic converts RPM to Frequency (Hz).

- **Fundamental Frequency:** `Freq = RPM / 60`.
- **Scaling:** `INS_HNTCH_REF` scales this value (e.g., if you want to target the 2nd harmonic).
- **Update Rate:** The filter center frequency is updated at 200Hz (or Loop Rate if configured).

3. THE FILTER (`HARMONICNOTCHFILTER`)

This is a digital signal processing filter running on the Gyro/Accel data.

- **Inputs:** Raw Gyro Data (1kHz - 8kHz).
- **Mechanism:** It creates a "Notch" (infinite attenuation) at the target frequency.
- **Result:** The noise is removed *before* it hits the PID controllers. This allows you to raise P-Gains significantly higher without oscillation.

KEY PARAMETERS

PARAMETER	DEFAULT	DESCRIPTION
`INS_HNTCH_ENABLE`	0	Master switch.
`INS_HNTCH_MODE`	0	1=Throttle, 4=RPM (Best).
`INS_HNTCH_FREQ`	80	(Hz) Base frequency (fallback if RPM lost).
`INS_HNTCH_BW`	40	(Hz) Bandwidth (width of the notch).

SOURCE CODE REFERENCE

- **Update Logic:** `AP_Vehicle::update_dynamic_notch()`

PRACTICAL GUIDE: SETTING UP BI-DIRECTIONAL DSHOT NOTCH

This is the standard setup for modern quads using BLHeli_32 or AM32 ESCs.

PREREQUISITES

1. **ESC Protocol:** `MOT_PWM_TYPE` = 4 (DShot150), 5 (DShot300), or 6 (DShot600).
2. **Bi-Directional:** `SERVO_BLH_BDMASK` must cover your motor channels (e.g., 15 for first 4 motors).
3. **Verification:** Check the "Status" tab in Mission Planner. You should see `esc1_rpm`, `esc2_rpm`, etc., changing when you spin the motors.

CONFIGURATION

1. **Enable the Notch:**
   - `INS_HNTCH_ENABLE = 1`

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





2. **Set the Mode:**
    - `INS_HNTCH_MODE = 3` (ESC Telemetry). This tells ArduPilot to read the RPM from the DShot stream.
3. **Configure Frequency Mapping:**
    - `INS_HNTCH_REF = 1.0`. (Use the raw RPM / 60 as Hz).
    - `INS_HNTCH_FREQ = 80` (Hz). This is a safe fallback if RPM data is lost.
    - `INS_HNTCH_BW = 40` (Hz).
4. **Target Harmonics:**
    - `INS_HNTCH_HMNCS = 3` (Target 1st and 2nd harmonics). This is usually sufficient.

VERIFICATION (THE FFT GRAPH)

1. Set `INS_LOG_BAT_MASK = 1` (Log Pre-Filter Gyro).
2. Fly a short hover.
3. Download the log and perform an **FFT Analysis** in Mission Planner (Ctrl+F → FFT).
4. **Before:** You should see a giant spike at ~100-200Hz (the prop noise).
5. **After:** That spike should be completely gone in the "Post-Filter" graph.

For more details, see the ArduPilot Wiki: Harmonic Notch Filter.

IMU ARCHITECTURE & FUSION

EXECUTIVE SUMMARY

The Inertial Measurement Unit (IMU) is the heart of the flight controller. It consists of **Gyroscopes** (rotation rate) and **Accelerometers** (force/gravity). Modern controllers like the Cube Orange have **Three IMUs** for redundancy. ArduPilot runs multiple EKF instances in parallel (Lanes) and dynamically switches to the healthiest IMU to prevent crashes during sensor failure.

THEORY & CONCEPTS

1. THE GRAVITY VECTOR (FINDING DOWN)

How does the drone know which way is up?

- **Stationary:** The Accelerometer measures 1G pointing Down.
- **Moving:** The Accelerometer measures Gravity + Acceleration (Centrifugal force in a turn).
- **The Fusion:** The EKF uses the Gyro to track short-term rotation and the Accelerometer to correct long-term drift (Gravity Reference).

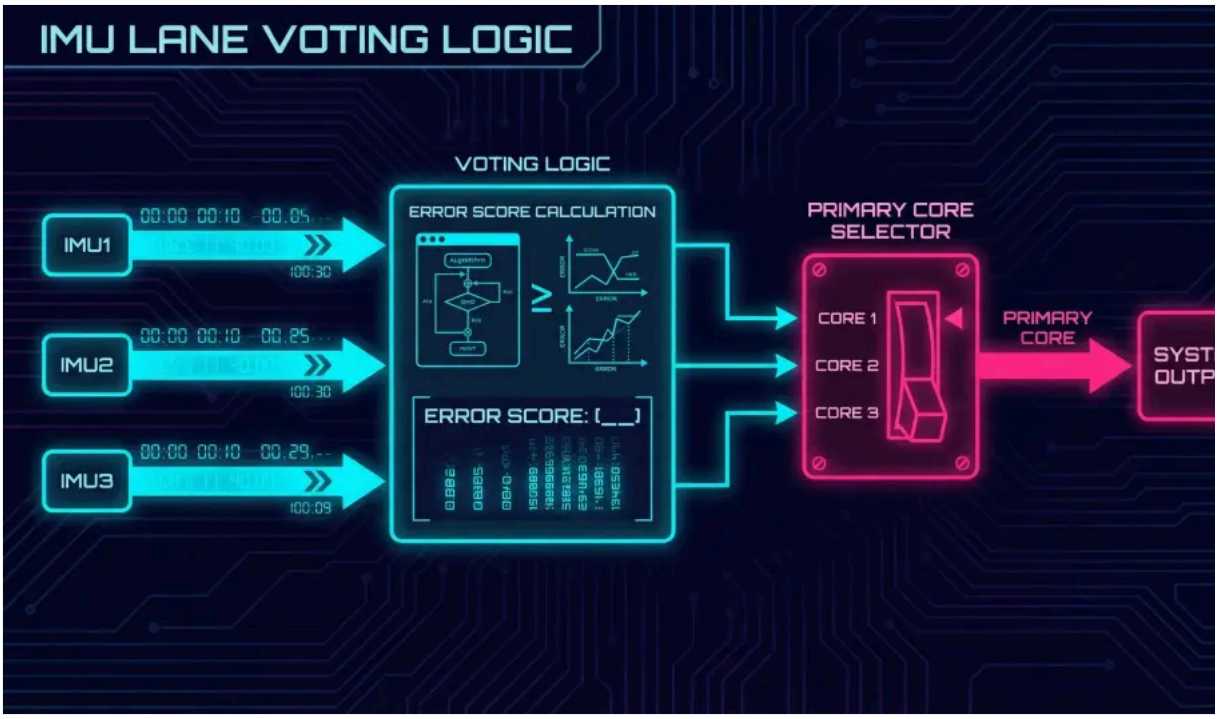
2. THE VOTING LOGIC (TRI-REDUNDANCY)

Why three IMUs?

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY







- **Two IMUs:** If IMU1 says "Up" and IMU2 says "Down", who is right? You don't know.
- **Three IMUs:** If IMU1 and IMU3 agree, but IMU2 disagrees, IMU2 is voted out.
- **Mechanism:** ArduPilot's EKF Selector monitors the "Innovation" (Error) of each lane. The lane with the lowest error relative to GPS/Baro is chosen as the **Primary**.

3. DCM VS EKF

- **DCM (Direction Cosine Matrix):** The old "Stabilize" algorithm. Simple, robust, but gets confused by long accelerations (e.g., spiraling plane).
- **EKF (Extended Kalman Filter):** The modern brain. It tracks "Bias" states, allowing it to realize that "My Gyro says we are spinning, but the Compass says we aren't → Therefore the Gyro is drifting."

ARCHITECTURE (THE ENGINEER'S VIEW)

1. HARDWARE ABSTRACTION (`AP_INERTIALSENSOR`)

This library manages the physical drivers (SPI/I2C).

- **Sampling:** Reads raw data at high speed (1kHz - 8kHz).
- **Filtering:** Applies Low-Pass Filters (LPF) and Harmonic Notch Filters to remove vibration noise.
- **Calibration:** Applies factory temp calibration and user offsets (`INS_ACCOFFS`, `INS_GYROOFFS`).

2. THE LANE SYSTEM (REDUNDANCY)

ArduPilot does not just "average" the three IMUs. It treats them as separate **Lanes**.

- **EKF Lane 1:** Uses IMU 1.
- **EKF Lane 2:** Uses IMU 2.
- **EKF Lane 3:** Uses IMU 3.
- *Benefit:* If IMU 1 goes crazy (mechanical failure or stuck bit), Lane 1 will diverge (high Innovation). Lane 2 and 3 will remain healthy. The EKF Selector will switch to Lane 2 instantly.

3. PRIMARY SELECTION

The **AHRS (Attitude Heading Reference System)** asks the EKF: "Which lane is best?"

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

- *Code Path:* `AP_AHRS::get_primary_IMU_index()` checks the EKF's `get_primary_core_index()`.
- *Result:* The flight control loops (Stabilize, Rate) always use the data from the "Winning" IMU.

4. SENSOR FUSION (GETTING ATTITUDE)

How do we know which way is Up?

- **Gyro:** Fast, precise, but drifts over time.
- **Accel:** Noisy, but the average vector points Down (Gravity).
- **Fusion:** The EKF combines them.
  - *High Frequency:* Trust the Gyro.
  - *Low Frequency:* Correct the Gyro drift using the Accel (Gravity vector).

KEY PARAMETERS

PARAMETER	DEFAULT	DESCRIPTION
`EK3_IMU_MASK`	3	Bitmask of which IMUs to use. 3 = IMU 1 and 2. 7 = All 3.
`INS_GYRO_FILTER`	20	(Hz) Low pass filter for gyro.
`INS_ACCEL_FILTER`	20	(Hz) Low pass filter for accel.
`INS_USE`	1	Enable/Disable specific IMUs.

SOURCE CODE REFERENCE

- **AHRS Update:** `AP_AHRS::update()`
- **IMU Driver:** `AP_InertialSensor::update()`

PRACTICAL GUIDE: THE FREEZER TEST (TCAL)

Factory IMU calibration is good, but if you fly in winter (-10C) and summer (+30C), your gyro bias will drift. ArduPilot can learn this curve.

STEP 1: ENABLE TCAL

- **Parameter:** `INS_TCAL_OPTIONS` (or similar depending on FW version).
- **Setting:** Set `TCAL_ENABLED = 1`.
- **Reboot.**

STEP 2: THE FREEZE

1. Put the flight controller (unpowered) in a Ziploc bag.
2. Put it in the freezer for 30 minutes. (Target: -10C).

STEP 3: THE COOK

1. Remove FC from freezer.
2. Immediately power it via USB.
3. **Do not move it.** Keep it perfectly still on a table.
4. As the board heats up (from self-heating or a hair dryer), ArduPilot records the gyro drift at every temperature step (-10, -5, 0, ... 60).
5. **Completion:** Watch the messages. "IMU1 temp cal done". "IMU2 temp cal done".
6. **Reboot:** The new calibration is saved to flash. Your horizon will now be level in any weather.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





CHAPTER 9: COMPANION COMPUTERS

EMBEDDED MAVLINK: INTEGRATING WITH ESP32 & ARDUINO

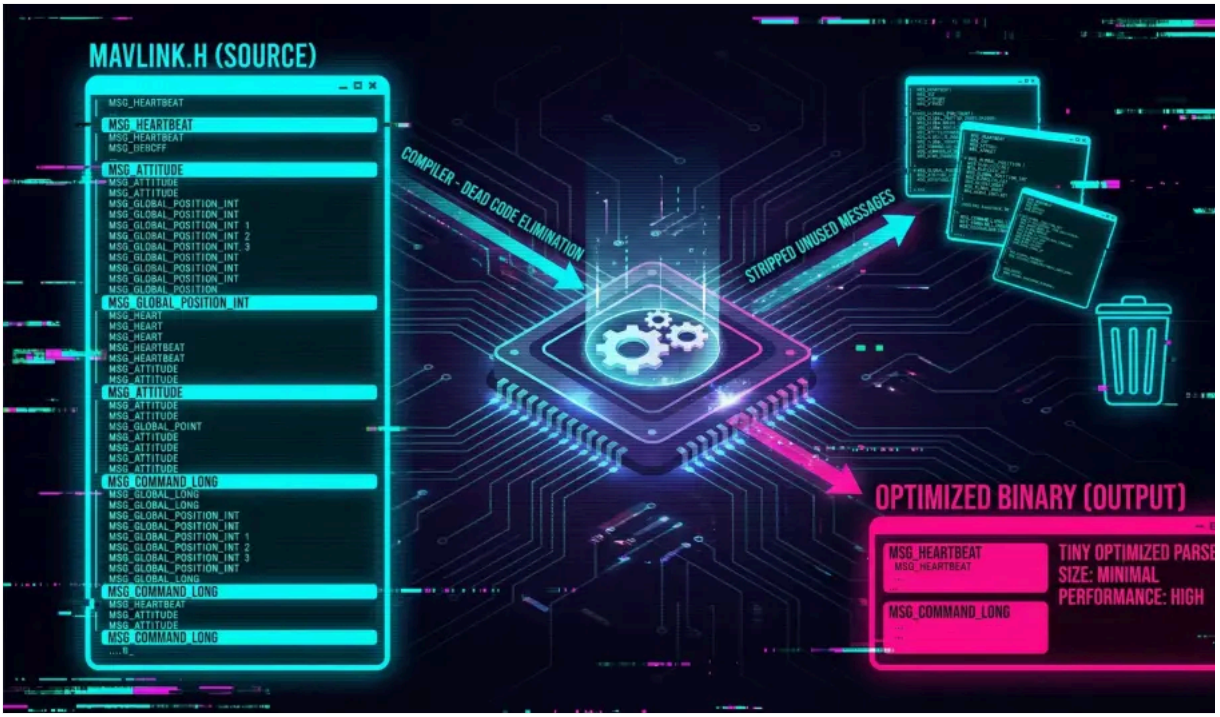
EXECUTIVE SUMMARY

For high-performance companion computers (like ESP32 or STM32), you cannot use Python (`pymavlink`). You must use the generated C/C++ headers. These headers provide a highly optimized, allocation-free way to serialize and parse MAVLink packets directly from the UART byte stream.

THEORY & CONCEPTS

1. HEADER-ONLY ARCHITECTURE

The MAVLink C library is "Header-Only."



- **What it means:** There is no `.cpp` file to compile. All the logic is in `.h` files using `static inline` functions.
- **Why:** This allows the compiler to optimize away unused messages, resulting in zero overhead for messages you don't use.
- **Integration:** You simply `#include "mavlink.h"` and point your compiler to the directory.

2. THE STATE MACHINE PARSER (`MAVLINK_PARSE_CHAR`)

MAVLink is a stream protocol. Bytes arrive one by one.

- **The Problem:** How do you find a packet in a continuous stream of bytes?
- **The Solution:** A state machine.
    1. Wait for Magic Byte (`0xFE` or `0xFD`).
    2. Read Length.
    3. Read Sequence.
    4. ...

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

5. Check CRC.

- **Result:** The parser returns `1` only when a valid, verified packet has been fully reassembled.

ARCHITECTURE (THE ENGINEER'S VIEW)

1. PACKING A MESSAGE

To send a message, you use a "Pack" function.

- **Function:** `mavlink_msg_heartbeat_pack([sysid](/field-manual/advanced-tuning/system-identification-mode.html), compid, &msg, type, autopilot, ...)`
- **Output:** It fills a `mavlink_message_t` struct.
- **Serialization:** You then call `mavlink_msg_to_send_buffer()` to convert that struct into a byte array for your UART driver.

2. PARSING A MESSAGE

To receive, you feed bytes into the parser one by one.

```
while (Serial.available()) {
    uint8_t c = Serial.read();
    if (mavlink_parse_char(MAVLINK_COMM_0, c, &msg, &status)) {
        // Packet Received!
        switch (msg.msgid) {
            case MAVLINK_MSG_ID_HEARTBEAT:
                handle_heartbeat(&msg);
                break;
        }
    }
}
```

KEY PARAMETERS FOR INTEGRATION

- **System ID:** Your device needs a unique ID (e.g., 51 for a Gimbal, 100 for a Camera). Don't use 1 (The Drone) or 255 (The GCS).
- **Component ID:** Defines *what* you are (e.g., `MAV_COMP_ID_ONBOARD_COMPUTER`).

SOURCE CODE REFERENCE

- **ArduPilot Wrapper:** `GCS_MAVLINK`
- **Core Parser:** `mavlink_helpers.h` (Generated)

PRACTICAL GUIDE: THE HEARTBEAT (HELLO WORLD)

If you don't send a heartbeat, ArduPilot won't talk to you. Here is the minimum code to establish a link.

1. THE SETUP

```
#include "mavlink/common/mavlink.h"

// My Identity
uint8_t my_sysid = 1; // Same vehicle
uint8_t my_compid = 191; // MAV_COMP_ID_ONBOARD_COMPUTER
```

2. THE LOOP (1HZ)

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





```
void send_heartbeat() {
    mavlink_message_t msg;
    uint8_t buf[MAVLINK_MAX_PACKET_LEN];

    // Pack the message
    // Type: Onboard Controller, Autopilot: Invalid (I'm not a flight controller)
    mavlink_msg_heartbeat_pack(my_sysid, my_compid, &msg,
                               MAV_TYPE_ONBOARD_CONTROLLER,
                               MAV_AUTOPILOT_INVALID,
                               0, 0, 0);

    // Serialize
    uint16_t len = mavlink_msg_to_send_buffer(buf, &msg);

    // Send via UART
    Serial.write(buf, len);
}
```

3. THE RESULT

Open Mission Planner → Ctrl+F → MAVLink Inspector.
You should see a new Component (191) appearing in the list, updating every second.

LUA SCRIPTING: THE ONBOARD COMPANION

EXECUTIVE SUMMARY

Historically, if you wanted custom logic (e.g., "If battery < 10\% AND altitude > 50m, turn on LEDs"), you needed a Companion Computer (Raspberry Pi) communicating via MAVLink. Now, you can run this logic *inside* the Flight Controller using **Lua Scripting**. This eliminates the weight, power, and complexity of an external computer for simple tasks.

ARCHITECTURE (THE ENGINEER'S VIEW)

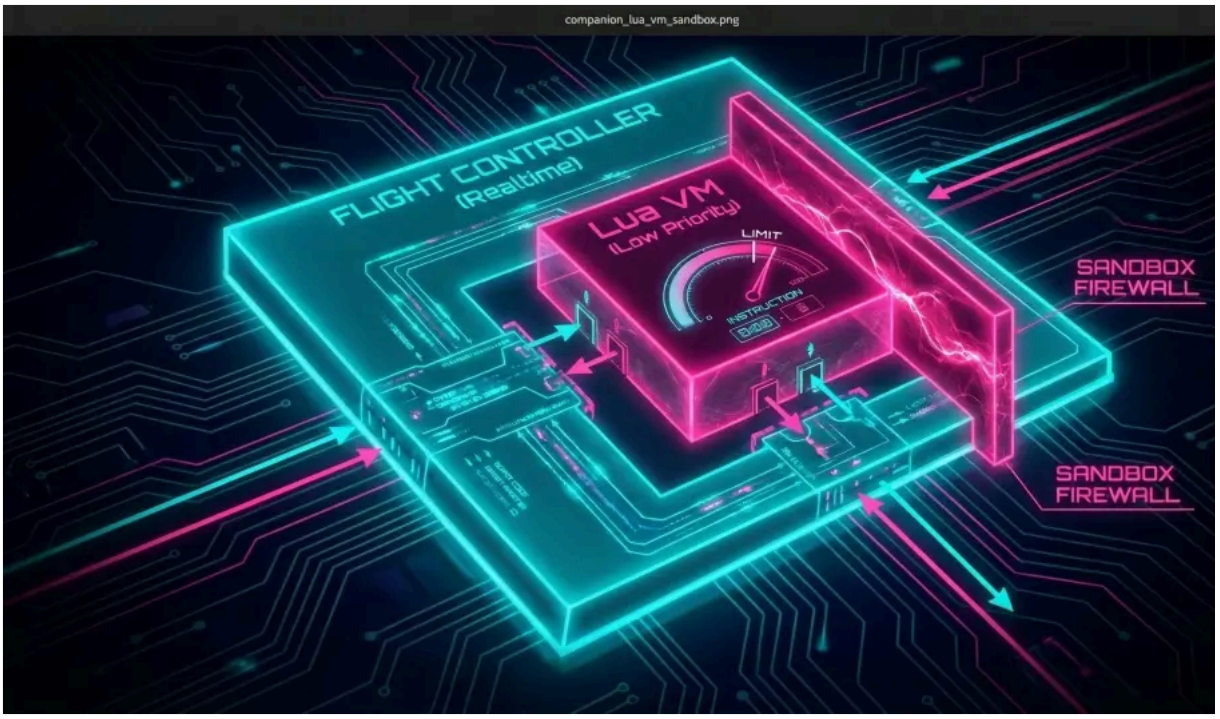
1. THE VIRTUAL MACHINE (LUA 5.3)

ArduPilot embeds a lightweight Lua 5.3 VM.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY







- **Isolation:** The VM runs in a separate thread with a low priority.
- **Safety:** The script is allocated a strict **Instruction Count Quota** (default 10,000 ops). If it runs too long, the scheduler pauses it. This ensures a buggy infinite loop in your script *cannot* crash the flight controller.
- **Memory:** The script gets a fixed heap (`SCR_HEAP_SIZE` , e.g., 40KB). If you leak memory, the script crashes, but the drone keeps flying.

2. THE BINDINGS (API)

The C++ code exposes specific functions to Lua via an automated binding generator.

- **Singletons:** Global objects like `vehicle` , `ahrs` , `gcs` .
  - `a = ahrs:get_roll()`
  - `gcs:send_text(6, "Hello World")`
- **Methods:** `param:set('RTL_ALT', 5000)`

3. USE CASES VS. COMPANION COMPUTER

- **Use Lua When:**
  - You need < 10ms latency (e.g., custom motor mixing).
  - You are interacting with hardware pins (LEDs, Relays).
  - The logic is simple state-machine stuff ("Smart Failsafe").
- **Use Companion Computer When:**
  - You need Computer Vision (OpenCV).
  - You need Internet Access (4G/LTE).
  - You need massive storage (Data logging).

KEY PARAMETERS

PARAMETER	DEFAULT	DESCRIPTION
`SCR_ENABLE`	0	Enable the VM. Requires reboot.
`SCR_HEAP_SIZE`	40000	(Bytes) RAM allocated to the script. Increase for complex scripts.
`SCR_VM_I_COUNT`	10000	Max instructions per loop.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

SOURCE CODE REFERENCE

- **Thread Runner:** `AP_Scripting::thread()`
- **Bindings:** `libraries/AP_Scripting/generator/description/bindings.desc`

HOW TO ENABLE LUA SCRIPTING

1. **Enable the Module:**
    - Connect to your flight controller via Mission Planner or QGroundControl.
    - Set the parameter `SCR_ENABLE` to **1**.
    - **Reboot the flight controller.** This is required to allocate the Lua VM memory.
2. **Configure Memory:**
    - Check `SCR_HEAP_SIZE`. The default (often 40KB) is enough for simple logic but too small for complex scripts.
    - Recommended: Increase to **100000** (100KB) or more if your board supports it (H7 boards have plenty of RAM).
3. **Install the Script:**
    - Remove the SD card from the flight controller or use MAVFTP.
    - Navigate to the `APM/scripts/` directory on the SD card. (Create it if it doesn't exist).
    - Copy your `.lua` file into this folder.
4. **Verify Execution:**
    - Reboot the flight controller.
    - Check the **Messages** tab in your GCS. You should see a message like "Lua: MyScript.lua start".
    - If you see "Lua: OOM", increase `SCR_HEAP_SIZE`.

PRACTICAL GUIDE: RUNNING YOUR FIRST SCRIPT

Use this template to create a script that listens to an RC switch and prints a message.

STEP 1: THE CODE (`HELLO.LUA`)

```
-- Define the polling rate (ms)
local UPDATE_INTERVAL_MS = 1000

-- Main Loop Function
function update()
  -- Read the state of RC Channel configured as "Scripting 1" (Option 300)
  -- rc:find_channel_for_option(300) finds the channel mapped to SCRIPTING_1
  local switch_ch = rc:find_channel_for_option(300)

  if switch_ch then
      local pwm = switch_ch:get_aux_switch_pos()
      -- 0=Low, 1=Mid, 2=High
      if pwm == 2 then
          gcs:send_text(6, "Lua: Switch is HIGH!")
      end
  else
      gcs:send_text(4, "Lua: RC Switch not configured!")
  end

  -- Schedule next run
  return update, UPDATE_INTERVAL_MS
end

-- Start the loop
gcs:send_text(6, "Lua: Hello World Script Started")
return update()
```


Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





STEP 2: SETUP

1. Save the code as `hello.lua` in `APM/scripts/`.
2. Set `RC7_OPTION = 300` (Scripting 1). This maps your transmitter's Channel 7 switch to the script.
3. Reboot.
4. Toggle your Channel 7 switch to High. You should see "Lua: Switch is HIGH!" in the GCS messages every second.

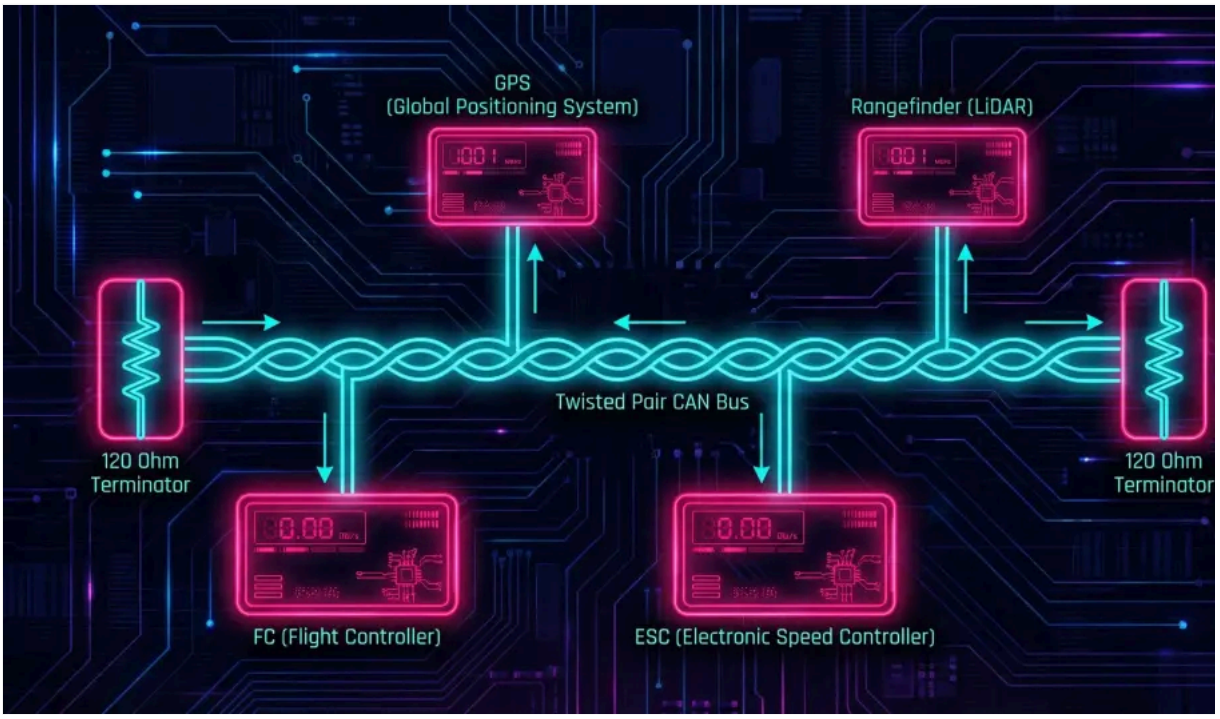
THE LIFECYCLE

1. **Boot:** ArduPilot initializes drivers.
2. **Script Load:** The Lua VM starts and parses your file from top to bottom.
    - *Tip:* Do your expensive setup (allocating arrays) here, outside the `update` function.
3. **Run:** The line `return update()` tells ArduPilot to schedule the `update` function.
4. **Loop:** Every 1000ms (as requested), ArduPilot wakes up the script, runs `update()`, and puts it back to sleep.

DRONECAN: THE MODERN BUS

EXECUTIVE SUMMARY

I2C and UART are legacy protocols. They are point-to-point, prone to noise, and have no standard configuration interface. **DroneCAN** (formerly UAVCAN v0) is a robust, differential-signaling bus. It allows you to daisy-chain GPS, Compass, ESCs, and Rangefinders on a single twisted pair of wires. It features "Plug and Play" node allocation and centralized configuration.



THEORY & CONCEPTS

1. PUBLISH / SUBSCRIBE ARCHITECTURE

MAVLink is often Request/Response. DroneCAN is **Pub/Sub**.

- **Publisher:** A GPS node broadcasts `uavcan.equipment.[gnss](/field-manual/sensor-architecture/gps-integration.html).Fix` messages on the bus. It doesn't know or care who is listening.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

- **Subscriber:** The Flight Controller subscribes to `uavcan.equipment.gnss.Fix`.
- **Benefit:** Multiple devices can listen to the same sensor (e.g., a Companion Computer *and* the Autopilot) without fighting for the serial port.

2. DYNAMIC NODE ALLOCATION (DNA)

On a CAN bus, every device needs a unique 7-bit Address (Node ID).

- **The Problem:** In manual systems (CANopen), you have to set DIP switches on every device to avoid conflicts.
- **The DNA Solution:**
  1. A new device boots up with Node ID 0 (Anonymous).
  2. It broadcasts a "Hello" message with a 16-byte Unique ID (UUID).
  3. ArduPilot (The DNA Server) hears this. It looks up the UUID in its database.
  4. ArduPilot assigns a permanent Node ID (e.g., 55) to that UUID.
  5. The device saves "55" to flash and uses it forever.

3. SLCAN (TUNNELING)

How do you configure a CAN GPS if it doesn't have a USB port?

- **The Tunnel:** ArduPilot acts as a bridge. It wraps raw CAN frames inside MAVLink packets (`CAN_FRAME`).
- **The Tool:** The "DroneCAN GUI Tool" on your PC sends these MAVLink packets. ArduPilot unwraps them and puts them on the CAN bus. The response follows the reverse path.
- *Result:* You can update firmware and change settings on a GPS module while it is installed in the drone.

KEY PARAMETERS

PARAMETER	DEFAULT	DESCRIPTION
`CAN_P1_DRIVER`	0	1=First Driver. Enables the CAN interface.
`CAN_D1_PROTOCOL`	1	1=DroneCAN.
`GPS_TYPE`	9	9=UAVCAN. Tells ArduPilot to look for GPS on the CAN bus.

SOURCE CODE REFERENCE

- **DNA Server:** `AP_DroneCAN_DNA_Server::handle_allocation()`

PRACTICAL GUIDE: CONFIGURING CAN NODES (SLCAN)

You bought a Here3 GPS. How do you change its LED color? You can't plug it into USB. You must tunnel through the Autopilot.

STEP 1: ENABLE THE DRIVER

1. Set `CAN_P1_DRIVER = 1`.
2. Set `CAN_D1_PROTOCOL = 1` (DroneCAN).
3. Reboot.

STEP 2: ESTABLISH THE TUNNEL

1. Connect Mission Planner via USB.
2. Press **Ctrl+U** (UAVCAN Screen).
3. Click **"SLCan Mode CAN1"**.
   - *What happens:* Mission Planner sends a command to ArduPilot to stop acting like a Flight Controller and start acting like a CAN Adapter. The MAVLink stream will die. This is normal.
4. You should see a list of nodes appear (e.g., "Here3 GPS", "ZTW ESC").

STEP 3: CONFIGURE THE NODE

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





1. Click the "Parameters" button next to the GPS node.
2. A new window appears showing the *GPS's internal parameters*.
3. Change `LED_COLOR` or whatever you need.
4. **Write Params**.
5. **Critical:** Reboot the flight controller to exit SLCAN mode and return to normal flight.

MSP & DJI OSD INTEGRATION

EXECUTIVE SUMMARY

For years, the FPV world was divided: ArduPilot users used MAVLink, and Betaflight users used MSP (MultiWii Serial Protocol). With the rise of DJI FPV, Walksnail, and HDZero, these digital video systems primarily support MSP. To use them, ArduPilot must **emulate** a Betaflight flight controller, speaking the MSP language to trick the goggles into displaying telemetry.

THEORY & CONCEPTS

1. THE "IMPULSOR" DECEPTION

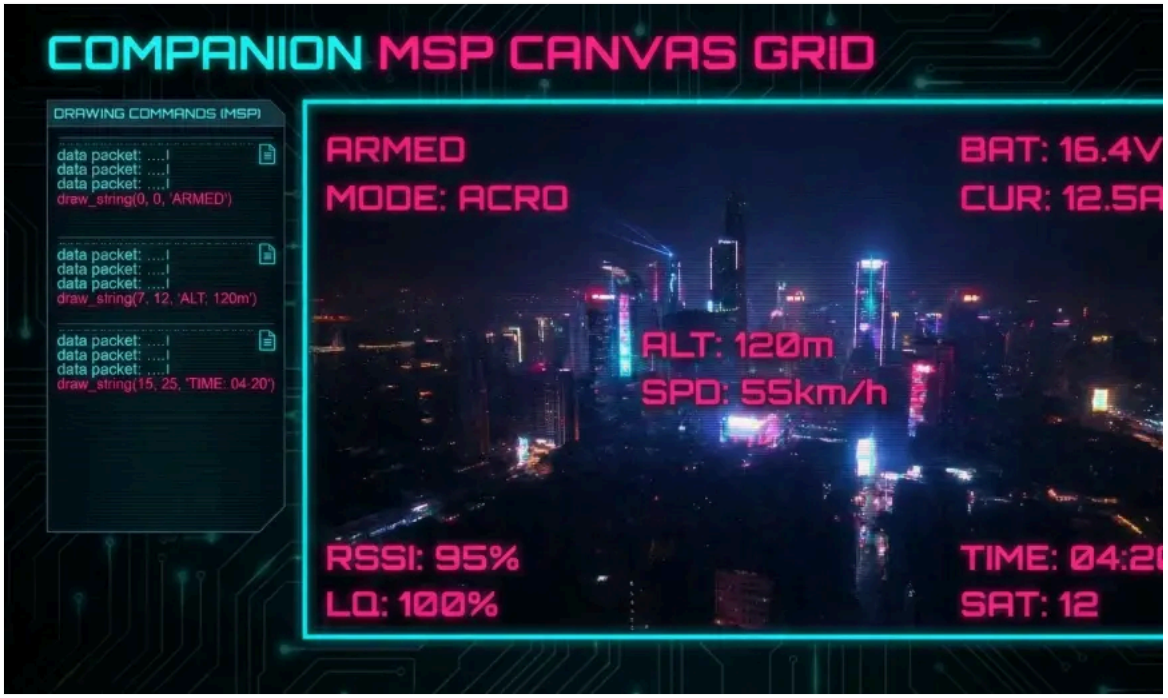
Digital OSD units (like the DJI Air Unit) query the flight controller: "Who are you?"

- **The Answer:** ArduPilot replies "I am Betaflight" (`BTFL`).
- **Why?** If it replied "ArduPilot," the DJI unit would ignore it.
- **The Translation:** ArduPilot takes its internal state (AHRS, Battery) and packages it into MSP messages (e.g., `MSP_ATTITUDE` , `MSP_BATTERY`).

2. CANVAS MODE (DISPLAYPORT)

Older "Native" DJI OSDs only supported a fixed list of elements (Voltage, Timer).

- **The Solution:** Canvas Mode (MSP DisplayPort).



Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

- **Mechanism:** Instead of sending values ("Voltage = 12.6"), ArduPilot sends **Drawing Commands** ("Draw '12.6V' at Row 10, Col 4").
- **Benefit:** This allows ArduPilot to render its entire OSD menu system, radar screens, and custom fonts on the HD goggles, bypassing the goggles' internal limitations.

ARCHITECTURE (THE ENGINEER'S VIEW)

1. THE BACKEND (`AP_MSP`)

This library handles the serial protocol.

- **Weighted Fair Queuing:** It prioritizes critical attitude updates over slow battery updates to ensure the artificial horizon is smooth.
- *Code Path:* `msp_process_out_fc_variant()` is where the "BTFL" handshake happens.

2. THE RENDERER (`AP_OSD`)

- **Abstraction:** `AP_OSD` draws to a virtual screen buffer (30×16 chars).
- **Driver:** The `AP_OSD_MSP_DisplayPort` backend reads this buffer and generates the MSP commands (`MSP_DISPLAYPORT_WRITE_STRING`) to mirror it on the remote screen.

KEY PARAMETERS

PARAMETER	DEFAULT	DESCRIPTION
`SERIALx_PROTOCOL`	33	33=DJI FPV (Basic), 42=DisplayPort (Canvas).
`OSD_TYPE`	1	1=Analog/Native, 5=MSP DisplayPort.
`MSP_OPTIONS`	0	Bit 0: Enable telemetry mode.

SOURCE CODE REFERENCE

- **Emulation Logic:** `AP_MSP_Telem_DJI::msp_process_out_fc_variant()`
- **Canvas Writer:** `AP_MSP_Telem_Backend::msp_displayport_write_string()`

MODERN ALTERNATIVE: DIRECT AR

While MSP emulation is excellent for traditional FPV goggles, it is limited by the VTX bandwidth and the 30×16 character grid.

MAVLink HUD takes a different approach:

- **High Resolution:** Renders a 1080p/4K interface on your Android device.
- **No VTX Required:** Telemetry comes over the ELRS radio link.
- **Augmented Reality:** By connecting USB-C AR glasses, you get a transparent "Glass Cockpit" overlay without blocking your line of sight.

For a list of compatible displays, see our Supported Hardware Guide.

PRACTICAL GUIDE: DJI & WALKSNAIL SETUP

The days of "Custom OSD" are over. Use DisplayPort.

STEP 1: PHYSICAL CONNECTION

Connect the Air Unit (DJI/Walksnail) to a spare UART (e.g., `SERIAL2`).

- TX → RX
- RX → TX
- GND → GND

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





STEP 2: ARDUPILOT CONFIG

1. **Protocol:** Set `SERIAL2_PROTOCOL = 42` (DisplayPort).
2. **Baud:** Set `SERIAL2_BAUD = 115` (115200).
3. **OSD Type:** Set `OSD_TYPE = 5` (MSP_DisplayPort).
4. **Reboot.**

STEP 3: GOGGLE CONFIG

1. **DJI (V2/2):** Ensure "Canvas Mode" (or "Custom OSD") is enabled in the goggles menu.
2. **Walksnail:** It works out of the box.
3. **Verification:** You should see the ArduPilot initialization text ("ArduCopter V4.x.x") on your HD feed. If you see "Waiting for OSD...", check your TX/RX swap.

AP_PERIPH: BUILDING CUSTOM NODES

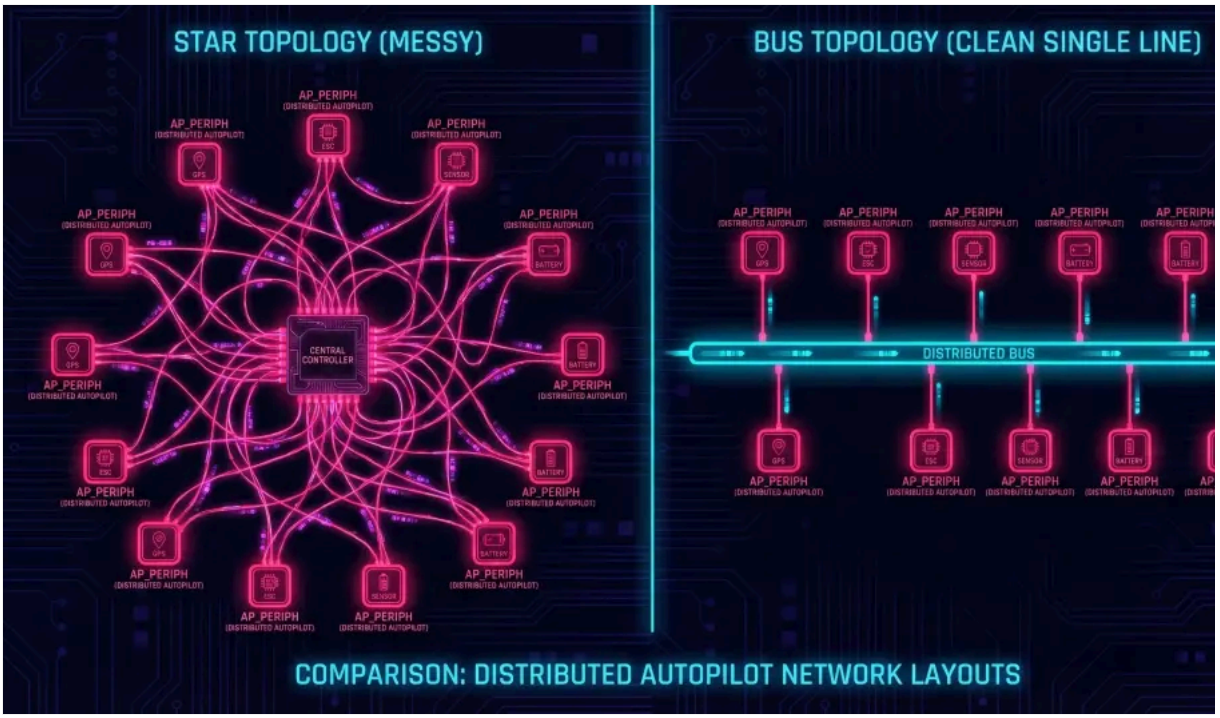
EXECUTIVE SUMMARY

AP_Periph is a stripped-down version of ArduPilot designed to run on peripheral microcontrollers (like STM32F3 or F4) instead of a Flight Controller. It allows you to turn almost any sensor (GPS, Compass, Airspeed, Rangefinder) into a **DroneCAN Node**.

THEORY & CONCEPTS

1. THE DISTRIBUTED AUTOPILOT

Traditional drones are a "Star Topology" (Everything plugs into the center).
AP_Periph enables a "Bus Topology".



- **The GPS Node:** Contains a GPS module, a Compass, an RGB LED, and a Buzzer. It runs `AP_Periph`. It connects to the Flight Controller with 4 wires (VCC, GND, CAN_H, CAN_L).
- **The Benefit:** Noise isolation. The sensitive compass is far away from the high-current ESCs. The wiring harness is simple.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

2. DRIVER REUSE

AP_Periph does not use "Special" drivers. It links against the *standard* ArduPilot libraries.

- **Implication:** If a sensor works in ArduCopter, it works in AP_Periph.
- **Maintenance:** You don't need to write custom CAN code for a new sensor. You just enable the existing driver in the `hwdef.dat` and compile.

ARCHITECTURE (THE ENGINEER'S VIEW)

1. THE MAIN LOOP (`AP_PERIPH_FW::UPDATE`)

Unlike ArduCopter (400Hz), AP_Periph runs a simpler loop.

1. **Sensor Read:** It polls all enabled libraries (`AP_GPS`, `AP_Compass`).
2. **CAN Transmit:** It packages the sensor data into DroneCAN packets (e.g., `uavcan.equipment.[gnss](/field-manual/sensor-architecture/gps-integration.html).Fix`).
3. **CAN Receive:** It listens for configuration parameters from the GCS.

2. THE BOOTLOADER

AP_Periph devices use a special bootloader that supports **CAN Firmware Updates**.

- **Mechanism:** The Flight Controller acts as a "Bridge." You can flash the GPS firmware *through* the Flight Controller USB port using the DroneCAN GUI Tool.

KEY PARAMETERS

These are set on the *Peripheral* itself (using the GUI Tool), not the Flight Controller.

- `CAN_NODE_ID`: 0 (Automatic DNA) is standard.
- `GPS_TYPE`: Selection of the sensor driver.

SOURCE CODE REFERENCE

- **Main Loop:** `AP_Periph_FW::update()`
- **CAN Handler:** `AP_Periph_FW::can_update()`

VIDEO & OSD ARCHITECTURE

EXECUTIVE SUMMARY

The On-Screen Display (OSD) overlays flight data onto the FPV video feed. ArduPilot supports both legacy Analog OSDs (MAX7456 chips) and modern Digital OSDs (DJI, Walksnail, HDZero). It uses a unified "Panel" architecture, where you arrange items (Altitude, Battery, Horizon) on a grid, and the backend translates that grid to the specific hardware protocol.

THEORY & CONCEPTS

1. THE CHARACTER GRID

Unlike a modern smartphone screen which draws pixels, OSDs are character-based.

- **The Grid:** Typically 30 columns x 16 rows.
- **The Font:** Icons (like a battery symbol) are just special characters in a custom font file (`font.bin`).
- **Limitations:** You cannot draw a smooth line or a circle. You can only place a pre-drawn character in a grid cell.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





2. ANALOG VIDEO (V-BLANK)

Analog OSD chips (MAX7456) overlay white pixels onto the NTSC/PAL signal.

- **SPI Bus:** The flight controller sends the character map to the OSD chip over SPI.
- **Timing:** The chip must sync with the video signal's Vertical Blanking Interval (V-Blank) to update the screen without tearing.

ARCHITECTURE (THE ENGINEER'S VIEW)

1. THE OSD THREAD

Rendering text is slow. To prevent it from blocking the main 400Hz flight loop, `AP_OSD` runs in its own thread.

- **Rate:** 10Hz (approx 100ms per frame).
- **Logic:**
  1. Calculates values (e.g., "Alt: 100m").
  2. Writes string to the backend buffer.
  3. Calls `flush()` to send the buffer to the hardware.
     - *Code Path:* `AP_OSD::osd_thread()`.

2. BACKEND ABSTRACTION

- **Analog:** `AP_OSD_MAX7456`. Writes to SPI. Optimizes bandwidth by only updating "Dirty" characters (chars that changed since last frame).
- **Digital:** `AP_OSD_MSP_DisplayPort`. Wraps the grid commands into MAVLink/MSP serial packets for the DJI/Walksnail unit to render locally.

KEY PARAMETERS

PARAMETER	DEFAULT	DESCRIPTION
`OSD_TYPE`	1	1=MAX7456 (Analog), 5=MSP_DisplayPort (Digital).
`OSD_CHAN`	0	The specific screen to display (Screen 1, 2, 3, 4).

SOURCE CODE REFERENCE

- **Thread Runner:** `AP_OSD::osd_thread()`

CLIENT-SIDE RENDERING (MAVLINK HUD)

The architecture described above relies on the Flight Controller (STM32) to generate the video overlay. This consumes CPU cycles and limits graphical fidelity.

MAVLink HUD inverts this model:

1. **Raw Data:** The Flight Controller sends raw telemetry values (not video pixels) via MAVLink.
2. **GPU Rendering:** The Android device uses its powerful GPU to render smooth, high-fidelity instruments.
3. **Display:** The result is output via USB-C to Supported AR Glasses, creating an infinite-resolution overlay.

- **Thread Runner:** `AP_OSD::osd_thread()`

COMPANION COMPUTER FAILSAFES

EXECUTIVE SUMMARY

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





When a Companion Computer (Raspberry Pi/Jetson) takes control of a drone, it becomes a single point of failure. If your Python script crashes or the USB cable vibrates loose, the drone must not fly away. ArduPilot provides a multi-layered "Deadman Switch" system to detect these failures and safely recover the vehicle.

THEORY & CONCEPTS

1. THE HEARTBEAT (LINK HEALTH)

The most basic check is: "Is the computer still talking?"

- **Mechanism:** The Companion Computer must send a `HEARTBEAT` message at least once every second.
- **Failure:** If the heartbeat stops, ArduPilot assumes the link is broken. It triggers the **GCS Failsafe**.

2. THE COMMAND STREAM (LOGIC HEALTH)

Your computer might be alive (sending heartbeats) but your script might be frozen (not sending commands).

- **Mechanism:** Guided Mode expects a constant stream of `SET_POSITION_TARGET` messages (e.g., 10Hz).
- **Failure:** If the stream stops for `GUIDED_TIMEOUT` seconds, ArduPilot assumes the logic has crashed. It stops the drone instantly.

ARCHITECTURE (THE ENGINEER'S VIEW)

1. GCS FAILSAFE LOGIC

- **Monitor:** `GCS_MAVLINK::handle_heartbeat()` updates `last_heartbeat_time`.
- **Trigger:** `Copter::failsafe_gcs_check()` runs at 1Hz. If `now - last_heartbeat > FS_GCS_TIMEOUT`, it triggers.
- **Action:** Depending on `FS_GCS_ENABLE`, it will:
  - 1: RTL.
  - 2: SmartRTL or RTL.
  - 3: SmartRTL or Land.
  - 4: Land.

2. GUIDED MODE TIMEOUT

- **Monitor:** `ModeGuided` tracks the timestamp of the last valid velocity/attitude command.
- **Action:** If the timeout expires, the vehicle enters a **"Brake"** state (zero velocity target). It does *not* land or RTL; it just hovers, waiting for the computer to come back.

KEY PARAMETERS

PARAMETER	DEFAULT	DESCRIPTION
`FS_GCS_ENABLE`	1	0=Disabled, 1=RTL, 2=SmartRTL.
`GUIDED_TIMEOUT`	2.4	(seconds) Timeout for velocity/attitude stream.
`SYSID_MYGCS`	255	The System ID of the computer to track. If 0, it tracks the first GCS that talks to it.

SOURCE CODE REFERENCE

- **Heartbeat Logic:** `GCS_MAVLINK::handle_heartbeat()`

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





CHAPTER 10: EXPRESSLRS

ELRS FOUNDATIONS: LORA & CSS PHYSICS

EXECUTIVE SUMMARY

ExpressLRS (ELRS) represents a paradigm shift from traditional FSK (Frequency Shift Keying) radios. It utilizes **LoRa (Long Range)** modulation, a technology originally designed for low-power IoT devices. By trading bitrate for sensitivity, ELRS achieves link budgets that were previously impossible for hobbyist gear, allowing for reliable control at 100km+ ranges or deep penetration through concrete.

THEORY & CONCEPTS

1. CHIRP SPREAD SPECTRUM (CSS)

Traditional radios transmit on a single frequency (Carrier). LoRa transmits a **Chirp**—a signal that sweeps across a bandwidth (e.g., 500kHz) over time.

- **The Sweep:** The chirp starts at a low frequency and sweeps to high (Up-Chirp) or vice-versa.
- **The Symbol:** Data is encoded by the *starting point* of the sweep.
- **Interference Immunity:** Narrowband noise (like Wifi) only corrupts a tiny slice of the chirp. The receiver can reconstruct the whole symbol from the remaining clean chirp. This allows LoRa to decode signals that are **below the noise floor**.

2. SPREADING FACTOR (SF)

The "Spreading Factor" determines the duration of the chirp.

- **SF5-7 (Fast):** Short chirps. High data rate (500Hz). Lower range (-105 dBm).
- **SF9-12 (Slow):** Long chirps. Low data rate (50Hz). Extreme range (-123 dBm).
- *The Trade-off:* Every step up in SF doubles the time-on-air (latency) but adds ~3dB of sensitivity (double the range).

3. ORTHOGONALITY

Different Spreading Factors are orthogonal. A receiver listening for SF7 (500Hz) is completely blind to SF12 (50Hz) signals, even on the same frequency. This allows multiple ELRS systems to coexist or for the system to switch modes dynamically.

ARCHITECTURE (THE ENGINEER'S VIEW)

1. THE HARDWARE LAYER

ELRS runs on Semtech transceivers.

- **SX1280 (2.4GHz):** Optimized for racing. High bandwidth, lower range.
- **SX1276 (900MHz):** Optimized for penetration. Low bandwidth, extreme range.

2. THE PACKET STRUCTURE

ELRS strips away all legacy bloat.

- **No Headers:** Packet length is fixed for a given rate.
- **Seed-CRC:** The Binding Phrase acts as a seed for the CRC. If you don't have the phrase, the CRC fails, and the packet is discarded. This eliminates the need for a "Receiver ID" in the packet header, saving bytes.

KEY PARAMETERS

- **Packet Rate:** 50Hz to 1000Hz.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- **Telemetry Ratio:** 1:2 to 1:128.
- **Switch Mode:** Hybrid vs Wide.

SOURCE CODE REFERENCE

- **Project:** ExpressLRS GitHub

PACKET RATES, LATENCY & RANGE

EXECUTIVE SUMMARY

One size does not fit all. ExpressLRS forces you to choose your **Packet Rate** (Update Frequency). This choice defines the fundamental capabilities of your link. A 1000Hz link feels instantaneous but is fragile behind trees. A 50Hz link feels sluggish but can punch through a concrete building.

THEORY & CONCEPTS

1. THE LATENCY TRIANGLE

You can pick two:

1. **Speed (Low Latency):** Good for racing. Requires fast packets.
2. **Range (Sensitivity):** Good for long-range. Requires slow packets (high Spreading Factor).
3. **Reliability:** Good for noise. Requires redundancy (DVDA/Gemini).

2. THE RATES

- **1000Hz / F1000:** The "God Mode" of racing.
  - *Latency:* ~1.5ms.
  - *Physics:* Uses FLRC (Fast Long Range Communication), not LoRa. It behaves like standard FSK. Sharp cliff effect at range limit.
- **500Hz:** The standard for Freestyle.
  - *Latency:* ~2.5ms.
  - *Physics:* Lowest LoRa setting. Good balance.
- **250Hz:** The "Sweet Spot" for MAVLink.
  - *Latency:* ~6ms.
  - *Reasoning:* It provides enough bandwidth for Telemetry (1:2 ratio) while maintaining excellent range.
- **50Hz:** The "Long Range King."
  - *Latency:* ~22ms.
  - *Physics:* High Spreading Factor. Can decode signals 18dB lower than 500Hz. This triples your range.

3. LINK QUALITY (LQ)

In legacy systems, RSSI (Signal Strength) was king. In ELRS, **LQ** is king.

- **LQ:** The percentage of packets successfully received.
- **The Cliff:** LoRa works perfectly until it doesn't. You can have perfect control at -120dBm (basically zero signal) as long as LQ is high.
- **The Warning:** If LQ drops below 70\%, you are losing data.

ARCHITECTURE (THE ENGINEER'S VIEW)

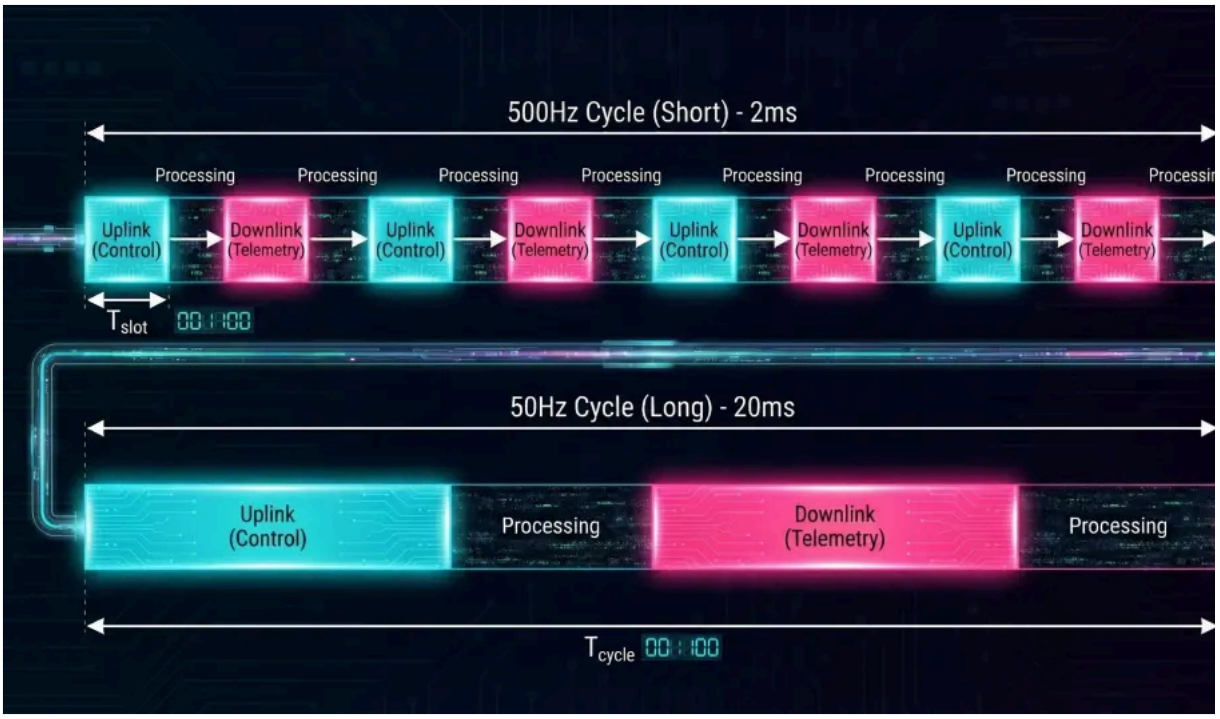
1. PACKET CYCLE

At 500Hz, a cycle is 2ms.

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY







- **Uplink (TX → RX):** Control Data.
- **Downlink (RX → TX):** Telemetry (if it's the telemetry slot).
- **Processing:** The Flight Controller must run its loop fast enough (400Hz/1kHz) to consume this data without aliasing.

2. MAVLINK IMPLICATIONS

For MAVLink HUD usage:

- **50Hz:** Too slow. 1:2 ratio = 25Hz telemetry. Max bandwidth ~200 bytes/sec. Not enough for smooth HUD.
- **250Hz:** Recommended. 1:2 ratio = 125Hz telemetry. Max bandwidth ~1000 bytes/sec. Smooth HUD.

KEY PARAMETERS

- `Packet Rate` (Lua Script): The master setting.
- `Telem Ratio` (Lua Script): How often to send data back.

SOURCE CODE REFERENCE

- **Documentation:** ELRS Modes

PRACTICAL GUIDE: UNLOCKING SMOOTH TELEMETRY

Many pilots connect ELRS and see MAVLink data at a crawl (0.5Hz - 1Hz). This is not a bug; it is ArduPilot reacting to your link health.

THE MECHANISM

ArduPilot monitors the "RF Mode" reported by your ELRS receiver via the CRSF protocol.

- **Low Speed (50Hz - 150Hz):** ArduPilot enters "Bandwidth Conservation Mode." It strictly limits telemetry to essential stats (Battery, GPS) at very low rates to prioritize RC control packets and prevent link congestion.
- **High Speed (250Hz+):** ArduPilot unlocks the "High Speed Profile," allowing much faster telemetry updates suitable for HUDs and real-time monitoring.

THE FIX

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

1. **On your Transmitter (Lua Script):** Set Packet Rate to **333Hz Full** or **500Hz**.
2. **Telemetry Ratio:** Set to **Std (1:64)** or **1:2**.
    - **Counter-Intuitive:** Even if you set a high ratio (1:2) at 50Hz, the *total* bandwidth is still too low for ArduPilot's scheduler to feel comfortable sending bulk data. You *must* increase the Packet Rate to unlock the scheduler.

VERIFICATION

Check the `LINK_STATISTICS` or `RADIO STATUS` message. If `rx_errors` is low but the stream rate is high, ArduPilot has successfully engaged the high-speed profile.

For more details on ArduPilot's CRSF integration, see the ArduPilot Wiki: CRSF Telemetry.

MAVLINK OVER ELRS: THE 'AIRPORT' BRIDGE

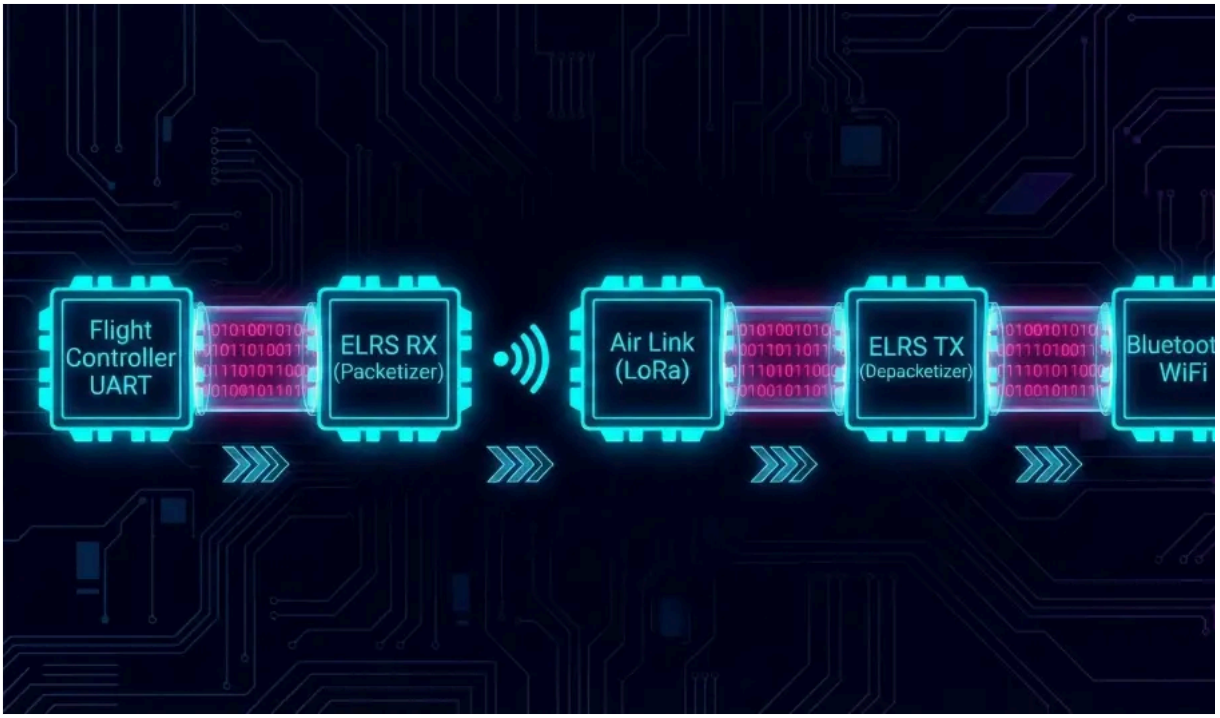
EXECUTIVE SUMMARY

Traditionally, Long Range pilots needed two radios: Crossfire for Control, and SiK Radio (900MHz) for MAVLink Telemetry. ELRS "Airport" combines them. By sacrificing some control bandwidth, you can turn the ELRS link into a transparent bidirectional serial bridge, allowing full Mission Planner / QGC connection without extra hardware.

THEORY & CONCEPTS

1. THE "AIRPORT" ARCHITECTURE

ELRS is usually a "Control" protocol (Channels). Airport turns it into a "Data" protocol (Bytes).



- **Packet Switching:** The ELRS firmware repacks the raw serial bytes from the Receiver's UART into LoRa packets.
- **The Cost:** This consumes airtime. If you enable Airport, you typically drop to a lower control rate (e.g., 50Hz control + data) to make room for the data.

2. BAUD RATE ALIGNMENT

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

MAVLink is a serial protocol.

- **The Bottleneck:** The UART buffer.
- **The Rule:** You **MUST** set both the Receiver UART and the Flight Controller UART to **460800 baud** (or higher).
- **Why?** ELRS sends data in high-speed bursts. If the UART is too slow (57600), the buffer overflows instantly, and packets drop. 460k ensures the UART is faster than the Air Link.

ARCHITECTURE (THE ENGINEER'S VIEW)

1. FLIGHT CONTROLLER SETUP

To use Airport, you bypass the `RCProtocol` decoder.

- `SERIALx_PROTOCOL = 2` (MAVLink 2).
- `SERIALx_BAUD = 460` (460800).
- *Note:* You do **NOT** use `23` (RCIN). You treat the ELRS receiver exactly like a SiK Radio.

2. THROUGHPUT REALITY

- **Uplink (TX → Drone):** Very fast. Good for upload.
- **Downlink (Drone → TX):** Limited by Telemetry Ratio.
  - *1:2 Ratio:* 50\% of packets are downlink. Good for HUD.
  - *1:64 Ratio:* 1.5\% of packets are downlink. Useless for HUD.

COMMON ISSUES & TROUBLESHOOTING

"MISSION PLANNER CONNECTS BUT PARAMS ARE SLOW"

- **Cause:** Bandwidth saturation. MAVLink overhead is high.
- **Fix:** Use 1:2 Ratio. Increase Packet Rate (e.g., 333Hz). Be patient.

"NO RC CONTROL"

- **Cause:** You set the Serial Port to MAVLink (2). ArduPilot is now listening for MAVLink commands, not RC Channels.
- **Fix:** You must send RC Override via MAVLink *OR* use a receiver that outputs CRSF on one pin (for control) and MAVLink on another pin (for data).

SOURCE CODE REFERENCE

- **CRSF Driver (Alternative):** `AP_RCProtocol_CRSF.cpp`

PRACTICAL GUIDE: CONFIGURING "AIRPORT"

WARNING: This disables standard RC control on the Airport pins. Ensure you have a failsafe or a separate RC connection (e.g. CRSF on a different UART) if you are just testing.

STEP 1: RECEIVER CONFIG (WIFI)

1. Connect to your ELRS Receiver WiFi.
2. Go to **Model** tab.
3. **Serial Protocol:** Transparent Serial (Airport).
4. **Baud Rate:** 460800.
5. **Save & Reboot.**

STEP 2: TRANSMITTER CONFIG (LUA)

1. Open ELRS Lua script.
2. **Packet Rate:** 333Hz Full (or 100Hz Full).

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





3. **Telem Ratio:** 1:2 (Max Bandwidth).

STEP 3: ARDUPILOT CONFIG

1. Identify the UART connected to the receiver (e.g., SERIAL1).
2. `SERIAL1_PROTOCOL = 2` (MAVLink 2).
3. `SERIAL1_BAUD = 460` (460800).
4. **Reboot.**

STEP 4: THE CONNECTION

1. Connect your PC/Tablet to the ELRS TX module (via WiFi, Bluetooth, or USB).
2. Open Mission Planner.
3. Connect via UDP/COM.
4. You should see the param download start. It will be slower than USB, but it works.

THE BACKPACK ARCHITECTURE: WI-FI & ESP8285

EXECUTIVE SUMMARY

The "Backpack" is a secondary processor (ESP8285) inside your ELRS Transmitter module. It acts as a bridge between the ELRS radio link and the outside world (Wi-Fi/Bluetooth). For MAVLink HUD users, the Backpack is the critical link that turns your Radio Controller into a **Flying Wi-Fi Hotspot** for your tablet or phone.

THEORY & CONCEPTS

1. THE SIDECAR DESIGN

The main ELRS chip (ESP32) is busy handling the high-speed LoRa timing (500Hz). It cannot handle Wi-Fi traffic simultaneously without jitter.

- **The Solution:** A second chip (ESP8285) handles the "Slow" stuff (VRX control, Wi-Fi, Logging).
- **Inter-Chip Link:** The main ESP32 talks to the Backpack ESP8285 via an internal UART (`CRSF` or `ESP-NOW`).

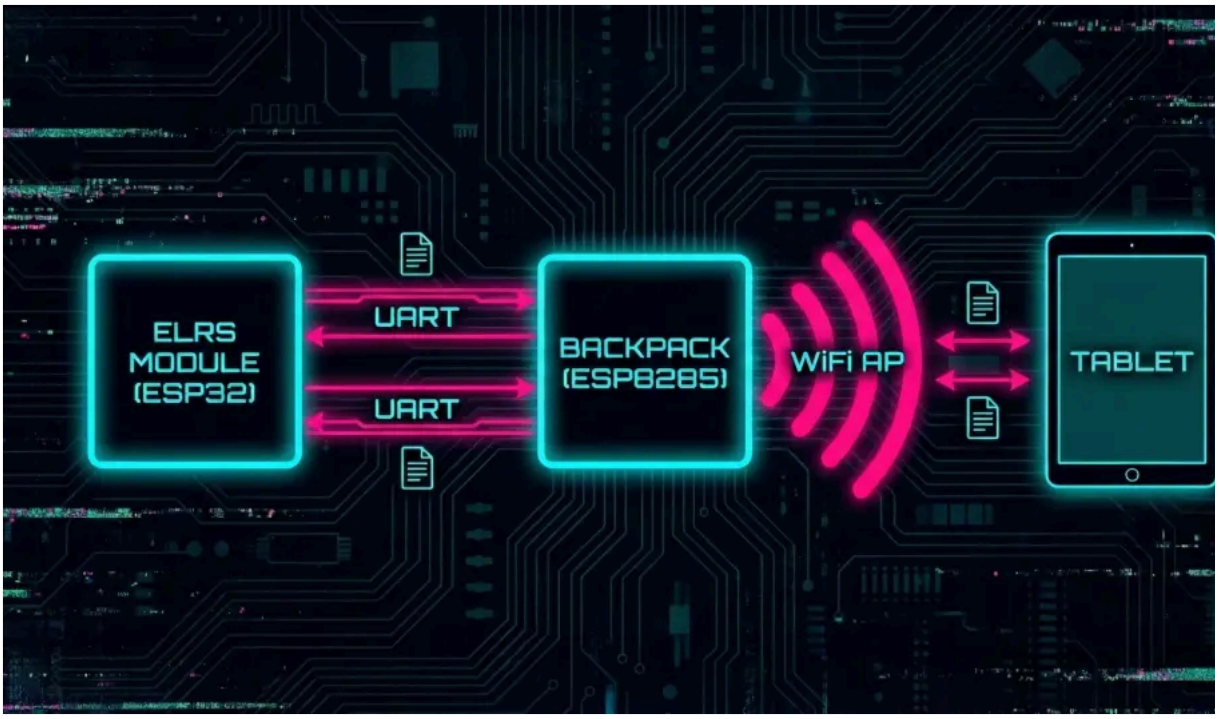
2. THE NETWORK TOPOLOGY

When you enable "Backpack Wi-Fi":

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY







1. The Backpack creates an Access Point (`ExpressLRS TX Backpack`).
2. IP Address: `10.0.0.1` (Gateway).
3. It opens a **UDP Server** on Port `14550` .
4. **Routing:** Any byte received from the Air (Drone) is forwarded to UDP 14550. Any byte received from UDP 14550 is forwarded to the Air.

ARCHITECTURE (THE ENGINEER'S VIEW)

1. THE MAVLINK HUD CONNECTION

Your Android App connects to the Backpack, not the Drone.

- **Target IP:** `10.0.0.1` (The Backpack).
- **Port:** `14550` (MAVLink Standard).
- *Data Flow:* App → WiFi → Backpack → Internal UART → ELRS TX → LoRa → ELRS RX → Drone UART → ArduPilot.

2. VRX INTEGRATION (VIDEO)

The Backpack was originally designed to switch video channels.

- **Logic:** When you change the VTX channel on your Radio Script, the Backpack sends a command (via ESP-NOW or Serial) to your Goggles (HDZero/Analog/Walksnail) to switch them to the same channel automatically.

COMMON ISSUES & TROUBLESHOOTING

- **"No Link" in HUD:** Ensure you are connected to the *Backpack* Wi-Fi, not the *TX Module* Wi-Fi (they are different). The Backpack usually has "Backpack" in the SSID.
- **Lag:** Wi-Fi is noisy. Ensure your phone is close to the radio. 5.8GHz video can interfere with 2.4GHz Wi-Fi; keep channels separated.

SOURCE CODE REFERENCE

- **Project:** ELRS Backpack Firmware

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

LINK SYMMETRY & TX-AS-RX

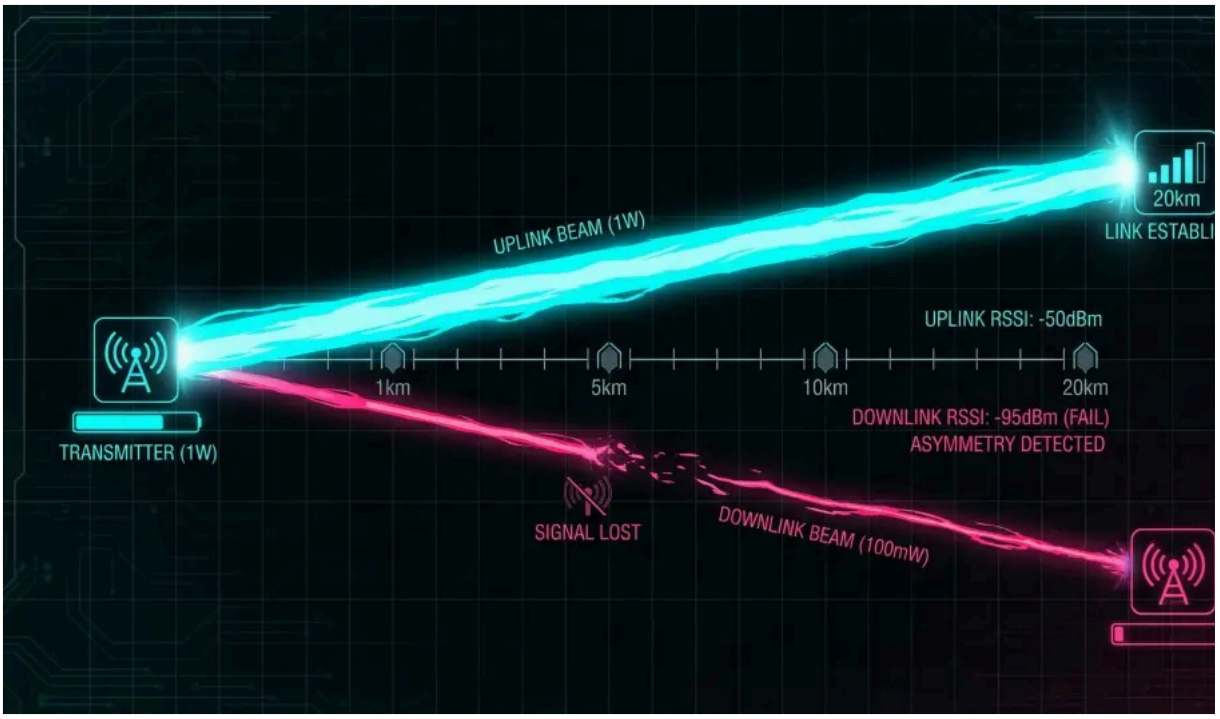
EXECUTIVE SUMMARY

Standard ELRS systems are **Asymmetric**. The Transmitter (Ground) has 1000mW of power. The Receiver (Air) often has only 100mW (telemetry). This means you can control the drone at 20km, but you lose telemetry (HUD data) at 5km. To solve this for professional MAVLink use, we use the "TX-as-RX" strategy: flashing a high-power 1000mW Transmitter Module with Receiver firmware.

THEORY & CONCEPTS

1. THE LINK BUDGET

Wireless range is determined by the **Link Budget**.



- `Received Power = Transmit Power + Gains - Losses`.
- **Uplink (Ground → Air):** 1W (+30dBm) TX → -110dBm Sensitivity = 140dB Budget.
- **Downlink (Air → Ground):** 100mW (+20dBm) TX → -110dBm Sensitivity = 130dB Budget.
- **The Gap:** The Uplink is 10dB stronger (10x power). You will lose MAVLink long before you lose Control.

2. SYMMETRICAL LINKS

For a reliable MAVLink HUD, you need a **Symmetrical Link** (1W Uplink, 1W Downlink).

- **The Hardware:** The RadioMaster Bandit (or similar 1W modules) contains the exact same RF chips (ESP32 + SX1280 + PA/LNA) as a high-end receiver. The only difference is the firmware.

ARCHITECTURE (THE ENGINEER'S VIEW)

1. FLASHING TX AS RX

ExpressLRS Configurator allows you to flash "RX" firmware onto a "TX" target.

- **Logic:** The firmware re-maps the pins (DIO) to act as a receiver.

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

- **Wiring:** You must wire the "TX Module" to the Flight Controller UART.
  - *Note:* TX modules usually don't have solder pads for this; you may need to hack the case or use the module bay pins (if supported by the "RX" firmware map).

2. POWER & THERMAL MANAGEMENT

A 1W transmitter gets hot.

- **Cooling:** Standard receivers rely on airflow. A 1W "TX-as-RX" unit might need a fan or heatsink, especially if buried inside a fuselage.
- **Power Supply:** A 1W module draws ~0.5A at 5V. Ensure your Flight Controller's BEC can handle this load, or power it directly from the battery (if the module supports VBAT).

KEY PARAMETERS

- `Telemetry Power` (Lua): Set to "Match TX" or a fixed high value (e.g., 250mW, 500mW) to maintain the link.
- `Fan Threshold`: Configure if using a module with an active fan.

SOURCE CODE REFERENCE

- **Hardware Targets:** ELRS Targets

TELEMETRY RATIOS & BANDWIDTH MATH

EXECUTIVE SUMMARY

Unlike traditional radios with fixed telemetry, ELRS allows you to tune the **Uplink/Downlink Balance**. The "Telemetry Ratio" determines how many packet slots are sacrificed to send data back from the drone. For a MAVLink HUD, choosing the wrong ratio (e.g., 1:64) results in unusable lag.

THEORY & CONCEPTS

1. THE RATIO MATH

The ratio defines the frequency of downlink packets.

- **Packet Rate (Air Rate):** The fundamental speed (e.g., 250Hz).
- **Ratio:** 1:2 means "1 Telemetry packet for every 2 Control packets."
- **Formula:** `Telemetry_Rate = Packet_Rate / Ratio`.

2. BANDWIDTH CALCULATION

How many bytes per second do you get?

- *Example:* 250Hz, 1:2 Ratio.
  - Telemetry Rate = 125 Hz.
  - Payload per packet = ~6 bytes (standard ELRS) or variable for Airport.
  - **Airport Bandwidth:** Optimized to fill the packet. Approx **1000 Bytes/Second**.
- *Counter-Example:* 50Hz, 1:64 Ratio.
  - Telemetry Rate = 0.78 Hz (Less than 1 packet per second!).
  - **Bandwidth:** ~5 Bytes/Second.
  - *Result:* Your HUD updates once every 2 seconds. Mission Planner times out.

ARCHITECTURE (THE ENGINEER'S VIEW)

1. THE MAVLINK CONSTRAINT

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





MAVLink is "heavy."

- **Heartbeat:** 9 bytes.
- **Attitude:** 28 bytes.
- **Global Position:** 28 bytes.
- **Sys Status:** 31 bytes.
- *Total:* A basic stream requires ~500 bytes/sec for 5Hz updates.
- *Conclusion:* You **MUST** use a high packet rate (250Hz+) and a high ratio (1:2) to support MAVLink.

2. RACE MODE VS. STANDARD

- **Standard:** Telemetry is always sent.
- **Race:** Telemetry is disabled when Armed (to reduce latency jitter).
- *Warning:* Never use "Race" mode with a HUD app. Your screen will freeze the moment you take off.

KEY PARAMETERS

- `Telemetry Ratio` (Lua): Set to **1:2** for MAVLink.
- `Packet Rate` (Lua): Set to **250Hz** (900MHz) or **333Hz/500Hz** (2.4GHz) for best results.

SOURCE CODE REFERENCE

- **Bandwidth Calculator:** ELRS Bandwidth Tool

ARDUPILOT INTEGRATION: SETUP & PARAMS

EXECUTIVE SUMMARY

Integrating ELRS with ArduPilot is straightforward, but optimizing it for MAVLink requires specific settings. The default "Auto-Detect" works for RC control, but high-speed telemetry requires locking the baud rates and configuring the RSSI feedback path.

THEORY & CONCEPTS

1. THE BAUD RATE BOTTLENECK

ELRS air rates have increased faster than the default serial rates.

- **Legacy:** CRSF defaults to 420kbaud.
- **The Problem:** At 500Hz or 1000Hz air rates, the serial link becomes the bottleneck. If the receiver receives packets faster than it can push them down the wire, it drops them.
- **The Fix:** ArduPilot and ELRS now standardize on **460,800 baud**. This is a standard integer multiple of 115200, making it more stable on STM32 UARTs than the "odd" 416k/420k rates.

ARCHITECTURE (THE ENGINEER'S VIEW)

1. PROTOCOL SELECTION

- **RC Control:** `SERIALx_PROTOCOL = 23` (RCIN). ArduPilot detects the CRSF header.
- **MAVLink Bridge:** `SERIALx_PROTOCOL = 2` (MAVLink 2). The UART is passed directly to the GCS router.

2. RSSI VS. LINK QUALITY

Legacy OSDs expect "RSSI" (Signal Strength). ELRS users care about "LQ" (Link Quality).

- **The Translation:** `RSSI_TYPE = 5` (Telemetry).
- `RC_OPTIONS` : Bit 8 (`USE_CRSF_LQ_AS_RSSI`).

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- *Effect:* ArduPilot maps the 0-100 LQ value onto the 0-100 RSSI range for OSDs that don't support a dedicated LQ field.

KEY PARAMETERS

PARAMETER	VALUE	DESCRIPTION
`SERIALx_PROTOCOL`	23	RCIN (for Control).
`SERIALx_BAUD`	460	460,800 Baud. Critical for high-speed telemetry.
`RC_OPTIONS`	256	Use LQ for RSSI (Bit 8).
`RSSI_TYPE`	5	Receive RSSI from Telemetry protocol.

SOURCE CODE REFERENCE

- **Driver:** `AP_RCProtocol_CRSF::process_link_stats_frame()`

PRACTICAL GUIDE: THE 460K BAUD STANDARD

If you use ELRS 3.0+ with ArduPilot 4.4+, you should standardize on 460,800 baud.

THE PROBLEM

- **Default:** Many ELRS receivers ship defaulting to 420,000 baud.
- **Symptoms:** You see occasional "Failsafe" warnings even when close, or telemetry is choppy. This is because 420k is not a "native" UART speed, leading to framing errors on some F4 flight controllers.

THE FIX

1. **WiFi Config:** Connect to your ELRS Receiver via WiFi.
2. **Model Tab:** Set "Serial Protocol" to **CRSF** and "Baud Rate" to **460800**.
3. **ArduPilot:** Set `SERIALx_BAUD = 460` (where `x` is your RC port).
4. **Reboot:** Reboot both the receiver and the flight controller.

HEALTH CHECK (OSD)

- **RxBt (Receiver Battery):** If this updates rapidly (several times a second), your telemetry link is healthy.
- **Link Quality (LQ):** Should stay at `9:100` (mode 2) or `100` (mode 1) during bench testing. If it dips to `50` or below just a few meters away, check your antenna orientation.

ADVANCED MODES: F, D, AND K

EXECUTIVE SUMMARY

ELRS is not static. It supports multiple modulation modes tailored for different environments. **F-Mode** (FLRC) is for pure racing speed. **D-Mode** (DVDA) is for reliability in noisy environments. **K-Mode** (Gemini) is the ultimate interference rejection system.

THEORY & CONCEPTS

1. FLRC (FAST LONG RANGE COMMUNICATION)

- **The Physics:** FLRC is *not* LoRa. It is an optimized FSK (Frequency Shift Keying).
- **The Trade:** It strips away the heavy error correction of LoRa to achieve ultra-low latency (1ms).
- **The Cost:** It has a "Digital Cliff." It doesn't degrade gracefully. You have perfect control, then instant failsafe.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- **Use Case:** Drone Racing.

2. DVDA (DEJA VU DIVERSITY AID)

- **The Logic:** Send every packet twice, on two different frequencies.
- **The Math:** If the probability of packet loss on Frequency A is 10\%, the probability of losing *both* packets is 1\% (10\% * 10\%).
- **The Result:** massive increase in Link Quality (LQ) consistency at the cost of halving the update rate (e.g., 500Hz → D250).

3. GEMINI (K-MODE)

- **The Hardware:** Requires dual RF chips (SuperD / Gemini TX).
- **The Operation:** It transmits on two frequencies *simultaneously*.
- **The Benefit:** Immunity to narrowband interference. If someone turns on a Wifi router on Channel 1, Channel 2 still gets through.

ARCHITECTURE (THE ENGINEER'S VIEW)

1. THE SWITCH

ELRS switches modes via the Lua script.

- **ArduPilot's Role:** ArduPilot is agnostic. It just receives RC packets.
- **However:** If you switch to a mode with a different packet rate (e.g., 500Hz → 50Hz), you *must* ensure your `SRx_` telemetry rates are lowered to match the new bandwidth constraints, or you will buffer-bloat the link.

KEY PARAMETERS

- `Packet Rate`: Select **F1000** for racing, **D250** for bando/freestyle, **50Hz** for Long Range.

SOURCE CODE REFERENCE

- **Modulation Logic:** ELRS GitHub

THE YAAPU PROBLEM & THE HUD SOLUTION

EXECUTIVE SUMMARY

For years, ArduPilot users have relied on the **Yaapu Telemetry Script** to see flight data on their radio screens. However, as pilots move to **MAVLink over ELRS** for professional control, they hit a wall: enabling MAVLink breaks Yaapu. You cannot have both. This document explains the technical reason for this conflict and why the **MAVLink HUD** architecture is the superior evolution for modern ground stations.

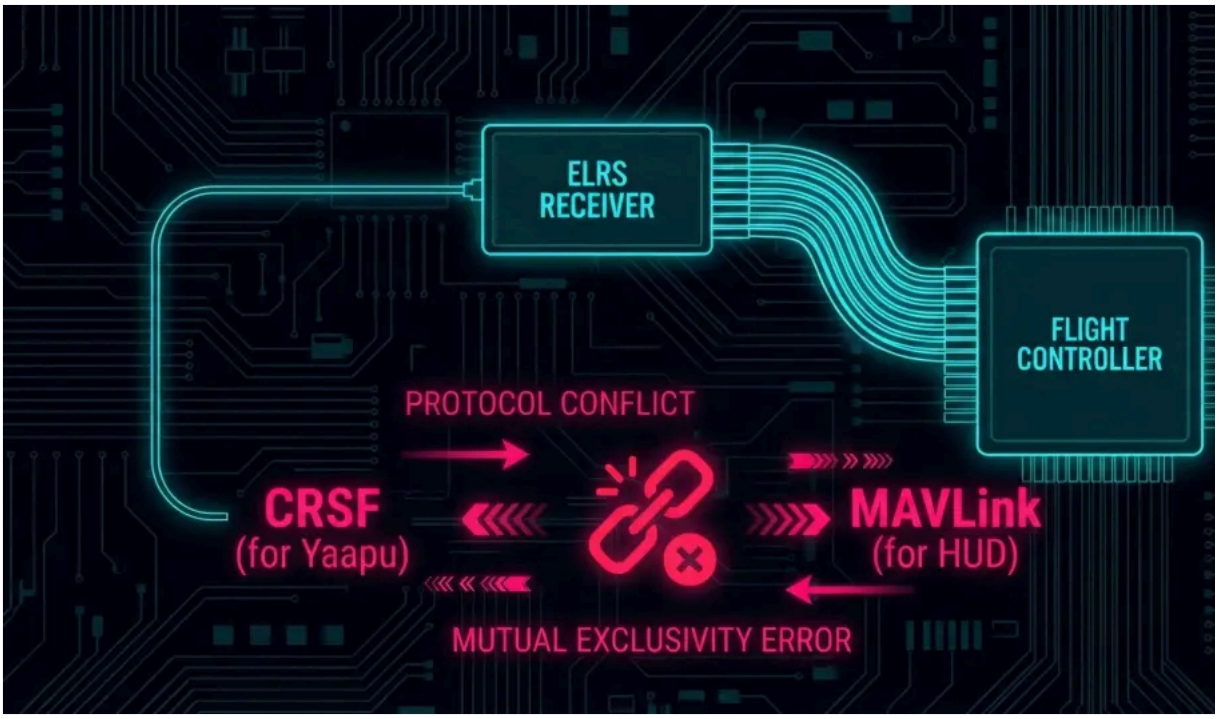
THE CONFLICT: PROTOCOL EXCLUSIVITY

A serial link (UART) can only speak one language at a time.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY







- **Yaapu (CRSF Telemetry):** Requires the receiver to speak "CRSF." ArduPilot translates its internal state into CRSF frames (GPS, Battery, Attitude) and sends them down the wire.
- **MAVLink HUD (Airport):** Requires the receiver to act as a **Transparent Bridge**. It doesn't "speak" anything; it just shuffles raw bytes back and forth.
- **The Problem:** If you enable "Airport" to get MAVLink for your tablet, the radio stops seeing CRSF Telemetry packets. The Yaapu script on your radio screen goes dark saying "No Telemetry."

THE "CONVERTER" NIGHTMARE

Some users try to hack around this by using an ESP32 "Converter" board.

- **The Hack:** They wire the receiver to the ESP32, which splits the signal. It sends MAVLink to the Wifi and translates MAVLink back to CRSF for the radio.
- **The Cost:** This adds latency, wiring complexity, and a new point of failure. It is a band-aid solution.

THE HUD SOLUTION: MOVE THE SCREEN

The fundamental question is: *Why are we trying to display complex flight data on a 128×64 monochrome LCD from 2015?*

1. THE SCREEN REAL ESTATE

- **Radio Screen:** Small, low resolution, mono/low-color. Good for voltage, bad for maps.
- **MAVLink HUD (Phone/Tablet):** 1080p+, High Contrast OLED, 60fps GPU rendering.
  - *Result:* We can render a true **Primary Flight Display (PFD)** with an artificial horizon, scrolling tapes, and a moving map that is readable in sunlight.

2. PROCESSING POWER

- **Radio (STM32):** Running a LUA script burdens the radio's CPU. Complex scripts can slow down the user interface or even crash the radio.
- **Phone (Snapdragon/Apple):** The HUD app runs on a dedicated multi-core processor with a GPU. It handles map rendering, logging, and voice synthesis without breaking a sweat.

3. THE ARCHITECTURE SHIFT

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

Instead of forcing the Radio Controller to be a Ground Control Station, we let the Radio focus on **Control** (High Speed ELRS) and offload the **Visualization** to the device best suited for it (The Tablet).

- **The Link:** The ELRS Backpack acts as the bridge. The Radio handles the sticks. The Backpack handles the Wi-Fi data stream to the HUD.
- **The Result:** A cleaner, more professional, and more robust system.

SUMMARY

If you are flying Line-of-Sight, Yaapu is great. If you are flying Long Range or Autonomous missions, **MAVLink is mandatory**. Don't fight the protocol; embrace the HUD architecture.

SOURCE CODE REFERENCE

- **CRSF Telemetry (Yaapu):** `AP_CRSF_Telem.cpp`
- **MAVLink Bridge (HUD):** `AP_RCProtocol_CRSF.cpp` (When in pure passthrough)

PRACTICAL GUIDE: CHOOSING YOUR PATH

You must choose one. You cannot have both on the same UART.

PATH A: THE RACER (YAAPU)

- **Hardware:** ELRS Receiver.
- **Protocol:** `SERIALx_PROTOCOL = 23` (RCIN).
- **Advantages:** Clean setup, data on radio.
- **Disadvantages:** No Mission Planner, no Maps, no "Click to Fly".

PATH B: THE PROFESSIONAL (MAVLINK HUD)

- **Hardware:** ELRS Receiver (Airport Mode) or separate Telemetry Radio.
- **Protocol:** `SERIALx_PROTOCOL = 2` (MAVLink 2).
- **Advantages:** Full GCS control, moving map, HD telemetry.
- **Disadvantages:** Your radio screen will be blank (except for basic LQ/RSSI from the module itself).

THE COMPROMISE (DUAL LINKS)

If you absolutely must have both:

1. **UART 1:** ELRS Receiver (RCIN) → Feeds Yaapu.
2. **UART 2:** ESP8266 WiFi Bridge (MAVLink) → Feeds Tablet.

- *Note:* This requires two UARTs on the flight controller.

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





CHAPTER 11: NAVIGATION & MISSION

NAVIGATION ARCHITECTURE

EXECUTIVE SUMMARY

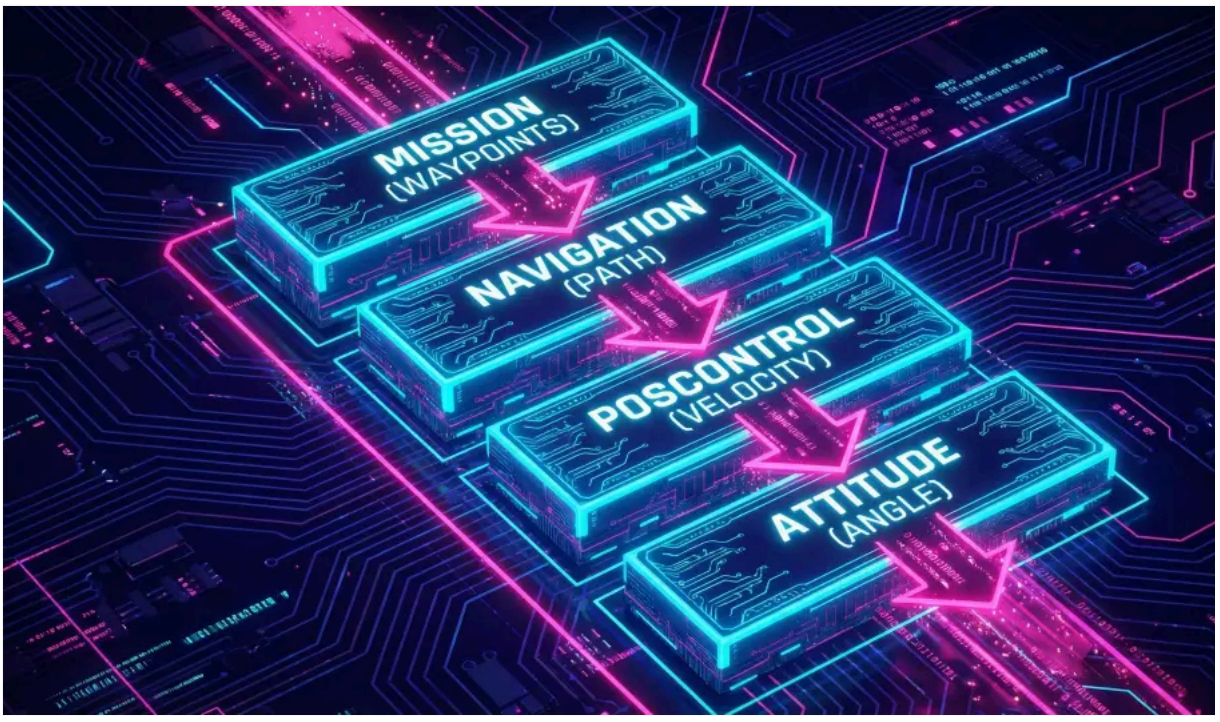
Navigation in ArduPilot is not a monolithic block; it is a layered architecture separating **Mission Management** (high-level intent) from **Position Control** (kinematic execution) and **Attitude Control** (stabilization).

This separation allows `AP_Mission` to remain vehicle-agnostic, simply feeding "Target Locations" to vehicle-specific libraries (`AC_WPNav` for Copter, `AP_L1_Control` for Plane).

THEORY & CONCEPTS

THE CONTROL LOOP HIERARCHY

The autopilot operates on a cascaded control loop architecture, where slower outer loops drive faster inner loops.



1. **Mission Layer (10Hz):** `AP_Mission` determines the *active command* (e.g., "Fly to Waypoint 4"). It monitors completion criteria (distance, altitude, time) and advances the index.
2. **Navigation Layer (50Hz):**
   - **Copter:** `AC_WPNav` generates a kinematic path (Spline or Straight Line) to the target. It outputs a **Desired Velocity Vector** and **Position Target**.
   - **Plane:** `AP_L1_Control` calculates a **Lateral Acceleration** to minimize cross-track error (L1 Guidance).
3. **Position/Velocity Layer (50Hz - 400Hz):** Converts Desired Velocity/Acceleration into Desired Lean Angles (Copter) or Control Surface Deflections (Plane).
4. **Attitude Layer (400Hz+):** The Rate Controller drives the motors/servos to achieve the Desired Lean Angles.

REFERENCE FRAMES

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

- **NEU (North-East-Up):** The standard internal frame for navigation relative to the EKF Origin (Home).
- **Body Frame:** X-forward, Y-right, Z-down. Used for sensor data and low-level control.
- **Terrain Frame:** Altitude relative to the loaded terrain database or rangefinder.

CODEBASE INVESTIGATION

1. THE MISSION DIRECTOR: `AP_MISSION::UPDATE()`

Located in `libraries/AP_Mission/AP_Mission.cpp`, this function is the heartbeat of autonomous operations.

- **Logic:**
  1. Checks `verify_command(_nav_cmd)` to see if the current objective is met.
  2. If `true`, it calls `advance_current_nav_cmd()` to load the next instruction.
  3. It also manages "Do Commands" (servos, relays) which run concurrently with navigation.

2. VEHICLE IMPLEMENTATION: COPTER VS. PLANE

COPTER (`ARDUCOPTER/MODE_AUTO.CPP`)

When `ModeAuto::run()` executes:

- It calls `wp_nav->update_wpnav()`.
- This moves the "Leash" (Target Position) towards the destination.
- The Position Controller (`pos_control->update_z_controller()`) pulls the drone towards the Leash.

PLANE (`ARDUPLANE/MODE_AUTO.CPP`)

Plane uses a different strategy based on fluid dynamics:

- It calculates `nav_roll_cd` (Commanded Centi-Degrees Roll) using L1 logic to track the line between waypoints.
- It calculates `nav_pitch_cd` and `throttle` using **TECS** (Total Energy Control System) to manage Speed and Height simultaneously.

SOURCE CODE REFERENCE

- **Mission Logic:** `libraries/AP_Mission/AP_Mission.cpp`
- **Copter Auto:** `ArduCopter/mode_auto.cpp`
- **Plane Auto:** `ArduPlane/mode_auto.cpp`

PRACTICAL GUIDE: MONITORING MISSION STATUS

Operators often need to know *why* a drone is pausing or what it is waiting for.

1. MAVLINK MESSAGES

Monitor `MISSION_CURRENT` (Msg ID 42).

- `seq`: The index of the *active* command.
- If `seq` stops incrementing, the drone is executing that command.

2. COMMON "STUCK" SCENARIOS

- **Takeoff Wait:** If the drone arms but doesn't spin up in Auto, check `MIS_OPTIONS`. Some configurations wait for a "Mission Start" MAVLink command.
- **Altitude Wait:** If `verify_nav_wp` never completes, check your Altitude Acceptance Radius (`WPNAV_RADIUS` or `WP_RADIUS`). If the drone cannot reach the exact Z-altitude due to baro drift or max throttle, it will circle indefinitely.

3. DEBUGGING WITH LOGS

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





Open the DataFlash log and look for `CMD` messages.

- **CTot:** Command Total.
- **CNum:** Current Command Number.
- **CId:** Command ID (e.g., 16 = Waypoint, 22 = Takeoff).
- *Tip:* If `CNum` changes but the drone behaves oddly, check the `P1` (Parameter 1) field. For Loiter commands, this is often the "Time" or "Turns" which might be set incorrectly to 0 (forever).

L1 CONTROL LOGIC

EXECUTIVE SUMMARY

The **L1 Controller** is ArduPilot's primary guidance algorithm for fixed-wing aircraft and rovers. Unlike a simple PID controller that reacts to cross-track error, L1 is a **Nonlinear Guidance Logic** that looks ahead on the path.

It computes a **Lateral Acceleration Demand** (`_latAccDem`) to curve the vehicle onto the desired path, similar to how a human driver looks down the road rather than at the hood ornament.

THEORY & CONCEPTS

1. THE "L1" DISTANCE

The core concept is the **L1 Point**: a virtual target point on the desired path at a specific distance (L_1) ahead of the vehicle.

- **Physics:** The vehicle attempts to fly a circular arc that intersects the L1 point.
- **Effect:**
  - **Long L_1:** Stable, smooth path converging (Under-damped).
  - **Short L_1:** Aggressive, fast convergence (Over-damped).

2. CENTRIPETAL ACCELERATION

The controller outputs a lateral acceleration command (a_s) based on the L1 distance and the angle eta (Nu) between the velocity vector and the L1 point.

$$a_s = 2 \\ frac V^2 L_1 \\ sin \\ eta$$

Where:

- V = Ground Speed
- eta = Angle to L1 point
- L_1 = Lookahead distance

3. VECTOR FIELDS

ArduPilot's implementation is actually a **Vector Field**. It calculates a desired velocity vector at every point in space. Even if the plane is flying away from the waypoint, the vector field "flows" back towards the track line.

CODEBASE INVESTIGATION

1. THE CORE UPDATE LOOP: `UPDATE_WAYPOINT()`

Located in `libraries/AP_L1_Control/AP_L1_Control.cpp`.

1. **Calculate L1 Distance:**

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





```
_L1_dist = MAX(0.3183099f * _L1_damping * _L1_period * groundSpeed, dist_min);
```

- The distance scales with **Ground Speed**. Faster planes look further ahead to stay stable.
- `NAVL1_PERIOD` determines the time constant.

2. **Calculate Nu (**eta**):**
   The code calculates the angle to the L1 point.
   - If far from track: `Nu` points directly at the line (Capture Mode).
   - If near track: `Nu` points along the track with an intercept angle (Track Mode).
3. **Compute Acceleration Demand:**

   ```
   _latAccDem = K_L1 * groundSpeed * groundSpeed / _L1_dist * sinf(Nu);
   ```

   This demand is fed into the roll controller (`nav_roll_cd`).

2. LOITER LOGIC: `UPDATE_LOITER()`

Loitering is treated as a continuous turn.

- **Capture Phase:** If outside the circle, it flies a tangent to the circle edge.
- **Hold Phase:** It balances Centripetal Acceleration (`latAccDemCircCtr`) with PD control (`latAccDemCircPD`) to correct radius errors.

KEY PARAMETERS

PARAMETER	DEFAULT	DESCRIPTION
`NAVL1_PERIOD`	20	(Seconds) The aggressiveness of the turn. Lower = sharper turns. Higher = smoother navigation.
`NAVL1_DAMPING`	0.75	Damping ratio. 0.7-0.8 is ideal. Higher values reduce overshoot but slow convergence.
`NAVL1_LIM_BANK`	0	(Degrees) Limits the bank angle specifically for loiters. 0 = Use standard roll limit.

SOURCE CODE REFERENCE

- **Implementation:** `libraries/AP_L1_Control/AP_L1_Control.cpp`
- **Header:** `libraries/AP_L1_Control/AP_L1_Control.h`

PRACTICAL GUIDE: TUNING L1 NAVIGATION

Common symptom: "My plane weaves (snakes) around the waypoint line."

STEP 1: CHECK `NAVL1_PERIOD`

This is the most common culprit.

- **Symptom:** Fast oscillation (weaving) around the track.
- **Fix: Increase** `NAVL1_PERIOD` . Default is 20. Try 25.
  - *Why?* The plane is reacting too fast to small errors. Increasing the period forces it to "look further ahead," smoothing out the path.

STEP 2: CHECK `NAVL1_DAMPING`

- **Symptom:** The plane turns onto the line but overshoots it, then corrects back and overshoots again (sloppy S-turns).
- **Fix: Increase** `NAVL1_DAMPING` by 0.05.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- *Why?* It needs more damping to settle onto the line without overshoot.

STEP 3: THE "TOO SLOW" TURN

- **Symptom:** Plane takes forever to turn towards a waypoint, flying a huge lazy arc.
- **Fix: Decrease** `NAVL1_PERIOD`. Try 15 or 12.
  - *Warning:* If you go too low (e.g., < 10), the plane may become unstable and induce roll oscillation.

SPLINE TRAJECTORIES

EXECUTIVE SUMMARY

For multirotors, stopping at every waypoint is inefficient and visually jarring. **Spline Navigation** allows the vehicle to fly a smooth, continuous curve through a series of waypoints without stopping.

ArduPilot uses **Cubic Hermite Splines**, which are defined by two points (Start, End) and two tangent vectors (Start Velocity, End Velocity). This ensures continuity of both **Position** (C^0) and **Velocity** (C^1) across segments.

THEORY & CONCEPTS

1. WHY SPLINES?

- **Straight Lines:** Requires stopping at corners (infinite acceleration) or "cutting the corner" (leash logic).
- **Splines:** Define a curve where the velocity vector enters and exits the waypoint exactly aligned with the path, allowing high-speed transit.

2. CUBIC HERMITE SPLINE MATH

The position $P(t)$ for t
$in[0, 1]$ is calculated using four basis functions ($h_{00}, h_{10}, h_{01}, h_{11}$) acting on the control points:

$$P(t) = h_{00}(t)P_0 + h_{10}(t)M_0 + h_{01}(t)P_1 + h_{11}(t)M_1$$

Where:

- P_0: Origin Point
- P_1: Destination Point
- M_0: Origin Velocity Tangent
- M_1: Destination Velocity Tangent

3. CENTRIPETAL ACCELERATION LIMITS

A tighter curve requires higher lateral acceleration ($a_c = v^2/r$). The controller dynamically limits the speed along the spline to ensure the vehicle never exceeds `WPNAV_ACCEL_C` (Corner Acceleration) or the physical lean angle limits.

CODEBASE INVESTIGATION

1. SETTING THE SPLINE: `AC_WPNAV::SET_SPLINE_DESTINATION()`

Located in `libraries/AC_WPNav/AC_WPNav.cpp`.

- It calculates the **Tangents** (Velocity Vectors) for the spline.
- **Crucial Logic:** The tangent at Waypoint N is parallel to the line connecting $N-1$ and $N+1$. This creates the smooth "flow" through the point.

2. THE SOLVER: `SPLINECURVE::UPDATE_SOLUTION()`

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





Located in `libraries/AP_Math/SplineCurve.cpp`.

- It pre-calculates the Hermite coefficients so the realtime loop is fast.

```
_hermite_solution[0] = origin;
_hermite_solution[1] = origin_vel;
_hermite_solution[2] = -origin*3.0f -origin_vel*2.0f + dest*3.0f - dest_vel;
_hermite_solution[3] = origin*2.0f + origin_vel -dest*2.0f + dest_vel;
```

3. EXECUTION: `ADVANCE_TARGET_ALONG_TRACK()`

- Moves the "Target Point" along the mathematical curve.
- It adjusts `dt` (time step) based on the kinematic limits. If the curve is tight, it slows down the "time" variable to keep physical acceleration within bounds.

SOURCE CODE REFERENCE

- **Navigation Logic:** `libraries/AC_WPNav/AC_WPNav.cpp`
- **Math Engine:** `libraries/AP_Math/SplineCurve.cpp`

PRACTICAL GUIDE: MISSION PLANNING WITH SPLINES

1. THE "LOOP-DE-LOOP" PROBLEM

A common issue is the drone making a weird loop or S-turn before a waypoint.

- **Cause:** The **Tangents** are too aggressive. If Waypoint 1, 2, and 3 form a sharp "V", the spline math tries to maintain velocity through the corner, forcing a wide overshoot.
- **Fix:** Avoid acute angles (< 90 degrees) in spline missions. Use two waypoints to round a sharp corner.

2. MIXING SPLINES AND LINES

You can mix `NAV_WAYPOINT` (Straight) and `NAV_SPLINE_WAYPOINT` (Curved) in the same mission.

- **Use Case:** Fly a straight grid for mapping, then use splines to turn around smoothly at the end of each row to save battery and time.

3. TUNING FOR CINEMATOGRAPHY

For smooth camera work:

- **Reduce** `WPNAV_ACCEL` : Makes the drone accelerate/decelerate slower along the curve.
- **Reduce** `WPNAV_JERK` : Softens the start/stop of movements.
- **Increase** `WPNAV_RADIUS` : Does NOT affect Splines (Splines always pass *through* the point), but affects straight-line transitions.

TERRAIN FOLLOWING

EXECUTIVE SUMMARY

Terrain Following allows a vehicle to maintain a constant height above the ground rather than above the takeoff point (Home). This is critical for low-level mapping or flying in mountainous regions.

ArduPilot manages this via the `AP_Terrain` library, which handles fetching, caching, and interpolating SRTM (Shuttle Radar Topography Mission) data.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





THEORY & CONCEPTS

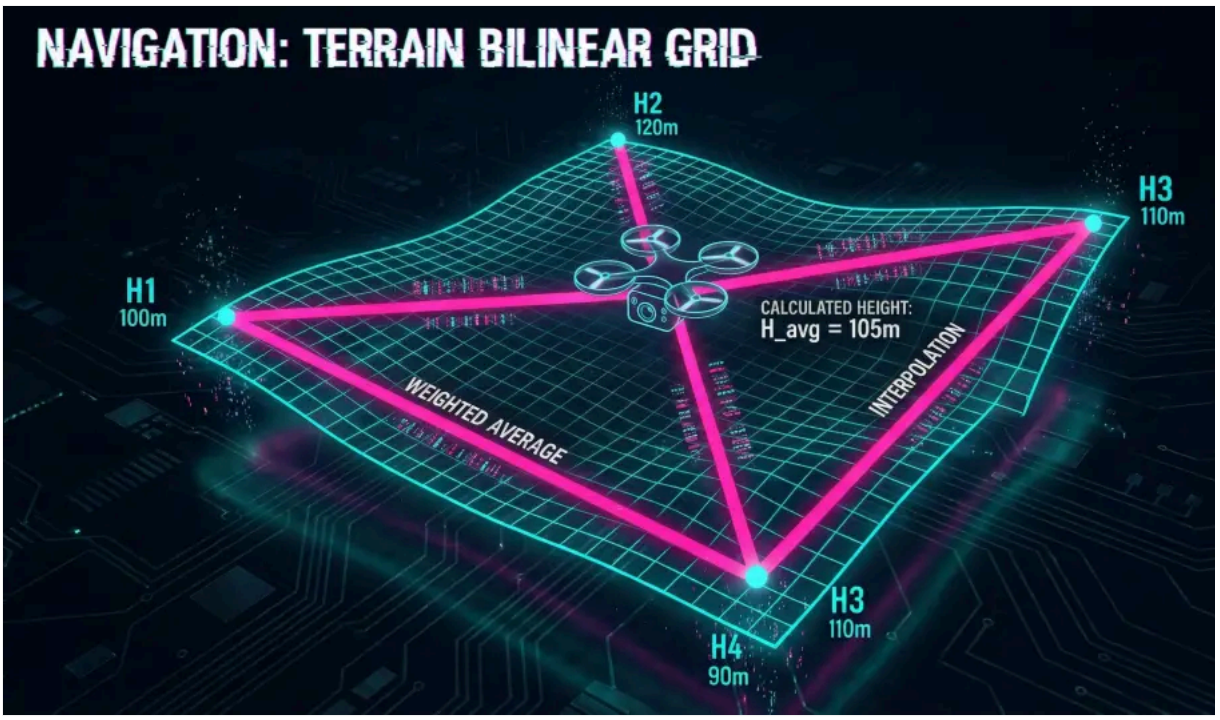
1. SRTM GRIDS

Terrain data is split into **Grids**.

- **Grid Point:** A single altitude value (int16, meters).
- **Grid Spacing:** The distance between points (default 100m, controlled by `TERRAIN_SPACING`).
- **Block:** A 2kB structure stored on the SD card containing an 8×7 array of 4×4 micro-grids.

2. BILINEAR INTERPOLATION

The vehicle is rarely exactly on top of a grid point. To find the height at the vehicle's location (x, y), `AP_Terrain` uses bilinear interpolation between the four surrounding grid points.



$$h(x, y) \\ approx \\ frac1(x_2 - x_1)(y_2 - y_1)(h_{11}(x_2 - x)(y_2 - y) + h_{21}(x - x_1)(y_2 - y) + \\ dots)$$

This ensures the ground "slope" is continuous and the drone doesn't "step" up and down as it crosses grid lines.

3. LOOKAHEAD

For fixed-wing aircraft, knowing the height *now* isn't enough. You need to know if there is a mountain ahead. `AP_Terrain::lookahead()` projects a line along the current bearing to calculate the "climb rate" needed to clear upcoming obstacles.

CODEBASE INVESTIGATION

1. THE CACHE SYSTEM: `AP_TERRAIN::ALLOCATE()`

Located in `libraries/AP_Terrain/AP_Terrain.cpp`.

- The system allocates a memory cache (LRU - Least Recently Used) of 12 blocks (`TERRAIN_CACHE_SZ`).

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

- It serves data from RAM if possible. If not, it schedules a background Disk I/O task (`schedule_disk_io`) to read from the SD card.

2. HEIGHT CALCULATION: `HEIGHT_AMSL()`

- Finds the correct grid block.
- Checks the `bitmap` to ensure valid data exists for the 4 surrounding points.
- Performs the bilinear interpolation:

```
float avg1 = (1.0f-info.frac_x) * h00  + info.frac_x * h10;
float avg2 = (1.0f-info.frac_x) * h01  + info.frac_x * h11;
float avg  = (1.0f-info.frac_y) * avg1 + info.frac_y * avg2;
```

3. FAILSAFES

If the system cannot find data (e.g., SD card error or GCS disconnect):

- It returns `false` .
- The navigation controller (`AC_WPNav`) will refuse to start a terrain-following mission or trigger a failsafe (RTL/Land) if already flying.

SOURCE CODE REFERENCE

- **Core Logic:** `libraries/AP_Terrain/AP_Terrain.cpp`
- **Data Structures:** `libraries/AP_Terrain/AP_Terrain.h`

PRACTICAL GUIDE: SAFE TERRAIN FOLLOWING

1. PRE-FLIGHT VALIDATION

Never launch a terrain mission blindly.

- **Check MAVLink Status:** The GCS should show "Terrain: 100\%".
- **Check** `TERRAIN_ENABLE` : Must be 1.
- **SD Card:** The `terrain/` folder on the SD card can grow large. Ensure it's not full.

2. THE `TERRAIN_SPACING` TRAP

- **Default:** 100m.
- **Problem:** If you fly in a canyon that is 50m wide, a 100m grid might completely miss the walls!
- **Solution:** For complex terrain, set `TERRAIN_SPACING` to 30 (meters). *Note: This increases download time and SD card usage significantly.*

3. "TERRAIN FAILSAFE"

What happens if the drone flies off the edge of your downloaded map?

- **Behavior:** It stops and hovers (Copter) or circles (Plane).
- **Recovery:** Switch to **AltHold** or **Stabilize** immediately. The auto-mission is stuck until data arrives.

MISSION STATE MACHINE

EXECUTIVE SUMMARY

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





The ArduPilot Mission State Machine is the "Brain" of Auto mode. It is responsible for sequencing events, ensuring that the drone finishes one task before starting the next.

It distinguishes between three types of commands:

1. **Navigation (NAV):** "Go somewhere" (blocking).
2. **Do (DO):** "Do something now" (non-blocking).
3. **Condition (COND):** "Wait for something" (blocking).

THEORY & CONCEPTS

THE COMMAND QUEUE

Conceptually, ArduPilot has two parallel execution queues:

1. **Navigation Queue:** Executes `NAV_WAYPOINT`, `NAV_LOITER`, `NAV_LAND`. Only **one** NAV command is active at a time. The drone physically moves to satisfy this command.
2. **Do Command Queue:** Executes `DO_SET_SERVO`, `DO_MOUNT_CONTROL`. These run *concurrently* with the NAV command.

- **Example:** A mission might have:
  - Command 1 (NAV): Fly to Waypoint A.
  - Command 2 (DO): Set Camera Tilt to 45 deg.
  - **Result:** The camera tilts *while* the drone is flying to Waypoint A.

CONDITION COMMANDS

Condition commands (e.g., `CONDITION_YAW`, `CONDITION_DELAY`) block the *navigation* queue from advancing but don't stop the vehicle's movement (unless it's a delay). They are "gates" that must be passed.

CODEBASE INVESTIGATION

1. THE VERIFICATION LOOP: `VERIFY_COMMAND()`

Located in `libraries/AP_Mission/AP_Mission.cpp` (and vehicle-specific implementations).
This function checks if the *current* command is finished.

- **NAV Commands:** Checks distance to target, altitude error, or loiter timer.
- **DO Commands:** almost always return `true` immediately (fire and forget).

2. ADVANCING THE MISSION: `ADVANCE_CURRENT_NAV_CMD()`

When `verify_command()` returns true:

1. `AP_Mission` looks at the next command index.
2. If it's a **DO** command, it executes it immediately and increments the index *again*.
3. It repeats this until it finds the next **NAV** command or a **COND** command.
4. This effectively "flushes" all instant actions between two waypoints.

3. JUMPS AND LOOPS

The `MAV_CMD_DO_JUMP` command modifies the command index directly.

- `AP_Mission` maintains a `_jump_tracking` array to remember how many times a loop has been repeated.
- **Safety:** The array size is limited (`AP_MISSION_MAX_NUM_DO_JUMP_COMMANDS`), preventing infinite recursion memory leaks.

SOURCE CODE REFERENCE

- **State Logic:** `libraries/AP_Mission/AP_Mission.cpp`
- **Command Handling:** `libraries/AP_Mission/AP_Mission_Commands.cpp`

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





PRACTICAL GUIDE: DEBUGGING MISSION STALLS

1. "THE DRONE WON'T MOVE ON"

If the drone reaches a waypoint but just sits there:

- **Check Acceptance Radius:** Is `WP_RADIUS` too small? If GPS drift > Radius, it never "hits" the target.
- **Check Altitude:** Is it trying to reach a specific altitude but maxed out on throttle?
- **Check Conditionals:** Did you put a `CONDITION_YAW` that requires it to face North, but the yaw PID is loose?

2. "THE CAMERA DIDN'T TRIGGER"

If `DO_DIGICAM_CONTROL` was skipped:

- Ensure it wasn't placed *after* a `NAV_LAND` or `NAV_RTL`. Once the mission enters a "Terminal" state, subsequent commands are often ignored.
- Place DO commands *before* the NAV command for the leg you want them active on.

3. INFINITE LOOPS

- **Symptom:** Drone repeats the same 3 waypoints forever.
- **Fix:** Check the `MAV_CMD_DO_JUMP` "repeat" parameter. `-1` or `0` often means infinite.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





CHAPTER 12: CHIBIOS INTEGRATION

CHIBIOS & ARDUPILOT ARCHITECTURE

EXECUTIVE SUMMARY

Since 2017, ArduPilot has run on top of **ChibiOS/RT**, a high-performance Real-Time Operating System (RTOS). This replaced the older "Flymaple" and custom scheduler approach for STM32 targets.

- **ChibiOS/RT:** Provides the kernel (threading, mutexes, semaphores).
- **ChibiOS/HAL:** Provides drivers for on-chip peripherals (UART, SPI, I2C, ADC).
- **AP_HAL_ChibiOS:** The "Glue Layer" that maps ArduPilot's generic `AP_HAL` classes to the specific ChibiOS implementation.

THEORY & CONCEPTS

1. THE KERNEL (CHIBIOS/RT)

ChibiOS/RT is a static, preemptive real-time kernel.

- **Static:** All kernel objects (threads, semaphores) are typically allocated at compile time (or from a static heap), avoiding dynamic allocation fragmentation.
- **Preemptive:** Higher priority threads *immediately* interrupt lower priority ones. This is critical for flight control where the gyro loop (Fast Loop) must run at precisely 400Hz/1kHz/4kHz regardless of what the logging thread is doing.

2. THE HAL (HARDWARE ABSTRACTION LAYER)

ChibiOS provides a standardized API for accessing hardware.

- Instead of writing registers directly (e.g., `UART1->DR = 'A'`), we use `sdPut(&SD1, 'A')`.
- This makes porting between STM32F4, F7, and H7 seamless, as ChibiOS handles the register differences.

CODEBASE INVESTIGATION

1. THE ENTRY POINT: `MAIN_LOOP()`

Located in `libraries/AP_HAL_ChibiOS/HAL_ChibiOS_Class.cpp`.
When the board boots:

1. **Kernel Init:** `chSysInit()` is called (typically in the startup code).
2. **HAL Init:** `halInit()` sets up drivers defined in `mcuconf.h`.
3. **Daemon Task:** The `main` function becomes the **Main Thread** (`daemon_task`).
4. **Priority Boost:** `chThdSetPriority(APM_MAIN_PRIORITY)` sets the main loop priority (typically very high).

2. THE SCHEDULER WRAPPER

ArduPilot's `AP_Scheduler` is *cooperative*, but it runs *inside* the preemptive ChibiOS main thread.

- **The Glue:** `libraries/AP_HAL_ChibiOS/Scheduler.cpp`
- The `delay_microseconds(n)` function yields the CPU to lower-priority threads (like Storage or USB) while waiting.

3. PRIORITY MAP

ArduPilot maps its abstract priorities to ChibiOS levels:

- `APM_MAIN_PRIORITY`: High (Flight Code).
- `APM_SPI_PRIORITY`: Higher (Bus Drivers).

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- `APM_IO_PRIORITY`: Low (Logging, SD Card).

SOURCE CODE REFERENCE

- **Integration Layer:** `libraries/AP_HAL_ChibiOS/HAL_ChibiOS_Class.cpp`
- **ChibiOS Upstream:** `modules/ChibiOS` (submodule in ArduPilot).

PRACTICAL GUIDE: DEBUGGING HARDFAULTS

If the OS crashes, ChibiOS provides a `HardFault_Handler` in `system.cpp`.

- **Common Cause:** Stack Overflow. Check `AP_HAL::util()->available_memory()`.
- **Diagnosis:** Connect a JTAG/SWD debugger (ST-Link). The handler saves the CPU registers (PC, LR) to `hal.util->persistent_data`, which allows post-mortem analysis even after a watchdog reset.

THREADING & SCHEDULING

EXECUTIVE SUMMARY

While the ArduPilot *Flight Code* (Copter/Plane) is largely single-threaded (the "Main Loop"), the underlying system is highly multi-threaded. `AP_HAL_ChibiOS` creates dedicated threads for I/O, storage, and RC handling to ensure that slow operations (like writing to an SD card) never block the flight stabilization loop.

THEORY & CONCEPTS

1. STATIC THREAD ALLOCATION

In embedded safety-critical systems, dynamic thread creation is risky (heap fragmentation). ChibiOS allows **Static Allocation**:

- **Working Area:** A statically defined array (e.g., `static THD_WORKING_AREA(_io_thread_wa, 2048);`) acts as the stack for the thread.
- **Benefits:** We know exactly how much RAM is used at compile time.

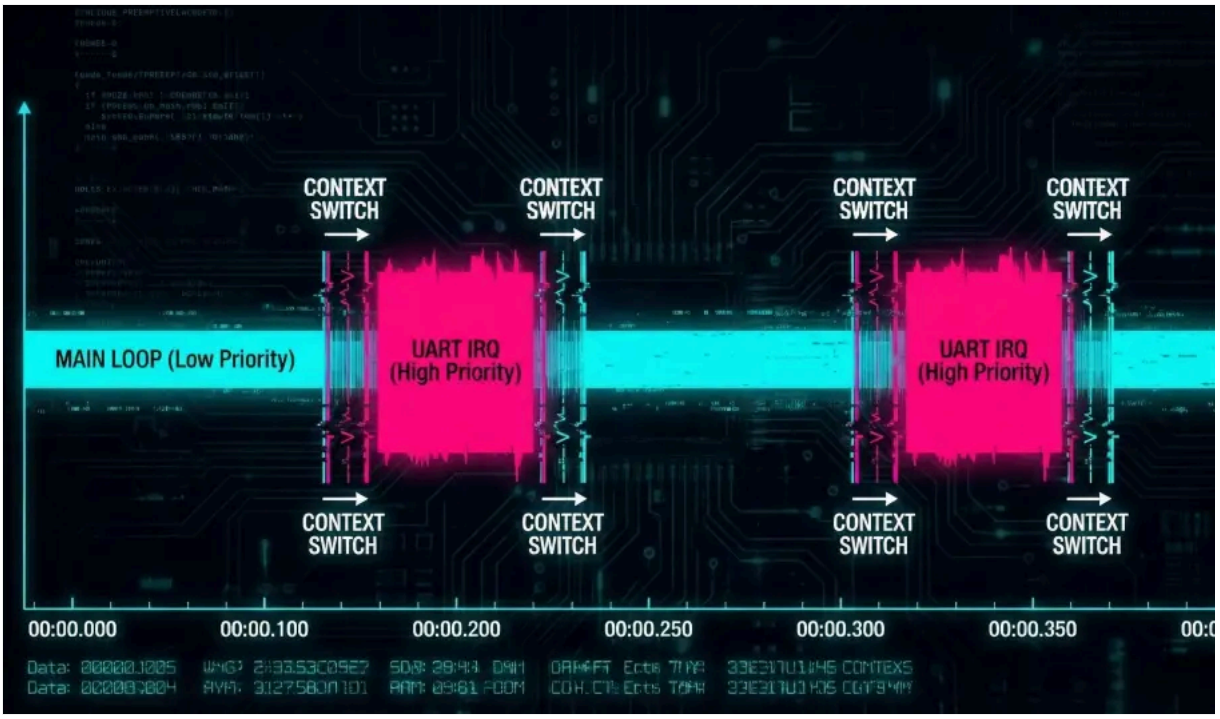
2. COOPERATIVE VS. PREEMPTIVE

- **ArduPilot Main Loop:** Runs in the `main` thread. It is *cooperative* internally (tasks must yield), but the thread itself is *preemptable* by the OS.
- **Driver Threads:** Run asynchronously. For example, the UART driver receives bytes via DMA and signals a semaphore to wake up the I/O thread.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY







CODEBASE INVESTIGATION

1. THREAD CREATION: `SCHEDULER::INIT()`

Located in `libraries/AP_HAL_ChibiOS/Scheduler.cpp`.
ArduPilot spawns the following core threads:

1. `_timer_thread`: Runs 1kHz periodic tasks.
2. `_rcin_thread`: Decodes PPM/SBUS signals.
3. `_io_thread`: Low-priority background tasks (logging, files).
4. `_storage_thread`: Handles EEPROM/FRAM emulation on flash.
5. `_monitor_thread`: Watchdog that checks for thread lockups.

2. THE YIELD: `DELAY_MICROSECONDS()`

When the main loop calls `hal.scheduler->delay(1)`, it calls `chThdSleep(ticks)`.

- This moves the Main Thread to the **SLEEPING** state.
- The ChibiOS scheduler immediately context-switches to the next highest priority READY thread (e.g., UART processing).

3. PRIORITY HIERARCHY

Priorities are defined in `AP_HAL_ChibiOS.h` (mapped to ChibiOS levels):

1. **SPI/I2C High Priority:** Drivers that need instant bus access.
2. **Main Thread (Flight Loop):** Above normal I/O.
3. **Timer Thread:** 1kHz ticks.
4. **RC Input:** Needs low latency capture.
5. **IO / Storage:** Lowest priority. Can take milliseconds to run without affecting flight.

SOURCE CODE REFERENCE

- **Thread Spawning:** `libraries/AP_HAL_ChibiOS/Scheduler.cpp`
- **Stack Definitions:** Look for `THD_WORKING_AREA` macros.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

PRACTICAL GUIDE: ANALYZING STACK USAGE

Running out of stack causes a **HardFault**.

- **Check:** Set `HAL_CHIBIOS_ENABLE_STACK_CHECK` to 1 in `hwdef.dat` (dev boards only).
- **Monitor:** `Scheduler::check_stack_free()` fills the stack with a pattern (0x55) and counts unused bytes.
- **Output:** 'Stack Low' messages in the console/logs if a thread gets dangerously close to its limit.

HAL & HARDWARE DRIVERS

EXECUTIVE SUMMARY

The power of `AP_HAL_ChibiOS` lies in its ability to abstract complex STM32 peripherals into standard interfaces. The most complex and critical part of this is the **Shared DMA** system, which allows ArduPilot to use more peripherals than there are DMA channels available.

THEORY & CONCEPTS

1. THE CHIBIOS HAL DRIVER MODEL

ChibiOS provides "High Level Drivers" (e.g., `SerialDriver`, `SPIDriver`) that abstract the register interface.

- **Config:** Drivers are configured with `Config` structs (baud rate, start bits) passed to `sdStart()`.
- **Events:** Drivers trigger callbacks or wake threads on interrupts (TX Complete, RX Not Empty).

2. DMA CONTENTION

STM32 chips have limited DMA Streams (e.g., 2 controllers x 8 streams = 16 streams).

- **The Problem:** We might have 10 UARTs, 3 SPIs, 2 I2Cs, and SD Card. That's > 16 potential users.
- **The Solution:** `Shared_DMA`. A custom ArduPilot allocator that assigns a DMA channel to a driver *only while a transaction is active* and releases it immediately after.

3. BOUNCE BUFFERS

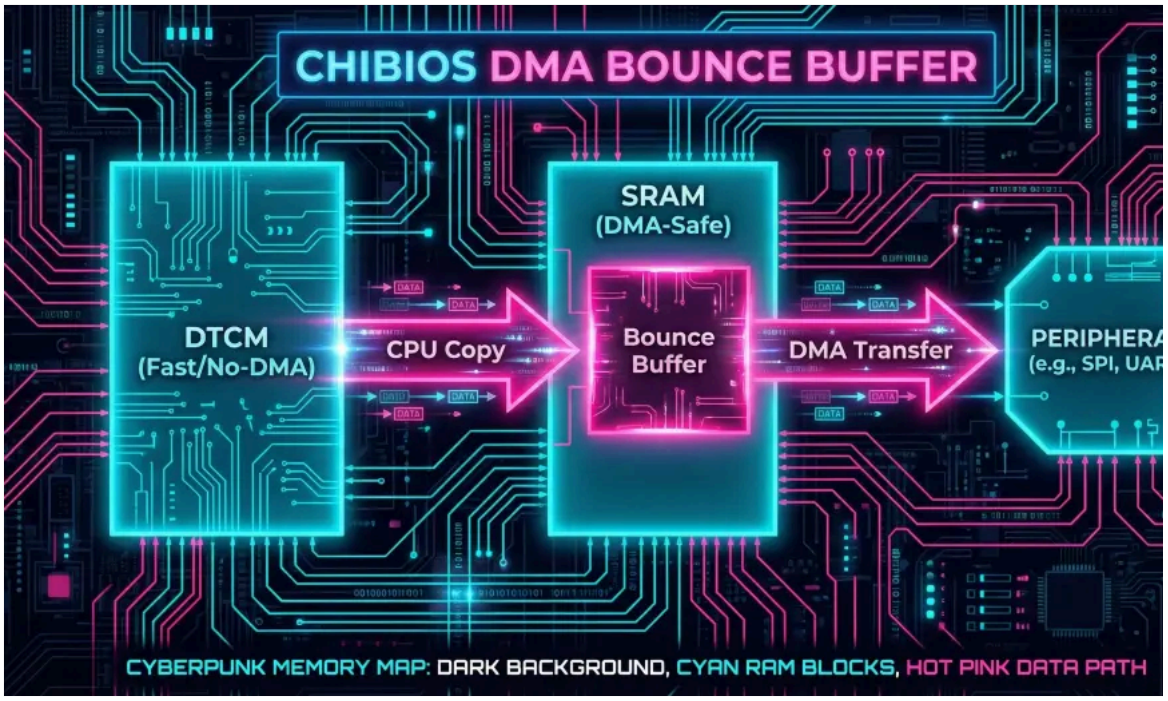
DMA engines access RAM directly. If that RAM is in a specific region (like CCM RAM on F4) or cached (D-Cache on F7/H7), DMA fails.

- **Solution:** We use "Bounce Buffers" in DMA-safe memory regions (SRAM1/2).

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY







- **Flow:** User Buffer → memcpy → Bounce Buffer → DMA → Peripheral.

CODEBASE INVESTIGATION

1. UART DRIVER: `UARTDRIVER.CPP`

Located in `libraries/AP_HAL_ChibiOS/UARTDriver.cpp`.

- **Initialization:** Calls `sdStart(&SDx, &config)`.
- **RX:** Uses a dedicated thread (`uart_rx_thread`) or 1kHz timer (`_rx_timer_tick`) to move data from the ChibiOS ring buffer to the ArduPilot ring buffer.
- **DMA:** `dmaStreamAllocI` requests a stream.

2. SHARED DMA: `SHARED_DMA.CPP`

This file implements a mutex-protected allocator.

- **Locking:** Before starting an SPI transaction, the bus driver calls `dma.lock()`.
- **Contention:** If the DMA stream is busy (used by another peripheral on the same stream ID), it blocks until available.

3. SPI DEVICE: `SPIDEVICE.CPP`

Handles the complexity of the "Device" abstraction (Sensors on a Bus).

- **Transactions:** `get_semaphore()->take()` ensures exclusive access to the bus (e.g., MPU6000 and Barometer on the same SPI bus).
- **CS Pin:** Manually toggles the Chip Select pin via `palClearLine(dev->cs_pin)`.

SOURCE CODE REFERENCE

- **UART:** `libraries/AP_HAL_ChibiOS/UARTDriver.cpp`
- **Shared DMA:** `libraries/AP_HAL_ChibiOS/shared_dma.cpp`

PRACTICAL GUIDE: PORTING DRIVERS

If you are adding a new sensor:

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

1. **Don't** write raw registers.
2. **Use** `AP_HAL::get_HAL().spi->device(...)` to get a handle.
3. **Respect** the semaphore. Always wrap transfers in `bus->get_semaphore()->take()`.

BOARD CONFIGURATION (HWDEF)

EXECUTIVE SUMMARY

Unlike traditional ChibiOS applications that manually edit `board.h` and `mcuconf.h`, ArduPilot uses a Python-based generator: `chibios_hwdef.py`. This script reads a high-level description file (`hwdef.dat`) and auto-generates the low-level C headers required by ChibiOS.

This system allows ArduPilot to support hundreds of boards with a single codebase, simply by defining the pinout in a text file.

THEORY & CONCEPTS

1. THE `HWDEF.DAT` FORMAT

The hardware definition file maps STM32 pins to ArduPilot functions.

- **Format:** `PinName Function Label [Flags]`
- **Example:** `PA9 USART1_TX TELEM1`
  - **Pin:** `PA9`
  - **Function:** `USART1_TX` (The script looks up the Alternate Function index automatically).
  - **Label:** `TELEM1` (Used in code as `HAL_GPIO_PIN_TELEM1`).

2. AUTO-GENERATION PIPELINE

1. **Input:** `libraries/AP_HAL_ChibiOS/hwdef/CubeOrange/hwdef.dat`
2. **Processor:** `chibios_hwdef.py`
3. **Outputs (in build directory):**
   - `board.h`: ChibiOS pin mode definitions (`PAL_MODE_ALTERNATE(7)`).
   - `mcuconf.h`: Enables/Disables peripherals (`#define STM32_SERIAL_USE_USART1 TRUE`).
   - `hwdef.h`: ArduPilot-specific macros (`HAL_SERIAL0_CONFIG`).

CODEBASE INVESTIGATION

1. THE PARSER: `CHIBIOS_HWDEF.PY`

Located in `libraries/AP_HAL_ChibiOS/hwdef/scripts/`.

- `ChibiOSHWDef` **Class:** Stores the pin map.
- `write_UART_config()`**:** Iterates through `SERIAL_ORDER` and generates the `HAL_SERIALx_CONFIG` structs used by `UARTDriver.cpp`.
- **Automatic DMA:** The script calculates DMA stream/channel assignments to minimize contention. If two enabled drivers use the same stream, it errors out or tries to find an alternative stream.

2. DRIVER ACTIVATION

If you define `PA9 USART1_TX` in `hwdef.dat`, the script automatically:

1. Sets `STM32_SERIAL_USE_USART1` to `TRUE` in `mcuconf.h`.
2. Allocates RX/TX DMA streams.
3. Generates the GPIO init code in `board.h`.

SOURCE CODE REFERENCE

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- **Generator Script:** `libraries/AP_HAL_ChibiOS/hwdef/scripts/chibios_hwdef.py`
- **Example hwdef:** `libraries/AP_HAL_ChibiOS/hwdef/CubeOrange/hwdef.dat`

PRACTICAL GUIDE: ADDING A CUSTOM BOARD

To support a new flight controller:

1. **Copy** an existing directory (e.g., `MatekH743`) to a new name.
2. **Edit** `hwdef.dat`:
   - Update `MCU`, `FLASH_SIZE_KB`.
   - Map pins for UARTs, SPI, I2C.
   - Define `SERIAL_ORDER` (UART1, UART2, ...).
3. **Build:** `./waf configure --board MyNewBoard`
   - *Tip:* If the build fails with "DMA Contention", try swapping UARTs or disabling DMA on slow ports (`NODMA`).

DEBUGGING & FAULT ANALYSIS

EXECUTIVE SUMMARY

When an STM32 crashes, it triggers a hardware exception (HardFault, MemManage, BusFault). Unlike a desktop OS that segfaults and keeps running, an embedded controller must either reboot immediately (Watchdog) or freeze for inspection.

ArduPilot on ChibiOS implements a robust fault capture system that saves the CPU state to persistent storage *before* rebooting, allowing post-mortem debugging without JTAG.

THEORY & CONCEPTS

1. THE HARDFAULT EXCEPTION

The ARM Cortex-M core jumps to the `HardFault_Handler` vector when an illegal operation occurs (e.g., accessing invalid memory, dividing by zero, executing a non-thumb instruction).

- **The Stack Frame:** The CPU automatically pushes `R0-R3`, `R12`, `LR`, `PC`, and `xPSR` to the stack.
- **Analysis:** By reading the Program Counter (`PC`) from the stack, we know exactly *where* the crash happened.

2. THE WATCHDOG

ArduPilot uses the STM32 Independent Watchdog (IWDG).

- **Timeout:** Typically 2 seconds.
- **The Pat:** `Scheduler::watchdog_pat()` resets the timer.
- **Thread Lockup:** If the main loop freezes (e.g., infinite loop), the pat stops, and the IWDG resets the chip.

CODEBASE INVESTIGATION

1. THE FAULT HANDLER: `HARDFAULT_HANDLER`

Located in `libraries/AP_HAL_ChibiOS/system.cpp`.

- It copies the stack frame to `struct port_extctx`.
- It calls `save_fault_watchdog()`, which writes the registers to `hal.util->persistent_data` (Backup SRAM or RTC registers).
- On the next boot, this data is logged to the SD card (`INTERNAL_ERROR` messages).

2. THE MONITOR THREAD: `_MONITOR_THREAD`

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





Located in `libraries/AP_HAL_ChibiOS/Scheduler.cpp`.

- A dedicated high-priority thread that wakes up every 10ms.
- **Deadlock Detection:** It checks if the main loop hasn't run for > 200ms.
- **Mitigation:** If it detects a semaphore deadlock, it attempts to "Force Release" the mutex (`try_force_mutex()`) to unstick the system before the watchdog bites.

SOURCE CODE REFERENCE

- **System Faults:** `libraries/AP_HAL_ChibiOS/system.cpp`
- **Watchdog/Monitor:** `libraries/AP_HAL_ChibiOS/Scheduler.cpp`

PRACTICAL GUIDE: ANALYZING A CRASH

1. "INTERNAL ERROR" MESSAGE

If your HUD says `Internal Error 0x...` or the logs show `WDOG` (Watchdog) events:

- **Get the Log:** Download the `.bin` DataFlash log.
- **Look for `MSG`:** It might print "HardFault at 0x0804..."
- **Addr2Line:** Run `arm-none-eabi-addr2line -e arducopter.elf 0x0804...` to find the exact C++ line number.

2. USING CRASHCATCHER

For deep debugging, enable `AP_CRASHDUMP_ENABLED` in `hwdef.dat`.

- **Behavior:** Instead of rebooting, the board freezes and blinks the LED.
- **Recovery:** Use `Tools/scripts/CrashCatcher.py` with a UART adapter to dump the *entire RAM* state for analysis in GDB.

3. STACK OVERFLOW

If the system reboots randomly during complex missions:

- **Check:** `SYS_STACK_FREE` in the logs.
- **Fix:** Increase stack sizes in `Scheduler.cpp` (`THD_WORKING_AREA`) or `hwdef.dat` (`PROCESS_STACK`).

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





CHAPTER 13: OBJECT AVOIDANCE

AVOIDANCE ARCHITECTURE & DATA FUSION

EXECUTIVE SUMMARY

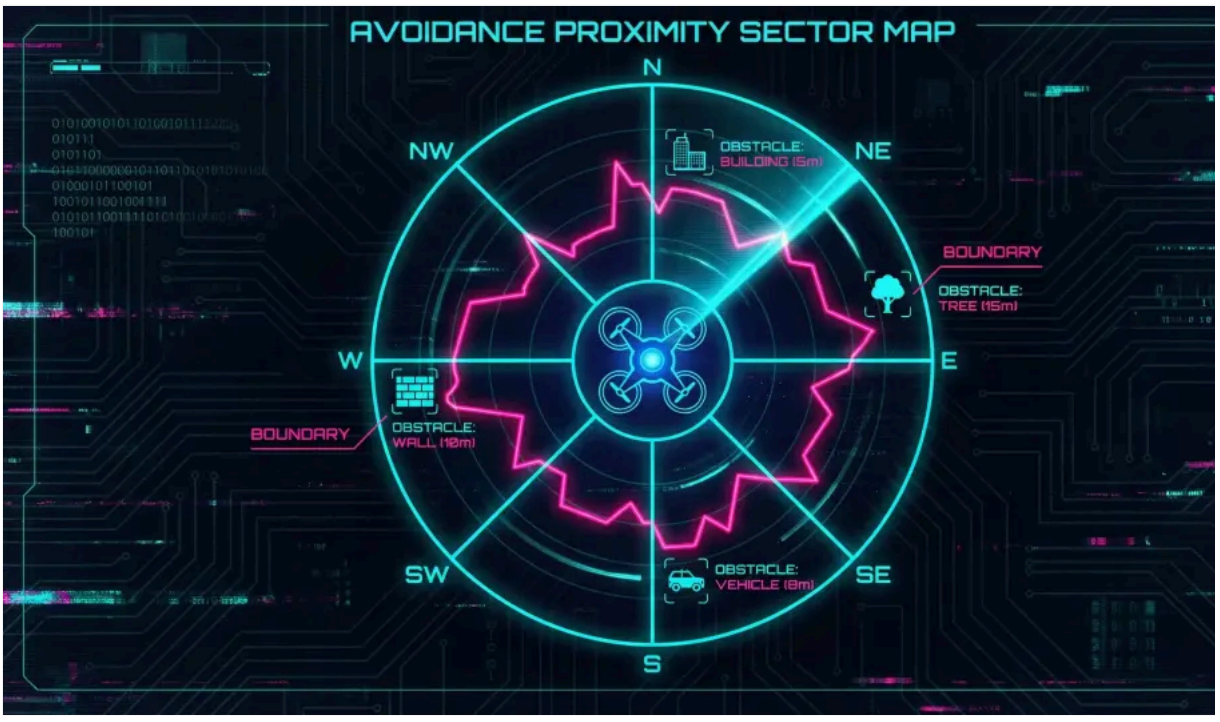
ArduPilot's avoidance system is a multi-layered stack that ingests raw sensor data (Lidar, Radar, Depth Cameras), fuses it into a unified "Boundary," and then modifies the vehicle's desired velocity to prevent collisions.

The architecture separates the **Sensor Driver** (AP_Proximity) from the **Control Logic** (AC_Avoidance).

THEORY & CONCEPTS

1. THE SENSOR LAYER (`AP_PROXIMITY`)

This library manages the hardware drivers. It supports 360-degree Lidars (RPLidar, SF40C) and 1D Rangefinders arranged in a ring.



- **The Boundary:** Sensors populate a "3D Boundary," which is essentially a low-resolution polar grid (sectors) storing the closest obstacle in that direction.
- **Fusion:** If multiple sensors see an object in the same sector, the closest distance is used (Safety Conservative).

2. THE CONTROL LAYER (`AC_AVOID`)

This library sits between the Pilot/Auto-Mission and the Position Controller.

- **Input:** Desired Velocity Vector (V_{des}).
- **Logic:** Checks if V_{des} intersects with the Boundary.
- **Output:** Modified Velocity Vector (V_{safe}).

CODEBASE INVESTIGATION

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

1. PROXIMITY BACKEND: `AP_PROXIMITY_BACKEND`

Located in `libraries/AP_Proximity/AP_Proximity_Backend.cpp`.

- Drivers call `update()` to fetch data from UART/CAN.
- They push data into the boundary using `frontend.boundary.set_face_distance()`.

2. THE ADJUST VELOCITY LOOP: `AC_AVOID::ADJUST_VELOCITY()`

Located in `libraries/AC_Avoidance/AC_Avoid.cpp`.

- This is the core function called by `ModeLoiter`, `ModeAuto`, etc.
- **Backing Up:** It calculates if the vehicle is *inside* the safety margin (`margin_cm`) and generates a "Back Away" velocity vector.
- **Limiting:** If the vehicle is approaching an obstacle, it clamps the velocity component parallel to the obstacle vector.

SOURCE CODE REFERENCE

- **Proximity Core:** `libraries/AP_Proximity/AP_Proximity.cpp`
- **Avoidance Core:** `libraries/AC_Avoidance/AC_Avoid.cpp`

PRACTICAL GUIDE: DEBUGGING PROXIMITY DATA

If avoidance isn't working, first check if the *data* is valid.

1. **Open MAVLink Inspector:** Look for `DISTANCE_SENSOR` messages.
2. **Check `PRX_LOG_RAW`:** Set this to 1 to log raw sensor values to the SD card.
3. **The "Fence" Display:** In Mission Planner/QGC, ensure the "Proximity" overlay is enabled. You should see a red boundary line appear around the drone when obstacles are detected.

HOW TO: SETUP A 360° LIDAR (RPLIDAR)

360-degree Lidars are the "gold standard" for omnidirectional awareness. Here is how to integrate a common unit like the RPLidar A2/A3.

1. PHYSICAL CONNECTION

- **Power:** Most Lidars require a dedicated 5V BEC (they draw too much current for the Flight Controller's internal regulator).
- **Data:** Connect TX (Lidar) to RX (FC) and RX (Lidar) to TX (FC) on a spare Serial port (e.g., `SERIAL4`).

2. CONFIGURATION

Set the following parameters in Mission Planner:

PARAMETER	VALUE	DESCRIPTION
`SERIAL4_PROTOCOL`	`11`	Set protocol to "Lidar360".
`SERIAL4_BAUD`	`115`	RPLidar typically runs at 115200 baud.
`PRX1_TYPE`	`4`	Select "RPLidar" driver.
PRX_ORIENT	`0`	Default orientation (0 = Forward). Adjust if you mounted it rotated.

3. VERIFICATION

1. Reboot the flight controller.
2. Open **Mission Planner > Data Screen > Proximity**.
3. You should see a "radar" display showing points around the vehicle.
4. Walk around the drone; the points should track your movement. If the points move *with* the drone when you rotate it, your compass or `PRX_ORIENT` is incorrect.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





SIMPLE AVOIDANCE (STOP & SLIDE)

EXECUTIVE SUMMARY

Simple Avoidance is the most basic form of collision prevention, used primarily in pilot-controlled modes (Loiter, AltHold, PosHold). It does not path-plan; it simply modifies the pilot's input to prevent the drone from touching a predefined boundary (fence or proximity sensor).

THEORY & CONCEPTS

1. THE PHYSICS OF "STOP"

When AVOID_BEHAVE is set to **Stop**:

- The controller projects the vehicle's stopping position based on current velocity and max deceleration (AVOID_ACCEL_MAX).
- If the stopping point breaches the AVOID_MARGIN, the controller overwrites the pilot's input velocity to 0 in that direction.

2. THE PHYSICS OF "SLIDE"

When `AVOID_BEHAVE` is set to **Slide**:

- Instead of stopping, the velocity vector is "deflected" along the tangent of the obstacle boundary.
- **Feel:** It feels like the drone is pressing against a glass wall. You can push it sideways along the wall, but not through it.

CODEBASE INVESTIGATION

1. THE STOP LOGIC: `AC_AVOID::LIMIT_VELOCITY_2D()`

Located in `libraries/AC_Avoidance/AC_Avoid.cpp`.

- It calculates the component of velocity *towards* the obstacle.
- If that component exceeds the safe speed for the given distance, it is reduced.

```
const float max_speed = get_max_speed(kP, accel_cmss, limit_distance_cm, dt);
```

    - `get_max_speed` uses a square-root controller ($sqrt2ax$) to determine the maximum speed that allows stopping within `limit_distance_cm`.

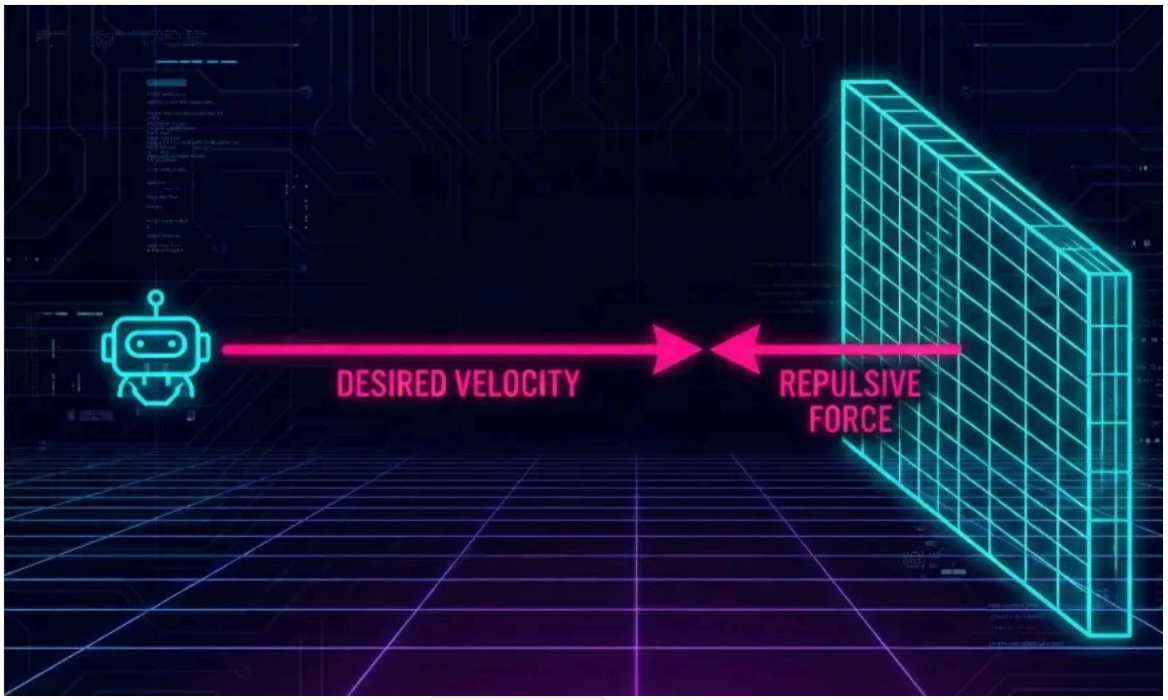
2. THE MARGIN CALCULATION

- **Input:** `_margin` (User parameter, e.g., 2m).
- **Logic:** The system treats the obstacle as being `_margin` meters closer than it actually is.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY







- **Breach:** If the vehicle drifts inside the margin, `adjust_velocity` generates a **Repulsive Vector** to back the vehicle out.

SOURCE CODE REFERENCE

- **Velocity Limiting:** `AC_Avoid::limit_velocity_2D`
- **Stopping Distance:** `AC_Avoid::get_stopping_distance`

PRACTICAL GUIDE: TUNING THE FEEL

- **Jerky Stops:** If the drone brakes too hard at the fence, reduce `AVOID_ACCEL_MAX`.
- **Drifting Through:** If the drone slides through the fence at high speed, increase `AVOID_MARGIN`. The loop latency might be too high for the speed.
- **Oscillation:** If it bounces off the invisible wall, increase AVOID_BACKUP_DZ (Deadzone) to add hysteresis to the backing-up logic.

HOW TO: ENABLE "STOP AT FENCE"

This is the most common use case: preventing a drone from flying out of a designated zone.

1. DEFINE THE FENCE

1. In Mission Planner, go to **Plan** screen.
2. Select **Polygon Fence**.
3. Draw a boundary around your safe flight area.
4. Right-click → **GeoFence** → **Upload**.

2. ENABLE FENCING LOGIC

Set these parameters to tell ArduPilot to respect the boundary:

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

PARAMETER	VALUE	DESCRIPTION
`FENCE_ENABLE`	`1`	Enables the Geo-Fence subsystem.
`FENCE_TYPE`	`4`	Select "Polygon". (Add `2` for Circle, `1` for Altitude).
`FENCE_ACTION`	`1`	Typically "RTL" or "Brake" on breach. Avoidance happens *before* this triggers.

3. ENABLE AVOIDANCE LOGIC

Now tell the controller to stop *before* hitting the fence:

PARAMETER	VALUE	DESCRIPTION
`AVOID_ENABLE`	`7`	Enable All (Fence + Proximity).
`AVOID_BEHAVE`	`1`	Set to **Stop**. (0 = Slide).
`AVOID_MARGIN`	`2.0`	Stop 2 meters before the line.

Test It: Fly slowly towards the boundary in Loiter mode. The drone should brake and refuse to fly further, like hitting an invisible wall.

THE BENDYRULER ALGORITHM

EXECUTIVE SUMMARY

BendyRuler is ArduPilot's local path planner for complex environments. Unlike "Simple Avoidance" which just stops or slides, BendyRuler actively searches for an open path around an obstacle.

It works by "feeling" around the vehicle with virtual feelers (rays) to find the direction that offers the best compromise between **Safety** (clearance from obstacles) and **Progress** (heading towards the destination).

THEORY & CONCEPTS

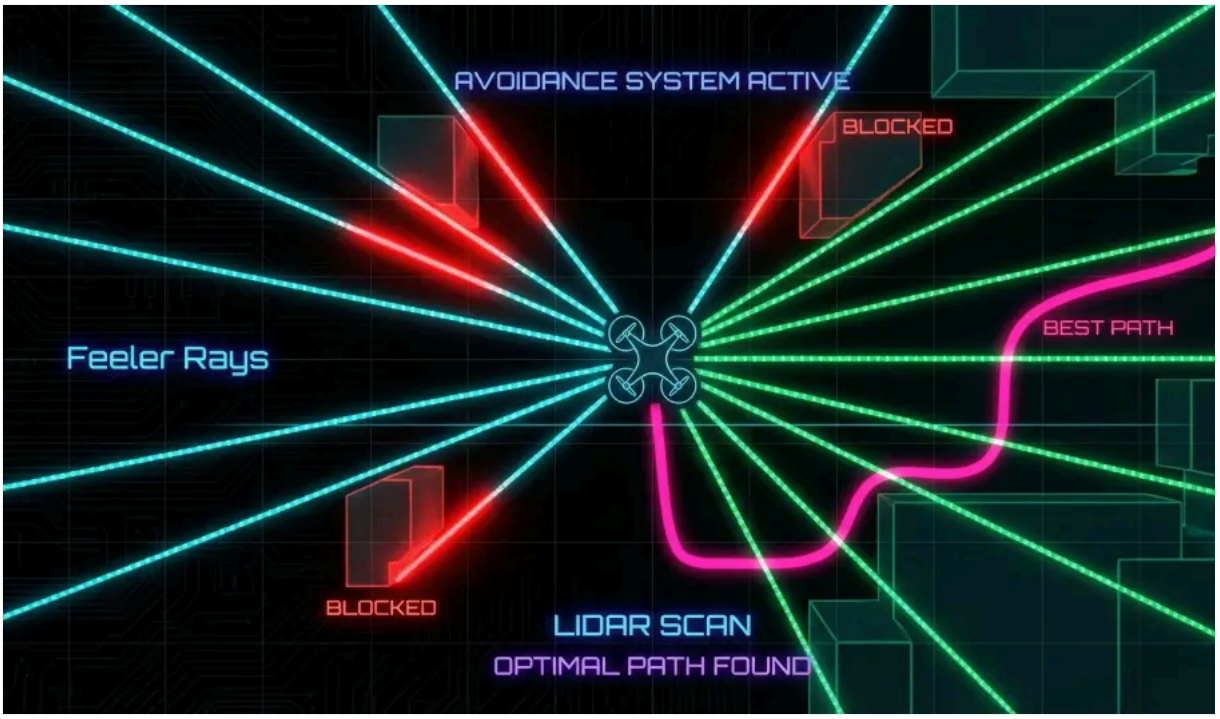
1. THE "FEELER" APPROACH

BendyRuler projects virtual lines (rays) in multiple directions around the vehicle.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY







- **Search Sector:** Typically +/- 45 to 90 degrees from the destination bearing.
- **Ray Length:** Determined by OA_BR_LOOKAHEAD.
- **Evaluation:** For each ray, it checks: "If I fly this way for X meters, will I hit anything?"

2. MARGIN MAXIMIZATION

The algorithm doesn't just look for *any* open path; it calculates a "Margin Score" for every candidate bearing.

- **Margin:** The minimum distance between the ray and the nearest obstacle.
- **Selection:** It chooses the bearing that has a sufficient margin ($>$ OA_MARGIN_MAX) and is closest to the target destination.

3. HYSTERESIS (ANTI-JITTER)

To prevent the drone from indecisively switching between a "Left" path and a "Right" path, BendyRuler implements a resistance mechanism. It will stick to the current evasion path unless a new path offers a significantly better margin (defined by OA_BR_TYPE).

CODEBASE INVESTIGATION

1. THE SEARCH LOOP: `SEARCH_XY_PATH()`

Located in `libraries/AC_Avoidance/AP_OABendyRuler.cpp`.

- Iterates through bearings in 5-degree increments.
- Calls `calc_avoidance_margin()` for each test ray.
- **Two-Stage Lookahead:**
  1. **Step 1:** Check immediate path (short lookahead).
  2. **Step 2:** If Step 1 is clear, check further out (long lookahead) to ensure we aren't flying into a cul-de-sac.

2. MARGIN CALCULATION: `CALC_AVOIDANCE_MARGIN()`

This function checks intersection against:

- Circular Fences.
- Polygon Fences.
- Proximity Database (AP_OADatabase - fused Lidar/Radar points).

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

3. STABILITY LOGIC: `RESIST_BEARING_CHANGE()`

- Input: `_bendy_ratio` (default 1.1).
- Logic: If the new best path is only 10\% better than the current path, ignore it. This dampens the decision logic. See `resist_bearing_change()`.

SOURCE CODE REFERENCE

- **Algorithm Implementation:** `libraries/AC_Avoidance/AP_OABendyRuler.cpp`

PRACTICAL GUIDE: TUNING BENDYRULER

1. "IT STOPS TOO EARLY"

- **Check:** `OA_BR_LOOKAHEAD`.
- **Fix:** If you are flying fast, you need a long lookahead. But if you are indoors, a 10m lookahead might see a wall that is actually far away. Reduce it for tight spaces.

2. "IT WIGGLES"

- **Symptom:** Drone oscillates left/right while facing an obstacle.
- **Fix:** Increase `OA_BR_TYPE` (Margin Ratio). Making it "stickier" (e.g., 1.5) stops it from switching paths constantly.

3. "IT GETS STUCK IN CORNERS"

- **Cause:** BendyRuler is a *local* planner. It doesn't know the maze layout.
- **Fix:** Enable Dijkstra (`OA_TYPE` **= 2**) for global pathfinding if you operate in complex polygon fences.

HOW TO: CONFIGURE BENDYRULER FOR COPTER

BendyRuler is the standard for "Dodge and Continue" missions. Here is the standard config for a 5-inch quadcopter.

1. ENABLE PATH PLANNING

- `OA_TYPE` = `1` (BendyRuler). This replaces the simple "Stop" behavior with "Pathfind".

2. SET PHYSICAL LIMITS

BendyRuler needs to know how agile your drone is.

- `OA_BR_LOOKAHEAD` : `10` (Meters). Look 10m ahead. Increase for fast drones (>15m/s).
- `OA_MARGIN_MAX` : `3` (Meters). The "Search" margin. If a path is closer than 3m to an obstacle, it is penalized.

3. TUNE BEHAVIOR

- `OA_BR_TYPE` : `1` (Horizontal Only). Most Copters only have 2D Lidar. Use `2` for 3D Avoidance if you have a Realsense/Depth Camera.
- **OA_BR_SO_TURN:** `1` (Spin on Spot). If blocked, the drone will stop and yaw to scan for a path before proceeding.

Test It: Set up two cardboard boxes 5m apart. Upload a mission that flies straight through them. The drone should deviate from the straight line, fly between the boxes (or around them), and return to the path.

DIJKSTRA PATH PLANNING

EXECUTIVE SUMMARY

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





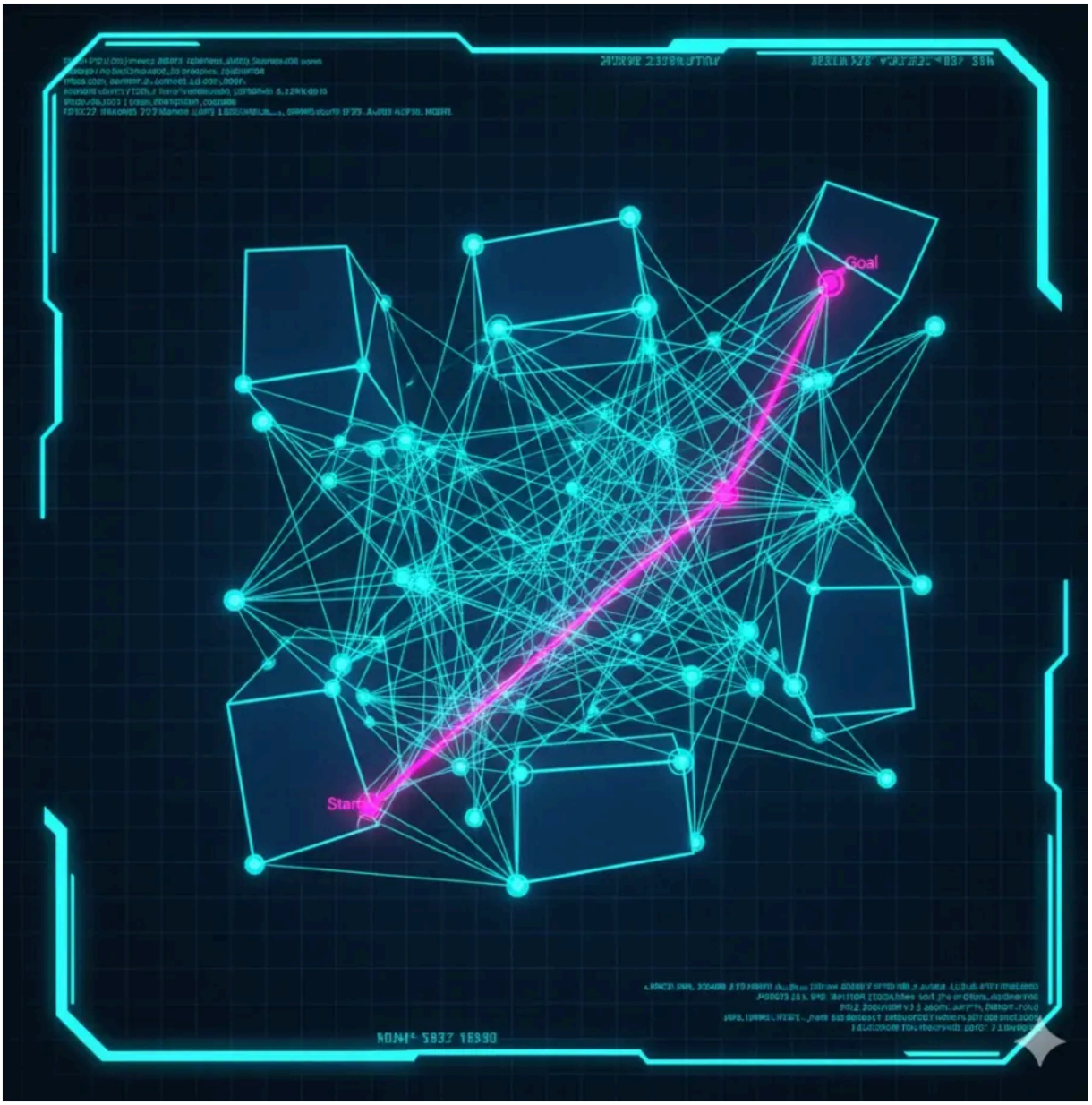
While BendyRuler is great for local, reactive avoidance, it can get stuck in "cul-de-sacs" (U-shaped obstacles). **Dijkstra's Algorithm** solves this by building a global map of the known fence and finding the mathematically optimal path around it.

It is computationally expensive but guarantees a solution if one exists within the fence boundaries.

THEORY & CONCEPTS

1. THE VISIBILITY GRAPH

To use Dijkstra on a continuous 2D plane, ArduPilot discretizes the world into a **Visibility Graph**.



- **Nodes:** The vertices of all Inclusion and Exclusion polygons (plus a safety margin).
- **Edges:** A line connects two nodes if the path between them does not intersect any fence line.
- **Logic:** The drone can fly safely along any edge in the graph.

2. THE SHORTEST PATH

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

Dijkstra's algorithm explores the graph starting from the vehicle's position (`Source`).

1. Assign tentative distance values to every node (0 for Source, Infinity for others).
2. Visit the unvisited node with the smallest tentative distance.
3. Calculate distances to its neighbors.
4. Repeat until the `Destination` node is visited.

CODEBASE INVESTIGATION

1. BUILDING THE GRAPH: `CREATE_FENCE_VISGRAPH()`

Located in `libraries/AC_Avoidance/AP_OADijkstra.cpp` .

- It expands the user-defined fence polygons by OA_MARGIN_MAX.
- It iterates through every pair of points (N^2 complexity) to check for line-of-sight connectivity using `intersects_fence()` .

2. THE SOLVER: `CALC_SHORTEST_PATH()`

Located in `libraries/AC_Avoidance/AP_OADijkstra.cpp` .

- Runs the standard Dijkstra loop.
- **Output:** A list of waypoints (`_shortest_path`) that the vehicle must follow to clear the obstacle.

3. DYNAMIC UPDATES

Unlike a static map, the graph must be rebuilt if the destination changes or if the fence is updated via MAVLink. This is a heavy operation, often causing a momentary loop delay on slower processors (F4).

SOURCE CODE REFERENCE

- **Implementation:** `libraries/AC_Avoidance/AP_OADijkstra.cpp`
- **Header:** `libraries/AC_Avoidance/AP_OADijkstra.h`

PRACTICAL GUIDE: WHEN TO USE DIJKSTRA?

1. COMPLEX GEO-FENCING

If you have a complex Exclusion Zone (e.g., a "Keep Out" zone for a building) and you want the drone to fly *around* it automatically during a mission, Dijkstra is the only option. BendyRuler might just stop at the edge.

2. PERFORMANCE LIMITS

- **Point Limit:** ArduPilot limits the total fence points to ~100 to keep calculation time reasonable.
- **Hardware:** Recommended only for H7 boards (Cube Orange, Matek H743). On F4 boards, expect sluggishness during path recalculation.

3. "DIJKSTRA FAILED" ERRORS

If you see this error:

- **Cause:** The destination is *inside* an exclusion zone or *outside* an inclusion zone.
- **Fix:** Ensure your WP mission is valid relative to the fence *before* uploading. (**`OA_TYPE` = 2** enables Dijkstra).

HOW TO: SETUP COMPLEX GEO-FENCING FOR DIJKSTRA

Dijkstra excels when you have "No Fly Zones" (NFZ) inside your mission area, like a building or a tree cluster.

1. ENABLE DIJKSTRA

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- `OA_TYPE` = `2` (Dijkstra).

2. DRAW THE FENCE

1. Open **Mission Planner > Plan**.
2. **Inclusion Zone:** Draw a large Polygon Fence around the entire flight field. Right-click → GeoFence → **Inclusion**.
3. **Exclusion Zone:** Draw a smaller polygon around the obstacle. Right-click → GeoFence → **Exclusion**.
4. **Upload:** Click "Write GeoFence".

3. PLAN A MISSION

1. Place Waypoint 1 on one side of the Exclusion Zone.
2. Place Waypoint 2 on the *opposite* side.
3. **Do not** place intermediate waypoints around the obstacle; Dijkstra will do that for you.

Result: When you switch to Auto, the drone will fly to WP1, then calculate a path *around* the Exclusion Zone (hugging the edge by `OA_MARGIN_MAX`) to reach WP2. You will see "OA: Path found" message on the HUD.

ADS-B COLLISION AVOIDANCE

EXECUTIVE SUMMARY

ADS-B (Automatic Dependent Surveillance–Broadcast) allows the drone to see manned aircraft. ArduPilot doesn't just display these on the GCS; it can actively dodge them.

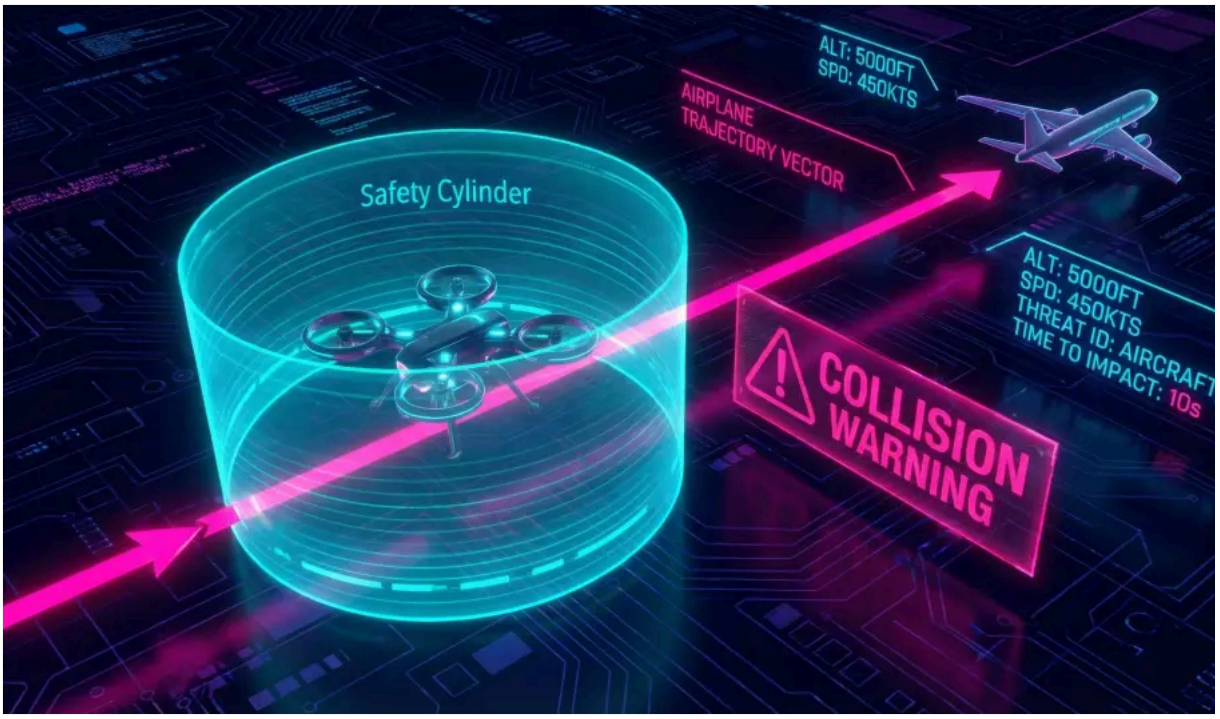
Unlike Lidar/Radar avoidance which is short-range and reactive, ADS-B avoidance is long-range and strategic, triggering specific failsafe flight modes.

THEORY & CONCEPTS

1. THE THREAT CYLINDER

ArduPilot defines a safety cylinder around the vehicle:

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY



- **Radius:** ADSB_LIST_RADIUS (e.g., 2km).
- **Altitude:** ADSB_LIST_ALT (e.g., 1000m).
- **Collision Zone:** If an aircraft is projected to enter a smaller, critical zone within a certain time, an avoidance action is triggered.

2. EVASIVE MANEUVERS

Unlike simple obstacle avoidance (Stop/Slide), ADS-B avoidance triggers a dedicated flight mode (`AVOID_ADSB` for Copter) or overrides the path.

- **Climb/Descend:** The most common reaction.
- **Perpendicular:** Fly 90 degrees away from the threat's velocity vector.

CODEBASE INVESTIGATION

1. THE SENSOR DRIVER: `AP_ADSB::UPDATE()`

Located in `libraries/AP_ADSB/AP_ADSB.cpp`.

- Maintains a list of `adsb_vehicle_t`.
- Handles timeouts (`VEHICLE_TIMEOUT_MS`) to remove stale targets.
- Pushes data to the GCS via `SRx_ADSB`.

2. COPTER AVOIDANCE LOGIC: `AP_AVOIDANCE_COPTER::HANDLE_AVOIDANCE()`

Located in `ArduCopter/avoidance_adsb.cpp`.

- **State Change:** When a threat is confirmed, it switches the flight mode:

```
copter.set_mode(Mode::Number::AVOID_ADSB, ModeReason::AVOIDANCE)
```

- **Vertical Evasion:** Checks if the aircraft is above or below.
  - If above, descend (but not below **ADSB_ALT_MIN**).
  - If below, climb.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

SOURCE CODE REFERENCE

- **ADS-B Core:** `libraries/AP_ADSB/AP_ADSB.cpp`
- **Copter Logic:** `ArduCopter/avoidance_adsb.cpp`

PRACTICAL GUIDE: SETTING UP ADS-B

1. HARDWARE

You need an ADS-B Receiver.

- **uAvionix PingRX:** Standard for small drones.
- **Cube Orange:** Has ADS-B In built into the carrier board (check your version).

2. KEY PARAMETERS

- **ADSB_ENABLE**: 1.
- **ADSB_BEHAVIOR**: Controls the reaction. `0` = None (Report only), `1` = Loiter/Hover, `2` = Avoid (Climb/Descend).
- **ADSB_ALT_MIN**: Critical. Set this to your Minimum Safe Altitude (e.g., 20m). You do NOT want the drone to descend into the trees to avoid a plane at 500ft.

3. TESTING

- **Simulator:** You can simulate ADS-B traffic in SITL using `sim_vehicle.py`.
- **Real World: Do not test this with real planes.** Trust the simulation or use a second drone broadcasting ADS-B Out (if legally permitted).

HOW TO: CONFIGURE ADS-B AVOIDANCE

This is a "Set and Forget" safety feature.

1. ENABLE THE HARDWARE

- **ADSB_ENABLE:** `1`.
- Reboot. Check MAVLink Inspector for `ADSB_VEHICLE` messages (if any traffic is nearby).

2. CONFIGURE THE AVOIDANCE LOGIC

The avoidance logic is separate from the driver.

- `AVD_ENABLE` : `1`. (Enables the avoidance library).
- `AVD_F_DIST_XY` : `1000` (Meters). If a plane gets within 1km horizontally...
- `AVD_F_DIST_Z` : `100` (Meters). ...AND within 100m vertically...
- `AVD_F_TIME` : `10` (Seconds). ...AND will collide within 10 seconds.
- `AVD_F_ACTION` : `1` (Climb). The drone will immediately climb to `AVD_F_ALT_MIN` or maintain altitude if higher.

Verification: In Mission Planner simulation, use the "ADSB Simulation" tab to inject a fake aircraft on a collision course. Your drone should switch to `AVOID_ADSB` mode and climb/descend automatically.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





CHAPTER 14: ADVANCED TUNING

FREQUENCY DOMAIN ANALYSIS

EXECUTIVE SUMMARY

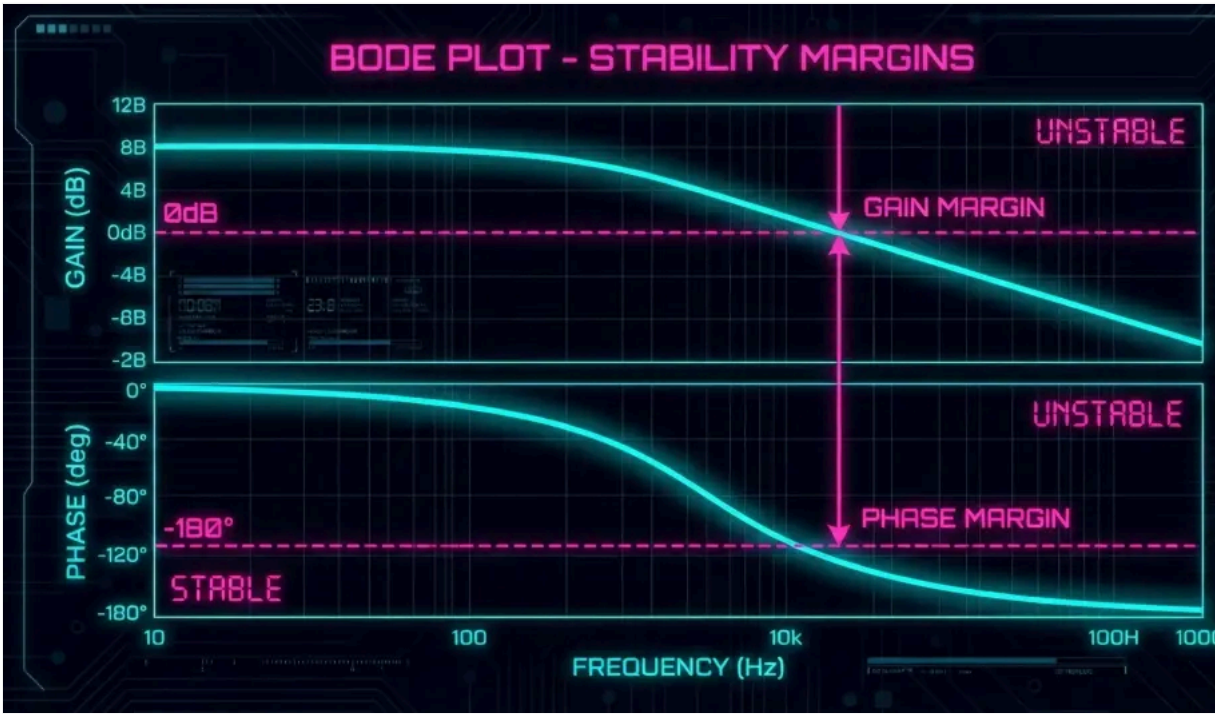
In advanced tuning, we stop thinking of the drone as a time-domain machine (reacting to inputs) and start thinking of it in the **Frequency Domain** (reacting to vibrations and oscillations).

Understanding the sampling rate, the Nyquist limit, and filter cutoffs is essential for dealing with propeller noise and avoiding "D-term heating."

THEORY & CONCEPTS

1. THE NYQUIST LIMIT

If your gyro samples at 1kHz, the maximum frequency it can accurately measure is 500Hz (Nyquist Frequency).



- **Aliasing:** Vibrations *above* 500Hz don't disappear; they fold back into the lower spectrum (e.g., a 600Hz vibe looks like 400Hz).
- **Solution:** We must filter out high-frequency noise *before* it down-samples, or use a very high sampling rate (Fast Sampling).

2. DIGITAL FILTERS (BIQUADS)

ArduPilot uses **2nd Order Biquad Low-Pass Filters** (LowPassFilter2p).

- **Phase Lag:** Every filter introduces delay. A 20Hz cutoff filter has much more lag than a 100Hz filter.
- **The Tradeoff:** Lower cutoff = smoother signal but slower reaction (worse PID performance).

CODEBASE INVESTIGATION

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

1. THE LOOP RATE: `AP_INERTIALSENSOR::INIT()`

Located in `libraries/AP_InertialSensor/AP_InertialSensor.cpp`.

- `_loop_rate` sets the main loop speed (e.g., 400Hz).
- `_sample_period_usec` is derived from this.
- **Fast Sampling:** If `_fast_sampling_mask` is enabled, the backend reads the gyro FIFO at 1kHz-8kHz, applies a hardware low-pass, and then averages down to the loop rate.

2. THE FILTER IMPLEMENTATION: `LOWPASSFILTER2P`

Located in `libraries/Filter/LowPassFilter2p.cpp`.

- It implements a standard Direct Form I or II transposed biquad.
- **Math:**

```
T delay_element_0 = sample - _delay_element_1 * params.a1 - _delay_element_2 * params.a2;
T output = delay_element_0 * params.b0 + _delay_element_1 * params.b1 + _delay_element_2 * params.b2;
```

  This mathematical recurrence relation determines how past inputs affect the current output (Infinite Impulse Response).

SOURCE CODE REFERENCE

- **Inertial Sensor:** `libraries/AP_InertialSensor/AP_InertialSensor.cpp`
- **Filter Logic:** `libraries/Filter/LowPassFilter2p.cpp`

PRACTICAL GUIDE: SETTING FILTER FREQUENCY

1. INS_GYRO_FILTER

- **Standard:** 20Hz for large quads, 40-80Hz for small racers.
- **Tuning:** Raise this value to reduce phase lag (improves flight feel). If motors get hot, lower it.

2. INS_ACCEL_FILTER

- **Standard:** 10-20Hz.
- **Role:** The accelerometer is NOT used in the fast rate loop (PID), only for angle estimation (Level horizon). High filtering here is fine and desirable to prevent horizon drift.

SYSTEM IDENTIFICATION (SYSID)

EXECUTIVE SUMMARY

System Identification (SysID) is an engineering method to build a mathematical model of a physical system by stimulating it with known inputs and measuring the output.

ArduPilot's **System ID Mode** injects a "Chirp" signal (sweeping frequency sine wave) into the control loops. By analyzing the vehicle's response (Gyro/Accel) compared to the input (Chirp), we can generate **Bode Plots** to determine the system's bandwidth, phase margin, and optimal PID gains.

THEORY & CONCEPTS

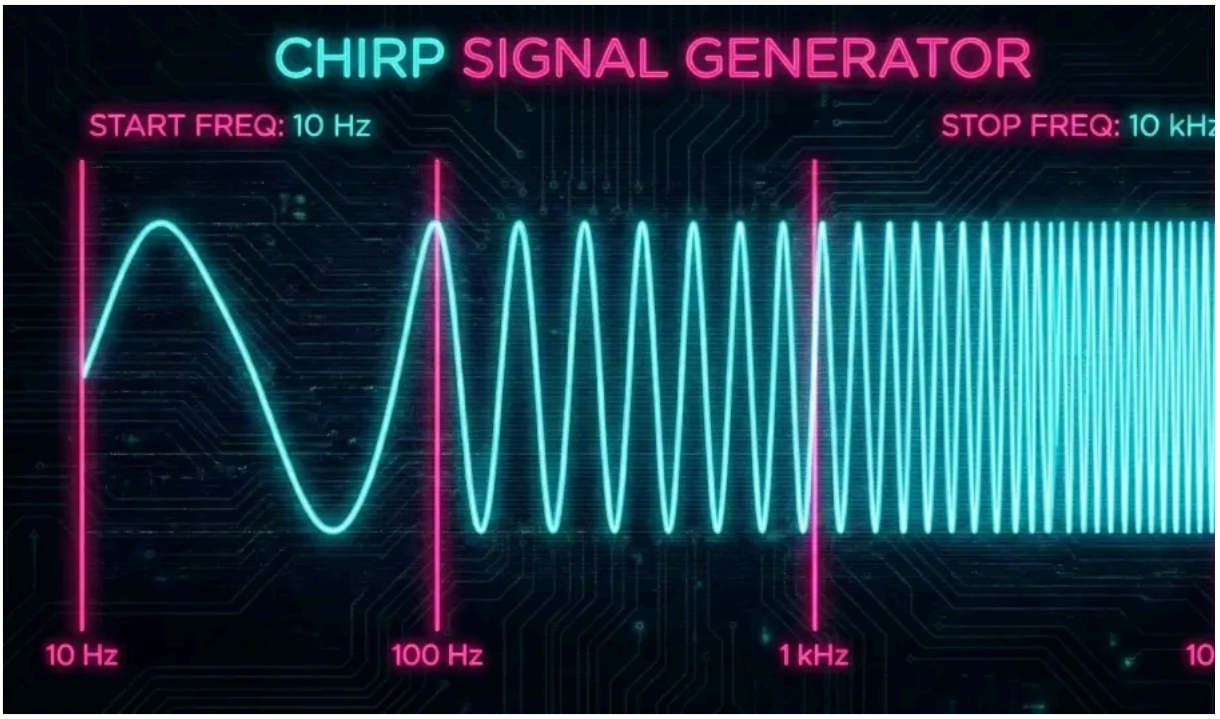
1. THE CHIRP SIGNAL

The "Chirp" is a sinusoidal signal that sweeps from a low frequency to a high frequency over a set duration.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY







- **Start Frequency (f_start):** Low frequency to test steady-state response.
- **Stop Frequency (f_stop):** High frequency (typically 20-40Hz) to test high-speed dynamics and noise rejection.
- **Amplitude:** The magnitude of the disturbance. It must be large enough to overcome friction/stiction but small enough to prevent the drone from flipping over.

2. FREQUENCY RESPONSE ANALYSIS (BODE PLOT)

The data from the Chirp is used to generate a **Bode Plot**, which shows the system's gain and phase lag across the frequency spectrum.

- **Gain Margin:** How much gain can be added before the system becomes unstable.
- **Phase Margin:** How much delay (lag) the system can tolerate.
- **Bandwidth:** The frequency where the system can no longer track the input.

CODEBASE INVESTIGATION

1. THE MODE LOGIC: `MODESYSTEMID::RUN()`

Located in `ArduCopter/mode_systemid.cpp`.

- It generates the waveform:

```
waveform_sample = chirp_input.update(waveform_time - SYSTEM_ID_DELAY, waveform_magnitude);
```

- It adds this sample to the target based on SID_AXIS. For example, if `SID_AXIS = 7` (Rate Roll):

```
attitude_control->rate_bf_roll_sysid(radians(waveform_sample));
```

2. LOGGING

Crucially, SysID logs high-rate data specifically for analysis tools.

- `SIDD` **Log Message:** Contains Time, Input Value, Current Frequency, and Gyro/Accel response.
- `PID` **Logs:** The standard PID logs are also critical for verifying tracking.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

SOURCE CODE REFERENCE

- **Mode Implementation:** `ArduCopter/mode_systemid.cpp`
- **Chirp Generator:** `libraries/AP_Math/control.cpp` (See `Chirp` class).

PRACTICAL GUIDE: RUNNING A SYSID FLIGHT

1. SAFETY FIRST

- **Space:** You need a large, open area. The drone *will* drift during the test.
- **AltHold:** SysID usually runs in a mode similar to Althold. You control throttle; the computer controls attitude (with the chirp overlaid).

2. CONFIGURATION

- `SID_AXIS`: Start with **0** (None) to get to the field. Set to **1** (Input Roll) or **2** (Input Pitch) for initial tuning.
- **SID_MAGNITUDE**: Start small. **5 degrees** for angle, **50** deg/s for rate.
- **SID_F_START** / **SID_F_STOP**: 0.5Hz to 20Hz is standard for large quads.

3. THE PROCEDURE

1. Take off in Loiter/AltHold.
2. Switch to **System ID Mode**.
3. Wait. The drone will pause (Fade In), then start oscillating.
4. **Do not touch the sticks** unless it drifts too far. Correcting the drift ruins the data (adds external inputs).
5. Wait for the chirp to finish (Fade Out).
6. Land and analyze the logs.

TOTAL ENERGY CONTROL SYSTEM (TECS)

1. CORE PHYSICS: ENERGY RATE VS. DISTRIBUTION

The Total Energy Control System (TECS) replaces classical independent altitude/airspeed PID loops with a physics-based MIMO (Multi-Input Multi-Output) controller. It manages the aircraft's energy state rather than just its spatial position.

THE ENERGY STATE

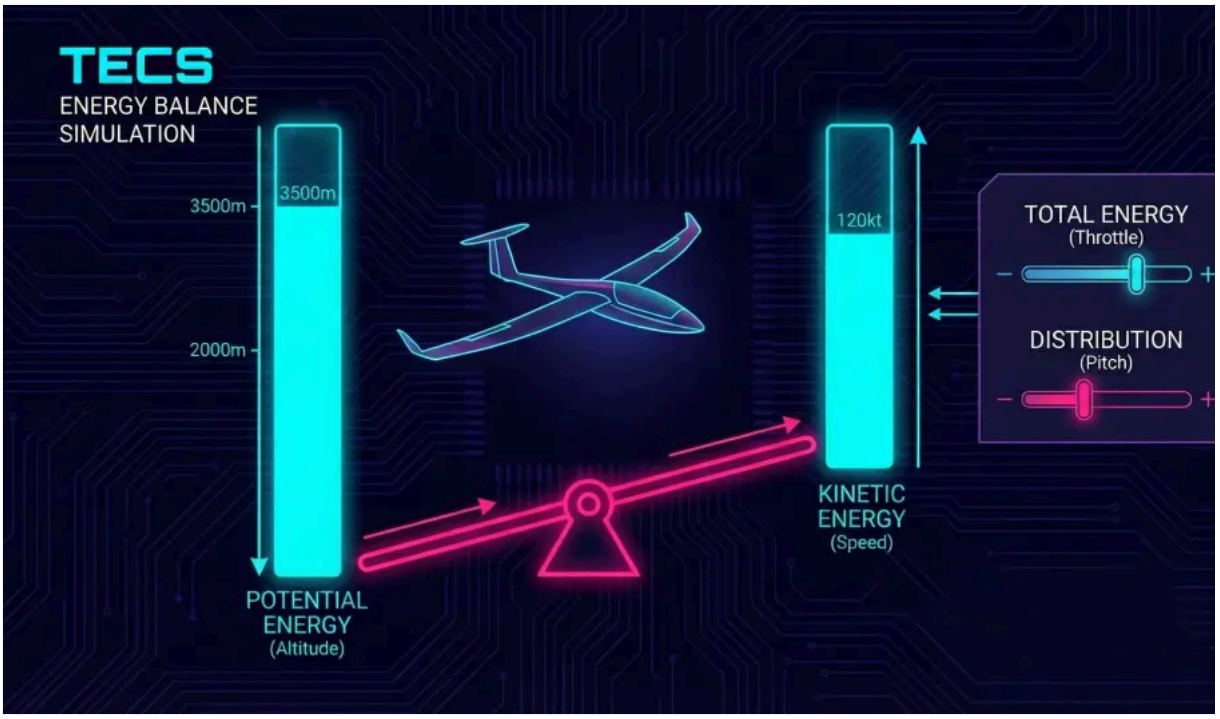
An aircraft has two primary energy reservoirs:

1. **Gravitational Potential Energy (E_P):** Function of Altitude (mgh).
2. **Translational Kinetic Energy (E_K):** Function of Airspeed ($\frac{1}{2}mV^2$).

CONTROL OBJECTIVES

TECS decouples control into two orthogonal loops:

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY



- **Total Energy Rate ($\dot{E}_{total}$):** Controlled by **THROTTLE**.
  - *Logic:* If the plane is both too low (low E_P) and too slow (low E_K), it has a total energy deficit. Only adding thrust can solve this. Elevator input effectively just trades one error for another.
  - *Equation:* $\dot{E}_{total} propto textThrust - textDrag$
- **Energy Distribution Rate ($\dot{E}_{dist}$):** Controlled by **PITCH**.
  - *Logic:* If the plane is high but slow, or low but fast, the Total Energy might be correct, but distributed wrongly. The elevator exchanges Potential for Kinetic energy (or vice versa).
  - *Equation:* $\dot{E}_{dist} propto \dot{h} - \frac{V\dot{V}}{g}$

---

2. ARDUPILOT ARCHITECTURE (AP_TECS)

The `AP_TECS` library sits between the Navigation Controller (`AP_L1_Control`/Mission Planner) and the Attitude Controller (`AP_AttitudeControl` - Roll/Pitch PIDs).

- **Inputs:** Demanded Speed (`spdem`), Demanded Height (`hdem`), State Estimates (EKF).
- **Processing:** Calculates Specific Energy Error and Energy Distribution Error. Applies `TECS_TIME_CONST` filters and `SPDWEIGHT` weighting.
- **Outputs:** Target Pitch (`pitch_dem`) $\rightarrow$ Inner Loop; Target Throttle (`throttle_dem`) $\rightarrow$ ESC.

CRITICAL ARCHITECTURAL CONSTRAINT: TECS assumes the inner pitch loop is perfect. If `ATT.Pitch` lags `ATT.DesPitch`, TECS receives delayed feedback on its energy distribution corrections, leading to divergent oscillation. **Tune Inner Loops First.**

---

3. PRE-TUNING CHECKLIST

DO NOT attempt TECS tuning until these physical baselines are verified.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

1. **Center of Gravity:** Must be mechanically correct. Tail-heavy aircraft require rapid stabilizer corrections that TECS misinterprets as energy distribution volatility.
2. **Airspeed Sensor:**
    - `ARSPD_RATIO` must be calibrated (flight circle/figure-8).
    - Kinetic energy is proportional to V^2; sensor errors are amplified exponentially in the energy calculation.
3. **Inner Loops (Attitude):**
    - Run `AUTOTUNE`.
    - Log Analysis: Verify `ATT.Pitch` tracks `ATT.DesPitch` with minimal latency (<200ms) and no oscillation.

---

4. TUNING PROTOCOL

Tuning is a process of **System Identification**, not random gain adjustment. We fly to measure physical limits, then map those limits to parameters.

PHASE 1: FLIGHT ENVELOPE IDENTIFICATION (MANUAL/FBWA)

Fly in **FBWA** mode to manually probe limits.

MANEUVER	ACTION	OBSERVATION	PARAMETER MAPPING
Cruise Trim	Fly level at cruise speed.	Record Throttle \% and Pitch Angle.	`TRIM_THROTTLE`
Max Climb	Full Throttle (`THR_MAX`), pull back to max safe climb angle.	Record stabilized Climb Rate (m/s).	`TECS_CLMB_MAX` (Set to ~80-90\% of measured)
Min Sink	Idle Throttle (`THR_MIN`), glide level.	Record Sink Rate (m/s).	`TECS_SINK_MIN`
Max Sink	Idle Throttle, pitch down to max safe descent.	Record Sink Rate (m/s).	`TECS_SINK_MAX`

PHASE 2: BENCH CONFIGURATION

Apply observed values to parameters.

- `TECS_CLMB_MAX`: Set to 80-90\% of observed max climb. *Warning: Setting this to 100\% causes integrator windup as the target is mathematically impossible.*
- `AIRSPEED_MIN`: Stall speed + 20\% safety margin.
- `AIRSPEED_MAX`: Slightly below structural V_{ne}.

PHASE 3: VERIFICATION (AUTONOMOUS)

Switch to **LOITER** or **CRUISE**.

1. **Loiter Stability:** Verify Altitude $pm 1m$, Speed $pm 1m/s$.
2. **Climb Step:** Command +50m alt change. Throttle should smoothly surge to `THR_MAX`.
3. **Descent Step:** Command -50m alt change. Throttle should cut to `THR_MIN`; Pitch should drop to maintain airspeed.

---

5. ADVANCED PARAMETER TUNING

Once the envelope is defined, tune the "personality" of the controller. (See Parameter Definitions in source).

5.1 `TECS_SPDWEIGHT` (PRIORITY)

Determines how the controller resolves conflict (e.g., Underspeed AND Low Altitude).

- **Range:** 0.0 - 2.0 (Default: 1.0)

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- **1.0 (Balanced):** Splits error correction between Speed and Height.
- **2.0 (Speed Priority): Mandatory for Gliders.** Sacrifices altitude to maintain airspeed (prevent stall).
- **0.0 (Height Priority):** Sacrifices speed to maintain altitude (Precision Landing).

5.2 `TECS_TIME_CONST` (AGGRESSIVENESS)

Low-pass filter time constant on energy errors.

- **Default:** 5.0
- **Tuning:**
  - *Porpoising (Sinusoidal oscillation):* Increase to 7.0 or 8.0 to dampen response.
  - *Sluggishness:* Decrease to 3.0 or 4.0 for tighter tracking (requires fast inner loops).

5.3 `THR_SLEWRATE` (THROTTLE SMOOTHNESS)

Max \% throttle change per second.

- **Electric:** 100\% (Instant).
- **Gas/Nitro:** 20-30\% to prevent engine choke on rapid transients.
- **Surging:** Reduce to dampen "throttle hunting" in gusty conditions.

5.4 `TECS_PTCH_FF_K` (FEED-FORWARD)

Predicts pitch change needed for a speed change.

- **Value:** -0.04 to -0.08.
- **Use Case:** High-efficiency gliders. improves responsiveness to speed demands without waiting for error integration.

---

6. FORENSICS & LOG ANALYSIS

Diagnosis of common energy pathologies using Dataflash logs.

6.1 PORPOISING (HEIGHT OSCILLATION)

- **Symptom:** Aircraft undulates sinusoidally.
- **Log:** `TECS.h` oscillates around `TECS.hdem`.
- **Root Cause 1:** Inner Loop Lag. Check `ATT.Pitch` vs `ATT.DesPitch`. → Tune Pitch PID.
- **Root Cause 2:** TECS too fast. → Increase `TECS_TIME_CONST` (+1.0 steps).
- **Root Cause 3:** Damping. → Increase `TECS_PTCH_DAMP` (+0.1 steps).

6.2 THROTTLE HUNTING (SURGING)

- **Symptom:** Motors rev up/down in level flight.
- **Log:** `TECS.th` shows sawtooth/square wave.
- **Fix 1:** Decrease `THR_SLEWRATE`.
- **Fix 2:** Increase `TECS_THR_DAMP`.
- **Fix 3:** Verify `TRIM_THROTTLE` matches actual cruise throttle.

6.3 ALTITUDE LOSS IN TURNS

- **Symptom:** Drops altitude when banking.
- **Physics:** Banking diverts lift vector; vertical component decreases.
- **Fix:** Increase `TECS_RLL2THR`. Adds feed-forward throttle based on bank angle to overcome induced drag.

---

7. SPECIAL CONFIGURATIONS

QUADPLANES (VTOL)

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- **Transition:** TECS does not fully own altitude control until transition is complete.
- `Q_ASSIST_SPEED` **:** Set this **below** `AIRSPEED_MIN` to prevent VTOL motors firing during normal fixed-wing energy maneuvers (which confuses TECS).

GLIDERS

- `THR_MAX` = 0 (or climb only).
- `TECS_SPDWEIGHT` = **2.0** (Must prioritize speed/stall prevention).
- `SOAR_ENABLE`: Use ArduPilot Soaring features to treat thermals as lift sources rather than altitude errors.

AUTOTUNE LOGIC

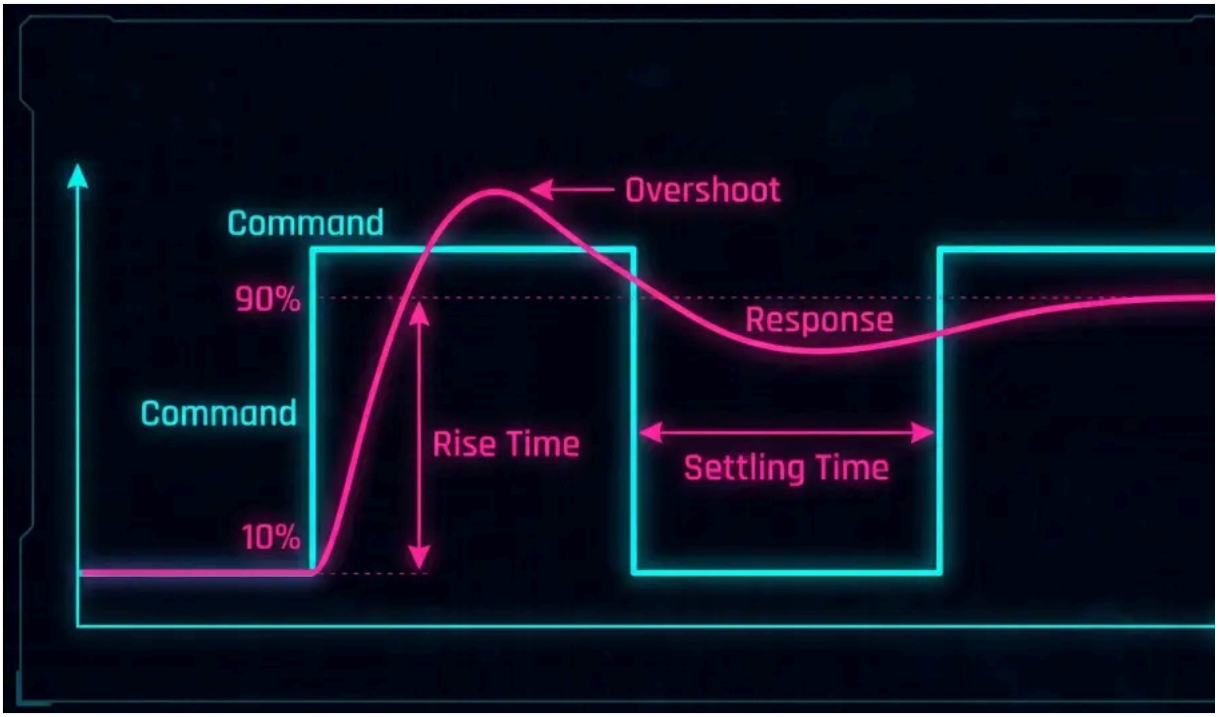
EXECUTIVE SUMMARY

AutoTune is a state machine that iteratively tunes the PID controller by injecting "Twitches" (step inputs) and measuring the vehicle's response. It balances **Performance** (fast rise time) against **Stability** (minimal overshoot/bounce-back).

THEORY & CONCEPTS

1. THE TWITCH TEST

The core mechanic is the **Twitch**.



1. **Demand:** Request a sharp 90 deg/s rotation.
2. **Measure:** Record the maximum rate reached and the "Bounce Back" (overshoot) when the stick is released.
3. **Adjust:**
    - If rise time is too slow
      $rightarrow$ Increase P.
    - If bounce back is too high
      $rightarrow$ Decrease D (or P if D is min).
    - If no bounce back

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

$rightarrow$ Increase D.

2. AGGRESSIVENESS (AUTOTUNE_AGGR)

This parameter controls the damping ratio target.

- **0.1 (High):** Allows 10\% overshoot. Results in a "locked-in" but potentially jittery tune.
- **0.05 (Low):** Over-damped. Smoother, softer video, but less propwash rejection.

CODEBASE INVESTIGATION

1. THE STATE MACHINE: `AC_AUTOTUNE_MULTI::RUN()`

The tuning sequence is implemented in `libraries/AC_AutoTune/AC_AutoTune_Multi.cpp`.
The sequence is:

1. `RD_UP`: Increase Rate D until oscillation or bounce-back is detected.
2. `RD_DOWN`: Decrease D until safe.
3. `RP_UP`: Increase Rate P to achieve the target response speed.
4. `SP_UP`: Increase Stabilize P (Angle P) to match the new rate loop speed.

2. TWITCH LOGIC: `TWITCHING_TEST_RATE()`

Located in `libraries/AC_AutoTune/AC_AutoTune_Multi.cpp`.

- It measures `meas_rate_max` (Peak) and `meas_rate_min` (Bounce).
- **Bounce Calculation:**

```
if (meas_rate_max - meas_rate_min > meas_rate_max * aggressiveness) {
    step = UPDATE_GAINS; // Too much bounce, back off
}
```

3. GAIN UPDATING: `UPDATING_RATE_D_UP()`

Located in `libraries/AC_AutoTune/AC_AutoTune_Multi.cpp`.

- Iteratively steps up gains (`tune_d += tune_d * tune_d_step_ratio`).
- If the drone becomes unstable, it reverts to the last known good gain.

SOURCE CODE REFERENCE

- **Implementation:** `libraries/AC_AutoTune/AC_AutoTune_Multi.cpp`

PRACTICAL GUIDE: AUTOTUNE SUCCESS

1. PRE-TUNE REQUIREMENTS

- **Filters:** You *must* set up the Harmonic Notch Filter *before* AutoTune. If noise enters the gyro, AutoTune will result in very low P/D gains (under-tuned).
- **CG:** Center of Gravity must be perfect.

2. "AUTOTUNE FAILED"

- **Cause:** The drone couldn't reach the target rate (90 deg/s) even with max gains.
- **Fix:** Increase ATC_ACCEL_R_MAX / ATC_ACCEL_P_MAX (Physical torque authority is too low), or check for mechanical binding.

3. POST-TUNE ADJUSTMENT

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- **Too Hot:** If motors chirp after AutoTune, multiply P, I, and D by 0.8.
- **Too Sloppy:** If it feels loose, increase ATC_INPUT_TC (Input Time Constant) to smooth the stick response without changing the PID gains.

HARMONIC NOTCH FILTERING

EXECUTIVE SUMMARY

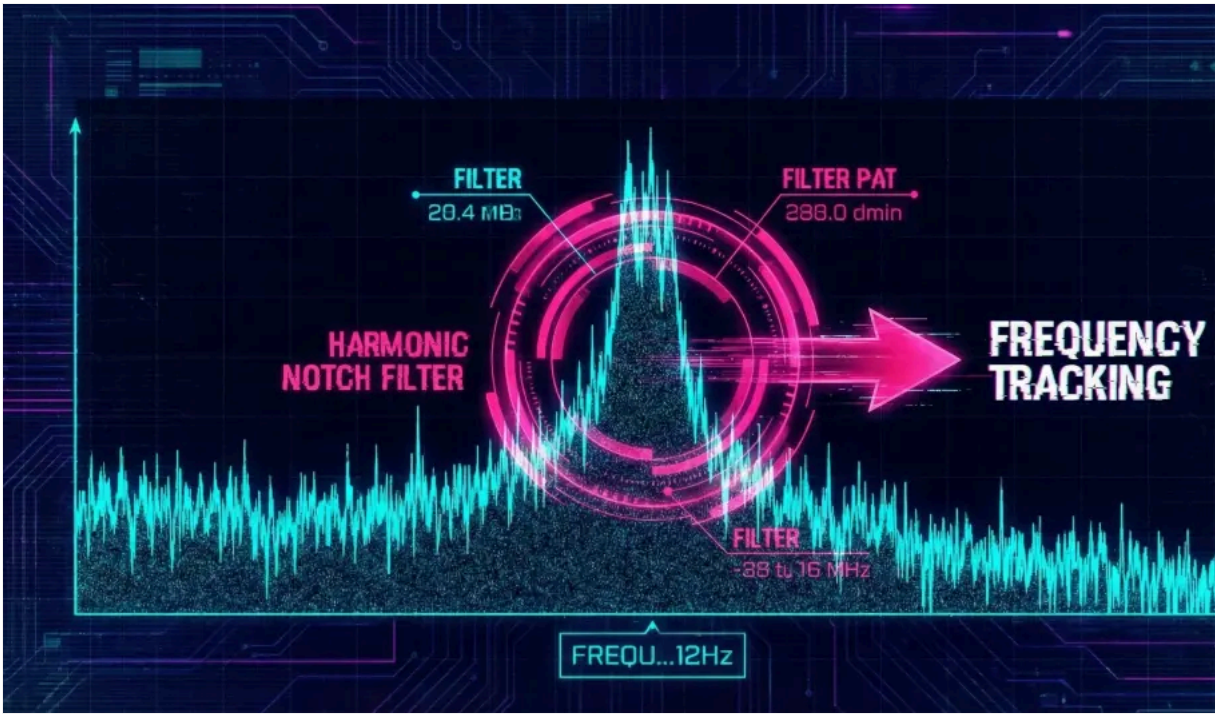
Propeller noise is the enemy of good tuning. It injects high-frequency vibration into the gyro signal, which the D-term amplifies, causing hot motors.

The **Dynamic Harmonic Notch** is a tracking filter that moves its center frequency in real-time to match the motor RPM, effectively surgically removing the noise without adding excessive phase lag at lower frequencies.

THEORY & CONCEPTS

1. THE MOTOR NOISE PEAK

Spinning propellers generate vibration at the fundamental frequency ($f = text{RPM}/60$) and its harmonics ($2f, 3f$).



- **Tracking:** As throttle increases, RPM increases, and the noise frequency shifts up. The filter must track this.
- **Bandwidth vs. Lag:** A narrow notch adds little lag but might miss the peak if tracking is slow. A wide notch catches the peak but adds more lag.

2. SOURCES OF TRUTH

- **Throttle:** Estimates RPM based on `MOT_THST_HOVER` and a linearization curve. Fast, but less accurate.
- **ESC Telemetry:** Real RPM from the ESCs. Accurate, but has latency/transport delay.
- **FFT:** Analyzes the gyro data itself to find the peak. Accurate but computationally expensive.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

CODEBASE INVESTIGATION

1. THE UPDATE LOGIC: `HARMONICNOTCHFILTER::UPDATE()`

Located in `libraries/Filter/HarmonicNotchFilter.cpp`.

- **Inputs:** `center_freq_hz` (calculated from RPM/Throttle).
- **Process:**
  1. Constrains frequency within Nyquist limits.
  2. Calculates `A` (Attenuation) and `Q` (Quality Factor) for the biquad.
  3. Updates the coefficients of the underlying `NotchFilter<T>`.

2. HARMONICS

The class manages multiple filters (`_filters[]`) to target $1f, 2f, 3f$, etc.

```
const float notch_center = constrain_float(center_freq_hz[center_n], 0.0f, nyquist_limit);
set_center_frequency(_num_enabled_filters++, notch_center, 1.0, harmonic_mul);
```

SOURCE CODE REFERENCE

- **Filter Logic:** `libraries/Filter/HarmonicNotchFilter.cpp`

PRACTICAL GUIDE: CONFIGURING THE NOTCH

1. THROTTLE-BASED (MODE 1)

- **Hover:** Hover the drone. Note the Throttle \% (e.g., 0.35) and the dominant noise frequency from the FFT log (e.g., 180Hz).
- **Calculate:**
  - **INS_HNTCH_REF** = 0.35
  - **INS_HNTCH_FREQ** = 180
- **Result:** At 70\% throttle, the filter will track to $approx 180 * sqrt{0.70 / 0.35} = 254 text{Hz}$.

2. ESC TELEMETRY (MODE 4)

- **Setup:** Enable BLHeli_32/S telemetry.
- **Set:** **INS_HNTCH_MODE** = 4.
- **Check:** `INS_HNTCH_REF` = 1 (Scaling factor).
- **Benefit:** Exact RPM matching, handles distinct RPMs (e.g., in a turn).

```

```

INPUT SHAPING & JERK LIMITING

EXECUTIVE SUMMARY

Input Shaping (often called "S-Curves" in the CNC world) transforms a square-wave pilot input into a smooth, kinematically feasible curve. This prevents the "jerk" (derivative of acceleration) from exceeding physical limits, which protects the airframe from stress and the camera from vibration.

In ArduPilot, this is split into **Attitude Input Shaping** (Pilot Feel) and **Position Input Shaping** (Navigation Smoothness).

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





THEORY & CONCEPTS

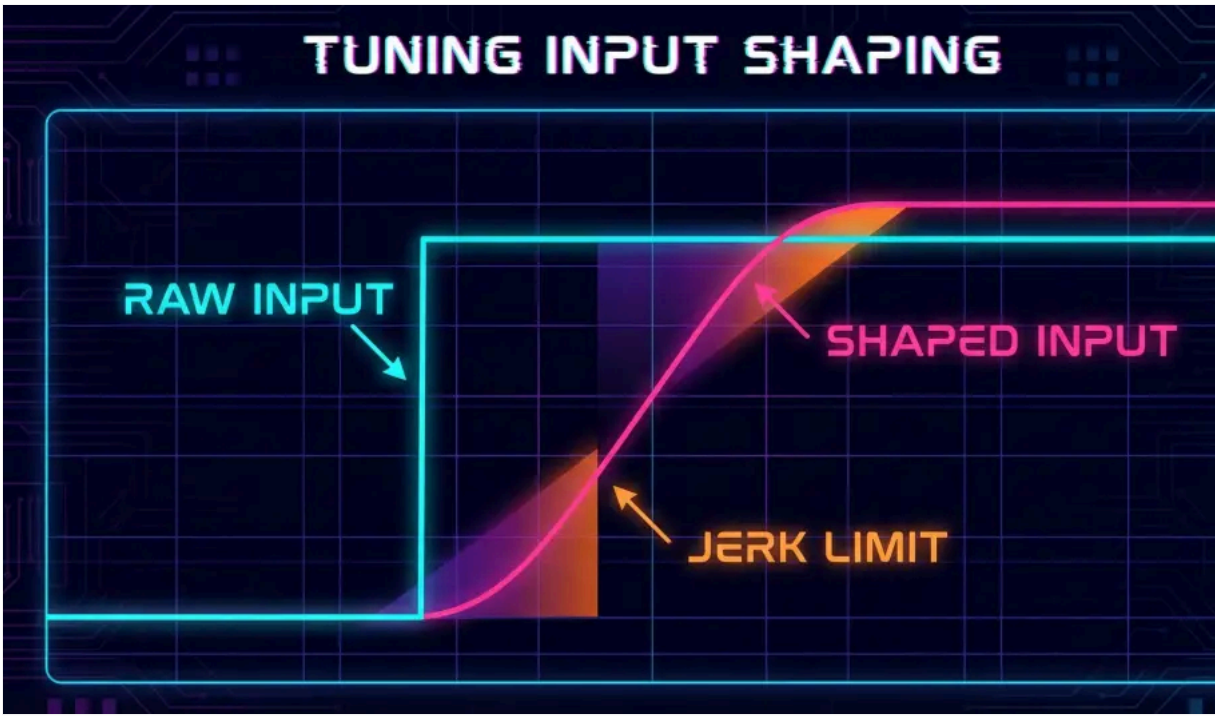
1. THE TIME CONSTANT (ATC_INPUT_TC)

This parameter defines the "softness" of the pilot's stick inputs.

- **Low TC (0.1):** Crisp, robotic response. The drone stops instantly when the stick is released.
- **High TC (0.3):** Organic, fluid response. The drone decelerates smoothly to a stop.
- **Math:** It acts as a low-pass filter on the *request*, not the output.

2. JERK LIMITING

Jerk (m/s^3) is the rate of change of acceleration.



- **Infinite Jerk:** Instant acceleration (Bang-Bang control). Impossible physically, causes overshoot.
- **Limited Jerk:** Ramps acceleration up linearly. Resulting velocity profile looks like an "S".

CODEBASE INVESTIGATION

1. ATTITUDE SHAPING: `INPUT_SHAPING_ANGLE()`

Located in `libraries/AC_AttitudeControl/AC_AttitudeControl.cpp`.

- Uses a square-root controller to calculate the velocity needed to close the angle error.
- **Logic:**

```
desired_ang_vel += sqrt_controller(error_angle, 1.0f / MAX(input_tc, 0.01f), accel_max, dt);
```

2. POSITION SHAPING: `SHAPE_POS_VEL_ACCEL_XY()`

Located in `libraries/AC_AttitudeControl/AC_PosControl.cpp`.

- Used in Auto and Guided modes.
- It calculates a valid kinematic path that respects `WPNAV_SPEED`, `WPNAV_ACCEL`, and `WPNAV_JERK`.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

- **Output:** Generates a "Leash" that the position controller follows.

SOURCE CODE REFERENCE

- **Attitude Shaper:** `libraries/AC_AttitudeControl/AC_AttitudeControl.cpp`
- **Position Shaper:** `libraries/AC_AttitudeControl/AC_PosControl.cpp`

PRACTICAL GUIDE: TUNING FOR FEEL

1. CINEMATIC FLYING

- **Goal:** Smooth stops, no "bobble" on stick release.
- **Set:** `ATC_INPUT_TC` = 0.25 or 0.30.
- **Set:** `LOIT_ACC_MAX` = $300 cm/s^2$ (Gentle braking).

2. FPV RACING / ACRO

- **Goal:** Instant response.
- **Set:** `ATC_INPUT_TC` = 0.10 or 0.05.
- **Warning:** Too low makes the drone feel robotic and exposes P-gain oscillation.

3. WAYPOINT MISSION SMOOTHNESS

- **Parameter:** WPNAV_JERK.
- **Default:** $1.0 m/s^3$.
- **Tuning:** Reducing this to 0.5 makes corners incredibly smooth but increases lap time.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





CHAPTER 15: LOG ANALYSIS

DATAFLASH STRUCTURE

EXECUTIVE SUMMARY

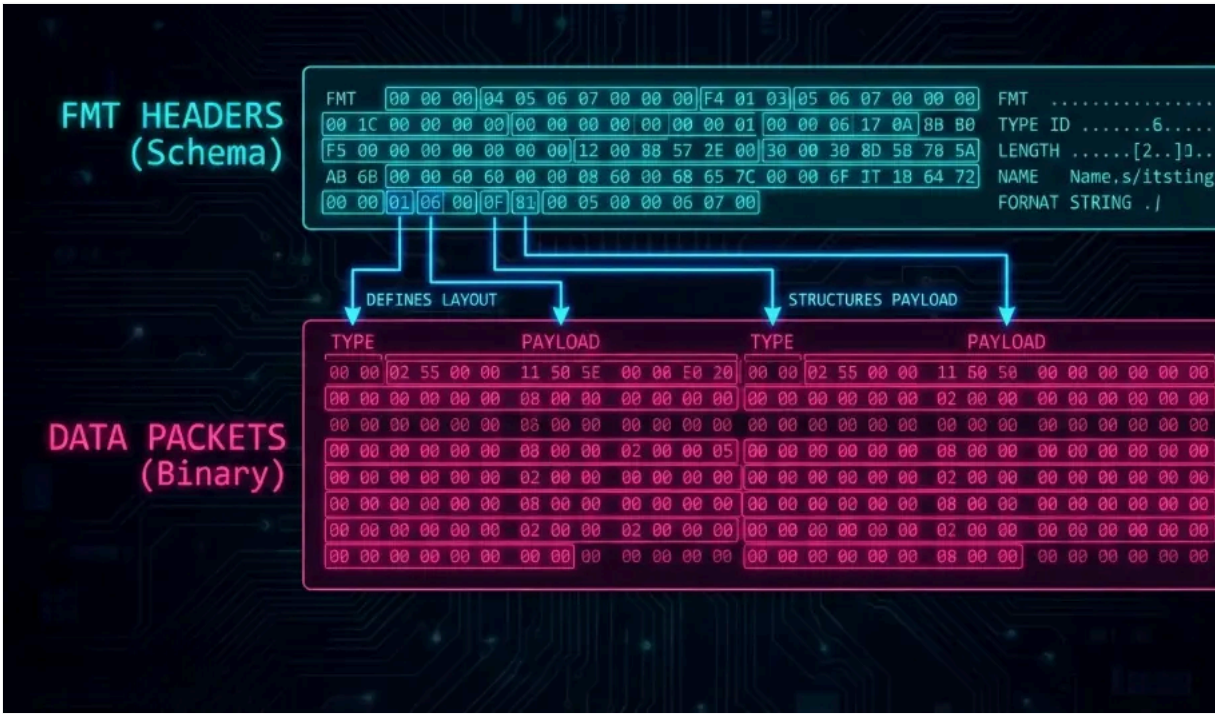
ArduPilot logs are **self-describing binary files**. They do not require a separate schema file to decode. The first few kilobytes of every `.bin` log contain `FMT` (Format) messages that define the structure, labels, units, and multipliers for every other message type in the file.

This efficiency allows ArduPilot to log high-rate data (400Hz+) with minimal CPU overhead compared to text-based formats like CSV or JSON.

THEORY & CONCEPTS

1. THE SELF-DESCRIBING HEADER

Every log starts with a series of `FMT` messages.



- **Format:** `Type, Length, Name, FormatString, Labels`
- **Example:** `FMT, 128, 89, FMT, BBnNZ, Type,Length,Name,Format,Columns`
  - This message defines the `FMT` message itself!
- **Format Strings:** Characters like `b` (int8), `f` (float), `Q` (uint64_t) define the C-type of the data.

2. UNITS & MULTIPLIERS

The `FMTU` message links a Format ID to specific SI units.

- `UNIT`: Defines the base unit (e.g., `s` for seconds, `m` for meters).
- `MULT`: Defines the scaler (e.g., `F` = 10^{-6} for microseconds).
- **Benefit:** Analysis tools (Mission Planner, PlotJuggler) can automatically label axes correctly.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

CODEBASE INVESTIGATION

1. THE LOG STRUCTURE DEFINITIONS: `LOGSTRUCTURE.H`

Located in `libraries/AP_Logger/LogStructure.h`.

- This file defines the C-structs that match the on-disk format.
- **Packed:** `struct PACKED log_ATT` ensures no padding bytes are inserted by the compiler.
- **Macros:** `LOG_PACKET_HEADER` adds the 3-byte sync header (`HEAD_BYTE1`, `HEAD_BYTE2`, `MSG_ID`).

2. WRITING DATA: `AP_LOGGER::WRITEBLOCK()`

Located in `libraries/AP_Logger/AP_Logger.cpp`.

- Data is written to a ring buffer (`_writebuf`).
- A background thread (`io_thread`) or the main loop (if no thread) flushes this buffer to the backend (SD Card, DataFlash chip, or MAVLink).

SOURCE CODE REFERENCE

- **Structure Definitions:** `libraries/AP_Logger/LogStructure.h`
- **Logger Core:** `libraries/AP_Logger/AP_Logger.cpp`

PRACTICAL GUIDE: PARSING LOGS

If you are writing a custom log parser (e.g., in Python):

1. **Read Header:** Parse all `FMT` messages first. Store them in a map `{msg_id: format_obj}`.
2. **Read Body:** Read byte-by-byte looking for `HEAD_BYTE1` (0xA3) and `HEAD_BYTE2` (0x95).
3. **Decode:** Read the `MSG_ID` byte, look it up in your map, read `Length` bytes, and unpack using the `FormatString`.

EKF INNOVATIONS & LANE SWITCHING

EXECUTIVE SUMMARY

The Extended Kalman Filter (EKF) is the brain of the autopilot. It fuses data from IMUs, GPS, Compass, and Barometer to estimate position and attitude.

Innovation is the difference between what the EKF *predicted* a sensor would read and what it *actually* read.

- **Low Innovation:** Sensors agree with the model. All good.
- **High Innovation:** Something is wrong (GPS glitch, compass interference, vibration).

If innovations get too high on the primary core (Lane), the EKF will switch to a healthy backup lane ("Lane Switch").

THEORY & CONCEPTS

1. INNOVATION VS. VARIANCE

- **Innovation:** $Measurement - Prediction.$
- **Test Ratio (Variance):**
  $fracInnovation^2Uncertainty.$
  - This normalizes the error. A 1 meter GPS error is huge if accuracy is 0.1m, but fine if accuracy is 5m.
  - ArduPilot logs the **Test Ratio** (normalized to < 1.0 is good).

2. EKF LANES

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





Modern ArduPilot runs multiple EKF instances ("Lanes") in parallel, typically one per IMU.

- **Primary Lane:** The one controlling the vehicle.
- **Lane Switching:** If the Primary Lane's "Error Score" exceeds the others, the system swaps control to the better lane.

CODEBASE INVESTIGATION

1. THE SWITCH LOGIC: `checkLaneSwitch()`

Located in `libraries/AP_NavEKF3/AP_NavEKF3.cpp`.

- It calculates an `errorScore` for each lane based on velocity, position, and magnetometer innovations.
- **Threshold:** It switches if `altErrorScore < lowestErrorScore`.
- **Hysteresis:** It won't switch more than once every 5 seconds (to prevent thrashing).

2. ERROR SCORE CALCULATION

The score is a weighted sum of test ratios:

- `velTestRatio` (Velocity)
- `posTestRatio` (Position)
- `hgtTestRatio` (Height - Baro/GPS)
- `magTestRatio` (Compass)

SOURCE CODE REFERENCE

- **Lane Switching:** `libraries/AP_NavEKF3/AP_NavEKF3.cpp`
- **Core Update:** `libraries/AP_NavEKF3/AP_NavEKF3_core.cpp`

PRACTICAL GUIDE: ANALYZING LOGS

1. KEY MESSAGES

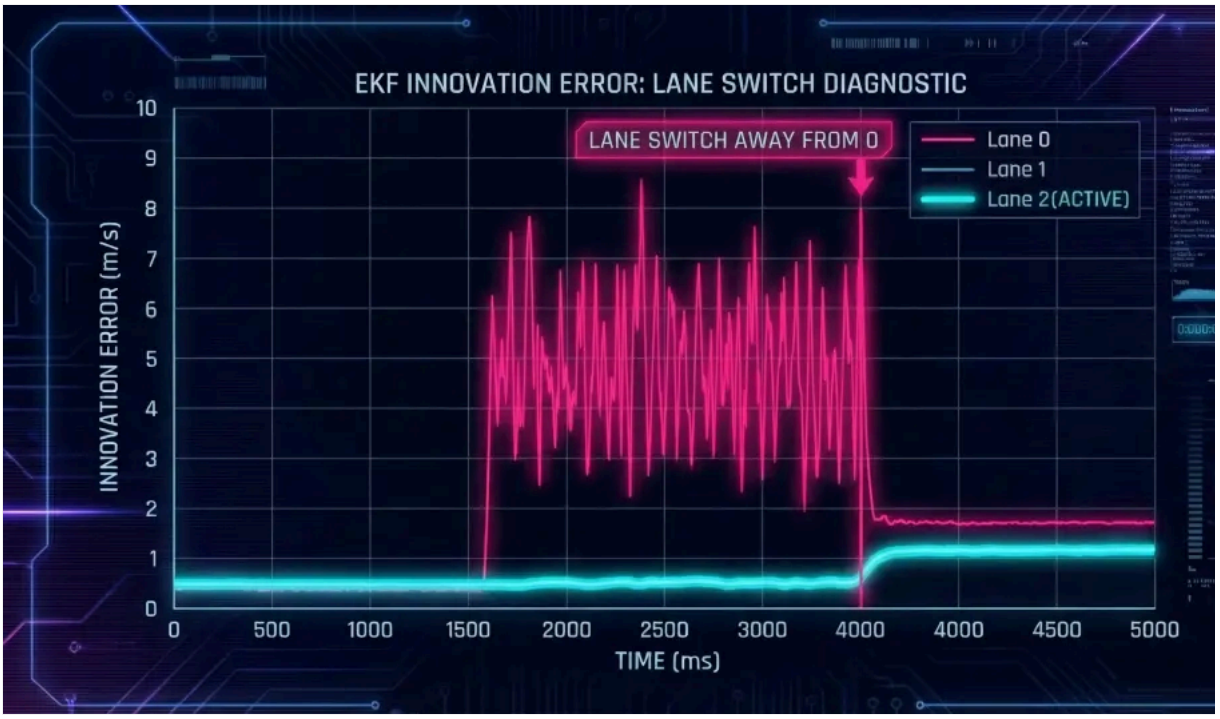
- `XKF1`: States (Roll, Pitch, Yaw, Velocity).
- `XKF4`: Innovations (The most important message for diagnostics).
  - `IV`: Velocity Innovation (GPS speed vs IMU prediction).
  - `IP`: Position Innovation.
  - `SM`: Magnetometer Innovation.
- `XKF3`: Lane Selection (`C` field tells you the active core: 0, 1, or 2).

2. IDENTIFYING A "LANE SWITCH"

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY







- Look at `XKF3.C`. If it changes from 0 to 1 mid-flight, a switch occurred.
- Correlate with `XKF4`. Did `IV` or `SM` spike on Core 0 just before the switch?

3. COMMON CAUSES

- **Compass Variance:** High `SM` (Compass Innovation). Usually magnetic interference from motors.
- **Velocity Variance:** High `IV`. Often vibration (aliasing) or bad GPS signal (multipath).

VIBRATION ANALYSIS & FFT

EXECUTIVE SUMMARY

Vibration is the root cause of many flight issues, from "Jello" video to EKF Lane Switching and even flyaways. ArduPilot provides two powerful tools to diagnose this: **VIBE** logs (broadband vibration level) and **FFT** logs (frequency-specific vibration).

THEORY & CONCEPTS

1. VIBRATION LEVELS (VIBE)

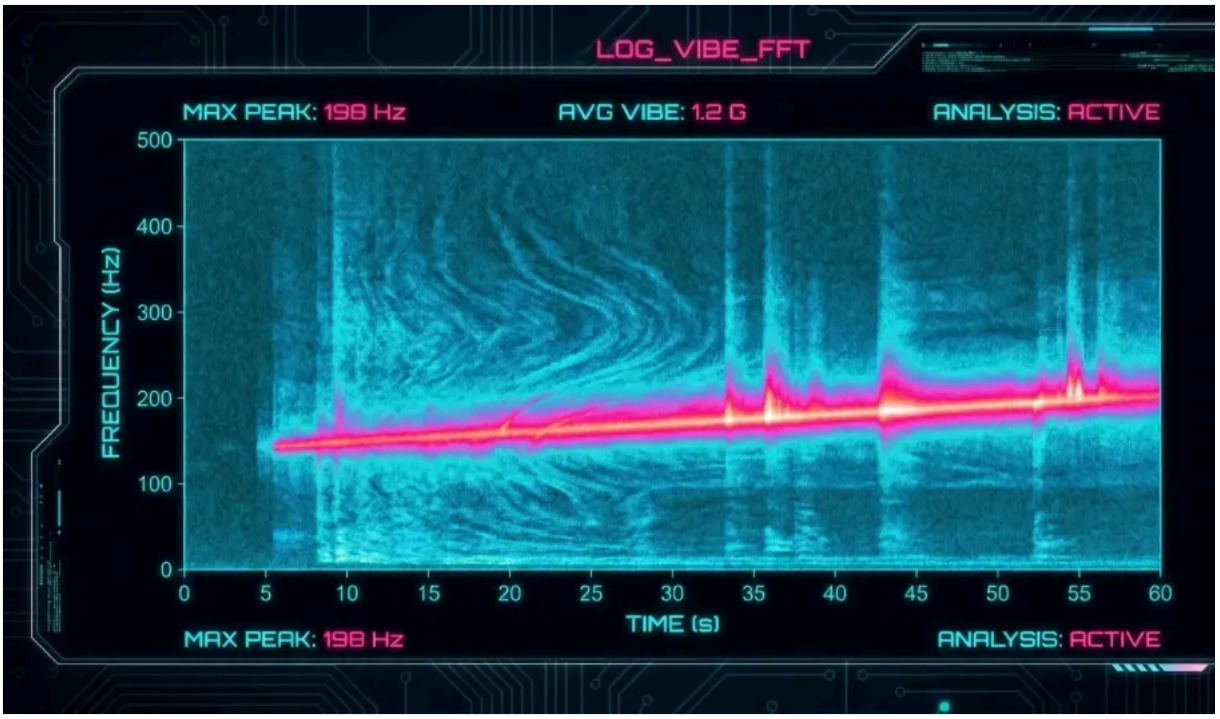
VIBE measures the "roughness" of the ride.

- **Metric:** Standard Deviation of the accelerometer signal in m/s^2.
- **Filtering:** ArduPilot calculates this by subtracting a 5Hz low-pass filtered signal from the raw signal. What remains is the high-frequency "noise."
- **Clipping:** When vibration exceeds the sensor's range (e.g., 16G), the accelerometer "clips" (flatlines). This destroys the EKF's ability to estimate gravity.

2. FAST FOURIER TRANSFORM (FFT)

VIBE tells you *how much* vibration exists. FFT tells you *at what frequency*.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY



- **Fundamental:** The primary frequency usually corresponds to the motor RPM ($RPM/60$).
- **Harmonics:** Multiples of the fundamental ($2x, 3x$).
- **Use Case:** We use FFT graphs to precisely tune the **Harmonic Notch Filter**.

CODEBASE INVESTIGATION

1. VIBE CALCULATION: `CALC_VIBRATION_AND_CLIPPING()`

Located in `libraries/AP_InertialSensor/AP_InertialSensor.cpp`.

- **Logic:**
  1. `accel_filt = _accel_vibe_floor_filter.apply(accel)` (5Hz LPF).
  2. `accel_diff = accel - accel_filt`.
  3. `vibe = _accel_vibe_filter.apply(accel_diff^2)` (2Hz LPF of the square).
- **Clipping:** If `fabsf(accel.x) > 16G`, increment `_accel_clip_count`.

2. FFT LOGGING

- The AP_GyroFFT library performs real-time frequency analysis on the gyro data.
- **Log Message:** `FTN` (Filter Tune). Logs the center frequency and bandwidth of the detected noise peaks.

SOURCE CODE REFERENCE

- **Vibe Logic:** `libraries/AP_InertialSensor/AP_InertialSensor.cpp`

PRACTICAL GUIDE: ANALYZING VIBRATION

1. ACCEPTABLE VIBE LEVELS

- **<** $15m/s^2$**:** Excellent.
- $15 - 30m/s^2$**:** Acceptable for most multirotors.
- **>** $30m/s^2$**:** Problematic. Position hold may wander.
- **>** $60m/s^2$**:** Dangerous. EKF may fail.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

2. DIAGNOSING WITH FFT

1. Enable "Batch Sampling" (`INS_LOG_BAT_MASK = 1`).
2. Hover the drone for 1 minute.
3. Open the log in Mission Planner → press **Ctrl+F** → **FFT**.
4. Look for the big spike. That is your `INS_HNTCH_FREQ`.

3. CLIPPING = CRASH

If `VIBE.Clip0`, `Clip1`, or `Clip2` increases during flight, you have a hard mounting issue or a damaged prop. **Do not fly** until fixed.

POWER FORENSICS

EXECUTIVE SUMMARY

Power failure is the most common cause of non-pilot-error crashes. Analyzing power logs allows you to distinguish between **Battery Exhaustion** (ran out of capacity), **Voltage Sag** (over-current/old battery), and **Brownouts** (flight controller power failure).

THEORY & CONCEPTS

1. VOLTAGE SAG VS. CAPACITY

- **Capacity (mAh):** The "Fuel" in the tank. Consumed over time.
- **Sag (Volts):** Instantaneous voltage drop due to load ($V_{drop} = I \\times R_{internal}$).
- **Danger:** High current causes deep sag. If voltage drops below the ESC cutoff (e.g., 3.0V/cell), the ESCs will reboot mid-air even if 50\% capacity remains.

2. BROWNOUTS

A brownout occurs when the 5V rail powering the Flight Controller drops below ~4.5V.

- **Effect:** The MCU resets. Log ends abruptly.
- **Cause:** Overloaded BEC (too many servos/LEDs) or short circuit.

CODEBASE INVESTIGATION

1. BATTERY MONITOR: `AP_BATTMONITOR::READ()`

Located in `libraries/AP_BattMonitor/AP_BattMonitor.cpp`.

- It reads raw ADC values for Voltage and Current.
- **Integration:**

```
_consumed_mah += (current_amps * dt) / 3600.0f;
```

- **Logging:** Writes `BAT` (Basic info) and `BCL` (Cell levels if available).

2. SYSTEM POWER: `LOG_POWR`

- `Vcc`: The voltage of the 5V rail powering the board.
- `VServo`: The voltage of the Servo rail.

SOURCE CODE REFERENCE

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

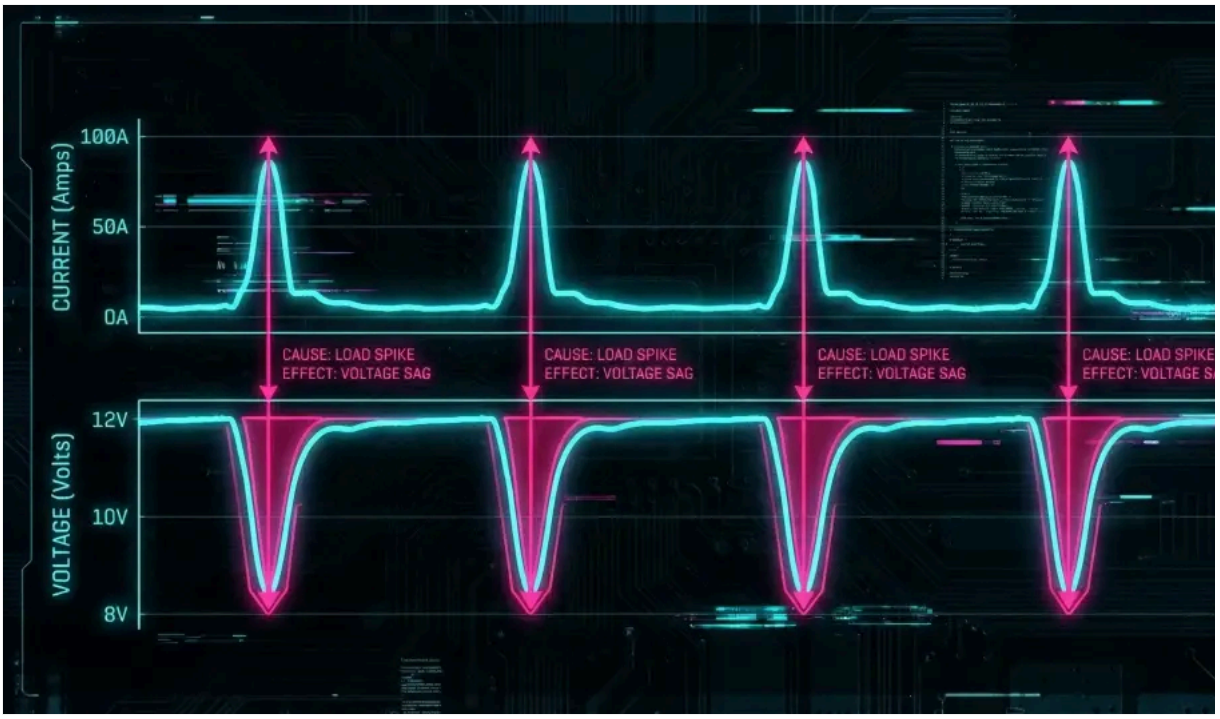




- **Monitor Core:** `libraries/AP_BattMonitor/AP_BattMonitor.cpp`

PRACTICAL GUIDE: FORENSICS

1. THE "V" SHAPE



- Look at `BAT.Volt` vs `BAT.Curr`.
- If Voltage dips sharply exactly when Current spikes (Throttle Punch), your battery has high internal resistance (old/weak/cold).
- **Fix:** Use higher C-rated batteries or warm them up.

2. THE ABRUPT END

- If the log ends mid-flight, check `POWR.Vcc` in the last few seconds.
- If `Vcc` drifts below 4.8V or has noise, your power module is failing.

3. TOTAL CONSUMPTION

- Compare `BAT.CurrTot` (mAh used) against your battery's rated capacity.
- **Rule of Thumb:** You should land before using 80\% of rated capacity.

WATCHDOG & SYSTEM FAULTS

EXECUTIVE SUMMARY

When the autopilot reboots unexpectedly or disarms mid-air, the **Watchdog** and **Fault** logs are the first place to look. These logs capture the "Black Box" data at the moment of failure.

- **Watchdog (WDOG):** The CPU locked up, and the independent hardware timer reset it.
- **Fault (ERR):** A software subsystem reported a critical failure.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

THEORY & CONCEPTS

1. THE WATCHDOG TIMER

The STM32 has an independent hardware timer that counts down. The main loop must "pet" (reset) this timer every loop. If the CPU freezes (infinite loop, DMA lockup), the timer expires and forces a hard reset.

- **Log:** `WDOG`.
- **Data:** Captures the Program Counter (PC) and Stack Pointer (SP) at the moment of death.

2. THE HARDFAULT HANDLER

If code tries to access invalid memory (Null Pointer, Stack Overflow), the CPU triggers a `HardFault`. ArduPilot catches this, saves the register state to a special area of RAM (no-init), reboots, and *then* writes it to the log on the next boot.

CODEBASE INVESTIGATION

1. WATCHDOG LOGGING

Located in `libraries/AP_Logger/AP_Logger.cpp`.

- On boot, `AP_Logger` checks if a watchdog reset occurred (`hal.util->was_watchdog_armed()`).
- If yes, it writes a `WDOG` message containing the fault registers.

2. ERROR SUBSYSTEMS: `LOG_WRITE_ERROR`

Located in `ArduCopter/Log.cpp` (and other vehicles).

- **Subsystem:** Where the error occurred (Compass, GPS, EKF, etc.).
- **Error Code:** Specific failure type (e.g., `ERROR_SUBSYSTEM_FAILSAFE_RADIO`).

SOURCE CODE REFERENCE

- **Logger Implementation:** `libraries/AP_Logger/AP_Logger.cpp`

PRACTICAL GUIDE: THE "CRASH" LOG

1. "INTERNAL ERROR"

- Search the log for `MSG` lines containing "Internal Error".
- Example: `Internal Error: 0x8000000 (map_fail)`.
- **Action:** This is almost always a firmware bug. Report it to the developers with the log.

2. THE WATCHDOG RESET

- If `WDOG` is present, the board reset in flight.
- **Task:** `SchR` (Scheduler Overrun). If this is high just before the reset, the CPU was overloaded.

3. "SUBSYS" CODES

- **1:** Main (Never good).
- **2:** Radio (RC Failsafe).
- **3:** Compass (Mag failure).
- **4:** OptFlow.
- **5:** FailSafe (General).

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





CHAPTER 16: BUILD SYSTEM

THE WAF BUILD SYSTEM

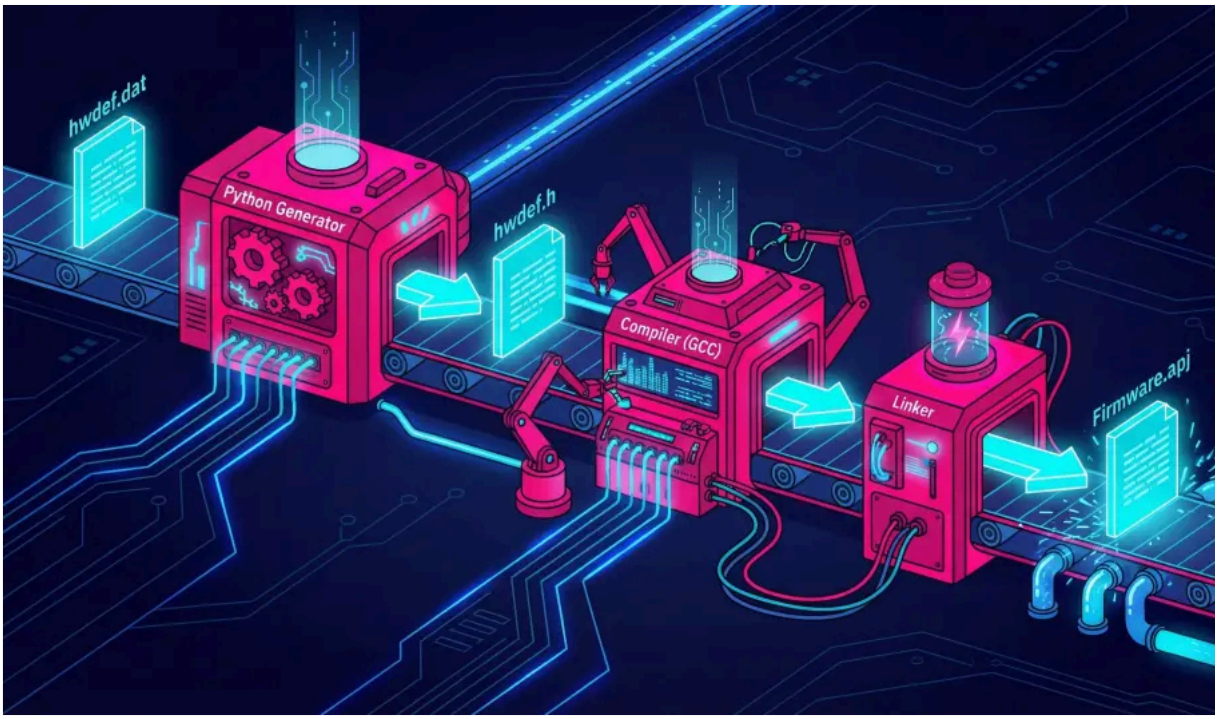
EXECUTIVE SUMMARY

ArduPilot uses **Waf**, a Python-based build automation tool, to manage the compilation of its massive codebase. Unlike Make or CMake, Waf scripts are valid Python programs, allowing for complex logic during the build process.

The `wscript` in the root directory is the entry point. It handles board configuration, feature selection (via `ap_config.h`), and task scheduling.

THEORY & CONCEPTS

1. CONFIGURATION VS. BUILD



- **Configure:** `./waf configure --board=CubeOrange`
  - This step scans your system for compilers (arm-none-eabi-gcc), checks dependencies, and generates a `build/CubeOrange/ap_config.h` file containing `#define` macros for the specific board.
- **Build:** `./waf copter`
  - This step compiles the code using the configuration generated in the previous step.

2. THE `AP_CONFIG.H` MECHANISM

Instead of passing thousands of flags to the compiler (e.g., `-DHAL_OSD_ENABLED=1`), Waf generates a single header file `ap_config.h` that is included in every C++ file. This ensures consistent feature flags across the entire build.

CODEBASE INVESTIGATION

1. THE ROOT `WSCRIPT`

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

Located in `wscript`.

- **Initialization:** The `init()` function loads the environment and sets up the build context.
- **Options:** The `options()` function defines command-line flags like `--enable-scripting`, `--[debug](/field-manual/mavlink-interface/debug.html)-symbols`, and `--disable-watchdog`.
- **Board Selection:** `cfg.get_board().configure(cfg)` calls the board-specific configuration logic found in `Tools/ardupilotwaf/boards.py`.

2. FEATURE SELECTION

The `wscript` dynamically enables/disables features based on board capabilities (flash size, memory).

```
if cfg.options.enable_scripting:
    cfg.env.ENABLE_SCRIPTING = True
```

This directly maps to `#define AP_SCRIPTING_ENABLED 1` in `ap_config.h`.

SOURCE CODE REFERENCE

- **Main Script:** `wscript`
- **Board Definitions:** `Tools/ardupilotwaf/boards.py`

PRACTICAL GUIDE: USEFUL WAF COMMANDS

1. DEBUGGING

- `./waf configure --board=[sitl](/field-manual/build-system/sitl-architecture.html) --debug --debug-symbols`
  - Disables optimizations (`-O0`) and adds debug symbols (`-g`), essential for GDB stepping.

2. UPLOADING

- `./waf copter --upload`
  - Compiles and immediately attempts to upload the firmware to a connected board via USB.

3. CLEANING

- `./waf clean`
  - Removes build artifacts. Essential if you switch branches or change fundamental defines in `hwdef.dat`.

CUSTOMIZING FEATURES

EXECUTIVE SUMMARY

ArduPilot is designed to be modified. There are two primary ways to inject custom logic: **User Hooks** (for simple logic) and **Build Configuration** (for hardware/driver changes).

The `UserCode.cpp` file provides predefined entry points into the main loop, allowing you to run custom code at 100Hz, 50Hz, 10Hz, etc., without altering the complex scheduler.

THEORY & CONCEPTS

1. USER HOOKS

The main vehicle scheduler (e.g., `Copter.cpp`) calls specific functions like `userhook_FastLoop()` at fixed intervals. By default, these functions are empty.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- **Compilation:** You must define macros like `USERHOOK_FASTLOOP` in your `ap_config.h` or via a custom header to enable these calls.

2. FEATURE STRIPPING

To run ArduPilot on smaller boards (1MB Flash), you often need to disable features.

- `hwdef.dat`: The hardware definition file allows you to exclude drivers (`define HAL_OSD_ENABLED 0`).
- `wscript`: You can disable entire subsystems like Scripting or OSD via command line flags.

CODEBASE INVESTIGATION

1. THE USER HOOKS: `ARDUCOPTER/USERCODE.CPP`

Located in `ArduCopter/UserCode.cpp`.

- Contains empty implementations of:
  - `userhook_init()`: Called once at startup.
  - `userhook_FastLoop()`: Called at 100Hz (or main loop rate).
  - `userhook_MediumLoop()`: Called at 10Hz.
  - `userhook_SlowLoop()`: Called at 3.3Hz.

2. ENABLING HOOKS

To actually use these, you create a file named `UserVariables.h` (which is included by `Copter.h` but git-ignored) and add:

```
#define USERHOOK_FASTLOOP 1
#include "UserVariables.h"
```

SOURCE CODE REFERENCE

- **User Code:** `ArduCopter/UserCode.cpp`

PRACTICAL GUIDE: ADDING A CUSTOM FEATURE

1. THE "BLINKING LED" HELLO WORLD

1. Create `ArduCopter/UserVariables.h`.
2. Add: `#define USERHOOK_SLOWLOOP`
3. In `UserCode.cpp`, modify `userhook_SlowLoop()`:

   ```
   void Copter::userhook_SlowLoop() {
       hal.gpio->toggle(HAL_GPIO_A_LED_PIN);
   }
   ```

4. Compile and upload.

2. SAVING FLASH SPACE

If your build fails with "region 'flash' overflowed":

1. Edit your board's `hwdef.dat` in `libraries/AP_HAL_ChibiOS/hwdef/`.
2. Add lines to disable unused features:

   ```
   define HAL_MOUNT_ENABLED 0
   define HAL_CAMERA_ENABLED 0
   define HAL_OSD_ENABLED 0
   ```


Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





3. Reconfigure and build: `./waf configure --board=YourBoard && ./waf copter`.

GDB DEBUGGING SETUP

EXECUTIVE SUMMARY

Debugging embedded firmware with `printf` is painful. **GDB (GNU Debugger)** allows you to freeze the processor, inspect memory, walk through code line-by-line, and catch crashes exactly where they happen.

To use GDB with ArduPilot, you need a **Hardware Debugger** (Blackmagic Probe or ST-Link) and a **Debug Build** of the firmware.

THEORY & CONCEPTS

1. DEBUG SYMBOLS

By default, the compiler strips variable names and line numbers to save space.

- **Flag:** `--debug-symbols` (or `-g`).
- **Result:** Generates an `.elf` file containing the map between binary machine code and your C++ source lines.

2. OPTIMIZATION LEVELS

- **Release (`-O2` or `-Os`):** The compiler reorders instructions and inlines functions for speed. Stepping through this in GDB is confusing (the cursor jumps around wildly).
- **Debug (`-O0`):** No optimization. Code executes exactly as written. Essential for stepping.

CODEBASE INVESTIGATION

1. WAF CONFIGURATION

To generate a debug-friendly build:

```
./waf configure --board=CubeOrange --debug --debug-symbols
./waf copter
```

This produces `build/CubeOrange/bin/arducopter.elf`.

2. CONNECTING THE HARDWARE

- **SWD (Serial Wire Debug):** The standard ARM debug interface.
- **Pins:** SWDIO, SWCLK, GND. (VTREF is optional but recommended).
- **Port:** On a Cube Orange, this is the small "Debug" JST-SH connector next to the USB port.

SOURCE CODE REFERENCE

- **Waf Options:** `wscript` (See `--debug-symbols`)

PRACTICAL GUIDE: THE GDB SESSION

1. LAUNCHING GDB

```
arm-none-eabi-gdb build/CubeOrange/bin/arducopter.elf
```


Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





2. CONNECTING (BLACKMAGIC PROBE)

```
target extended-remote /dev/ttyACM0
monitor swdp_scan
attach 1
```

3. CONNECTING (ST-LINK / OPENOCD)

Start OpenOCD in a separate terminal:

```
openocd -f interface/stlink.cfg -f target/stm32h7x.cfg
```

In GDB:

```
target remote localhost:3333
```

4. ESSENTIAL COMMANDS

- `load`: Flashes the firmware (slow, but works).
- `b loop`: Set a breakpoint at the main loop.
- `c`: Continue execution.
- `bt`: Backtrace (Show the call stack - critical for crashes).
- `p var`: Print the value of variable `var`.

SITL ARCHITECTURE

EXECUTIVE SUMMARY

Software In The Loop (SITL) allows ArduPilot to run on a standard PC (Linux/Windows/macOS) as if it were running on a flight controller.

It achieves this by replacing the hardware drivers (AP_HAL) with a simulation layer (`AP_HAL_SITL`) that communicates with a physics engine (`libraries/SITL` or external simulators like Gazebo) via TCP/UDP sockets.

THEORY & CONCEPTS

1. THE HAL LAYER

ArduPilot code writes to `hal.gpio->write()` or `hal.i2c->transfer()`.

- **On Hardware (ChibiOS):** This toggles a register on the STM32.
- **On SITL:** This writes a packet to a socket or shared memory buffer. The "Physics" side reads this packet (e.g., "Motor 1 at 50\%") and calculates the resulting motion.

2. THE PHYSICS ENGINE

The simulator calculates the vehicle's position, velocity, and sensor data based on the motor outputs.

- **Internal:** Simple kinematics (F=ma) built into ArduPilot (`SIM_Multicopter.cpp`).
- **External:** Complex physics engines like Gazebo, X-Plane, or AirSim.

CODEBASE INVESTIGATION

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





1. AP_HAL_SITL

Located in `libraries/AP_HAL_SITL`.

- `Scheduler.cpp`: Replaces the RTOS scheduler with a standard OS thread/sleep loop.
- `UARTDriver.cpp`: Redirects "Serial Ports" to TCP ports (5760 = Serial0, 5762 = Serial2).

2. SIM LIBRARIES

Located in `libraries/SITL`.

- `SIM_Aircraft.cpp`: The base class for all vehicle physics models.
- `SIM_Sensor.cpp`: Generates fake sensor noise and drift to make the simulation realistic.

SOURCE CODE REFERENCE

- **HAL Implementation:** `libraries/AP_HAL_SITL`
- **Physics Models:** `libraries/SITL`

PRACTICAL GUIDE: RUNNING CUSTOM PHYSICS

You can write your own physics backend!

1. Inherit from `SIM_Aircraft`.
2. Implement `update(const struct sitl_input &input)`.
3. Calculate new state (Lat/Lon/Alt/Attitude).
4. Register it in `SITL.cpp`.

SIGNING & SECURE BOOT

EXECUTIVE SUMMARY

Secure Boot ensures that only authorized firmware can run on the flight controller. This is critical for preventing tampering, cloning, or malicious code injection.

ArduPilot uses **Monocypher** (Ed25519) for asymmetric cryptography. You hold the **Private Key** (used to sign the firmware) and the bootloader holds the **Public Key** (used to verify the signature).

THEORY & CONCEPTS

1. THE SIGNED FIRMWARE FORMAT (`.APJ`)

The standard `.apj` file is just a JSON file containing the firmware binary (base64 encoded) and metadata.

- **Signed:** A signed firmware includes a cryptographic signature of the binary data.
- **Unsigned:** Standard builds have no signature field or a null signature.

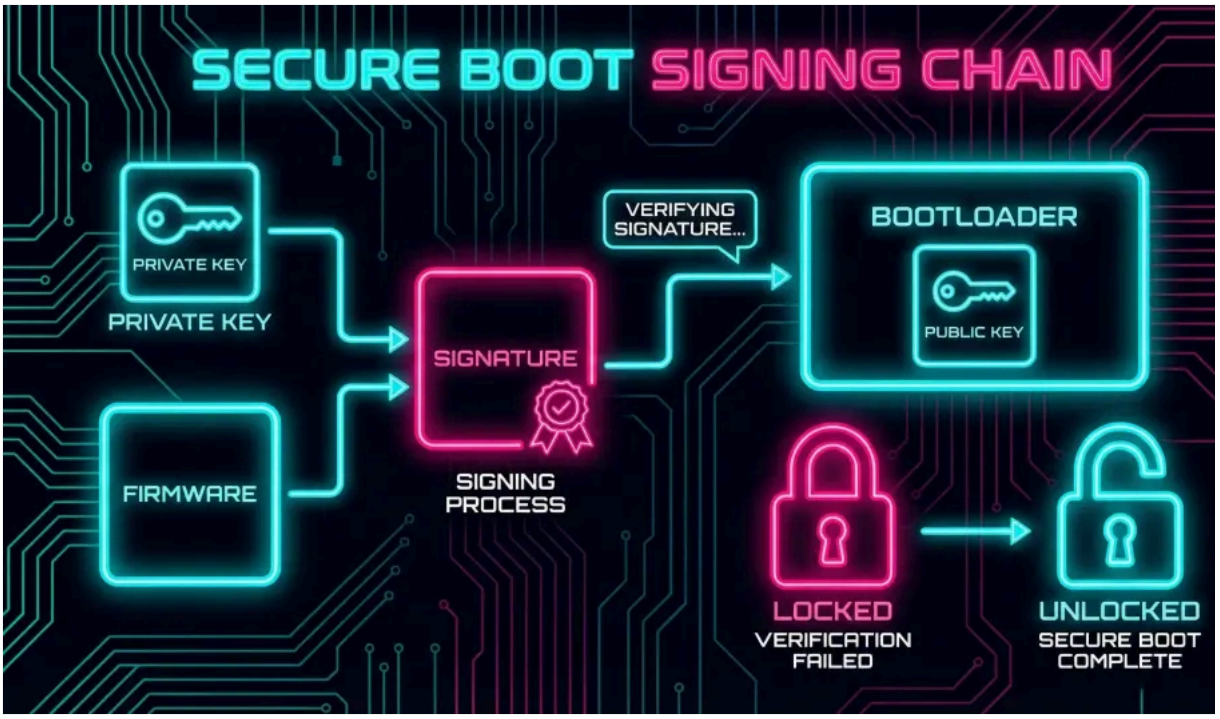
2. THE SECURE BOOTLOADER

The standard bootloader accepts any valid firmware. The **Secure Bootloader** checks the signature against its embedded Public Key.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY







- **Valid Signature:** Boot proceeds.
- **Invalid/Missing:** Boot halts. The board stays in bootloader mode.

CODEBASE INVESTIGATION

1. KEY GENERATION: `GENERATE_KEYS.PY`

Located in `Tools/scripts/signing/generate_keys.py`.

- Generates a `private_key.dat` (KEEP THIS SECRET) and `public_key.dat`.
- Also generates a C header file containing the public key for compiling into the bootloader.

2. SIGNING SCRIPT: `MAKE_SECURE_FW.PY`

Located in `Tools/scripts/signing/make_secure_fw.py`.

- Takes an unsigned `.apj` and the `private_key.dat`.
- Calculates the Ed25519 signature.
- Outputs a new, signed `.apj`.

SOURCE CODE REFERENCE

- **Signing Tools:** `Tools/scripts/signing/`

PRACTICAL GUIDE: SECURING YOUR FLEET

1. GENERATE KEYS

```
python3 Tools/scripts/signing/generate_keys.py MySecretKeys
```

2. BUILD THE SECURE BOOTLOADER

You must compile a custom bootloader with your Public Key.

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

```
./waf configure --board=CubeOrange --bootloader --signed-fw --private-key=MySecretKeys/private_key.dat
./waf bootloader
```

(Note: The build system embeds the key automatically if configured correctly).

3. SIGN YOUR FIRMWARE

```
python3 Tools/scripts/signing/make_secure_fw.py --key MySecretKeys/private_key.dat --img build/CubeOrange/bin/arducopter.apj
```

4. LOCK THE BOARD

Flash the signed bootloader. Once installed, it will reject any unsigned firmware. **Warning:** If you lose your private key, you cannot update the firmware ever again (unless you use a hardware programmer to wipe the chip).

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





CHAPTER 17: FLASHING & CLONING

CLONING FLIGHT CONTROLLERS

To scale production from a single prototype to multiple identical aircraft, we must eliminate the time-consuming and error-prone process of manually calibrating and tuning each flight controller. This guide outlines the definitive, low-level process for creating a **Golden Master** image from a perfectly tuned flight controller and flashing it as a perfect clone onto new boards.

This ensures that every aircraft produced is a 1:1 replica of the master prototype, capturing not just the firmware, but also all parameters, sensor calibrations (IMU, compass, etc.), and RC setups.

TECHNICAL CONCEPT: THE "GOLDEN MASTER"

The flash memory of a flight controller contains two distinct types of data:

1. **Firmware:** The compiled ArduPilot code (`.apj` or `.hex` file).
2. **User Data:** A dedicated memory region that stores all parameters and calibrations.

By using a hardware debugger (ST-Link) and OpenOCD, we can bypass high-level protocols like MAVLink and perform a **Raw Memory Dump**. This captures the entire 2MB flash (on H743 chips) into a single binary file. Flashing this single file onto an identical board guarantees a bit-for-bit clone.

---

1. PREREQUISITES

Before beginning, you must install the necessary communication and debugging tools.

```
#### Update your package list
sudo apt-get update

#### Install OpenOCD (The communication bridge)
sudo apt-get install openocd

#### Install the ARM toolchain (Required for GDB debugging)
sudo apt-get install gcc-arm-none-eabi gdb-arm-none-eabi
```

---

2. LOCATE YOUR "DRIVERS" (SCRIPT DISCOVERY)

OpenOCD uses pre-written scripts to understand how to talk to your specific programmer and chip. You don't need to write these; they are installed with OpenOCD.

2.1 FIND THE PROGRAMMER SCRIPT

For most flight controllers, we use the **ST-Link v2**. Find its script:

```
find /usr/share/openocd/scripts -name "stlink.cfg"
```

2.2 FIND THE CHIP SCRIPT

For the MicoAir 743, we are looking for the **STM32H7** target script:

```
find /usr/share/openocd/scripts -name "stm32h7x.cfg"
```


Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





3. FORCING CONNECTION (DFU MODE)

Flight controllers running ArduPilot often "lock" the debug pins as soon as they boot to protect the CPU or reuse pins for peripherals. To ensure a stable connection, you must prevent ArduPilot from starting by booting into **DFU Mode**.

1. **Unplug** the USB and ST-Link from the board.
2. Press and **HOLD** the physical **BOOT button** on the flight controller.
3. While holding the button, **plug in the USB cable** to your computer.
4. Wait 1 second and **release the button**.
5. The board is now in a "Paused" state and ready for a debugger connection.

---

4. UNDERSTANDING THE MEMORY MAP

To clone a board without guessing, you need the **Start Address** and the **Size**.

4.1 THE START ADDRESS: `0x08000000`

On virtually every STM32 microcontroller, the internal flash memory is hard-wired to start at the address `0x08000000`. This is where the **Bootloader** lives. By starting here, we capture the entire system.

4.2 THE SIZE: HOW MUCH TO READ?

You must determine the flash size of your specific chip.

- **H743VI:** 2MB (2048 KB) → Hex: `0x200000`
- **F405RG:** 1MB (1024 KB) → Hex: `0x100000`

---

5. PHASE A: CREATING THE "GOLDEN MASTER"

Connect your **fully tuned and calibrated** master aircraft to the ST-Link. We will perform a "Raw Memory Dump."

```
#### Template: dump_image [filename] [start_address] [size]
#### This command pulls the full 2MB from the starting address.
openocd -f interface/stlink.cfg -f target/stm32h7x.cfg \
  -c "init; halt; dump_image golden_master.bin 0x08000000 0x200000; exit"
```

---

6. PHASE B: FLASHING THE CLONE

Connect the **new, blank** flight controller (also in DFU mode). We use the `program` command which handles erasing, writing, and verification.

CRITICAL NOTE FOR H743: The standard OpenOCD scripts often fail to see the second 1MB bank automatically. You must manually define it in the command line:

```
#### Manual Bank Definition: 0x08100000 is the 1MB offset for Bank 1.
openocd \
  -f interface/stlink.cfg \
  -f target/stm32h7x.cfg \
  -c "flash bank bank1 stm32h7x 0x08100000 0x100000 0 0 stm32h7x.cpu0" \
  -c "init; program golden_master.bin verify reset exit 0x08000000"
```

Once the command finishes with **"Verified OK"**, the new board is a perfect bit-for-bit clone of your master prototype.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





UPDATING FIRMWARE VIA OPENOCD

While **Cloning** captures the entire state of a flight controller (2MB), a standard **Firmware Update** only overwrites the code itself. This guide explains how to use OpenOCD to flash new ArduPilot builds while ensuring your hard-earned calibrations and parameters ("User Data") remain untouched.

---

1. GENERATING THE FIRMWARE (THE BUILD)

Before you can flash, you must compile the ArduPilot source code for your specific hardware. ArduPilot uses the **Waf** build system.

1.1 CONFIGURE THE BOARD

First, tell the build system which board you are using. For the MicoAir 743, use the following command from the root of the `ardupilot` directory:

```
#### Configuration for MicoAir 743
./waf configure --board MicoAir743
```

Note: You only need to run this once, or whenever you change board types.

1.2 COMPILE THE BINARY

Next, run the build command for your vehicle type (Plane, Copter, Rover, etc.):

```
#### Build ArduPlane
./waf plane

#### Or for ArduCopter:
#### ./waf copter
```

1.3 LOCATE THE ARTIFACTS

Once the build completes, your files will be located in:
`build/MicoAir743/bin/`

For OpenOCD flashing, we specifically want the **`arduplane_with_bl.hex`** file. This version includes the ArduPilot bootloader, making it "self-healing" if the board's existing bootloader is missing or corrupted.

---

2. THE DISTINCTION: UPDATES VS. CLONING

It is vital to understand what happens to your memory during different operations:

OPERATION	FILE TYPE	DATA COVERED	EFFECT ON PARAMETERS
Cloning	`.bin` (2MB)	100\% of Flash	**Overwrites All** (Resets to Master)
Updating	`.hex` (~4MB)	~60\% of Flash	**Preserves User Data**

WHY IS THE `.HEX` FILE SO BIG?

You may notice that `arduplane_with_bl.hex` is roughly **4MB**, which is larger than the **2MB** physical capacity of the chip. This is normal. The Intel HEX format is a **text-based encoding**. Every 1 byte of binary data is represented by multiple ASCII characters. When OpenOCD flashes the file, it extracts the ~1.4MB of binary data and ignores the text overhead.

---

2. WHY USER DATA IS SAFE

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





ArduPilot allocates specific sectors at the **very end** of the flash memory for parameter storage (User Data).

- **Firmware Range:** Typically `0x08000000` to `0x08160000`.
- **User Data Range:** Typically `0x081FE000` to `0x081FFFFF`.

Because OpenOCD's `program` command is **sector-aware**, it only erases the areas where the firmware has data. Since the firmware binary does not reach the final sectors of the chip, your calibrations are never touched.

---

3. THE "1MB BORDER" REQUIREMENT

Even though we are not dumping the full 2MB, the manual bank definition is **still required** for the H743.

ArduPilot firmware for the H7 is usually ~1.3MB. This means the code starts in **Bank 0** but "crosses the border" into **Bank 1** at the 1024KB mark. If you do not define `bank1`, OpenOCD will crash the moment it tries to write the 1,024,001st byte.

---

4. THE UPDATE COMMAND

Use this command to update your firmware. Note that we do **not** provide a start address (`0x08000000`) at the end because `.hex` files have their own target addresses built-in.

```
#### Manual Bank Definition is still required to cross the 1MB boundary
openocd \
  -f interface/stlink.cfg \
  -f target/stm32h7x.cfg \
  -c "flash bank bank1 stm32h7x 0x08100000 0x100000 0 0 stm32h7x.cpu0" \
  -c "init; program arduplane_with_bl.hex verify reset exit"
```

SUCCESS INDICATORS:

1. **Device Info:** "flash size probed value 2048k".
2. **Programming Started:** OpenOCD will report "Device: STM32H74x/75x".
3. **Verified OK:** The new code is verified.
4. **Reset:** The board reboots and your parameters are still there.

---

5. WHEN TO USE `.APJ` VS `.HEX`

- **Mission Planner / QGC:** Use the `.apj` file. It contains metadata for the GCS.
- **OpenOCD / ST-Link:** Use the `_with_bl.hex` file. It includes the bootloader, ensuring the board can start even if the previous firmware was corrupted.

```

```


Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





CHAPTER 18: REMOTE ID

REMOTE ID: CORE CONCEPTS & REGULATIONS

EXECUTIVE SUMMARY

Remote ID (14 CFR Part 89) is a "digital license plate" for drones. It broadcasts the drone's position, altitude, and ID to nearby receivers.

For ArduPilot integrators, the critical distinction is between **Standard Remote ID** (deeply integrated, allows advanced ops) and **Broadcast Modules** (velcro-on, limits operations).

LEGAL CLASSES (FAA PART 89)

1. STANDARD REMOTE ID DRONE

- **Definition:** The drone was produced with Remote ID built-in.
- **Integration:** The Flight Controller (ArduPilot) communicates directly with the Remote ID transmitter.
- **Capability:** Allows for Beyond Visual Line of Sight (BVLOS) operations (with appropriate waivers).
- **Requirement:** The drone manufacturer must submit a **Declaration of Compliance (DOC)** to the FAA asserting the system meets ASTM F3411-22a.
- **Tamper Resistance:** The system must verify the functionality of the Remote ID subsystem *before* arming.

2. REMOTE ID BROADCAST MODULE

- **Definition:** A standalone device attached to an existing drone.
- **Integration:** None. It has its own GPS and battery. ArduPilot is unaware of it.
- **Limitations: Strictly Visual Line of Sight (VLOS) only.**
- **Use Case:** Retrofitting older fleets or hobbyist builds.

TECHNICAL STANDARDS

ASTM F3411-22A

This is the technical standard the FAA references. ArduPilot's `AP_OpenDroneID` library is designed to meet this standard.

- **Message Protocol:** Bluetooth 4.0/5.x or Wi-Fi (NaN/Beacon).
- **Update Rate:** 1 Hz.
- **Latency:** < 1 second.
- **Barometric Altitude:** Must be referenced to standard pressure (1013.25 mb).

CODEBASE INVESTIGATION

1. THE CORE LIBRARY: `AP_OPENDRONEID`

Located in `libraries/AP_OpenDroneID/AP_OpenDroneID.cpp`.

- **Rate Limiting:** Dynamic messages (Location) are sent at 1Hz (`_mavlink_dynamic_period_ms`).
- **State Machine:** `send_static_out()` cycles through `NEXT_MSG_BASIC_ID` , `NEXT_MSG_SYSTEM` , etc.

2. TAMPER RESISTANCE

ArduPilot enforces the "Tamper Resistance" requirement via `AP_Arming` .

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- If `DID_ENABLE` is 1 and `DID_OPTIONS` includes "Enforce Arming", the drone **will not arm** if the Remote ID module is disconnected or reports a failure.
- This satisfies the FAA requirement that the drone cannot takeoff if Remote ID is not functioning.

HOW TO: VERIFY COMPLIANCE

1. CHECK PRE-ARM STATUS

- **Error:** "RemoteID: System not available" means the Operator Location hasn't been received from the GCS yet.

2. VERIFY OUTPUT

- Open **Mission Planner > MAVLink Inspector**.
- Look for `OPEN_DRONE_ID_LOCATION` messages.
- **Check `status` field:**
  - `2`: Airborne (Armed)
  - `3`: Emergency (Crashed/Failsafe)

HARDWARE INTEGRATION

EXECUTIVE SUMMARY

ArduPilot supports Remote ID hardware via two primary protocols: **MAVLink (Serial)** and **DroneCAN**. The hardware acts as the transmitter (WiFi/Bluetooth), while the flight controller acts as the data source (GPS, Telemetry).

THEORY & CONCEPTS

1. MAVLINK VS. DRONECAN

- **MAVLink:** The module connects to a UART. ArduPilot pushes `OPEN_DRONE_ID_*` MAVLink messages to it.
  - *Pros:* Simple to debug (readable messages).
  - *Cons:* Burns a UART, slightly higher CPU overhead for framing.
- **DroneCAN:** The module connects to the CAN bus. ArduPilot publishes `dronecan.[remoteid](/field-manual/remote-id/core-concepts-and-regulations.html).*` messages.
  - *Pros:* Robust bus, daisy-chainable, no UART needed.
  - *Cons:* Requires DroneCAN-capable hardware.

2. THE FEEDBACK LOOP

Communication isn't one-way. The transmitter sends back `ARM_STATUS` messages to ArduPilot. If the transmitter says "Error" or stops talking, ArduPilot prevents arming (if `DID_OPTIONS` EnforceArming is set).

CODEBASE INVESTIGATION

1. DRONECAN BACKEND

Located in `libraries/AP_OpenDroneID/AP_OpenDroneID_DroneCAN.cpp`.

- Uses `Canard::Publisher` to broadcast:
  - `dronecan_remoteid_Location` (High Rate)
  - `dronecan_remoteid_BasicID` (Low Rate)
- Receives `dronecan_remoteid_ArmStatus` via callback `handle_arm_status`.

2. MAVLINK ROUTING

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





Located in `libraries/AP_OpenDroneID/AP_OpenDroneID.cpp`.

- Checks `_mav_port` parameter to determine which Serial port to use.
- Uses standard `mavlink_msg_open_drone_id_location_send_struct()` functions.

SOURCE CODE REFERENCE

- **DroneCAN:** `libraries/AP_OpenDroneID/AP_OpenDroneID_DroneCAN.cpp`

HOW TO: SET UP CUBE ID (DRONECAN)

1. WIRING

Connect the Cube ID to the **CAN1** or **CAN2** port on your flight controller using the 4-pin JST-GH cable.

2. CONFIGURATION

Set the following parameters:

- **`CAN_P1_DRIVER`**: `1` (Enable Driver 1)
- **`CAN_D1_PROTOCOL`**: `1` (DroneCAN)
- **`DID_ENABLE`**: `1`
- **`DID_CANDRIVER`**: `1` (Use CAN Driver 1)
- **`DID_MAVPORT`**: `-1` (Disable Serial)

3. VERIFICATION

1. Reboot.
2. Go to **Setup > Optional Hardware > UAVCAN**.
3. Click **SLCAN Mode CAN1**.
4. You should see the Cube ID appear in the node list.
5. Check the MAVLink Inspector for `OPEN_DRONE_ID_ARM_STATUS`. `Status` should be `0` (Good to Arm) after GPS lock is obtained.

ARDUPILOT CONFIGURATION

EXECUTIVE SUMMARY

Configuring Remote ID involves enabling the subsystem (`DID_ENABLE`), selecting the communication backend (Serial or CAN), and verifying the persistent "Tamper-Proof" ID parameters.

Unlike standard parameters, the UAS ID is often stored in a secure, persistent memory area that survives parameter resets.

THEORY & CONCEPTS

1. THE SUBSYSTEM (`AP_OPENDRONEID`)

Enabling `DID_ENABLE` allocates the `AP_OpenDroneID` singleton. This orchestrates the data flow:

- Fetches GPS/Baro data.
- Formats it into ODID packets.
- Routes it to the selected output driver.

2. PERSISTENT PARAMETERS

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





To comply with "Tamper Resistance" requirements, the drone's unique ID (`UAS_ID`) is often stored in the bootloader's persistent storage, not the main EEPROM.

- **Implication:** You might not see `DID_UAS_ID` in the full parameter list if it hasn't been set yet.
- **Mechanism:** ArduPilot checks `hal.util->get_persistent_param_by_name()` on boot.

CODEBASE INVESTIGATION

1. PERSISTENT LOADING: `LOAD_UAS_ID_FROM_PERSISTENT_MEMORY()`

Located in `libraries/AP_OpenDroneID/AP_OpenDroneID.cpp`.

- It looks for keys: `"DID_UAS_ID"`, `"DID_UAS_ID_TYPE"`, `"DID_UA_TYPE"`.
- If found, it populates the Basic ID message automatically.

2. PARAMETER TABLE

- **`DID_ENABLE`**: Master switch.
- **`DID_MAVPORT`**: Serial port selection (-1 to disable).
- **`DID_CANDRIVER`**: DroneCAN driver index (0 to disable).
- **`DID_OPTIONS`**: Bitmask for enforcement (e.g., prevent arming if ID fails).

SOURCE CODE REFERENCE

- **Parameter Definitions:** `libraries/AP_OpenDroneID/AP_OpenDroneID.cpp`

HOW TO: CONFIGURE REMOTE ID PARAMETERS

1. ENABLE THE SYSTEM

1. Set **`DID_ENABLE`** to `1`.
2. Reboot the Flight Controller.

2. SET THE ID (THE TRICKY PART)

If your module is "Standard Remote ID" (integrated), you must provide the ID.

- **Option A (Mavlink):** Use Mission Planner's "Remote ID" tab (if available).
- **Option B (Parameters):**
  - Set **DID_UA_TYPE**: `1` (Aeroplane) or `2` (Copter).
  - Set **DID_UAS_ID_TYPE**: `1` (Serial Number) or `2` (CAA ID).
  - Set **DID_UAS_ID**: The actual string (e.g., `1596328479815`).
  - *Note: These might need to be set via the "Full Parameter Tree" or a script if they are hidden.*

3. OPERATOR ID

This is **not** persistent on the drone (usually). It is injected by the GCS (Ground Control Station) upon connection.

- **Mission Planner:** Config/Tuning → User Info → Operator ID.
- **QGroundControl:** Application Settings → General → Remote ID.
- *Without this step, you will likely get a "Remote ID: System not available" arming error.*

OPENDRONEID & ANDROID INTEGRATION

EXECUTIVE SUMMARY

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY



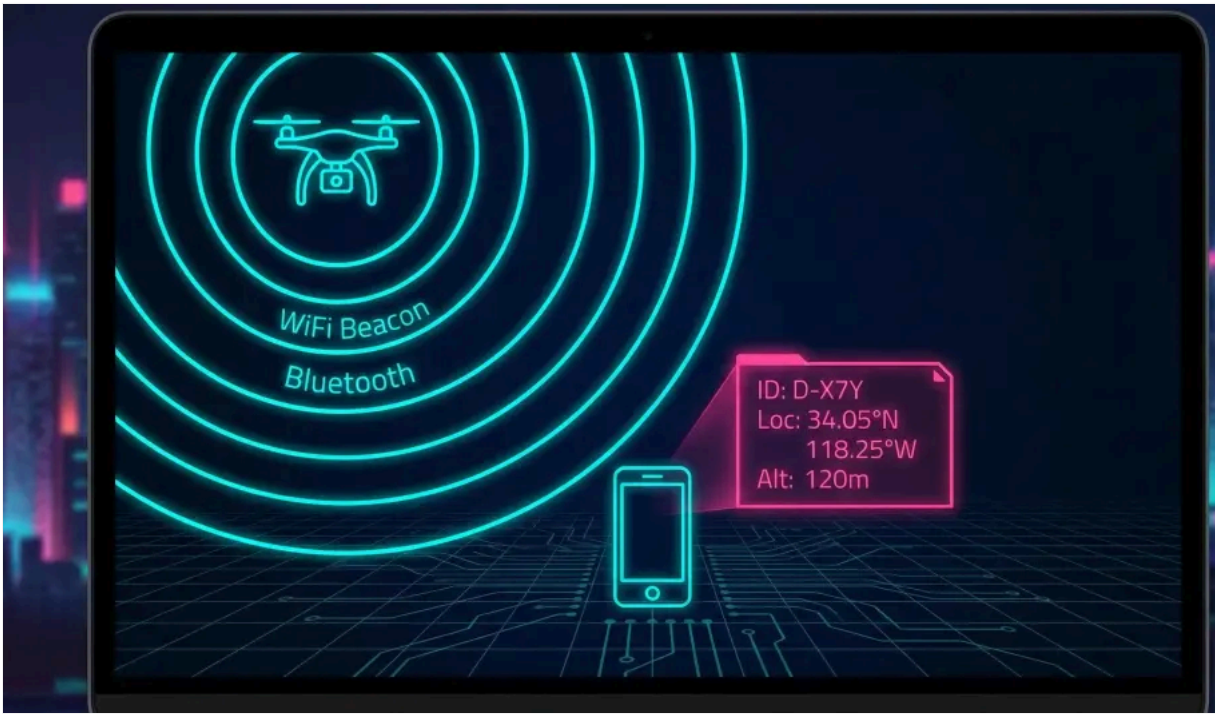


OpenDroneID acts as a digital beacon. Any smartphone within range can pick up the signal using the free OpenDroneID app. This provides public accountability and situational awareness.

THEORY & CONCEPTS

1. THE BROADCAST MEDIUMS

Standard Remote ID modules must broadcast on specific frequencies:



- **Wi-Fi Beacon (NaN):** Uses the "Neighbor Awareness Networking" protocol. High bandwidth, medium range.
- **Bluetooth 4.0 (Legacy):** Short range, widely compatible.
- **Bluetooth 5.0 (Long Range):** Can reach up to 1km+, but requires modern hardware.

2. WHAT THE PUBLIC SEES

Anyone with the app sees:

- **Drone Position:** Lat/Lon, Altitude (Height), Direction.
- **Velocity:** Horizontal and Vertical speed.
- **Operator Position:** Where the pilot is standing.
- **ID:** The Serial Number or Session ID.

CODEBASE INVESTIGATION

1. DATA LIMITS: `AP_OPENDRONEID.H`

Located in `libraries/AP_OpenDroneID/AP_OpenDroneID.h`.

- **Speed Cap:** `ODID_MAX_SPEED_H` is 254.25 m/s. Anything faster is clipped.
- **Altitude Cap:** `ODID_MAX_ALT` is 31767.5 m.
- **Direction:** 0-360 degrees.

2. MAVLINK HUD INTEGRATION?

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

Currently, MAVLink HUD acts as a GCS for *your* drone. It does not contain an SDR to sniff *other* drones.

- **Potential Future:** If the connected radio (e.g., TBS Crossfire or ELRS) reported nearby traffic via ADSB-style messages, the HUD could display them.
- **Current State:** You rely on the separate OpenDroneID app to see others.

HOW TO: TEST VISIBILITY

1. DOWNLOAD THE APP

Get the "OpenDroneID" app from the Google Play Store (or GitHub).

2. POWER UP

Power your drone (with props off). Wait for GPS lock.

3. SCAN

Open the app on your phone.

- **Result:** You should see your drone's icon appear on the map.
- **Details:** Tap the icon to see the "Self ID" text and your Operator ID.
- **Troubleshooting:** If you don't see it, ensure your phone supports Bluetooth 5.0 or Wi-Fi NAN, and that you are physically close to the module.

TROUBLESHOOTING & COMPLIANCE

EXECUTIVE SUMMARY

Remote ID adds a layer of complexity to the arming process. Because it is a legal requirement in many jurisdictions, ArduPilot treats it as a critical safety system by default. Most issues stem from **Hardware Connection** or **Operator Location** data.

THEORY & CONCEPTS

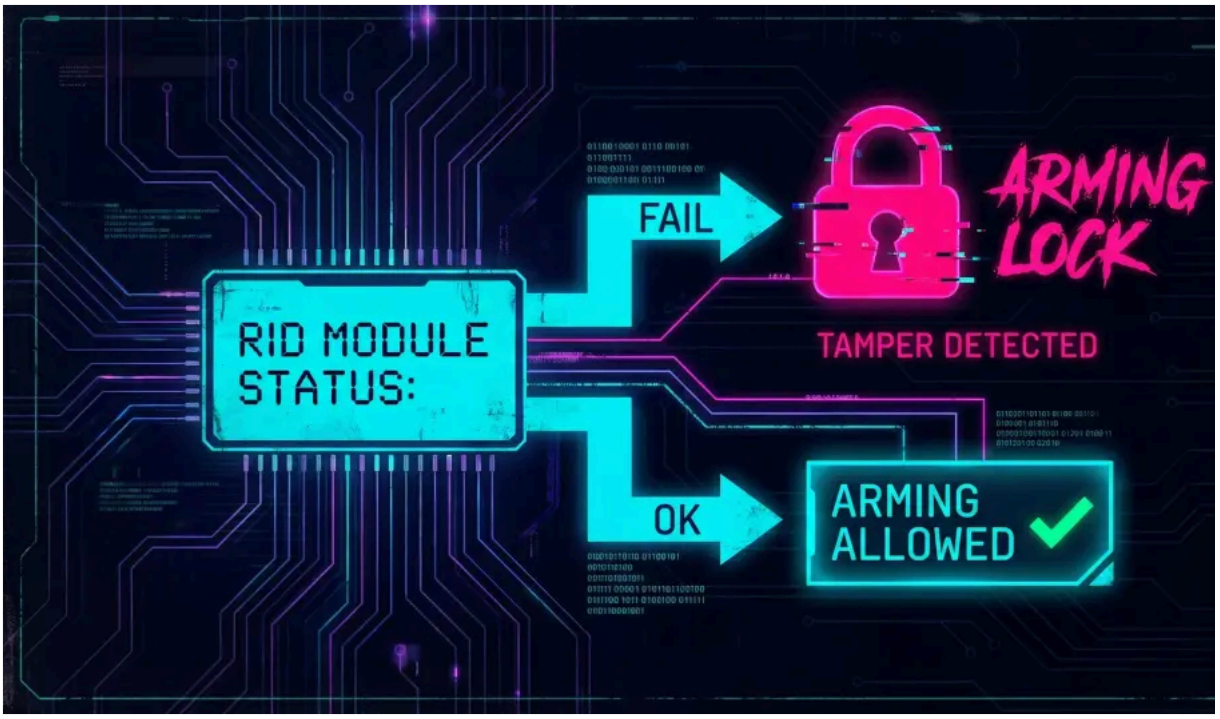
1. THE ARMING CHECK

When `DID_ENABLE` is true and `DID_OPTIONS` (bit 0) is set to **EnforceArming**, ArduPilot will prevent arming if the Remote ID system is unhealthy.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY







- **Health Criteria:**
  - Hardware connected and talking.
  - Drone GPS Location valid (3D Fix).
  - Operator Location valid (received from GCS).
  - Basic ID configured (UA Type, ID Type, Serial Number).

CODEBASE INVESTIGATION

1. ARMING CHECKS: `AP_ARMING::OPENDRONEID_CHECKS()`

Located in `libraries/AP_Arming/AP_Arming.cpp`.

- Calls `AP_OpenDroneID::pre_arm_check`.
- **Common Error:** `"[RemoteID](/field-manual/remote-id/core-concepts-and-regulations.html): System not available"`.
  - **Cause:** The GCS hasn't sent the Operator Location packet (`OPEN_DRONE_ID_SYSTEM_UPDATE`).
  - **Fix:** Ensure your phone/tablet has GPS enabled and is connected to the drone.

2. HARDWARE MONITORING

Located in `libraries/AP_OpenDroneID/AP_OpenDroneID.cpp`.

- **Error:** `"[ODID](/field-manual/remote-id/core-concepts-and-regulations.html): lost transmitter"`.
  - **Cause:** No heartbeat from the RID module for > 5 seconds.
  - **Fix:** Check wiring (RX/TX swapped?), Baud rate (`SERIALx_BAUD`), or CAN termination.

HOW TO: TROUBLESHOOT COMMON ERRORS

1. "REMOTEID: SYSTEM NOT AVAILABLE"

- **Meaning:** ArduPilot doesn't know where *you* (the pilot) are.
- **Fix:** Connect Mission Planner or QGC. Wait for the GCS to get a GPS lock on your computer/phone. The GCS automatically sends this to the drone.

2. "UA_TYPE REQUIRED IN BASICID"

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

- **Meaning:** You haven't told ArduPilot what kind of drone this is.
- **Fix:** Set **DID_UA_TYPE** to `2` (Multicopter) or `1` (Aeroplane).

3. "REMOTEID: ARM STATUS FAILURE"

- **Meaning:** The Remote ID hardware itself is reporting an internal error (e.g., its own GPS is bad, or internal self-test failed).
- **Fix:** Check the module's LEDs/documentation.

COMPLIANCE CHECKLIST

1. **Hardware:** Module installed and powered.
2. **Config:** `DID_ENABLE=1`, `DID_CANDRIVER` or `DID_MAVPORT` set.
3. **Identity:** `DID_UAS_ID` matches your label.
4. **GCS:** Connected and providing Operator Location.
5. **Test:** Use the OpenDroneID Android app to verify you are broadcasting before takeoff.

COMMERCIAL CERTIFICATION & SELF-DECLARATIONS

EXECUTIVE SUMMARY

Building a drone with ArduPilot and a Remote ID module does **not** automatically make it a "Standard Remote ID Drone" in the eyes of the FAA.

To sell a "Standard Remote ID Drone," you (the integrator) become the **Manufacturer**. You must submit a **Declaration of Compliance (DOC)** to the FAA, asserting that your specific hardware/software combination meets ASTM standards.

THE SELF-CERTIFICATION TRAP

It is easy to create a DOC on the FAA DroneZone. It is harder to defend it in court.

- **The Risk:** If your drone is involved in an incident and investigators find your Remote ID implementation was non-compliant (e.g., could be disabled by a user switch), you are liable for falsifying federal records.
- **Retroactive Rigor:** "It worked on my bench" is not a defense. You need engineering logs proving the system refuses to arm when the ID module is disconnected.

TECHNICAL REQUIREMENTS FOR DOC

1. TAMPER RESISTANCE

The regulations state the Remote ID system cannot be disabled by the user without "special tools" or "breaking the case."

- **ArduPilot Implementation:** Use `DID_OPTIONS = 1` (Enforce Arming).
- **Parameter Locking:** You should "hide" or "read-only" the `DID_ENABLE` parameter in your custom parameter set to prevent users from turning it off.

2. SERIAL NUMBER INTEGRITY

The broadcasted Serial Number must match the physical label on the drone.

- **Standard:** ANSI/CTA-2063-A.
- **Provisioning:** You must burn this ID into the persistent storage (`DID_UAS_ID`).
- **Locking:** Once set, the ID should not be user-changeable. ArduPilot supports locking the ID via the `LockUASIDOnFirstBasicIDRx` option.

3. ERROR REPORTING

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





The system must notify the pilot if Remote ID fails.

- **ArduPilot:** Sends `STATUSTEXT` messages ("RemoteID: Failed") to the GCS. You must ensure your GCS displays these prominently (Red Banner).

THE CERTIFICATION WORKFLOW

1. **Engineering Test:** Validate that disconnecting the CAN cable prevents arming. Validate that GPS loss flags the Remote ID status as "Emergency" or "Undeclared" correctly.
2. **Lab Test:** Use a spectrum analyzer (or a certified lab) to verify the Bluetooth/WiFi RF power output meets FCC Part 15.
3. **Documentation:** Create a "Means of Compliance" (MOC) document tracking your testing.
4. **Submission:** Submit the DOC to FAA DroneZone.
5. **Production:** Every unit sold must have the specific Serial Number format listed in your DOC.

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





CONDITION_DELAY (112)

SUMMARY

The `CONDITION_DELAY` command pauses the mission state machine for a specific number of seconds. Unlike a `NAV_DELAY` or a loiter with time, which are navigation-level commands, `CONDITION_DELAY` is a flow-control item that prevents the *next* mission item from starting until the timer expires.

STATUS

Supported (All Vehicles)

DIRECTIONALITY

- **RX (Receive):** The vehicle receives this command as part of a mission upload.

MISSION STORAGE (AP_MISSION)

- **Packet Param 1 (Time):** Stored in the internal `condition_value` field (seconds).
- **Mechanism:** When the mission reaches this item, it records the current system time in `condition_start`.

EXECUTION (ENGINEER'S VIEW)

STATE MACHINE LOGIC

ArduPilot's mission engine treats "Condition" commands as gates.

1. **Verification:** Every loop, the `verify_wait_delay()` function is called (mode_auto.cpp).
2. **Comparison:** The function checks if the elapsed time since the command started exceeds the target delay.

$$\text{current_time} - \text{condition_start} > \text{condition_value} \cdot 1000$$

3. **Completion:** Once the condition is met, the gate opens, and the mission state machine increments to the next command ID.

VEHICLE BEHAVIOR DURING DELAY

It is critical to understand that `CONDITION_DELAY` **does not stop the vehicle's movement** if it was already performing a navigation task that supports concurrent execution (though typically conditions are placed *between* nav points).

- **Copter:** If placed after a `NAV_WAYPOINT`, the drone will hover at that waypoint while the delay runs.
- **Plane:** The aircraft will continue its previous navigation state (e.g., loitering) while the condition timer counts down.

DATA FIELDS (MAVLINK)

- `param1` **(Time):** Delay time in seconds.
- `param2` to `param7`: Unused.

THEORY: SYNCHRONOUS VS. ASYNCHRONOUS DELAYS

In computer science, a delay can be blocking or non-blocking.

- **MAVLink HUD Context:** `CONDITION_DELAY` is a non-blocking delay for the flight controller (it still stabilizes the airframe), but a blocking delay for the **Mission Script**.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- **Race Conditions:** If a GCS sends a "Set Current Waypoint" command while a `CONDITION_DELAY` is active, the timer is typically reset or abandoned as the mission engine jumps to the new index.

PRACTICAL USE CASES

1. **Camera Boot-up:**
   - *Scenario:* A high-end mapping camera takes 10 seconds to initialize after being powered on via a relay.
   - *Action:* `DO_SET_RELAY` → `CONDITION_DELAY (10s)` → `NAV_WAYPOINT`.
2. **Safety Buffer:**
   - *Scenario:* Ensuring a drone has come to a complete, stable hover before dropping a payload.
   - *Action:* `NAV_WAYPOINT` → `CONDITION_DELAY (3s)` → `DO_GRIPPER`.

KEY PARAMETERS

- `MIS_OPTIONS`: Can affect how the mission engine handles pauses and restarts.

KEY CODEBASE LOCATIONS

- **ArduCopter/mode_auto.cpp:2192**: `verify_wait_delay` implementation.
- **ArduPlane/commands_logic.cpp**: Plane condition handling.

CONDITION_DISTANCE (114)

SUMMARY

The `CONDITION_DISTANCE` command creates a proximity-based gate in the mission. It prevents the next mission item from starting until the vehicle is within a specific distance (meters) of the *next* navigation waypoint.

STATUS

Supported (All Vehicles)

DIRECTIONALITY

- **RX (Receive):** The vehicle receives this command as part of a mission upload.

MISSION STORAGE (AP_MISSION)

- **Packet Param 1 (Distance):** Stored in the internal `condition_value` field (meters).

EXECUTION (ENGINEER'S VIEW)

VERIFICATION LOGIC

The mission engine checks the proximity in every loop cycle using `verify_within_distance()` (mode_auto.cpp).

1. **Distance Calculation:** The flight controller calculates the 2D (horizontal) distance between the current GPS coordinate and the coordinate of the `NAV_WAYPOINT` immediately following the condition.
2. **Comparison:**

$$\text{dist_to_wp} < \text{condition_value}$$

3. **Completion:** As soon as the vehicle crosses the "distance threshold," the condition returns true, and the next command (which is usually a "Do" command like `DO_DIGICAM_CONTROL`) is executed.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





DATA FIELDS (MAVLINK)

- `param1` **(Distance):** Distance in meters.
- `param2` to `param7`: Unused.

THEORY: THE "INNER CIRCLE"

`CONDITION_DISTANCE` allows for spatial triggering that is more precise than simple waypoint arrival.

- **Waypoint Acceptance:** Standard waypoints use a radius (e.g., `WP_RADIUS`) to determine when to turn.
- **Condition Precision:** By using `CONDITION_DISTANCE`, you can trigger an action (like a laser fire or camera snap) at a much tighter distance (e.g., 1 meter) than the navigation waypoint's acceptance radius (e.g., 5 meters).

PRACTICAL USE CASES

1. **Precise Photo Capture:**
   - *Scenario:* A surveyor needs a photo taken exactly as the drone passes over a marker.
   - *Action:* `NAV_WAYPOINT (Marker)` → `CONDITION_DISTANCE (0.5m)` → `DO_DIGICAM_CONTROL`.
2. **Obstacle Proximity Actions:**
   - *Scenario:* A drone needs to turn on high-intensity landing lights as it approaches a narrow dock.
   - *Action:* `NAV_WAYPOINT (Dock)` → `CONDITION_DISTANCE (10m)` → `[DO_SET_SERVO](/mission-planning/do-commands.html#DO_SET_SERVO) (Lights ON)`.

KEY PARAMETERS

- `WPNAV_RADIUS`: (Copter) The horizontal radius used for waypoint completion; often used in conjunction with `CONDITION_DISTANCE`.

KEY CODEBASE LOCATIONS

- **ArduCopter/mode_auto.cpp:2201**: `verify_within_distance` implementation.
- **ArduPlane/commands_logic.cpp**: Plane condition handler.

CONDITION_YAW (115)

SUMMARY

The `CONDITION_YAW` command forces the vehicle to rotate to a specific heading. This command is a "blocking" mission item; the mission will not advance to the next item until the aircraft has successfully achieved the target yaw angle within a small tolerance.

STATUS

Supported (ArduCopter and QuadPlane Only)

DIRECTIONALITY

- **RX (Receive):** The vehicle receives this command as part of a mission upload.

MISSION STORAGE (AP_MISSION)

- **Angle (Param 1):** Target angle in degrees.
- **Rate (Param 2):** Rotation speed in deg/s.
- **Direction (Param 3):** -1 for Counter-Clockwise, 1 for Clockwise.
- **Relative (Param 4):** 0 for Absolute (North-up), 1 for Relative (Offset from current heading).

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





EXECUTION (ENGINEER'S VIEW)

ARDUCOPTER IMPLEMENTATION

In Copter, the command triggers `ModeAuto::do_yaw` (mode_auto.cpp).

1. **Request:** The command is passed to the `auto_yaw` controller, which sets a `FIXED` yaw target.
2. **Tracking:** The autopilot calculates the shortest path (or the path requested by the direction parameter) to the new heading.
3. **Completion:** The verification function `ModeAuto::verify_yaw()` checks if the current heading matches the target:
$$|\text{Yaw}_{current} - \text{Yaw}_{target}| < 2^\circ$$
4. **Priority:** Note that if a subsequent `NAV_WAYPOINT` command has a different yaw requirement (like "Face Next Waypoint"), it will override the `CONDITION_YAW` once the condition is met and the mission advances.

DATA FIELDS (MAVLINK)

- `param1` **(Angle):** Target angle in degrees [0-360].
- `param2` **(Rate):** Desired turn rate in deg/s.
- `param3` **(Direction):** -1: CCW, 1: CW.
- `param4` **(Relative):** 0: Absolute, 1: Relative.
- `param5` to `param7`: Unused.

THEORY: THE SINGULARITY OF NORTH

Yaw management in ArduPilot must account for the 360/0 degree wrap-around.

- **Angular Error:** The autopilot uses the `wrap_180()` function to calculate the smallest error.
- **Magnetic vs. True:** Mission angles are typically relative to **Magnetic North** unless the system has been configured with a high-precision GPS (Moving Baseline) or a specific declination offset.

PRACTICAL USE CASES

1. **Fixed-Camera Alignment:**
   - *Scenario:* A drone is inspecting a solar panel array. The camera is fixed (no gimbal).
   - *Action:* `CONDITION_YAW` is used to point the entire drone airframe at the panels before the next survey leg.
2. **Antenna Tracking:**
   - *Scenario:* A drone needs to point a high-gain directional antenna back at the GCS for a data burst.
   - *Action:* `CONDITION_YAW` targets the bearing to the Home point.

KEY PARAMETERS

- `ATC_SLEW_YAW`: Limits the maximum rate of change for yaw to prevent mechanical stress on the frame.
- `WP_YAW_BEHAVE`: Determines how the drone handles yaw during navigation *between* `CONDITION_YAW` commands.

KEY CODEBASE LOCATIONS

- **ArduCopter/mode_auto.cpp:1928**: `do_yaw` implementation.
- **ArduCopter/mode_auto.cpp:2211**: `verify_yaw` completion check.

DO_AUTOTUNE_ENABLE (212)

SUMMARY

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





The `DO_AUTOTUNE_ENABLE` command triggers ArduPilot's automated PID tuning process during a mission. This allows the vehicle to optimize its control gains in real-time while flying a specific leg of a mission, which is useful for airframes whose dynamics change significantly with different payloads.

STATUS

Supported (ArduPlane Only)

DIRECTIONALITY

- **RX (Receive):** The vehicle receives this command as part of a mission upload or as a direct command from the GCS.

MISSION STORAGE (AP_MISSION)

- **Enable (Param 1):**
  - 0: Disable Autotune.
  - 1: Enable Autotune.
- **Packing:** Stored in the internal `p1` field.

EXECUTION (ENGINEER'S VIEW)

TUNING LOGIC

The command utilizes the `AP_AutoTune` library (GCS_Mavlink.cpp:1207).

1. **Safety Verification:** The aircraft must be in a stable flight state (typically FBWA or Cruise) for Autotune to be effective.
2. **Perturbation:** When enabled, the autopilot injects small, controlled step-inputs into the Roll and Pitch axes.
3. **Analysis:** It monitors the vehicle's response (Rate and Acceleration) to calculate the ideal P, I, and D gains.
4. **Completion:** The mission advances immediately. The Autotune process runs in the background. It is common to follow this command with a long, straight `WAYPOINT` to give the tuner enough time to converge.

DATA FIELDS (MAVLINK)

- `param1` **(State):** 0: Disable, 1: Enable.
- `param2` to `param7`: Unused.

THEORY: SYSTEM IDENTIFICATION

AutoTune is a form of **Online System Identification**.

- **Excitation:** The step-inputs "excite" the airframe's natural frequencies.
- **Damping Ratio:** The algorithm looks for the "Overshoot" and "Settling Time" to ensure the resulting PID gains provide a damping ratio (ζ) near 0.707 (Critically Damped).
- **Risk:** If Autotune is enabled on a poorly balanced or structurally weak airframe, the perturbations can trigger oscillations. ArduPilot's tuner includes "divergence protection" to automatically abort if the aircraft's attitude becomes unstable.

PRACTICAL USE CASES

1. **Post-Payload Calibration:**
   - *Scenario:* A plane drops a heavy 2kg sensor mid-flight.
   - *Action:* The mission triggers `DO_AUTOTUNE_ENABLE (1)` on the return leg to re-calibrate the PIDs for the now much lighter (and potentially differently balanced) airframe.
2. **Maiden Flight Script:**
   - *Scenario:* Automating the first flight of a new aircraft.
   - *Action:* Mission includes a 5km straight leg with Autotune enabled to ensure a perfect tune before the first landing.

KEY PARAMETERS

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- `AUTOTUNE_LEVEL` : Defines how "aggressive" the resulting tune should be.
- `AUTOTUNE_AXES` : Selection of which axes to tune (Roll, Pitch, Yaw).

KEY CODEBASE LOCATIONS

- **ArduPlane/GCS_Mavlink.cpp:1207**: Autotune trigger logic.
- **libraries/AP_AutoTune/AP_AutoTune.cpp**: The core gain-calculation engine.

DO_AUX_FUNCTION (218)

SUMMARY

The `DO_AUX_FUNCTION` command allows a mission to trigger any of ArduPilot's numerous "Auxiliary Functions." These are the same functions typically assigned to physical switches on a radio transmitter (e.g., "Save Waypoint," "Camera Trigger," "Arm/Disarm"). This command effectively allows the mission to "flick a virtual switch."

STATUS

Supported (All Vehicles)

DIRECTIONALITY

- **RX (Receive):** The vehicle receives this command as part of a mission upload or as a direct command.

MISSION STORAGE (AP_MISSION)

- **Function (Param 1):** The ID of the auxiliary function (e.g., 7 = Save Waypoint).
- **Switch Position (Param 2):**
  - 0: Low (Off).
  - 1: Middle.
  - 2: High (On).
- **Packing:** Stored in the `auxfunction` content struct.

EXECUTION (ENGINEER'S VIEW)

UNIVERSAL SWITCH LOGIC

Execution is handled by the `RC_Channel` library's auxiliary function dispatcher.

1. **Virtual Input:** ArduPilot treats the mission command as a virtual RC channel.
2. **Mapping:** The `Param 1` value is matched against the list of internal functions (e.g., `AUXSW_MOTOR_INTERLOCK`, `AUXSW_GRIPPER`).
3. **Action:** The autopilot executes the code associated with that switch transition (e.g., if set to `High`, it runs the `on_switch_high()` method for that function).

DATA FIELDS (MAVLINK)

- `param1` **(Function):** Auxiliary function ID.
- `param2` **(Position):** 0: Low, 1: Mid, 2: High.
- `param3` to `param7` : Unused.

THEORY: ABSTRACTED LOGIC

`DO_AUX_FUNCTION` represents the ultimate in **Logic Abstraction**.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- **The Power:** ArduPilot has over 100 aux functions. Instead of creating 100 individual mission commands, this one command provides access to all of them.
- **Unified Control:** It ensures that whether an action is triggered by a human on a radio, a GCS button, or an autonomous mission, it always calls the same verified block of C++ code.

PRACTICAL USE CASES

1. **Waypoint Saving:**
   - *Scenario:* A pilot is flying a survey and wants the mission to automatically record its own progress.
   - *Action:* `DO_AUX_FUNCTION (Function: 7, Position: 2)`.
2. **Emergency Stop (Motor Interlock):**
   - *Scenario:* A high-risk mission segment (e.g., flying through a tunnel).
   - *Action:* A script can inject `DO_AUX_FUNCTION (Function: 32, Position: 0)` to instantly kill the motors if a failure is detected.

KEY PARAMETERS

- `RCx_OPTION`: Used to identify the ID of specific functions.

KEY CODEBASE LOCATIONS

- **libraries/RC_Channel/RC_Channel.cpp**: Auxiliary function dispatcher.
- **libraries/AP_Mission/AP_Mission.cpp**: Command-to-RC-logic routing.

DO_CHANGE_SPEED (178)

SUMMARY

The `DO_CHANGE_SPEED` command allows the autopilot to dynamically adjust the vehicle's speed and throttle limits during a mission. This is essential for missions that require high-speed transit between work areas but slow, precise flight during data collection.

STATUS

Supported (All Vehicles)

DIRECTIONALITY

- **RX (Receive):** The vehicle receives this command as part of a mission upload or as a direct command (Guided Mode).

MISSION STORAGE (AP_MISSION)

- **Speed Type (Param 1):** 0 = Airspeed, 1 = Ground Speed.
- **Target Speed (Param 2):** Target speed in m/s.
- **Throttle (Param 3):** Desired throttle percentage (0-100).
- **Mechanism:** Stored in the internal `speed` content struct.

EXECUTION (ENGINEER'S VIEW)

ARDUCOPTER IMPLEMENTATION

In Copter, the command triggers `ModeAuto::do_change_speed` (mode_auto.cpp).

1. **WPNav Update:** The command updates the `AC_WPNav` library's internal speed variables.
2. **Overrides:**
   - **Horizontal:** Sets `set_speed_xy` (cm/s).

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- **Vertical:** If the type is set to climb or descent (ArduPilot extension), it updates `set_speed_up` or `set_speed_down`.
3. **Persistence:** The new speed remains in effect until the end of the mission or until another `DO_CHANGE_SPEED` command is encountered.

ARDUPLANE IMPLEMENTATION

In Plane, the command updates the navigation controller's target velocity.

1. **Airspeed vs. Groundspeed:**
   - If **Airspeed** is selected: The plane adjusts pitch and throttle to maintain the target airspeed (IAS/EAS).
   - If **Groundspeed** is selected: The plane uses its "Min Groundspeed" logic (`MIN_GNDSPD_CM`) to ensure it maintains progress against headwinds.
2. **Safety Limits:** The autopilot will always clamp the requested speed between `ARSPD_FBW_MIN` and `ARSPD_FBW_MAX`.

DATA FIELDS (MAVLINK)

- `param1` **(Type):** 0:Airspeed, 1:Groundspeed, 2:Climb speed, 3:Descent speed.
- `param2` **(Speed):** Target speed in m/s.
- `param3` **(Throttle):** Throttle setpoint (0-100). -1 to ignore.
- `param4` **(Relative):** 0: Absolute speed, 1: Offset from default.

THEORY: GROUNDSPEED VS. AIRSPEED

Understanding the difference is critical for safety:

- **Fixed-Wing:** Airspeed is what keeps you in the air (lift). Changing airspeed affects your stall margin. `DO_CHANGE_SPEED` in a plane is often used to fly slowly for photography or fast for "dash" legs.
- **Multicopter:** Multicopters primarily care about Groundspeed. Airspeed is only relevant in high-wind scenarios where "Lean Angle" limits (e.g., `ANGLE_MAX`) might prevent the drone from achieving the requested groundspeed.

PRACTICAL USE CASES

1. **Long Range Transit:**
   - *Scenario:* A delivery drone needs to fly 10km to a destination.
   - *Action:* `TAKEOFF` → `DO_CHANGE_SPEED (25 m/s)` → `WAYPOINT (Destination)`.
2. **Precision Mapping:**
   - *Scenario:* A high-resolution camera requires a slow ground speed to prevent motion blur.
   - *Action:* `DO_CHANGE_SPEED (5 m/s)` → `WAYPOINT (Start of Grid)`.

KEY PARAMETERS

- `WPNAV_SPEED`: (Copter) Default mission speed.
- `TRIM_ARSPD_CM`: (Plane) Default cruise airspeed.
- `ARSPD_FBW_MIN`: Absolute minimum airspeed safety limit.

KEY CODEBASE LOCATIONS

- **ArduCopter/mode_auto.cpp:1936**: Copter implementation.
- **ArduPlane/commands_logic.cpp**: Plane implementation.

DO_DIGICAM_CONFIGURE (202) / DO_DIGICAM_CONTROL (203)

SUMMARY

The `DO_DIGICAM_CONFIGURE` and `DO_DIGICAM_CONTROL` commands are the legacy interface for interacting with onboard digital cameras. While modern systems prefer the MAVLink Camera Protocol (v2), these commands remain vital for supporting a vast

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





array of existing hardware, including CHDK-enabled Canon cameras, Sony NEX series via IR/Multi-port, and simple PWM-triggered DSLRs.

STATUS

Legacy / Supported (All Vehicles with AP_Camera enabled)

DIRECTIONALITY

- **RX (Receive):** The vehicle receives this command as part of a mission upload or as a direct command.

MISSION STORAGE (AP_MISSION)

ArduPilot uses a highly efficient packing strategy to store these complex commands:

- **202 (Configure):** Stored in the `digicam_configure` struct. Includes Shutter Speed, Aperture, and ISO.
- **203 (Control):** Stored in the `digicam_control` struct. Includes Zoom absolute, Zoom step, and Focus locking.
- **Packing:** Both utilize the standard 15-byte mission storage limit by bit-packing smaller values where possible.

EXECUTION (ENGINEER'S VIEW)

TRIGGERING AND HANDSHAKING

The commands are routed through the `AP_Camera` library (AP_Camera.cpp).

1. **Direct Shutter:** If `Shooting Command (Param 5)` is set to 1, the autopilot triggers the shutter.
2. **Focus Lock:** ArduPilot supports a "Half-press" logic for cameras that require focusing before the final capture.
3. **Engine Cut-off:** A unique feature of `DO_DIGICAM_CONFIGURE` is the **Engine Cut-off Time (Param 7)**. For gas-powered vehicles with extreme vibration, the autopilot can momentarily kill the engine or reduce throttle during the capture to ensure a blur-free image.

DATA FIELDS (MAVLINK)

DO_DIGICAM_CONTROL (203)

- `param1` **(Session):** 0:Ignore, 1:Show Lens, 2:Hide Lens.
- `param2` **(Zoom):** Absolute position.
- `param3` **(Step):** Zoom step (offset).
- `param4` **(Focus):** 0:Ignore, 1:Lock, 2:Unlock.
- `param5` **(Shot):** 1:Take picture.

THEORY: THE LATENCY OF MECHANICAL SHUTTERS

Unlike digital sensors, mechanical shutters have **Trigger Latency** ($\Delta t_{shutter}$).

- **The Drift:** If a drone is flying at 20 m/s and the shutter has a 200ms lag, the photo will be taken 4 meters past the intended coordinate.
- **ArduPilot Compensation:** High-end configurations use the `CAM_FEEDBACK_PIN` (Hot Shoe). When the shutter actually fires, the camera sends a signal back to the flight controller, which then captures the *exact* GPS/IMU state for that microsecond, ensuring sub-centimeter mapping accuracy.

PRACTICAL USE CASES

1. **Vibration-Sensitive Long Exposure:**
   - *Scenario:* A high-altitude plane taking photos in low light.
   - *Action:* `DO_DIGICAM_CONFIGURE (Engine Cut-off: 0.5s)`. The motor stops for half a second, the photo is taken in still air, and the motor restarts automatically.
2. **Canon CHDK Integration:**
   - *Scenario:* Using a cheap Canon Point-and-Shoot for mapping.

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- *Action:* `DO_DIGICAM_CONTROL` sends the PWM pulse required by the CHDK script to trigger the capture.

KEY PARAMETERS

- `CAM_TRIGG_TYPE` : Must be set to `Relay` or `PWM`.
- `CAM_DURATION` : The length of the shutter pulse.

KEY CODEBASE LOCATIONS

- **libraries/AP_Mission/AP_Mission_Commands.cpp:115**: Command parsing.
- **libraries/AP_Camera/AP_Camera.cpp**: Core execution logic.

DO_ENGINE_CONTROL (223)

SUMMARY

The `DO_ENGINE_CONTROL` command provides a mission-level interface for Internal Combustion Engines (ICE). It is used to start or stop the engine, manage "Cold Start" procedures, and control height-based delays for engine engagement. This is critical for high-endurance large-scale drones and hybrid power systems.

STATUS

Supported (ArduPlane and ArduCopter with ICE enabled)

DIRECTIONALITY

- **RX (Receive):** The vehicle receives this command as part of a mission upload or as a direct command.

MISSION STORAGE (AP_MISSION)

- **Start/Stop (Param 1):** 1: Start Engine, 0: Stop Engine.
- **Cold Start (Param 2):** 1: Cold Start (enables extended cranking/glow plug time), 0: Warm Start.
- **Height Delay (Param 3):** Height in meters to wait before attempting a start (unit converted to cm in storage).
- **Options (Param 4):** Bitmask for start behavior (e.g., Allow start while disarmed).
- **Mechanism:** Stored in the internal `do_engine_control` content struct.

EXECUTION (ENGINEER'S VIEW)

ICE STATE MACHINE

The command is handled by the `AP_ICEngine` library.

1. **Cranking Logic:** When `Start (1)` is commanded, the autopilot manages the Starter Motor (via a Relay or ESC) and Ignition system.
2. **Telemetry Check:** ArduPilot monitors the engine RPM via an onboard sensor. The "Start" is only considered successful when the RPM exceeds the `ICE_START_RPM`.
3. **Kill Switch:** The `Stop (0)` command instantly terminates the ignition/fuel relay, ensuring a rapid shutdown for safety or post-mission power-down.
4. **VTOL Integration:** In hybrid QuadPlanes, this command can be used to start the generator engine once the aircraft is clear of the ground and transitioning to fixed-wing flight.

DATA FIELDS (MAVLINK)

- `param1` **(Start):** 1: Start, 0: Stop.
- `param2` **(Cold):** 1: Cold Start, 0: Warm.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- `param3` **(Height):** Delay start until this height (meters).
- `param4` **(Options):** Bit 0: Allow start while disarmed.
- `param5` to `param7`: Unused.

THEORY: COMBUSTION SAFETY

Starting a combustion engine autonomously is a significant safety risk.

- **Vibration:** Engines produce high-frequency vibration that can saturate the EKF. ArduPilot's engine controller includes logic to check the "Health" of the IMUs during the cranking phase.
- **Height Delays:** The `Height Delay (Param 3)` is a critical safety feature. It prevents a hot, powerful engine from starting on the ground where it could injure people. The engine is only engaged once the vehicle is safely "In the Air."

PRACTICAL USE CASES

1. **Hybrid Range Extension:**
   - *Scenario:* A VTOL drone launches using battery power for noise abatement.
   - *Action:* Mission includes `DO_ENGINE_CONTROL (Start, Height Delay: 20m)`. The engine starts only after the drone is high enough to minimize ground noise.
2. **Autonomous Landing Shutdown:**
   - *Scenario:* A gas-powered drone completes its mission.
   - *Action:* `NAV_LAND` → `[CONDITION_DELAY](/mission-planning/condition-commands.html#CONDITION_DELAY) (5s)` → `DO_ENGINE_CONTROL (Stop)`. Ensures the engine is cool and stopped before humans approach the airframe.

KEY PARAMETERS

- `ICE_START_RPM`: RPM threshold to consider the engine "Running."
- `ICE_PIN`: Relay pin used for the Starter.
- `ICE_PWM_IGN`: PWM value to enable ignition.

KEY CODEBASE LOCATIONS

- **libraries/AP_ICEngine/AP_ICEngine.cpp**: The core ICE driver and state machine.
- **ArduPlane/commands_logic.cpp:183**: Plane mission integration.

DO_FENCE_ENABLE (207)

SUMMARY

The `DO_FENCE_ENABLE` command allows the mission script to dynamically enable or disable the vehicle's Geofence system. This is particularly useful for missions where a drone must transition from a restricted "test area" into an open flight corridor, or vice-versa.

STATUS

Supported (All Vehicles)

DIRECTIONALITY

- **RX (Receive):** The vehicle receives this command as part of a mission upload or as a direct command from the GCS.

MISSION STORAGE (AP_MISSION)

- **Enable (Param 1):**
  - 0: Disable all fences.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





    - 1: Enable all fences.
    - 2: Disable "Floor" only (Altitude minimum).
- **Packing:** Stored in the internal `p1` field.

EXECUTION (ENGINEER'S VIEW)

DYNAMIC FENCING LOGIC

Execution is handled by the `AC_Fence` library (AC_Fence.cpp).

1. **State Change:** The command updates the internal fence enable bitmask.
2. **Safety Verification:** If enabling the fence, the autopilot immediately checks the current vehicle position against the fence boundaries.
3. **Breech Handling:** If the vehicle is *already* outside the boundary when the fence is enabled via mission command, the autopilot will immediately trigger a **Fence Failsafe** (typically RTL or Land).

DATA FIELDS (MAVLINK)

- `param1` **(State):** 0: Disable, 1: Enable, 2: Disable Floor.
- `param2` to `param7`: Unused.

THEORY: ADAPTIVE AIRSPACE

Dynamic fencing enables **Adaptive Airspace** management.

- **Corridors:** A mission can be "fenced" into a narrow pipe for a BVLOS transit. Once it reaches a designated high-altitude workspace, the `DO_FENCE_ENABLE (Disable)` command can be used to allow for freer movement during manual inspection tasks.
- **Payload Protection:** A drone carrying hazardous material might enable a very tight fence only during the transport phase, then disable it for recovery once the payload is released.

PRACTICAL USE CASES

1. **Transitioning to Open Sea:**
    - *Scenario:* A search-and-rescue drone launches from a crowded beach.
    - *Action:* Mission starts with Geofence Enabled. Once the drone is 500m offshore, `DO_FENCE_ENABLE (Disable)` is triggered to allow it to scan a wide area without nuisance alerts.
2. **Autonomous Testing:**
    - *Scenario:* Testing a new Lua script.
    - *Action:* The script is only allowed to run while the drone is inside a "Safety Box." The mission enables the box before starting the script and disables it upon completion.

KEY PARAMETERS

- `FENCE_ENABLE`: Global master switch for fencing.
- `FENCE_ACTION`: Determines what happens when a fence is breached (RTL, Land, etc.).

KEY CODEBASE LOCATIONS

- **libraries/AC_Fence/AC_Fence.cpp**: Core fence logic.
- **ArduCopter/mode_auto.cpp:1008**: Mission command intake.

DO_GIMBAL_MANAGER_PITCHYAW (1000)

SUMMARY

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





The `DO_GIMBAL_MANAGER_PITCHYAW` command provides advanced, high-level control for modern MAVLink gimbals. It allows for simultaneous control of both angles and rates, and includes flags for specifying the frame of reference (Body vs. Earth). This is the "Modern" version of the legacy `DO_MOUNT_CONTROL`.

STATUS

Supported (All Vehicles with Gimbal Manager backends)

DIRECTIONALITY

- **RX (Receive):** The vehicle receives this command as part of a mission upload or as a direct command.

MISSION STORAGE (AP_MISSION)

- **Pitch (Param 1):** Angle in degrees.
- **Yaw (Param 2):** Angle in degrees.
- **Pitch Rate (Param 3):** Deg/s.
- **Yaw Rate (Param 4):** Deg/s.
- **Flags (Param 5):** Bitmask for coordinate frames and modes.
- **Gimbal ID (Param 7):** Target instance.
- **Packing:** Stored in the `gimbal_manager_pitchyaw` content struct.

EXECUTION (ENGINEER'S VIEW)

GIMBAL MANAGER PROTOCOL

Unlike simple servo gimbals, modern gimbals (like the DJI H20T or specialized Gremsy units) handle their own stabilization. ArduPilot acts as a "Manager."

1. **Setpoints:** ArduPilot extracts the 5D setpoint (Angle P, Angle Y, Rate P, Rate Y, Frame).
2. **Targeting:** The command is forwarded to the specific Gimbal ID via the MAVLink Gimbal Protocol v2.
3. **Frame Switching:**
    - **Body Frame:** The camera follows the drone's nose.
    - **Earth Frame:** The camera stays locked to a compass heading.
4. **Rate Overlays:** The inclusion of "Rate" parameters allows for smooth "Cinematic Pans" where the gimbal moves at a constant speed to a target angle, rather than jumping instantly.

DATA FIELDS (MAVLINK)

- `param1` **(Pitch):** deg.
- `param2` **(Yaw):** deg.
- `param3` **(PRate):** deg/s.
- `param4` **(YRate):** deg/s.
- `param5` **(Flags):** See GIMBAL_MANAGER_FLAGS.

THEORY: RATE VS. POSITION CONTROL

In control systems, **Position Control** (P) is prone to "jerk" ($J = \frac{da}{dt}$).

- **The Improvement:** By specifying a **Rate Limit** (ω), the gimbal manager implements a **Velocity-Limited Position Controller**.
- **Mathematics:** The target angle $\theta(t)$ follows a ramp:

$$\theta(t) = \theta_0 + \omega \cdot t$$

until $\theta(t) = \theta_{target}$. This results in the smooth, professional camera movements seen in high-end cinema drones.

PRACTICAL USE CASES

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





1. **Survey Pivot:**
    - *Scenario:* A triple-lens camera needs to pan from visual to thermal while maintaining a steady rate to avoid motion blur.
    - *Action:* `DO_GIMBAL_MANAGER_PITCHYAW (Yaw: 90, YawRate: 5)`. The camera slowly turns to the right over 18 seconds.
2. **Tracking a Moving Target:**
    - *Scenario:* The GCS is streaming a target velocity.
    - *Action:* The command uses the Rate parameters to "Lead" the target.

KEY PARAMETERS

- `MNT1_TYPE`: Must be set to `MAVLinkV2`.

KEY CODEBASE LOCATIONS

- **libraries/GCS_MAVLink/GCS_Common.cpp:5519**: Command forwarding logic.
- **libraries/AP_Mount/AP_Mount.cpp**: High-level mount coordination.

DO_GO_AROUND (191)

SUMMARY

The `DO_GO_AROUND` command is an safety override used during an autonomous landing. It instructs the vehicle to immediately abort the landing sequence, climb to a safe altitude, and typically enter a loiter or wait for further instructions.

STATUS

Supported (ArduPlane and QuadPlane Only)

DIRECTIONALITY

- **RX (Receive):** The vehicle receives this command as a direct override from the GCS or as an automatic response to a landing sensor failure.

MISSION STORAGE (AP_MISSION)

- **Packet Param 1 (Altitude):** Altitude to climb to (meters).
- **Mechanism:** Stored as an immediate execution marker in the landing state machine.

EXECUTION (ENGINEER'S VIEW)

ARDUPLANE IMPLEMENTATION

When a Go-Around is triggered:

1. **State Reset:** The landing state machine (Flare/Touchdown detection) is immediately terminated.
2. **Climb-out:** The plane applies full throttle and targets the "Abort Altitude" specified in the command (or the last takeoff altitude).
3. **Navigation:** The plane typically returns to the `DO_LAND_START` waypoint or the waypoint immediately preceding the landing sequence to re-attempt the approach.

QUADPLANE IMPLEMENTATION

For hybrid VTOLs, the Go-Around behavior depends on the current stage:

- **Fixed-Wing Approach:** Performs a standard fixed-wing abort climb.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- **VTOL Descent:** The vehicle transitions to a multicopter climb, rising vertically to a safe height before either loitering or transitioning back to fixed-wing flight.

DATA FIELDS (MAVLINK)

- `param1` **(Altitude):** Target altitude for the abort climb.
- `param2` to `param7`: Unused.

THEORY: ENERGY VS. SAFETY

A Go-Around is a trade-off between **Impact Risk** and **Stall Risk**.

- **The Hazard:** During landing, a plane is at its lowest energy state (low speed, low altitude).
- **The Reaction:** ArduPilot prioritized **Airspeed first**. It will prioritize gaining speed over gaining altitude to ensure the aircraft remains controllable during the high-stress transition away from the ground.

PRACTICAL USE CASES

1. **Runway Incursion:**
    - *Scenario:* A vehicle or person enters the runway while a drone is on final approach.
    - *Action:* The GCS operator clicks "Abort Landing," sending a `MAV_CMD_DO_GO_AROUND`. The drone climbs away and circles until the runway is clear.
2. **Lidar/Rangefinder Glitch:**
    - *Scenario:* The landing Lidar fails 5 meters above the ground.
    - *Action:* ArduPilot's internal safety logic triggers an automatic Go-Around to prevent a blind landing.

KEY PARAMETERS

- `TECS_LAND_SINK`: Controls the sink rate that might trigger an automatic abort if exceeded.
- `LAND_ABORT_THR`: (Plane) Throttle level used during the abort climb.

KEY CODEBASE LOCATIONS

- **ArduPlane/commands_logic.cpp**: Landing logic handler.
- **libraries/AP_Landing/AP_Landing.cpp**: Shared landing abort state machine.

DO_GRIPPER (211)

SUMMARY

The `DO_GRIPPER` command manages the state of a mechanical cargo gripper (claw, magnetic latch, or drop-hook). This is a dedicated actuator command that is safer and more descriptive than a raw `DO_SET_SERVO` because it integrates with the autopilot's landing and failsafe logic.

STATUS

Supported (All Vehicles)

DIRECTIONALITY

- **RX (Receive):** The vehicle receives this command as part of a mission upload or as a direct command.

MISSION STORAGE (AP_MISSION)

- **Instance (Param 1):** The gripper ID (ArduPilot typically supports 1 primary gripper).

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- **Action (Param 2):**
  - 0: Release (Open).
  - 1: Grab (Close).
- **Packing:** Stored in the internal `gripper` content struct.

EXECUTION (ENGINEER'S VIEW)

GRIPPER LOGIC

Execution is handled by the `AP_Gripper` library (AP_Gripper.cpp).

1. **Safety Interlocks:** ArduPilot can be configured to prevent a "Release" action if the vehicle is not within a safe altitude range or if a "Touchdown" has not been detected.
2. **Actuator Type:** `AP_Gripper` supports multiple hardware backends:
   - **Servo:** Moves a servo to a specific PWM (Open/Closed).
   - **EPM (Electro-Permanent Magnet):** Sends a pulse to flip the magnetic polarity (no power required to hold).
3. **Completion:** The mission script advances immediately after the command is sent to the actuator; it does not wait for a "sensor confirmed closed" signal unless combined with a `CONDITION_DELAY`.

DATA FIELDS (MAVLINK)

- `param1` **(ID):** Gripper instance number.
- `param2` **(Action):** 0: Release, 1: Grab.
- `param3` to `param7`: Unused.

THEORY: ENERGY STATES OF ACTUATORS

- **Standard Servos:** Require continuous power to maintain a "Grab" state against a heavy load. This causes heat and battery drain.
- **EPMs:** Use a high-current pulse to re-align the magnetic domains of an Alnico core. Once "Grabbed," the payload is held by a permanent magnetic field with zero power consumption. This is the preferred gripper type for long-endurance ArduPilot missions.

PRACTICAL USE CASES

1. **Automated Package Delivery:**
   - *Scenario:* A drone delivering a box to a backyard.
   - *Action:* `NAV_WAYPOINT` → `NAV_PAYLOAD_PLACE` → `DO_GRIPPER (Release)`.
2. **Sample Collection:**
   - *Scenario:* A rover picking up a rock.
   - *Action:* `DO_GRIPPER (Grab)` once the rover is over the target.

KEY PARAMETERS

- `GRIP_ENABLE`: Global switch.
- `GRIP_TYPE`: Selection between Servo and EPM.
- `GRIP_CAN_ON_PWM`: The PWM value used to Close the gripper.

KEY CODEBASE LOCATIONS

- **libraries/AP_Gripper/AP_Gripper.cpp**: Core hardware interface.
- **ArduCopter/mode_auto.cpp**: Mission integration.

DO_GUIDED_LIMITS (222)

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





SUMMARY

The `DO_GUIDED_LIMITS` command sets safety constraints for Guided Mode. It defines the maximum time and distance an external controller (e.g., an onboard companion computer or an GCS script) is allowed to control the vehicle before the autopilot automatically intervenes and terminates the command.

STATUS

Supported (All Vehicles)

DIRECTIONALITY

- **RX (Receive):** The vehicle receives this command as part of a mission upload or as a direct setup command.

MISSION STORAGE (AP_MISSION)

- **Time Limit (Param 1):** Max time in seconds for external control.
- **Alt Min (Param 2):** Minimum altitude (meters).
- **Alt Max (Param 3):** Maximum altitude (meters).
- **Horiz Max (Param 4):** Maximum horizontal distance from the command start point (meters).
- **Packing:** Stored in the `guided_limits` content struct.

EXECUTION (ENGINEER'S VIEW)

SAFETY BUFFER LOGIC

ArduPilot handles this in the `Guided` flight mode logic (mode_auto.cpp).

1. **Handover:** When the mission reaches `NAV_GUIDED_ENABLE` (or enters Guided mode via MAVLink), these limits are activated.
2. **Continuous Monitoring:** Every loop, the autopilot checks:
   - **Time:** `millis() - start_time > Param1`.
   - **Altitude:** `current_alt < Param2` OR `current_alt > Param3`.
   - **Radius:** `2D_Distance(start_pos, current_pos) > Param4`.
3. **Failsafe:** If any limit is breached, the autopilot instantly terminates Guided mode and typically returns to the mission or enters a failsafe state (RTL).

DATA FIELDS (MAVLINK)

- `param1` **(Time):** Max execution time (s).
- `param2` **(Alt Min):** Min altitude (m).
- `param3` **(Alt Max):** Max altitude (m).
- `param4` **(Radius):** Max horizontal distance (m).

THEORY: THE SANDBOX PRINCIPAL

`DO_GUIDED_LIMITS` implements a **Sandbox** for external intelligence.

- **The Problem:** Companion computer scripts (running ROS or Python) can crash or "go rogue" due to bugs.
- **The Solution:** The flight controller (which is the ultimate arbiter of safety) creates a "Virtual Box" and a "Watchdog Timer." The companion computer is free to experiment inside the box, but the flight controller will "reel it back in" the moment it tries to exit the safe boundaries.

PRACTICAL USE CASES

1. **AI Landing Research:**
   - *Scenario:* Testing a new vision-based landing algorithm on a Raspberry Pi.
   - *Action:* `DO_GUIDED_LIMITS (Time: 30s, Alt Min: 2m)`. If the AI fails to land within 30 seconds or drops below 2 meters unexpectedly, the flight controller takes over.

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





2. **GCS "Follow-Me":**
    - *Scenario:* A tablet app is commanding the drone to follow a cyclist.
    - *Action:* Use horizontal limits to ensure the drone never wanders too far from the cyclist if the app loses the tracking lock.

KEY PARAMETERS

- `GUID_OPTIONS`: Can affect how Guided mode behaves when limits are hit.

KEY CODEBASE LOCATIONS

- **ArduCopter/mode_auto.cpp:785**: Mission command intake.
- **ArduCopter/mode_guided.cpp**: Limit enforcement logic.

DO_INVERTED_FLIGHT (210)

SUMMARY

The `DO_INVERTED_FLIGHT` command instructions the autopilot to fly the aircraft upside-down. This is primarily used in ArduPlane for aerobatics or specialized flight maneuvers.

STATUS

Supported (ArduPlane Only)

DIRECTIONALITY

- **RX (Receive):** The vehicle receives this command as part of a mission upload or as a direct command from the GCS.

MISSION STORAGE (AP_MISSION)

- **Inverted (Param 1):**
    - 0: Normal flight (Upright).
    - 1: Inverted flight (Upside-down).
- **Packing:** Stored in the internal `p1` field.

EXECUTION (ENGINEER'S VIEW)

ARDUPLANE IMPLEMENTATION

In Plane, the command updates the `auto_state.inverted_flight` flag (commands_logic.cpp).

1. **Attitude Target:** When inverted flight is enabled, the AHRS (Attitude and Heading Reference System) redefines the Roll target as 180 degrees.
2. **Controller Inversion:** The PID controllers for Aileron and Elevator must account for the inverted state.
    - **Pitch:** Pulling "Up" on the stick while inverted will cause the aircraft to move towards the ground.
    - **Self-Leveling:** The autopilot automatically handles the inversion of these vectors to ensure that standard "Fly-By-Wire" (FBW) controls remain intuitive for the pilot (if in a semi-autonomous mode) or consistent for the mission engine.
3. **Oil/Fuel Handling:** This command is only recommended for airframes with inverted-flight capable propulsion systems (e.g., fuel-injected engines with header tanks or electric systems).

DATA FIELDS (MAVLINK)

- `param1` **(State):** 0: Normal, 1: Inverted.
- `param2` to `param7`: Unused.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





THEORY: THE LIFT VECTOR INVERSION

In normal flight, lift (L) acts upwards to oppose gravity (W).

$$L = W$$

In inverted flight, the aircraft must maintain a negative angle of attack (α) to generate lift "upwards" relative to the Earth, even though the wing is upside down.

$$L_{inverted} = \frac{1}{2}\rho v^2 S C_{L,inverted}$$

ArduPilot's navigation controller (L1) calculates the required bank angle to maintain the ground track while accounting for the reduced lift efficiency of most non-symmetrical airfoils when flying inverted.

PRACTICAL USE CASES

1. **Aerobatic Display:**
    - *Scenario:* An autonomous drone show involving complex maneuvers.
    - *Action:* `WAYPOINT (A)` → `DO_INVERTED_FLIGHT (1)` → `WAYPOINT (B)`. The plane rolls 180 degrees and flies the segment between A and B upside-down.
2. **Sensor Calibration:**
    - *Scenario:* Calibrating an IMU or magnetometer by exposing it to symmetrical forces.
    - *Action:* Flying a specific leg upright, then repeating it inverted to cancel out bias.

KEY PARAMETERS

- `ROLL_LIMIT_DEG`: Still applies while inverted.

KEY CODEBASE LOCATIONS

- **ArduPlane/commands_logic.cpp:134**: Mission command intake.
- **ArduPlane/attitude.cpp**: High-level attitude control logic for inversion.

DO_JUMP (177)

SUMMARY

The `DO_JUMP` command provides procedural flow control within a mission. It allows the autopilot to jump back (or forward) to a specific mission item number and repeat that sequence a set number of times. This is the primary mechanism for creating loops (e.g., repeating a survey grid or a loiter-and-check sequence).

STATUS

Supported (All Vehicles)

DIRECTIONALITY

- **RX (Receive):** The vehicle receives this command as part of a mission upload.

MISSION STORAGE (AP_MISSION)

- **Target (Param 1):** The sequence number of the mission item to jump to.
- **Repeat Count (Param 2):** The number of times to perform the jump.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- **Mechanism:** ArduPilot maintains an internal `jump_tracking` array (AP_Mission.cpp) to persist the state of each jump command across reboots or mode changes.

EXECUTION (ENGINEER'S VIEW)

LOGIC FLOW

When the mission engine encounters a `DO_JUMP`:

1. **Check Counter:** It retrieves the current `num_times_run` for this specific command index from the `jump_tracking` RAM.
2. **Comparison:**
   - If `num_times_run` < `Repeat Count`, it increments the counter and sets the current mission index to the **Target**.
   - If `num_times_run` == `Repeat Count`, it ignores the jump and proceeds to the *next* mission item (breaking the loop).
3. **Unlimited Loops:** If the `Repeat Count` is set to a high value (like 255 or 65535, depending on the GCS), ArduPilot can be configured to loop indefinitely.

JUMP TRACKING CONSTRAINTS

- **Max Jumps:** ArduPilot typically supports tracking up to 15-20 individual `DO_JUMP` commands in a single mission (defined by `AP_MISSION_MAX_NUM_DO_JUMP_COMMANDS`).
- **Nested Loops:** While technically possible, nesting jumps can lead to complex state behavior. ArduPilot's tracker is indexed by the *index of the jump command itself*, which prevents collisions between different loops.

DATA FIELDS (MAVLINK)

- `param1` **(Item #):** Mission sequence number to jump to.
- `param2` **(Repeat):** Total number of times to perform the jump.
- `param3` to `param7`: Unused.

THEORY: FINITE STATE MACHINES AND HALTING

A mission is essentially a **Linear Finite State Machine**. `DO_JUMP` introduces **Cycles** into the graph.

- **Determinism:** Because ArduPilot tracks the repeat count in non-volatile-ready RAM, the mission remains deterministic. If the vehicle loses power and reboots, the mission can resume and "remember" how many loops it has already completed.
- **The Halting Problem:** Infinite loops are dangerous in autonomous flight. Always ensure a `DO_JUMP` has a finite repeat count unless the vehicle is in a monitored "holding pattern" state.

PRACTICAL USE CASES

1. **Survey Re-runs:**
   - *Scenario:* A drone is scanning for a lost hiker. The search grid needs to be flown 3 times to ensure coverage.
   - *Action:* `WP 1` ... `WP 10` → `DO_JUMP (Target: 1, Repeat: 3)`.
2. **Delayed Entry:**
   - *Scenario:* A plane must loiter at a waypoint until a specific time, but ArduPilot's clock isn't synchronized yet.
   - *Action:* Use a `DO_JUMP` to a loiter point with a small repeat count to "wait" for GCS synchronization.

KEY PARAMETERS

- `MIS_RESTART`: Controls whether the jump counters are reset when the mission is restarted.

KEY CODEBASE LOCATIONS

- **libraries/AP_Mission/AP_Mission.cpp:2311**: `increment_jump_times_run` implementation.
- **libraries/AP_Mission/AP_Mission.h**: Definition of the jump tracking structure.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





JUMP_TAG (600) / DO_JUMP_TAG (601)

SUMMARY

The `JUMP_TAG` and `DO_JUMP_TAG` commands provide an advanced, identifier-based flow control system for missions. Unlike the standard `DO_JUMP` (which uses sequence numbers), the tag-based system uses a "Named Marker" (a numeric tag). This allows the mission to be edited—inserting or deleting waypoints—without breaking the jump logic, as the autopilot searches for the tag rather than a fixed index.

STATUS

Supported (All Vehicles)

DIRECTIONALITY

- **RX (Receive):** The vehicle receives these commands as part of a mission upload.

MISSION STORAGE (AP_MISSION)

- **Tag (Param 1):** A user-defined integer ID (1-255).
- **Repeat (Param 2):** (601 only) Number of times to jump to that tag.
- **Search Logic:** When `DO_JUMP_TAG` is reached, the autopilot scans the entire mission command list from the beginning to find the first `JUMP_TAG` with a matching ID (AP_Mission.cpp:2279).

EXECUTION (ENGINEER'S VIEW)

ROBUST FLOW CONTROL

1. **Tag Discovery:** The mission engine calls `find_tag_index(tag)`. This is a $O(N)$ operation where N is the number of mission items.
2. **Persistence:** Like standard jumps, ArduPilot tracks the `num_times_run` for each `DO_JUMP_TAG` using the command's unique mission index.
3. **Failure State:** If a `DO_JUMP_TAG` points to a tag that does not exist in the mission list, ArduPilot logs a warning and the mission **completes immediately** for safety.

DATA FIELDS (MAVLINK)

JUMP_TAG (600)

- `param1` **(Tag ID):** The identifier for this marker.

DO_JUMP_TAG (601)

- `param1` **(Tag ID):** The target marker to jump to.
- `param2` **(Repeat):** Number of times to jump.

THEORY: SEMANTIC VS. POSITIONAL ADDRESSING

In computer science, `DO_JUMP` is equivalent to a **GOTO** statement with a line number. `DO_JUMP_TAG` is equivalent to a **GOTO** with a **Label**.

- **Maintainability:** Positional jumps are "Brittle." If you add a waypoint at the start of a mission, every `DO_JUMP` that follows must be manually updated to point to the new shifted indices.
- **Flexibility:** `DO_JUMP_TAG` is "Semantic." The jump points to the *meaning* of the location (e.g., "Start of Search Area"), ensuring the mission logic remains valid regardless of minor path adjustments.

PRACTICAL USE CASES

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





1. **Multi-Stage Search:**
    - *Scenario:* A drone needs to orbit a site, then fly a grid, then repeat.
    - *Action:* Place `JUMP_TAG (ID: 50)` at the start of the orbit. Use `DO_JUMP_TAG (Target: 50, Repeat: 5)` at the end of the grid.
2. **Dynamic Scripting:**
    - *Scenario:* A GCS script wants to loop a specific section of a mission that was just uploaded.
    - *Action:* The script searches for the Tag ID instead of parsing the whole mission list for indices.

KEY PARAMETERS

- `MIS_RESTART`: Resetting the mission will reset the tag counters.

KEY CODEBASE LOCATIONS

- **libraries/AP_Mission/AP_Mission.cpp:2279**: Tag discovery loop.
- **libraries/AP_Mission/AP_Mission.cpp:2176**: Mission re-indexing and tag validation.

DO_LAND_START (189)

SUMMARY

The `DO_LAND_START` command acts as a marker or "entry point" for the landing sequence in a mission. It is not an active flight command itself, but rather a metadata tag that tells the autopilot where the landing-specific mission items begin.

STATUS

Supported (All Vehicles)

DIRECTIONALITY

- **RX (Receive):** The vehicle receives this command as part of a mission upload or as a direct command (to trigger an immediate mission landing).

MISSION STORAGE (AP_MISSION)

- **Latitude/Longitude/Altitude:** Optional. If provided, they help the autopilot find the *closest* landing sequence if multiple exist in a single mission.
- **Packing:** Stored as a standard location-based mission item.

EXECUTION (ENGINEER'S VIEW)

AUTOPILOT LOGIC

ArduPilot uses `DO_LAND_START` in several critical scenarios:

1. **Mission Landing Trigger:** When the vehicle enters RTL or a Battery Failsafe, and the mission contains a `DO_LAND_START` marker, the autopilot will skip all intermediate waypoints and jump directly to the item *following* the marker to begin a structured approach and landing.
2. **Closest Sequence Discovery:** If a mission contains multiple landing sequences (e.g., for different wind directions), ArduPilot's `get_landing_sequence_start()` function (AP_Mission.cpp) will calculate which `DO_LAND_START` is geographically closest to the vehicle's current position and use that sequence.
3. **Arming Check:** ArduPilot can be configured to require a `DO_LAND_START` in the mission as a pre-arm safety check for certain high-value autonomous operations.

DATA FIELDS (MAVLINK)

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- `param1` to `param4`: Unused.
- `x` **(Latitude):** Optional coordinate for proximity matching.
- `y` **(Longitude):** Optional coordinate for proximity matching.
- `z` **(Altitude):** Optional coordinate for proximity matching.

THEORY: THE NON-DESTRUCTIVE JUMP

Standard missions are sequential. Failsafes usually involve returning home (`RTL`). `DO_LAND_START` enables a **Mission-Aware Failsafe**.

- **The Problem:** RTL is often a direct line that might cross restricted airspace or terrain.
- **The Solution:** By marking a landing sequence, the pilot defines a known-safe approach path (e.g., spiral down → align with runway → land). `DO_LAND_START` ensures the autopilot uses this "known-good" procedure instead of a "naive" RTL.

PRACTICAL USE CASES

1. **Runway Alignment (Plane):**
   - *Scenario:* A plane needs to land on a narrow runway.
   - *Action:* `DO_LAND_START` → `WAYPOINT (Approach)` → `WAYPOINT (Final)` → `NAV_LAND`. If a failsafe occurs, the plane jumps to "Approach" instead of flying directly to the runway.
2. **Emergency Rally Landing (Copter):**
   - *Scenario:* Mapping a large forest.
   - *Action:* Include multiple `DO_LAND_START` markers at various clearings. The drone will choose the closest one during a low-battery event.

KEY PARAMETERS

- `MIS_OPTIONS`: Controls how the mission engine handles the landing sequence.

KEY CODEBASE LOCATIONS

- **libraries/AP_Mission/AP_Mission.cpp:2395**: `get_landing_sequence_start` implementation.
- **ArduPlane/commands_logic.cpp**: Plane-specific landing sequence handling.

DO_MOUNT_CONTROL (205)

SUMMARY

The `DO_MOUNT_CONTROL` command allows the mission script to set specific Pitch, Roll, and Yaw angles for a camera mount or antenna gimbal. Unlike ROI (which targets a coordinate), `MOUNT_CONTROL` targets specific body-relative or Earth-relative angles.

STATUS

Supported (All Vehicles)

DIRECTIONALITY

- **RX (Receive):** The vehicle receives this command as part of a mission upload or as a direct command.

MISSION STORAGE (AP_MISSION)

- **Pitch (Param 1):** Angle in degrees.
- **Roll (Param 2):** Angle in degrees.
- **Yaw (Param 3):** Angle in degrees.
- **Mechanism:** Stored in the `mount_control` content struct.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





EXECUTION (ENGINEER'S VIEW)

CONTROLLING THE MOUNT

ArduPilot directs these commands to the `AP_Mount` library (AP_Mount.cpp).

1. **Angle Mapping:** The command defines the target orientation.
2. **Mount Mode:** Encountering this command typically switches the mount to `MAVLINK_TARGETING` mode.
3. **Airframe Yaw Integration:** A critical feature in ArduCopter: If the vehicle has a camera mount that does *not* support a pan (Yaw) axis (e.g., a 2-axis gimbal), ArduPilot will rotate the **entire vehicle airframe** to achieve the requested Yaw angle (mode_auto.cpp).

DATA FIELDS (MAVLINK)

- `param1` **(Pitch):** Target pitch angle (deg).
- `param2` **(Roll):** Target roll angle (deg).
- `param3` **(Yaw):** Target yaw angle (deg).
- `param7` **(Mode):** Selection between angle and rate control (if supported).

THEORY: BODY-RELATIVE VS. EARTH-RELATIVE

- **Body-Relative:** The angles are relative to the drone's nose. If the drone turns 90 degrees, the camera turns 90 degrees.
- **Earth-Relative (North-Up):** The camera locks onto a compass heading. If the drone turns, the gimbal compensates to stay pointed North.
- **ArduPilot Default:** `MAV_CMD_DO_MOUNT_CONTROL` is typically interpreted as Earth-Relative for Pitch (tilt) and Yaw (pan), but Body-Relative for Roll.

PRACTICAL USE CASES

1. **Vertical Mapping (Nadir):**
   - *Scenario:* A surveyor needs the camera to point straight down for the entire flight.
   - *Action:* `DO_MOUNT_CONTROL (Pitch: -90, Roll: 0, Yaw: 0)`.
2. **Forward Scouting:**
   - *Scenario:* A search and rescue drone needs to look 15 degrees below the horizon and 45 degrees to the right to scan a coastline.
   - *Action:* `DO_MOUNT_CONTROL (Pitch: -15, Yaw: 45)`.

KEY PARAMETERS

- `MNT1_TYPE`: Enables the gimbal driver.
- `MNT1_PITCH_MIN/MAX`: Defines the mechanical limits to prevent servo damage.

KEY CODEBASE LOCATIONS

- **libraries/AP_Mount/AP_Mount.cpp:555**: Command intake.
- **ArduCopter/mode_auto.cpp:1987**: Yaw-override logic for 2-axis gimbals.

DO_PARACHUTE (208)

SUMMARY

The `MAV_CMD_DO_PARACHUTE` command triggers or manages the vehicle's emergency recovery parachute. In a mission context, this is rarely used for standard recovery (which is usually an RTL) but is vital for "Total Safety" missions or testing environments where a soft impact must be guaranteed at a specific stage.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





STATUS

Supported (All Vehicles)

DIRECTIONALITY

- **RX (Receive):** The vehicle receives this command as part of a mission upload or as a direct "PANIC" command from the GCS.

MISSION STORAGE (AP_MISSION)

- **Action (Param 1):**
  - 0: Disable parachute release.
  - 1: Enable parachute (Armed).
  - 2: Release parachute (Fire immediately).
- **Packing:** Stored in the internal `p1` field.

EXECUTION (ENGINEER'S VIEW)

EMERGENCY LOGIC

The command utilizes the `AP_Parachute` library (AP_Parachute.cpp).

1. **Safety Interlocks:** If `Release (2)` is commanded, the autopilot **instantly** stops all flight motors (ArduCopter/GCS_Mavlink.cpp:978). This prevents the parachute lines from tangling in the props.
2. **Deployment:** A high signal is sent to the assigned parachute relay or servo.
3. **Logging:** A "PARACHUTE" event is recorded in the DataFlash log.

DATA FIELDS (MAVLINK)

- `param1` **(Action):** 0: Disable, 1: Enable, 2: Release.
- `param2` to `param7`: Unused.

THEORY: THE POINT OF NO RETURN

Parachute deployment is a **Terminating Event**.

- **Motor Inhibition:** Once a parachute is fired, ArduPilot will not allow the motors to restart until the vehicle has been disarmed and rebooted.
- **Altitude Constraint:** Parachutes require a minimum altitude to inflate. ArduPilot uses the `CHUTE_ALT_MIN` parameter to prevent deployment if the vehicle is too low (which could lead to a prop-entanglement before inflation).

PRACTICAL USE CASES

1. **Sensitive Equipment Recovery:**
   - *Scenario:* A drone is carrying a $100k prototype sensor.
   - *Action:* At the end of the mission, instead of a standard landing, the mission triggers `DO_PARACHUTE (Release)` over a designated soft-target area to ensure the sensor is never subjected to a landing impact.
2. **Safety Drills:**
   - *Scenario:* Testing a new airframe.
   - *Action:* The GCS "Kill Switch" is mapped to `DO_PARACHUTE (Release)`.

KEY PARAMETERS

- `CHUTE_ENABLED`: Global master switch.
- `CHUTE_TYPE`: Defines if the trigger is a Relay or Servo.
- `CHUTE_SERVO_ON`: The PWM value used to fire the chute.

KEY CODEBASE LOCATIONS

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- **libraries/AP_Parachute/AP_Parachute.cpp**: Core deployment logic.
- **ArduCopter/GCS_Mavlink.cpp:978**: Critical motor-kill logic.

DO_PAUSE_CONTINUE (193)

SUMMARY

The `DO_PAUSE_CONTINUE` command provides a mechanism to suspend or resume the mission's horizontal movement without changing the flight mode. It is primarily used to "Pause" the vehicle at its current location during a mission.

STATUS

Supported (ArduCopter and Rover)

DIRECTIONALITY

- **RX (Receive):** The vehicle receives this command as a direct override from the GCS.

MISSION STORAGE (AP_MISSION)

- **Packing:** This command is typically sent as an immediate command (`COMMAND_LONG`), not stored in a mission. If encountered in a mission, ArduPilot treats it as a state change.

EXECUTION (ENGINEER'S VIEW)

ARDUCOPTER IMPLEMENTATION

In Copter, the command calls `ModeAuto::pause()` or `ModeAuto::resume()` (mode_auto.cpp).

1. **WPNav Interaction:** The autopilot tells the `AC_WPNav` library to freeze the current trajectory.
2. **Position Hold:** The drone enters a "GPS Position Hold" at its current coordinates.
3. **Vertical Behavior:** The drone maintains its current mission altitude.
4. **Resumption:** When `Continue (1)` is sent, the `WPNav` resumes following the path from where it was paused.

DATA FIELDS (MAVLINK)

- `param1` **(State):** 0: Pause, 1: Continue.
- `param2` to `param7` : Unused.

THEORY: SUSPENDING THE VECTOR

Most flight modes are "State-based." `AUTO` mode is "Vector-based."

- **The Pause:** `DO_PAUSE_CONTINUE` zero-scales the mission's velocity vector while keeping the PID loops active.
- **Benefit:** This is safer than switching to `LOITER` and back to `AUTO` , as the mission state machine (sequence number, jump counters, timers) remains exactly where it was.

PRACTICAL USE CASES

1. **Bird Incursion:**
   - *Scenario:* A pilot sees a hawk approaching the drone during a mapping grid.
   - *Action:* The pilot clicks "Pause." The drone stops moving immediately. Once the bird leaves, the pilot clicks "Continue."
2. **Visual Inspection:**
   - *Scenario:* During an autonomous tower inspection, the inspector needs more time to look at a specific bracket.
   - *Action:* Send `Pause` , inspect, then `Continue` .

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





KEY PARAMETERS

- `WPNAV_ACCEL`: Determines how quickly the drone stops when paused.

KEY CODEBASE LOCATIONS

- **ArduCopter/mode_auto.cpp:2339**: Implementation of pause/resume methods.

DO_RETURN_PATH_START (188)

SUMMARY

The `DO_RETURN_PATH_START` command defines the beginning of a safe return segment in a mission. This is an advanced "failsafe-aware" mission marker that allows a vehicle to rejoin a pre-defined safe corridor if an RTL is triggered, rather than flying a direct (and potentially dangerous) line back to Home.

STATUS

Supported (ArduPlane / All Vehicles using Mission re-entry)

DIRECTIONALITY

- **RX (Receive):** The vehicle receives this command as part of a mission upload.

MISSION STORAGE (AP_MISSION)

- **Segment Boundary:** This command marks the start of the safe segment. The segment ends at the next encountered `DO_LAND_START` or mission end.
- **Coordinates:** Optional. If provided, they define the geometric start of the segment.

EXECUTION (ENGINEER'S VIEW)

CORRIDOR LOGIC

When a vehicle using missions for return-to-launch triggers a failsafe:

1. **Search:** The autopilot calls `get_return_path_start()` (AP_Mission.cpp).
2. **Closest re-entry:** It identifies the segment between `DO_RETURN_PATH_START` and `DO_LAND_START`.
3. **Orthogonal Projection:** The autopilot calculates the closest point on that line segment to the vehicle's current position.
4. **Action:** The vehicle flies to that "Re-entry Point" and then follows the mission forward towards the landing sequence.

DATA FIELDS (MAVLINK)

- `param1` to `param4`: Unused.
- `x` **(Latitude):** Optional coordinate.
- `y` **(Longitude):** Optional coordinate.
- `z` **(Altitude):** Optional coordinate.

THEORY: CORRIDOR MISSIONS

Standard RTL logic is "Point A to Home." **Corridor Logic** is "Point A to the nearest safe pipe, then through the pipe to Home."

- **Geofence Integration:** If a mission is flown through a narrow canyon or a corridor bounded by Inclusion Fences, a direct RTL would immediately trigger a fence breach.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- **Geometric Join:** ArduPilot mathematically treats the mission between these markers as a vector. The vehicle joins the vector tangentially to ensure it never exits the "safe zone" during the recovery.

PRACTICAL USE CASES

1. **Canyon Mapping:**
   - *Scenario:* A drone is mapping a deep canyon. It is flying at 50m, but the canyon walls rise to 200m on either side.
   - *Action:* Place `DO_RETURN_PATH_START` at the canyon entrance. If the drone loses link deep in the canyon, it will not attempt to climb and fly over the walls; it will fly back along the canyon floor until it exits the marked segment.
2. **Urban BVLOS:**
   - *Scenario:* Flying along a pre-approved corridor between skyscrapers.
   - *Action:* The markers define the safe air-road. Recovery always follows the approved path.

KEY PARAMETERS

- `MIS_OPTIONS`: Must be configured to allow Mission-based RTL.

KEY CODEBASE LOCATIONS

- **libraries/AP_Mission/AP_Mission.cpp:2467**: `get_return_path_start` logic.

DO_SEND_SCRIPT_MESSAGE (217)

SUMMARY

The `DO_SEND_SCRIPT_MESSAGE` command provides a bridge between the static Mission Engine and ArduPilot's dynamic **Lua Scripting** environment. It allows a mission to pass numeric data to a running script, which can then perform complex logic, such as changing flight parameters, interacting with custom hardware, or modifying the mission on-the-fly.

STATUS

Supported (All Vehicles with AP_Scripting enabled)

DIRECTIONALITY

- **RX (Receive):** The vehicle receives this command as part of a mission upload.

MISSION STORAGE (AP_MISSION)

- **Target (Param 1):** An ID used by the Lua script to identify the specific message.
- **Values (Param 2-4):** Three numeric values (p1, p2, p3) passed to the script.
- **Packing:** Stored in the `scripting` content struct.

EXECUTION (ENGINEER'S VIEW)

SCRIPTING BRIDGE

The command utilizes the `AP_Scripting` library (AP_Scripting.cpp).

1. **Event Generation:** When the mission reaches this item, the autopilot generates a `MISSION_ITEM_REACHED` event internally.
2. **Script Polling:** A Lua script running on the flight controller uses the `mission:get_last_script_message()` binding to retrieve the data.
3. **Action:** The script parses the `Param 1` ID and executes the corresponding custom code.
4. **Completion:** The mission advances immediately. The "wait" for the script to finish must be handled manually via a `CONDITION_DELAY` or a script-controlled mission override.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





DATA FIELDS (MAVLINK)

- `param1` **(ID):** Message identifier.
- `param2` **(p1):** First numeric parameter.
- `param3` **(p2):** Second numeric parameter.
- `param4` **(p3):** Third numeric parameter.
- `param5` to `param7`: Unused.

THEORY: EXTENDING THE STATE MACHINE

Missions are typically limited to the commands defined in the MAVLink spec. `DO_SEND_SCRIPT_MESSAGE` transforms the Mission Engine into a **Programmable Logic Controller (PLC)**.

- **Custom Payloads:** A drone carrying a non-standard sensor (e.g., a spectral scanner) can use this command to tell a Lua script to start a specific calibration routine.
- **External Comms:** A script can listen for this message and then send a custom HTTP/UDP packet to an onboard companion computer or a cloud server.

PRACTICAL USE CASES

1. **Dynamic Parameter Tuning:**
   - *Scenario:* A plane needs different PID gains for a low-altitude "nap-of-the-earth" segment.
   - *Action:* `DO_SEND_SCRIPT_MESSAGE (ID: 10, p1: 1)` triggers a Lua script to call `param:set('ATC_RAT_RLL_P', 0.15)`.
2. **Robotic Integration:**
   - *Scenario:* A drone landing on a moving rover needs to "Handshake" with the rover's local network.
   - *Action:* Mission triggers the message, and the script handles the custom socket communication.

KEY PARAMETERS

- `SCR_ENABLE`: Enables the Lua scripting engine.
- `SCR_VM_I_COUNT`: Instruction count (performance) for scripts.

KEY CODEBASE LOCATIONS

- **libraries/AP_Scripting/AP_Scripting.cpp**: Lua binding implementation.
- **ArduCopter/mode_auto.cpp**: Command-to-script event routing.

DO_SET_CAM_TRIGG_DIST (206)

SUMMARY

The `DO_SET_CAM_TRIGG_DIST` command enables automatic camera triggering based on the 2D distance traveled by the vehicle. This is the primary mechanism for aerial mapping (photogrammetry), ensuring consistent image overlap regardless of groundspeed fluctuations caused by wind.

STATUS

Supported (All Vehicles)

DIRECTIONALITY

- **RX (Receive):** The vehicle receives this command as part of a mission upload or as a direct command.

MISSION STORAGE (AP_MISSION)

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- **Distance (Param 1):** The distance in meters between triggers.
- **Shutter (Param 2):** Shutter integration time (typically unused by ArduPilot, which relies on hardware-specific drivers).
- **Trigger Once (Param 3):** If set to 1, the camera triggers immediately upon receiving the command.
- **Mechanism:** Stored in the `cam_trigg_dist` content struct within `AP_Mission`.

EXECUTION (ENGINEER'S VIEW)

TRIGGERING LOGIC

The command utilizes the `AP_Camera` library (AP_Camera.cpp).

1. **Distance Threshold:** The autopilot records the GPS position of the *last* trigger.
2. **Continuous Check:** Every loop, it calculates the 2D horizontal distance (d) from the last trigger point.
3. **Firing:** When $d \geq \text{TriggDist}$, the autopilot sends a signal to the camera shutter (via Relay, PWM, MAVLink, or DroneCAN) and updates the last-trigger position.
4. **Disabling:** Sending this command with `Param 1 = 0` disables automatic triggering.

DATA FIELDS (MAVLINK)

- `param1` **(Distance):** Meters between shots. 0 to disable.
- `param2` **(Shutter):** Integration time (ms).
- `param3` **(Trigger):** 1 to trigger one shot immediately.
- `param4` to `param7`: Unused.

THEORY: PHOTOGRAMMETRY OVERLAP

Mapping accuracy depends on **Frontal Overlap** (O_f). For a camera with sensor height H_s and focal length f, flying at altitude A with a groundspeed V_g:

- **Ground Sample Distance (GSD):** The real-world size of one pixel.
- **Trigger Distance (D):**

$$D = \text{Footprint}_{height} \cdot (1 - O_f)$$

  Using `DO_SET_CAM_TRIGG_DIST` ensures that even if the drone slows down when flying into a headwind, the images are still taken at the mathematically correct spatial intervals to maintain the required O_f for 3D reconstruction.

PRACTICAL USE CASES

1. **Ortho-Mosaic Mapping:**
   - *Scenario:* Mapping a farm at 80\% overlap.
   - *Action:* Mission starts with `DO_SET_CAM_TRIGG_DIST (20m)`. The drone flies the survey grid, and the camera fires every 20 meters.
2. **Corridor Inspection:**
   - *Scenario:* Inspecting a pipeline.
   - *Action:* `DO_SET_CAM_TRIGG_DIST (50m)` ensures high-resolution coverage without filling the SD card with redundant images.

KEY PARAMETERS

- `CAM_TRIGG_TYPE`: Defines if the trigger is via Relay, PWM, or MAVLink.
- `CAM_DURATION`: How long the shutter signal is held high.

KEY CODEBASE LOCATIONS

- **libraries/AP_Camera/AP_Camera.cpp:291**: Implementation of distance-based triggering.
- **ArduCopter/mode_auto.cpp**: Integration with the mission engine.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





DO_SET_HOME (179)

SUMMARY

The `DO_SET_HOME` command redefines the vehicle's "Home" position. The Home position is used as the reference point for RTL (Return to Launch), altitude-above-home calculations, and distance-from-home failsafes.

STATUS

Supported (All Vehicles)

DIRECTIONALITY

- **RX (Receive):** The vehicle receives this command as part of a mission upload or as a direct command (Guided/Command Mode).

MISSION STORAGE (AP_MISSION)

- **Use Current (Param 1):**
  - 1: Use the vehicle's current location as home.
  - 0: Use the Lat/Lon/Alt provided in the command's x/y/z fields.
- **Packing:** Stored in the standard location struct.

EXECUTION (ENGINEER'S VIEW)

LOGIC

The command calls `AP_Vehicle::set_home()` (or vehicle-specific wrappers like `ModeAuto::do_set_home`).

1. **Safety Check:** ArduPilot generally allows setting Home while disarmed. If the vehicle is armed, setting home is typically restricted to the current location to prevent the vehicle from performing an RTL to a dangerously distant or unreachable coordinate.
2. **Coordinate Update:** The EKF (Extended Kalman Filter) uses the new Home as the origin for its local NE (North-East) coordinate system if the "Absolute Altitude" is updated.
3. **GCS Notification:** Upon successful update, the vehicle broadcasts a new `HOME_POSITION` (242) MAVLink message to all connected Ground Control Stations.

DATA FIELDS (MAVLINK)

- `param1` **(Flag):** 1: Current Location, 0: Specified Location.
- `x` **(Latitude):** New home latitude.
- `y` **(Longitude):** New home longitude.
- `z` **(Altitude):** New home altitude.

THEORY: THE EKF ORIGIN VS. HOME

It is important to distinguish between the **EKF Origin** and the **Home Position**.

- **EKF Origin:** The absolute GPS coordinate where the Kalman Filter initialized. It never changes during a flight.
- **Home Position:** A user-defined coordinate used for navigation. It can be changed multiple times.
- **Math:** Internal navigation is done in meters relative to the EKF Origin. When you "Set Home," ArduPilot calculates the meter-offset from the Origin and stores that as the new Home reference.

PRACTICAL USE CASES

1. **Mobile Landing Platform:**

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- *Scenario:* A drone takes off from a moving boat.
- *Action:* As the boat moves, the GCS periodically sends `DO_SET_HOME (Current Location: 1)` to ensure the drone's RTL point stays near the boat.

2. **Long Corridor Missions:**
   - *Scenario:* A drone is inspecting 50km of power lines.
   - *Action:* The mission can include `DO_SET_HOME` commands at safe landing clearings along the route, so if a failsafe occurs, the drone flies to the *nearest* safe point rather than all the way back to the start.

KEY PARAMETERS

- `FS_GCS_ENABL` : GCS failsafe.
- `RTL_ALT` : Altitude used when returning to the (potentially new) home.

KEY CODEBASE LOCATIONS

- **ArduCopter/mode_auto.cpp:1954**: `do_set_home` implementation.
- **libraries/AP_Vehicle/AP_Vehicle.cpp**: Core Home management logic.

DO_SET_RELAY (181) / DO_REPEAT_RELAY (182)

SUMMARY

The `DO_SET_RELAY` and `DO_REPEAT_RELAY` commands provide simple digital control over the vehicle's GPIO pins. Relays are typically used to trigger non-MAVLink hardware like power switches, smoke generators, water pumps, or simple camera shutters.

STATUS

Supported (All Vehicles)

DIRECTIONALITY

- **RX (Receive):** The vehicle receives this command as part of a mission upload or as a direct command (Guided Mode).

MISSION STORAGE (AP_MISSION)

- **Relay Number (Param 1):** The index of the relay to control (typically 0-5).
- **State / Repeat (Param 2):**
  - **181:** 0: Off, 1: On.
  - **182:** The number of times to toggle the relay.
- **Cycle Time (Param 3):** (182 only) The duration of each on/off cycle in seconds.
- **Packing:** Stored in the `relay` or `repeat_relay` content struct within `AP_Mission`.

EXECUTION (ENGINEER'S VIEW)

LOGIC IMPLEMENTATION

The commands utilize the `AP_ServoRelayEvents` library (AP_ServoRelayEvents.cpp).

1. **Pin Mapping:** ArduPilot maps "Relay 0" to a physical GPIO pin on the flight controller via the `RELAY_PIN` parameter.
2. **Toggle Logic (Repeat):**
   - The `delay_ms` is calculated as `Param3 * 500` (half of the cycle time).
   - The relay state is flipped every `delay_ms` until the repeat count is exhausted.
3. **Conflict Handling:** If a new relay command is received for a pin currently running a "Repeat" sequence, the existing sequence is immediately cancelled in favor of the new state.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

DATA FIELDS (MAVLINK)

DO_SET_RELAY (181)

- `param1` **(Index):** Relay instance number.
- `param2` **(State):** 0: OFF, 1: ON.

DO_REPEAT_RELAY (182)

- `param1` **(Index):** Relay instance number.
- `param2` **(Count):** Number of toggle cycles.
- `param3` **(Time):** Cycle time in seconds.

THEORY: PWM VS. GPIO

Most flight controller pins are configured for PWM (Pulse Width Modulation) by default to drive servos.

- **Relay Transformation:** When a pin is assigned to a Relay, the autopilot reconfigures the hardware timer/DMA for that pin into **Digital I/O Mode**.
- **Voltage:** The output is typically 3.3V (standard microcontrollers) or 5V (level-shifted boards), which is used to trigger a transistor or a physical mechanical relay module.

PRACTICAL USE CASES

1. **Water Sprayer (Agri-Drone):**
   - *Scenario:* A crop-dusting drone needs to spray only while over a specific field.
   - *Action:* `WAYPOINT (Start of Field)` → `DO_SET_RELAY (On)` ... `WAYPOINT (End of Field)` → `DO_SET_RELAY (Off)`.
2. **Emergency Strobe:**
   - *Scenario:* A drone needs to flash its high-intensity lights during an RTL.
   - *Action:* Mission starts with `DO_REPEAT_RELAY (Count: 999, Time: 1s)` to create a persistent blink.

KEY PARAMETERS

- `RELAY_PIN`: Defines which physical pin is assigned to which relay instance.
- `RELAY_DEFAULT`: The state (On/Off) the relay enters upon autopilot boot.

KEY CODEBASE LOCATIONS

- **libraries/AP_ServoRelayEvents/AP_ServoRelayEvents.cpp**: Main execution handler.
- **libraries/AP_Relay/AP_Relay.cpp**: Low-level GPIO driver.

DO_SET_RESUME_REPEAT_DIST (215)

SUMMARY

The `DO_SET_RESUME_REPEAT_DIST` command defines a "Rewind Distance" for mission resumption. If a mission is interrupted (e.g., due to a mode change or a battery failsafe), and then resumed, the autopilot will not simply fly to the next waypoint. Instead, it will backtrack along the mission path by the specified distance before continuing forward.

STATUS

Supported (All Vehicles)

DIRECTIONALITY

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- **RX (Receive):** The vehicle receives this command as part of a mission upload.

MISSION STORAGE (AP_MISSION)

- **Distance (Param 1):** The backtrack distance in meters.
- **Packing:** Stored in the internal `_repeat_dist` variable in the `AP_Mission` library.

EXECUTION (ENGINEER'S VIEW)

RESUMPTION LOGIC

ArduPilot handles this command within the `AP_Mission::resume()` function (AP_Mission.cpp).

1. **Interrupt Discovery:** The autopilot tracks the last successful waypoint passed (`LAST_WP_PASSED`).
2. **Rewind Calculation:** If `_repeat_dist > 0`, the autopilot calls `calc_rewind_pos()`.
   - It looks at the segment between the last WP and the current WP.
   - It calculates a 3D coordinate that is `Param1` meters *backwards* along that segment.
3. **Path Re-entry:** The vehicle first flies to this calculated rewind point and then resumes the original mission track.

DATA FIELDS (MAVLINK)

- `param1` **(Distance):** Rewind distance in meters.
- `param2` to `param7`: Unused.

THEORY: THE OVERLAP REQUIREMENT

In mapping and sensor missions, an interrupt usually results in a "data gap."

- **The Hazard:** Most autopilots resume at the *next* waypoint, leaving a section of the mission un-scanned.
- **The Solution:** By setting a repeat distance (e.g., 50 meters), the pilot ensures that the drone "re-scans" the last section of the previous leg, providing a overlap buffer that guarantees data continuity for photogrammetry or LiDAR processing.

PRACTICAL USE CASES

1. **Battery Swap Recovery:**
   - *Scenario:* A mapping drone triggers a low-battery RTL in the middle of a 1km leg.
   - *Action:* The mission includes `DO_SET_RESUME_REPEAT_DIST (100m)`. After the pilot swaps the battery and clicks "Resume," the drone flies back to the point 100m *before* it left the track, ensuring the map has no holes.
2. **Loss of Signal (LOS) Buffer:**
   - *Scenario:* A drone is inspecting a long bridge and loses telemetry.
   - *Action:* Backtracking ensures that any missed photos are re-taken upon reconnection.

KEY PARAMETERS

- `MIS_RESTART`: Affects whether the mission resets entirely or supports this resume logic.

KEY CODEBASE LOCATIONS

- **libraries/AP_Mission/AP_Mission.cpp:158**: `resume` logic and rewind calculation.

DO_SET_REVERSE (194)

SUMMARY

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





The `DO_SET_REVERSE` command tells the autopilot to change its primary direction of travel. This is used by Rovers and Boats to backup, and by Planes equipped with reversible ESCs to perform steep descents or reverse-thrust landings.

STATUS

Supported (ArduPlane, Rover, and Boat)

DIRECTIONALITY

- **RX (Receive):** The vehicle receives this command as part of a mission upload or as a direct command.

MISSION STORAGE (AP_MISSION)

- **Reverse (Param 1):**
  - 0: Forward.
  - 1: Reverse.
- **Packing:** Stored in the internal `p1` field.

EXECUTION (ENGINEER'S VIEW)

ARDUPLANE IMPLEMENTATION

In Plane, the command enables the use of **Negative Thrust** (reverse_thrust.cpp).

1. **ESC Configuration:** Requires a bi-directional ESC (DShot or specialized PWM).
2. **Navigation:** The autopilot maintains its orientation but reverses the throttle output.
3. **Use Case:** Typically used for "Beta Range" aerodynamic braking during a steep approach to clear obstacles on short runways.

ROVER/SUB/BOAT IMPLEMENTATION

In Rover, this command is used to navigate "Tail First."

1. **Ackermann Logic:** For cars, the steering logic is inverted when in reverse.
2. **Pivot Turns:** Skid-steer rovers use this to back out of dead ends.

DATA FIELDS (MAVLINK)

- `param1` **(State):** 1: Reverse, 0: Forward.
- `param2` to `param7`: Unused.

THEORY: THRUST VECTORING AND POLARITY

In standard propulsion, thrust is a scalar value $T \in [0, 1]$. `DO_SET_REVERSE` redefines the propulsion model as a signed value $T \in [-1, 1]$.

- **The Transition:** Transitioning from forward to reverse requires the motor to come to a complete stop (to prevent back-EMF spikes) before spinning in the opposite direction. ArduPilot's motor library manages this "Zero-Crossing" safely.

PRACTICAL USE CASES

1. **Boat Docking:**
   - *Scenario:* An autonomous boat needs to back into a slip.
   - *Action:* `WAYPOINT (Front of Slip)` → `DO_SET_REVERSE (1)` → `WAYPOINT (Back of Slip)`.
2. **STOL (Short Takeoff and Landing):**
   - *Scenario:* A plane landing on a very short mountain strip.
   - *Action:* After `NAV_LAND` flare, the mission triggers `DO_SET_REVERSE (1)` to use the prop as an airbrake.

KEY PARAMETERS

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- `USE_REV_THRUST` : (Plane) Global enable for reverse thrust logic.
- `SERVOX_REVERSED` : (Rover) Hardware-level motor direction.

KEY CODEBASE LOCATIONS

- **ArduPlane/reverse_thrust.cpp**: Plane-specific reverse logic.
- **libraries/AP_Motors/AP_Motors_Class.cpp**: Low-level bidirectional motor control.

DO_SET_ROI (201) / DO_SET_ROI_LOCATION (195) / DO_SET_ROI_NONE (197)

SUMMARY

The "Region of Interest" (ROI) commands instruction the vehicle to point its camera (and potentially its entire airframe) at a specific coordinate or object. This is a critical command for surveillance, cinematography, and inspection missions where the sensor's target is independent of the vehicle's flight path.

STATUS

Supported (All Vehicles)

DIRECTIONALITY

- **RX (Receive):** The vehicle receives this command as part of a mission upload or as a direct command (Guided Mode).

MISSION STORAGE (AP_MISSION)

- **Mode (Param 1):**
  - 0: No ROI (Cancels current ROI).
  - 1: ROI at next waypoint.
  - 2: ROI at specific mission item.
  - 3: ROI at fixed Lat/Lon location.
- **Location (x, y, z):** The GPS coordinates of the target (used for 195 and Mode 3).
- **Mechanism:** Stored as a standard location item or a mode-switch item.

EXECUTION (ENGINEER'S VIEW)

GIMBAL AND AIRFRAME INTEGRATION

ArduPilot handles ROI via the `AP_Mount` and `AutoYaw` libraries (mode_auto.cpp).

1. **3D Geometry:** The autopilot calculates the vector between the vehicle's current 3D position (GPS/INS) and the target 3D coordinate.
2. **Point of Aim:**
   - **Gimbal:** If a stabilized gimbal is present, ArduPilot sends Pitch and Yaw angles to the mount controller.
   - **Airframe:** If no gimbal is present, or if the gimbal lacks a pan (yaw) axis, the multicopter will rotate its entire body (Yaw) to face the target while keeping the camera centered.
3. **Persistence:** An ROI command is "sticky." Once set, the vehicle will continue to track that coordinate even as it flies through subsequent waypoints, until a `DO_SET_ROI_NONE` command is encountered or a new ROI is defined.

DATA FIELDS (MAVLINK)

- `param1` **(Mode):** Selection between location, waypoint, or cancel.
- `param2` **(Item #):** Mission index if mode=2.
- `x` **(Latitude):** Target latitude.
- `y` **(Longitude):** Target longitude.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- `z` **(Altitude):** Target altitude.

THEORY: TRIGONOMETRY OF TRACKING

To point the camera, the autopilot solves for the Azimuth (ψ) and Elevation ($theta$) angles relative to North:

$$\psi = \text{atan2}(\Delta\text{East}, \Delta\text{North})$$

$$\theta = \text{atan2}(\Delta\text{Alt}, \sqrt{\Delta\text{North}^2 + \Delta\text{East}^2})$$

The EKF solution provides high-frequency updates to these angles to ensure the camera remains locked even during aggressive vehicle maneuvering.

PRACTICAL USE CASES

1. **POI (Point of Interest) Orbit:**
   - *Scenario:* Filming a lighthouse from a circling drone.
   - *Action:* `DO_SET_ROI (Lighthouse)` → `NAV_LOITER_TURNS`. The drone circles while the gimbal automatically tilts and pans to keep the lighthouse in the center of the frame.
2. **Static Security Watch:**
   - *Scenario:* A plane is patrolling a perimeter but needs to keep its camera locked on a specific gate.
   - *Action:* `DO_SET_ROI (Gate)` followed by a patrol mission.

KEY PARAMETERS

- `MNT1_TYPE`: Defines the type of gimbal hardware.
- `WP_YAW_BEHAVE`: Must be aware that ROI will override the default "Face Next Waypoint" behavior.

KEY CODEBASE LOCATIONS

- **ArduCopter/mode_auto.cpp:1975**: `do_roi` implementation.
- **libraries/AP_Mount/AP_Mount.cpp**: Core tracking mathematics.

DO_SET_SERVO (183) / DO_REPEAT_SERVO (184)

SUMMARY

The `DO_SET_SERVO` and `DO_REPEAT_SERVO` commands allow the mission script to directly override the PWM (Pulse Width Modulation) output of specific flight controller pins. This is used to control auxiliary hardware like mechanical grippers, gimbal pitches (non-stabilized), or robotic arms.

STATUS

Supported (All Vehicles)

DIRECTIONALITY

- **RX (Receive):** The vehicle receives this command as part of a mission upload or as a direct command.

MISSION STORAGE (AP_MISSION)

- **Channel (Param 1):** The Servo/Output number to control (typically 1-16).
- **PWM Value / Repeat (Param 2):**

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





    - **183:** The target PWM value (typically 1000 to 2000 microseconds).
    - **184:** The target PWM value for the "pulsed" state.
- **Repeat Count (Param 3):** (184 only) The number of times to pulse.
- **Cycle Time (Param 4):** (184 only) The duration of each pulse cycle in seconds.
- **Packing:** Stored in the `servo` or `repeat_servo` content struct.

EXECUTION (ENGINEER'S VIEW)

HANDLING LOGIC

Execution is managed by `AP_ServoRelayEvents::do_set_servo` (AP_ServoRelayEvents.cpp).

1. **Safety Check:** ArduPilot checks the "Servo Function" (`SERVOX_FUNCTION`) of the requested pin.
    - If the pin is assigned to a critical flight function (e.g., `Motor 1` or `Aileron`), the mission command is **rejected** and an info message is sent to the GCS: "ServoRelayEvent: Channel \%d is already in use".
    - The pin must be set to `0` (Disabled), `1` (RCPassThru), or another non-critical auxiliary function.
2. **Output:** The autopilot writes the target PWM value directly to the hardware timer registry for that pin.
3. **Repeat Pattern (184):** The autopilot toggles between the target **PWM Value** and the pin's **Trim Value** at the requested frequency.

DATA FIELDS (MAVLINK)

DO_SET_SERVO (183)

- `param1` **(Index):** Output channel number.
- `param2` **(PWM):** PWM value [1000-2000].

DO_REPEAT_SERVO (184)

- `param1` **(Index):** Output channel number.
- `param2` **(PWM):** PWM value for the pulse.
- `param3` **(Count):** Number of cycles.
- `param4` **(Time):** Cycle time (seconds).

THEORY: THE DUTY CYCLE

Servos are controlled by the width of a pulse sent at a specific frequency (typically 50Hz).

- **Pulse Width (au):** 1000aus is typically "Full Left/Closed," 1500aus is "Center," and 2000aus is "Full Right/Open."
- **Timing:** ArduPilot's main loop handles the logic, but the actual high-precision pulse generation is offloaded to the SoC's hardware timers (DMA/PWM) to ensure jitter-free control.

PRACTICAL USE CASES

1. **Cargo Hook:**
    - *Scenario:* A drone needs to drop a package at a specific waypoint.
    - *Action:* `DO_SET_SERVO (Channel: 9, PWM: 2000)` triggers the hook to open.
2. **Bait Dropper (Fishing Drone):**
    - *Scenario:* Releasing a fishing line into the surf.
    - *Action:* `DO_REPEAT_SERVO (Channel: 10, PWM: 2000, Count: 2, Time: 1s)` ensures the release mechanism triggers twice to prevent a snag.

KEY PARAMETERS

- `SERVOX_FUNCTION` : Must be configured as `Disabled` or `Scripting` for mission commands to work on that channel.
- `SERVOX_MIN/MAX` : Bounds for the PWM output.

KEY CODEBASE LOCATIONS

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- **libraries/AP_ServoRelayEvents/AP_ServoRelayEvents.cpp:30**: Command verification and safety logic.
- **libraries/SRV_Channel/SRV_Channels.cpp**: Low-level servo output manager.

DO_SPRAYER (216)

SUMMARY

The `DO_SPRAYER` command provides mission-level control for agricultural liquid sprayers. It is used to enable or disable the pump and manage the flow rate based on the vehicle's ground speed to ensure even chemical application.

STATUS

Supported (ArduCopter and ArduPlane with AC_Sprayer enabled)

DIRECTIONALITY

- **RX (Receive):** The vehicle receives this command as part of a mission upload or as a direct command.

MISSION STORAGE (AP_MISSION)

- **Enable (Param 1):**
  - 0: Disable Sprayer (Pump Off).
  - 1: Enable Sprayer (Pump On).
- **Packing:** Stored in the internal `p1` field.

EXECUTION (ENGINEER'S VIEW)

SPRAYER CONTROL LOGIC

Execution is handled by the `AC_Sprayer` library (AC_Sprayer.cpp).

1. **Velocity Compensation:** One of the most advanced features of `AC_Sprayer` is speed-scaling. The pump's PWM output is modulated based on groundspeed (V_g).

$$\text{PWM}_{out} = \text{BaseFlow} \cdot \left(\frac{V_{ground}}{V_{nominal}} \right)$$

2. **Spin-up/down:** The controller manages the ramp-up time for the pump to prevent motor surges.
3. **Automatic Cutoff:** ArduPilot automatically disables the sprayer if the vehicle comes to a stop or enters a failsafe state, preventing chemical pooling.

DATA FIELDS (MAVLINK)

- `param1` **(State):** 0: Disable, 1: Enable.
- `param2` to `param7`: Unused.

THEORY: LINEAR APPLICATION RATE

In precision agriculture, the goal is to apply a specific volume of liquid per unit of area (L/Ha).

- **The Challenge:** Drones decelerate at waypoints. If the pump stayed at a constant speed, the area near the waypoint would receive a massive overdose of chemicals.
- **The Solution:** The `AC_Sprayer` library integrates with the navigation controller to vary the pump speed in real-time, maintaining a constant **Linear Application Rate** regardless of flight dynamics.

PRACTICAL USE CASES

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





1. **Crop Mapping & Spraying:**
    - *Scenario:* Spraying a rectangular field.
    - *Action:* `WAYPOINT (A)` → `DO_SPRAYER (1)` → `WAYPOINT (B)`. The drone flies the transect, spraying only between A and B, and adjusting the flow as it accelerates out of A and decelerates into B.
2. **Mosquito Abatement:**
    - *Scenario:* Targeting a specific swampy area.
    - *Action:* Mission uses ROI to circle the swamp while `DO_SPRAYER` manages the payload.

KEY PARAMETERS

- `SPRAY_ENABLE` : Global switch.
- `SPRAY_PUMP_RATE` : Nominal pump speed (\%).
- `SPRAY_SPEED_MIN` : Groundspeed below which the pump is shut off.

KEY CODEBASE LOCATIONS

- **libraries/AC_Sprayer/AC_Sprayer.cpp**: Flow-scaling and pump control.

DO_WINCH (42600)

SUMMARY

The `DO_WINCH` command provides advanced control for robotic winches slung beneath a vehicle. It supports releasing a specific length of cable, controlling the cable's rate of descent, or "relaxing" the motor. This is the primary command for air-to-ground delivery systems where the drone must remain high while the payload descends.

STATUS

Supported (ArduCopter)

DIRECTIONALITY

- **RX (Receive):** The vehicle receives this command as part of a mission upload or as a direct command.

MISSION STORAGE (AP_MISSION)

- **Action (Param 2):**
    - 0: Relax (Motor powered off).
    - 1: Position (Release length).
    - 2: Rate (Descent speed).
- **Length (Param 3):** Distance in meters (Negative to reel in).
- **Rate (Param 4):** Speed in m/s.
- **Packing:** Stored in the `winch` content struct.

EXECUTION (ENGINEER'S VIEW)

WINCH CONTROL STATE MACHINE

The command is handled by the `AP_Winch` library (AP_Winch.cpp).

1. **Safety Verification:** ArduPilot checks the `WINCH_TYPE`. It supports PWM winches (servos), DroneCAN winches, and Daedalus-protocol winches.
2. **Position Control:** If `Action: 1` is selected, the autopilot uses a PID loop to move the cable to the absolute length requested. This typically requires an encoder on the winch motor.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





3. **Rate Control:** If `Action: 2` is selected, the autopilot maintains a steady cable velocity. This is safer for heavy payloads to prevent "yanking" the drone out of the air if the cable snags.
4. **Completion:** The mission advances immediately. The winch continues to work in the background. It is highly recommended to follow a winch command with a `CONDITION_DELAY` if the next mission leg requires the payload to be fully deployed or retracted.

DATA FIELDS (MAVLINK)

- `param1` **(Instance):** Winch number (typically 0).
- `param2` **(Action):** 0:Relax, 1:Length, 2:Rate.
- `param3` **(Length):** m.
- `param4` **(Rate):** m/s.

THEORY: TENSION AND PENDULUMS

Slung loads introduce **Pendulum Dynamics** into the drone's flight model.

- **Oscillation:** A long cable (L) has a natural frequency $f = \frac{1}{2\pi}\sqrt{\frac{g}{L}}$.
- **Damping:** ArduPilot's position controller is aware of the cable length if `DO_WINCH` is used, allowing the EKF to potentially compensate for the shifting center of gravity.
- **Safety:** The `Relax (0)` action is critical for emergencies. If the payload snags on a tree, relaxing the winch allows the drone to fly away rather than being pulled into the ground.

PRACTICAL USE CASES

1. **Cable-Drop Delivery:**
   - *Scenario:* Delivering a parcel to a balcony.
   - *Action:* Drone hovers 10m above the balcony. `DO_WINCH (Action: 1, Length: 10m)` lowers the parcel. `[DO_GRIPPER](/mission-planning/do-commands.html#DO_GRIPPER) (Release)` drops it. `DO_WINCH (Action: 1, Length: -10m)` reels the hook back in.
2. **Water Sampling:**
   - *Scenario:* Lowering a sensor into a lake.
   - *Action:* `DO_WINCH (Action: 2, Rate: 0.5m/s)` ensures a smooth entry for the sensor.

KEY PARAMETERS

- `WINCH_TYPE` : Hardware selection.
- `WINCH_MAX_LENGTH` : Safety limit to prevent the motor from unspooling the entire drum.

KEY CODEBASE LOCATIONS

- **libraries/AP_Winch/AP_Winch.cpp**: Main driver logic.
- **ArduCopter/mode_auto.cpp:1997**: Mission command intake.

IMAGE_START_CAPTURE (2000) / IMAGE_STOP_CAPTURE (2001)

SUMMARY

The `IMAGE_START_CAPTURE` and `IMAGE_STOP_CAPTURE` commands provide precise control over high-resolution still image acquisition. They support single-shot, interval-based (time), and burst-based capture modes, making them the standard choice for professional survey and reconnaissance missions.

STATUS

Supported (All Vehicles with AP_Camera enabled)

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





DIRECTIONALITY

- **RX (Receive):** The vehicle receives this command as part of a mission upload or as a direct command.

MISSION STORAGE (AP_MISSION)

- **Instance (Param 1):** The target camera (1-6). 0 for "All."
- **Interval (Param 2):** Time between shots in seconds.
- **Total Images (Param 3):** Number of images to take. 0 for "Continuous/Unlimited."
- **Start Seq (Param 4):** Sequence number for the first image (often ignored by ArduPilot storage).
- **Packing:** Stored in the `image_start_capture` content struct.

EXECUTION (ENGINEER'S VIEW)

CAPTURE LOGIC

Execution is handled by `AP_Camera` (AP_Camera.cpp).

1. **Single vs. Multiple:**
   - If `Param 3 = 1`, ArduPilot triggers a single `take_picture()` event.
   - If `Param 3 > 1` or `0`, it initializes a timer-based loop `take_multiple_pictures()`.
2. **Hardware Handshake:** For MAVLink-enabled cameras (e.g., Sony Airpeak or specialized gimbals), the autopilot sends the `IMAGE_START_CAPTURE` packet down the bus to the camera. For simple cameras, it uses the Relay/PWM shutter trigger.
3. **Termination:** `IMAGE_STOP_CAPTURE` (2001) instantly kills any running timers and sends a "Stop" packet to MAVLink backends.

DATA FIELDS (MAVLINK)

- `param1` **(ID):** Camera ID.
- `param2` **(Interval):** s.
- `param3` **(Total):** Count.
- `param4` **(Seq):** Start number.

THEORY: THE DATA PIPELINE

Still images capture significantly more detail than video frames because they use the sensor's full resolution without compression artifacts.

- **Storage Bandwidth:** Capturing 42MP images every 1.5 seconds requires a high-speed UHS-II SD card. ArduPilot's mission engine handles the *triggering*, but the camera backend manages the actual file I/O.
- **Feedback:** ArduPilot listens for `CAMERA_IMAGE_CAPTURED` messages from smart backends to log the precise GPS coordinate where each image was actually written to disk.

PRACTICAL USE CASES

1. **High-Res Photogrammetry:**
   - *Scenario:* Mapping a construction site.
   - *Action:* `IMAGE_START_CAPTURE (Interval: 2s, Total: 0)`. The drone flies the grid, snapping every 2 seconds until the mission ends.
2. **Point of Interest Burst:**
   - *Scenario:* Inspecting a cracked turbine blade.
   - *Action:* Mission hovers at the blade. `IMAGE_START_CAPTURE (Interval: 0.5s, Total: 10)`. The drone captures 10 high-speed shots to ensure a clear image is obtained despite vibration.

KEY PARAMETERS

- `CAM_TRIGG_TYPE`: Defines the physical trigger method.
- `CAM_FEEDBACK_PIN`: Used to log image coordinates from a camera's "Hot Shoe" signal.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





KEY CODEBASE LOCATIONS

- **libraries/AP_Camera/AP_Camera.cpp:350**: Still image capture handler.
- **libraries/AP_Camera/AP_Camera_MAVLink.cpp**: MAVLink camera bridge.

NAV_ALTITUDE_WAIT (83)

SUMMARY

The `NAV_ALTITUDE_WAIT` command is a specialized mission item designed for high-altitude balloon launches or gliders. It puts the vehicle into an "Idle" or "Wait" state until it reaches a target altitude or a specific vertical speed, allowing for autonomous activation after a balloon burst or a release from a mother-ship.

STATUS

Supported (ArduPlane Only)

DIRECTIONALITY

- **RX (Receive):** The vehicle receives this command as part of a mission upload.

MISSION STORAGE (AP_MISSION)

- **Altitude (Param 1):** Target altitude (meters).
- **Descent Rate (Param 2):** Vertical speed (m/s) to trigger completion (e.g., detecting a burst).
- **Wiggle Time (Param 3):** Frequency (seconds) to move control surfaces to prevent freezing in extreme cold.
- **Packing:** Stored in the `altitude_wait` content struct.

EXECUTION (ENGINEER'S VIEW)

HIGH ALTITUDE LOGIC

The command is managed in `Plane::do_altitude_wait` (commands_logic.cpp).

1. **Idle State:** The autopilot sets `auto_state.idle_mode = true`. In this state, servos are held at trim, and the motor is typically disabled.
2. **The "Wiggle":** At high altitudes ($> 15km$), temperatures drop below $-50^{\circ}C$. Lubricants in servos can thicken or freeze. ArduPilot uses `Param 3` to periodically cycle the servos, using internal friction to generate enough heat to keep the linkages moving.
3. **Burst Detection:** The autopilot monitors the vertical velocity (V_z). If V_z becomes more negative than `Param 2` (indicating the balloon has burst and the vehicle is falling), it completes the command immediately.
4. **Completion:** Once the target altitude is reached or a burst is detected, the mission advances to the next item (usually a `NAV_TAKEOFF` or `WAYPOINT`), and full flight control is restored.

DATA FIELDS (MAVLINK)

- `param1` **(Alt):** Target altitude (m).
- `param2` **(Descent):** Descent rate (m/s).
- `param3` **(Wiggle):** Time between wiggles (s).
- `param4` to `param7`: Unused.

THEORY: ENERGY CONSERVATION IN NEAR-SPACE

In a balloon-launch mission, the vehicle is a passenger for 90\% of the ascent.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- **Thermal Management:** Passive electronics cooling is non-existent in the thin upper atmosphere. By idling the CPU and servos, the vehicle minimizes internal heat generation until the "Real" flight begins.
- **Pressure Dynamics:** The autopilot uses the EKF's altitude solution, which fuses barometric and GPS data. At very high altitudes, barometric pressure becomes non-linear; ArduPilot's `AP_Baro` library handles the transition to GPS-dominant altitude sensing safely.

PRACTICAL USE CASES

1. **Weather Balloon Glider:**
   - *Scenario:* A glider is carried to 30,000m by a balloon.
   - *Action:* `NAV_ALTITUDE_WAIT (Alt: 30000, Descent: 5m/s, Wiggle: 60s)`. The drone stays "asleep" during the 2-hour ascent, wiggling every minute to stay limber, and wakes up the moment the balloon pops.
2. **Drop-Test:**
   - *Scenario:* Dropping a test airframe from a larger plane.
   - *Action:* Mission starts with `NAV_ALTITUDE_WAIT` to detect the rapid descent after release.

KEY PARAMETERS

- `TKOFF_THR_DELAY`: Often used after this command to delay motor start until clear of the balloon debris.

KEY CODEBASE LOCATIONS

- **ArduPlane/commands_logic.cpp:90**: Mission command initialization.
- **ArduPlane/pullup.cpp**: Transition to active flight after wait.

NAV_ATTITUDE_TIME (42703)

SUMMARY

The `NAV_ATTITUDE_TIME` command instructs the vehicle to maintain a specific attitude (Roll, Pitch, and Yaw) and a constant climb/descent rate for a set duration. This is an advanced "Open Loop" navigation command used for airframe testing, specialized physics research, or high-speed dashes where GPS-based position hold is not required.

STATUS

Supported (ArduCopter)

DIRECTIONALITY

- **RX (Receive):** The vehicle receives this command as part of a mission upload.

MISSION STORAGE (AP_MISSION)

- **Time (Param 1):** Duration in seconds.
- **Roll (Param 2):** Degrees.
- **Pitch (Param 3):** Degrees.
- **Yaw (Param 4):** Degrees.
- **Climb Rate (x):** Meters per second.
- **Packing:** Stored in the `nav_attitude_time` content struct.

EXECUTION (ENGINEER'S VIEW)

DIRECT ATTITUDE CONTROL

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





The command bypasses the standard "Waypoint-to-Waypoint" navigation and interacts directly with the **Attitude Controller** (mode_auto.cpp:739).

1. **Targeting:** The autopilot feeds the requested Roll, Pitch, and Yaw angles to the `AC_AttitudeControl` library.
2. **Vertical Control:** The vertical position controller maintains the climb rate specified in the `x` field (Param 5).
3. **Horizontal Behavior:** The drone **drifts with the wind**. Because no Lat/Lon target is provided, the drone behaves as if it were in a manual mode (like AltHold or Stabilize) but with a robotic pilot holding the sticks at fixed angles.
4. **Completion:** The mission advances once the system time exceeds `Param 1` seconds from the start of the command.

DATA FIELDS (MAVLINK)

- `param1` **(Time):** s.
- `param2` **(Roll):** deg.
- `param3` **(Pitch):** deg.
- `param4` **(Yaw):** deg.
- `x` **(Climb Rate):** m/s.

THEORY: FLIGHT DYNAMICS RESEARCH

`NAV_ATTITUDE_TIME` is the primary tool for **Empirical Aerodynamic Modeling**.

- **The Problem:** GPS navigation masks the "True" physics of the drone by constantly correcting for error.
- **The Solution:** By flying at a fixed pitch and roll (e.g., 5 degrees pitch forward) for 10 seconds, engineers can measure the resulting airspeed to calculate the airframe's **Drag Coefficient** (C_d) and **Thrust Curve**.
- **Safety:** It is critical to ensure enough "Clear Air" is available, as the vehicle will move in the direction of the lean without regard for geofences or waypoints during the execution.

PRACTICAL USE CASES

1. **High-Speed Dash:**
   - *Scenario:* A drone needs to cross a field as fast as possible.
   - *Action:* `NAV_ATTITUDE_TIME (Pitch: -45, Time: 5s)`. The drone tilts aggressively and accelerates until the timer pops, ignoring GPS braking logic.
2. **Structural Vibration Testing:**
   - *Scenario:* Identifying resonant frequencies at specific bank angles.
   - *Action:* A script cycles through various `NAV_ATTITUDE_TIME` commands while recording IMU data.

KEY PARAMETERS

- `ANGLE_MAX`: Still limits the absolute maximum tilt to prevent tip-over.
- `ATC_INPUT_TC`: Determines how quickly the drone snaps to the requested attitude.

KEY CODEBASE LOCATIONS

- **ArduCopter/mode_auto.cpp:739**: Command intake.
- **ArduCopter/mode_auto.cpp:2327**: Timer verification.

NAV_DELAY (93)

SUMMARY

The `NAV_DELAY` command pauses the vehicle's navigation progress for a specified duration or until a specific UTC time. Unlike `CONDITION_DELAY` (which is a mission-logic gate), `NAV_DELAY` is a navigation-level command that usually involves the vehicle hovering or loitering at its current location.

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





STATUS

Supported (All Vehicles)

DIRECTIONALITY

- **RX (Receive):** The vehicle receives this command as part of a mission upload.

MISSION STORAGE (AP_MISSION)

- **Duration (Param 1):** Seconds to delay (relative).
- **Time (Param 2-4):** Hour, Minute, and Second (UTC) for absolute time delay.
- **Packing:** Stored in the `nav_delay` content struct.

EXECUTION (ENGINEER'S VIEW)

DELAY LOGIC

Execution is handled by `ModeAuto::do_nav_delay` (mode_auto.cpp).

1. **Timer Modes:**
   - **Relative:** The timer starts the moment the vehicle reaches the waypoint.
   - **Absolute (UTC):** The mission will not proceed until the onboard GPS clock matches the requested UTC time. This requires a valid GPS lock and PPS (Pulse Per Second) synchronization.
2. **Vehicle State:** While delaying, the vehicle maintains its current coordinates.
   - **Copter:** Position Hold (Loiter).
   - **Plane:** Loiter (Orbit).
3. **Resumption:** Once `millis() - start_time > duration`, the mission state machine is released to the next item.

DATA FIELDS (MAVLINK)

- `param1` **(Delay):** Seconds.
- `param2` **(Hour):** UTC Hour [0-23].
- `param3` **(Min):** UTC Minute [0-59].
- `param4` **(Sec):** UTC Second [0-59].

THEORY: SYNCHRONIZED SWARMS

The UTC delay is a foundational tool for **Multi-Vehicle Synchronization**.

- **The Problem:** GPS signals take different times to reach different drones. Onboard clocks can drift.
- **The Solution:** By using UTC time (derived from the atomic clocks on GPS satellites), multiple drones can be commanded to "Start Scan" at the exact same microsecond, regardless of when they took off or how far they traveled.

PRACTICAL USE CASES

1. **Golden Hour Photography:**
   - *Scenario:* A drone needs to take a photo at exactly sunset.
   - *Action:* `NAV_WAYPOINT` → `NAV_DELAY (UTC Time of Sunset)`. The drone flies to the spot and orbits until the lighting is perfect.
2. **Coordinated Drop:**
   - *Scenario:* Two drones dropping a heavy net.
   - *Action:* Both missions use `NAV_DELAY` to synchronize the release at the same UTC second.

KEY PARAMETERS

- `GPS_TYPE` : Required for UTC time accuracy.
- `WP_LOITER_RAD` : (Plane) Orbit size during delay.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





KEY CODEBASE LOCATIONS

- **ArduCopter/mode_auto.cpp:723**: Mission command intake.
- **ArduCopter/mode_auto.cpp:2186**: Relative timer verification.

NAV_FENCE_RETURN_POINT (5000) / VERTEX (5001-5002) / CIRCLE (5003-5004)

SUMMARY

The `NAV_FENCE` family of commands defines the geometry of ArduPilot's **Geofence** system directly within the mission list. This allows the vehicle to carry its own "Containment Logic" in non-volatile memory, ensuring that even if the ground station link is lost, the drone remains bound by the pre-approved spatial boundaries.

STATUS

Supported (All Vehicles)

DIRECTIONALITY

- **RX (Receive):** The vehicle receives these commands during a mission upload (specifically when using the MAVLink Fence Protocol).

COMMANDS AND STORAGE

- **5000: Return Point:** The "Safe Zone" where the drone will fly if a fence breach occurs. (Only 1 allowed).
- **5001: Polygon Inclusion:** The drone **must** stay inside the shape defined by these vertices.
- **5002: Polygon Exclusion:** The drone **must not** enter the shape.
- **5003: Circle Inclusion:** A circular keep-in zone.
- **5004: Circle Exclusion:** A circular keep-out zone.
- **Mechanism:** Stored as specialized location items in the `AP_Mission` buffer.

EXECUTION (ENGINEER'S VIEW)

CONTAINMENT MATHEMATICS

The `AC_Fence` library implements **Point-in-Polygon (PIP)** algorithms to monitor the vehicle's position.

1. **Ray Casting:** For polygons (5001, 5002), the autopilot mathematically casts a ray from the vehicle's position. If the number of intersections with the polygon edges is odd, the vehicle is inside.
2. **Distance-Squared:** For circles (5003, 5004), the autopilot calculates the 2D distance:
$$D^2 = (Lat_v - Lat_c)^2 + (Lon_v - Lon_c)^2$$
If $D > \text{Radius}$ for an Inclusion circle, a breach is triggered.
3. **The Floor and Ceiling:** While not defined by these commands, the Geofence system also monitors altitude via global parameters like `FENCE_ALT_MAX`.

DATA FIELDS (MAVLINK)

- `param1` **(Count/Radius):** Vertex count for polygons, or Radius (m) for circles.
- `x` **(Latitude):** Vertex coordinate.
- `y` **(Longitude):** Vertex coordinate.
- `z` **(Altitude):** Reserved.

THEORY: THE "SAFE PIPE"

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





`NAV_FENCE` commands enable **Mission-Geometric Coupling**.

- **The Problem:** A drone is programmed to fly a path. But what if a sensor fails and the drone drifts?
- **The Solution:** By wrapping the `NAV_WAYPOINT` items with a `NAV_FENCE_POLYGON_VERTEX_INCLUSION` sequence, the operator defines a high-integrity "Pipe." The drone is physically unable to exit this pipe while the mission is active.

PRACTICAL USE CASES

1. **Sensitive Airspace Protection:**
   - *Scenario:* A drone is mapping near an airport runway.
   - *Action:* Use `NAV_FENCE_POLYGON_VERTEX_EXCLUSION` to draw a box around the runway. The drone will treat this as a "Solid Object" and trigger a failsafe if it approaches the boundary.
2. **Indoor/Hangar Safety:**
   - *Scenario:* Flying inside a warehouse.
   - *Action:* `NAV_FENCE_CIRCLE_INCLUSION` centered on the hangar to ensure the drone never crashes into the walls if it loses its local positioning fix.

KEY PARAMETERS

- `FENCE_TYPE`: Bitmask selection of which fences are active (Circle, Polygon, etc.).
- `FENCE_MARGIN`: The distance (meters) before the boundary where the drone will begin to brake.

KEY CODEBASE LOCATIONS

- **libraries/AC_Fence/AC_Fence.cpp**: Ray-casting and distance math.
- **libraries/GCS_MAVLink/MissionItemProtocol_Fence.cpp**: Fence packet parsing.

NAV_LAND (21)

SUMMARY

The `NAV_LAND` command initiates the final descent and landing sequence. This command transitions the vehicle from flight to a grounded state, typically concluding with the motors disarming once the autopilot detects a successful touchdown.

STATUS

Supported (All Vehicles)

DIRECTIONALITY

- **RX (Receive):** The vehicle receives this command as part of a mission upload or as a direct command (Guided Mode).

MISSION STORAGE (AP_MISSION)

- **Packet Param 1 (Abort Alt):**
  - **Plane:** Stored in the internal `p1` field as a 16-bit integer (meters). This defines the altitude to climb to if the landing is aborted.
- **Packet Param 4 (Yaw):**
  - **Plane:** Stored in the `loiter_ccw` field if `param4` is negative (used for deepstall direction).

EXECUTION (ENGINEER'S VIEW)

ARDUCOPTER IMPLEMENTATION

In Copter, the command triggers `ModeAuto::do_land` (mode_auto.cpp).

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





1. **Vertical Descent:** The copter descends at the speed specified by `LAND_SPEED`.
2. **Touchdown Detection:** The flight controller monitors the throttle and vertical velocity. If the throttle is at minimum and the altitude is not changing for a specific duration, it sets the "Landed" flag.
3. **Disarm:** Once "Landed" is confirmed, the motors are automatically disarmed.
4. **Repositioning:** If `LAND_REPOSITION` is enabled, the pilot can "nudge" the copter horizontally during the descent to avoid obstacles.

ARDUPLANE IMPLEMENTATION

In Plane, the logic resides in the `AP_Landing` library and `Plane::do_land`.

1. **Flare Logic:** The plane follows a glideslope until it reaches the `LAND_FLARE_ALT`. At this point, it raises the nose and cuts the throttle to "flare" for a smooth touchdown.
2. **Abort Support:** If the pilot triggers an abort or the vehicle enters a go-around state, it climbs to the "Abort Altitude" specified in the command's `param1`.
3. **Deepstall (Advanced):** For airframes without wheels, Plane supports "Deepstall" landing, where the aircraft intentionally stalls its wing at high altitude to fall vertically onto a soft target (like a net or tall grass).

DATA FIELDS (MAVLINK)

- `param1` **(Abort Alt):** Altitude to climb to on abort (Plane only).
- `param2`**:** Empty.
- `param3`**:** Empty.
- `param4` **(Yaw):** Desired yaw heading (Copter only).
- `x` **(Latitude):** Target landing point.
- `y` **(Longitude):** Target landing point.
- `z` **(Altitude):** Target altitude (typically 0).

THEORY: THE GROUND EFFECT

As a multicopter nears the ground (within ~0.5m), it enters **Ground Effect**. The downwash from the rotors creates a high-pressure cushion that increases lift efficiency.

- **The Hazard:** If the autopilot is not tuned correctly, this extra lift can cause the vehicle to "bounce" off the cushion, making it difficult to detect a true touchdown.
- **The Code:** ArduPilot's landing detector uses a robust statistical filter to distinguish between ground effect bounces and a firm touchdown.

PRACTICAL USE CASES

1. **Standard Mission End:**
   - *Scenario:* A mapping mission completes its grid and returns to the takeoff point.
   - *Action:* The last item is `NAV_LAND` at the Home coordinates.
2. **Emergency Forced Landing:**
   - *Scenario:* The battery reaches a critical level during a mission.
   - *Action:* The GCS or onboard script sends a `MAV_CMD_NAV_LAND` at the vehicle's current location to terminate the flight safely.

KEY PARAMETERS

- `LAND_SPEED` : Descent rate in cm/s during final landing.
- `LAND_ALT_LOW` : Altitude (cm) below which the land speed is reduced to the "slow" rate.
- `LAND_FLARE_ALT` : (Plane) Altitude to begin the flare.

KEY CODEBASE LOCATIONS

- **ArduCopter/mode_auto.cpp:1651**: `do_land` implementation.
- **ArduPlane/commands_logic.cpp:402**: `do_land` implementation.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





NAV_LOITER_TIME (19)

SUMMARY

The `NAV_LOITER_TIME` command instructs the vehicle to fly to a location and loiter (hover or circle) for a specific duration in seconds. The timer begins only after the vehicle has reached the target coordinate.

STATUS

Supported (All Vehicles)

DIRECTIONALITY

- **RX (Receive):** The vehicle receives this command as part of a mission upload.

MISSION STORAGE (AP_MISSION)

- **Time (Param 1):** Stored in the internal `p1` field as a 16-bit integer (seconds).
- **Significance:** Since it uses all 16 bits for time, there is **no room** in the packed mission structure to store a custom radius for this specific command. It will always use the vehicle's default loiter radius.

EXECUTION (ENGINEER'S VIEW)

ARDUCOPTER IMPLEMENTATION

In Copter, the command triggers `ModeAuto::do_loiter_time` (mode_auto.cpp).

1. **Waypoint Entry:** The copter flies to the target Lat/Lon/Alt using the standard S-Curve path.
2. **Timer Start:** The `loiter_time` variable is initialized to 0. Once `reached_wp_destination()` returns true, the current system time is recorded.
3. **Completion:** The command completes when:

$$\text{millis}() - \text{startTime} \geq \text{Param1} \cdot 1000$$

ARDUPLANE IMPLEMENTATION

In Plane, the logic resides in `Plane::verify_loiter_time`.

1. **Target Radius:** Always uses the global `WP_LOITER_RAD` parameter.
2. **Timer Logic:** Similar to Copter, the timer starts only when `reached_loiter_target()` is true.
3. **Exit Heading:** Once the time expires, Plane transitions to a secondary goal: **verify_loiter_heading**. It will continue orbiting until it points toward the next mission item, ensuring it exits the circle on a tangent.

DATA FIELDS (MAVLINK)

- `param1` **(Time):** Duration to loiter in seconds.
- `param2` **:** Empty.
- `param3` **:** (Ignored by ArduPilot storage).
- `param4` **(Yaw):** Desired Yaw angle (deg).
- `x` **(Latitude):** Target location.
- `y` **(Longitude):** Target location.
- `z` **(Altitude):** Target altitude.

THEORY: TIME VS. TURNS

While `NAV_LOITER_TURNS` is distance-dependent, `NAV_LOITER_TIME` is purely temporal.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- **Wind Impact:** In a high-wind scenario, a Plane might complete fewer turns in 60 seconds than in a no-wind scenario because it spends more time fighting the headwind.
- **Consistency:** For a Copter (multirotor), `LOITER_TIME` is the preferred method for creating pauses in a mission (e.g., waiting for a camera buffer to clear).

PRACTICAL USE CASES

1. **Timed Surveillance:**
   - *Scenario:* A security drone must orbit a gate for exactly 5 minutes every hour.
   - *Action:* Use `NAV_LOITER_TIME` with `Time = 300`.
2. **Sensor Warm-up:**
   - *Scenario:* A specialized gas sensor requires 30 seconds of stable airflow to calibrate.
   - *Action:* Insert a `NAV_LOITER_TIME` at the start of the survey grid to allow the sensor to stabilize.

KEY PARAMETERS

- `WP_LOITER_RAD` : (Plane) Controls the circle size.
- `LOITER_REPOSITION` : (Copter) Allows the pilot to "nudge" the loiter position with stick inputs without breaking the mission.

KEY CODEBASE LOCATIONS

- **ArduCopter/mode_auto.cpp:1738**: `do_loiter_time` implementation.
- **ArduPlane/commands_logic.cpp:710**: `verify_loiter_time` implementation.

NAV_LOITER_TO_ALT (31)

SUMMARY

The `NAV_LOITER_TO_ALT` command instructions the vehicle to fly to a location and loiter until a target altitude is reached. This is an essential "climb/descend" command for missions where altitude changes must be handled safely before continuing to the next waypoint.

STATUS

Supported (All Vehicles)

DIRECTIONALITY

- **RX (Receive):** The vehicle receives this command as part of a mission upload.

MISSION STORAGE (AP_MISSION)

- **Radius (Param 2):** Stored in the internal `p1` field as a 16-bit integer (meters).
- **Significance:** This command *does* support a custom radius in EEPROM storage, unlike `NAV_LOITER_TIME`.

EXECUTION (ENGINEER'S VIEW)

ARDUCOPTER IMPLEMENTATION

In Copter, the command triggers `ModeAuto::do_loiter_to_alt` (mode_auto.cpp).

1. **Vertical Convergence:** The copter holds the XY position while the vertical controller (Z) targets the altitude specified in `param7` (packet's `z` field).
2. **Completion:** The command completes only when:
   - `reached_destination_xy` is true (vehicle is over the coordinate).

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

- `reached_alt` is true (vertical error is within tolerance).

ARDUPLANE IMPLEMENTATION

In Plane, the logic resides in `Plane::verify_loiter_to_alt`.

1. **Stuck Detection:** ArduPilot includes a safety feature for planes. If the plane has been loitering but is **unable to achieve the target altitude** (e.g., due to low power or extreme downdrafts), it will eventually time out and report "Loiter to alt was stuck" to the GCS, allowing the mission to continue rather than circling until the battery dies.
2. **Exit Strategy:** Once altitude is reached, the plane continues loitering until its heading aligns with the next waypoint (Tangent Exit).

DATA FIELDS (MAVLINK)

- `param1` **(Heading Required):** If 1, the aircraft will not leave the loiter until heading toward the next waypoint (Plane only).
- `param2` **(Radius):** Orbit radius in meters.
- `param3`**:** Empty.
- `param4` **(XTrack):** Exit tangent control (Plane).
- `x` **(Latitude):** Target location.
- `y` **(Longitude):** Target location.
- `z` **(Altitude):** Target altitude.

THEORY: SPIRAL CLIMBS AND ENERGY MANAGEMENT

For fixed-wing aircraft, `LOITER_TO_ALT` is safer than a direct waypoint climb.

- **Stall Prevention:** By climbing in a circle, the aircraft maintains a consistent airspeed and bank angle. A direct climb might result in a high pitch angle and potential stall if the autopilot attempts to climb too aggressively.
- **Thermal Hunting:** In glider modes (SOAR), `LOITER_TO_ALT` is used to stay within a rising thermal until a safe cruise altitude is reached.

PRACTICAL USE CASES

1. **Mountain Clearance:**
   - *Scenario:* A drone needs to cross a 2000m ridge from a takeoff point at 500m.
   - *Action:* Place a `NAV_LOITER_TO_ALT` at 2100m before the ridge waypoint. This ensures the drone circles and gains the required altitude *before* attempting the crossing.
2. **Safe Landing Approach:**
   - *Scenario:* A plane is returning at high altitude and needs to descend to 50m for a landing flare.
   - *Action:* Use `NAV_LOITER_TO_ALT` to spiral down to 50m over the runway threshold, preventing an overspeed approach.

KEY PARAMETERS

- `WP_LOITER_RAD`: Default radius.
- `ALT_HOLD_RTL`: (Plane) Minimum altitude for RTL/Loiter safety.

KEY CODEBASE LOCATIONS

- **ArduCopter/mode_auto.cpp:2175**: `verify_loiter_to_alt` implementation.
- **ArduPlane/commands_logic.cpp:766**: `verify_loiter_to_alt` implementation.

NAV_LOITER_TURNS (18)

SUMMARY

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





The `NAV_LOITER_TURNS` command instructs the vehicle to fly to a specified location and orbit it for a specific number of full 360-degree rotations. This is commonly used to ensure a camera has multiple opportunities to capture a target or to allow a vehicle to shed altitude/airspeed before a landing approach.

STATUS

Supported (All Vehicles)

DIRECTIONALITY

- **RX (Receive):** The vehicle receives this command as part of a mission upload.

MISSION STORAGE (AP_MISSION)

ArduPilot uses complex bit-packing to store the number of turns and the radius within the limited EEPROM space.

- **Turns (Param 1):**
  - Stored in the low byte of `p1`.
  - Supports fractional turns (e.g., 1.5 turns) by multiplying by 256 and setting a "Fractional" bit in storage.
- **Radius (Param 3):**
  - Stored in the high byte of `p1`.
  - If the radius is > 255m, it is stored divided by 10, and a "Large Radius" bit is set.

EXECUTION (ENGINEER'S VIEW)

ARDUCOPTER IMPLEMENTATION

In Copter, this command utilizes the `AC_CircleNav` library via `ModeAuto::do_circle` (mode_auto.cpp).

1. **Transition:** The copter first flies to the *edge* of the circle (`CIRCLE_MOVE_TO_EDGE`).
2. **Tracking:** Once on the perimeter, it begins counting the cumulative angle traveled.
3. **Completion:** The command completes when:

$$\frac{|\text{Total Angle Traveled}|}{2\pi} \geq \text{Target Turns}$$

4. **Yaw:** The vehicle can be configured to face the center, face the direction of travel, or hold a fixed heading via the `CIRCLE_YAW_BEHAVE` parameter.

ARDUPLANE IMPLEMENTATION

In Plane, the logic resides in `Plane::verify_loiter_turns`.

1. **L1 Orbit:** The plane enters an L1 loiter at the specified radius.
2. **Cumulative Counting:** The autopilot integrates the change in bearing to the center to track rotations.
3. **Exit Strategy:** Unlike Copter, Plane implements a **Secondary Heading Goal**. Once the turns are completed, the plane does not immediately exit. It continues orbiting until it is pointing toward the *next* waypoint, ensuring a smooth tangential exit.

DATA FIELDS (MAVLINK)

- `param1` **(Turns):** Number of full orbits to complete.
- `param2` **:** Empty.
- `param3` **(Radius):** Orbit radius in meters. Positive = Clockwise, Negative = Counter-Clockwise.
- `param4` **(XTrack):**
  - **0:** Cross-track from the center of the loiter.
  - **1:** Cross-track from the tangent exit location (Plane only).
- `x` **(Latitude):** Center of the orbit.
- `y` **(Longitude):** Center of the orbit.
- `z` **(Altitude):** Target altitude.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





THEORY: ANGULAR MOMENTUM VS. GROUND TRACK

In the presence of wind, a constant airspeed orbit results in a "drifted" ground track. ArduPilot's L1 controller mathematically solves for the bank angle required to maintain a perfect circle over the ground.

- **Wind Speed (V_w):** Affects the ground speed (V_g).
- **Maximum Bank Angle:** Constrained by `ROLL_LIMIT_DEG`. If the wind is too high, the plane may be unable to maintain the radius and will "blow out" of the circle.

PRACTICAL USE CASES

1. **Aerial Cinematography:**
   - *Scenario:* A photographer wants a 360-degree reveal of a mountain peak.
   - *Action:* Use `NAV_LOITER_TURNS` with `Turns = 1` and `CIRCLE_YAW_BEHAVE = 1` (Face Center). The drone will orbit the peak while the camera stays locked on the target.
2. **Communications Relay:**
   - *Scenario:* A plane acts as a data link between a ground station and a distant rover.
   - *Action:* Loiter for 50 turns over the rover's location to provide persistent coverage.

KEY PARAMETERS

- `CIRCLE_RADIUS` : Default radius if mission radius is 0.
- `CIRCLE_RATE` : Maximum angular speed (deg/s) for the orbit.
- `WP_LOITER_RAD` : (Plane) Default loiter radius.

KEY CODEBASE LOCATIONS

- **ArduCopter/mode_auto.cpp:2256**: `verify_circle` implementation.
- **ArduPlane/commands_logic.cpp:740**: `verify_loiter_turns` implementation.

NAV_LOITER_UNLIM (17)

SUMMARY

The `NAV_LOITER_UNLIM` command instructs the vehicle to fly to a specified location and loiter (circle or hover) there indefinitely. This is a "blocking" command; the mission will not proceed to the next waypoint unless the pilot manually skips the item or changes the flight mode.

STATUS

Supported (All Vehicles)

DIRECTIONALITY

- **RX (Receive):** The vehicle receives this command as part of a mission upload.

MISSION STORAGE (AP_MISSION)

- **Packet Param 3 (Radius/Direction):**
  - Stored in the internal `p1` field.
  - **Radius:** The absolute value is stored as a 16-bit integer (meters).
  - **Direction:** The sign indicates direction (`-` = Counter-Clockwise, `+` = Clockwise).
- **Packet Param 4 (Yaw/XTrack):**
  - For Copter, `param4` represents the desired yaw (0 to 360 deg).
  - For Plane, `param4` > 0 enables "Tangent Exit" crosstracking.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





EXECUTION (ENGINEER'S VIEW)

ARDUCOPTER IMPLEMENTATION

In Copter, the command triggers `ModeAuto::do_loiter_unlimited` (mode_auto.cpp).

1. **Target Acquisition:** The location is passed to the `AC_WPNav` library.
2. **Behavior:**
   - The copter flies to the target Lat/Lon/Alt.
   - It enters `WP_NAV` loiter mode, holding position against wind using the GPS/INS solution.
   - **Heading:** The vehicle will face the next waypoint (if one exists) or hold the current heading, unless overridden by a `CONDITION_YAW` command.

ARDUPLANE IMPLEMENTATION

In Plane, the logic resides in `commands_logic.cpp` and `verify_loiter_unlim`.

1. **Loiter Radius:** The plane uses the `param3` radius. If `param3` is 0, it defaults to the global `WP_LOITER_RAD` parameter.
2. **Orbit Direction:**
   - **Positive Radius:** Clockwise.
   - **Negative Radius:** Counter-Clockwise.
3. **L1 Controller:** The plane maintains the orbit using L1 guidance, adjusting bank angle to compensate for wind drift.

DATA FIELDS (MAVLINK)

- `param1`: Empty.
- `param2`: Empty.
- `param3` **(Radius):** Radius in meters. Positive = Clockwise, Negative = Counter-Clockwise. If 0, uses default.
- `param4` **(Yaw/XTrack):** Desired Yaw (deg) [Copter] or XTrack location [Plane].
- `x` **(Latitude):** Target latitude.
- `y` **(Longitude):** Target longitude.
- `z` **(Altitude):** Target altitude.

THEORY: THE L1 LOITER LOGIC (PLANE)

For fixed-wing aircraft, loitering is not static; it is a dynamic energy management state.

- **Wind Compensation:** To maintain a perfect circle over the ground, the plane must vary its bank angle and ground speed.
- **Downwind:** Ground speed increases; bank angle must increase to increase centripetal force ($F_c = \frac{mv^2}{r}$).
- **Upwind:** Ground speed decreases; bank angle must decrease.

$$extBankAngle \propto \arctan\left(\frac{v_{ground}^2}{g \cdot r}\right)$$

PRACTICAL USE CASES

1. **Observation Point:**
   - *Scenario:* A surveillance drone needs to watch a specific intersection for an unknown duration.
   - *Action:* The operator inserts a `NAV_LOITER_UNLIM` command. The drone holds station until the operator commands it to "Resume Mission" or "Return to Launch".
2. **Holding Pattern:**
   - *Scenario:* Air traffic control (ATC) requires a delay before landing.
   - *Action:* The aircraft enters a holding pattern at a safe altitude.

KEY CODEBASE LOCATIONS

- **ArduCopter/mode_auto.cpp**: `do_loiter_unlimited` implementation.
- **libraries/AP_Mission/AP_Mission.cpp:1085**: Parameter packing for storage.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





NAV_PAYLOAD_PLACE (94)

SUMMARY

The `NAV_PAYLOAD_PLACE` command (often called "Payload Delivery") instructions the vehicle to descend vertically until it detects that a winch-slung or fixed payload has touched the ground. Once touchdown is detected, the vehicle releases the payload (if a gripper is attached) and climbs back to its original altitude.

STATUS

Supported (ArduCopter and QuadPlane)

DIRECTIONALITY

- **RX (Receive):** The vehicle receives this command as part of a mission upload.

MISSION STORAGE (AP_MISSION)

- **Packet Param 1 (Max Descent):** Stored in the internal `p1` field.
  - **Unit Conversion:** ArduPilot converts this value from meters to centimeters ($cm = m \cdot 100$) before storing it in EEPROM.
- **x, y, z:** The delivery coordinates.

EXECUTION (ENGINEER'S VIEW)

ARDUCOPTER IMPLEMENTATION

In Copter, the command triggers `ModeAuto::do_payload_place` (mode_auto.cpp).

1. **Approach:** The drone flies to the target Lat/Lon at the current mission altitude.
2. **Descent:** The drone begins a vertical descent at the speed defined by `PLDP_SPEED_DN`.
3. **Touchdown Detection:** The autopilot does not use simple altitude for detection. Instead, it monitors the **Thrust/Weight Ratio**.
   - As the payload touches the ground, the load on the motors decreases.
   - When the thrust required to maintain the descent rate drops below the threshold defined by `PLDP_THRESH`, the autopilot considers the payload "Placed".
4. **Release & Recovery:**
   - If a gripper or winch is configured, the autopilot triggers the release action.
   - The drone waits for `PLDP_DELAY` seconds to ensure a clean release.
   - The drone then climbs back to the initial altitude to continue the mission.

DATA FIELDS (MAVLINK)

- `param1` **(Max Descent):** The maximum distance (meters) the vehicle is allowed to descend. If the ground is not reached within this distance, the command is aborted to prevent a crash.
- `x` **(Latitude):** Target delivery location.
- `y` **(Longitude):** Target delivery location.
- `z` **(Altitude):** Current flight altitude.

THEORY: THE LOAD-SENSING ALGORITHM

Payload delivery in high-wind or turbulent environments is difficult because the vehicle's thrust is constantly fluctuating. ArduPilot uses a filtered approach to detect the "offloading" of weight:

$$\text{Placed} = \left(\frac{T_{current}}{T_{average_descent}} \right) < \text{PLDP_THRESH}$$

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





Where $T_{current}$ is the instantaneous thrust output and $T_{average_descent}$ is the learned thrust required to descend at a steady rate with the payload.

PRACTICAL USE CASES

1. **Automated Logistics:**
    - *Scenario:* Delivering a medical package to a remote village.
    - *Action:* The drone uses `NAV_PAYLOAD_PLACE`. It descends until the box touches the ground, opens the gripper, and climbs away without needing the pilot to see the ground or manage the descent manually.
2. **Scientific Sensor Deployment:**
    - *Scenario:* Placing a seismometer on a steep, inaccessible slope.
    - *Action:* The `PLDP_THRESH` logic ensures the sensor is firmly on the ground before release, even if the terrain altitude is not perfectly known.

KEY PARAMETERS

- `PLDP_THRESH`: The fraction of hover thrust that indicates a touchdown (Default 0.9).
- `PLDP_RNG_MAX`: Uses a downward-facing rangefinder to prevent "false touchdowns" if the drone is too high.
- `PLDP_SPEED_DN`: The vertical velocity during the delivery phase.

KEY CODEBASE LOCATIONS

- **ArduCopter/mode_auto.cpp:2024**: Copter delivery state machine.
- **ArduPlane/quadplane.cpp**: QuadPlane implementation.

NAV_RALLY_POINT (5100)

SUMMARY

The `NAV_RALLY_POINT` command defines an alternative "Safe Harbor" location for the vehicle. Unlike the Home position (which is usually where the drone took off), Rally Points are used as backup landing or loiter sites that the autopilot can choose from during a failsafe or RTL event based on proximity.

STATUS

Supported (All Vehicles)

DIRECTIONALITY

- **RX (Receive):** The vehicle receives these commands during a mission upload (specifically when using the MAVLink Rally Protocol).

MISSION STORAGE (AP_MISSION)

- **Packing:** Rally points are stored in a dedicated region of the EEPROM, separate from the main waypoint mission, allowing the drone to maintain its safety anchors even if the main mission is cleared.
- **Quantity:** ArduPilot typically supports up to 20-50 individual rally points depending on hardware.

EXECUTION (ENGINEER'S VIEW)

INTELLIGENT RTL

When an RTL (Return to Launch) is triggered:

1. **Comparison:** The autopilot calls `calc_best_rally_or_home_location()` (AP_Rally.cpp).

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





2. **Distance Heuristic:** It calculates the 2D distance to Home and *all* loaded Rally Points.
3. **Selection:** It chooses the **Geographically Closest** point.
4. **Action:** The vehicle flies to the chosen point and enters a loiter (Plane) or hover (Copter).

DATA FIELDS (MAVLINK)

- `x` **(Latitude):** Rally location.
- `y` **(Longitude):** Rally location.
- `z` **(Altitude):** Rally location.

THEORY: DISTRIBUTED RECOVERY

Standard recovery is centralized (Home). Rally Points enable **Distributed Recovery**.

- **The Hazard:** In long-distance missions (Linear Corridors), the vehicle may be 20km from Home when a battery failsafe occurs. It might not have enough energy to return 20km.
- **The Solution:** By placing Rally Points every 2km along the path, the autopilot ensures it never has to fly more than 1km to find a safe "Lifeboat."

PRACTICAL USE CASES

1. **Cross-Country FPV:**
   - *Scenario:* A plane is flying between two mountain peaks.
   - *Action:* Rally points are placed at known-clear meadows. If the radio link fails, the plane flies to the nearest meadow rather than trying to fly back over the peaks.
2. **Redundant Landing Pads:**
   - *Scenario:* A drone landing at a busy facility.
   - *Action:* Multiple rally points represent different landing pads. The drone chooses the one it can reach most efficiently.

KEY PARAMETERS

- `RALLY_LIMIT_KM` : The maximum distance the autopilot will look for a rally point.
- `RALLY_INCL_HOME` : Defines if Home should be considered in the proximity calculation.

KEY CODEBASE LOCATIONS

- **libraries/AP_Rally/AP_Rally.cpp**: Proximity comparison logic.
- **libraries/GCS_MAVLink/MissionItemProtocol_Rally.cpp**: Communication protocol for rally points.

NAV_RETURN_TO_LAUNCH (20)

SUMMARY

The `NAV_RETURN_TO_LAUNCH` (RTL) command instructs the vehicle to return to the home location or a designated rally point. This command is the primary safety mechanism for terminating a mission and returning the aircraft to the operator.

STATUS

Supported (All Vehicles)

DIRECTIONALITY

- **RX (Receive):** The vehicle receives this command as part of a mission upload or as a direct command (Failsafe trigger).

MISSION STORAGE (AP_MISSION)

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- **Packing:** This command contains no additional parameters (`param1` through `param7` are unused in missions). It simply serves as a state-change marker to trigger the vehicle's RTL mode.

EXECUTION (ENGINEER'S VIEW)

ARDUCOPTER IMPLEMENTATION

In Copter, the command triggers `ModeAuto::do_RTL()` (mode_auto.cpp).

1. **State Transition:** The vehicle switches from `AUTO` mode to `RTL` mode.
2. **Safety Logic:**
    - **Climb:** The vehicle first climbs to the `RTL_ALT` (or stays at current altitude if already higher, depending on `RTL_CLIMB_MIN`).
    - **Return:** It flies in a straight line towards Home.
    - **Descent:** Once over home, it hovers for the duration of `RTL_LOIT_TIME` before initiating the final land.

ARDUPLANE IMPLEMENTATION

In Plane, the logic resides in `Plane::verify_RTL` (commands_logic.cpp).

1. **Path Planning:** Plane identifies the "best" return location, which could be the Home point or the nearest Rally Point.
2. **Altitude Management:** The aircraft targets the `RTL_ALTITUDE` (Plane specific).
3. **Orbit:** Upon arrival, the plane enters a loiter (circle) at the return point. Unlike Copter, a standard Plane RTL does **not** automatically land unless the mission specifically contains a landing sequence or a landing failsafe is triggered.

DATA FIELDS (MAVLINK)

- `param1` to `param7`: Reserved / Unused.

THEORY: THE "CONE OF SAFETY"

A critical concept in RTL logic is the **Return Cone**.

- **Problem:** If a drone is very far away, returning at a low altitude might hit terrain. If it's very close, climbing to a high "Return Altitude" is a waste of energy.
- **Solution:** Advanced ArduPilot configurations use a "Safe Return Path" or Rally Points to ensure the vehicle always has a clear line of sight to a safe recovery zone.

PRACTICAL USE CASES

1. **Mission Completion:**
    - *Scenario:* A mapping mission has finished its last photo transect.
    - *Action:* The mission list ends with `NAV_RETURN_TO_LAUNCH` to bring the drone back to the takeoff area for recovery.
2. **Radio Failsafe:**
    - *Scenario:* The control link between the GCS and the drone is severed.
    - *Action:* The internal failsafe logic injects a `NAV_RETURN_TO_LAUNCH` command (equivalent) to recover the asset automatically.

KEY PARAMETERS

- `RTL_ALT`: (Copter) Altitude to return at.
- `RTL_ALTITUDE`: (Plane) Altitude to return at.
- `RTL_RADIUS`: (Plane) Radius of the loiter circle over home.

KEY CODEBASE LOCATIONS

- **ArduCopter/mode_auto.cpp:2184**: `verify_RTL` implementation.
- **ArduPlane/commands_logic.cpp:795**: `verify_RTL` implementation.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





NAV_SCRIPT_TIME (42702)

SUMMARY

The `NAV_SCRIPT_TIME` command is a powerful hybrid command that combines mission flow control with **Lua Scripting**. Unlike the "Do" version (`DO_SEND_SCRIPT_MESSAGE`), `NAV_SCRIPT_TIME` is a **Blocking Navigation Command**. The mission will not advance to the next item until the Lua script explicitly signals completion or a timeout is reached.

STATUS

Supported (All Vehicles with Lua Scripting enabled)

DIRECTIONALITY

- **RX (Receive):** The vehicle receives this command as part of a mission upload.

MISSION STORAGE (AP_MISSION)

- **Command ID (Param 1):** A user-defined ID the script uses to know what to do.
- **Timeout (Param 2):** Max duration in seconds before the mission auto-advances.
- **Arg 1 & 2 (Param 3-4):** Floats passed to the script.
- **Arg 3 & 4 (x, y):** Integers passed to the script.
- **Packing:** Stored in the `nav_script_time` content struct.

EXECUTION (ENGINEER'S VIEW)

HANDSHAKE LOGIC

The command creates a tight loop between the Mission Engine and the Lua Virtual Machine (AP_Scripting.cpp).

1. **Trigger:** The mission state machine sets the `nav_scripting.done` flag to `false` and records the start time.
2. **Script Detection:** The Lua script polls `mission:get_nav_script_time_id()` .
3. **Operation:** The script performs its task (e.g., complex pattern flying or sensor analysis).
4. **Completion Signal:** The script calls `mission:nav_script_time_done(id)` .
5. **Watchdog:** If the script fails to call the "done" method before `Param 2` seconds pass, the mission engine logs a warning and proceeds to the next item regardless.

DATA FIELDS (MAVLINK)

- `param1` **(ID):** Custom command ID.
- `param2` **(Timeout):** Seconds.
- `param3` **(Arg1):** Float.
- `param4` **(Arg2):** Float.
- `x` **(Arg3):** Int.
- `y` **(Arg4):** Int.

THEORY: THE CO-PROCESSOR MODEL

`NAV_SCRIPT_TIME` implements an **Asynchronous Co-Processor** model for mission logic.

- **Decoupling:** The "Heavy Lifting" (AI, Image Processing, Path Re-planning) happens in the Lua VM, which is sandboxed and memory-managed.
- **Deterministic Safety:** The main C++ flight code remains simple and stable. If the script crashes, the mission watchdog (Timeout) ensures the drone doesn't loiter until it crashes.

PRACTICAL USE CASES

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





1. **AI Search Pattern:**
    - *Scenario:* A drone finds a target but needs to perform a "spiral-out" search to find more.
    - *Action:* `NAV_SCRIPT_TIME (ID: 101, Timeout: 60s)`. The Lua script takes control of the position setpoints, flies the spiral, and signals "Done" once the search is complete.
2. **External Hardware Sync:**
    - *Scenario:* A drone landing on a robotic dock that takes 15 seconds to open.
    - *Action:* The script communicates with the dock, waits for the "Open" signal, and then releases the mission to continue to `NAV_LAND`.

KEY PARAMETERS

- `SCR_ENABLE`: Enables the scripting engine.
- `SCR_HEAP_SIZE`: Important for complex scripts to avoid OOM errors.

KEY CODEBASE LOCATIONS

- **libraries/AP_Scripting/AP_Scripting.cpp**: Implementation of the "Done" signal and binding.
- **ArduCopter/mode_auto.cpp:2311**: Verification logic.

NAV_SPLINE_WAYPOINT (82)

SUMMARY

The `NAV_SPLINE_WAYPOINT` command defines a 3D coordinate that the vehicle must pass through using a curved, "spline" trajectory. Unlike the standard linear waypoint, which produces sharp corners, the spline waypoint calculates a smooth path that considers the position of the previous and next waypoints to ensure the vehicle never has to come to a complete stop.

STATUS

Supported (ArduCopter Only)

DIRECTIONALITY

- **RX (Receive):** The vehicle receives this command as part of a mission upload.

MISSION STORAGE (AP_MISSION)

- **Packet Param 1 (Delay):** Stored in the internal `p1` field as a 16-bit integer (seconds).
- **Packet Param 2-4:** Empty.
- **x, y, z:** Target Latitude, Longitude, and Altitude.

EXECUTION (ENGINEER'S VIEW)

ARDUCOPTER IMPLEMENTATION

In Copter, the command triggers `ModeAuto::do_spline_wp` (mode_auto.cpp).

1. **Catmull-Rom Spline:** ArduPilot uses a modified **Catmull-Rom** spline algorithm. To calculate the curve for the current segment (Segment B), the algorithm requires four points:
    - P_0: The waypoint *before* the previous one.
    - P_1: The previous waypoint (start of the curve).
    - P_2: The current spline waypoint (target).
    - P_3: The *next* mission item (used to determine the exit velocity vector).
2. **Velocity Control:** The spline ensures that the velocity vector at P_2 is tangent to the curve, allowing for high-speed passes without the "jerk" associated with linear cornering.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





3. **Completion:** If `Delay (p1)` is 0, the waypoint is considered complete as soon as the vehicle passes through the coordinate, allowing it to transition immediately to the next leg. If `Delay > 0`, the vehicle will decelerate to a stop at the coordinate and wait before continuing.

DATA FIELDS (MAVLINK)

- `param1` **(Delay):** Hold time in seconds at the waypoint.
- `x` **(Latitude):** Target latitude.
- `y` **(Longitude):** Target longitude.
- `z` **(Altitude):** Target altitude.

THEORY: THE HERMITE SPLINE FORMULATION

The path between waypoints P_1 and P_2 is defined by a cubic polynomial in terms of time $t \in [0, 1]$:

$$\mathbf{P}(t) = (2t^3 - 3t^2 + 1)\mathbf{P}_1 + (t^3 - 2t^2 + t)\mathbf{T}_1 + (-2t^3 + 3t^2)\mathbf{P}_2 + (t^3 - t^2)\mathbf{T}_2$$

Where:

- $\mathbf{P}_1, \mathbf{P}_2$ are the positions.
- $\mathbf{T}_1, \mathbf{T}_2$ are the tangent vectors (velocities) at those points, derived from the surrounding waypoints.

PRACTICAL USE CASES

1. **Cinematic Fly-bys:**
   - *Scenario:* A drone needs to fly a smooth arc around a building.
   - *Action:* Use a sequence of `NAV_SPLINE_WAYPOINT` items. The resulting path will be a fluid curve, preventing the "robotic" stop-and-turn behavior of standard waypoints.
2. **Obstacle Avoidance in Speed Runs:**
   - *Scenario:* A racing drone needs to navigate a series of gates at maximum speed.
   - *Action:* Spline waypoints allow the drone to maintain its kinetic energy by "rounding" the corners optimally.

KEY PARAMETERS

- `WPNAV_SPEED`: Maximum horizontal speed between waypoints.
- `WPNAV_ACCEL`: Maximum acceleration used to define the spline's "tightness."

KEY CODEBASE LOCATIONS

- **ArduCopter/mode_auto.cpp:1614**: Spline waypoint initialization.
- **libraries/AC_WPNav/AC_WPNav.cpp**: The core spline math implementation.

NAV_TAKEOFF (22)

SUMMARY

The `NAV_TAKEOFF` command instructs the vehicle to lift off from the ground and climb to a specified altitude. This is typically the first command in an autonomous mission.

STATUS

Supported (All Vehicles)

DIRECTIONALITY

- **RX (Receive):** The vehicle receives this command as part of a mission upload or as a direct command (Guided Mode).

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





MISSION STORAGE (AP_MISSION)

- **Packet Param 1 (Pitch/Ignored):**
  - **Plane:** Minimum pitch angle during takeoff (degrees).
  - **Copter:** Ignored (usually 0).
- **Packet Param 7 (Altitude):**
  - Target altitude in meters.
  - *Note:* The frame of reference depends on the mission command frame (Relative, Absolute, or Terrain).

EXECUTION (ENGINEER'S VIEW)

ARDUCOPTER IMPLEMENTATION

In Copter, the command triggers `ModeAuto::do_takeoff`, which utilizes the `Mode::_TakeOff` class logic (takeoff.cpp).

1. **State Machine:**
   - The copter verifies it is armed and on the ground.
   - It spools up motors to the `THROTTLE_UNLIMITED` state.
2. **Climb Logic:**
   - The vertical position controller is fed a target altitude curve.
   - **Departure:** If the copter is still on the ground, it slews the throttle until it detects a climb (acceleration > threshold) or throttle saturation.
   - **No Navigation Zone:** If `WP_NAVALT_MIN` is set, the copter climbs straight up without horizontal navigation until it clears the "danger zone" (e.g., fence posts, people).
3. **Completion:** Takeoff is considered complete when the target altitude is reached (within a tolerance).

ARDUPLANE IMPLEMENTATION

In Plane, the logic resides in `Plane::do_takeoff` (commands_logic.cpp).

1. **Pitch Target:** The aircraft aims for the pitch angle specified in `param1`. If 0, it defaults to a safe minimum (typically 10-15 degrees depending on tuning).
2. **Heading Lock:** The plane locks its heading to the ground course projected towards the *next* waypoint. It uses the rudder (and potentially differential thrust) to fight crosswinds while on the ground.
3. **Throttle Management:** Throttle is clamped to maximum until the `TKOFF_THR_MIN_ACC` (acceleration) or `TKOFF_THR_MIN_SPD` (airspeed) thresholds are met, preventing premature motor spindown.

DATA FIELDS (MAVLINK)

- `param1` **(Pitch):** Minimum pitch angle in degrees (Plane only).
- `param2` **:** Empty.
- `param3` **:** Empty.
- `param4` **(Yaw):** Desired Yaw angle (deg). NaN to use current heading.
- `x` **(Latitude):** Target latitude (optional, used for initial heading alignment).
- `y` **(Longitude):** Target longitude (optional).
- `z` **(Altitude):** Target altitude in meters.

THEORY: THE "NO-NAV" ZONE

A critical concept in autonomous takeoff is the **"No-Nav" Zone** (controlled by `WP_NAVALT_MIN`).

- **Problem:** GPS positions drift. If a drone tries to navigate horizontally while still touching the ground, the landing gear might snag on grass or uneven terrain, causing a tip-over (Dynamic Rollover).
- **Solution:** The autopilot suppresses all horizontal position corrections (Roll/Pitch for Copter) until the vehicle has climbed to a safe altitude (e.g., 2 meters). It effectively shoots straight up like a rocket before engaging the navigation controller.

PRACTICAL USE CASES

1. **Hand Launch (Plane):**

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- *Scenario:* Launching a fixed-wing surveyor without a runway.
- *Action:* The pilot shakes the plane to arm it (Shake-to-Wake). The `NAV_TAKEOFF` command manages the throttle delay and initial climb out, ensuring the prop doesn't spin up until the plane is thrown.

2. **Confined Area (Copter):**
   - *Scenario:* Taking off from a deep urban canyon or forest clearing.
   - *Action:* `WP_NAVALT_MIN` is set to 15m. The `NAV_TAKEOFF` command ensures the drone rises vertically above the treeline before attempting to fly to the first waypoint.

KEY CODEBASE LOCATIONS

- **ArduCopter/takeoff.cpp**: Copter takeoff state machine.
- **ArduPlane/commands_logic.cpp:372**: Plane takeoff logic.
- **libraries/AP_Mission/AP_Mission.cpp**: Command unpacking.

NAV_VTOL_LAND (85)

SUMMARY

The `NAV_VTOL_LAND` command instructs a hybrid aircraft (QuadPlane) to transition from fixed-wing cruise to multicopter mode and perform a vertical descent to a precise location. This allows long-range fixed-wing assets to land in confined spaces without a runway.

STATUS

Supported (ArduPlane / QuadPlane Only)

DIRECTIONALITY

- **RX (Receive):** The vehicle receives this command as part of a mission upload or as a RTL/Failsafe action.

MISSION STORAGE (AP_MISSION)

- **Packet Param 1 (Options):** Stored in the internal `p1` field. This bitmask allows for advanced landing behaviors, such as "Descending Loiter" before the final drop.

EXECUTION (ENGINEER'S VIEW)

ARDUPLANE (QUADPLANE) IMPLEMENTATION

In QuadPlane, the command triggers a multi-stage state machine (quadplane.cpp).

1. **Back-Transition:**
   - The aircraft approaches the target waypoint in fixed-wing mode.
   - It cuts the forward motor and engages the VTOL motors (multicopter) as it decelerates below the stall speed.
2. **Approach:**
   - **Spiral Descent:** If configured, the aircraft loiters over the landing point while descending.
   - **Wind Alignment:** The autopilot attempts to align the vehicle's heading into the wind to minimize lateral drift during the hover phase.
3. **Final Descent:**
   - The vehicle uses the multicopter vertical controller to descend at `Q_LAND_SPEED`.
   - It uses the `landing_detect` logic to verify touchdown before disarming.

DATA FIELDS (MAVLINK)

- `param1` **(Options):** Bitmask for landing behavior.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- `param2`: Empty.
- `param3`: Empty.
- `param4` **(Yaw):** Desired heading for landing.
- `x` **(Latitude):** Target landing location.
- `y` **(Longitude):** Target landing location.
- `z` **(Altitude):** Final altitude (typically 0).

THEORY: THE Q-APPROACH

The "Q-Approach" is a sophisticated landing method designed for high-performance VTOLs.

- **Aerodynamic Transition:** Unlike multicopters, QuadPlanes have significant wing lift. If they descend too quickly while moving forward, the wing can generate asymmetric lift (if rolling), leading to instability.
- **Airbrake Effect:** ArduPilot can use the multicopter motors as "airbrakes" by spinning them up at low power while still in forward flight, creating high drag to slow the aircraft down for the final hover.

PRACTICAL USE CASES

1. **Precision Recovery:**
   - *Scenario:* A 3-meter wingspan drone returning to a 10m x 10m clearing.
   - *Action:* `NAV_VTOL_LAND` ensures the aircraft flies the long-distance return at high efficiency, then converts to a hovering drone for a pin-point landing.
2. **Autonomous Charging Dock:**
   - *Scenario:* A drone landing on a robotic charging platform.
   - *Action:* Use `NAV_VTOL_LAND` in conjunction with a Precision Landing sensor (IR/Camera) to land within centimeters of the charging pins.

KEY PARAMETERS

- `Q_TRANS_DECEL`: Deceleration rate (m/s/s) during back-transition.
- `Q_LAND_FINAL_ALT`: Altitude above ground at which the aircraft enters its final, slower descent phase.
- `Q_FWD_THR_USE`: Allows the pusher motor to assist with station-keeping in high winds during the vertical descent.

KEY CODEBASE LOCATIONS

- **ArduPlane/quadplane.cpp**: Main VTOL landing handler.
- **libraries/AP_Landing/AP_Landing.cpp**: Shared landing library.

NAV_VTOL_TAKEOFF (84)

SUMMARY

The `NAV_VTOL_TAKEOFF` command instructs a QuadPlane (or Tilt-Rotor) to lift off vertically using its multicopter motors and then transition to forward, fixed-wing flight. This command is distinct from a standard `NAV_TAKEOFF`, as it explicitly manages the high-energy state transition between hovering and cruising.

STATUS

Supported (ArduPlane / QuadPlane Only)

DIRECTIONALITY

- **RX (Receive):** The vehicle receives this command as part of a mission upload.

MISSION STORAGE (AP_MISSION)

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- **Packing:** ArduPilot stores the command in the standard location structure. The altitude is extracted from the `z` field of the MAVLink packet.

EXECUTION (ENGINEER'S VIEW)

ARDUPLANE (QUADPLANE) IMPLEMENTATION

In QuadPlane, the command triggers `QuadPlane::do_vtol_takeoff` (quadplane.cpp).

1. **Vertical Phase:**
    - The vehicle uses the vertical position controller (`pos_control`) to rise to the target altitude.
    - **Time-to-Altitude Estimate:** The autopilot calculates the expected duration of the climb using the following kinematic model:

$$t_{accel} = \frac{V_{max} - V_z}{a}$$

$$d_{accel} = V_z \cdot t_{accel} + \frac{1}{2} a \cdot t_{accel}^2$$

$$t_{total} = \max(t_{accel}, 0) + \max\left(\frac{d_{total} - d_{accel}}{V_{max}}, 0\right)$$

2. **Transition Phase:**
    - Once the vertical target is reached, the autopilot begins the **Transition to Fixed-Wing**.
    - It engages the forward motor (pusher) while maintaining multicopter attitude control.
    - Once the airspeed reaches the `ARSPD_FBW_MIN`, the multicopter motors are phased out, and the vehicle becomes a standard airplane.

DATA FIELDS (MAVLINK)

- `param1`: Empty.
- `param2`: Empty.
- `param3`: Empty.
- `param4` **(Yaw):** Target heading (degrees).
- `x` **(Latitude):** Target latitude.
- `y` **(Longitude):** Target longitude.
- `z` **(Altitude):** Target altitude (meters).

THEORY: THE "V-ALPHA" TRANSITION

The transition from vertical to horizontal flight is the most dangerous phase of flight for a VTOL aircraft.

- **Wing Loading:** During transition, lift is shared between the vertical rotors and the wing.
- **Pitch Control:** As forward speed increases, the elevators gain authority, but the multicopter pitch controller is still active. ArduPilot uses a "Blend" logic to ensure smooth control handover.
- **Stall Risk:** If the transition is too slow, the battery may deplete. If too fast, the aircraft may pitch up violently due to the increased airspeed over the wing.

PRACTICAL USE CASES

1. **Runway-Free Long Endurance:**
    - *Scenario:* A fixed-wing plane needs to map 100km of pipeline but must take off from a small jungle clearing.
    - *Action:* Use `NAV_VTOL_TAKEOFF` to rise 30m above the trees before accelerating into high-efficiency fixed-wing mode.
2. **Shipboard Launch:**
    - *Scenario:* Launching from a moving vessel.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- *Action:* VTOL takeoff allows the aircraft to clear the deck and masts vertically before engaging the pusher motor to match the ship's speed and move away safely.

KEY PARAMETERS

- `Q_TAKEOFF_ALT` : Default altitude if mission altitude is 0.
- `Q_TRANSITION_MS` : Duration of the forward motor ramp-up.
- `Q_TILT_MAX` : (Tilt-Rotor only) Maximum tilt angle during transition.

KEY CODEBASE LOCATIONS

- **ArduPlane/quadplane.cpp:3298**: VTOL takeoff initialization.
- **ArduPlane/quadplane.cpp:3390**: Transition logic handler.

NAV_WAYPOINT (16)

SUMMARY

The `NAV_WAYPOINT` command is the fundamental building block of autonomous missions. It defines a 3D coordinate (Latitude, Longitude, Altitude) that the vehicle must navigate to. Depending on the vehicle type and parameters, the vehicle may either stop at the waypoint or transition smoothly through it towards the next destination.

STATUS

Supported (All Vehicles)

DIRECTIONALITY

- **RX (Receive):** The vehicle receives this command as part of a mission upload (`MISSION_ITEM` or `MISSION_ITEM_INT`).

MISSION STORAGE (AP_MISSION)

When a `NAV_WAYPOINT` is stored in ArduPilot's EEPROM, the parameters are packed for efficiency:

- **Copter:** `param1` (Delay) is stored in the `p1` field.
- **Plane:** `param2` (Acceptance Radius) and `param3` (Pass-by Distance) are packed into the `p1` field.

EXECUTION (ENGINEER'S VIEW)

ARDUCOPTER IMPLEMENTATION

In Copter, the command is triggered in `ModeAuto::do_nav_wp` (mode_auto.cpp).

1. **Path Generation:** The location is passed to the `AC_WPNav` library. ArduCopter uses **S-Curve** trajectory generation to ensure smooth acceleration and deceleration.
2. **Cornering Logic:**
   - If **Delay (p1)** is 0, the flight controller looks ahead to the *next* waypoint. It plans a path that rounds the corner, maintaining velocity.
   - If **Delay (p1)** is greater than 0, the vehicle is forced to stop exactly at the coordinate and wait for the specified time before proceeding.
3. **Yaw Control:** The vehicle will typically face the next waypoint unless a `CONDITION_YAW` or `ROI` command has overridden the heading logic.

ARDUPLANE IMPLEMENTATION

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





In Plane, the logic resides in `commands_logic.cpp`.

1. **L1 Controller:** Plane uses an L1 guidance algorithm which creates a lateral acceleration command to guide the aircraft onto the track between waypoints.
2. **Acceptance Radius (p2):** Defines a cylinder around the waypoint. As soon as the aircraft enters this cylinder, the waypoint is considered "complete." If set to 0, the global `WP_RADIUS` parameter is used.
3. **Pass-By (p3):** Allows the plane to start the turn early. If the aircraft comes within `p3` meters of the waypoint, it transitions to the next leg.

DATA FIELDS (MAVLINK)

- `param1` **(Delay):** Time to loiter at the waypoint in seconds (Copter only).
- `param2` **(Acceptance Radius):** Distance in meters at which the waypoint is considered reached (Plane only).
- `param3` **(Pass-by Distance):** For planes, the distance at which the aircraft should begin its turn towards the next waypoint.
- `param4` **(Yaw):** Desired yaw angle (Copter only).
- `x` **(Latitude):** Target latitude.
- `y` **(Longitude):** Target longitude.
- `z` **(Altitude):** Target altitude.

PRACTICAL USE CASES

1. **Survey Grid:**
   - *Scenario:* Mapping a field with a camera.
   - *Action:* A series of `NAV_WAYPOINT` commands are used to define the "lawnmower" pattern. `Delay` is set to 0 for continuous flight.
2. **Inspection Stop:**
   - *Scenario:* Inspecting a power line insulator.
   - *Action:* The drone flies to a `NAV_WAYPOINT` positioned near the insulator with a `Delay` of 5 seconds, allowing the camera to capture high-quality images while stationary.

KEY CODEBASE LOCATIONS

- **ArduCopter/mode_auto.cpp:1547**: `do_nav_wp` implementation for Copter.
- **ArduPlane/commands_logic.cpp**: Plane mission command logic.
- **libraries/AP_Mission/AP_Mission.cpp:1065**: Mission storage packing for waypoints.

SET_CAMERA_FOCUS (532)

SUMMARY

The `SET_CAMERA_FOCUS` command manages the focus state of an onboard camera. It supports triggering Auto-Focus (AF) routines or setting manual focus levels, ensuring sharp imagery for automated inspections where the distance to the subject varies.

STATUS

Supported (All Vehicles with AP_Camera enabled)

DIRECTIONALITY

- **RX (Receive):** The vehicle receives this command as part of a mission upload or as a direct command from the GCS.

MISSION STORAGE (AP_MISSION)

- **Focus Type (Param 1):**
  - 0: Auto.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





    - 1: Continuous (Manual rate).
    - 2: Range (Manual percentage 0-100).
- **Focus Value (Param 2):** The value corresponding to the manual types.
- **Packing:** Stored in the `set_camera_focus` content struct.

EXECUTION (ENGINEER'S VIEW)

FOCUS MANAGEMENT

The command is processed by the `AP_Camera` library (AP_Camera.cpp).

1. **Auto-Focus Trigger:** If `Auto (0)` is selected, ArduPilot commands the camera backend to perform a "One-shot AF" or "Continuous AF" depending on the camera's internal capabilities.
2. **Manual Overrides:** If `Range (2)` is selected, ArduPilot maps the input to the lens's focal range ($F_{min} \rightarrow F_{max}$).
3. **Lens Compatibility:** This command requires a camera with an electronically controlled focus motor (e.g., Sony block cameras, Gremsy-integrated sensors). Fixed-focus mapping cameras (like those used for photogrammetry) will ignore this command.

DATA FIELDS (MAVLINK)

- `param1` **(Type):** 0:Auto, 1:Continuous, 2:Range, 3:Meters (Rarely supported).
- `param2` **(Value):** Rate or Percentage.
- `param3` to `param7`: Unused.

THEORY: THE CIRCLE OF CONFUSION

Focusing is the process of minimizing the **Circle of Confusion** (c) on the image sensor.

- **Depth of Field (DOF):** At high zoom levels, the DOF becomes extremely shallow.
- **Vibration:** Out-of-focus images cannot be corrected in post-processing. `SET_CAMERA_FOCUS` is used to "Lock" focus before a high-vibration segment to prevent the camera's internal AF from "hunting" due to motion blur.

PRACTICAL USE CASES

1. **Macro Inspection:**
    - *Scenario:* A drone is inspecting a weld on a bridge at a distance of 1.5 meters.
    - *Action:* `SET_CAMERA_FOCUS (Auto)` followed by `IMAGE_START_CAPTURE`.
2. **Infinity Lock:**
    - *Scenario:* Mapping from 100m altitude.
    - *Action:* Mission starts with `SET_CAMERA_FOCUS (Range: 100\%)` to lock the lens at infinity, preventing hunting during flight.

KEY PARAMETERS

- `CAM_TYPE`: Selection of camera hardware.

KEY CODEBASE LOCATIONS

- **libraries/AP_Camera/AP_Camera.cpp:307**: Focus control implementation.

SET_CAMERA_SOURCE (534)

SUMMARY

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

The `SET_CAMERA_SOURCE` command allows the mission to dynamically switch between different sensors (lenses) on a multi-sensor camera system. This is common on modern dual-sensor payloads that feature both a visual (RGB) and a thermal (EO/IR) sensor.

STATUS

Supported (All Vehicles with AP_Camera enabled)

DIRECTIONALITY

- **RX (Receive):** The vehicle receives this command as part of a mission upload or as a direct command.

MISSION STORAGE (AP_MISSION)

- **Instance (Param 1):** The camera ID (1-6). 0 for "All."
- **Primary Source (Param 2):** Selection for the primary stream (e.g., RGB).
- **Secondary Source (Param 3):** Selection for the secondary stream (e.g., IR).
- **Packing:** Stored in the `set_camera_source` content struct.

EXECUTION (ENGINEER'S VIEW)

MULTICAM LOGIC

The command is handled by `AP_Camera::set_camera_source` (AP_Camera.cpp).

1. **Index Translation:** ArduPilot maps "Camera 1" to the first available hardware driver.
2. **Source Selection:** The autopilot sends a command to the camera backend (MAVLink or specialized driver) to reconfigure the video stream or the image capture target.
3. **Use Cases:**
    - Switching from Wide-Angle to Telephoto on a triple-lens system.
    - Switching from Visual to Night-Vision (Thermal) for a search leg.

DATA FIELDS (MAVLINK)

- `param1` **(Instance):** Camera instance number.
- `param2` **(Primary):** 0:No change, 1:RGB, 2:IR, 3:NDVI.
- `param3` **(Secondary):** Same as above.
- `param4` to `param7`: Unused.

THEORY: SENSOR FUSION

In modern autonomous flight, the "Source" is not just a video feed; it defines the vehicle's **Primary Intelligence Input**.

- **RGB:** Used for photogrammetry and human monitoring.
- **IR:** Used for finding heat signatures (SAR) or hot-spots (Industrial Inspection).
- **The Switch:** `SET_CAMERA_SOURCE` allows a single drone to perform two missions in one flight by reconfiguring its brain for the task at hand.

PRACTICAL USE CASES

1. **Thermal Inspection:**
    - *Scenario:* Inspecting high-voltage lines.
    - *Action:* Mission flies to the pole. `SET_CAMERA_SOURCE (IR)` is used to check for heat, then `SET_CAMERA_SOURCE (RGB)` is used to take high-res visual confirmation of the hardware.
2. **Multi-Spectrally Survey:**
    - *Scenario:* Mapping a field for crop health.
    - *Action:* Mission alternates sources to capture both RGB and NDVI (Normalized Difference Vegetation Index) data.

KEY PARAMETERS

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- `CAM_TYPE`: Selection of camera hardware.

KEY CODEBASE LOCATIONS

- **libraries/AP_Camera/AP_Camera.cpp:329**: Logic for multi-sensor switching.

SET_CAMERA_ZOOM (531)

SUMMARY

The `SET_CAMERA_ZOOM` command controls the focal length of an onboard camera during a mission. It allows for both incremental (rate-based) and absolute (percentage-based) zoom adjustments, enabling detailed inspections or wide-area reconnaissance from the same mission script.

STATUS

Supported (All Vehicles with AP_Camera enabled)

DIRECTIONALITY

- **RX (Receive):** The vehicle receives this command as part of a mission upload or as a direct command from the GCS.

MISSION STORAGE (AP_MISSION)

- **Zoom Type (Param 1):**
  - 0: Step (Not supported in storage).
  - 1: Continuous (Rate-based).
  - 2: Range (Percentage-based 0-100).
- **Zoom Value (Param 2):** The value corresponding to the requested type.
- **Packing:** Stored in the `set_camera_zoom` content struct.

EXECUTION (ENGINEER'S VIEW)

CAMERA DRIVER INTERACTION

The command is routed through the `AP_Camera` library (AP_Camera.cpp).

1. **Rate-Based Zoom:** If `Continuous (1)` is selected, the autopilot commands the lens motor to move at a specific speed. This is typically used with gimbal-integrated cameras where the operator wants to zoom in "until clear."
2. **Percentage Zoom:** If `Range (2)` is selected, the autopilot maps the 0.0-100.0 input to the camera's internal zoom range ($Z_{min} \rightarrow Z_{max}$).
3. **Backend Support:** This command works with MAVLink-enabled cameras (like SToRM32 or Tarot), DroneCAN cameras, and specialized drivers (like the Gremsy or Sony cameras). If the camera backend does not support zoom, the command is ignored.

DATA FIELDS (MAVLINK)

- `param1` **(Type):** 0:Step, 1:Continuous, 2:Range.
- `param2` **(Value):** Rate (speed) or Percentage.
- `param3` to `param7`: Unused.

THEORY: MAGNIFICATION VS. RESOLUTION

Zooming in does not increase the sensor's physical resolution; it changes the **Field of View (FOV)**.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- **GSD Impact:** Doubling the zoom ($2\times$) effectively halves the Ground Sample Distance (GSD), assuming the altitude remains constant.
- **Vibration Sensitivity:** As FOV decreases (Zoom increases), the image becomes exponentially more sensitive to high-frequency vibration ($Z_{sens} \propto \frac{1}{\text{FOV}}$). This often requires the gimbal's PID gains to be adjusted dynamically (handled automatically by advanced backends).

PRACTICAL USE CASES

1. **In-Mission Target Detail:**
   - *Scenario:* A drone is surveying a fence. It detects a breach at a waypoint.
   - *Action:* `SET_CAMERA_ZOOM (Type: 2, Value: 80\%)`. The drone zooms in to capture high-detail evidence before continuing the survey.
2. **Dynamic Reconnaissance:**
   - *Scenario:* A search-and-rescue plane is looking for a boat.
   - *Action:* The plane orbits at $1\times$ zoom to cover the area. Once a suspect object is found, it uses a script to trigger `SET_CAMERA_ZOOM` to $10\times$ for confirmation.

KEY PARAMETERS

- `CAM_TYPE`: Selection of camera hardware.
- `MNT1_TYPE`: Often required as gimbals handle the physical lens motors.

KEY CODEBASE LOCATIONS

- **libraries/AP_Camera/AP_Camera.cpp:297**: Mission command intake.
- **libraries/AP_Camera/AP_Camera_MAVLinkCamV2.cpp**: Translation to MAVLink camera protocol.

VIDEO_START_CAPTURE (2500) / VIDEO_STOP_CAPTURE (2501)

SUMMARY

The `VIDEO_START_CAPTURE` and `VIDEO_STOP_CAPTURE` commands control the recording state of onboard video sensors. They allow a mission to automatically record only the segments of flight that are relevant to the objective, saving storage space and power.

STATUS

Supported (All Vehicles with AP_Camera enabled)

DIRECTIONALITY

- **RX (Receive):** The vehicle receives this command as part of a mission upload or as a direct command.

MISSION STORAGE (AP_MISSION)

- **Stream ID (Param 1):** The target video stream or camera instance.
- **Packing:** Stored in the `video_start_capture` content struct.

EXECUTION (ENGINEER'S VIEW)

RECORDING LOGIC

Execution is handled by `AP_Camera::record_video` (AP_Camera.cpp).

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





1. **Direct Control:** If the camera is a simple GoPro-style camera triggered via a PWM signal, ArduPilot moves the servo/pin to the "Record" position.
2. **MAVLink Control:** For high-end cameras (e.g., DJI, Sony), ArduPilot sends the standard MAVLink command to the camera's component ID.
3. **Conflict Management:** If a "Start" is commanded while the camera is already recording, ArduPilot typically ignores the second command to prevent file corruption.

DATA FIELDS (MAVLINK)

- `param1` **(ID):** Video stream/camera ID.
- `param2` **(Freq):** Record frequency (FPS). Typically ignored (set in camera menu).
- `param3` to `param7`: Unused.

THEORY: BANDWIDTH VS. DETAIL

Video recording is a continuous energy drain on both the battery and the onboard processor.

- **The Overload:** Modern 4K/60fps video generates significant electromagnetic interference (EMI).
- **Autonomous Recording:** By using mission commands to start recording *after* the takeoff and *stop* before the landing, the pilot minimizes the noise exposure during the most critical flight phases (where GPS/Compass health is most vital).

PRACTICAL USE CASES

1. **Evidence Collection:**
   - *Scenario:* A security drone patrolling a site.
   - *Action:* `WAYPOINT (Start of Patrol)` → `VIDEO_START_CAPTURE`. The drone records the entire patrol leg and stops once it begins its return-to-launch.
2. **Cinematic Reveal:**
   - *Scenario:* A drone flying a specific path for a film shot.
   - *Action:* `VIDEO_START_CAPTURE` is triggered precisely at the start of the camera movement.

KEY PARAMETERS

- `CAM_TYPE`: Selection of camera hardware.

KEY CODEBASE LOCATIONS

- **libraries/AP_Camera/AP_Camera.cpp:411**: Implementation of video recording logic.
- **libraries/AP_Camera/AP_Camera_Gopro.cpp**: Specialized GoPro control.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





CHAPTER 19: MISSION SCENARIOS

MISSION SCENARIO: PHOTOGRAMMETRY MAPPING

OBJECTIVE

Create a high-resolution 2D orthomosaic or 3D model of a 5-hectare construction site. The drone must maintain a consistent ground resolution (GSD) and ensure 80\% frontal overlap between photos.

VEHICLE CONFIGURATION

- **Type:** Multirotor (Quad/Hex)
- **Payload:** High-resolution RGB Camera (e.g., Sony A7R) pointing Nadir (straight down).
- **Speed:** 8 m/s (optimized for shutter speed vs. motion blur).
- **Altitude:** 60m AGL (Above Ground Level).

KEY CONCEPTS

- **Distance Triggering:** Using `DO_SET_CAM_TRIGG_DIST` instead of time intervals guarantees overlap even if the drone slows down against the wind.
- **Overshoot:** The drone flies past the edge of the survey area before turning. This ensures the vehicle is stable and moving at constant velocity before entering the next capture leg.

MISSION PLAN

SEQ	COMMAND	PARAMETERS	DESCRIPTION
1	`NAV_TAKEOFF`	Alt: 30m	Climb safely above obstacles before navigating.
2	`DO_CHANGE_SPEED`	Speed: 8 m/s	Set survey cruise speed.
3	`DO_MOUNT_CONTROL`	Pitch: -90, Roll: 0, Yaw: 0	Point camera straight down (Nadir).
4	`NAV_WAYPOINT`	Lat/Lon: [Start of Grid]	Fly to the first survey point.
5	`DO_SET_CAM_TRIGG_DIST`	Dist: 18.5m	Start capturing a photo every 18.5 meters (calculated for 80\% overlap).
6	`NAV_WAYPOINT`	Lat/Lon: [End of Leg 1]	Fly the first transect.
7	`NAV_WAYPOINT`	Lat/Lon: [Turn Point]	Turn 90 degrees to next lane.
8	`NAV_WAYPOINT`	Lat/Lon: [Start of Leg 2]	Align for return pass.
9	`NAV_WAYPOINT`	Lat/Lon: [End of Leg 2]	Fly the second transect.
...	...	...	*Repeat pattern...*
50	`DO_SET_CAM_TRIGG_DIST`	Dist: 0	**Stop capturing** photos to save storage/processing time.
51	`DO_MOUNT_CONTROL`	Pitch: 0, Roll: 0, Yaw: 0	Retract/Level gimbal for landing.
52	`NAV_RETURN_TO_LAUNCH`	-	Return home and auto-land.

PRO-TIPS

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- **Terrain Following:** If the site is hilly, use `TERRAIN` frame for all waypoints to ensure the GSD (pixel size) remains constant across the entire map.
- **Shutter Priority:** Set your camera to Shutter Priority (Tv) mode with at least 1/1000s shutter speed to eliminate motion blur at 8 m/s.

MISSION SCENARIO: PIPELINE INSPECTION

OBJECTIVE

Inspect a 5km segment of oil pipeline for leaks or corrosion. The drone must fly a linear path but pause and capture high-zoom images at specific valve stations.

VEHICLE CONFIGURATION

- **Type:** VTOL QuadPlane (for range and hover capability).
- **Payload:** Dual-sensor (Thermal/RGB) gimbal with 10x Optical Zoom.
- **Speed:** 15 m/s (Cruise), 0 m/s (Inspection).
- **Altitude:** 40m AGL.

KEY CONCEPTS

- **Hybrid Flight:** Using the efficiency of fixed-wing flight for the long transit, but transitioning to multicopter mode for detailed inspection.
- **Region of Interest (ROI):** Using `DO_SET_ROI` ensures the camera stays locked on the valve station while the drone maneuvers around it.
- **Zoom Control:** Using `SET_CAMERA_ZOOM` to get detailed imagery without needing to fly dangerously close to the infrastructure.

MISSION PLAN

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





SEQ	COMMAND	PARAMETERS	DESCRIPTION
1	`NAV_VTOL_TAKEOFF`	Alt: 40m	Vertical takeoff to clear fencing.
2	`NAV_WAYPOINT`	Lat/Lon: [Start of Pipe]	Transition to fixed-wing flight and begin patrol.
3	`NAV_WAYPOINT`	Lat/Lon: [Valve Station 1 Approach]	Fly towards the first target.
4	`DO_VTOL_TRANSITION`	Mode: 3 (Multicopter)	Transition to hover mode before reaching the valve.
5	`DO_SET_ROI`	Lat/Lon: [Valve Station 1]	Lock gimbal onto the valve.
6	`NAV_WAYPOINT`	Lat/Lon: [Inspection Point A]	Move to first viewing angle.
7	`SET_CAMERA_ZOOM`	Type: Continuous, Val: 5x	Zoom in for detail.
8	`IMAGE_START_CAPTURE`	Interval: 0, Total: 1	Take a high-res photo.
9	`NAV_WAYPOINT`	Lat/Lon: [Inspection Point B]	Move to second viewing angle (creating parallax).
10	`IMAGE_START_CAPTURE`	Interval: 0, Total: 1	Take second photo.
11	`SET_CAMERA_ZOOM`	Type: Continuous, Val: 1x	Zoom out.
12	`DO_SET_ROI`	Mode: 0 (Cancel)	Release gimbal lock.
13	`DO_VTOL_TRANSITION`	Mode: 4 (Fixed Wing)	Accelerate back to cruise speed.
...	...	...	*Repeat for next station...*
50	`NAV_VTOL_LAND`	Lat/Lon: [Landing Pad]	Vertical landing at recovery site.

PRO-TIPS

- **Thermal First:** Use `SET_CAMERA_SOURCE` to switch to the Thermal view during the initial approach to check for leaks (temperature deltas), then switch to RGB for the zoomed inspection.
- **Wind Alignment:** When hovering in VTOL mode, ArduPilot automatically weathervanes into the wind to save power, but the ROI command keeps the camera focused on the target regardless of the vehicle's heading.

MISSION SCENARIO: AUTONOMOUS CARGO DELIVERY

OBJECTIVE

Deliver a 2kg medical package to a remote clinic. The landing zone is small and surrounded by trees, requiring a precision winch delivery instead of a landing.

VEHICLE CONFIGURATION

- **Type:** Hexacopter (Heavy Lift).
- **Payload:** Winch system with an electro-magnetic gripper.
- **Failsafes:** Battery Failsafe set to "Land" (not RTL) due to distance.

KEY CONCEPTS

- **Precision Placement:** `NAV_PAYLOAD_PLACE` automates the descent and touchdown detection, which is safer than manual piloting when depth perception is limited.
- **Mission Re-Entry:** Using `DO_RETURN_PATH_START` ensures that if the drone aborts the delivery, it flies a safe exit vector away from the trees before returning home.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY

MISSION PLAN

SEQ	COMMAND	PARAMETERS	DESCRIPTION
1	`NAV_TAKEOFF`	Alt: 50m	Climb to cruise altitude.
2	`DO_GRIPPER`	Action: Grab	Ensure package is secured (redundancy check).
3	`NAV_WAYPOINT`	Lat/Lon: [Clinic Approach]	Transit to delivery site.
4	`DO_LAND_START`	-	Marker: Begin approach sequence.
5	`DO_WINCH`	Action: Length, Val: 10m	Lower package 10m below drone while still high above trees.
6	`NAV_LOITER_TIME`	Time: 5s	Pause to let the load stabilize (dampen pendulum swing).
7	`NAV_PAYLOAD_PLACE`	Max Descent: 20m	Descend slowly until winch load decreases (touchdown).
8	`DO_GRIPPER`	Action: Release	Drop the package.
9	`DO_WINCH`	Action: Length, Val: 0m	Retract cable immediately to avoid snagging.
10	`NAV_WAYPOINT`	Alt: 50m	Climb vertically back to cruise height.
11	`NAV_RETURN_TO_LAUNCH`	-	Return home.

PRO-TIPS

- **Slung Load Stability:** Use `LOITER_UNLIM` combined with `NAV_SCRIPT_TIME` to run a Lua script that detects load oscillation. Only proceed to the drop once the swing amplitude is < 5 degrees.
- **Emergency Release:** Map a "Emergency Winch Relax" switch on your RC transmitter. If the cable snags on a tree during the climb out, you must instantly unspool the winch to save the aircraft.

MISSION SCENARIO: PRECISION CROP SPRAYING

OBJECTIVE

Spray a rectangular cornfield with liquid fertilizer. The drone must maintain a consistent flow rate per hectare and automatically return to the refill station when the tank is empty.

VEHICLE CONFIGURATION

- **Type:** Octocopter (Agri-class).
- **Payload:** Pump and Boom system controlled by `AC_Sprayer`.
- **Sensors:** Radar Altimeter for terrain following.

KEY CONCEPTS

- **Flow Compensation:** `DO_SPRAYER` automatically adjusts pump PWM based on groundspeed. As the drone slows down for a turn, the pump slows down to prevent burning the crop.
- **Terrain Awareness:** Using `TERRAIN` altitude frames ensures the boom height stays constant over rolling hills.

MISSION PLAN

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





SEQ	COMMAND	PARAMETERS	DESCRIPTION
1	`DO_SET_HOME`	Current: 1	Set Refill Station as Home.
2	`NAV_TAKEOFF`	Alt: 5m	Low altitude takeoff.
3	`DO_CHANGE_SPEED`	Speed: 6 m/s	Set optimal spray speed.
4	`NAV_WAYPOINT`	Lat/Lon: [Field Corner 1]	Transit to start.
5	`DO_SPRAYER`	Enable: 1	**Pump ON.**
6	`NAV_WAYPOINT`	Lat/Lon: [Field Corner 2]	Spray Leg 1.
7	`DO_SPRAYER`	Enable: 0	**Pump OFF** for turn.
8	`NAV_WAYPOINT`	Lat/Lon: [Field Corner 3]	Move to next row.
9	`DO_SPRAYER`	Enable: 1	**Pump ON.**
10	`NAV_WAYPOINT`	Lat/Lon: [Field Corner 4]	Spray Leg 2.
...	...	...	*Repeat for entire grid...*
50	`NAV_RETURN_TO_LAUNCH`	-	Return to refill station.

ADVANCED LOGIC: THE "EMPTY TANK" FAILSAFE

Instead of guessing when the tank is empty, use a Lua script or a flow meter:

1. **Detection:** When the flow sensor indicates 0L remaining.
2. **Action:** The script triggers `NAV_RETURN_TO_LAUNCH`.
3. **Resume:** After refilling, the operator uses the "Resume Mission" function. Because we documented `DO_SET_RESUME_REPEAT_DIST`, the drone will fly back to the exact point it left off, back up 10 meters, and resume spraying to ensure no crop is missed.

PRO-TIPS

- **Turn Radius:** Use `NAV_SPLINE_WAYPOINT` for turns. Smooth turns maintain momentum, allowing the sprayer logic to keep the pump running more consistently than stop-and-go 90-degree turns.

MISSION SCENARIO: DRONE LIGHT SHOW (SWARM)

OBJECTIVE

Coordinate 50 drones to launch simultaneously and form a geometric shape in the sky. Timing must be synchronized to the millisecond to prevent collisions and ensure visual coherence.

VEHICLE CONFIGURATION

- **Type:** Small Quadcopters (RTK GPS required).
- **Payload:** RGB LED Module.
- **Comms:** Broadcast MAVLink over Wifi (UDP).

KEY CONCEPTS

- **UTC Synchronization:** Using `NAV_DELAY` with absolute UTC time ensures every drone executes the next command at the exact same instant, regardless of boot time or minor GPS variances.

Take Your Professional Drone Operations
to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY





- **Geofencing:** Strict `NAV_FENCE_POLYGON` boundaries prevent "rogue" drones from leaving the performance volume.

MISSION PLAN (DRONE #12 OF 50)

SEQ	COMMAND	PARAMETERS	DESCRIPTION
1	`NAV_DELAY`	Time: [20:00:00 UTC]	Wait for the "Show Start" time.
2	`DO_SET_SERVO`	Ch: 5, PWM: 1500	LEDs: **RED** (Standby).
3	`NAV_TAKEOFF`	Alt: 30m	Climb to formation altitude.
4	`NAV_DELAY`	Time: [20:00:15 UTC]	Wait for entire swarm to reach altitude.
5	`DO_SET_SERVO`	Ch: 5, PWM: 1900	LEDs: **GREEN** (Formation Start).
6	`NAV_WAYPOINT`	Lat/Lon: [Shape Pos 12]	Move to assigned vertex in the "Star" shape.
7	`NAV_DELAY`	Time: [20:00:45 UTC]	Hold shape for 30 seconds.
8	`DO_SET_SERVO`	Ch: 5, PWM: 1100	LEDs: **BLUE** (Transition).
9	`NAV_SPLINE_WAYPOINT`	Lat/Lon: [Helix Pos 12]	Fly a curved path to the "DNA Helix" shape.
10	`NAV_RETURN_TO_LAUNCH`	-	End of show.

PRO-TIPS

- **RTK is Mandatory:** For swarm spacing of < 2 meters, standard GPS (accuracy 2.5m) is dangerous. RTK (accuracy 0.02m) is required.
- **LED Control:** Use `DO_SET_SERVO` if your LEDs act like servos (PWM). If you use WS2812 LEDs connected to a flight controller output, use Lua Scripting and `DO_SEND_SCRIPT_MESSAGE` to trigger complex animations like "Rainbow Fade" or "Sparkle" which are stored on the drone itself.

Take Your Professional Drone Operations to the next level with MAVLink HUD
GET IT ON GOOGLE PLAY