ME4250-1

ROBOT AUTO-BALANCÍN

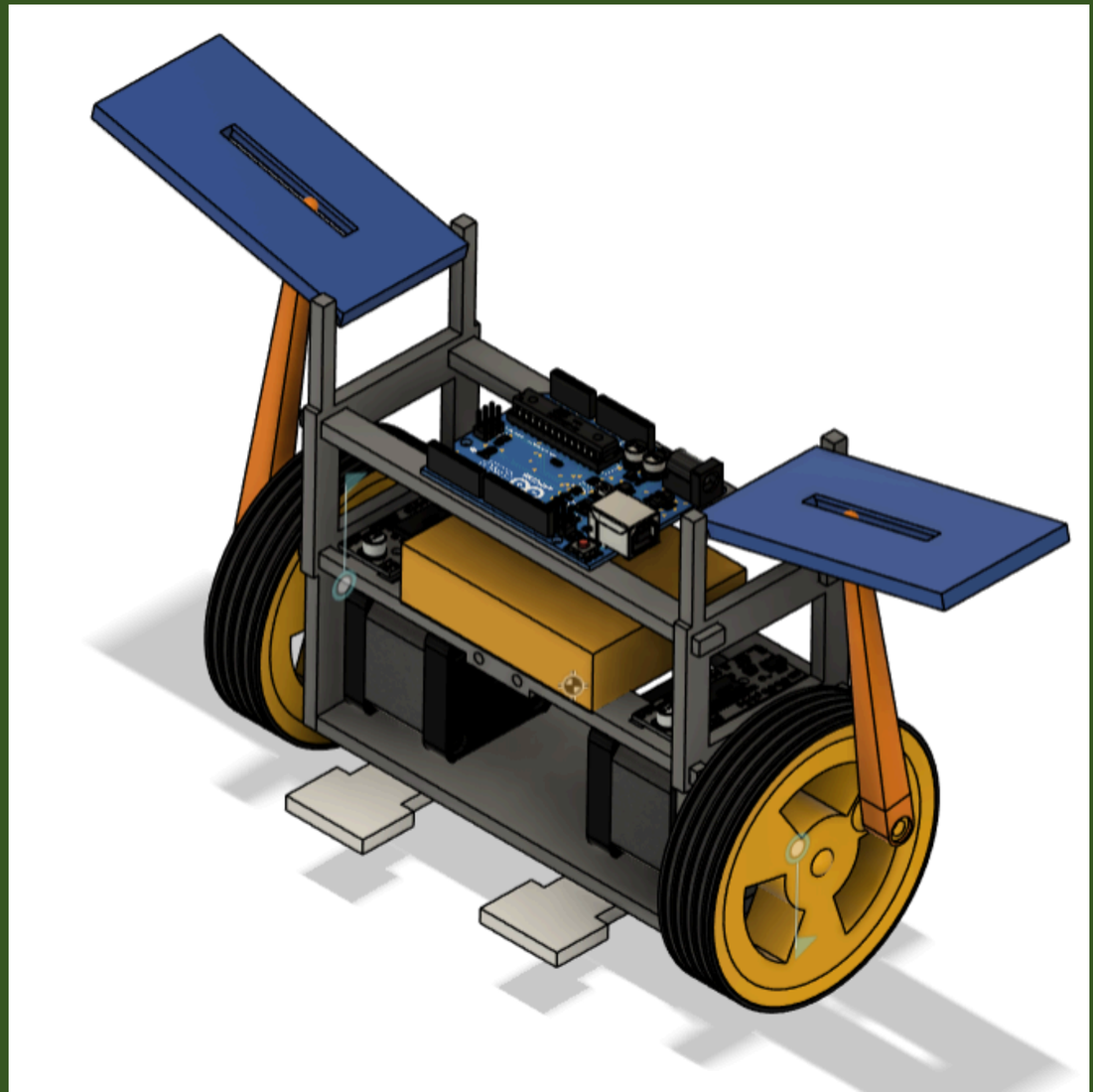
AVANCE 3

GRUPO 3

Integrantes:
Marina Olmedo.
Félicie Nguyen.
Joaquin Poblete Morales.
Enrique Rebolledo.
Pablo Varetto.

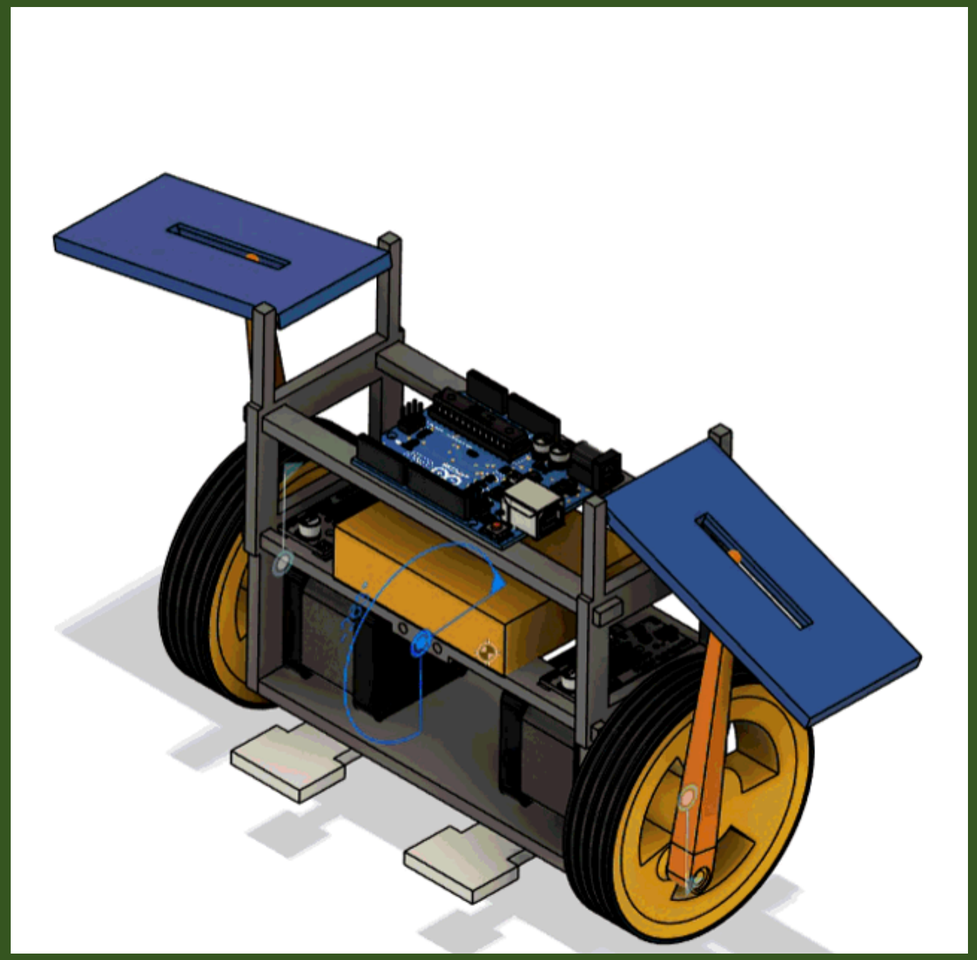
1 - MODELO

Carcasa y elementos del robot :



1 - MODELO

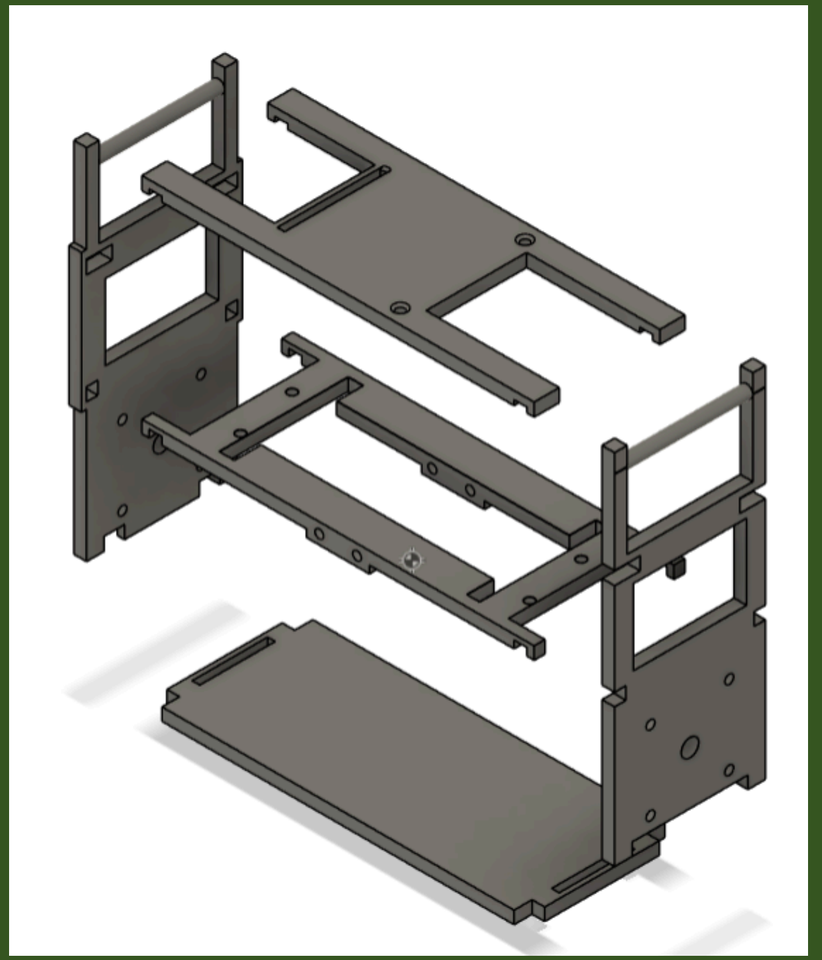
Simulacion del funcionamiento del robot :



1 - MODELO

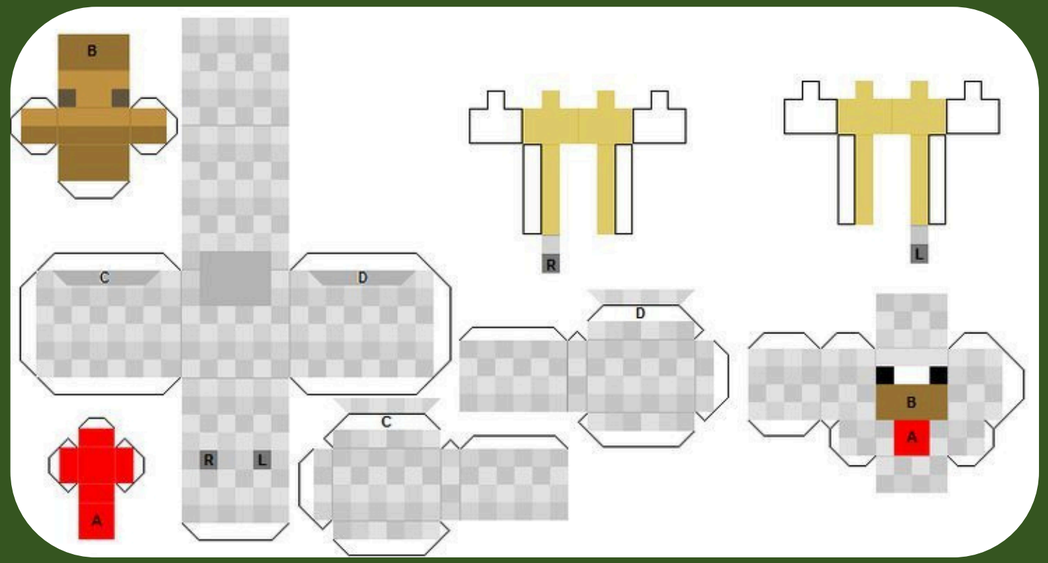
Carcasa :

- Simplificacion de las impresiones
- Facilidad de montaje
- Posibilidad de remplazar rapidamente si una pieza se rompe

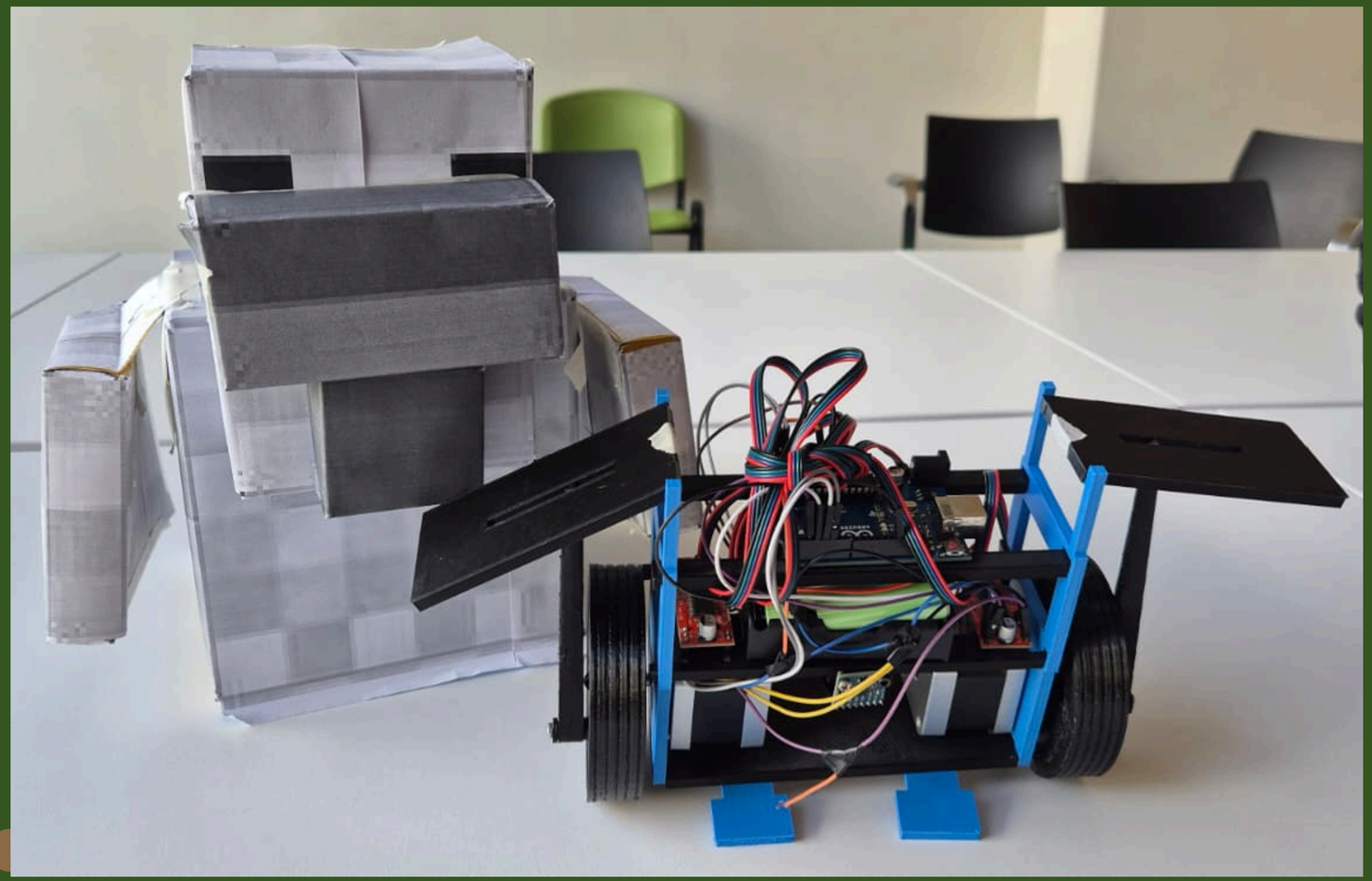


1 - MODELO

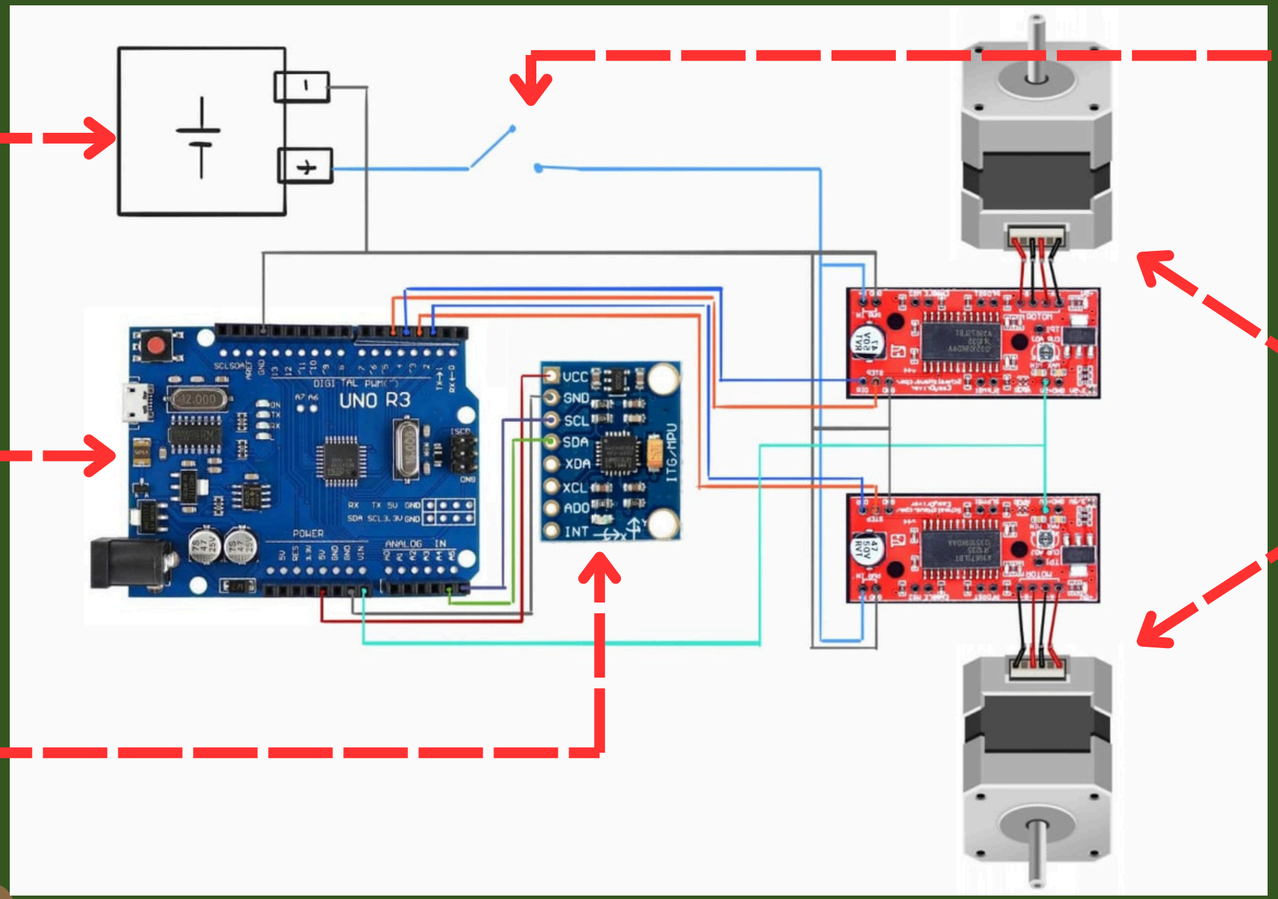
Diseño exterior :



1- MODELO



2 - CIRCUITO


Caja de pilas
Arduino UNO
Sensor (girometro)
Interuptor
Motor NEMA 17 +
EasyDriver A3967
UNO R3
VCC
GND
SCL
SDA
XDA
XCL
ADO
INT

3- CODIGOS

Motores :

```
// Librerias //

#include <Wire.h> // Para comunicacion del sensor
#define PI 3.14159265
#include <Stepper.h> // Para control de motores

////// DEFINICION DE MOTOR /////

const int stepsPerRevolution = 200;

// Motor 1 en pines STEP=3, DIR=2
Stepper stepper1(stepsPerRevolution, 3, 2);

// Motor 2 en pines STEP=5, DIR=4
Stepper stepper2(stepsPerRevolution, 5, 4);
```

3- CODIGOS

Sensor :

```
////// DEFINICION DE SENSOR /////

const int MPU_ADDR = 0x68;

int16_t ax, ay, az;
int16_t gx, gy, gz;

int16_t gx_offset = 0, gy_offset = 0, gz_offset = 0;

float roll = 0, pitch = 0, yaw = 0;
unsigned long lastTime;
float dt;
```

3- CODIGOS

PID:

```
30  ////// DEFINICION DE PID /////
31
32
33  ////////////////////// PARÁMETROS DEL CONTROL PID //////////////////////
34  // Setpoint: El ángulo deseado (para un sistema auto-balanceante es 0 grados)
35  float setpoint = 0.0;
36
37  // Ganancias PID (Ajustables)
38  float Kp = 2.0;         // Proporcional: reacciona al error actual
39  float Ki = 0.6;         // Integral: elimina el error acumulado
40  float Kd = 0.1;         // Derivativa: predice el error futuro
41
42  ////////////////////// VARIABLES DEL CÁLCULO PID //////////////////////
43
44  // Variables de cálculo del PID
45  float error = 0.0;
46  float error_prev = 0.0;
47  float integral = 0.0;
48  float derivada = 0.0;
49  float salida = 0.0;    // La salida del PID es la corrección a aplicar (torque/velocidad)
50
```

```
// Control de tiempo para el cálculo del PID y del sensor
const unsigned long sampleTime = 10; // ms (100 Hz, un buen punto de partida para control)

////////////////////// VARIABLES DE CONTROL DE MOTOR //////////////////////
// La salida del PID se convierte en la velocidad/dirección de los motores.
int velocidadMotor1 = 0;
int velocidadMotor2 = 0;


////////// VOID SETUP //////////////////////

void setup() {
    // 1. Configuración de Comunicación Serial
    Serial.begin(115200);
    Serial.println("=== SISTEMA DE CONTROL PID DE EQUILIBRIO ===");

    // 2. Inicialización del MPU-6050 (I2C)
    Wire.begin();
    Wire.beginTransmission(MPU_ADDR);
    Wire.write(0x6B); // Power management register
    Wire.write(0);    // Wake up MPU-6050
    Wire.endTransmission(true);

    // 3. Calibración del Sensor
    Serial.println("=== MPU-6050 Calibration ===");
    Serial.println("Place the sensor still and press any key + ENTER to start...");
    while (!Serial.available());  // Wait for user input
    while (Serial.available()) Serial.read();  // Clear input buffer
    calibrateGyro();
    Serial.println("Calibration complete!");

    // 4. Inicialización de Motores
    stepper1.setSpeed(0); // en RPM
    stepper2.setSpeed(0);

    // 5. Configuración de tiempo final
    lastTime = millis();
    Serial.println("Graficar en Serial Plotter: Roll, Setpoint, Salida");
}
```

3- CODIGOS

```
93
94  void loop() {
95
96      // --- CONTROL DE TIEMPO (Solo se ejecuta cada 'sampleTime' ms) ---
97      unsigned long now = millis();
98
99      if (now - lastTime >= sampleTime) {
100         // Calcular dt para el PID y reajustar el tiempo de la última ejecución
101         float dt = (now - lastTime) / 1000.0; // Diferencia de tiempo en segundos
102         lastTime = now; // Reinicia el contador de tiempo para el siguiente ciclo
103
104         // --- 1. LECTURA Y CÁLCULO DE ÁNGULO (SENSOR) ---
105         readMPU6050();
106
107         // Convertir int16_t a float
108         float ax_f = (float)ax;
109         float ay_f = (float)ay;
110         float az_f = (float)az;
111
112         // Cálculo de Roll (Input al PID)
113         float denominator = sqrt(ay_f * ay_f + az_f * az_f);
114         if (denominator < 0.0001) denominator = 0.0001;
115         roll = atan2(ay_f, az_f) * 180.0 / PI;
116
117         // (Opcional, el Pitch y Yaw del sensor original)
118         // pitch = atan2(-ax_f, denominator) * 180.0 / PI;
119         // float gyroZ = (gz - gz_offset) / 131.0;
120         // yaw += gyroZ * dt;
121
122         // --- 2. CÁLCULO PID ---
123
124         // Calcular error: cuánto nos desviamos del setpoint (0 grados)
125         error = setpoint - roll;
126
127         // Cálculo de las componentes PID
128         float P = Kp * error;
129         integral += error * dt;
130         float I = Ki * integral;
131         derivada = (error - error_prev) / dt;
132         float D = Kd * derivada;
133
134         salida = P + I + D; // La salida es el valor de corrección (velocidad RPM)
135
```

```
136         // --- 3. SATURACIÓN y ANTI-WINDUP ---
137         // Utilizamos el rango de RPM que definimos antes (ej. -50 a 50)
138         const float MAX_OUTPUT_SPEED = 50.0;
139
140         if (salida > MAX_OUTPUT_SPEED) {
141             salida = MAX_OUTPUT_SPEED;
142             integral -= error * dt; // Anti-windup
143         } else if (salida < -MAX_OUTPUT_SPEED) {
144             salida = -MAX_OUTPUT_SPEED;
145             integral -= error * dt; // Anti-windup
146         }
147
148         // --- 4. APLICAR AL MOTOR ---
149         aplicarControlMotor(salida);
150
151         // --- 5. MONITOREO SERIAL ---
152         Serial.print(roll);
153         Serial.print(",");
154         Serial.print(setpoint);
155         Serial.print(",");
156         Serial.println(salida);
157
158         // --- 6. Guardar error anterior ---
159         error_prev = error;
160     }
161
162 }
```

3- CODIGOS

```
168  void readMPU6050() {
169    Wire.beginTransmission(MPU_ADDR);
170    Wire.write(0x3B);
171    Wire.endTransmission(false);
172    Wire.requestFrom(MPU_ADDR, 14, true);
173
174    ax = Wire.read() << 8 | Wire.read();
175    ay = Wire.read() << 8 | Wire.read();
176    az = Wire.read() << 8 | Wire.read();
177    Wire.read(); Wire.read(); // Skip temperature
178    gx = Wire.read() << 8 | Wire.read();
179    gy = Wire.read() << 8 | Wire.read();
180    gz = Wire.read() << 8 | Wire.read();
181  }
182
183  void calibrateGyro() {
184    long sumX = 0, sumY = 0, sumZ = 0;
185    const int samples = 100;
186
187    for (int i = 0; i < samples; i++) {
188      readMPU6050();
189      sumX += gx;
190      sumY += gy;
191      sumZ += gz;
192      delay(5);
193    }
194
195    gx_offset = sumX / samples;
196    gy_offset = sumY / samples;
197    gz_offset = sumZ / samples;
198  }
```

```
200  // Función de actuación para los motores paso a paso
201  void aplicarControlMotor(float salida) {
202
203    // Rango máximo de velocidad (en RPM)
204    const float MAX_RPM = 50.0;
205
206    // Limita la salida PID (Constrain)
207    salida = constrain(salida, -MAX_RPM, MAX_RPM);
208
209    // La salida PID (float) se convierte en la velocidad (int)
210    int velocidadMotor = (int)salida;
211
212    // Aplicar la velocidad (positiva/negativa indica dirección)
213    stepper1.setSpeed(velocidadMotor);
214    stepper2.setSpeed(velocidadMotor);
215
216    // Mover un paso para que el motor empiece a girar a la nueva velocidad
217    stepper1.step(1);
218    stepper2.step(1);
219  }
220
```

4- FUNCIONARA ?

MUCHAS GRACIAS