Multirotor Drone Flight Control and Systems

1 2.1.1: How a Multirotor Drone Flies and Achieves 6 Degrees of Freedom

Understanding the Magic Behind the Hovering Beast

Let's take an example of quadcopter it's a beautiful dance of physics, math, and real-time control. Here's what I've come to understand — explained in the way I wished someone had explained it to me when I started.

Lift: Fighting Gravity with Rotors

Each of the four rotors is connected to a motor that spins a propeller. When a rotor spins, it accelerates air downward — creating an upward reactive force (thrust), thanks to Newton's 3rd law.

$$\text{Thrust} = \text{mass flow rate} \times \text{change in velocity}$$

When all four rotors spin at equal speed, the drone lifts vertically — a motion called **heave** (along the Z-axis). Increase thrust ⇒ go up. Decrease thrust ⇒ descend.

Yaw: The Subtle Game of Torque

Now here's the part that fascinated me the most — **yaw**, or rotation about the vertical (Z) axis.

A drone typically has:

- Rotors 1 and 3 spinning Clockwise (CW)
- Rotors 2 and 4 spinning Counter-Clockwise (CCW)

In hover, all torques cancels out:

$$\tau_{\text{CW}} + \tau_{\text{CCW}} = 0 \Rightarrow \text{No rotation}$$

But suppose we increase the speed of CCW rotors (2 and 4), and decrease the speed of CW rotors (1 and 3). What happens?

The net torque becomes:

$$\tau_{\text{net}} = \tau_{\text{CCW}} - \tau_{\text{CW}} > 0 \Rightarrow \text{The drone yaws to the right (clockwise)}$$

At a basic physics level, this is Newton's 3rd Law again. If the rotors are applying more angular momentum in one direction, the body must rotate in the opposite direction to conserve angular momentum:

$$\Delta L_{\text{rotors}} + \Delta L_{\text{body}} = 0$$

That's why a drone yaws!

Roll and Pitch: Tipping to Translate

To move forward, backward, or sideways, the drone tilts its body — this redirects some of the lift vector into horizontal motion.

- **Pitch (Y-axis rotation):** Increase rear rotor thrust and decrease front rotor thrust ⇒ the drone tilts forward.
- **Roll (X-axis rotation):** Increase left rotor thrust and decrease right rotor thrust ⇒ the drone tilts to the right.

Why does this work?
Because once the drone tilts, the thrust vector has a horizontal component:

$$\vec{T}_{\text{tilted}} = T \begin{bmatrix} \sin(\theta) \\ 0 \\ \cos(\theta) \end{bmatrix} \Rightarrow \text{Horizontal force} = T \cdot \sin(\theta)$$

This results in:

- **Surge (forward/back motion)** from pitch
- **Sway (left/right motion)** from roll

The Full 6 Degrees of Freedom — Like a Bird

With that, here are the 6 Degrees of Freedom (DoF) and how they're achieved in a multirotor drone:

Final Thoughts

Before writing this, I thought drones just "spin their fans and fly." But now I see — every bit of their motion is a delicate manipulation of physics, from torque balancing to thrust vectoring. It's like flying a spacecraft that's constantly trying to fall out of the sky — and yet remains steady because of brilliant control logic and real-time motor mixing.

To me, that's nothing short of magical engineering.

DOF	Motion	Achieved By
Heave (Z)	Up/Down	Increase/decrease total rotor thrust
Surge (X)	Forward/Backward	Tilt via pitch (front/rear rotor thrust difference)
Sway (Y)	Left/Right	Tilt via roll (left/right rotor thrust difference)
Roll (X-axis rotation)	Tilt left/right	Torque from differential side thrust
Pitch (Y-axis rotation)	Tilt forward/backward	Torque from front/back thrust difference
Yaw (Z-axis rotation)	Rotate left/right	Net torque from opposite-spinning rotors

Table 1: How a quadcopter achieves all 6 Degrees of Freedom

2 2.1.3: Autorotation in Gyrocopters and Understanding MTOW

Autorotation — The Most Beautiful Emergency Trick in Aviation

While researching this, I found autorotation to be not just a clever aerodynamic phenomenon — but a real life-saver.

A gyrocopter (or autogyro) doesn't power its rotor directly like a helicopter. Instead, the rotor spins freely in the wind — that's **autorotation**. Initially, this felt counterintuitive: "how can something unpowered lift you off the ground?" But the physics checks out beautifully.

How Autorotation Works

In flight, the forward motion of the gyrocopter forces air to flow *upward* through the rotor disc (from below). This upward airflow hits the rotor blades at a slight angle (called the **angle of attack**), which causes them to spin — much like how a windmill works.

The rotor system then behaves like a spinning wing:

$$\text{Lift} = \frac{1}{2}\rho V^2 S C_L$$

Except here, the relative wind comes from below, and the spinning blade segments generate lift as they move through it.

Important: The engine doesn't spin the rotor! The engine only drives a rear-facing propeller, giving the craft forward motion. The rotor just autorotates.

What Happens During Engine Failure?

Here's where it gets life-saving.

If the engine fails in a helicopter, the pilot must enter autorotation quickly. He/she reduces the collective pitch, letting the air spin the rotor upward. The stored rotational energy in the rotor is then used to slow the descent and perform a controlled landing.

In a gyrocopter, this state is already the norm. The rotor is always in autorotation, so if the engine fails, the craft can simply glide downward — as long as forward speed is maintained.

The Physics Behind Autorotation

Imagine a rotor blade divided into segments along its span. Each segment may either:

- **Accelerate the rotor** (producing torque),
- **Decelerate the rotor** (absorbing torque), or
- **Neither** (the equilibrium zone).

The total aerodynamic torque over all segments determines if the rotor speeds up or slows down. In autorotation, the rotor settles into a speed where the energy input from air balances the aerodynamic drag of spinning.

Key condition:

$$\text{Power in (from airflow)} = \text{Power lost (to drag)}$$

MTOW — Maximum Takeoff Weight

While studying aircraft specs, MTOW kept coming up. At first, I thought it was just "how heavy the aircraft is allowed to be" — but it's more nuanced.

MTOW (Maximum Takeoff Weight) is:

- The **maximum legally and structurally safe weight** the aircraft can take off with.
- Includes: aircraft empty weight + fuel + pilot/passenger(s) + payload.
- Determined by: structural strength, engine power, rotor efficiency, and flight dynamics.

For gyrocopters, MTOW affects:

- The amount of thrust required for takeoff (especially if rolling on wheels).
- The rotor's ability to generate enough lift while autorotating.
- Glide/descent rate in engine-off scenarios.

If the actual weight exceeds MTOW:

- The craft may not achieve safe takeoff speed within the runway.
- In flight, the rotor may not sustain autorotation efficiently.
- Structural failure may occur in stress-limited parts.

A Personal Insight

What struck me was this: gyrocopters are always gliding — even when flying forward. The only thing keeping them in the air is the dance between air and rotor, not a gearbox or engine. It made me appreciate how nature (and physics) often gives us backup plans — if we design for them.

Autorotation is like a trust-fall where physics catches you.

TL;DR: Gyrocopters fly because the rotor auto-spins due to incoming airflow. MTOW is the ceiling of what the aircraft can safely carry — push beyond it, and you risk failing both in takeoff and in the safety margin of autorotation.

3 2.1.5: What is Counteracting Torque in Drones and Gyrocopters? How to Minimize It?

Torque — The Unseen Twist That Tries to Flip Your Craft

When I first got into drone mechanics, I knew that motors spin propellers to generate thrust. But I didn't initially realize that every spinning propeller also creates **rotational torque** — trying to twist the whole drone body in the *opposite direction.*

And here lies the problem: if we don't actively balance these torques, the drone or gyrocopter won't just lift — it'll start spinning uncontrollably.

Newton's Third Law Strikes Again

Let's say a motor spins a rotor clockwise. According to Newton's Third Law:

$$\text{Rotor exerts a clockwise torque on air} \Rightarrow]$$

Air exerts an equal and opposite torque (counter-clockwise) on drone body

This is the source of what we call **reaction torque**. If unbalanced, it causes the entire drone or helicopter to yaw or spin involuntarily.

In Drones — Balanced by Opposite-Spinning Pairs

In multirotor drones (like quadcopters), counteracting torque is cleverly solved by using:

- Two rotors spinning clockwise (CW)
- Two spinning counter-clockwise (CCW)

This symmetric setup cancels out the net torque:

$$\tau_{\text{net}} = \tau_{\text{CW}} + \tau_{\text{CCW}} = 0$$

What if you want to yaw? Then the flight controller intentionally *creates a small imbalance* by speeding up one pair and slowing the other (as explained in section 2.1.1).

This means that counteracting torque isn't just a problem — it's a tool used for controlled rotation.

In Gyrocopters — It's a Bit Trickier

Gyrocopters usually have:

- An unpowered, freely-spinning main rotor (autorotating)
- A **powered propeller** (usually at the back — like a pusher aircraft)

In this setup:

- The rotor creates minimal torque (since it's unpowered)
- The engine-driven propeller, however, *does* generate torque

That torque causes the fuselage to twist — just like in a traditional airplane.
How do gyrocopters counteract this?

1. **Vertical Stabilizer or Rudder:** Helps produce aerodynamic yaw force to oppose engine torque.
2. **Engine Placement:** Slight lateral offset of the propeller axis can generate a reaction that neutralizes torque.
3. **Tail Design:** Horizontal and vertical surfaces in the prop wash help resist uncommanded yawing.

How to Minimize Counteracting Torque — Design Principles

Here's what I learned that designers do to reduce torque-related issues:

- **Rotor Pairing (in drones):** Use an even number of motors with alternating spin directions (CW, CCW).
- **Coaxial Designs:** In some drones and helicopters, a second rotor is placed above or below the first, spinning in the opposite direction — cancelling torque directly.
- **Contra-Rotating Propellers:** Used in some advanced systems — one shaft spins CW, the other CCW.
- **Balanced Motor Thrust:** In control code, motor mixing ensures yaw torque remains zero when hovering.
- **Symmetrical Frame Layout:** Minimizes off-axis torque effects.

Why It Matters So Much

If torque isn't handled, your aircraft:

- Spins when it's not supposed to
- Becomes harder to control or even unstable
- Wastes power fighting itself

In other words — *torque management isn't optional, it's the backbone of control.*

One Final Thought

When I first flew a drone, I noticed it had an elegant smoothness in the air — it didn't just move, it *balanced itself constantly.* That balance comes from precisely managing torques — especially yaw-related ones. And the more I studied it, the more I appreciated how these machines handle such chaotic forces so gracefully.

It's like riding a spinning top — and always staying upright.

4 2.5.4: What is Sensor Fusion? Give an Example

When One Sensor Isn't Enough

Early in my UAV learning journey, I thought each sensor worked like a solo hero — GPS tells you where you are, IMU tells you how you're rotating, barometer gives altitude. But I quickly realized: individually, these sensors are imperfect. They have noise, drift, delays, and blind spots.

That's where **sensor fusion** comes in.

Definition

Sensor fusion is the process of combining data from multiple sensors to get a more accurate, reliable, and complete picture of a system's state (e.g., position, velocity, orientation) than any one sensor can provide on its own.

It's like getting opinions from different friends — one may be slow but accurate, another may be quick but sometimes wrong. Together, their insights are stronger.

Why Sensor Fusion is Important in UAVs

UAVs are fast-moving systems operating in 3D space. To maintain stable flight and navigate effectively, they must:

- Know their exact orientation (roll, pitch, yaw)

- Track their position over time
- React quickly to sudden changes (like gusts of wind)

No single sensor can provide all of this with sufficient accuracy and responsiveness.

Example: IMU + GPS + Barometer Fusion

Let's consider a common case: fusing data from:

- **IMU (Inertial Measurement Unit)** — fast updates (100–1000 Hz), but drifts over time.
- **GPS** — accurate position, but slow updates (1–10 Hz) and vulnerable to signal loss.
- **Barometer** — provides relative altitude with high resolution, but affected by air pressure changes.

What happens when we fuse them?

- The IMU provides rapid motion estimates — like dead-reckoning.
- GPS corrects long-term drift by anchoring position estimates to real-world coordinates.
- The barometer fine-tunes vertical accuracy, especially for hover stability.

Together, they form a closed-loop system that's accurate, smooth, and robust.

Behind the Scenes: Filters and Estimators

Sensor fusion isn't just about adding values — it's done using mathematical tools like:

- **Complementary Filter** — a simple method that blends fast/short-term sensors (IMU) with slow/long-term ones (GPS/baro).
- **Kalman Filter / Extended Kalman Filter (EKF)** — a more advanced statistical estimator that predicts, updates, and optimally weighs sensor inputs based on their noise and confidence levels.

EKF Example:

- Uses IMU data to *predict* the next state (position, orientation).
- When new GPS/barometer data arrives, it *corrects* the prediction using observed measurements.
- The result is an accurate estimate, even when sensors have different rates or occasional errors.

Analogy That Helped Me Understand

Imagine trying to walk in the dark:

- Your **IMU** is like your sense of balance — it tells you how you're moving right now.
- Your **GPS** is like checking your phone map — slow, but tells you where you are overall.
- Your **Barometer** is like touching the wall — good for vertical guidance.

Alone, each sense can fail. But when combined, you navigate confidently.

Conclusion

Sensor fusion is what makes drones "smart." It lets them not just react, but **understand** their motion with precision. To me, it's one of the most elegant examples of teamwork in engineering — different imperfect pieces coming together to create a whole that's stronger than any individual part.

5 3.1: Designing the Flight Computer (Simplified + Pseudocode)

What This Question Means

This question asks me to design the **brain of the drone** — the part that reads all sensor inputs, processes them, and makes control decisions to stabilize and guide the aircraft.

In other words, it's like building a mini autopilot that can:

- Know the drone's orientation (tilt, rotation)
- Track its altitude and position
- Communicate with the ground station
- Make decisions based on sensor inputs

My Component Choices and Why I Picked Them

- **Microcontroller: Teensy 4.0** A powerful and fast controller (600 MHz) that supports real-time data reading and control loops. It has many I/O pins for sensors and actuators.
- **IMU (Inertial Measurement Unit): MPU-9250** Combines 3-axis gyroscope, accelerometer, and magnetometer. Helps the drone sense its orientation (roll, pitch, yaw).

- **Barometer: BMP280** Measures air pressure, which can be converted to altitude. Useful for maintaining a steady hover.
- **GPS Module: u-blox NEO-6M** Gives position (latitude, longitude) and speed. Essential for outdoor navigation.
- **Power Regulators:**
  - **LM2596 Buck Converter** — Converts 11.1V (from LiPo) to 5V.
  - **AMS1117-3.3V LDO** — Converts 5V to 3.3V for noise-sensitive sensors.

Estimated Power Consumption (Approximate)

Component	Current Draw
Teensy 4.0	100 mA
IMU (MPU-9250)	10 mA
Barometer (BMP280)	5 mA
GPS Module	45 mA
Spare/Other	40 mA
Total	**200 mA @ 5V**

Power = $5\,\text{V} \times 0.2\,\text{A} = 1.0\,\text{W}$ (With margin: around 1.2W)

How Everything Connects

- **Battery (11.1V)** powers the drone.
- Voltage is stepped down using the **buck converter** to get clean 5V.
- 5V powers the Teensy, GPS, and barometer.
- 3.3V LDO powers the IMU.
- The microcontroller reads sensors and runs control code to keep the drone stable.

Pseudocode: How the Flight Computer Works

Here's a simplified view of what the main control loop inside the flight computer might look like:

```
Initialize sensors (IMU, GPS, Barometer)
Initialize PWM outputs to motors
Set timer to run loop every 5 milliseconds

Loop:
    Read IMU data (acceleration, gyro)
    Read barometer for altitude
```

```
    Read GPS for position

    Estimate orientation (pitch, roll, yaw)
    Estimate position and speed

    Compute errors between desired and current orientation
    Run PID controllers for roll, pitch, yaw

    Compute motor outputs based on PID results
    Send PWM signals to ESCs to adjust motor speeds

    Wait until next loop cycle
End Loop
```

In Simple Words

I learned that designing a flight computer is not just about picking parts — it's about making sure they all talk to each other at the right speed, with the right power, and with minimal noise.

It's a mini nervous system, and when it all works, the drone feels alive.

6 3.2: Comparing Telemetry Options for 5 km and 20 km Range

What Is This Question About?

This question asks: If we want to communicate with the drone from the ground (called telemetry), which wireless system should we use for different distances — like 5 km or even 20 km?

This means looking at radio frequency (RF) modules that can send and receive data between the drone and a ground station.

To make a good choice, we need to understand:

- How far the signal can travel (range)
- How much power it needs
- How much data it can send (data rate)
- How much interference or signal loss occurs

Common Telemetry Options

Here are the most common options used in UAV telemetry:

- **LoRa (915 MHz)** – Low Power, Long Range

- **433 MHz ISM band** – Longer range but lower data rate
- **868 MHz ISM band** – Used in Europe, similar to 915 MHz
- **2.4 GHz Wi-Fi or Telemetry** – Fast, but shorter range
- **VHF (118–136 MHz, aviation band)** – Used in real aircraft for voice/-data

How to Compare Them

To compare these systems for 5 km and 20 km, I looked at a basic physics formula:

$$\text{Free-Space Path Loss (FSPL)} = 20\log_{10}(d) + 20\log_{10}(f) + 92.45$$

Where:

- d = distance (in kilometers)
- f = frequency (in GHz)

Example FSPL Calculations:

At 5 km:

- 433 MHz → FSPL 99 dB
- 868 MHz → FSPL 105 dB
- 2.4 GHz → FSPL 114 dB
- VHF (130 MHz) → FSPL 89 dB

At 20 km (4x the distance):

- Increase FSPL by 12 dB (because $20\log_{10}(4) \approx 12$)
- So 433 MHz → 111 dB, and 2.4 GHz → 126 dB

What These Numbers Mean

The higher the FSPL, the weaker the received signal will be. So lower frequency signals (like 433 MHz or VHF) can go much further without losing strength.

LoRa (915 MHz) is special — even with low power (like 100 mW), it can reach 10–15 km with a good antenna because it uses spread-spectrum technology.

Option	Data Rate	Range (Line of Sight)	Power Needed
LoRa (915 MHz)	Low (0.3–37 kbps)	Up to 15–20 km	Low
433 MHz ISM	Low–Medium	10–20+ km	Medium
868 MHz ISM	Medium	5–10 km	Medium
2.4 GHz	High (Wi-Fi speeds)	500 m – 2 km	High
VHF (130 MHz)	Very Low (mainly voice)	50+ km	High (licensed use)

Table 2: Comparison of telemetry links

Comparison Table

My Suggestions

- **For 5 km:** 868 MHz or 915 MHz LoRa is a great choice. It's power-efficient, reliable, and can handle slow sensor telemetry.
- **For 20 km:** Go with 433 MHz or high-quality LoRa modules with directional antennas. Avoid 2.4 GHz — too much signal loss.
- **For long distances (beyond 20 km):** VHF/UHF aviation-grade radios or satellite systems are best — but they require licenses and are expensive.

What I Learned

Before this, I didn't know signals could get so weak just by traveling through air! Now I understand why:

- Low-frequency signals travel farther
- High-frequency signals need more power and better antennas
- It's not just about "range", but about matching data rate, power, and interference

Telemetry is like the "voice" of the drone — and choosing the right one is the difference between a successful mission and a lost aircraft.

7 3.4: PID Control Using Potentiometer and Servo Motor (Microcontroller)

Understanding the Setup

This question asked me to design a basic control system using:

- A **potentiometer** — which acts like a user-controlled dial (input).
- A **servo motor** — which physically moves to match the input (output).

- A **microcontroller** — which reads the potentiometer and drives the servo using a **PID controller**.

To me, this was a mini version of what real autopilots do — they read a sensor (pot) and adjust an actuator (servo) to match a desired value, smoothly and accurately.

What Is PID Control?

PID stands for:

- **P = Proportional:** Reacts to the current error
- **I = Integral:** Reacts to accumulated past error (eliminates bias)
- **D = Derivative:** Reacts to how fast the error is changing (adds damping)

The PID output is calculated as:

$$u(t) = K_p \cdot e(t) + K_i \cdot \int_0^t e(\tau)d\tau + K_d \cdot \frac{de(t)}{dt}$$

Where:

- $e(t)$ = error = desired position (from pot) – current position (servo)
- K_p, K_i, K_d are the tuning constants

How the System Works (Overview)

1. The potentiometer is connected to an analog input pin of the microcontroller.
2. The servo is connected to a PWM output pin.
3. The microcontroller reads the pot (desired position), compares it to actual servo position (if feedback is available), calculates the error, and applies PID logic.
4. It outputs a corrected PWM signal to move the servo accordingly.

Circuit Description

- Potentiometer center pin → Analog pin A0
- Potentiometer outer pins → 5V and GND
- Servo signal pin → PWM pin (e.g., D9)
- Servo VCC → 5V (separate if needed), GND → GND

Pseudocode: PID Loop for Servo Control

```
Initialize:
    Set Kp, Ki, Kd
    Set servoPin and potPin
    lastError = 0
    integral = 0

Loop (every 10 ms):
    // 1. Read desired position from potentiometer
    potValue = analogRead(potPin)          // 0 to 1023
    setpoint = map(potValue, 0, 1023, 0, 180) // convert to degrees

    // 2. Read actual servo position (if feedback available)
    current = readServoAngle()             // or assume ideal movement

    // 3. Compute error
    error = setpoint - current

    // 4. PID calculations
    integral += error * dt
    derivative = (error - lastError) / dt
    output = Kp * error + Ki * integral + Kd * derivative

    // 5. Clamp output to servo limits (0 to 180 degrees)
    output = constrain(output, 0, 180)

    // 6. Send PWM signal to servo
    servo.write(output)

    lastError = error
    delay(dt)
```

Explanation of Each PID Term

- **Proportional (P):** Makes the servo move fast when the error is large. But alone, it may overshoot.
- **Integral (I):** Slowly builds up correction if the servo is always a little off-target.
- **Derivative (D):** Predicts and slows down motion to avoid overshooting — adds "braking".

What I Learned

This project really helped me "feel" the value of each PID term. With just P, the motion was fast but unstable. Adding D smoothed it. Adding I helped eliminate drift.

This tiny setup gave me hands-on intuition about how flight controllers stabilize drones in real-time — and how powerful simple control logic can be.

8 3.5.1: Achieving a 1 ms PID Loop Using an IMU and Interrupts

What the Question Asks

This question asks how to run a **PID control loop every 1 millisecond (1 ms)** using:

- An **IMU sensor** (to get orientation data)
- **Interrupts** (a way to trigger the loop precisely and quickly)

To me, this is like trying to make a microcontroller do the same job as a drone's autopilot — and do it very fast, 1000 times per second.

—

Why 1 ms?

- In drones, fast changes in tilt or wind can destabilize flight in milliseconds. - Running the PID loop every 1 ms ensures the drone can react quickly to keep itself stable. - This is especially important for small, agile drones that need high-frequency updates.

—

How Interrupts Help

Normally, you use 'delay()' or 'millis()' in loops to control timing. But these can be inaccurate and slow.

Interrupts are special functions that are triggered automatically when: - A hardware timer reaches a set value - A signal (like from an IMU) says "new data is ready"

This means your code reacts immediately — ideal for precise timing like 1ms control loops.

—

Steps to Implement a 1 ms PID Loop

1. **Initialize the IMU** (like MPU-6050 or MPU-9250) to generate a Data Ready (DRDY) interrupt every time it has new data — ideally at 1000 Hz.
2. **Attach an interrupt handler** to the IMU's DRDY pin using 'attachInterrupt()'.
3. In the interrupt function, read the IMU data and run the PID loop.
4. Use a flag or buffer system to avoid long code inside the interrupt (keep it short).

—

Pseudocode (Simplified)

from chatgpt

```
// Global variables
volatile bool imuReady = false;
float Kp, Ki, Kd;
float error, lastError, integral;

void setup() {
    initializeIMU();  // Setup IMU for 1000 Hz update rate
    attachInterrupt(digitalPinToInterrupt(DRDY_pin), imuInterrupt, RISING);
}

void loop() {
    if (imuReady) {
        imuReady = false;  // Reset the flag

        // 1. Read IMU data
        angle = getIMUOrientation();

        // 2. Compute error
        error = setpoint - angle;

        // 3. PID calculations
        integral += error * dt;
        derivative = (error - lastError) / dt;
        output = Kp * error + Ki * integral + Kd * derivative;

        // 4. Apply to motor or actuator
        setMotorPWM(output);
```

```
        lastError = error;
    }
}

// This gets called exactly every 1 ms by IMU interrupt
void imuInterrupt() {
    imuReady = true;
}
```

—

Timing Notes

- **dt (time between loops)** is 0.001 seconds (1 ms) - Make sure the IMU sampling rate is also set to 1000 Hz - Avoid printing in interrupts or PID loop — it slows things down

—

What I Learned

This question helped me understand how real-time systems work. At first, I didn't know why interrupts were important — now I see they are crucial for precision and speed.

To build a drone or robot that responds instantly, you need: - Fast loops - Reliable sensor readings - Efficient math (like PID)

And you need it all to run on time — *every single millisecond*.

9 3.5.2: Why Not Use delay() in a PID Loop?

What the Question Means

This question asks why using `delay()` — a common function in Arduino — is a bad practice when running real-time control systems like PID loops for drones.

Simple Answer: Because delay() wastes time

When we call `delay(10)`, the microcontroller:

- Stops doing everything else
- Waits blindly for 10 milliseconds
- Misses sensor updates or critical events

Why This Is a Problem for PID Control

1. **PID loops must run on time.** A drone may need to run its PID loop every 1 ms to stay stable. `delay()` adds unpredictable delay and breaks this timing.
2. **It blocks sensor reading.** While waiting inside `delay()`, the microcontroller cannot check the IMU or read GPS.
3. **It prevents multitasking.** Drones must do many things at once — control motors, monitor battery, transmit telemetry. `delay()` prevents that.

What to Use Instead

- **millis() or micros():** These functions return current time, so you can calculate when to run the next loop without stopping the system.
- **Hardware Timers and Interrupts:** You can set up a timer interrupt to run the PID loop every 1 ms accurately, without blocking the rest of the code.

Final Thought

Using `delay()` is okay for blinking LEDs or basic projects — but in drones and robotics, where fast and accurate responses matter, it causes more harm than good.

"A delay in code means a delay in reaction — and drones don't have that luxury in the air."

10 3.6: Controlling a Gimbal Using Joystick Input (via Microcontroller)

Understanding the System

A **gimbal** is a mechanical device with motors (usually 2 or 3) that keep a camera or sensor stabilized. You can also manually control it to pan or tilt the camera using a **joystick**.

In this system:

- The **joystick** is used by the user to send directional input.
- The **microcontroller** reads the joystick input.
- The microcontroller drives the **servo or brushless motors** on the gimbal to change its position.

Joystick Basics

A joystick has two axes:

- X-axis: Left–Right movement
- Y-axis: Up–Down movement

Internally, each axis is connected to a **potentiometer** that changes its voltage as you move the stick. This voltage can be read by the microcontroller using an `analogRead()`.

—

How It Works – Step-by-Step

1. Connect the joystick's X and Y outputs to analog input pins.
2. Map joystick values (usually from 0 to 1023) to gimbal angles (e.g., 0° to 180°).
3. Send the mapped value to servo motors connected to the gimbal using PWM.
4. Update these values continuously in the loop.

—

Pseudocode (Simplified)

```
// Initialize pins
joystickX = A0
joystickY = A1
servoX = 9
servoY = 10

void setup() {
    attachServo(servoX)
    attachServo(servoY)
}

void loop() {
    // 1. Read joystick values
    xVal = analogRead(joystickX)   // 0 to 1023
    yVal = analogRead(joystickY)

    // 2. Map to angles
    xAngle = map(xVal, 0, 1023, 0, 180)
    yAngle = map(yVal, 0, 1023, 0, 180)
```

```
    // 3. Send to servo motors
    servoX.write(xAngle)
    servoY.write(yAngle)

    delay(20) // Small delay for servo update
}
```

—

What I Learned

At first, I thought controlling a gimbal would need complex software. But now I see that with a joystick and two servo motors, it's possible to create a basic, manual gimbal control system.

Of course, professional gimbals use IMUs and PID loops to stabilize against motion — but this project taught me the basics of translating **analog input into smooth physical movement**, which is the core of many robotic systems.

Final Thoughts: A Respectful Suggestion for Broader Vision

While the concept of a single-seater flying car prototype is both thrilling and technically ambitious, I believe it's also important to occasionally pause and reflect on the broader purpose and long-term societal impact of such innovation.

As I explored the avionics, control systems, and flight dynamics for this project, a question naturally emerged in my mind:

Is putting people in the sky the only way to solve ground-level congestion? Or could the same technology — with a shift in purpose — serve humanity even more efficiently?

If our primary goal is to reduce traffic and enhance delivery logistics, then perhaps a more scalable and sustainable approach lies in leveraging drones not for air taxis, but for automated goods transportation. Imagine a city-wide infrastructure of autonomous aerial vehicles delivering:

- Groceries, medicines, and essential supplies
- Time-sensitive medical items (blood, vaccines, organs)
- Tools and components for emergency response
- Even patients in ultra-light emergency pods in specific cases

Such a system would:

- Reduce human risk (no onboard passengers)
- Simplify regulation and certification requirements

- Demand less power and structural redundancy
- Allow for full autonomy using current AI and swarm technologies
- Be more modular, serviceable, and scalable across cities and towns

To be clear, this is not a critique of the Vihang concept — which I deeply admire as a technical challenge and innovation platform — but rather a complementary vision that builds on its core principles. Sometimes, the same technology, when redirected, can solve multiple problems more efficiently.

In fact, exploring both tracks — human-carrying flying vehicles and autonomous logistic drones — could lead to hybrid designs: vehicles that serve as both personal flyers and smart delivery platforms, adapting based on mission.

As someone deeply passionate about aerospace and robotics, I found myself not just building flight systems, but imagining what they mean in the bigger picture. That reflection, I believe, is just as important as the engineering itself.

"The future isn't only built on better machines — it's built on better questions."

With curiosity and respect,
Owais Khan