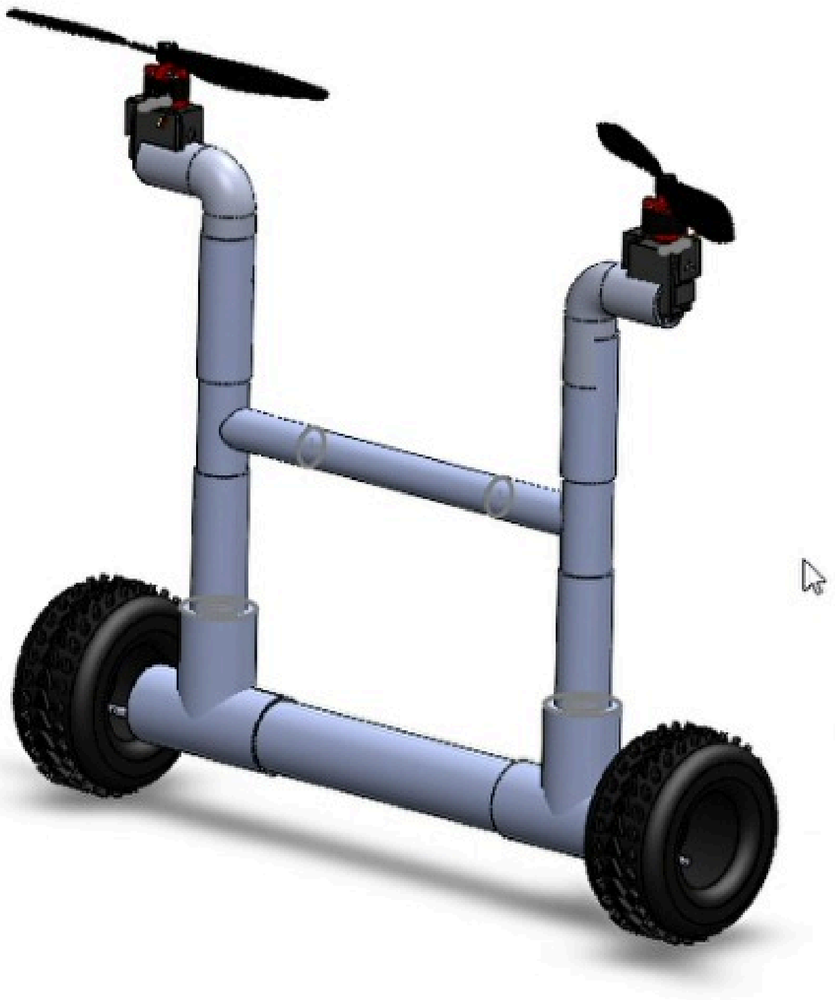
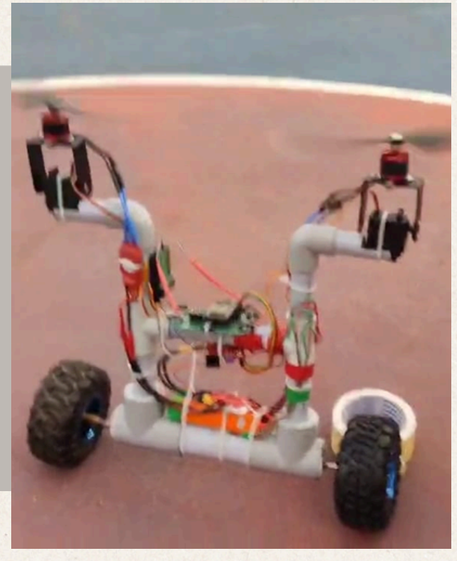


TURBO X2

Hybrid Bicopter with Balancing Bot Configuration

Ground Mode

Aerial Mode



- This project focuses on designing and developing a hybrid bicopter with a balancing bot configuration for surveillance applications. The hybrid system combines the VTOL capabilities of a bicopter with the self balancing stability of a wheeled robot, allowing seamless transitions between aerial and ground modes.

- The system will integrate IMU sensors and wireless communication for enhanced control.

What we did...

Modelling our drone - SOLIDWORKS

Balancing Bot Simulation - SIMULINK

PID Tuning - Balancing Bot - Arduino

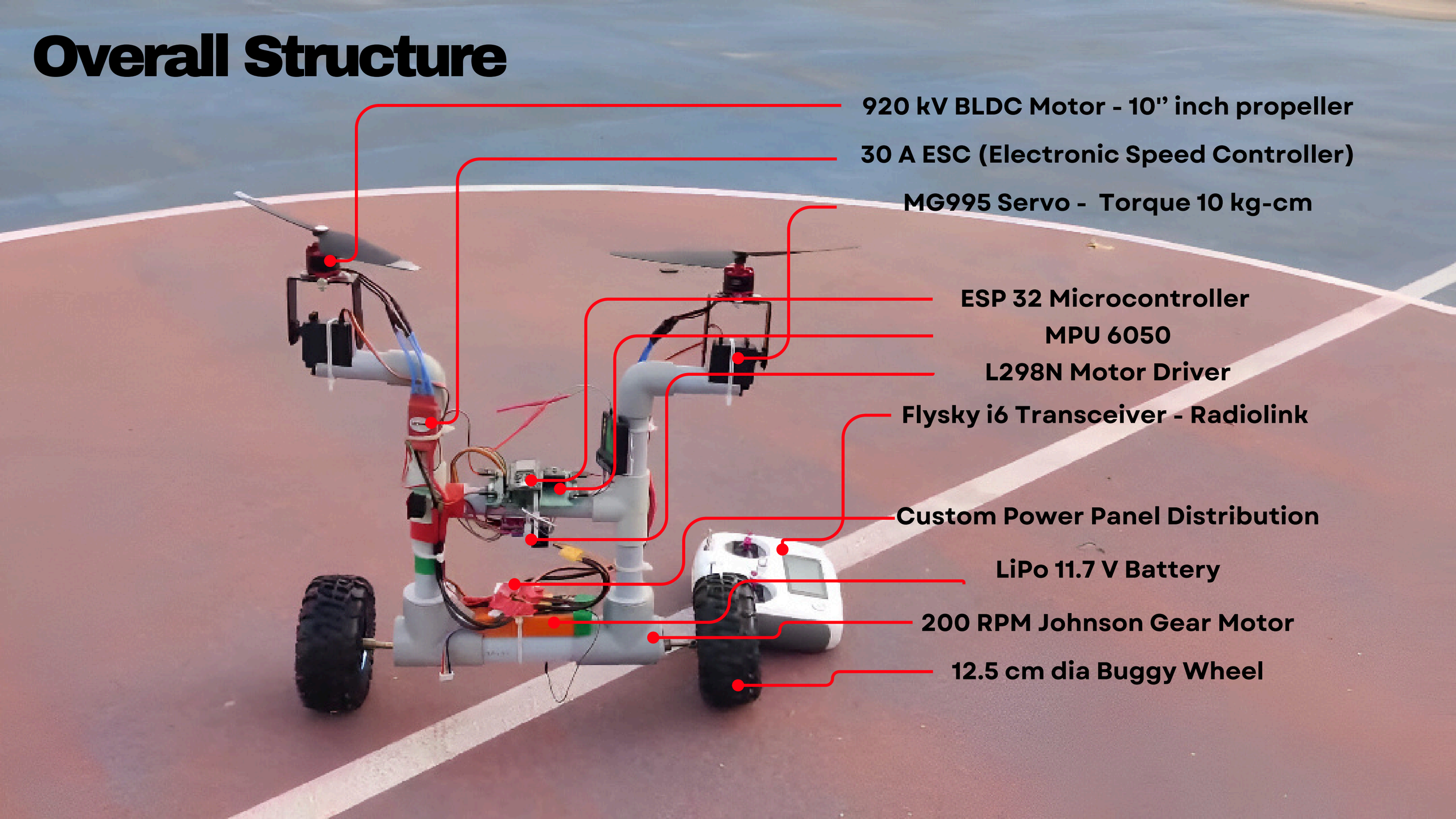
Building and testing the servo mechanism

Remodelling our frame due to weight contrains

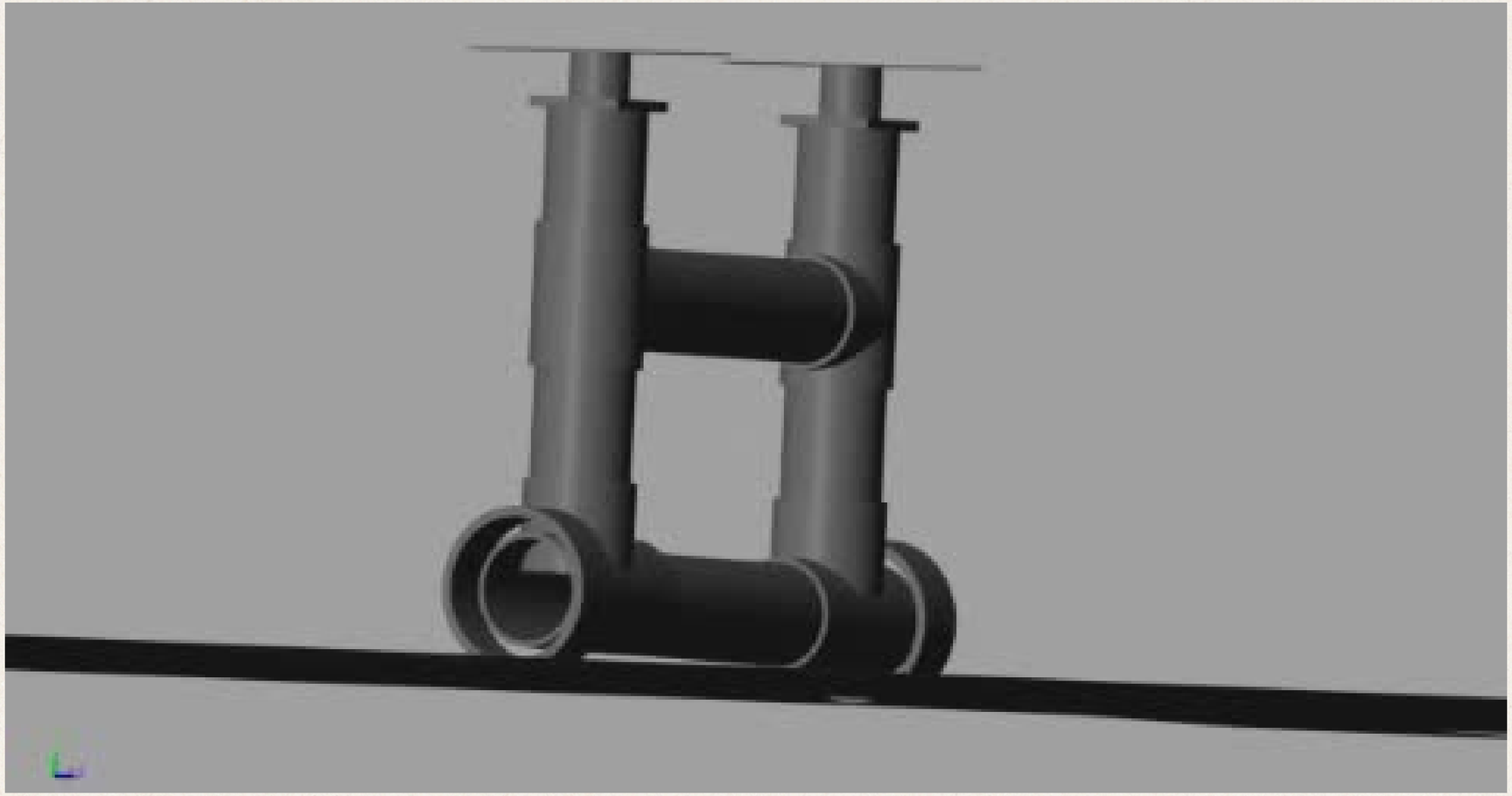
Implementing PPM - Bicopter control

Switching between two modes

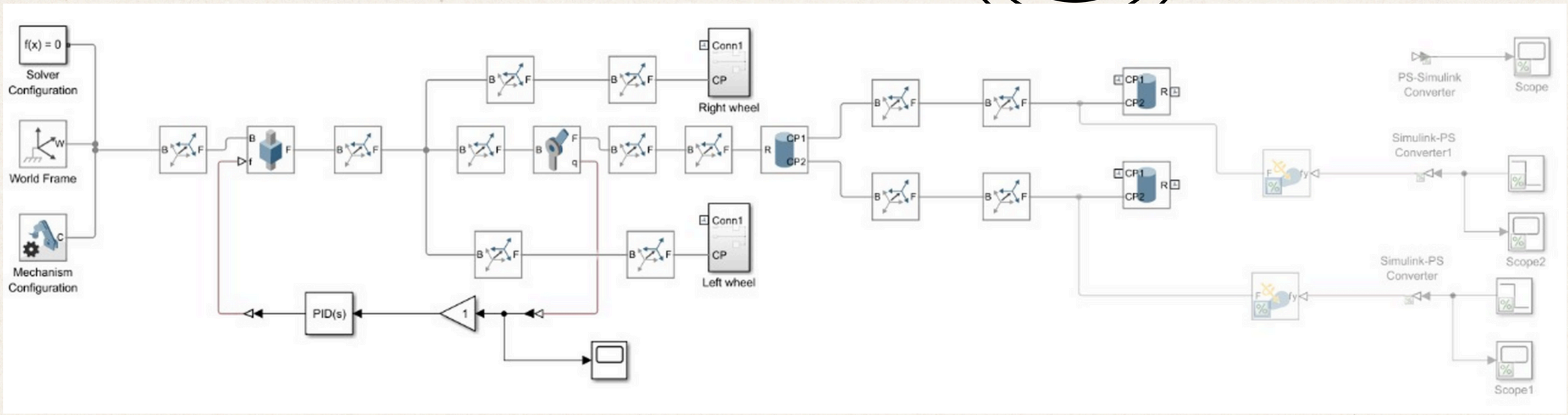
Overall Structure



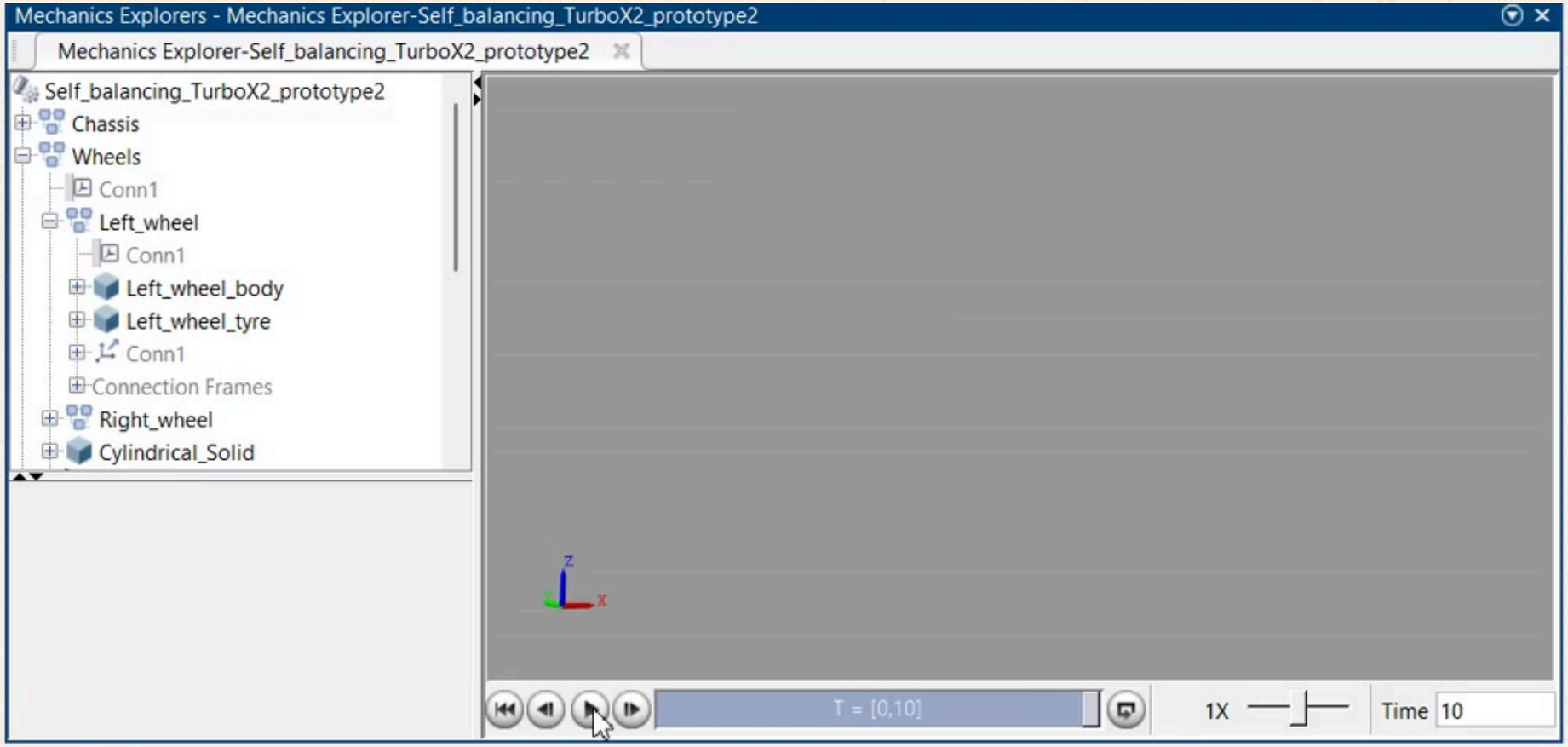
Modelling in Simulink



Simulink Model – Balancing Bot Dynamics



Inverted Pendulum Model

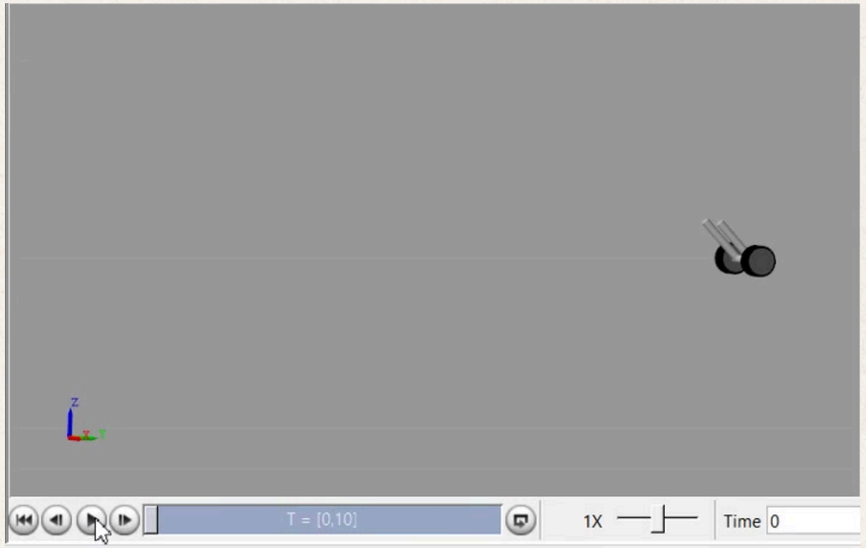


Simulation For Self-Balancing Bot

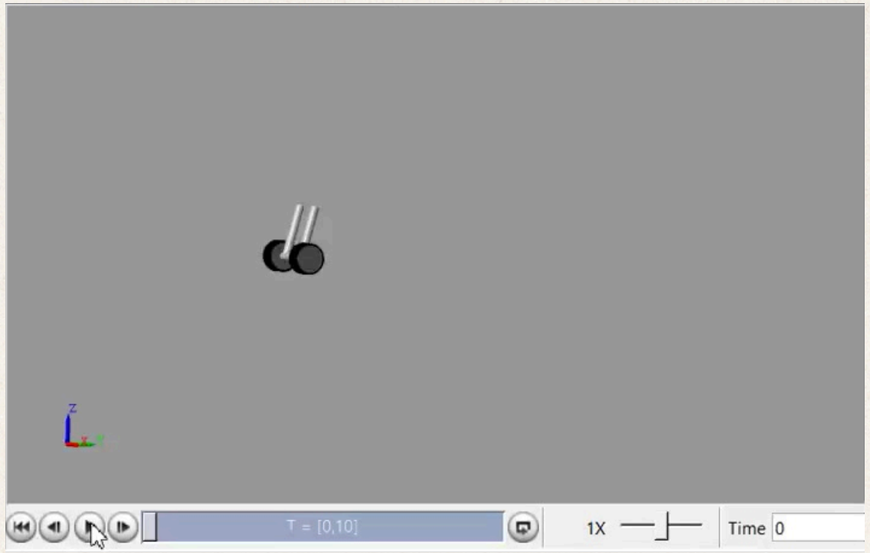
Untuned Bot

Tuned Bot

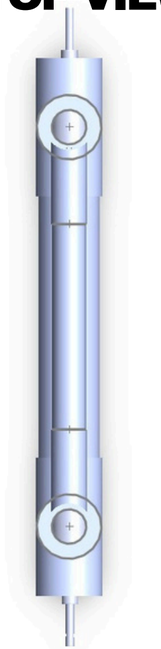
Initial - +45' inclination



Initial - -15' inclination



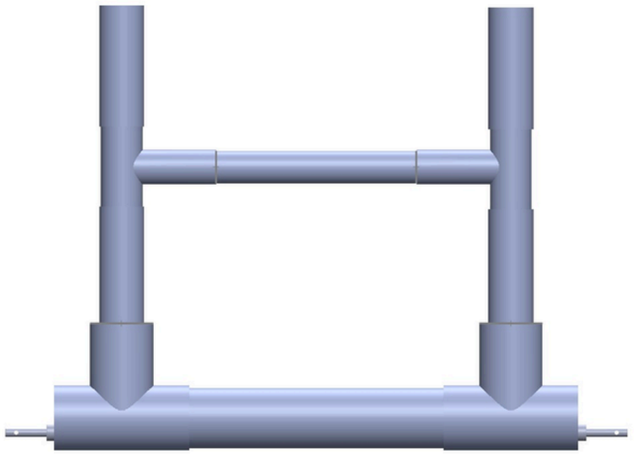
TOP VIEW



BOTTOM VIEW

SIDE VIEW

FRONT VIEW

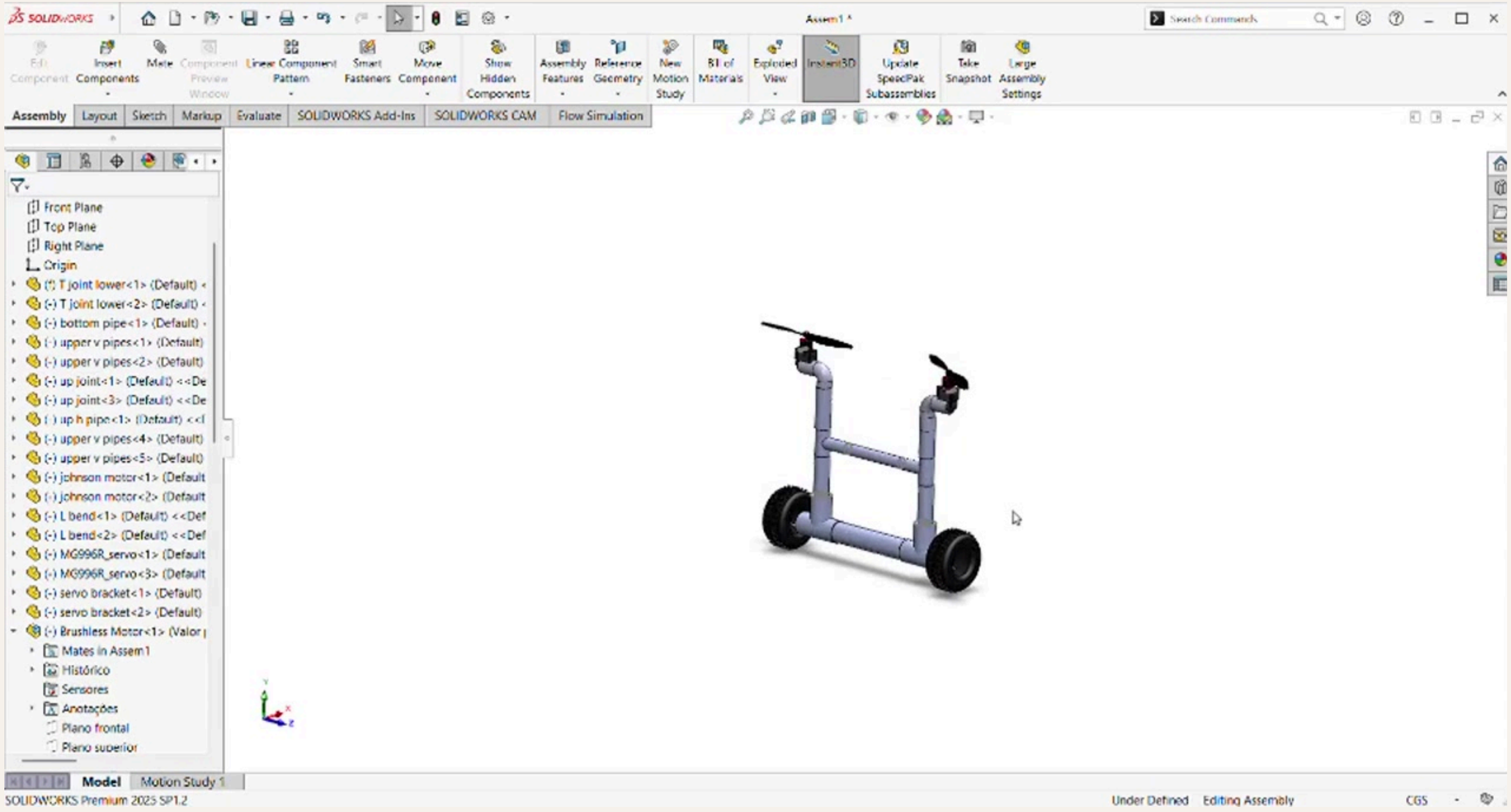


SolidWorks - Design of frame

We tried designing our Turbo X2 frame using **SolidWorks - CAD software** focusing the component compatibility for a hybrid bicopter with a self-balancing bot.

The frame integrates two main modules: the vertical balancing chassis and the rotor arm structure, ensuring seamless coordination between aerial and ground modes.

3D Model - SolidWorks



Self Balancing Bot - PID controller

ESP32_SelfBalancing.ino

```
43  void loop() {
44      int16_t ax, ay, az, gx, gy, gz;
45      m                6(&ax, &ay, &az, &gx, &gy, &gz);
46
47      // Calculate angle from accelerometer
48      input = atan2(ax, az) * 180 / PI;
49
50      // Compute PID
51      unsigned long now = millis();
52      float elapsedTime = (now - lastTime) / 1000.0;
53      float error = setPoint - input;
54      integral += error * elapsedTime;
55      float derivative = (error - previousError) / elapsedTime;
56      output = (Kp * error) + (Ki * integral) + (Kd * derivative);
57      previousError = error;
58      lastTime = now;
```

Loading...

- The MPU6050 gives the tilt angle.
- The setpoint is ideally 0° (upright).
- The PID loop compares the current angle to the setpoint.
- Based on the result (output), the bot moves motors forward or backward to correct the tilt.

PPM (Pulse Position Modulation)

- Sends multiple RC channel signals - single wire
- Each pulse gap represents a channel value.
- Uses hardware interrupt to capture pulses (**attachInterrupt()**)
- **micros()** function for precise pulse timing
- Channels stored sequentially: CH1 = ppmChannels[0], CH2 = ppmChannels[1]
- Handles up to 10 channels in a single frame

Bicopter - PPM Implementation

```
28  // RC input (PPM values for 10 channels)
29  volatile uint16_t ppmChannels[10];
30  volatile uint8_t ppmIndex = 0;
31  uint32_t lastRise = 0;
32
33  void IRAM_ATTR ppmISR() {
34    uint32_t now = micros();
35    uint16_t duration = now - lastRise;
36    lastRise = now;
37
38    if (duration >= 300 && duration <= 2100) {
39      if (ppmIndex < 10) {
40        ppmChannels[ppmIndex++] = duration;
41      }
42    } else if (duration >= 3000) {
43      ppmIndex = 0; // Start of new frame
44    }
45  }
```

Switching Modes

```
bool modeBicopter = ch[5] > 1500;  // Switch mode via Channel 6

if (modeBicopter) {
  float pitchCmd = (ch[1] - 1500) / 10.0;
  int throttle = constrain(ch[2], 1000, 2000);
  float pid = computePid(pitchCmd - pitch, kPB, kIB, kDB, prevErrB, intB, lastB);
  runBicopter(throttle, pid);
} else {
  float pid = computePid(setPtBot - pitch, kPBot, kIBot, kDBot, prevErrBot, intBot, lastBot);
  runBalancer(pid);
}
```

- Uses RC Channel 6 (ch[5]) to toggle between modes
  - ch[5] > 1500 → Bicopter Mode
  - ch[5] ≤ 1500 → Balancer Mode
- Uses noInterrupts() and interrupts() to avoid race conditions
- Mode status stored in bool modeBicopter

Serial PID Tuning

- Updates appropriate PID variables
  - kPBot, kIBot, kDBot (for Self-balancing Bot)
  - kPB, kIB, kDB (for Bicopter)
- Live updates shown via Serial.print() for confirmation

Advantages

- Real-time PID tuning without re-uploading code
- Enables smooth switching and stabilization during field testing

```
// === Serial PID Tuning ===
void handleSerialTuning() {
  if (!Serial.available()) return;
  String cmd = Serial.readStringUntil('\n'); cmd.trim();
  if (cmd.length() == 0) return;

  String target = cmd.substring(0, cmd.indexOf(' '));
  cmd = cmd.substring(cmd.indexOf(' ') + 1);
  String param = cmd.substring(0, cmd.indexOf(' '));
  float val = cmd.substring(cmd.indexOf(' ') + 1).toFloat();

  if (target == "BOT") {
    if (param == "KP") kPBot = val;
    if (param == "KI") kIBot = val;
    if (param == "KD") kDBot = val;
    Serial.printf("[BOT PID] KP=%.2f, KI=%.2f, KD=%.2f\n", kPBot, kIBot, kDBot);
  } else if (target == "BICOPTER") {
    if (param == "KP") kPB = val;
    if (param == "KI") kIB = val;
    if (param == "KD") kDB = val;
    Serial.printf("[BICOPTER PID] KP=%.2f, KI=%.2f, KD=%.2f\n", kPB, kIB, kDB);
  }
}
```

THANK YOU

Hariharan
Jagakishan
Pragadeesh
Madhavshankar