Universidad Nacional Autónoma de México

Facultad de Ingeniería

Departamento de Ingeniería Mecatrónica

Control Automático

Robot Péndulo Invertido
Entrega Final del Proyecto

Profesor:

Dr. Edmundo Gabriel Rocha Cozátl

Alumnos:

Diaz Carmona Javier Axel

Mujica Zeballos Carla

Nahuatlato Juárez Wilbert Miguel

Equipo 5incontroles *Grupo: 3 - Semestre: 2025-1*

Fecha de entrega: 3 de noviembre de 2024

Índice

1. Descripción del proyecto

Robot balancín/Robot péndulo invertido

El robot balancín es un proyecto que integra una estructura móvil, la cual se trata de estabilizar en una posición angular deseada, por lo que nos apoyaremos del acelerómetro y giroscopio MPU 6050 para hacer las lecturas de las variables necesarias. Mientras que el control PID nos permite hacer los ajustes continuamente para minimizar el error entre la posición angular actual y la posición angular deseada.

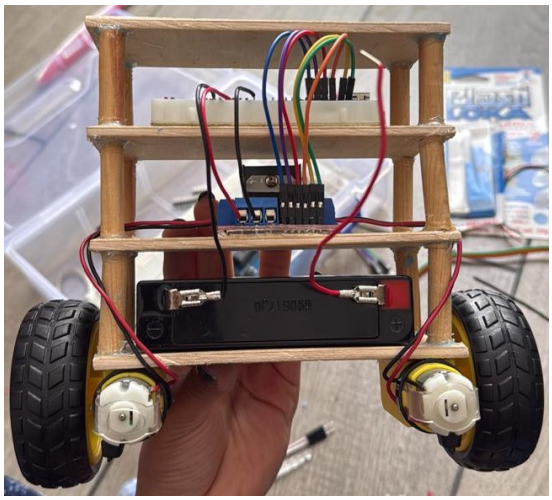

Figura 1: Robot Pendulo Invertido

El Arduino Nano es ideal para este tipo de proyecto dado su tamaño compacto y bajo costo. Ofrece las mismas capacidades que el Arduino Uno, pero en un formato mucho más pequeño, lo cual resulta importante para mantener el robot ligero y compacto. Cuenta con los suficientes pines de entrada y salida para controlar los motores y leer los datos del acelerómetro MPU 6050, lo cual lo hace compatible con nuestro diseño. Como alternativas encontramos la Raspberry Pi y la ESP 32.

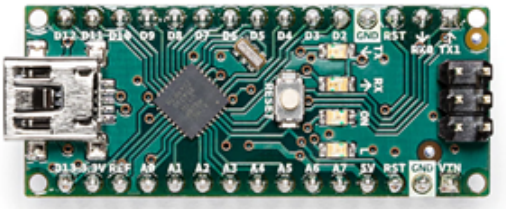

Figura 2: Arduino Nano

El MPU 6050 es una gran selección gracias a que incorpora un acelerómetro y un giroscopio en un mismo paquete lo cual simplifica el diseño. Este sensor es ampliamente utilizado en proyectos de equilibrio dinámico debido a su precisión y bajo costo. Nos proporciona la información necesaria para calcular la inclinación y la velocidad angular, esenciales para mantener el equilibrio del robot balancín.

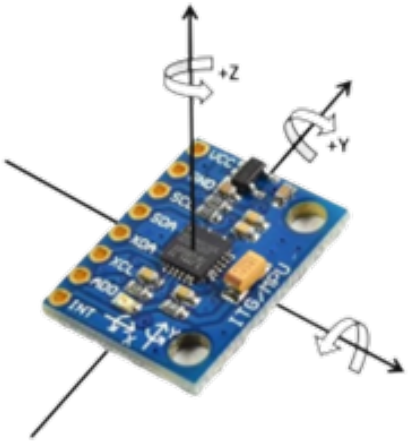

Figura 3: MPU6050

El interruptor de encendido y apagado nos permite controlar de una manera simple la alimentación lo cual es importante para evitar consumir la batería cuando el robot no está en uso. Además de ser un componente estándar, de bajo costo y fácil de integrar.

Figura 4: Interruptor ON/OFF

La Protoboard nos permite realizar todas las conexiones necesarias sin tener que soldar los componentes lo cual nos facilita hacer ajustes y modificaciones en el diseño, podemos seleccionar el tamaño necesario para el número de conexiones que el robot requiere manteniendo el diseño limpio

y compacto.
El módulo L298N puente H es un controlador de motores que puede manejar hasta 2 motores de corriente continua y controlar dirección y velocidad de ellos. Es una opción robusta y económica, con capacidad para manejar los voltajes y Corrientes típicos de los motores reductores utilizados en el diseño.

Figura 5: Puente H L298N 2A

El conector de la batería de 9 V es necesario para conectar la fuente de alimentación principal al circuito. Es un componente fácil de usar y bastante económico. Este conector nos permitirá llevar a cabo la conexión de manera sencilla la alimentación para los componentes de bajo consumo como el Arduino y el sensor MPU 6050.

Los motorreductores seleccionados proporcionan un equilibrio entre velocidad y torque, lo cual es necesario para mover el robot con precisión y mantener el equilibrio. Las llantas incluidas facilitan la movilidad del robot y el sistema de reducción permite controlar la velocidad de manera efectiva, estos motores de corriente directa son más fáciles de controlar y más económicos lo cual buscamos para este proyecto.

Figura 6: Motorreductor

Los cables jumper nos permiten realizar todas las colecciones necesarias para el proyecto de una forma rápida y sencilla además de que son económicos y reutilizables, nos facilitarán el montaje en la propia voz además de que son bastante flexibles en caso de necesitar algún ajuste a lo largo del montaje y las pruebas.

Figura 7: Cables Jumper

La tabla triplay y los palos de madera nos proporciona una estructura ligera y resistente para montar todos los componentes del robot la madera es un material fácil de trabajar, barato y suficientemente resistente para hacer nuestro prototipo o inclusive diseño final de nuestro proyecto.

Figura 8: Medidas de Triplay

La batería de 9 V alimenta los componentes de bajo consumo como lo es el argentino y el sensor MPU 6050, lo cual reduce la carga de la batería principal del robot resulta en una fuente de energía común y fácil de reemplazar.

Finalmente, la batería Lipo de 3S proporciona suficiente voltaje (3-12 [V] [Para este caso 11.1[V]]) y corriente suficiente para alimentar los motores, necesario para obtener la fuerza requerida en los motores reductores.

Figura 9: Batería Lipo 500mAh

Este tipo de batería es ligera y tiene una alta densidad de energía lo cual lo hace versátil en aplicaciones de robótica móvil.

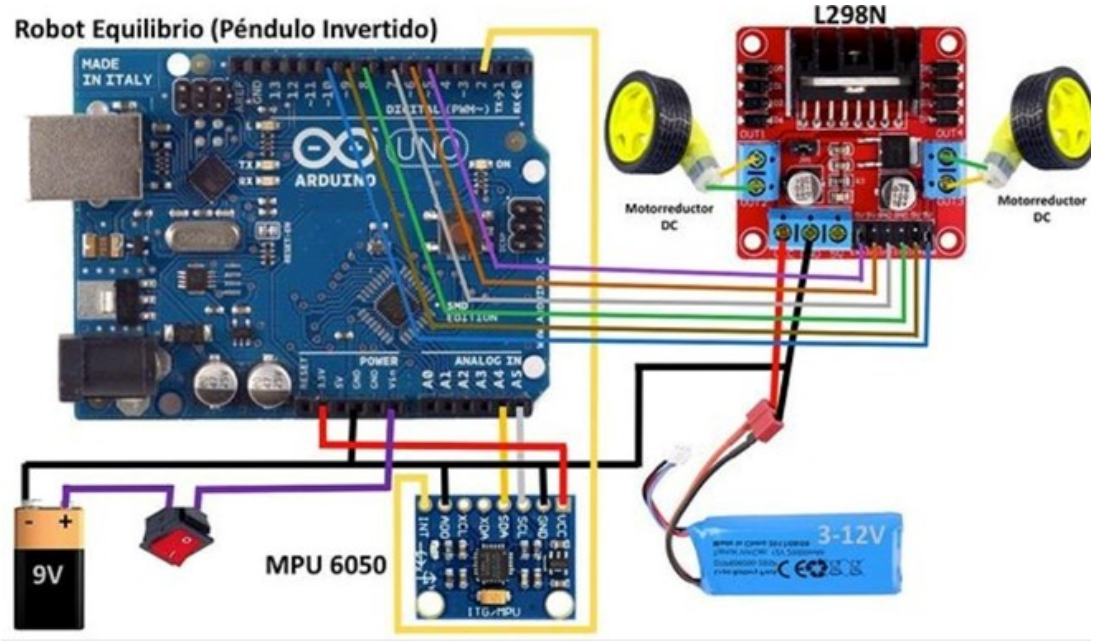

Figura 10: Diagrama de conexiones

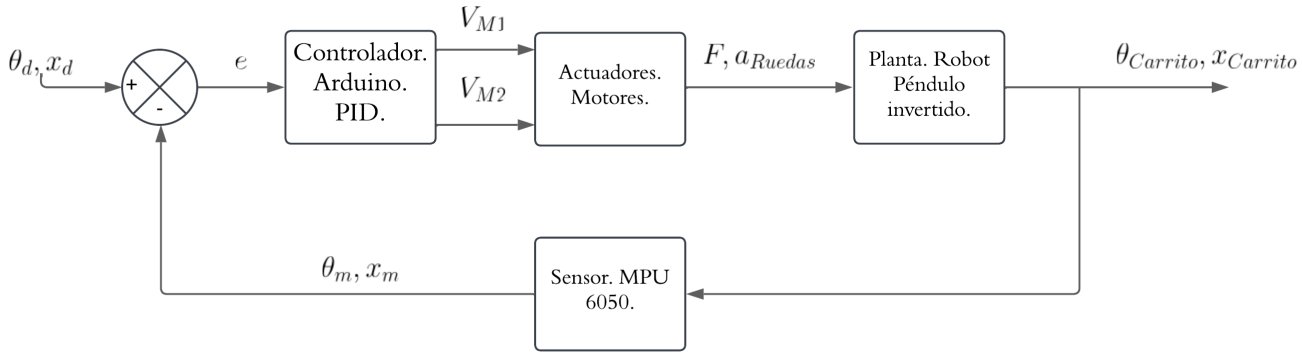

Figura 11: Diagrama de Bloques Del Sistema

2. Objetivos

Estabilidad

La estabilidad es el objetivo primario del sistema, especialmente para mantener el péndulo en posición vertical. Con un controlador bien diseñado, como un PID, el sistema puede ser estabilizado en condiciones ideales.

Regulación

La regulación implica mantener el péndulo en una posición deseada y el carrito en un punto fijo. Este objetivo es alcanzable con un buen diseño del controlador.

Desempeño

El desempeño se mide en términos de tiempo de respuesta, sobrepaso y error estacionario. Un controlador bien ajustado puede optimizar estos parámetros.

3. Materiales

3.1. Arduino Nano

- Microcontrolador basado en el ATmega328. Cerebro del robot, encargado de procesar las señales del acelerómetro-giroscopio y controlar los motores.
- **Voltaje de operación:** 5 [V].
- **Voltaje de entrada (a través de V_in):** 7-12 [V].
- **Pines digitales:** 20 (6 PWM). **Pines analógicos:** 8.
- **Memoria Flash:** 32 kB (2 kB bootloader).
- **Frecuencia de reloj:** 16 MHz.
- Soporta protocolos de comunicación **I2C, UART y SPI**.

3.2. Acelerómetro y giroscopio MPU6050

- Sensor que proporciona datos de aceleración y rotación en 3 ejes, fundamental para mantener el equilibrio del robot balancín.
- **Rango de aceleración:** ±2g, ±4g, ±8g, ±16g.
- **Rango de giroscopio:** ±250, ±500, ±1000, ±2000 grados/s.
- **Interfaz:** I2C.
- **Voltaje de operación:** 3.3 [V] o 5 [V].
- **Precisión:** 16 bits por canal (3 para aceleración y 3 para giro).

3.3. Interruptor On/Off

- Dispositivo mecánico para desconectar o conectar la alimentación del robot.
- **Voltaje y corriente:** 3 A/250 V-AC o 6 A/125 V-AC.
- **Terminales:** 2.
- **Acción:** Un polo, un tiro.

3.4. Protoboard Mediana

- Base para conectar temporalmente los componentes del circuito sin necesidad de soldadura.
- **Dimensiones:** 84 x 53.3 x 8.3 [mm].
- **Rango:** 36 [V]/2 [A].
- **Puntos de conexión:** 400.

3.5. Módulo L298N - Puente H

- Controlador de motores que permite controlar la dirección y velocidad de los motores del robot.

- **Dimensiones:** 42 x 43 x 27 [mm].
- **Voltaje:** 5-35 [V].
- **Número de canales:** 2.
- **Capacidad de corriente:** 2 A (picos de hasta 3 A).

3.6. Conector de batería de 9V

- Adaptador para conectar la batería de 9V al circuito eléctrico del robot.
- **Tipo de conector:** Snap-on de 9V.

3.7. Motorreductores con llantas (2 unidades)

- Motores con sistema de engranajes que proporcionan fuerza y control de velocidad a las llantas del robot.
- **Reducción:** 120:1.
- **Corriente sin carga:**
  - 3 V: 80 mA.
  - 5 V: 100 mA.
  - 6 V: 130 mA.
- **Corriente máxima de paro (eje detenido):**
  - 3 V: 500 mA.
  - 5 V: 850 mA.
  - 6 V: 950 mA.
- **Velocidad sin carga (aproximada):** 90 RPM.

- **Torque máximo:** 1.4 kg.cm (dependiendo del voltaje y la carga aplicada al eje).
- **Diámetro del eje de salida:** 5.3 mm (aplanado a 3.6 mm).

3.8. Cables Jumper

- **Tipos:** Macho-Macho (10 unidades) y Hembra-Macho (15 unidades).
- **Material:** Cobre/Plástico ABS.
- **Longitudes:** 20 cm y 30 cm.
- **Calibre:** 26 AWG.
- Espaciado estándar de 0.1 (2.54 mm) entre conexiones.
- Excelente conductividad eléctrica y flexibilidad.

3.9. Tabla Triplay y Palos de Madera

- **Tabla Triplay:**
  - Dimensiones: 100 x 50 mm.
  - Espesor: 3 mm.
  - Material: Madera contrachapada.
- **Palos de madera:**
  - Longitud: 50 mm.
  - Material: Madera.

3.10. Baterías

- **Batería de 9V:**
  - Voltaje: 9 V.

- Capacidad nominal: 150 mAh.
- Dimensiones: 4.8 cm x 2.6 cm x 1.7 cm.

- **Batería Lipo 3S:**
  - Voltaje: 11.1 V.
  - Capacidad: 500 mAh.
  - Capacidad de descarga: 65C.
  - Celdas: 3S.
  - Conector: XT30.
  - Dimensiones: 57 x 30 x 13 mm.

4. Desarrollo

4.1. Estructura: Armazón del robot

Como estudiantes de mecatrónica, consideramos que el diseño y los materiales utilizados para la estructura son fundamentales para el funcionamiento del proyecto. A continuación, se detalla los aspectos más relevantes sobre la construcción del esqueleto del robot:

4.1.1. Materiales

La madera, especialmente Tryplay, fue seleccionada por su ligereza y facilidad de corte, características que permiten una estructura funcional y maniobrable. La silicona caliente se utilizó como adhesivo, proporcionando una unión sólida y confiable entre las piezas.

4.1.2. Diseño

El diseño de la estructura tuvo en cuenta los siguientes aspectos clave:

- **Resistencia:** La estructura debe soportar el peso de los componentes electrónicos y los motores.
- **Ligereza:** Es necesario que el peso total sea lo suficientemente bajo para facilitar el movimiento

y equilibrio del robot.

- **Centro de gravedad:** Ubicar el centro de gravedad en una posición óptima es crucial para que el robot se balancee de manera efectiva.
- **Estabilidad:** Se diseñó una base con suficiente ancho para evitar vuelcos inesperados.
- **Accesibilidad:** Se priorizó un diseño que permita realizar modificaciones o reparaciones futuras de forma sencilla.

4.2. Conexiones: El cerebro y los nervios del robot

El sistema electrónico es el núcleo principal del robot. Se organizó cuidadosamente para garantizar una conexión eficiente y ordenada de los componentes.

4.2.1. Protoboard

La protoboard sirve como un "panal"donde se realizan las conexiones temporales de los componentes, permitiendo flexibilidad y rapidez durante el desarrollo y pruebas.

4.2.2. Arduino Nano

El Arduino Nano actúa como el cerebro del robot. A través de sus pines digitales y analógicos, conecta y controla los demás componentes, interpretando las señales del sensor y enviando comandos al driver de motores.

4.2.3. MPU6050

Este acelerómetro-giroscopio proporciona datos en tiempo real sobre la inclinación y velocidad angular del robot, esenciales para mantener el equilibrio. Se conecta al Arduino mediante el protocolo I2C.

4.2.4. Driver de Motor L298N

El módulo L298N controla la dirección y velocidad de los motores del robot. Recibe señales del Arduino y las traduce en movimiento.

4.2.5. Motores reductores

Los motores reductores proporcionan la fuerza necesaria para mover el robot y ajustar su posición para mantener el equilibrio.

4.3. Programación

La programación es una de las etapas más importantes del desarrollo del robot, ya que permite transformar los datos del sensor y las órdenes de control en acciones físicas. Este proyecto utiliza un controlador PID y un sensor MPU6050 para mantener el equilibrio del robot. A continuación, se describe el funcionamiento del sistema, incluyendo las bibliotecas utilizadas y las secciones clave del código implementado.

4.3.1. Descripcion General del Codigo

El codigo esta disenado para controlar un sistema de balanceo, como un robot de dos ruedas, utilizando el sensor MPU6050 para la lectura de inclinaciones y un controlador PID para ajustar la velocidad de los motores en funcion de los datos de inclinacion (*pitch*).

Listing 1: Codigo para el control de un sistema de balanceo con PID y MPU6050

```
#include <PID_v1.h>
#include <LMotorController.h>
#include "I2Cdev.h"
#include "MPU6050_6Axis_MotionApps20.h"
#include <SoftwareSerial.h>
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
#include "Wire.h"
#endif
#define MIN_ABS_SPEED 30
MPU6050 mpu;

//************************************* Ajustes
    *************************************
double MotorVelocidadIzq = 0.5; // Velocidad del motor izquierdo (0.5 es
    un valor de ejemplo)
```

```
double MotorVelocidadDer = 0.5; // Velocidad del motor derecho (0.5 es
    un valor de ejemplo)
double PuntoEquilibrio = 180.8; // Angulo de equilibrio en grados (valor
     ideal para mantener el equilibrio)
SoftwareSerial Serial_2 (4, 3); // Configuracion del puerto serial para
    comunicacion Bluetooth (pines 4 y 3)
//--------------------Control de Motores
int ENA = 5; // Pin para habilitar el motor izquierdo
int IN1 = 6; // Pin de direccion del motor izquierdo
int IN2 = 7; // Pin de direccion del motor izquierdo
int IN3 = 9; // Pin de direccion del motor derecho
int IN4 = 8; // Pin de direccion del motor derecho
int ENB = 10; // Pin para habilitar el motor derecho
//--------------------Los valores de PID cambian con cada disenio
double Kp = 60;  // Coeficiente proporcional del PID
double Kd = 3.5;  // Coeficiente derivativo del PID
double Ki = 250;  // Coeficiente integral del PID

//
    ************************************************************************

int estado = 'g';           // Inicializa el estado del robot como
    detenido

// MPU control/status vars
bool dmpReady = false; // Indica si la inicializacion del DMP fue
    exitosa
uint8_t mpuIntStatus; // Almacena el estado de la interrupcion del MPU
uint8_t devStatus; // Almacena el estado despues de cada operacion del
    dispositivo (0 = exito, != 0 = error)
uint16_t packetSize; // Tamano esperado del paquete DMP (por defecto es
    de 42 bytes)
uint16_t fifoCount; // Cuenta los bytes actualmente en el FIFO
```

```
uint8_t fifoBuffer[64]; // Buffer de almacenamiento para los datos del
    FIFO

// Variables de orientacion y movimiento
Quaternion q; // Contenedor de quaternion [w, x, y, z]
VectorFloat gravity; // Vector de gravedad [x, y, z]
float ypr[3]; // Contenedor para [yaw, pitch, roll] y el vector de
    gravedad

// Variables PID
double originalSetpoint = PuntoEquilibrio;    // Angulo de equilibrio
    original
double setpoint = originalSetpoint; // Angulo de referencia (se puede
    ajustar durante la operacion)
double movingAngleOffset = 0.1; // Desviacion del angulo de movimiento
double input, output; // Variables para la entrada y salida del PID

// Configuracion del PID
PID pid(&input, &output, &setpoint, Kp, Ki, Kd, DIRECT);

double motorSpeedFactorLeft = MotorVelocidadIzq; // Factor de velocidad
    para el motor izquierdo
double motorSpeedFactorRight = MotorVelocidadDer; // Factor de velocidad
     para el motor derecho

// Controlador de los motores
LMotorController motorController(ENA, IN1, IN2, ENB, IN3, IN4,
    motorSpeedFactorLeft, motorSpeedFactorRight);

// Interrupcion para el MPU
volatile bool mpuInterrupt = false; // Indica si el pin de interrupcion
    del MPU se ha activado
void dmpDataReady()
{
```

```
 mpuInterrupt = true; // Marca la interrupcion cuando hay nuevos datos
    listos
}

void setup()
{
    Serial_2.begin(9600);     // Inicia la comunicacion serial con el
       Bluetooth
    //Serial.begin(9600);
    // Inicia el bus I2C (la libreria I2Cdev no lo hace automaticamente)
    #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
    Wire.begin();
    TWBR = 24; // Configura la frecuencia del reloj I2C a 400kHz (200kHz
        si el CPU es de 8MHz)
    #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
    Fastwire::setup(400, true);
    #endif

    mpu.initialize(); // Inicializa el sensor MPU6050

    devStatus = mpu.dmpInitialize(); // Inicializa el DMP del MPU

    // Configura los offsets del giroscopio, para minimizar los errores
    mpu.setXGyroOffset(220);
    mpu.setYGyroOffset(76);
    mpu.setZGyroOffset(-85);
    mpu.setZAccelOffset(1788); // Offset para el acelerometro

    // Verifica si la inicializacion fue exitosa (devStatus debe ser 0
       si es exitoso)
    if (devStatus == 0)
    {
        // Activa el DMP ahora que esta listo
        mpu.setDMPEnabled(true);
```

```
        // Habilita la deteccion de interrupciones en Arduino
        attachInterrupt(0, dmpDataReady, RISING); // Interrupcion en el
            pin 0
        mpuIntStatus = mpu.getIntStatus(); // Obtiene el estado de la
            interrupcion

        // Marca la bandera de DMP listo para que la funcion loop()
            pueda usarlo
        dmpReady = true;

        // Obtiene el tamano del paquete esperado para futuras
            comparaciones
        packetSize = mpu.dmpGetFIFOPacketSize();

        // Configura el PID
        pid.SetMode(AUTOMATIC); // Modo automatico del PID
        pid.SetSampleTime(10); // Establece el tiempo de muestreo
        pid.SetOutputLimits(-255, 255); // Limita la salida del PID para
             que sea apropiada para los motores
    }
    else
    {
        // ERROR si la inicializacion falla
        Serial.print(F("Error en la inicializacion del DMP (codigo "));
        Serial.print(devStatus);
        Serial.println(F(")"));
    }
}

void loop()
{
    // Si la inicializacion fallo, no hace nada
    if (!dmpReady) return;
```

```
// Espera a que se active la interrupcion del MPU o que haya mas de
   un paquete de datos disponible
while (!mpuInterrupt && fifoCount < packetSize)
{
    // Si no hay datos del MPU, se realiza el calculo del PID y se
       mueve el robot
    pid.Compute();
    motorController.move(output, MIN_ABS_SPEED); // Mueve el robot
       con la salida del PID y una velocidad minima
}

// Restablece la bandera de interrupcion y obtiene el estado de la
   interrupcion
mpuInterrupt = false;
mpuIntStatus = mpu.getIntStatus();

// Obtiene el numero actual de elementos en el FIFO
fifoCount = mpu.getFIFOCount();

// Revisa si hay desbordamiento del FIFO (esto no deberia ocurrir a
   menos que el codigo sea ineficiente)
if ((mpuIntStatus & 0x10) || fifoCount == 1024)
{
    // Restablece el FIFO si hay desbordamiento
    mpu.resetFIFO();
    Serial.println(F("Desbordamiento de FIFO"));
}
// Si hay nuevos datos del DMP, los procesa
else if (mpuIntStatus & 0x02)
{
    // Espera a que haya suficientes datos disponibles
    while (fifoCount < packetSize) fifoCount = mpu.getFIFOCount();
```

```
        // Lee los datos del FIFO
        mpu.getFIFOBytes(fifoBuffer, packetSize);

        // Actualiza el contador de FIFO
        fifoCount -= packetSize;

        // Extrae el quaternion y la gravedad de los datos del FIFO
        mpu.dmpGetQuaternion(&q, fifoBuffer);
        mpu.dmpGetGravity(&gravity, &q);
        mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);

        // Calcula el angulo de inclinacion (pitch) y lo ajusta a grados
        input = ypr[1] * 180/M_PI + 180;
    }
}
```

Bibliotecas y Definiciones

- **Bibliotecas Importadas:**
  - `PID_v1.h`: Implementa el controlador PID.
  - `LMotorController.h`: Controla los motores mediante un puente H.
  - `MPU6050_6Axis_MotionApps20.h`: Integra el MPU6050 utilizando DMP (Digital Motion Processing).
  - `I2Cdev.h`: Facilita la comunicación I2C entre el microcontrolador y el sensor MPU6050.
  - `Wire.h`: Biblioteca estándar para el protocolo I2C.
- **Configuración de Pines:** Los pines `ENA`, `ENB`, `IN1`, `IN2`, `IN3`, `IN4` controlan un puente H para manejar la dirección y velocidad de los motores.
- **Constantes PID:**
  - `Kp`: Constante proporcional.

- `Ki`: Constante integral.
- `Kd`: Constante derivativa.

Estas constantes afectan la rapidez y estabilidad con la que el sistema ajusta su posición.

- **Punto de Equilibrio:** `PuntoEquilibrio = 180.8` representa el ángulo deseado para que el robot se mantenga balanceado.

Inicialización (`setup()`)

- **MPU6050:**
  - Inicializa el sensor y configura los offsets del giroscopio y acelerómetro.
  - Habilita el procesamiento DMP para obtener datos de orientación (Quaternion, Yaw, Pitch, Roll).
- **PID:**
  - Configurado en modo automático con límites de salida entre `-255` y `255`, correspondientes al rango de PWM para controlar los motores.
  - Frecuencia de cálculo definida en 10 ms (`SetSampleTime(10)`).
- **Interrupciones:** Configura una interrupción en el pin 2 para recibir notificaciones del MPU6050 cuando hay nuevos datos en el buffer FIFO.

Bucle Principal (`loop()`)

- **Validación del Estado del Sensor:** El programa verifica que el DMP esté listo y que haya datos disponibles en el FIFO.
- **Cálculo del PID:** Si no hay datos disponibles, el PID ajusta la salida con el valor actual de inclinación (`input`), y envía la salida al controlador de motores con `motorController.move(output, MIN_ABS_SPEED)`.
- **Lectura de Datos del MPU6050:** Si hay datos en el FIFO:

- Se leen y procesan.
- Calcula el *pitch* (inclinación) en grados y lo asigna a la variable `input`.
- Este valor se utiliza en el PID para ajustar la velocidad de los motores y mantener el equilibrio.

- **Control de Motores:** La salida del PID determina la dirección y velocidad de los motores mediante `motorController.move(output, MIN_ABS_SPEED)`.

Explicación del Control

- **Entrada del PID (`input`):** Corresponde a la inclinación actual del robot, medida por el MPU6050.
- **Salida del PID (`output`):** Es el valor calculado que ajusta la velocidad de los motores, positiva o negativa según la dirección necesaria para corregir la inclinación.
- **Setpoint:** Es el ángulo objetivo para el robot (punto de equilibrio). En este caso, 180.8°.
- **Controlador de Motores (`motorController`):** Ajusta la velocidad de los motores proporcionalmente a la salida del PID.

Notas Adicionales

- **Compensación de Desplazamiento:** `movingAngleOffset` ajusta el ángulo de equilibrio según pequeños movimientos del robot.
- **Manejo de Errores:** El código incluye validaciones para evitar desbordes en el buffer FIFO del MPU6050.
- **Tiempos Críticos:** La frecuencia del cálculo PID (`SetSampleTime(10)`) debe ser lo suficientemente rápida para evitar retrasos que comprometan la estabilidad del sistema.

Recomendaciones

- Afinar las constantes del PID (`Kp`, `Ki`, `Kd`) para mejorar la estabilidad y respuesta del sistema.

- Asegurarse de calibrar correctamente los offsets del MPU6050 para obtener lecturas precisas.
- Monitorear variables clave como `input`, `output` y `setpoint` mediante impresiones seriales para depuración.
- Implementar un sistema de seguridad para detener los motores en caso de fallos críticos, como pérdida de datos del MPU6050.

5. Conclusiones

Díaz Carmona Javier Axel. A lo largo de este proyecto se cumplen los objetivos de control establecidos en este documento, quizá no de una manera perfecta pero sí parcial y suficiente para entender la importancia de la implementación del control PID en sistemas mecatrónicos o afines. De manera que la investigación, construcción y programación de este proyecto nos aporta de buena manera para poder dar soluciones en el mundo real como futuro ingeniero. Tanto ver la importancia de manejar una buena selección de materiales como de conocer y saber aplicar los conceptos de la materia por medio de programación, aunque esta sea por librerías.

Mujica Zeballos Carla Fabiana. A lo largo del proyecto, enfocado en construir un carrito/robot péndulo invertido con un controlador proporcional-integral-derivativo (PID), se observó que la implementación de este controlador fue clave para lograr un funcionamiento que cumpla los objetivos planteados. La selección de materiales es clave para poder dar vida al proyecto y que responda de manera adecuada y esperada.

Nahuatlato Juárez Wilbert Miguel. El objetivo principal del proyecto fue encontrar la estabilidad en el robot péndulo invertido, que reaccionara de manera adecuada a los cambios de orientación. Para lograr esto, modificamos los valores de PID mediante un proceso de prueba y error. Al principio, con valores muy bajos para todos los parámetros, el sistema no mostraba variaciones en la respuesta. Sin embargo, al aumentar los valores de PID, notamos una mejora significativa en la respuesta, con una reducción de las oscilaciones y una mayor estabilidad. La implementación de un controlador PID en proyectos de control resultó ser esencial. Gracias a la librería de Arduino, pudimos ajustar fácilmente los parámetros de Kp, Ki y Kd, lo que facilitó obtener la respuesta deseada.

Referencias

[1] Arduino, "A000005: Datasheet," [Online]. Available: https://docs.arduino.cc/resources/datasheets/A000005-datasheet.pdf.

[2] W. Bolton, *Mecatrónica. Sistemas de control electrónico en la ingeniería mecánica y eléctrica. Un enfoque multidisciplinario*, 5ta ed., Alfaomega Grupo Editor, México, 2013. ISBN: 978-0-273-74286-9.

[3] TDK InvenSense, "MPU-6000 Register Map," [Online]. Available: https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Register-Map1.pdf.

[4] K. Ogata, *Ingeniería de Control Moderna*, PEARSON EDUCACIÓN, S.A., Madrid, 2010. ISBN: 978-84-8322-660-5.

[5] N. S. Nise, *Control Systems Engineering*, 6th ed., John Wiley & Sons, Inc., California State Polytechnic University, Pomona.

[6] SparkFun, "Breadboard Datasheet," [Online]. Available: https://cdn.sparkfun.com/datasheets/Prototyping/breadboard.pdf.

[7] Steren, "Pila Alcalina 9V Cuadrada," [Online]. Available: https://www.steren.com.mx/pila-alcalina-9v-cuadrada.html.

[8] Uelectronics, "Cable para Batería de 9V con Broche," [Online]. Available: https://uelectronics.com/producto/cable-para-bateria-de-9v-con-broche/.

[9] Uelectronics, "IMU MPU6050 - 6 Grados de Libertad," [Online]. Available: https://uelectronics.com/producto/imu-mpu6050-6-grados-de-libertad/.

[10] Uelectronics, "Interruptor On-Off Rojo, 2 Pines, 125V 6A," [Online]. Available: https://uelectronics.com/producto/interruptor-on-off-rojo-2-pines-125v-6a/.

[11] Uelectronics, "Motorreductor Amarillo para Carrito," [Online]. Available: https://uelectronics.com/producto/motorreductor-amarillo-para-carrito/.

[12] e-Robots, "Batería Lipo 500mAh 11.1V 65C," [Online]. Available: https://e-robots.com.mx/product/bateria-lipo-500mah-11-1v-65c/.

[13] YouTube, "Sistema de control PID explicado," [Online]. Available: https://youtu.be/gtsZ2hswKJk?si=rlIsuSi2YtN6orH_.

[14] YouTube, "Controladores PID desde cero," [Online]. Available: https://youtu.be/pjrxAMo-apA?si=4yQammBhFgCyqBgD.

[15] YouTube, "Construcción de un robot equilibrado," [Online]. Available: https://youtu.be/vUns-5b15Qc?si=UKnjyf_m8fjtsSpD.