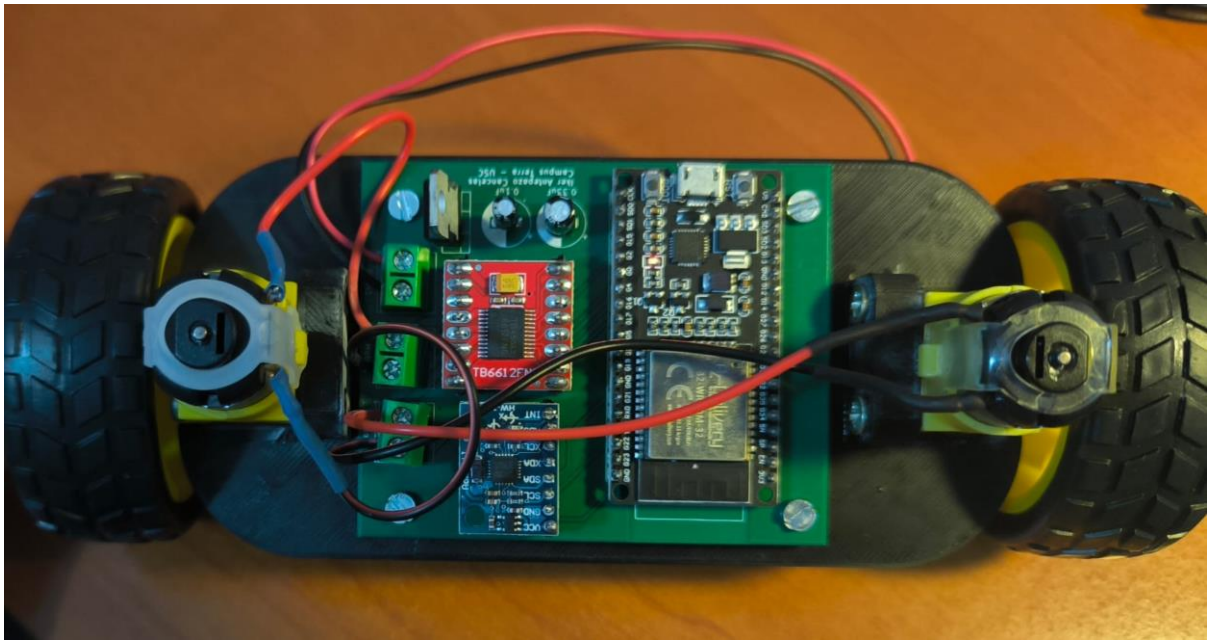
PENDULO INVERTIDO

Iker Antepazo Cancelas



Introducción:

En este documento recogeremos el proceso llevado a cabo para la creación y construcción de un robot llamado “péndulo invertido”, cuya función será de tratar de mantenerse en posición vertical con una tolerancia de ±5◦.
El sistema debe compensar los empujones externos y volver a la posición vertical y debe ser capaz de poder iniciar su funcionamiento desde cualquier posición.

Metodología seguida

1. **Buscar proyectos similares:** Investigar si ya existe un proyecto de un robot similar que sea funcional y esté documentado. Tanto en respositorios de github como en mi caso en videos de la plataforma YouTube y en la página “grabcad”.

2. **Investigación general:** Como no existe ningún proyecto bien hecho y bien documentado que esté completo, procedemos a investigar el modus operandi que trataremos de llevar a cabo el proyecto partiendo de las ideas que sacamos de los sitios web antes mencionados.

3. **Fijación de objetivos:** Es importante la fijación de ciertas metas para saber qué hardware necesitamos. Tanto es así para nuestro chasis como para nuestra placa PCB.

4. **Selección del Hardware:** Una vez que sabemos las metas que queremos lograr conseguir con nuestro robot, tenemos que buscar el hardware necesario que se adapte a la perfección con lo que necesitaremos.

5. **Diseño de la PCB:** Deberemos crear desde 0 una placa PCB diseñada por KiCad que contenga los elementos que emplearemos para el robot y las pistas con los mismos elementos conectados entre sí a través de los pines.

6. **Diseño del Chasis:** Deberemos resolver dos problemáticas: tener el peso compensado y tener el centro de gravedad bajo.

7. **Prueba de componentes por separado:** Verificamos que todo funcione por separado para luego soldar todo en la placa.

8. **Montaje:** Montamos todos los componentes en el chasis para poder probar el robot al completo.

9. **Programación inicial:** Hacemos un primer código que solo mantenga el robot estático en un mismo lugar.

10. **Ajuste del PID:** Ajustamos los valores del PID para que el robot se mantenga estático.

Objetivos

Estabilidad Perpendicular al Suelo:
Garantizar que el robot sea capaz de mantenerse estable en la posición perpendicular al suelo (vertical). ✓

Compensación de Empujones:
Desarrollar la capacidad del robot para compensar empujones y así evitar las caídas. ✓

Arranque desde Cualquier Posición: Lograr que el robot pueda iniciar su funcionamiento desde cualquier posición inicial. ✓

Reducción de Posibilidades de Volcar: Minimizar las posibilidades de que el robot vuelque y se caiga y asegurar que el robot no pueda volver a una posición no deseada. ✓

Investigación

En relación con la búsqueda de otros proyectos de este estilo, he detectado una problemática: hay varios proyectos hechos con el mismo principio, pero nadie ha creado una documentación completa y descriptiva del proyecto, esto conlleva a un mayor gasto y empleo de tiempo en la búsqueda de los elementos hardware que debemos comprar y buscar, al igual que para el desarrollo de la pcb empleada y del diseño del chasis. En parte tuve suerte ya que gracias a toda la información recopilada por las páginas web y gracias al trabajo del compañero Daniel Acevedo, que fue el primero en hacer este proyecto y gracias al cual he sacado adelante este robot. Aunque hay esta escasez de información, he podido rescatar algo de información examinando videos. De ellos, he determinado el hardware que se necesita y el tipo de control para manejar el movimiento (PID).

Hardware

Para la selección del hardware, se buscó que el robot se hiciera con elementos básicos y comunes para que sea un robot de bajo costo y no dependa de cierta solvencia para poder realizarse, ya que como no hay verdaderamente un informe completo y descriptivo de todos los pasos a seguir podría atascarse el proyecto y ser necesarias varias pruebas como fue mi caso.

ESP32:

Para el microcontrolador, he empleado una ESP32. En concreto, se ha utilizado una ESP32 que ronda los 11€ en Amazon. He consultado un par de páginas web para informarme al respecto de esta placa, y a continuación dejaré todo lo necesario a conocer sobre esta placa: ESP32 .
Este elemento es el encargado de mediante una conexión con cable USB al ordenador en la aplicación de Arduino, recibir los programas empleados.

MPU-6050:

Este sensor tiene incorporado un acelerómetro y un giroscopio, los cuales vamos a usar para medir la orientación e inclinación del robot, que como hemos comentado al principio del documento buscamos que pueda mantener la verticalidad sin sobrepasar los 5 grados de inclinación angular. Las fórmulas e información necesaria que he necesitado para usar este sensor están en esta página web: MPU6050 .

En el caso de este componente tuve serios problemas para encontrarlo ya que lo compré tres veces, pero me daba errores al probarlo hasta que acabé encontrando el idóneo.

Condensadores:
Empleé dos condensadores, uno de 0.1 uF y otro de 0.22uF: 0,1uF ,0,22uF

Bloques terminales:
En este aspecto emplee tres bloques terminales que fueron los encargados de conectarse con los motores y la batería. A continuación, dejaré un enlace para el acceso a sus características: Bloques terminales

TB6612FNG:
El driver que he elegido es una solución más moderna que el común puente H; su precio es de menos de 2€ y nos permite controlar 2 motores simultáneamente y de manera independiente. He seleccionado este componente al igual que los demás gracias a la documentación presentada por Daniel Acevedo. Al igual que con el anterior sensor, la página de la que extraje toda la información la dejo en el siguiente enlace: TB6612FNG

Batería: En este aspecto he empleado un portapilas de 4 pilas AA conectadas a los bornes de la placa. Portapilas AA

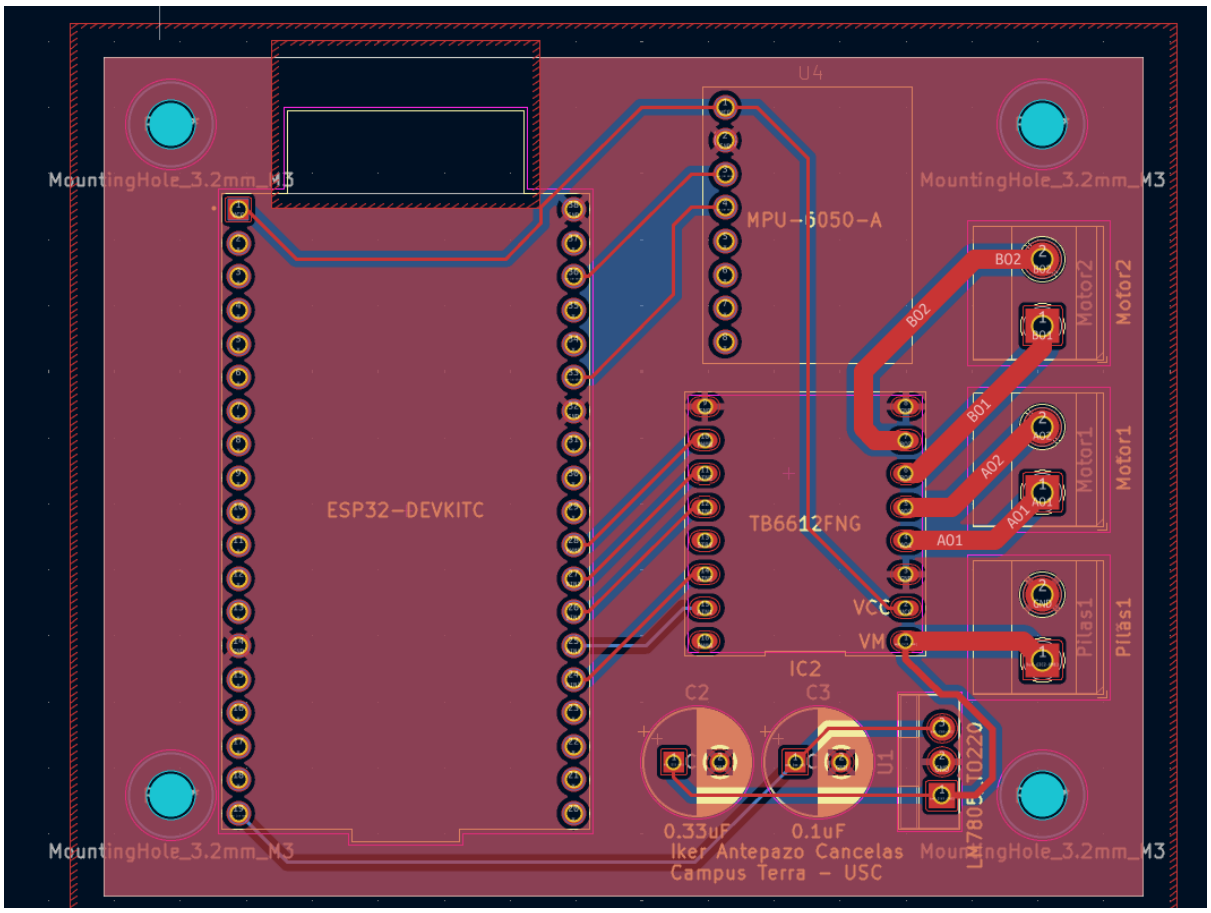
Motores y ruedas:
En este proyecto se han usado los comúnmente usados motores de DC con una reductora, la cual permite un fácil acople de una rueda que normalmente se vende con el propio motor. Motores y ruedas

Regulador de voltaje:
En este aspecto emplee un regulador de voltaje LM7805 que me servirá para controlar el flujo de voltaje en el circuito. Aquí dejaré una datasheet para comprender su funcionamiento y características: Regulador de voltaje

PCB

He diseñado una placa PCB capaz de emplear todos los elementos antes mencionados y que estén conectados entre sí de forma que nos funcione todo a la perfección. En este aspecto debemos buscar la obtención de una placa que no tenga mucho lío de pistas para ver la función de cada una de ellas a la perfección, buscamos una claridad importante para en caso de alguna duda presentada a la hora de la soldadura de los elementos con el estañador saber de qué pin a qué otro pin va cada pista.

Asímismo es de vital importancia el contar con una placa de pocas dismensiones para así facilitarnos el proceso de creación del chasis, porque no queremos un robot muy grande, con lo cual esto será muy importante.



Chasis

He diseñado 2 tipos de chasis para el robot el primero fue para tantear más o menos, y con el segundo, gracias a los consejos de Daniel he logrado hacer funcionar el robot ya que cumple correctamente con los requisitos del centro del centro de gravedad que estaba muy centrado y a una altura baja, lo que proporcionaba una gran estabilidad.

Lo importante a tener en cuenta para el desarrollo del chasis son los siguientes dos requisitos:

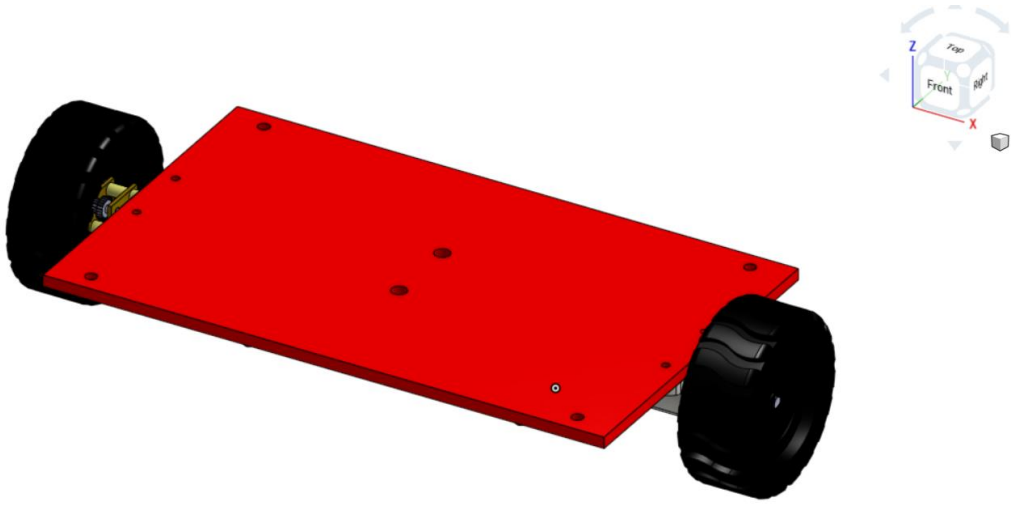
Centro de gravedad: Es determinante que el centro de gravedad del robot sea el más bajo posible. Si el peso está muy elevado respecto al suelo, se hará prácticamente imposible que el robot sea capaz de mantenerse derecho.

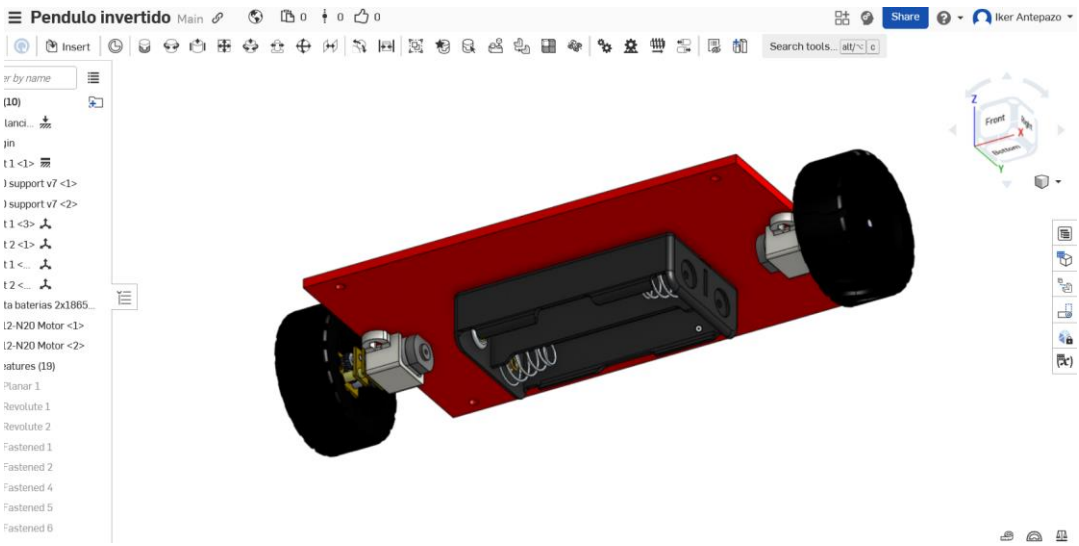
Peso centrado: Al igual que el centro de gravedad bajo, es decisivo que el peso del robot esté centrado en el eje de las ruedas, lo que hará que,

combinado con el apartado anterior, el robot debería mantenerse en la posición que queremos sin necesidad de que el robot esté encendido. Cuanto más estable sea el robot gracias al diseño del chasis, más sencillo nos será ajustar el PID hasta lograr que el robot funcione correctamente. Ahora voy a explicar los problemas de mis distintos chasis hasta el momento en que fui capaz de hacer que el robot funcionara.

Primer chasis:

Después de corregir los errores de los dos chasis mencionados por Daniel Acevedo en su documentación, logré hacer un chasis con el que robot funcionara. En este caso, el centro de gravedad estaba muy centrado y a una altura baja, lo que proporcionaba una gran estabilidad incluso cuando el robot estaba apagado. Pero no se adaptaba completamente con mis necesidades debido al uso de los motores antes mencionados.

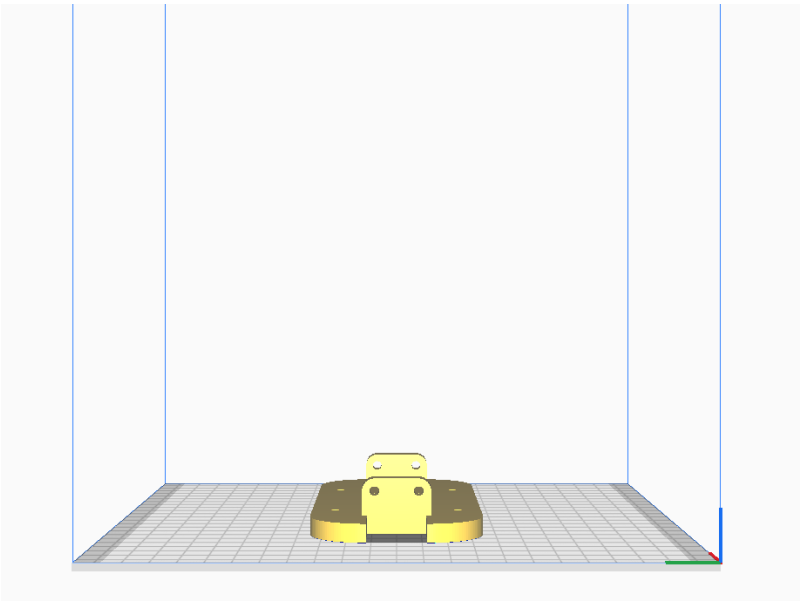




Este primer chasis lo he desarrollado a través de la página web de ”onshape” lo que me permitió diseñar incluso el portapilas pero se me hizo muy duro y difícil terminar con este proceso con lo cual para el segundo chasis me pasé a FreeCad.

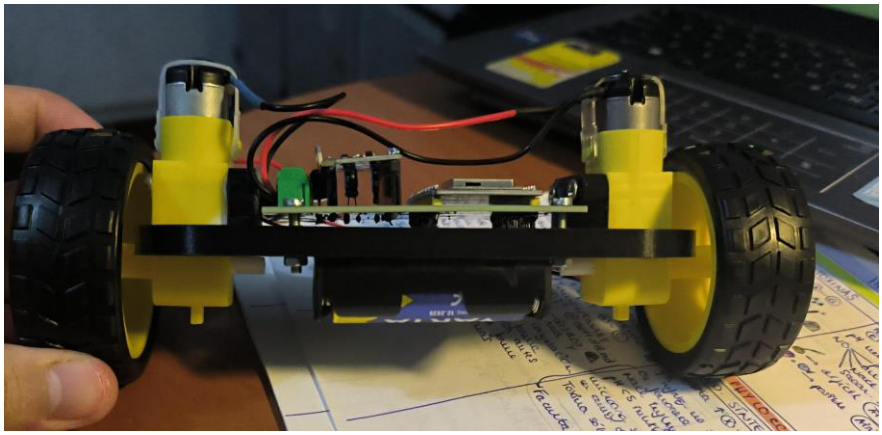
Segundo chasis:

Este segundo chasis lo he diseñado con la ayuda de FreeCad, le he agregado calculando las dimensiones de la placa PCB unos agujeros para permitirnos atornillar la placa al chasis con tuercas y tornillos asegurándonos así la fijación de esta. Al igual que esta novedad también le he agregado unas elevaciones en los laterales de la placa para poder atornillar nuestros motores en posición vertical permitiéndonos de este modo emplear los motores tradicionales antes mencionados.



Este fue el diseño final del chasis ya que se adaptaba a la perfección con lo deseado, con lo cual generamos el archivo .gcode para enviar a la impresora 3d. Esto se hizo gracias a la aplicación Ultimaker Cura.

A partir de aquí la impresión duró 4 horas aproximadamente, pero tuve bastantes problemas con las impresoras del centro debido a que se quedaba atascado el material y había ciertos errores de calibración en la impresora empleada.



Prueba de componentes por separado

MPU-6050:

Realizamos una primera prueba para verificar que el sensor funcione

```
// Librerias I2C para controlar el mpu6050
// la libreria MPU6050.h necesita I2Cdev.h, I2Cdev.h necesita Wire.h
```

```
  Serial.print(ax); Serial.print("\t");
  Serial.print(ay); Serial.print("\t");
  Serial.print(az); Serial.print("\t");
  Serial.print(gx); Serial.print("\t");
  Serial.print(gy); Serial.print("\t");
  Serial.println(gz);

  delay(100);
}
```

Listing 1: Código para probar el MPU6050

```
void setup() {
  Serial.begin(57600);     //Iniciando puerto serial
  Wire.begin();            //Iniciando I2C
  sensor.initialize();     //Iniciando el sensor

  if (sensor.testConnection()) Serial.println("Sensor iniciado correctamente");
  else Serial.println("Error al iniciar el sensor");
}

void loop() {
  // Leer las aceleraciones y velocidades angulares
  sensor.getAcceleration(&ax, &ay, &az);
  sensor.getRotation(&gx, &gy, &gz);

  //Mostrar las lecturas separadas por un [tab]
  Serial.print("a[x y z] g[x y z]:\t");
```

Como vemos que el sensor nos proporciona medidas coherentes, procedemos a calibrar el sensor en la posición en la que estará midiendo.

```ino
// Librerias I2C para controlar el mpu6050
// la libreria MPU6050.h necesita I2Cdev.h, I2Cdev.h necesita Wire.h
#include "I2Cdev.h"
#include "MPU6050.h"
#include "Wire.h"

// La direcci n del MPU6050 puede ser 0x68 o 0x69, dependiendo
// del estado de AD0. Si no se especifica, 0x68 estar  implicito
MPU6050 sensor;

// Valores RAW (sin procesar) del acelerometro y giroscopio en los ejes x,y,z
int ax, ay, az;
int gx, gy, gz;

//Variables usadas por el filtro pasa bajos
long f_ax,f_ay, f_az;
int p_ax, p_ay, p_az;
long f_gx,f_gy, f_gz;
int p_gx, p_gy, p_gz;
int counter=0;

//Valor de los offsets
int ax_o,ay_o,az_o;
int gx_o,gy_o,gz_o;

void setup() {
  Serial.begin(57600);   //Iniciando puerto serial
  Wire.begin();           //Iniciando I2C
  sensor.initialize();    //Iniciando el sensor

  if (sensor.testConnection()) Serial.println("Sensor iniciado correctamente");

  // Leer los offset los offsets anteriores
  ax_o=sensor.getXAccelOffset();
  ay_o=sensor.getYAccelOffset();
  az_o=sensor.getZAccelOffset();
  gx_o=sensor.getXGyroOffset();
  gy_o=sensor.getYGyroOffset();
  gz_o=sensor.getZGyroOffset();

  Serial.println("Offsets:");
  Serial.print(ax_o); Serial.print("\t");
  Serial.print(ay_o); Serial.print("\t");
  Serial.print(az_o); Serial.print("\t");
  Serial.print(gx_o); Serial.print("\t");
  Serial.print(gy_o); Serial.print("\t");
  Serial.print(gz_o); Serial.print("\t");
  Serial.println("nnEnvie cualquier caracter para empezar la calibracionnn");
  // Espera un caracter para empezar a calibrar
  while (true){if (Serial.available()) break;}
  Serial.println("Calibrando, no mover IMU");

}

void loop() {
  // Leer las aceleraciones y velocidades angulares
  sensor.getAcceleration(&ax, &ay, &az);
```

```
  sensor.getRotation(&gx, &gy, &gz);

  // Filtrar las lecturas
  f_ax = f_ax-(f_ax>>5)+ax;
  p_ax = f_ax>>5;

  f_ay = f_ay-(f_ay>>5)+ay;
  p_ay = f_ay>>5;

  f_az = f_az-(f_az>>5)+az;
  p_az = f_az>>5;

  f_gx = f_gx-(f_gx>>3)+gx;
  p_gx = f_gx>>3;

  f_gy = f_gy-(f_gy>>3)+gy;
  p_gy = f_gy>>3;

  f_gz = f_gz-(f_gz>>3)+gz;
  p_gz = f_gz>>3;

  //Cada 100 lecturas corregir el offset
  if (counter==100){
    //Mostrar las lecturas separadas por un [tab]
    Serial.print("promedio:"); Serial.print("t");
    Serial.print(p_ax); Serial.print("\t");
    Serial.print(p_ay); Serial.print("\t");
    Serial.print(p_az); Serial.print("\t");
    Serial.print(p_gx); Serial.print("\t");
    Serial.print(p_gy); Serial.print("\t");
    Serial.println(p_gz);

    //Calibrar el acelerometro a 1g en el eje z (ajustar el offset)
    if (p_ax>0) ax_o--;
    else {ax_o++;}
    if (p_ay>0) ay_o--;
    else {ay_o++;}
    if (p_az-16384>0) az_o--;
    else {az_o++;}

    sensor.setXAccelOffset(ax_o);
    sensor.setYAccelOffset(ay_o);
    sensor.setZAccelOffset(az_o);

    //Calibrar el giroscopio a 0  /s en todos los ejes (ajustar el offset)
    if (p_gx>0) gx_o--;
    else {gx_o++;}
    if (p_gy>0) gy_o--;
    else {gy_o++;}
    if (p_gz>0) gz_o--;
    else {gz_o++;}

    sensor.setXGyroOffset(gx_o);
    sensor.setYGyroOffset(gy_o);
    sensor.setZGyroOffset(gz_o);

    counter=0;
  }
  counter++;
}
```

Listing 2: Código para calibrar el MPU6050

Finalmente, probamos el código que utilizaremos para medir el ángulo de inclinación del robot.

```
// Librerias I2C para controlar el mpu6050
// la libreria MPU6050.h necesita I2Cdev.h, I2Cdev.h necesita Wire.h
#include "I2Cdev.h"
#include "MPU6050.h"
#include "Wire.h"

// La direcci n del MPU6050 puede ser 0x68 o 0x69, dependiendo
// del estado de AD0. Si no se especifica, 0x68 estar  implicito
MPU6050 sensor;

// Valores RAW (sin procesar) del acelerometro y giroscopio en los ejes x,y,z
int ax, ay, az;
int gx, gy, gz;
long tiempo_prev;
float dt;
float ang_x, ang_y;
float ang_x_prev, ang_y_prev;

void setup() {
  Serial.begin(57600);     //Iniciando puerto serial
  Wire.begin();            //Iniciando I2C
  sensor.initialize();     //Iniciando el sensor

  if (sensor.testConnection()) Serial.println("Sensor iniciado correctamente");
  else Serial.println("Error al iniciar el sensor");
}

void loop() {
  // Leer las aceleraciones y velocidades angulares
  sensor.getAcceleration(&ax, &ay, &az);
  sensor.getRotation(&gx, &gy, &gz);

  dt = (millis()-tiempo_prev)/1000.0;
  tiempo_prev=millis();

  //Calcular los  ngulos  con acelerometro
  float accel_ang_x=atan(ay/sqrt(pow(ax,2) + pow(az,2)))*(180.0/3.14);
  float accel_ang_y=atan(-ax/sqrt(pow(ay,2) + pow(az,2)))*(180.0/3.14);

  //Calcular angulo de rotaci n con giroscopio y filtro complemento
  ang_x = 0.98*(ang_x_prev+(gx/131)*dt) + 0.02*accel_ang_x;
  ang_y = 0.98*(ang_y_prev+(gy/131)*dt) + 0.02*accel_ang_y;


  ang_x_prev=ang_x;
  ang_y_prev=ang_y;

  //Mostrar los angulos separadas por un [tab]

  Serial.print("Rotacion en X:  ");
  Serial.print(ang_x);
  Serial.print("tRotacion en Y: ");
  Serial.println(ang_y);

  delay(10);
}
```

Listing 3: Código completo MPU 6050

TB6612FNG:

Al igual que los códigos del sensor anterior, los he modificado antes de subirlos a la ESP32, ajustando los pines para que sean compatibles con la ESP32:

```
const int pinPWMA = 6;
const int pinAIN2 = 7;
const int pinAIN1 = 8;
const int pinBIN1 = 9;
const int pinBIN2 = 10;
const int pinPWMB = 11;
const int pinSTBY = 12;
const int waitTime = 2000;  //espera entre fases
const int speed = 200;     //velocidad de giro
const int pinMotorA[3] = { pinPWMA, pinAIN2, pinAIN1 };
const int pinMotorB[3] = { pinPWMB, pinBIN1, pinBIN2 };

enum moveDirection {
  forward,
  backward
};

enum turnDirection {
  clockwise,
  counterClockwise
};

void setup()
{
  pinMode(pinAIN2, OUTPUT);
  pinMode(pinAIN1, OUTPUT);
  pinMode(pinPWMA, OUTPUT);
  pinMode(pinBIN1, OUTPUT);
  pinMode(pinBIN2, OUTPUT);
  pinMode(pinPWMB, OUTPUT);
}

void loop()
{
  enableMotors();
  move(forward, 180);
  delay(waitTime);

  move(backward, 180);
  delay(waitTime);

  turn(clockwise, 180);
  delay(waitTime);

  turn(counterClockwise, 180);
  delay(waitTime);

  fullStop();
  delay(waitTime);
}

//Funciones que controlan el vehiculo
void move(int direction, int speed)
{
  if (direction == forward)
  {
    moveMotorForward(pinMotorA, speed);
    moveMotorForward(pinMotorB, speed);
  }
  else
  {
    moveMotorBackward(pinMotorA, speed);
    moveMotorBackward(pinMotorB, speed);
  }
}

void turn(int direction, int speed)
{
  if (direction == forward)
  {
    moveMotorForward(pinMotorA, speed);
    moveMotorBackward(pinMotorB, speed);
  }
  else
  {
    moveMotorBackward(pinMotorA, speed);
    moveMotorForward(pinMotorB, speed);
  }
}

void fullStop()
{
  disableMotors();
```

```
  stopMotor(pinMotorA);
  stopMotor(pinMotorB);
}

//Funciones que controlan los motores
void moveMotorForward(const int pinMotor[3], int speed)
{
  digitalWrite(pinMotor[1], HIGH);
  digitalWrite(pinMotor[2], LOW);

  analogWrite(pinMotor[0], speed);
}

void moveMotorBackward(const int pinMotor[3], int speed)
{
  digitalWrite(pinMotor[1], LOW);
  digitalWrite(pinMotor[2], HIGH);

  analogWrite(pinMotor[0], speed);
}

void stopMotor(const int pinMotor[3])
{
  digitalWrite(pinMotor[1], LOW);
  digitalWrite(pinMotor[2], LOW);

  analogWrite(pinMotor[0], 0);
}

void enableMotors()
{
  digitalWrite(pinSTBY, HIGH);
}

void disableMotors()
{
  digitalWrite(pinSTBY, LOW);
}
```

Listing 4: Código prueba del driver y motores

Montaje

El montaje es muy simple, debemos conectar todo de la manera en la que hemos diseñado la PCB. Además de las conexiones de los sensores, necesitamos alimentar los motores y la ESP32. Ambos trabajan a voltajes distintos, por lo que requerimos un regulador de voltaje. Los motores los he alimentado a 9V, y la ESP32 se alimenta a 5V en el pin 5V.

Ajuste del PID

Para poder ajustar el PID, es necesario comprender la función de cada una de las tres componentes de este sistema de control.

Empezar desde 0:

Es crucial comenzar con todos los valores desde 0 y ajustar los valores de manera ordenada e independiente.

Orden de calibración:

Se recomienda iniciar ajustando los valores de Kp hasta lograr que el robot sea capaz de levantarse desde el suelo. Una vez logrado esto, se anota el valor optimo y se establece la variable a 0.

Luego, se ajusta Kd, buscando un valor en el cual el robot no oscile exageradamente y casi se mantenga en la posición deseada.

En el tercer paso, se restablece Kp al valor anotado anteriormente y se deja Kd en el valor optimo encontrado. En este punto, Ki aun se mantiene en 0, pero primero se verifica que el robot casi se mantenga en la posición deseada.

Finalmente, se ajusta Ki. Esta componente solo ayuda al robot a alcanzar la posición precisa, si las otras dos componentes no se han ajustado correctamente, Ki no cambiara radicalmente el comportamiento del robot. En resumen, Kp y Ki son las más importantes.

Valores pequeños:

Los valores del PID dependen en gran medida de la construcción del robot. Es importante tener en cuenta que cuanto mayores sean los valores, más agresivo será el movimiento del robot. Se recomienda comenzar con valores menores a 1 y realizar incrementos pequeños para evitar que el robot se mueva bruscamente.

Programación inicial

Al combinar los códigos previamente agregados a esta documentación con un control PID, podemos lograr que el robot funcione. Sin embargo, es necesario ajustar los valores del PID para que se adapten a nuestro caso específico.

En este punto fue donde en mi caso el robot dejó de funcionar, esto debido a que debe haber alguna conexión en la placa que hace que la ESP32 se quede congelada y no acabe de enviar el programa el componente pertinente. Trataré de solucionar este problema más adelante por mi cuenta, pero debido al cierre de actas cercano aquí es dónde me he quedado.

El código que deberíamos emplear sería:

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

MPU6050 sensor;

// Definiciones de pines de motores
const int pinPWMA = 16;
const int pinAIN2 = 17;
const int pinAIN1 = 18;
const int pinPWMB = 22;
const int pinBIN1 = 19;
const int pinBIN2 = 21;
const int pinSTBY = 23;


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

// Variables PID
float Kp = 0; // Ajusta estos 3 valores seg n sea necesario
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);
  sensor.initialize();

  // 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);
}

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

  // Calcular  ngulos  con el aceler metro
  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  ngulo  de rotaci n con giroscopio y filtro complementario
  dt = (millis() - tiempo_prev) / 1000.0;
  tiempo_prev = millis();
  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 p ndulo invertido usando PID
  float desired_angle = 0.0;  //  ngulo  deseado para mantener el equilibrio
  float error = desired_angle - ang_x;  // Calcular error

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

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

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

  // Mostrar los  ngulos  y la velocidad de los motores
  Serial.print("Rotacion en X: ");
  Serial.print(ang_x);
  Serial.print(", Error: ");
  Serial.print(error);
  Serial.print(", Velocidad de los motores: ");
  Serial.println(motor_speed);

  // Actualizar el error anterior
  previous_error = error;

  delay(1); // Reducir el tiempo de espera para aumentar la frecuencia de muestreo
}

// Funciones adicionales
void moveMotor(int pinPWMA, int pinIN1, int pinIN2, float speed) {
  if (speed > 0) {
    digitalWrite(pinIN1, HIGH);
    digitalWrite(pinIN2, LOW);
    analogWrite(pinPWMA, speed);
  } else {
    digitalWrite(pinIN1, LOW);
    digitalWrite(pinIN2, HIGH);
    analogWrite(pinPWMA, -speed);
  }
}

void enableMotors() {
  digitalWrite(pinSTBY, HIGH);
}
```

Listing 5: Código inicial completo