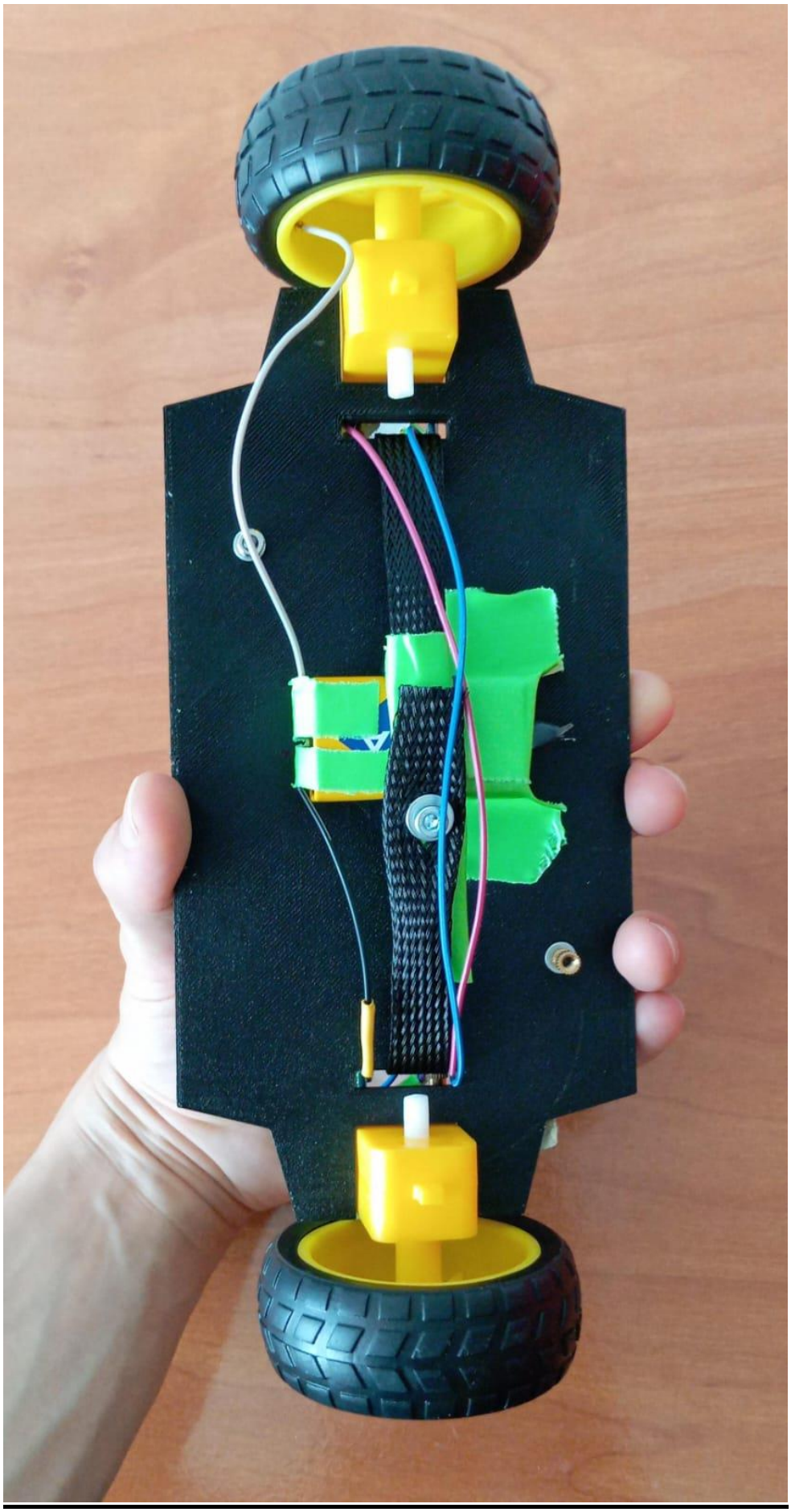
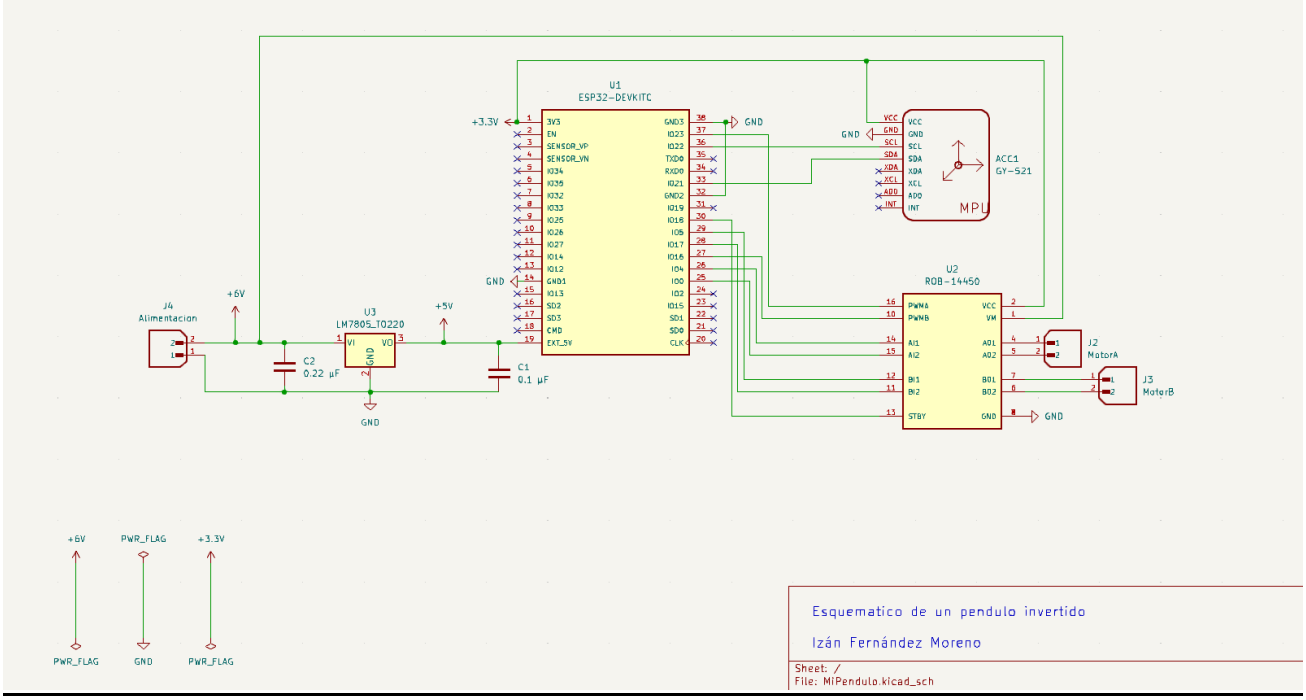
Izán Fernández Moreno

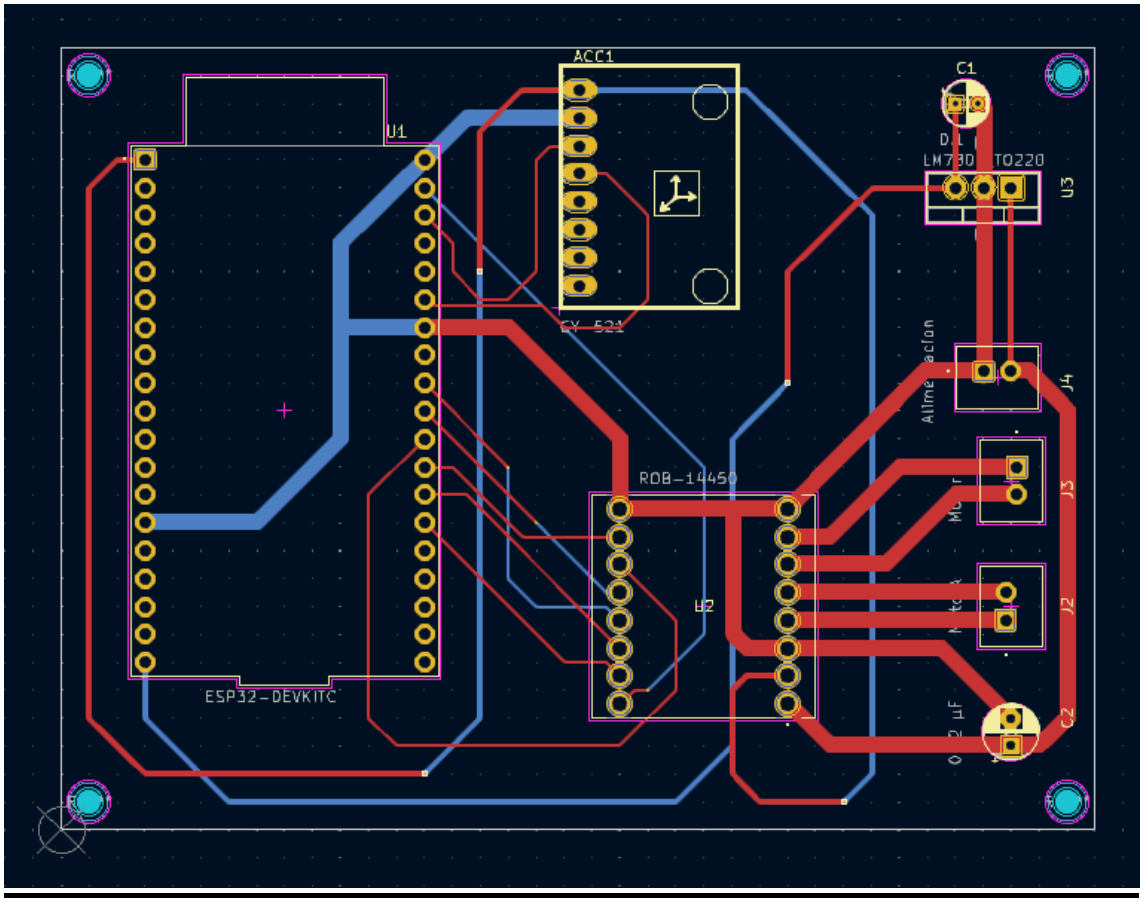
Pendulo Invertido



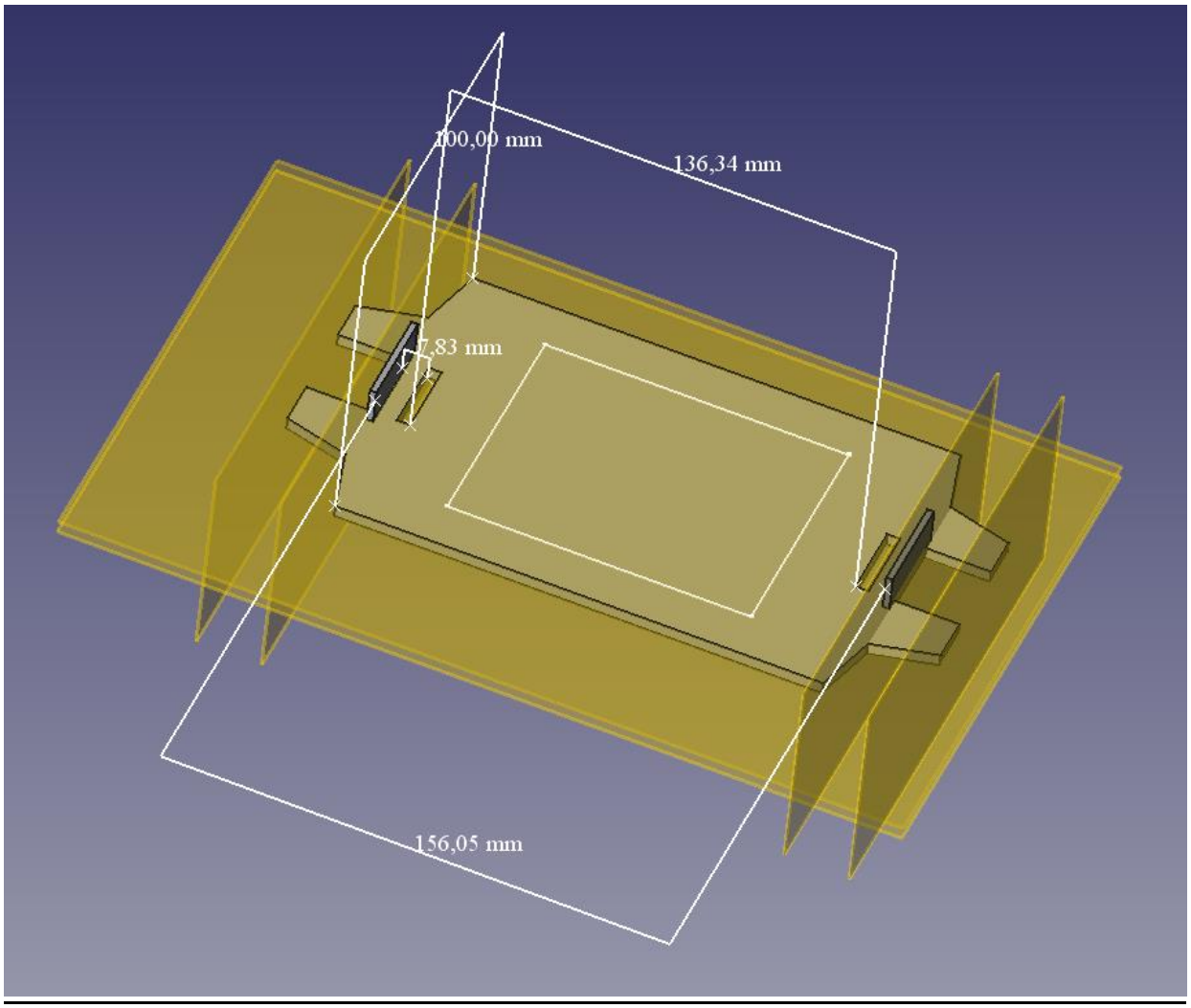
Esquematico del Circuito

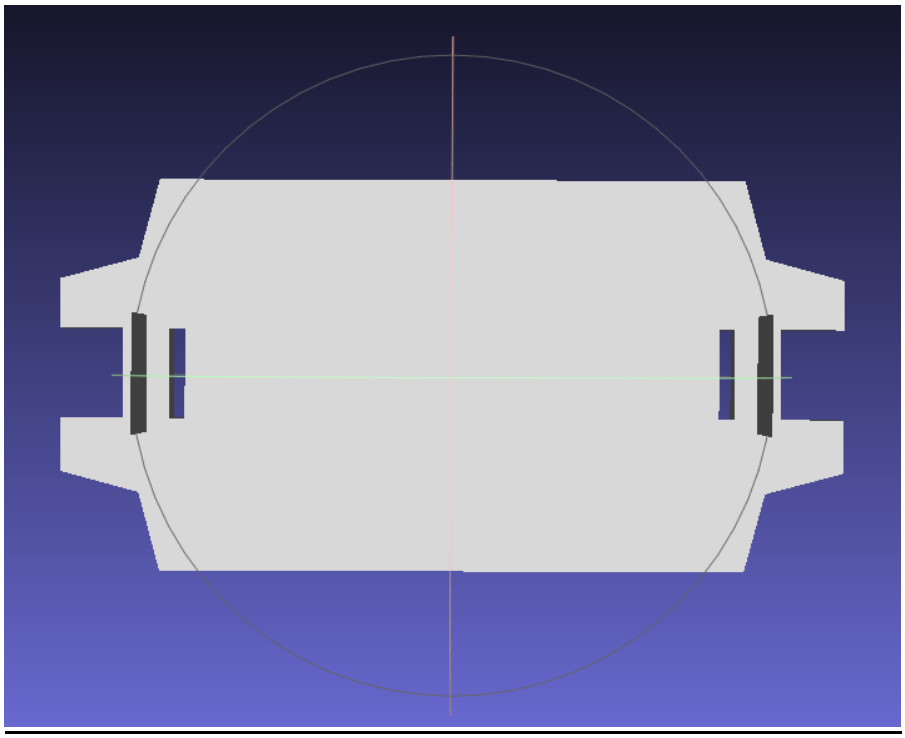


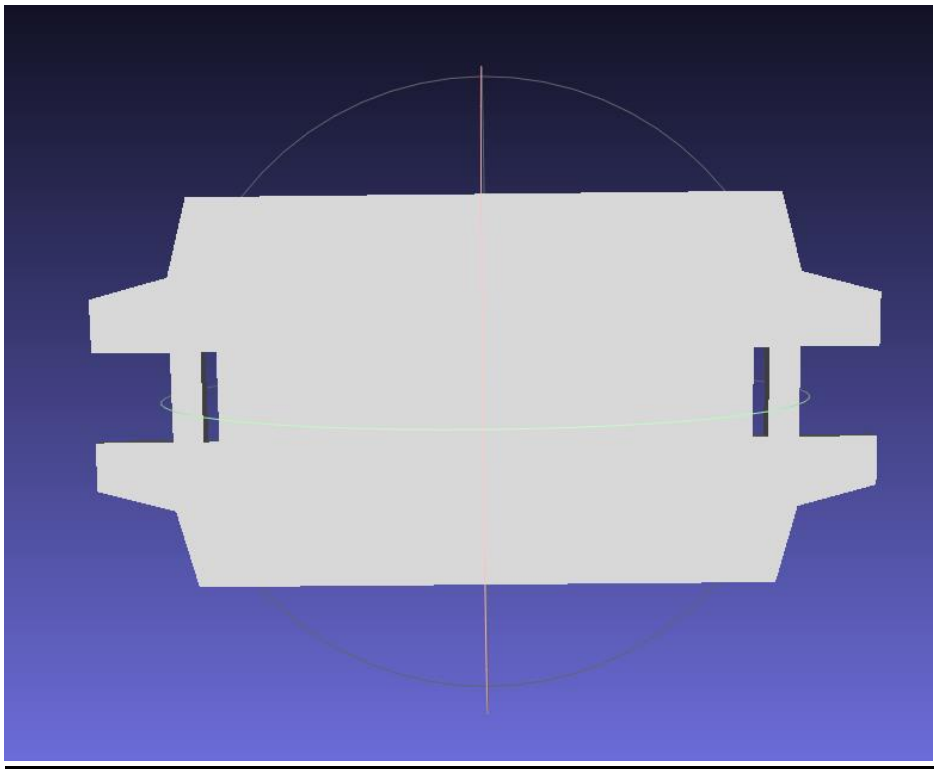
Diseño PCB

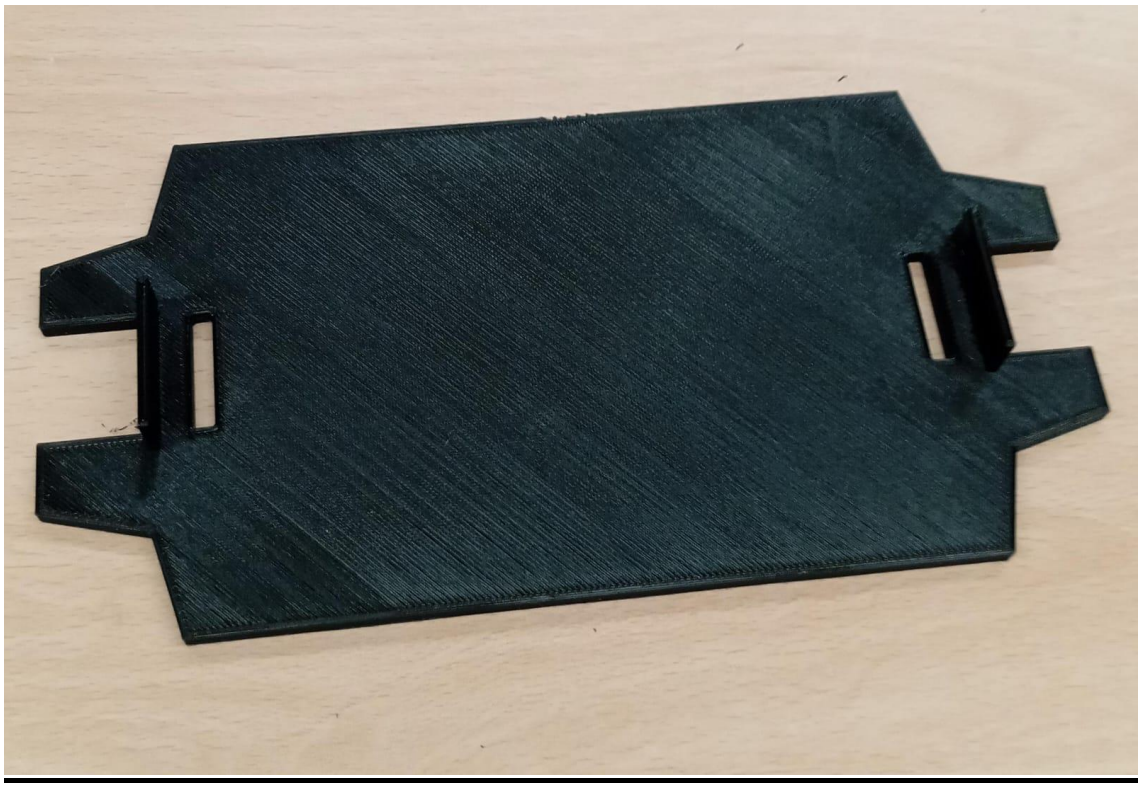


Diseño chasis









Codigo

```
#include "Wire.h"
#include "I2Cdev.h"
#include "MPU6050.h"

MPU6050 sensor;

// Definiciones de pines de motores (usando números GPIO reales)
const int pinPWMA = 23;
const int pinAIN2 = 0;
const int pinAIN1 = 4;
const int pinPWMB = 16;
const int pinBIN1 = 5;
const int pinBIN2 = 17;
```

```
const int pinSTBY = 18;

// Canales PWM para ESP32
const int pwmChannelA = 0;
const int pwmChannelB = 1;

// Variables del sensor MPU6050
int16_t ax, ay, az;
int16_t gx, gy, gz;
unsigned long tiempo_prev;
float dt;
float ang_x, ang_y;
float ang_x_prev = 0, ang_y_prev = 0;

// Variables PID
float Kp = 0;
float Ki = 0;
float Kd = 0;
float integral = 0.0;
float derivative = 0.0;
float previous_error = 0.0;

void setup() {
  Serial.begin(115200);
  Wire.begin(32, 33);  // SDA, SCL
  sensor.initialize();
```

```
  if (!sensor.testConnection()) {
    Serial.println("MPU6050 no detectado. Verifica conexión.");
    while (1);
  }

  // Desactiva motores por seguridad
  digitalWrite(pinAIN1, LOW);
  digitalWrite(pinAIN2, LOW);
  digitalWrite(pinBIN1, LOW);
  digitalWrite(pinBIN2, LOW);
  digitalWrite(pinSTBY, LOW);

  // Configurar pines de motores
  pinMode(pinAIN2, OUTPUT);
  pinMode(pinAIN1, OUTPUT);
  pinMode(pinPWMA, OUTPUT);
  pinMode(pinBIN1, OUTPUT);
  pinMode(pinBIN2, OUTPUT);
  pinMode(pinPWMB, OUTPUT);
  pinMode(pinSTBY, OUTPUT);

  // Configurar PWM en ESP32
  ledcAttachPin(pinPWMA, pwmChannelA);
  ledcSetup(pwmChannelA, 5000, 8);  // 5 kHz, resolución de 8 bits (0-255)

  ledcAttachPin(pinPWMB, pwmChannelB);
  ledcSetup(pwmChannelB, 5000, 8);
```

```
  tiempo_prev = millis();
}

void loop() {
  // Obtener datos del MPU6050
  sensor.getAcceleration(&ax, &ay, &az);
  sensor.getRotation(&gx, &gy, &gz);

  // Calcular ángulos con el acelerómetro
  float accel_ang_x = atan(ay / sqrt(pow(ax, 2) + pow(az, 2))) *
(180.0 / PI);
  float accel_ang_y = atan(-ax / sqrt(pow(ay, 2) + pow(az, 2))) *
(180.0 / PI);

  // Calcular ángulo de rotación con giroscopio y filtro
complementario
  unsigned long tiempo_actual = millis();
  dt = (tiempo_actual - tiempo_prev) / 1000.0;
  tiempo_prev = tiempo_actual;

  ang_x = 0.98 * (ang_x_prev + (gx / 131.0) * dt) + 0.02 *
accel_ang_x;
  ang_y = 0.98 * (ang_y_prev + (gy / 131.0) * dt) + 0.02 *
accel_ang_y;

  ang_x_prev = ang_x;
  ang_y_prev = ang_y;
```

```
  // Control del péndulo invertido usando PID
  float desired_angle = 0.0; // Ángulo deseado para mantener el equilibrio
  float error = desired_angle - ang_x;

  integral += error * dt;
  derivative = (error - previous_error) / dt;

  float motor_speed = Kp * error + Ki * integral + Kd * derivative;
  previous_error = error;

  // Controlar los motores
  enableMotors();
  moveMotor(pinPWMA, pinAIN1, pinAIN2, motor_speed, pwmChannelA);
  moveMotor(pinPWMB, pinBIN1, pinBIN2, motor_speed, pwmChannelB);

  // Mostrar por monitor serie
  Serial.print("Rotación en X: ");
  Serial.print(ang_x);
  Serial.print(", Error: ");
  Serial.print(error);
  Serial.print(", Velocidad PWM: ");
  Serial.println(motor_speed);

  delay(1); // Frecuencia alta (~1000 Hz)
}
```

```
// Activa el chip del driver de motores
void enableMotors() {
  digitalWrite(pinSTBY, HIGH);
}

// Funcion de control de un motor con PWM
void moveMotor(int pinPWM, int pinIN1, int pinIN2, float speed, int pwmChannel) {
  int pwmValue = constrain(abs(speed), 0, 255);

  if (speed > 0) {
    digitalWrite(pinIN1, HIGH);
    digitalWrite(pinIN2, LOW);
  } else if (speed < 0) {
    digitalWrite(pinIN1, LOW);
    digitalWrite(pinIN2, HIGH);
  } else {
    digitalWrite(pinIN1, LOW);
    digitalWrite(pinIN2, LOW);
  }

  ledcWrite(pwmChannel, pwmValue);
}
```