MAVLINK GPS FIELD MANUAL

OPERATOR'S HANDBOOK

VERSION 1.0GENERATED 2026

MAVLink GPS
Indoor Navigation for ArduPilot
AVAILABLE ON GOOGLE PLAY


Table of Contents

MAVLink GPS
Indoor Navigation for ArduPilot
AVAILABLE ON GOOGLE PLAY

MAVLink GPS
Indoor Navigation for ArduPilot
AVAILABLE ON GOOGLE PLAY


CHAPTER 1: PHILOSOPHY OF VIRTUAL NAVIGATION

THE "INDOOR PROBLEM" EXPLAINED

THE PHYSICS OF FRAGILITY

Global Navigation Satellite Systems (GNSS), whether GPS (USA), Galileo (EU), or GLONASS (RU), all suffer from the same fundamental weakness: **Physics.**

Standard civilian GPS signals operate in the L-Band, specifically L1 at **1575.42 MHz**.

- **Wavelength:** ~19 cm (7.5 inches).
- **Power:** Signals arrive at Earth's surface at approximately **-160 dBm**. This is equivalent to detecting a 25-watt lightbulb from 10,000 miles away.

CONCRETE ATTENUATION

Because the signal is so incredibly weak, it relies on "Line of Sight" (LoS).

- **Drywall/Wood:** Passes signal with minor attenuation (3-5 dB loss). You might get a lock in a wood-frame house.
- **Concrete/Steel:** Concrete is a mixture of aggregate and water. Water absorbs RF energy at 1.5 GHz. Rebar (steel) acts as a Faraday cage.
- **Result:** A standard concrete roof provides **30-40 dB** of attenuation. The signal drops below the receiver's thermal noise floor, and the GPS module goes deaf.

THE MULTIPATH NIGHTMARE

Sometimes, you *do* get a signal indoors, but it's worse than no signal at all. This is **Multipath Interference**.

1. **The Mechanism:** GPS calculates position by measuring the "Time of Flight" of the radio wave.
2. **The Error:** In an urban canyon or warehouse, the direct signal is blocked. The receiver locks onto a signal that bounced off a nearby building or wall.
3. **The Drift:** The bounced path is longer than the direct path. The receiver thinks the satellite is further away, shifting your calculated position by 10-50 meters.
4. **The "Twitch":** As the drone moves, the reflection geometry changes rapidly, causing the position solution to jump wildly.

ARDUPILOT'S PERSPECTIVE

To the Flight Controller, this manifests as a degradation of the `GPS_Status` enum (defined in `libraries/AP_GPS/AP_GPS.h`).

- `GPS_OK_FIX_3D` **(3):** Ideal state. Lock on 4+ satellites with good geometry (HDOP < 2.0).
- `GPS_OK_FIX_2D` **(2):** Dangerous. Only 3 satellites visible. Altitude is unknown/unreliable. EKF will likely reject this.
- `NO_FIX` **(1):** Signal lost. The EKF stops fusing position data. If flying in Loiter/Auto, the drone triggers a "GPS Failsafe" and switches to Land or AltHold.

THE NETWORK FUSION SOLUTION

"Virtual GPS" solves this by abandoning the L-Band entirely.

- **WiFi:** 2.4 GHz signals are stronger and emitted from terrestrial routers *inside* the building.
- **Cellular:** 700-2100 MHz signals from towers have significantly higher transmit power than satellites.

MAVLink GPS
Indoor Navigation for ArduPilot
AVAILABLE ON GOOGLE PLAY


- **The Result:** By triangulating these strong, local signals, we can generate a valid 3D position fix where satellite signals simply do not exist.

SOURCE CODE REFERENCE

- **GPS Status Codes:** `libraries/AP_GPS/AP_GPS.h` - See `GPS_Status` enum values.

VIRTUAL GPS VS. OPTICAL FLOW: A DEEP DIVE

THE "INDOOR PROBLEM"

When GNSS signals (GPS/Galileo/BeiDou) are blocked by concrete or steel, drones lose their absolute positioning. To prevent a failsafe (switching to AltHold or Land), the autopilot needs an alternative source of X/Y velocity or position data. The two most common solutions are **Optical Flow** and **Virtual GPS (Network Fusion)**.

While both enable non-GPS flight, they rely on fundamentally different physical principles and suffer from opposing failure modes. This guide breaks down the architectural differences to help you choose the right tool for the job.

ARCHITECTURAL COMPARISON

FEATURE	OPTICAL FLOW	VIRTUAL GPS (NETWORK FUSION)
Data Type	Relative Velocity (`AID_RELATIVE`)	Absolute Position (`AID_ABSOLUTE`)
Primary Sensor	Downward-facing Camera	WiFi / Cellular Radio
Environment Req	Textured Surface, Light	WiFi Access Points, Cell Towers
Failure Mode	Darkness, Smooth Floors, High Speed	Faraday Cages, Remote Areas
EKF Handling	Secondary Aiding	Primary Source (Treated as GPS)

WHY OPTICAL FLOW FAILS

Optical Flow sensors (like the ThoneFlow, HereFlow, or PX4Flow) calculate velocity by tracking the movement of contrast features (pixels) on the ground. They are essentially low-resolution cameras running a continuous image processing loop.

1. THE TEXTURE TRAP

The underlying algorithm (often Lucas-Kanade) requires "features"—distinct points of contrast—to track frame-to-frame.

- **The Physics:** If you point a camera at a white wall and move it, the image doesn't change. The algorithm sees zero pixel displacement and reports zero velocity, even if the drone is drifting at 2 m/s.
- **The Code:** The EKF3 (Extended Kalman Filter) monitors a metric called `surface_quality` (or `SQUAL`). In `AP_NavEKF3_Measurements.cpp` , the EKF has a hard gate: if `surface_quality` drops below a threshold (often 0), the measurement is rejected entirely.
- **The Reality:** Shiny floors (gymnasiums), water, or uniform concrete (warehouses) have "low texture." The camera sees a uniform blur. With no features to track, `surface_quality` drops, and the EKF rejects the data, causing a "loss of navigation" failsafe.

2. THE LIGHTING CONSTRAINT

Flow sensors are passive cameras. They need photons to generate contrast.

MAVLink GPS
Indoor Navigation for ArduPilot
AVAILABLE ON GOOGLE PLAY


- **The Reality:** Flying at night, in dim warehouses, or under bridges often provides insufficient light for the sensor's electronic shutter speed to freeze motion. The result is motion blur, which destroys feature tracking.

3. MOTION LIMITS

- **The Code:** The EKF3 has a hard-coded gate for angular rates. If the gyro reports rotation > `4.2 $rad/s$`, flow data is ignored.
- **The Reality:** If the drone yaws or pitches too aggressively, the pixel motion blur exceeds the sensor's correlation capability. The EKF stops trusting the sensor precisely when you need it most (during a rapid maneuver).

THE VIRTUAL GPS ADVANTAGE

"Virtual GPS" bypasses the visual spectrum entirely by using the RF (Radio Frequency) spectrum. It injects position data directly into the EKF's top-tier logic.

1. THE EKF HIERARCHY (`AID_ABSOLUTE`)

ArduPilot's EKF3 uses a state machine to decide which sensor to trust.

- **Priority 1: GPS (`AID_ABSOLUTE`)** - The preferred mode. It constrains both position and velocity drift.
- **Priority 2: Optical Flow (`AID_RELATIVE`)** - A fallback mode. It only constrains velocity. Position is estimated by integrating velocity, which accumulates error (drift) over time.

Virtual GPS provides `AID_ABSOLUTE` data. This means the EKF treats it with the same high priority as a real u-blox GPS module. It does not "downgrade" to relative navigation; it stays in the highest accuracy mode.

2. IMMUNE TO "VISUAL" NOISE

- **Darkness:** WiFi signals penetrate darkness. You can fly in pitch black.
- **Texture:** WiFi signals don't care if the floor is shiny, matte, or liquid.
- **Smoke/Dust:** In emergency response (fire/rescue), smoke obscures optical sensors. RF signals punch through smoke, allowing for reliable positioning in low-visibility environments.

3. SCENARIO: THE WAREHOUSE TRANSITION

Imagine a drone flying from a well-lit loading dock into a dark, unlit aisle.

- **With Optical Flow:** As the drone enters the dark aisle, `surface_quality` drops. The EKF rejects the flow data. The drone enters "Dead Reckoning" mode (`AID_NONE`), rapidly drifting until it hits a rack.
- **With Virtual GPS:** The phone continues to see the WiFi Access Points through the walls. The position fix remains valid. The EKF stays in `AID_ABSOLUTE` mode, and the drone holds its position in the dark aisle.

SOURCE CODE REFERENCE

- **Aiding Mode Logic:** `libraries/AP_NavEKF3/AP_NavEKF3_Control.cpp` - See `setAidingMode` and the preference for `readyToUseGPS()` over `readyToUseOptFlow()`.
- **Flow Gating:** `libraries/AP_NavEKF3/AP_NavEKF3_Measurements.cpp` - See `writeOptFlowMeas` and the `surface_quality` check.

THE "HUMAN SENSOR" ANALOGY

DRONE PROPRIOCEPTION

To understand why "Virtual GPS" is difficult, we must understand how a drone perceives the world.

MAVLink GPS
Indoor Navigation for ArduPilot
AVAILABLE ON GOOGLE PLAY


A drone's internal IMU (Inertial Measurement Unit) is like the human **proprioceptive system** (muscle memory and nerves).

- **Accelerometer:** Senses gravity and linear acceleration. It knows *down*.
- **Gyroscope:** Senses rotation. It knows *spin*.
- **Update Rate:** 400Hz - 1000Hz (Fast).

The drone knows *exactly* how its body is moving instant-to-instant. But without eyes (GPS/Optical Flow), it has no idea **where** it is in the room. It is like a blindfolded gymnast—perfect balance, but eventually, they will drift into a wall.

THE PHONE AS THE VESTIBULAR SYSTEM

Injecting Android location data is like giving that blindfolded gymnast a spotter who shouts coordinates every few milliseconds.

The Android phone acts as the **Vestibular System** (Inner Ear) + **Vision**.

- **Network Fusion:** Provides the "Absolute Position" (X, Y, Z coordinates).
- **Update Rate:** 1Hz - 10Hz (Slow).

THE LATENCY CHALLENGE: " THE DIZZY BAT"

The core engineering challenge in MAVLink GPS is **Latency**.

1. **The IMU (Drone)** feels a movement at `T=0ms`.
2. **The Phone (Android)** detects the position change at `T=100ms` (due to filtering delay).
3. **The USB Link** transmits the data at `T=120ms`.
4. **The Flight Controller** receives it at `T=130ms`.

If the EKF (Extended Kalman Filter) simply applied the position update *now* (`T=130ms`), it would correct for a movement that happened 130ms ago. This is like trying to balance on one foot while your inner ear is delayed by half a second. The result is **Oscillation** (the "Toilet Bowl Effect").

THE SOLUTION: TIME TRAVEL

This is why the `time_usec` field in the MAVLink `GPS_INPUT` message is critical.

- We timestamp the location *when Android measured it* (`T=100ms`).
- We send it to the drone.
- The EKF3 receives it at `T=130ms`.
- **The Magic:** The EKF3 keeps a "history buffer" of its state. It looks at the timestamp (`T=100ms`), "rewinds" its internal math to that moment, applies the correction, and then "fast forwards" back to the present (`T=130ms`).

This allows the slow, delayed phone data to perfectly fuse with the fast, instant IMU data, stabilizing the drone without oscillation.

SAFETY FIRST: THE "DEAD MAN'S SWITCH"

THE DANGER OF "STALE" DATA

In standard GPS operations, a signal doesn't just "cut out" instantly. It fades. As satellites slip behind buildings or trees, the HDOP (Horizontal Dilution of Precision) spikes, and the position estimate starts to wander.

If an external app (like MAVLink GPS) simply stops updating the position but keeps the connection open, the Flight Controller might think the drone is perfectly stationary.

- **The Scenario:** You minimize the app to check a text message.

MAVLink GPS
Indoor Navigation for ArduPilot
AVAILABLE ON GOOGLE PLAY


- **The Result:** The app pauses. The last sent coordinate was "Lat: X, Lon: Y". The Flight Controller receives nothing new.
- **The Crash:** The EKF (Extended Kalman Filter) sees the drone physically drifting (via accelerometers) but the GPS says "Still at X,Y". The EKF fights the IMU, potentially causing a "Toilet Bowl" oscillation or a fly-away as it tries to correct a phantom error.

ARDUPILOT'S FAILSAFE LOGIC

ArduPilot is designed to handle a *complete loss* of signal better than a *frozen* signal.

1. THE 10-SECOND RULE (GCS FAILSAFE)

If you are sending data via MAVLink (simulating a Ground Control Station), ArduPilot has a FS_GCS_ENABLE parameter.

- **Timeout:** 5.0 seconds (default).
- **Action:** RTL or LAND.
- **Problem:** 5 seconds is an eternity for an indoor drone. It will hit a wall before this triggers.

2. THE EKF DATA GATE

The EKF3 is much faster. It monitors the "age" of the GPS data.

- **Logic:** If `GPS_INPUT` data is older than ~500ms (variable based on settings), the EKF stops fusing it to prevent state divergence.
- **Goal:** We want to trigger this *immediately* when confidence is lost.

THE ANDROID SOLUTION: "KILL ON SIGHT"

To align with ArduPilot's safety architecture, MAVLink GPS implements a ruthless "Dead Man's Switch."

1. FOREGROUND SERVICE NECESSITY

Android destroys background processes aggressively to save battery (Doze Mode). To guarantee real-time delivery:

- The app runs a **Foreground Service** with a persistent notification.
- This forces the Android Scheduler to treat the GPS thread with the same priority as a voice call.

2. THE SCREEN-OFF KILL SWITCH

If the user turns off the screen or minimizes the app, we cannot guarantee the 10Hz update rate required for stable hover.

- **Action:** The app proactively **closes the USB Serial port** or **terminates the UDP stream**.
- **Result:** ArduPilot immediately sees a "No GPS" state (not a "Frozen GPS" state).
- **Outcome:** The drone switches to `AltHold` (Altitude Hold) or `Land` instantly, preventing a fly-away. It is safer to drift slowly in `AltHold` than to violently correct for frozen GPS data.

BEST PRACTICE

Never fly with the screen off. Always keep the app in the foreground. If you need to switch apps, land the drone first.

SOURCE CODE REFERENCE

- **ArduPilot Failsafe:** `ArduCopter/failsafe.cpp` - Logic for GCS and GPS loss handling.
- **EKF Fusion:** `libraries/AP_NavEKF3/AP_NavEKF3_PosVelFusion.cpp` - See `fusingGPS` logic and data age checks.

MAVLink GPS
Indoor Navigation for ArduPilot
AVAILABLE ON GOOGLE PLAY


CHAPTER 2: ANDROID LOCATION ENGINEERING

INSIDE THE FUSED LOCATION PROVIDER (FLP)

DECONSTRUCTING THE BLACK BOX

When you ask Android for `ACCESS_FINE_LOCATION`, you aren't just turning on a GPS chip. You are engaging a massive, multi-sensor fusion engine called the **Fused Location Provider (FLP)**.

THE INPUTS

The FLP consumes data from three primary hardware layers:

1. **GNSS HAL:** Raw satellite data (GPS, GLONASS, Galileo, BeiDou).
   - *Strengths:* High accuracy outdoors.
   - *Weaknesses:* Fails indoors, slow cold start.
2. **Network Provider:** Scans for WiFi BSSIDs (Router MAC addresses) and Cell Tower IDs.
   - *Strengths:* Works instantly indoors. No sky view needed.
   - *Weaknesses:* Lower accuracy (10m - 50m).
3. **Sensor Hub:** Accelerometer, Gyroscope, Magnetometer, Barometer.
   - *Strengths:* High frequency (200Hz+). Detects steps/movement.
   - *Weaknesses:* Drifts over time.

THE FUSION ENGINE

Just like ArduPilot's EKF3, the Android FLP uses a Kalman Filter to blend these inputs.

- **Scenario:** You walk into a mall.
- **GPS:** Signal fades and vanishes.
- **Network:** The phone sees 50 WiFi routers. It queries Google's massive database to triangulate your position based on signal strength (RSSI).
- **Sensors:** The accelerometer detects your "Step Count" and walking direction (PDR - Pedestrian Dead Reckoning).
- **Result:** The blue dot continues to move smoothly through the mall, transitioning seamlessly from Satellite to WiFi navigation.

AOSP VS. GMS

It is important to note that the *smart* version of the FLP is proprietary.

- **AOSP (Android Open Source Project):** Contains a basic `FusedLocationProvider` that simply passes through the best available provider.
- **GMS (Google Mobile Services):** Contains the advanced logic, WiFi RTT (Round Trip Time) ranging, and the massive cloud database of WiFi router locations.

MAVLink GPS relies on the **GMS** implementation to provide "Concrete Penetrating" navigation.

SOURCE CODE REFERENCE

- **Framework Proxy:** `frameworks/base/services/core/java/com/android/server/location/provider/proxy/ProxyLocationProvider.java` - The bridge between the system and the GMS service.

MAVLink GPS
Indoor Navigation for ArduPilot
AVAILABLE ON GOOGLE PLAY


THE "THROTTLING" PROBLEM

THE BATTERY WAR

For the last decade, Android updates have had one primary goal: **Battery Life.** The biggest enemy of battery life is the GPS chip. To combat this, Android aggressively "throttles" or kills apps that try to use the GPS while the screen is off.

THE CONSTRAINTS

1. BACKGROUND LOCATION LIMITS (ANDROID 8.0+)

If an app is in the background (user switched to another app or screen is off), Android limits location updates to **a few times per hour**.

- **For Drones:** This is fatal. We need 5-10 updates per *second*.

2. DOZE MODE

If the phone is stationary for a while (e.g., sitting on the ground), Android enters "Doze."

- **Effect:** Network access (UDP) is cut. CPU wakes are deferred. The app goes into a coma.

THE SOLUTION: FOREGROUND SERVICES

To survive, MAVLink GPS declares itself as a **Foreground Service** of type `location`.

THE "STICKY" NOTIFICATION

You will notice a persistent notification in your status bar while the app is running. This is not just for UI; it is a mandatory system flag.

- **The Signal:** It tells the Android Scheduler: *"The user is actively using this app right now, even if they aren't looking at it."*
- **The Result:** The app is elevated to `PERCEPTIBLE` priority (similar to a music player or an active phone call). It is exempt from background location throttling and Doze mode restrictions.

WHY SCREEN-ON IS SAFER

Even with a Foreground Service, some aggressive OEM battery managers (Samsung, Xiaomi) may still kill the USB connection to save power.

- **Recommendation:** Use the app's **"Keep Screen On"** setting. This guarantees the highest possible CPU and sensor priority.

THE WIFI SCAN THROTTLE

WHY INDOOR GPS LAGS

Users often notice that while outdoor GPS updates at 5-10Hz, the indoor "Network Fusion" position sometimes stutters or updates slowly (e.g., once every 30 seconds). This is not a sensor failure; it is an Android battery preservation feature.

THE "4 SCANS PER 2 MINUTES" RULE

Starting with Android 9 (Pie), Google introduced strict limits on how often an app can trigger a WiFi scan, even when in the foreground.

MAVLink GPS
Indoor Navigation for ArduPilot
AVAILABLE ON GOOGLE PLAY


THE SOURCE CODE

The logic is enforced in the system's `ScanRequestProxy.java`.

- **Foreground Apps:** Limited to **4 scans every 120 seconds**.
- **Background Apps:** Limited to **1 scan every 30 minutes**.

```
// From com.android.server.wifi.ScanRequestProxy.java
public static final int SCAN_REQUEST_THROTTLE_TIME_WINDOW_FG_APPS_MS = 120 * 1000;
public static final int SCAN_REQUEST_THROTTLE_MAX_IN_TIME_WINDOW_FG_APPS = 4;
public static final int SCAN_REQUEST_THROTTLE_INTERVAL_BG_APPS_MS = 30 * 60 * 1000;
```

THE IMPACT ON FLIGHT

If you are flying indoors using WiFi RTT or standard WiFi triangulation:

1. The app requests a scan. Position updates.
2. App requests another. Position updates.
3. App requests a fifth scan. **Blocked.**
4. The drone's position "freezes" for the next ~110 seconds until the throttle window resets.

THE FIX: DEVELOPER OPTIONS

For reliable indoor flight, you **MUST** disable this throttling.

1. Enable **Developer Options** (Tap Build Number 7 times).
2. Find **"WiFi scan throttling"** (under Networking).
3. **Turn it OFF.**

This allows MAVLink GPS to scan as fast as the WiFi chip allows (typically 1-2Hz), ensuring smooth position updates.

SOURCE CODE REFERENCE

- **Throttling Logic:** `packages/modules/Wifi/service/java/com/android/server/wifi/ScanRequestProxy.java` - Search for `SCAN_REQUEST_THROTTLE_MAX_IN_TIME_WINDOW_FG_APPS`.

DECONSTRUCTING DOZE MODE

THE MYTH OF THE "FOREGROUND SERVICE"

Many developers (and pilots) believe that simply showing a persistent notification (Foreground Service) makes an app immune to Android's power saving features. **This is false.**

TWO TYPES OF "SLEEP"

Android has two distinct power-saving mechanisms:

1. APP STANDBY (THE BUCKET SYSTEM)

- **What it does:** Sorts apps into "Active", "Working Set", "Frequent", and "Rare" buckets. Limits job execution and network access for "Rare" apps.
- **Foreground Service Effect: Exempt.** As long as the notification is visible, the app is considered "Active".
- **Result:** The app won't be throttled *individually*.

MAVLink GPS
Indoor Navigation for ArduPilot
AVAILABLE ON GOOGLE PLAY


2. DOZE MODE (THE COMA)

- **What it does:** When the device is stationary with the screen off for a period of time, the *entire system* enters a deep sleep.
- **Restrictions:**
  - **Network:** Blocked. UDP packets from your drone will be dropped.
  - **Wake Locks:** Ignored. The CPU suspends even if you hold a partial wake lock.
  - **Alarms:** Deferred.
- **Foreground Service Effect: NOT Exempt.** A foreground service *will* be suspended by Doze.

THE SOLUTION: "IGNORE BATTERY OPTIMIZATIONS"

To survive Doze Mode, MAVLink GPS must be added to the **User Power Save Whitelist**.

- **Action:** The user must go to *Settings > Apps > MAVLink GPS > Battery > Unrestricted*.
- **AOSP Mechanism:** This adds the package to the `mPowerSaveWhitelistUserApps` list in `DeviceIdleController.java`.
- **Result:** The system grants the app network access and honors its wake locks even when the rest of the device is dozing.

USB HOST MODE EXCEPTION

There is one loophole. When the phone is acting as a **USB Host** (powering the Flight Controller or communicating via OTG):

- The kernel driver often holds a low-level wake lock to manage the bus traffic.
- This *can* prevent the device from entering deep sleep, effectively bypassing Doze without user intervention.
- *Warning:* This depends heavily on the specific kernel implementation of the phone manufacturer (Samsung vs. Pixel vs. Xiaomi). Do not rely on it. Always whitelist the app.

SOURCE CODE REFERENCE

- **Idle Controller:** `frameworks/base/apex/jobscheduler/service/java/com/android/server/DeviceIdleController.java` - Manages the whitelist and Deep Doze transitions.

DUAL-FREQUENCY GNSS (L1/L5)

THE HARDWARE ADVANTAGE

For indoor-adjacent or transition-zone flight (stadiums with open roofs, glass atriums), the physical GNSS chip in the smartphone is the primary bottleneck. Standard single-frequency receivers are prone to "Urban Canyon" multipath, but **Dual-Frequency L1/L5** support changes the physics of the fix.

THE MULTIPATH PROBLEM (L1 ONLY)

Legacy GNSS chips rely on the **L1 Band** (1575.42 MHz).

- **Chip Rate:** 1.023 MHz.
- **Resolution:** A single code bit is roughly 300 meters wide.
- **The Trap:** In a reflective environment like a stadium seating bowl, a signal bouncing off a concrete wall arrives just slightly after the direct signal. Because the L1 pulses are so wide, the receiver's correlator smears them together. It cannot distinguish the "echo" from the "truth," leading to position jumps of 50-100 meters.

THE L5 SOLUTION

MAVLink GPS
Indoor Navigation for ArduPilot
AVAILABLE ON GOOGLE PLAY


Modern flagships track the **L5 Band** (1176.45 MHz) simultaneously.

- **Chip Rate:** 10.23 MHz (10x faster).
- **Resolution:** A code bit is only ~30 meters wide.
- **Leading Edge Detection:** Because the pulses are narrower, the receiver sees two distinct "peaks" in the correlation window: the Early (Direct Path) and the Late (Reflected Path).
- **Rejection:** The chip simply discards any peak arriving after the first one. This allows a drone to maintain a stable 3D fix even when satellite signals are bouncing off metallic truss structures.

VALIDATED CHIPSETS

To ensure reliable transition from indoor RTT to outdoor GPS, verify your device uses one of these subsystems:

1. **Broadcom BCM47755+:** The pioneer chip (Mi 8, Pixel 4/5).
2. **Qualcomm FastConnect 6900/7800:** Found in Snapdragon 8 Gen 2/3 devices.
3. **Google Tensor:** Uses integrated Exynos modems with robust L5 software compensation.

SOURCE CODE REFERENCE

- **Carrier Frequency:** `android/location/GnssMeasurement.java` - The API's `getCarrierFrequencyHz()` is the programmatic way to verify your app is actually receiving L5 packets.

GNSS RAW MEASUREMENTS (RTK FEASIBILITY)

THE DREAM: PHONE AS RTK BASE

A common question in the ArduPilot community: *"My phone has L1/L5 GPS. Can I use it as an RTK Base Station for my drone?"*

The theoretical answer is **Yes**. The practical answer is **No**.

THE ANDROID API (`GNSSMEASUREMENT`)

Android 7.0 (Nougat) introduced the ability to read raw GNSS observables, bypassing the standard "processed" location.

KEY DATA FIELDS

To perform Real-Time Kinematic (RTK) processing, you need **Carrier Phase** data. Android exposes this via:

- `getAccumulatedDeltaRangeMeters()`: This is the raw Carrier Phase measurement.
- `getAccumulatedDeltaRangeState()`: Tells you if the phase lock is valid or if a "Cycle Slip" occurred.
- `getCarrierFrequencyHz()`: Distinguishes L1 (1575.42 MHz) from L5 (1176.45 MHz).

```
// From android.location.GnssMeasurement
public double getAccumulatedDeltaRangeMeters() {
    return mAccumulatedDeltaRangeMeters;
}
```

THE HARDWARE BOTTLENECK

While the *software* supports RTK, the *hardware* (Antenna) usually does not.

1. LINEARLY POLARIZED ANTENNAS

MAVLink GPS
Indoor Navigation for ArduPilot
AVAILABLE ON GOOGLE PLAY


Survey-grade RTK antennas are **Circularly Polarized** (RHCP) to reject multipath reflections (which reverse polarization upon bounce). Phone antennas are **Linearly Polarized** (to fit in the slim case).

- **Result:** Phones accept reflected signals as valid data. This causes "Cycle Slips" where the receiver loses count of the carrier wave cycles.

2. DUTY CYCLING

To save battery, phone GNSS chips often turn off the RF front-end between updates (Duty Cycling).

- **Result:** The continuous phase lock required for RTK is broken 5 times per second.
- *Note:* Android 9 introduced `forceFullGnssMeasurements(true)` to disable this, but it drains battery fast.

CONCLUSION

You can use raw measurements for **Post-Processed Kinematic (PPK)** or smoothed DGPS to get sub-meter accuracy. But achieving a reliable **RTK Fixed** (1-2cm accuracy) solution with a phone's internal antenna is physically impossible in most environments due to multipath susceptibility.

SOURCE CODE REFERENCE

- **API Definition:** `android/location/GnssMeasurement.java` - Search for `getAccumulatedDeltaRangeMeters`.

MASTERING USB PERMISSIONS

THE "ALWAYS ALLOW" CHECKBOX

One of the most frustrating aspects of using Android with custom hardware is the relentless permission popup:

> **Allow MAVLink GPS to access [Device Name]?**

Even if you check **"Always open MAVLink GPS when ... is connected"**, the popup often returns the next day. Why?

THE ANATOMY OF A MATCH (`DEVICEFILTER`)

When you check that box, Android creates a "Filter" based on the connected device's unique signature and saves it to a system XML file. To bypass the popup next time, the new connection must match *every* field in that saved filter.

The comparison logic is strict (AOSP `DeviceFilter.java`):

1. **Vendor ID (VID):** Must match.
2. **Product ID (PID):** Must match.
3. **Serial Number:** Must match exactly (if stored).

THE SERIAL NUMBER TRAP

This is where cheap Flight Controllers fail.

SCENARIO 1: THE "RANDOMIZER" (BAD)

Some bootloaders generate a **random** USB Serial Number on every boot to avoid conflicts.

- **Day 1:** You connect. Serial=`ABC-123`. You check "Always Allow". Android saves `[VID:1234, PID:5678, Serial:ABC-123]`.
- **Day 2:** You reboot the drone. Serial=`DEF-456`.

MAVLink GPS
Indoor Navigation for ArduPilot
AVAILABLE ON GOOGLE PLAY


- **The Check:** Android compares `DEF-456` vs stored `ABC-123`. **Mismatch.**
- **Result:** The permission is denied, and the popup appears again.

SCENARIO 2: THE "CLONE" (GOOD?)

Cheap clones often have **no** serial number or a generic one (`00000000`).

- If the device reports *no* serial, Android saves `[VID:1234, PID:5678, Serial:null]`.
- Any subsequent device with that VID/PID will match. This actually makes the "Always Allow" feature work reliably, albeit less securely.

HARDWARE RECOMMENDATIONS

To ensure a true "Plug and Fly" experience, use flight controllers from reputable manufacturers (Hex Cube, Holybro, mRo) that burn a unique, static UUID into the STM32 USB descriptor.

SOURCE CODE REFERENCE

- **Matching Logic:** `frameworks/base/core/java/android/hardware/usb/DeviceFilter.java` - See the `matches(UsbDevice device)` method.

POSITION SOLVERS & SAFETY GATES

THE TRILATERATION PROBLEM

Ranging is only the first half of the equation. To fly, the Android app must solve for (x, y, z) given the distances $(d_1, d_2, d_3, \dots, d_n)$ to n Access Points at known coordinates.

NON-LINEAR LEAST SQUARES (NLLS)

Since RTT measurements are noisy, a standard geometric intersection will fail. You must use an NLLS solver to find the position that minimizes the error across all anchors.

- **Algorithm:** The **Levenberg-Marquardt** algorithm is the industry standard for this.
- **Performance:** For flight control, this should be implemented in **C++ via the Android NDK**. Java/Kotlin garbage collection spikes can add fatal latency to the solver loop.

THE SAFETY GATE (VALIDITY CHECKING)

Sending corrupt or jumpy data to ArduPilot is more dangerous than sending no data at all. The Extended Kalman Filter (EKF) can handle a lost signal, but it can be "distracted" by a jumpy one into a crash.

IMPLEMENTING A THRESHOLD

Your app must implement a "Validity Gate" before sending any MAVLink packets:

1. **Standard Deviation:** Calculate the σ of the latest burst of RTT measurements.
2. **Threshold:** If $\sigma > 2.0$ meters, **STOP** sending `GPS_INPUT` messages immediately.
3. **Hysteresis:** Do not resume sending data until σ has remained below 1.0m for at least 500ms.

THE DRONE'S REACTION

When the app stops sending packets, ArduPilot will detect a "GPS Timeout."

MAVLink GPS
Indoor Navigation for ArduPilot
AVAILABLE ON GOOGLE PLAY


- **If you have an Optical Flow sensor:** The drone will automatically switch to `FlowHold`, maintaining a steady hover.
- **If not:** The drone will switch to `AltHold`, and the pilot must take manual control.
- **The Benefit:** This is infinitely safer than the drone "teleporting" 5 meters and crashing into a stadium wall because of a multipath glitch.

MAVLink GPS
Indoor Navigation for ArduPilot
AVAILABLE ON GOOGLE PLAY

CHAPTER 3: THE GEODETIC BRIDGE

THE LUMPY EARTH: ELLIPSOID VS. GEOID

WHY IS MY DRONE FLYING INTO THE CEILING?

A common "First Flight" problem with Virtual GPS is altitude error. You hover the phone at 1 meter, but Mission Planner reports -25 meters or +40 meters. This is not a bug; it is a **Geodetic Reference Frame** mismatch.

To understand why, we must look at the mathematical shape of the Earth.

THE TWO SHAPES OF EARTH

1. THE ELLIPSOID (WGS84)

GPS satellites orbit the center of mass. They "see" the Earth as a perfect mathematical shape called an **Oblate Spheroid**. This surface is smooth and defined by the WGS84 standard.

- **Altitude (h):** Height above this mathematical surface.
- **Android API:** Older versions of `Location.getAltitude()` return this raw Ellipsoid height.

2. THE GEOID (EGM96 / MSL)

The Earth is not smooth. Mountains have mass; trenches have voids. This uneven mass distribution creates "lumps" in the gravity field.
The **Geoid** is an equipotential gravitational surface—essentially, where water would settle if it covered the entire planet (Mean Sea Level).

- **Altitude (H):** Height above this gravity surface. This is what barometers measure and what humans consider "Altitude".

THE FUNDAMENTAL EQUATION

The relationship between these two frames is defined by the **Geoid Undulation (N)**:

$$h = H + N$$

Where:

- h: **Ellipsoid Height** (Raw GPS).
- H: **Orthometric Height** (MSL / Geoid).
- N: **Geoid Undulation** (The separation between the two surfaces).

THE "N" VALUE

Depending on where you are on Earth, N varies significantly.

- **New York:** $N \approx -32m$.
- **London:** $N \approx 47m$.
- **India:** $N \approx -100m$.

USE CASE: THE "30-METER DROP"

Imagine you are flying in a warehouse in Chicago ($N \approx -34m$).

1. **The Mistake:** Your app sends raw Ellipsoid height (h) to ArduPilot. Let's say $h = 200m$.

MAVLink GPS
Indoor Navigation for ArduPilot
AVAILABLE ON GOOGLE PLAY


2. **The Interpretation:** ArduPilot expects MSL (H). It assumes the $200m$ it received is MSL.
3. **The Reality:** The actual MSL height is $H = h - N = 200 - (-34) = 234m$.
4. **The Crash:** There is a **34-meter vertical discrepancy** between the GPS frame and the Barometer frame. When the EKF attempts to fuse these, it may aggressively dive or climb to reconcile the conflicting data, leading to a crash into the ceiling or floor.

THE ARDUPILOT REQUIREMENT

ArduPilot's navigation logic (EKF) explicitly expects **MSL (Geoid) Altitude** for the `alt` field in the MAVLink `GPS_INPUT` message. It does *not* perform the EGM96 conversion itself for MAVLink-injected data.

Therefore, the Android application **MUST** subtract N before sending the packet:

$$\text{MAVLink}_{alt} = \text{Android}_{raw} - N_{local}$$

SOURCE CODE REFERENCE

- **MAVLink Handler:** `libraries/AP_GPS/AP_GPS_MAV.cpp` - The `handle_msg` function reads altitude directly without conversion.

SOLVING THE "N" VALUE

CALCULATING GEOID UNDULATION ON ANDROID

To send valid MSL altitude to ArduPilot, we must solve for N (Geoid Undulation) locally on the phone.

1. LEGACY APPROACH: NMEA PARSING

Older Android devices expose raw NMEA sentences (`$GPGGA`) via `OnNmeaMessageListener`.

- **The Fix:** The `$GPGGA` sentence contains the MSL altitude directly (calculated by the GNSS chip's firmware).
- **The Problem:** Not all phones expose NMEA, and parsing strings at 10Hz is CPU intensive.

2. THE GEOGRAPHICLIB APPROACH (EGM96)

We embed a compressed binary grid of the **EGM96** (Earth Gravitational Model 1996) dataset within the app.

- **Input:** Latitude, Longitude (from Android Location).
- **Process:** Look up the N-value from the grid using bilinear interpolation.
- **Output:** N (in meters).
- **Calculation:** `Altitude_MAVLink = Location.getAltitude() - N`

This guarantees consistent altitude regardless of the phone manufacturer's GNSS implementation.

3. ANDROID 14: ALTITUDECONVERTER

API Level 34 introduced `android.location.altitude.AltitudeConverter`.

- **Feature:** A native system class that adds MSL altitude to a `Location` object.
- **Mechanism:** It presumably uses a system-level Geoid model (likely EGM2008 or similar) to perform the conversion efficiently.
- **Adoption:** MAVLink GPS uses this API when available, falling back to our internal EGM96 grid for older devices.

WHY ACCURACY MATTERS

MAVLink GPS
Indoor Navigation for ArduPilot
AVAILABLE ON GOOGLE PLAY


An error of 5 meters in X/Y is annoying. An error of 5 meters in Z (Altitude) causes a crash.
By ensuring the N-value is subtracted correctly *before* the data hits the wire, we ensure the drone's barometer and the GPS agree on where "Zero" is.

SOURCE CODE REFERENCE

- **Android API:** `AltitudeConverter` documentation.

ANDROID 14 ALTITUDECONVERTER

NATIVE GEOID SUPPORT

Historically, Android only provided `getAltitude()` as WGS84 Ellipsoid height. Developers were forced to bundle massive binary grids (like EGM96) to calculate MSL altitude manually.

THE API 34 REVOLUTION

Android 14 (API Level 34) introduced the `android.location.altitude` package, specifically the `AltitudeConverter` class.

HOW IT WORKS

The system now maintains a shared, high-resolution Geoid model. Apps can simply request an MSL conversion without doing the heavy lifting.

1. **S2 Geometry:** The converter uses Google's **S2 Geometry** library to index the earth. It calculates the S2 Cell ID for the current latitude/longitude.
2. **Bilinear Interpolation:** It retrieves the geoid height (N-value) for the four corners of the S2 map square containing the user. It then performs bilinear interpolation to calculate the precise offset for the specific location.
3. **Correction:** It subtracts this offset from the WGS84 altitude:

   ```
   MSL = WGS84 - Offset
   ```

IMPLEMENTATION

- **Method:** `addMslAltitudeToLocation(Context context, Location location)`
- **Result:** It injects the MSL altitude into the `Location` object (accessible via `getMslAltitudeMeters()` in API 34+).
- **Source Reference:** See `AltitudeConverter.java` in AOSP.

WHY THIS MATTERS FOR DRONES

1. **Accuracy:** The S2-based interpolation is smoother than standard grid lookups, preventing "steps" in altitude as you fly across grid boundaries.
2. **Efficiency:** It offloads the math to the OS, saving CPU cycles in the app's critical 10Hz loop.
3. **Future Proofing:** As magnetic models and geoids update, the OS updates them via Play Services, ensuring your drone's altitude reference stays valid without app updates.

MAVLINK GPS IMPLEMENTATION

Our app uses a "Polyfill" strategy:

- **If Android 14+:** We call `AltitudeConverter`.
- **If Android 8-13:** We fall back to our internal, optimized EGM96 grid engine.

This ensures you get the best possible altitude precision regardless of your phone's age.

MAVLink GPS
Indoor Navigation for ArduPilot
AVAILABLE ON GOOGLE PLAY


SOURCE CODE REFERENCE

- **AOSP Implementation:** `android/location/altitude/AltitudeConverter.java`

MAVLink GPS
Indoor Navigation for ArduPilot
AVAILABLE ON GOOGLE PLAY


CHAPTER 4: THE MAVLINK PROTOCOL

THE GPS_INPUT MESSAGE (#232)

OVERVIEW

The `GPS_INPUT` message (ID 232) is the mechanism used to inject high-precision Android location data into ArduPilot. Unlike standard GPS modules which communicate via NMEA over a serial wire, this MAVLink message allows your phone to act as a "Virtual GPS" sensor.

PACKET STRUCTURE

The payload consists of **63 bytes**. While it looks like a standard GPS packet, how you fill it determines if the drone flies stable or oscillates.

C STRUCT DEFINITION

```
typedef struct __mavlink_gps_input_t {
    uint64_t time_usec;        /*< [us] Timestamp (UNIX Epoch or Boot time) */
    uint32_t time_week_ms;     /*< [ms] GPS time of week */
    int32_t lat;               /*< [degE7] Latitude */
    int32_t lon;               /*< [degE7] Longitude */
    float alt;                 /*< [m] Altitude (MSL) */
    float hdop;                /*< [m] GPS HDOP */
    float vdop;                /*< [m] GPS VDOP */
    float vn;                  /*< [m/s] GPS velocity in N direction */
    float ve;                  /*< [m/s] GPS velocity in E direction */
    float vd;                  /*< [m/s] GPS velocity in D direction */
    float speed_accuracy;      /*< [m/s] GPS speed accuracy */
    float horiz_accuracy;      /*< [m] GPS horizontal accuracy */
    float vert_accuracy;       /*< [m] GPS vertical accuracy */
    uint16_t ignore_flags;     /*< Bitmap of fields to ignore */
    uint16_t time_week;        /*< GPS week number */
    uint8_t gps_id;            /*< ID of the GPS for multiple GPS inputs */
    uint8_t fix_type;          /*< 0-1: no fix, 2: 2D fix, 3: 3D fix */
    uint8_t satellites_visible;/*< Number of satellites visible */
    uint16_t yaw;              /*< [cdeg] Yaw of vehicle relative to True North */
} mavlink_gps_input_t;
```

ADVANCED LOGIC: JITTER & LATENCY

Android is not a Real-Time OS. A measurement taken at $T = 0$ might not be sent until $T = 200ms$.

1. **The "Rewind" Effect:** If you provide a valid `time_week` and `time_week_ms`, ArduPilot applies **Jitter Correction**. It uses these fields to calculate a "lag-compensated" timestamp. The EKF3 can then "rewind" its state history to fuse the measurement at the exact moment it was captured, rather than where the drone is "now."
2. **Accuracy Weighting:** The `horiz_accuracy` field directly scales the **Observation Noise** in the Kalman Filter. If your phone reports a 2.0m accuracy, ArduPilot trusts its own IMU more. If you report 0.5m, it will aggressively snap the drone to the phone's coordinates.

HANDLING GPS TIME IN ANDROID

MAVLink GPS
Indoor Navigation for ArduPilot
AVAILABLE ON GOOGLE PLAY


Android `Location` objects use Unix Epoch (ms since 1970). You must convert this to the GPS Epoch (ms since Jan 6, 1980) and account for leap seconds:

$$t_{gps} = t_{unix} - 315,964,800,000 + 18,000$$

(Note: 18,000ms is the leap second offset as of 2024)

SOURCE CODE REFERENCE

- **ArduPilot Handler:** `libraries/AP_GPS/AP_GPS_MAV.cpp` - See `AP_GPS_MAV::handle_msg` for how the EKF ingests this data.

THE IGNORE_FLAGS BITMASK

WHY WE IGNORE DATA

In MAVLink GPS injection, sometimes "No Data" is safer than "Bad Data." The `ignore_flags` field is a bitmask that tells the Flight Controller to disregard specific values in the packet.

KEY FLAGS

BIT	DECIMAL	FLAG NAME	USAGE SCENARIO
0	1	`LAT_LON`	Never used. Position is primary.
1	2	`ALTITUDE`	Never used. Altitude is primary.
3	8	`VEL_HORIZ`	**Hovering.** When the phone is stationary, Doppler velocity is noisy. We set this to force EKF to rely on position delta.
4	16	`VEL_VERT`	**Always Set.** Phone Z-axis velocity is extremely noisy. We force ArduPilot to rely on the Barometer for climb rate.
5	32	`SPEED_ACC`	Optional.
6	64	`HORIZ_ACC`	Never used. Critical for EKF weighting.
7	128	`VERT_ACC`	Used if vertical accuracy is unknown/estimated poorly.

STRATEGY: THE "VEL VERT" LOCK

MAVLink GPS **always** sets bit 4 (Value 16).

- **Reason:** Phones are handheld. Small vertical movements (breathing, walking) creates noise in the vertical velocity estimate.
- **Effect:** The EKF ignores the `vd` (Velocity Down) field entirely. It fuses the `alt` (Position) with the `Barometer` (Pressure) to estimate climb rate. This provides a much smoother altitude hold.

STRATEGY: THE "ZERO CLAMP"

When the app detects the phone is stationary (e.g., set on a table or tripod):

1. It stops sending velocity data.
2. It sets `VEL_HORIZ` (Bit 3, Value 8).
3. **Result:** The EKF knows not to trust a velocity of "0.00". It calculates velocity purely by observing the change in Latitude/Longitude over time, which is mathematically cleaner for stationary station-keeping.

MAVLink GPS
Indoor Navigation for ArduPilot
AVAILABLE ON GOOGLE PLAY


SERIAL VS. UDP

THE TRANSPORT LAYER

MAVLink GPS supports two methods of connecting to the Flight Controller: **USB Serial** and **UDP Network**.

1. USB SERIAL (DIRECT)

This is the "Hardwired" approach. You connect the Android phone's USB-C port to the Flight Controller's USB-C (or Micro-USB) port using an OTG cable.

- **Latency:** Extremely low (< 5ms). This is critical for the "tightness" of the position hold loop.
- **Jitter:** Near zero. The data stream is isochronous.
- **Power:** The Flight Controller can charge the phone (or vice-versa, depending on cabling).
- **Verdict: REQUIRED FOR FLIGHT.** You should never trust a critical navigation sensor to a wireless link that can be jammed or dropped.

2. UDP (WIFI BRIDGE)

This approach routes the data through an ESP8266 or similar WiFi bridge running MAVLink.

- **Mechanism:** Android → WiFi → ESP8266 → UART → Flight Controller.
- **Latency:** High (20-100ms+). WiFi is a shared medium with collisions.
- **Packet Loss:** UDP makes no guarantee of delivery. Missing `GPS_INPUT` packets causes the EKF "Data Age" to spike, leading to rough flight performance.
- **Verdict: TESTING ONLY.** Useful for bench testing or developing the app without cables, but dangerous for actual flight navigation.

WHY USB IS KING

The EKF3 assumes the GPS data arrives with consistent timing.

- **Serial:** Consistent 10Hz heartbeat.
- **UDP:** Bursty. You might get 5 packets in 10ms, then silence for 200ms. This "Cluster/Gap" pattern confuses the velocity estimator.

SETTING GPS_TYPE = 14

CONFIGURING ARDUPILOT FOR MAVLINK INJECTION

To tell ArduPilot to "listen" for GPS data coming from a MAVLink message rather than a physical UART wire, you must configure the GPS driver backend.

1. THE PRIMARY PARAMETER

- **Parameter:** GPS_TYPE (or GPS1_TYPE on newer firmware).
- **Value: 14** (MAVLink).
- **Effect:** This loads the `AP_GPS_MAV` backend driver. This driver subscribes to the internal message bus, waiting specifically for `MAVLINK_MSG_ID_GPS_INPUT` (#232).

2. THE TRANSPORT LAYER

MAVLink GPS
Indoor Navigation for ArduPilot
AVAILABLE ON GOOGLE PLAY


Unlike a U-Blox module where you set `SERIAL1_PROTOCOL = 5` (GPS), for Virtual GPS you set the port to **MAVLink**.

- **Parameter:** SERIAL0_PROTOCOL (USB) or SERIAL1_PROTOCOL (Telem 1).
- **Value: 2** (MAVLink 2).
- **Why:** The `GPS_INPUT` message is just a standard MAVLink packet. It travels over the same link as your telemetry, heartbeat, and RC overrides.

3. EKF CONFIGURATION

Ensure the Extended Kalman Filter is active and ready to fuse GPS data.

- `EKF3_ENABLE` = 1
- AHRS_EKF_TYPE = 3

EKF3 SOURCE SELECTION (COPTER 4.1+)

To specifically prioritize the Phone's GPS data for horizontal position but use LiDAR for altitude (highly recommended):

- EK3_SRC1_POSXY = **3** (GPS)
- EK3_SRC1_VELXY = **3** (GPS)
- EK3_SRC1_POSZ = **2** (RangeFinder) -- **CRITICAL: Do not use GPS for indoor altitude.**
- EK3_SRC1_VELZ = **0** (None) -- Let EKF derive vertical velocity from IMU/LiDAR.

VERIFICATION

Once connected:

1. Open Mission Planner > Flight Data.
2. Look at the **GPS Status** label.
3. **No Fix:** The driver is loaded (`GPS_TYPE=14`), but no data is arriving from the phone.
4. **3D Fix:** The driver is receiving valid `GPS_INPUT` packets from the phone.

MULTIPLE GPS SETUP

You can use the phone as a secondary GPS.

- GPS1_TYPE = 1 (Auto) → Physical U-Blox on the roof.
- GPS2_TYPE = 14 (MAVLink) → Virtual GPS via USB.
- **Result:** ArduPilot will blend or switch between them based on GPS_AUTO_SWITCH logic.

MAVLink GPS
Indoor Navigation for ArduPilot
AVAILABLE ON GOOGLE PLAY


CHAPTER 5: FLIGHT DYNAMICS & EKF3 TUNING

THE "TOILET BOWL" EFFECT

DIAGNOSIS

You switch to Loiter. The drone holds for a moment, then starts to circle. The circle gets wider and faster with every revolution. This is the **Toilet Bowl Effect** (TBE). It is a classic instability in the position control loop.

THE MECHANICS OF FAILURE

TBE is caused by a disagreement between the drone's **Heading** (where it thinks it is pointing) and its **Track** (where the GPS says it is moving).

THE VECTOR ROTATION

The flight controller calculates a position error vector ($\vec{E}$) to correct drift.

$$\vec{E} = \vec{P}_{target} - \vec{P}_{current}$$

It then attempts to apply a correction acceleration ($\vec{A}$) by rotating this error vector into the body frame using the Compass Heading (ψ).

$$\vec{A}_{body} = R(\psi) \cdot \vec{E}_{NED}$$

THE ERROR

If your Compass is rotated by $\theta = 90^\circ$ (due to magnetic interference):

1. **Drift:** The drone drifts North.
2. **GPS:** Reports "Moving North".
3. **FC Logic:** "To stop moving North, I must lean South."
4. **Reality:** Because ψ is wrong, "South" in the body frame is actually **West**.
5. **Result:** The drone leans West. It accelerates sideways.
6. **Loop:** The GPS now sees Westward drift. The FC tries to correct... and pushes South. The error vector rotates continuously, driving the drone into a widening spiral.

VIRTUAL GPS SPECIFICS

Indoor environments are notoriously bad for compasses. Steel beams, reinforced concrete, and electrical wiring create magnetic fields that warp the Earth's field, changing ψ dynamically as you fly past pillars.

THE SOLUTION: MAG-LESS FLIGHT

If you have a solid Virtual GPS fix, you might not need a compass at all.

- **GPS Heading:** If the drone is moving, the EKF can infer heading from the velocity vector (EK3_MAG_CAL options).
- **External Yaw:** If your positioning system provides Yaw (e.g., Visual Odometry or Dual Antenna GPS), you can inject this via the `yaw` field in `GPS_INPUT` (centidegrees) and disable the internal compass.

SOURCE CODE REFERENCE

MAVLink GPS
Indoor Navigation for ArduPilot
AVAILABLE ON GOOGLE PLAY


- **State Prediction:** `libraries/AP_NavEKF3/AP_NavEKF3_core.cpp` - See `UpdateStrapdownEquationsNED` where the attitude quaternion rotates accelerations.
- **Fusion Logic:** `libraries/AP_NavEKF3/AP_NavEKF3_PosVelFusion.cpp` - See `FuseVelPosNED`.

EKF3 "REWIND": TIME HORIZONS

THE LATENCY REALITY

In a perfect world, sensors are instant. In the real world, especially with Android, latency is unavoidable.

1. **Sensor Lag:** The GPS chip takes time to calculate a fix.
2. **OS Lag:** Android buffers the location intent.
3. **Network Lag:** USB/UDP transmission takes milliseconds.
4. **Processing Lag:** The Flight Controller parses the MAVLink message.

By the time ArduPilot "sees" a position update, it is already **100-300ms old**.

THE OLD WAY: "JUST IGNORE IT"

Legacy flight controllers (like APM 2.6) assumed GPS data was "now".

- **Result:** If the drone moved 1 meter in that 300ms delay, the controller would calculate an error based on where it *is*, not where it *was*. This lag in the feedback loop causes Pilot-Induced Oscillation (PIO) or "Toilet Bowling."

THE EKF3 WAY: TIME TRAVEL

The Extended Kalman Filter 3 (EKF3) maintains a **History Buffer** (or "Time Horizon").

1. THE IMU BUFFER

The EKF stores a ring buffer of IMU states (Gyro, Accel) for the last ~375ms (configurable via `maxTimeDelay`). It knows exactly where the drone was and how it was oriented for every millisecond of the recent past.

2. THE REWIND EVENT

When a `GPS_INPUT` message arrives with a valid `time_usec` timestamp:

1. **Recall:** The EKF pauses the current frame.
2. **Rewind:** It looks up the vehicle state (Position/Velocity) from the buffer at the exact moment of `time_usec`.
3. **Innovate:** It compares the GPS measurement to the *historical* state to calculate the error (Innovation).
4. **Propagate:** It applies that correction to the historical state and then re-integrates the IMU data forward to the present moment.

3. THE RESULT

The Flight Controller effectively "travels back in time" to apply the correction when the measurement actually happened. This mathematically eliminates the phase lag caused by Android/USB latency.

CRITICAL REQUIREMENT: ACCURATE TIMING

For this to work, the `time_usec` in the MAVLink message must be accurate to the millisecond relative to the flight controller's clock.

MAVLink GPS
Indoor Navigation for ArduPilot
AVAILABLE ON GOOGLE PLAY


- **Timesync:** MAVLink GPS performs a `TIMESYNC` handshake on connection to calculate the offset between the phone's clock and the autopilot's boot time.
- **Jitter:** If the timestamp jitters, the "rewind" targets the wrong moment, introducing noise.

SOURCE CODE REFERENCE

- **Buffer Calculation:** `libraries/AP_NavEKF3/AP_NavEKF3_core.cpp` - Logic for `maxTimeDelay_ms`.
- **Recall Logic:** `libraries/AP_NavEKF3/AP_NavEKF3_PosVelFusion.cpp` - See `storedGPS.recall`.

MANAGING VARIANCE (HORIZ_ACCURACY)

THE BLUE CIRCLE IS MATH

When you look at Google Maps on your phone, you see a blue dot (your position) surrounded by a light blue circle. That circle represents the **Horizontal Accuracy** (68\% confidence interval). To the casual user, it means "I am somewhere in this circle." To the **Extended Kalman Filter (EKF)**, it is a critical statistical weight that defines the **Observation Noise Covariance**.

MAVLink GPS passes this radius (in meters) directly to ArduPilot via the `horiz_accuracy` field in the `GPS_INPUT` message.

THE KALMAN GAIN EQUATION

The EKF3 is constantly arbitrating a debate between two sensors:

1. **The IMU (Prediction):** "I felt us accelerate forward at $2m/s^2$, so we must be at position $x_{k|k-1}$."
2. **The GPS (Observation):** "I see us at position z_k."

The arbiter of this debate is the **Kalman Gain (K_k)**. The simplified equation for the gain is:

$$K_k = \frac{P_{k|k-1}H^T}{HP_{k|k-1}H^T + R_k}$$

Where:

- $P_{k|k-1}$: The **Error Covariance** of the IMU prediction (How much we trust the gyro/accel).
- R_k: The **Measurement Noise Covariance** (How much we trust the GPS).

The `horiz_accuracy` value from your phone drives R_k directly.

- **Small Accuracy Circle (e.g., 2m):** R_k is small. The denominator decreases. K_k approaches 1. The EKF **trusts the GPS**.
- **Large Accuracy Circle (e.g., 20m):** R_k is large. The denominator increases. K_k approaches 0. The EKF **trusts the IMU**.

FLIGHT BEHAVIOR & TUNING

Understanding this math explains the "feel" of the drone in flight.

SCENARIO A: HIGH CONFIDENCE ($R_K \downarrow$)

- **Environment:** Open atrium, glass roof, clear line of sight to APs.
- **Phone Report:** `horiz_accuracy = 1.5m`.
- **Flight Feel:** "Tight." The drone locks into position. However, if the GPS solution "jumps" by 1 meter due to a reflection, the drone will jerk 1 meter to match it because K_k is high.

SCENARIO B: LOW CONFIDENCE ($R_K \uparrow$)

- **Environment:** Concrete hallway, sparse WiFi.
- **Phone Report:** `horiz_accuracy = 15.0m`.

MAVLink GPS
Indoor Navigation for ArduPilot
AVAILABLE ON GOOGLE PLAY


- **Flight Feel:** "Loose" or "Drifty." The drone resists rapid position corrections. It relies heavily on the IMU integration. It is smoother, but it may drift 1-2 meters before the GPS "pulls" it back.

PRACTICAL "HOW TO": SYNTHETIC TUNING

If you are developing a custom MAVLink application, you can "hack" the flight feel by manipulating the `horiz_accuracy` before sending it to the drone.

1. **The Clamp Strategy:** If you want smoother flight and trust your optical flow more than your network location, mathematically inflate the reported accuracy:

$$\text{ReportedAcc} = \max(\text{RealAcc}, 5.0)$$

   This forces the EKF to treat the position as a "soft suggestion" rather than a hard truth.

2. **The Lock Strategy:** If you are hovering over a charging pad and need precision, you might artificially clamp the accuracy down to 1.0m (dangerously trusting the signal) to force the EKF to converge tightly. *Use with caution.*

THE SAFETY GATE

ArduPilot has a hard limit to prevent EKF blowouts.

- **Threshold:** If `horiz_accuracy > 5.0m` (default), the EKF declares a **GPS Glitch** or prevents Arming (`PreArm: GPS Horiz Accuracy`).
- **Indoor Implication:** If your phone reports 10m accuracy indoors, you might not be able to arm, even if the position looks stable on the map. This is a safety feature to prevent fly-aways when the "Circle of Confusion" is larger than the room you are flying in.

SOURCE CODE REFERENCE

- **Observation Noise:** `libraries/AP_NavEKF3/AP_NavEKF3_PosVelFusion.cpp` - Uses `sq(gpsPosAccuracy)`.
- **Safety Check:** `libraries/AP_NavEKF3/AP_NavEKF3_VehicleStatus.cpp` - The 5.0m check.

DEALING WITH "JITTER"

THE PROBLEM WITH "STILL"

GPS receivers are never truly still. Even if you glue the phone to a table, the calculated position will "wander" by a few centimeters every second, and the velocity report will flicker between `0.0 m/s` and `0.2 m/s`.

EKF SENSITIVITY

The ArduPilot EKF3 is highly sensitive to velocity innovations.

- **Input:** GPS reports "Moving North at 0.2 m/s".
- **Reaction:** EKF tilts the drone South to counter the movement.
- **Reality:** The drone wasn't moving. Now it *is* moving South.
- **Result:** A constant, twitchy "dance" as the drone fights phantom velocity spikes.

THE ZERO-VELOCITY CLAMP

To solve this, MAVLink GPS implements a **Zero-Velocity Clamp** algorithm.

LOGIC

MAVLink GPS
Indoor Navigation for ArduPilot
AVAILABLE ON GOOGLE PLAY


The app monitors the phone's internal accelerometer and the GPS speed.

1. **Condition A:** Is the phone's physical acceleration near zero? (Stationary)
2. **Condition B:** Is the GPS reported speed < 0.3 m/s? (Noise floor)

If both are true, the app assumes the vehicle is stationary (Loitering).

ACTION

It engages the clamp:

1. **Bitmask:** It sets the `GPS_INPUT_IGNORE_FLAG_VEL_HORIZ` bit (Value 8).
2. **Data:** It stops sending velocity vectors (`vn`, `ve` = 0).

THE EKF RESPONSE

When the `VEL_HORIZ` ignore flag is set, the EKF stops listening to the GPS velocity completely.

- It relies purely on **Position Delta** (Change in Lat/Lon over time) to estimate velocity.
- Since position changes much slower than velocity jitter, the estimate smooths out.
- **Flight Behavior:** The drone transitions from "twitchy" to "rock solid."

MAVLink GPS
Indoor Navigation for ArduPilot
AVAILABLE ON GOOGLE PLAY


CHAPTER 6: HARDWARE INTEGRATION

ANDROID DEVICE SELECTION GUIDE

THE "HOLY GRAIL" SPECIFICATION

Selecting the right smartphone for an autonomous drone payload is a multidimensional optimization problem. You are balancing **Weight** (flight time/agility), **Sensor Fidelity** (GNSS/IMU/RTT), and **Cost** (risk of crash).

To function as a primary navigation computer, a device **must** meet these criteria:

1. **Android 9+ (Pie):** Native 802.11mc (RTT) support was introduced here.
2. **Hardware Flag** `android.hardware.wifi.rtt`**:** Must be enabled in the kernel/ROM.
3. **Dual-Frequency GNSS:** Simultaneous L1/L5 tracking (Broadcom BCM47755 or Snapdragon 8 Gen 2+).
4. **Payload Efficiency:** Low mass relative to battery size.

TIER A: THE REFERENCE STANDARDS (GOOGLE PIXEL)

The Pixel series is the "Golden Image" for Android development. Google tests the Wi-Fi RTT API on these devices, ensuring the most compliant driver stacks.

GOOGLE PIXEL 5 (THE KING)

- **Weight: 151g** (The lightest modern device).
- **Body:** Resin-coated aluminum (durable, RF-transparent) vs. heavy glass.
- **RTT:** Excellent 802.11mc support with high bandwidth.
- **GNSS:** L1/L5 Dual Band.
- **Verdict: Primary Recommendation.** The weight-to-performance ratio is unmatched.

GOOGLE PIXEL 6 / 7 / 8 / 9 PRO

- **Weight:** ~197g - 212g.
- **Processor:** Google Tensor (Samsung Exynos Modem).
- **Trade-off:** The ~50g weight penalty reduces flight time by 10-20\% on a standard 5-inch quad. While capable, they add inertia that complicates PID tuning.
- **Use Case:** Heavy-lift platforms where the superior camera or Neural Engine is required for non-navigation tasks.

TIER B: THE LIGHTWEIGHT CHALLENGERS

Devices that offer high performance but come with software caveats.

XIAOMI MI 8

- **Weight:** 175g.
- **History:** The world's first L1/L5 smartphone (Broadcom BCM47755).
- **Warning:** Stock MIUI software aggressively kills background processes, which will cause your drone to failsafe mid-flight. **Must be flashed with a custom ROM (LineageOS)** to be usable.
- **Verdict:** Best budget option (<$100 used) if you are comfortable flashing ROMs.

XIAOMI 13 / 14

MAVLink GPS
Indoor Navigation for ArduPilot
AVAILABLE ON GOOGLE PLAY


- **Chipset:** Snapdragon 8 Gen 2/3 (Qualcomm FastConnect 7800).
- **RTT:** Supports 802.11az (Next Gen).
- **Verdict:** High performance, but high cost ($700+) makes them a risky payload for experimental flight.

TIER C: THE "ULTRA" TRAP (AVOID)

Flagship "Ultra" or "Max" phones often have the best specs on paper but fail as flight computers.

SAMSUNG GALAXY S24 ULTRA

- **Weight: 233g+**
- **Verdict: Unsuitable.** The excessive weight requires a 7-inch+ drone class to carry efficiently. The OIS (Optical Image Stabilization) camera modules are also prone to damage from high-frequency drone vibrations.

CHIPSET NUANCES: QUALCOMM VS. TENSOR

- **Qualcomm Snapdragon (FastConnect):** Generally provides the lowest variance (jitter) in RTT measurements. Dedicated hardware blocks for Wi-Fi sensing.
- **Google Tensor (Exynos Modem):** Early drivers had higher jitter, but Google's software compensation in the Pixel series has largely mitigated this. Reliable for flight.

MOUNTING THE PHONE

THE PHYSICAL LINK

Unlike a standard GPS module which is light and insensitive to vibration, a smartphone is a heavy, complex sensor package. How you mount it directly affects flight performance.

HARD VS. SOFT MOUNTING

For MAVLink GPS, **Hard Mounting** is generally preferred.

WHY HARD MOUNT?

1. **The "Zero Clamp":** The app uses the phone's internal accelerometer to detect if the drone is stationary. If you mount the phone on soft foam or a swaying gimbal, the phone will "feel" motion even when the drone is loitering. This prevents the Zero-Velocity Clamp from engaging, leading to drift.
2. **Latency:** Soft mounts introduce a mechanical phase lag. The phone moves *after* the drone moves. This adds to the existing USB/EKF latency, destabilizing the control loop.

VIBRATION CONCERNS

While ArduPilot's internal IMU needs vibration isolation, modern phones have aggressive internal filtering on their IMUs. They can handle the high-frequency vibe of a rigid mount better than the low-frequency sway of a soft mount.

PLACEMENT

- **Center of Gravity:** Mount the phone as close to the center of the drone as possible.
- **Lever Arm Effect:** If you mount the phone far out on an arm, a simple yaw rotation looks like a lateral translation (X/Y movement) to the GPS. The EKF can compensate (`GPS_POS_X/Y/Z`), but it adds complexity.

MAVLink GPS
Indoor Navigation for ArduPilot
AVAILABLE ON GOOGLE PLAY


ORIENTATION

- **Sky View:** The top half of the phone (where the GPS antenna usually is) must have a clear view of the sky (or ceiling/WiFi APs).
- **Camera:** If using Visual Odometry features (future), the camera needs a clear view.
- **Compass:** Keep the phone away from high-current ESC wires to prevent magnetic interference.

USB-OTG CABLES & POWER

THE "HOST" DILEMMA

USB communication requires one device to be the **Host** (Master) and one to be the **Device** (Slave).

- **Standard:** The Android phone acts as the Host.
- **Implication:** The Host provides 5V power to the Slave.

POWER ARCHITECTURE

1. PHONE-POWERED (STANDARD OTG)

- **Setup:** OTG adapter on phone side.
- **Flow:** Phone Battery → USB Cable → Flight Controller.
- **Risk:** The phone is powering the FC's CPU, receiver, and GPS. This drains the phone battery rapidly.
- **Safety:** Ensure your FC's BEC (Battery Eliminator Circuit) is powerful enough to handle the voltage drop if the phone stops supplying power (unlikely in this direction, but voltage contention can occur).

2. DRONE-POWERED (CHARGE-WHILE-USE)

This is the ideal setup for long endurance.

- **Setup:** You need a "Y-Cable" or a specific OTG hub that supports **Power Injection**.
- **Flow:** Drone LiPo → 5V BEC → USB Cable → Phone.
- **Requirement:** The phone must support "Charge and Data" simultaneously. Most modern USB-C phones do, but older Micro-USB phones often do not.

CABLE HYGIENE

- **Length:** Keep it short (10-15cm). Long loops induce magnetic interference.
- **Shielding:** Use high-quality shielded data cables. Unshielded USB traffic generates noise at 480MHz, which can generate harmonics that interfere with 900MHz telemetry or 433MHz RC links.
- **Ferrites:** A clip-on ferrite bead near the FC connector can reduce RF noise injection into the autopilot.

MAG INTERFERENCE

THE COMPASS PROBLEM

ArduPilot relies on a Magnetometer (Compass) to know which way is North.

- **Outdoors:** The Earth's magnetic field is clean.
- **Indoors:** Steel beams, rebar, and high-current electrical wiring warp the magnetic field.

MAVLink GPS
Indoor Navigation for ArduPilot
AVAILABLE ON GOOGLE PLAY


- **Result:** The "North" vector spins as you fly past a pillar. The EKF becomes confused, rotates the velocity vector, and causes the drone to fly sideways ("Toilet Bowling").

SOLUTION 1: GAUSSIAN SUM FILTER (GSF)

You can fly completely without a compass by letting the EKF "learn" heading from movement.

CONFIGURATION

1. **Disable Compass:** `COMPASS_ENABLE = 0` (or `COMPASS_USE = 0` for all instances).
2. **Enable GSF:** `EK3_SRC1_YAW = 8` (GSF).

THE FLIGHT PROCEDURE

The GSF cannot determine heading while sitting on the ground. It needs **Velocity** to calculate **Course**.

1. **Arm & Takeoff:** You *must* take off in **AltHold** (non-GPS mode). Do not try Loiter yet.
2. **Align:** Fly forward briskly for 5-10 meters.
3. **Lock:** The EKF will observe that the GPS velocity vector aligns with the IMU accelerometer vector. It "snaps" the heading to the correct value.
4. **Loiter:** Once the "EKF Yaw Alignment" message appears (or you see the heading snap on the GCS), you can switch to Loiter.

SOLUTION 2: EXTERNAL YAW (GPS YAW)

If MAVLink GPS provides a valid heading (e.g., from Visual Odometry or a Dual-Antenna setup):

- **Config:** `EK3_SRC1_YAW = 2` (GPS).
- **Benefit:** The heading is valid instantly on the ground. No "alignment flight" needed.

SOURCE CODE REFERENCE

- **Yaw Sources:** `libraries/AP_NavEKF/AP_NavEKF_Source.h` - See `SourceYaw` enum.
- **GSF Logic:** `libraries/AP_NavEKF/EKFGSF_yaw.cpp` - Implementation of the emergency yaw reset.

LIDAR & ALTITUDE HOLD

THE VERTICALITY GAP

While Wi-Fi RTT is excellent for Latitude/Longitude (X/Y), it is physically unreliable for Altitude (Z). In a stadium, ground reflections and poor anchor geometry create "Altitude Jumps" that can crash a drone.

To fly safely indoors, you **must** supplement the Android phone with a dedicated downwards-facing rangefinder.

RECOMMENDED HARDWARE

1. BENEWAKE TF-MINI PLUS

- **Weight:** 12g (Ultra-light).
- **Range:** 12 meters.
- **Interface:** UART / I2C.
- **Verdict:** Best for small to mid-sized indoor drones where every gram counts.

MAVLink GPS
Indoor Navigation for ArduPilot
AVAILABLE ON GOOGLE PLAY


2. GARMIN LIDAR-LITE V3/V4

- **Range:** 40 meters.
- **Reliability:** High performance over varied surfaces (grass, concrete, polished floors).
- **Verdict:** The gold standard for professional stadium flight.

ARDUPILOT INTEGRATION

Once the LiDAR is wired to your Flight Controller's TELEM or I2C port, set the following:

- RNGFND1_TYPE = [Sensor Type]
- RNGFND1_MIN_CM = 20
- RNGFND1_MAX_CM = [Max range of your sensor]
- **EK3_SRC1_POSZ = 2 (RangeFinder)**

By setting the EKF3 source to RangeFinder, the drone will ignore the "noisy" altitude data from the phone's GPS stream and lock its height to the solid physical floor. This prevents the "pumping" effect and ensures stable Loiter performance even if Wi-Fi signal quality fluctuates.

MAVLink GPS
Indoor Navigation for ArduPilot
AVAILABLE ON GOOGLE PLAY

CHAPTER 7: VENUE OPTIMIZATION

THE "SEEDING" RUN

THE "FLOATING AP" PROBLEM

Google's "Network Fusion" relies on a massive, crowdsourced database mapping WiFi BSSIDs (MAC addresses) to physical global coordinates. This works perfectly for permanent infrastructure like Starbucks or an airport.

However, event professionals often deploy **Temporary Infrastructure**—a Pelican case filled with Mesh Routers. When you first plug these in at a new stadium, Google's database either:

1. **Doesn't know them:** They are "Floating," offering no location data.
2. **Remembers the old location:** If you last used them in Las Vegas and now you are in New York, your drone might momentarily teleport across the country ("The Teleport Glitch").

To fly safely, you must perform a **Seeding Run** to "teach" Google the new physical location of your beacons.

THE HEURISTIC: HOW GOOGLE LEARNS

Google's backend builds a confidence map based on correlation.

- **Fact A:** "I have a high-confidence GPS lock (Accuracy < 5m) at Lat/Lon X,Y."
- **Fact B:** "I can see BSSID `aa:bb:cc:dd` with a signal strength of -40dBm."
- **Conclusion:** "BSSID `aa:bb:cc:dd` must be within 10 meters of X,Y."

If 50 people walk past the router with GPS phones, the estimated position of that router converges to a precise point. As a venue operator, you must simulate this "crowd" manually.

THE SEEDING PROTOCOL

PHASE 1: PREPARATION

1. **Phone Settings:**
   - Enable **Location** (High Accuracy).
   - Enable **WiFi Scanning** and **Bluetooth Scanning** in Android Location Services.
   - **Disable WiFi Scan Throttling** in Developer Options (Critical).
2. **The App:** Open Google Maps. Do *not* use MAVLink GPS for seeding; use the native Maps app as it feeds data directly to the GMS backend.

PHASE 2: THE "GOLDEN PATH"

You need to drag the "Blue Dot" from the outdoors (GPS dominant) to the indoors (WiFi dominant) without breaking the chain of trust.

1. **The Satellite Lock:** Start *outside* the venue. Wait until Google Maps shows a small, precise blue circle (GPS Lock).
2. **The Breach:** Walk slowly through a large open door. Keep the phone held high (not in a pocket) to maintain GPS line-of-sight as long as possible.
3. **The Perimeter:** Walk the *inside* perimeter of the venue walls. This is where GPS signal bleeds in, and WiFi signal is strong. This overlapping zone is where the "hand-off" happens.
4. **The Spiral:** Slowly spiral inward toward the center of the venue.
5. **Dwell Time:** Do not run. Stop every 10 meters for 10-15 seconds. Android's scan cycle is not continuous; it scans in bursts. Giving it time ensures it captures multiple "snapshots" of the radio environment.

MAVLink GPS
Indoor Navigation for ArduPilot
AVAILABLE ON GOOGLE PLAY


PHASE 3: VALIDATION

1. **Wait:** It typically takes **24 to 48 hours** for these updates to propagate to the global database.
2. **The Test:** Return to the venue (or turn off mobile data to force offline mode). Open Maps. If the blue dot snaps to your location *instantly* without GPS access, the seeding was successful.
3. **Accuracy Check:** The `horiz_accuracy` circle should be under 20 meters. If it is huge (>50m), the seeding failed or the database is conflicted.

WIFI RTT (802.11MC)

THE TIME-OF-FLIGHT PARADIGM

Standard WiFi positioning uses **RSSI** (Received Signal Strength Indication). It is notoriously susceptible to multipath fading and shadowing—a person standing in front of a router interprets as "distancing" the device.

WiFi RTT (Round Trip Time), defined in the 802.11mc standard, utilizes the **Fine Timing Measurement (FTM)** protocol. It ignores signal power and instead measures the precise nanosecond duration it takes for a radio packet to bounce between an Initiator (Smartphone) and a Responder (Access Point).

BANDWIDTH AND RESOLUTION

The accuracy of the ranging result is physically limited by the bandwidth of the WiFi channel:

- **20 MHz (2.4GHz):** Coarse resolution, errors of several meters.
- **80/160 MHz (5GHz/6GHz):** Sharper correlation peaks, allowing sub-meter resolution and the ability to distinguish the "first arriving path" from late reflections.

ENTERPRISE LOGIC: OPEN LOCATE

For stadium-scale deployments, enterprise hardware offers a critical software advantage:

- **Self-Location:** APs like the **Aruba 500 Series** can self-locate via built-in GPS and broadcast their precise Latitude/Longitude/Altitude coordinates directly in the FTM frames.
- **Dynamic Trilateration:** This allows the drone to enter a venue and begin positioning immediately without needing a pre-loaded database of anchor locations.

VALIDATED HARDWARE

CONSUMER / REFERENCE

- **Google Wifi / Nest Wifi:** The primary reference hardware for Android. Guaranteed to work as Google uses them for internal testing.
- **Linksys Velop:** Confirmed 802.11mc support in most modern firmware versions.

ENTERPRISE (RECOMMENDED)

- **Aruba 505 / 515:** Supports the "Open Locate" protocol.
- **Cisco Catalyst 91xx:** High sensitivity, but requires a controller-based config to enable Responder mode.

SOURCE CODE REFERENCE

- **Android RTT Manager:** `android/net/wifi/rtt/WifiRttManager.java` - The API for initiating ranging requests and handling the `onRangingResults()` callback.

MAVLink GPS
Indoor Navigation for ArduPilot
AVAILABLE ON GOOGLE PLAY


BEACON DENSITY & GEOMETRY

THE GEOMETRY OF PRECISION (GDOP)

Accuracy is driven by **Angle of Arrival**, mathematically described as **Geometric Dilution of Precision (GDOP)**. If your APs are too close to each other or arranged in a straight line, the intersection of their range "circles" creates a massive area of uncertainty.

The Fix: Always stagger your APs in a **Diamond** or **Staggered Grid** configuration.

THE Z-AXIS INSTABILITY TRAP

Most indoor navigation failures occur in the vertical domain. This is due to two physical factors:

1. HIGH VDOP (THE PANCAKE EFFECT)

If all your Access Points are placed at the same height (e.g., on tripods at field level), the vertical separation is zero. The Vertical Dilution of Precision (VDOP) is essentially infinite. The drone will know its latitude/longitude, but its altitude will jump wildly between 1m and 15m.

2. GROUND REFLECTIONS (MULTIPATH)

Radio waves reflect off concrete floors and stadium seating.

- **Constructive/Destructive Interference:** The direct wave and the reflected wave can cancel each other out at specific altitudes (nulls).
- **The Reaction:** If you use BLE beacons for altitude, the drone might fly into a signal null, interpret it as a sudden 10-meter climb, and cut motor power to descend—resulting in a hard crash.

DEPLOYMENT STRATEGY

To ensure 3D stability, you must create **Vertical Diversity**:

1. **50\% Low:** Mount anchors at 2m (tripods/hoardings).
2. **50\% High:** Mount anchors at 8m-12m (stands/rigging).
3. **Avoid 90-degree Zenith:** Do not fly directly underneath an AP. RTT accuracy is best when the angle of incidence is shallow.

LAYOUT DENSITY

- **Optimal Spacing:** 20-30 meters.
- **Stadium Coverage:** A standard pitch requires 8 to 12 APs for robust coverage. A 4-corner setup is insufficient as it leaves a massive dead zone in the center.

BLUETOOTH (BLE): THE SECONDARY CONSTRAINT

While BLE beacons (iBeacon/Eddystone) are too jumpy for primary flight control (especially on the Z-axis), they are extremely valuable as **coarse logical constraints**.

1. ZONE DETECTION

MAVLink GPS
Indoor Navigation for ArduPilot
AVAILABLE ON GOOGLE PLAY


Stick a BLE beacon to a wall or near an expensive scoreboard. Your app can use the signal strength to define a **"Hard No-Fly Zone."** Even if the Wi-Fi RTT fix drifts, a high RSSI from a specific BLE ID can trigger an immediate software-level override to prevent a collision.

2. WAYPOINT PASSING (GATE CORRECTION)

In a racing or survey scenario, place a beacon on each gate or corner. When the drone passes through a gate, the app "hears" the beacon and can reset its accumulated drift error for that specific coordinate.

The Rule: Use BLE for **"Am I near here?"** logic. Never use it for **"How high am I?"** flight control.

MAVLink GPS
Indoor Navigation for ArduPilot
AVAILABLE ON GOOGLE PLAY


CHAPTER 8: TROUBLESHOOTING 'THE ERROR CODES'

"GPS GLITCH" / "BAD VELOCITY"

WHAT IT MEANS

When your HUD flashes **GPS GLITCH**, it does not necessarily mean the GPS signal is lost. It means the GPS is **lying** (or the EKF thinks it is).

The Flight Controller constantly integrates the IMU (Accelerometers) to predict where the drone *should* be. If the GPS reports a position that is statistically impossible based on the physics of the last few seconds, the EKF declares a Glitch.

THE TRIGGER MECHANISM

The EKF calculates two metrics:

- `posTestRatio`: Position Error / Gate Size
- `velTestRatio`: Velocity Error / Gate Size

The **Gate Size** is typically 5 Standard Deviations (EK3_POS_I_GATE = 500).

- **Ratio < 1.0:** Normal flight. GPS and IMU agree within reason.
- **Ratio > 1.0:** Glitch. The error is larger than 5x the expected uncertainty.

COMMON CAUSES IN VIRTUAL GPS

1. LATENCY MISMATCH

If the `time_usec` timestamp is wrong, the "Rewind" mechanism fails.

- **Scenario:** You push the stick forward. The drone accelerates.
- **IMU:** "Moving forward at T=0."
- **GPS:** "Still at Start at T=0." (Because the packet was delayed but not timestamped correctly).
- **Result:** The EKF sees a massive disagreement between the IMU prediction (Moved) and GPS report (Stationary). **GPS GLITCH.**

2. THE MULTIPATH TELEPORT

In an urban canyon, a reflection might cause the position solution to jump 10 meters instantly.

- **Physics:** A drone cannot move 10 meters in 0.1 seconds (100 m/s velocity).
- **Reaction:** The EKF rejects the jump as physically impossible. It triggers a glitch and ignores the GPS until it "settles."

RECOVERY

To clear the glitch, the GPS and IMU must agree (Ratio < 1.0) for **10 continuous seconds**. This prevents the drone from latching onto a bad signal that is just fluctuating wildly.

"PREARM: GPS VERTICAL POS"

MAVLink GPS
Indoor Navigation for ArduPilot
AVAILABLE ON GOOGLE PLAY


THE ERROR

You are ready to fly. You arm the drone. The HUD rejects the command with:

> **PreArm: GPS Vertical Pos**

WHAT IT MEANS

The Extended Kalman Filter (EKF) has detected that the vertical position reported by the GPS is inconsistent or unstable compared to the Barometer.

THE SENSOR CONFLICT

1. **Barometer:** Measures air pressure. Very stable in the short term. Defines "Altitude = 0" at boot.
2. **GPS (Phone):** Measures triangulation from satellites/towers. Noisy in the Z-axis.

If the GPS reports an altitude that drifts up and down by 10 meters while the Barometer says "We haven't moved," the EKF flags the GPS as unreliable.

COMMON CAUSES

1. THE GEOID OFFSET (N-VALUE)

If your phone sends raw WGS84 Ellipsoid altitude, but ArduPilot expects MSL (Geoid) altitude, there is a constant offset (e.g., -35 meters).

- **Symptom:** The EKF sees a constant, massive discrepancy.
- **Fix:** Ensure the MAVLink GPS app has "Geoid Correction" enabled (it is by default).

2. VERTICAL WANDER

Phone GPS altitude is notoriously poor, often having a variance of +/- 20 meters indoors.

- **The Check:** ArduPilot checks `vert_accuracy`. If the phone reports `vert_accuracy > 5m` (or if the EKF calculates the innovation is high), it triggers this PreArm error.

SOLUTIONS

1. **Wait:** Sometimes the EKF just needs time to converge. Let the drone sit for 30-60 seconds.
2. **Reposition:** Move the phone to a window or an area with better WiFi visibility to improve the Z-axis fix.
3. **Disable Vertical Fusion:** If you only care about X/Y position hold and trust the Barometer for altitude:
    - Set EK3_GPS_CHECK to ignore vertical variance (Advanced).
    - Ideally, just let the EKF do its job. If it says the GPS is bad, it usually is.

"NO GPS FIX"

SYMPTOM

The MAVLink GPS app is running, the "Radar" shows a fix, but Mission Planner (or the HUD) shows **No GPS**.

TROUBLESHOOTING STEPS

1. CHECK THE DRIVER (GPS_TYPE)

MAVLink GPS
Indoor Navigation for ArduPilot
AVAILABLE ON GOOGLE PLAY


ArduPilot will not listen to MAVLink GPS data unless you tell it to.

- **Parameter:** GPS_TYPE (or GPS1_TYPE).
- **Required Value: 14** (MAVLink).
- *Note: You must reboot the flight controller after changing this.*

2. CHECK THE BAUD RATE

The USB connection (SERIAL0) requires a specific baud rate to match the app.

- **Parameter:** SERIAL0_BAUD.
- **Required Value: 115** (115200).
- **Parameter:** SERIAL0_PROTOCOL.
- **Required Value: 2** (MAVLink 2).

3. CHECK ANDROID PERMISSIONS

Android requires explicit user permission to access the USB device.

- **Action:** When you plug in the USB cable, a popup asks "Allow MAVLink GPS to access...?"
- **Requirement:** You **MUST** click OK. If you clicked Cancel, unplug and replug the cable.

4. CHECK THE "TRAFFIC"

Does the app see the drone?

- **App UI:** Look at the top bar. Does it say "Connected: ArduPilot"?
- **Heartbeat:** If the app says "Waiting for Heartbeat...", the TX line (Phone → Drone) might be working, but the RX line (Drone → Phone) is broken. Check your cable.

MAVLink GPS
Indoor Navigation for ArduPilot
AVAILABLE ON GOOGLE PLAY