INGENIERÍA MECATRÓNICA

IMEC

Universidad Católica Boliviana "San Pablo"

Departamento de Ciencias de la Tecnología e Innovación – Tarija

Práctica 04 – Comunicación Digital: I²C y SPI

Asignatura: IMT-222 Sistemas Embebidos I
Docente: E. Alan Cornejo Q.
Correo: ecornejo@ucb.edu.bo
Lugar / Fecha: Tarija, 03 de noviembre de 2025

Dificultad: Baja **Requisito:** Sin supervisión obligatoria

Alejandro Bejarano R. – Susan Baldiviezo C.- Benjamin Soruco G. – Florencia Frigerio A.

Formato UCB – Versión base (2025)

Índice

1. Objetivos

Implementar la comunicación digital entre un microcontrolador y un sensor mediante los protocolos I²C o SPI. Los objetivos específicos son:

- Configurar la interfaz de comunicación I²C entre un Arduino UNO (maestro) y el sensor MPU6050 (esclavo).
- Leer y mostrar los datos obtenidos desde un sensor digital en el monitor serial.
- Comprender el flujo de transmisión y recepción de datos a nivel de bit en el bus PC (condiciones START/STOP, dirección del esclavo, bits de datos y ACK/NACK).
- Documentar y explicar detalladamente el funcionamiento del protocolo I²C y del código modular implementado en los archivos MPU6050.h, MPU6050.cpp y practica04_MPU6050.ino

2. Materiales y Equipos

2.1. Hardware

- Placa de desarrollo Arduino UNO (ATmega328P, lógica a 5 V).
- Módulo IMU MPU6050 (acelerómetro y giroscopio de 3 ejes, comunicación
- I²C, dirección por defecto 0x68).
- Cables Dupont macho-hembra.
- Cable USB para programación y alimentación del Arduino

2.2. Software

- Entorno de desarrollo Arduino IDE.
- Software Wokwi para simulaciones.
- Librería estándar Wire.h para comunicación I²C.
- Monitor Serial de Arduino IDE configurado a 115200 baudios.

3. Metodología

El trabajo se desarrollará tomando como base la guía oficial de la práctica 04, adaptándola al caso concreto de la comunicación entre Arduino UNO y el sensor MPU6050 mediante I²C. Se sigue una estrategia modular: el código se divide en un archivo principal (.ino) y una librería propia compuesta por MPU6050.h y MPU6050.cpp.

3.1. Fase 1 — Configuración del entorno

Antes de iniciar la programación es recomendable comprobar que el sensor MPU6050 está correctamente conectado y responde en el bus I²C. Para ello se utiliza un escáner I²C, un programa simple que recorre todas las direcciones disponibles (1–127) enviando una condición START y verificando si algún dispositivo responde con un ACK.

Si la conexión es correcta, el escáner debe detectar al MPU6050 típicamente en la dirección 0x68, o 0x69 si el pin AD0 está en nivel alto. Esta verificación inicial permite asegurar que el sensor está activo, alimentado y listo para la comunicación, evitando errores posteriores durante la lectura de datos.

3.1.1 Conexión física del sensor al Arduino UNO

El sensor MPU6050 se comunica mediante el protocolo I²C por lo que su conexión al Arduino UNO requiere únicamente cuatro líneas básicas: alimentación, tierra y las líneas SDA y SCL. El módulo GY-521 (donde viene integrado el MPU6050) incorpora un regulador y conversión de nivel permitiendo su uso directo con los **5 V** del Arduino.

Las conexiones recomendadas son:

- VCC → 5V
- GND → GND
- SDA → A4 (línea de datos I²C)
- SCL → A5 (línea de reloj I²C)

La mayoría de módulos incluyen resistencias **pull-up** en SDA y SCL si no las tuviera, deben agregarse externamente para garantizar el funcionamiento del bus. Finalmente, es buena práctica ejecutar un **escáner I²C** para confirmar que el MPU6050 responde en la dirección **0x68**

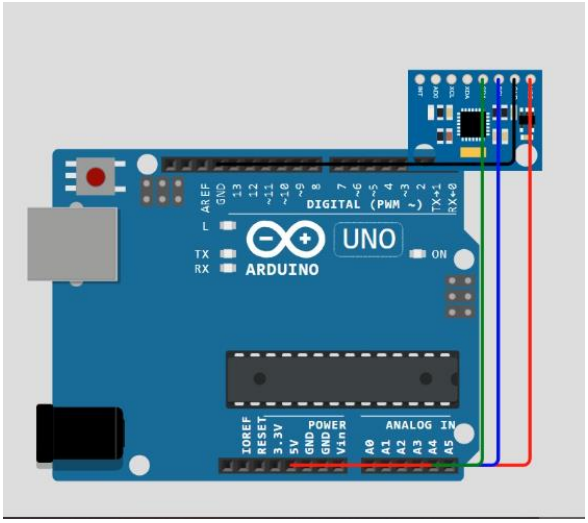


Figura 1: Conexión física entre el Arduino UNO y el sensor MPU6050 mediante bus I²C. Fuente: Elaboración propia.

3.1.2 Verificación del dispositivo en el bus I²C.

3.1.3 Arquitectura interna y funcionamiento del MPU6050

El MPU6050 es un sensor IMU de seis ejes que integra un acelerómetro triaxial, un giroscopio triaxial y un procesador digital de movimiento (DMP) dentro de una misma arquitectura. Cada eje posee su propio sistema de acondicionamiento analógico y un ADC de 16 bits, lo que permite obtener mediciones precisas y simultáneas de aceleración y velocidad angular.

Los datos convertidos se almacenan en un conjunto de registros internos, desde donde el microcontrolador puede leerlos a través del bus I²C. El sensor funciona como dispositivo esclavo y utiliza la dirección 0x68 o 0x69, dependiendo del estado del pin AD0. A través de este bus se configuran parámetros como los rangos de medición y se accede a los registros que contienen la información del acelerómetro, giroscopio y temperatura.

Gracias a esta arquitectura integrada y al uso de una interfaz de comunicación sencilla, el MPU6050 permite obtener datos de movimiento de forma eficiente, reduciendo la carga de procesamiento en el microcontrolador y facilitando su integración en sistemas embebidos, proyectos de control, robótica y estabilización.

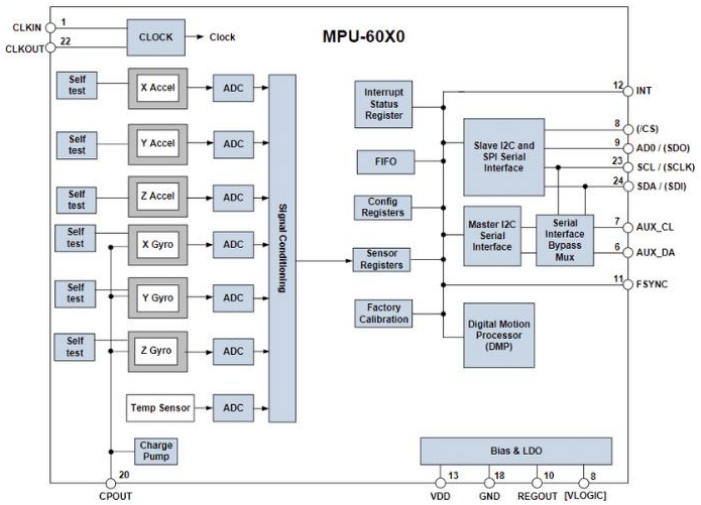


Figura 2: Arquitectura interna del sensor MPU-6050
Fuente: Datasheet MPU-6050

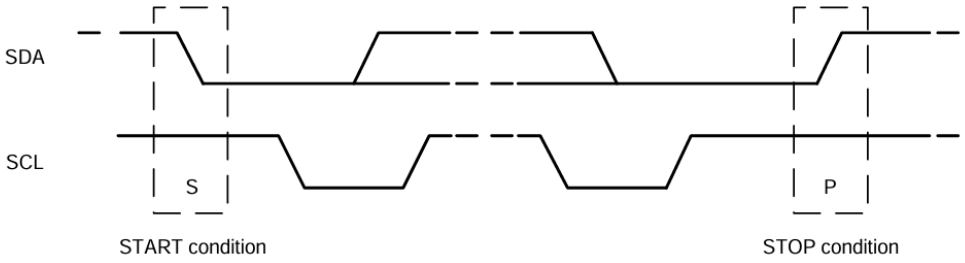


Figura 3: Condiciones de inicio y parada.

Fuente: Datasheet MPU6050

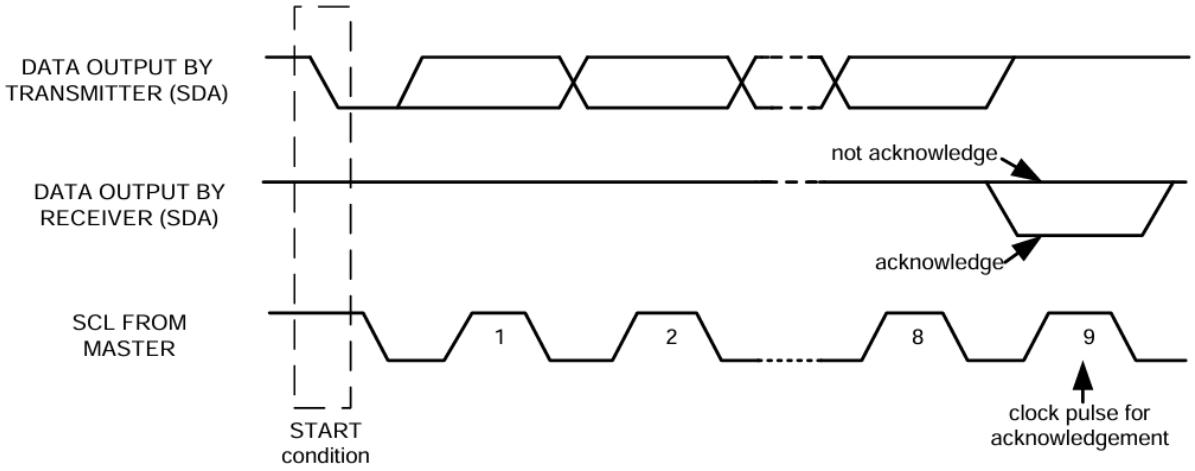
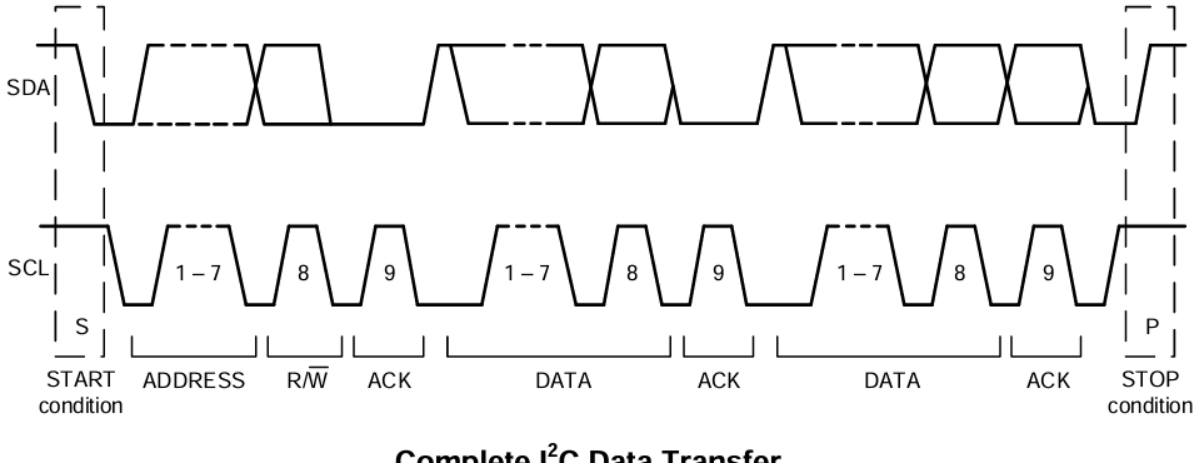


Figura 4: Funcionamiento del Acknowledge en el Bus I2C.

Fuente: Datasheet MPU6050



Complete I²C Data Transfer

Figura 5: Transferencia de datos I2C.

Fuente: Datasheet MPU6050

3.2. Fase 2 — Implementación de la comunicación

En esta fase se inicializa el bus I²C, se despierta el sensor MPU6050 y se implementa la lectura y el procesamiento de los datos. Para mantener el código organizado y fácil de entender, se utiliza una estructura modular basada en tres archivos: el programa principal `.ino`, el archivo de cabecera `MPU6050.h` y el archivo de implementación `MPU6050.cpp`, donde se desarrolla la lógica de cada función.

La comunicación inicia con `Wire.begin()`, que habilita el bus I²C y establece al Arduino UNO como dispositivo maestro. Luego, se despierta el sensor escribiendo el valor `0x00` en el registro PWR_MGMT_1 (0x6B), ya que el MPU6050 se encuentra en modo de reposo por defecto. Una vez activo, el microcontrolador selecciona el registro inicial ACCEL_XOUT_H (0x3B) y realiza una lectura secuencial de 14 bytes, correspondientes a acelerómetro, temperatura y giroscopio.

Los valores obtenidos se almacenan como datos crudos (`int16_t`) y posteriormente se convierten a unidades físicas utilizando los factores de sensibilidad del sensor (g, °/s y °C). Con estos datos ya convertidos, el sistema calcula los ángulos Roll, Pitch y Yaw combinando la integración del giroscopio con las estimaciones del acelerómetro. Para mejorar la estabilidad de la medición se aplica un filtro complementario, que mezcla el comportamiento rápido del giroscopio con la estabilidad del acelerómetro.

Finalmente, el archivo principal ejecuta de manera continua las funciones de lectura, procesamiento y visualización, incluyendo un retardo controlado que fija la frecuencia de muestreo alrededor de 10 Hz. Esta estrategia modular facilita la comprensión del código, permite depurar cada parte del sistema de forma independiente y asegura que el flujo de comunicación entre el Arduino y el sensor sea claro, preciso y estable.

3.3. Fase 3 — Presentación y análisis

En esta fase se realiza la exposición y evaluación del funcionamiento del sistema implementado. Primero, se explica la estructura modular del código, destacando el rol del archivo principal y la librería personalizada del MPU6050. Se detalla también el flujo de comunicación I²C entre el Arduino UNO y el sensor, incluyendo la importancia de las líneas SDA y SCL, las condiciones START y STOP, la dirección del esclavo y la función de los bits ACK y NACK durante la transmisión.

Posteriormente, se analiza el procesamiento de los datos obtenidos. A partir de los valores crudos de acelerómetro y giroscopio, se calcula la orientación del sensor en términos de Roll, Pitch y Yaw, utilizando integración temporal y el filtro complementario para reducir ruido y deriva. Esta explicación es acompañada de una demostración práctica, donde se visualiza en el Monitor Serial la lectura en tiempo real, permitiendo observar cómo varían los ángulos al mover el módulo MPU6050.

Finalmente, se discuten los resultados obtenidos y se interpretan los cambios en la orientación del sensor, relacionándolos con su comportamiento físico. La presentación concluye mostrando que el sistema implementado es capaz de medir dinámicamente el movimiento, procesarlo correctamente y presentarlo de forma estable y comprensible, cumpliendo así los objetivos de la práctica.

4. Explicación del código

4.1 Estructura general del proyecto Estructura general del proyecto

El proyecto está organizado en tres archivos que trabajan juntos:

- practica04_MPU6050.ino → Programa principal (sketch de Arduino)

- MPU6050.h → Archivo de cabecera: definición de la clase y sus miembros

- MPU6050.cpp → Archivo de implementación: código de los métodos de la clase

Que el proyecto esté separado en estos tres archivos no es por capricho, sino por orden y reutilización:

- El archivo .h define la "forma" de la clase, es decir, qué variables y métodos tiene.

- El archivo .cpp implementa el comportamiento real de esos métodos.

- El archivo .ino es el programa principal que crea un objeto de esa clase y lo usa

.

4.2. Archivo MPU6050.h (cabecera)

4.2.1 Comentario Inicial

-Aquí se indica el propósito del archivo: es una librería para manejar el sensor MPU6050 usando comunicación I2C con un Arduino UNO.

```
/*
  MPU6050.h - Libreria para manejo del sensor MPU6050
  Comunicacion I2C con Arduino UNO
*/
```

4.2.2 Guardias de inclusión

```
#ifndef MPU6050_H
#define MPU6050_H
```

#endif

-Estas líneas son las llamadas “guardias de inclusión”. Sirven para evitar que el archivo se incluya más de una vez durante la compilación.

- #ifndef significa “if not defined”.

- Si MPU6050_H aún no está definido, se define y se incluye todo el contenido del archivo.

- Si ya estaba definido, el compilador se salta el contenido, evitando redefiniciones de la clase que causarían errores.

4.2.3 Inclusión de librerías necesarias

```
 8
 9    #include <Arduino.h>
10    #include <Wire.h>
11
```

- Arduino.h proporciona funciones básicas como pinMode, digitalWrite, Serial, millis, etc.

- Wire.h es la librería que permite la comunicación I2C, con funciones como Wire.begin, Wire.write, Wire.read, Wire.requestFrom, etc.

4.2.4 Definición de la clase MPU6050

```
class MPU6050 {
  private:
    const int direccion = 0x68;

    // Variables crudas del sensor
    int16_t acelerometroX, acelerometroY, acelerometroZ;
    int16_t giroscopioX, giroscopioY, giroscopioZ;
    int16_t temperatura;
```

```
// Variables de angulos
float anguloAcX, anguloAcY;
float anguloGyX, anguloGyY, anguloGyZ;

// Variables de tiempo
float tiempo, tiempoActual, tiempoAnterior;

public:
  // Variables publicas de salida
  float roll, pitch, yaw;

  // Metodos publicos
  MPU6050();
  void inicializar();
  void leerDatos();
  void calcularAngulos();
  void mostrarDatos();
};
```

-La palabra class indica que se va a definir una clase, que es una plantilla para crear objetos.

- La sección private contiene variables internas que no se pueden modificar directamente desde fuera de la clase.

- La sección public contiene las variables y métodos accesibles desde el código principal.

4.2.5 Variables privadas: dirección I2C

```
const int direccion = 0x68;
```

- const indica que es una constante: no va a cambiar en tiempo de ejecución.

- 0x68 es la dirección I2C del sensor MPU6050 en hexadecimal.

-Esta dirección identifica al sensor en el bus I2C. El uso de const evita cambios accidentales y deja explícito que es un valor fijo.

4.2.6 Variables privadas: datos crudos del sensor

```
int16_t acelerometroX, acelerometroY, acelerometroZ;
int16_t giroscopioX, giroscopioY, giroscopioZ;
int16_t temperatura;
```

int16_t es un entero con signo de 16 bits. El rango típico va de -32768 a 32767.

El MPU6050 entrega sus valores en este formato de 16 bits, por eso se usa este tipo:

- acelerometroX/Y/Z: aceleraciones medidas en cada eje.

- giroscopioX/Y/Z: velocidades angulares crudas de cada eje.

- temperatura: temperatura interna del sensor (aunque en el código base no se use, se lee igualmente).

4.2.7 Variables privadas: ángulos y tiempo

```
// Variables de angulos
float anguloAcX, anguloAcY;
float anguloGyX, anguloGyY, anguloGyZ;

// Variables de tiempo
float tiempo, tiempoActual, tiempoAnterior;
```

- float permite almacenar números con decimales.

- anguloAcX e anguloAcY almacenan los ángulos calculados a partir del acelerómetro.

- anguloGyX, anguloGyY, anguloGyZ almacenan los ángulos integrados a partir del giroscopio.

- tiempoActual y tiempoAnterior guardan las marcas de tiempo en milisegundos mediante millis().

- tiempo es la diferencia en segundos entre dos lecturas consecutivas, necesaria para integrar la velocidad angular del giroscopio.

4.2.8 Variables y Metodos Publicos

En la sección public se declaran:

float roll, pitch, yaw;

Estas son las salidas finales de la clase:

- roll: inclinación lateral (rotación alrededor del eje X).

- pitch: inclinación hacia adelante/atrás (rotación alrededor del eje Y).

- yaw: giro horizontal (rotación alrededor del eje Z).

También se declaran los métodos públicos:

```
// Métodos públicos
MPU6050();
void inicializar();
void leerDatos();
void calcularAngulos();
void mostrarDatos();
```

- MPU6050() es el constructor: se ejecuta cuando se crea un objeto de la clase.

- inicializar() configura la comunicación I2C, "despierta" el sensor y configura el Serial.

- leerDatos() lee los valores crudos del acelerómetro, temperatura y giroscopio.

- calcularAngulos() procesa esos valores para obtener roll, pitch y yaw.

- mostrarDatos() imprime esos resultados por el puerto serial.

4.3 Archivo MPU6050.cpp (implementación)

4.3.1 Inclusión de la cabecera

```
#include "MPU6050.h"
```

Se incluye el archivo de cabecera para que el compilador conozca la definición de la clase y sus miembros.

4.3.2 Constructor de la clase

Este código asegura que las variables que se usan para acumular e inicializar estados empiecen en cero. Si no se inicializan, podrían contener cualquier valor residual en memoria.

```
// Constructor
MPU6050::MPU6050() {
  anguloGyX = 0;
  anguloGyY = 0;
  anguloGyZ = 0;
  roll = 0;
  pitch = 0;
  yaw = 0;
  tiempoAnterior = 0;
}
```

4.3.3 Metodo Inicializador

```
// Inicializar sensor MPU6050
void MPU6050::inicializar() {
  Wire.begin();
  Wire.beginTransmission(direccion);
  Wire.write(0x6B);  // Registro PWR_MGMT_1
  Wire.write(0);     // Despertar el MPU6050
  Wire.endTransmission(true);

  Serial.begin(115200);
  Serial.println("========================================");
  Serial.println("    MPU6050 INICIALIZADO CORRECTAMENTE");
  Serial.println("========================================");
  delay(2000);
}
```

Desglose:

- Wire.begin(): habilita el hardware I2C del Arduino en los pines SDA y SCL.

- Wire.beginTransmission(direccion): inicia una transmisión hacia la dirección 0x68.

- Wire.write(0x6B): selecciona el registro PWR_MGMT_1 del MPU6050 (gestión de energía).

- Wire.write(0): escribe el valor 0 en ese registro, lo que despierta el sensor (bit SLEEP en 0).

- Wire.endTransmission(true): termina la transmisión y envía una condición STOP.

Luego se inicia el puerto serie a 115200 baudios y se imprime un mensaje de confirmación. El delay(2000) da tiempo a que el usuario vea el mensaje y a que el sensor se estabilice.

4.3.4 Método leerDatos()

```
// Leer datos crudos del sensor
void MPU6050::leerDatos() {
  Wire.beginTransmission(direccion);
  Wire.write(REG_ACCEL_XOUT_H);  // Registro inicial de datos
  Wire.endTransmission(false);
  Wire.requestFrom(direccion, NUM_BYTES_LECTURA, true);

  // Leer acelerometro
  acelerometroX = Wire.read() << 8 | Wire.read();
  acelerometroY = Wire.read() << 8 | Wire.read();
  acelerometroZ = Wire.read() << 8 | Wire.read();

  // Leer temperatura (no se usa)
  temperatura = Wire.read() << 8 | Wire.read();

  // Leer giroscopio
  giroscopioX = Wire.read() << 8 | Wire.read();
  giroscopioY = Wire.read() << 8 | Wire.read();
  giroscopioZ = Wire.read() << 8 | Wire.read();
}
```

```
// Inicializar sensor MPU6050
void MPU6050::inicializar() {
  Wire.begin();
  Wire.beginTransmission(direccion);
  Wire.write(REG_PWR_MGMT_1);  // Registro PWR_MGMT_1
  Wire.write(VALOR_WAKE_UP);   // Despertar el MPU6050
  Wire.endTransmission(true);

  Serial.begin(BAUDRATE_SERIAL);
  Serial.println("    MPU6050 INICIALIZADO CORRECTAMENTE");
  delay(2000);
}
```

Explicación:

- Se indica al sensor que queremos empezar a leer desde el registro 0x3B (ACCEL_XOUT_H).

- Se llama a Wire.endTransmission(false) para NO enviar STOP y permitir un "reinicio" de la comunicación (RESTART) para lectura inmediata.

- Wire.requestFrom(direccion, 14, true) solicita 14 bytes seguidos al sensor.

Los 14 bytes corresponden a:

- 2 bytes para acelerómetro X

- 2 bytes para acelerómetro Y

- 2 bytes para acelerómetro Z

- 2 bytes para temperatura

- 2 bytes para giroscopio X

- 2 bytes para giroscopio Y

- 2 bytes para giroscopio Z

Cada par de bytes se combina con la operación:

valor = (byteAlto << 8) | byteBajo;

El operador << desplaza los bits 8 posiciones a la izquierda y el OR bit a bit (|) completa los 16 bits del entero.

4.3.5 Método calcularAngulos()

```
-

// Calcular angulos Roll, Pitch, Yaw
void MPU6050::calcularAngulos() {
  // Calcular tiempo transcurrido
  tiempoAnterior = tiempoActual;
  tiempoActual = millis();
  tiempo = (tiempoActual - tiempoAnterior) / 1000.0;
```

```
// Calcular angulos desde acelerometro
anguloAcX = (atan(acelerometroY / sqrt(pow(acelerometroX, 2) + pow(acelerometroZ, 2))) * 180 / PI);
anguloAcY = (atan(-1 * acelerometroX / sqrt(pow(acelerometroY, 2) + pow(acelerometroZ, 2))) * 180 / PI);

// Convertir giroscopio a grados/segundo
float gyroX = giroscopioX / SENS_GYRO_250DPS;
float gyroY = giroscopioY / SENS_GYRO_250DPS;
float gyroZ = giroscopioZ / SENS_GYRO_250DPS;
```

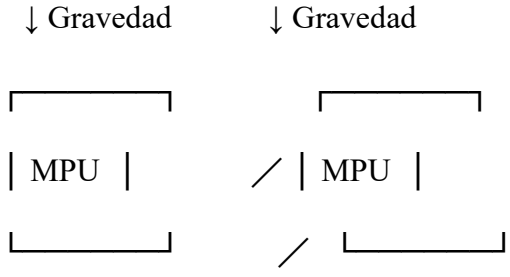
```
    // Calcular angulos desde giroscopio
    anguloGyX = anguloGyX + gyroX * tiempo;
    anguloGyY = anguloGyY + gyroY * tiempo;
    yaw = yaw + gyroZ * tiempo;

    // Filtro complementario (96% giroscopio + 4% acelerometro)
    roll = 0.96 * anguloGyX + 0.04 * anguloAcX;
    pitch = 0.96 * anguloGyY + 0.04 * anguloAcY;

    // Actualizar valores del giroscopio
    anguloGyX = roll;
    anguloGyY = pitch;
}
```

¿Cómo el acelerómetro mide ángulos?

Cuando inclinas el sensor, la gravedad (9.8 m/s²)

se distribuye entre los ejes:

Sensor horizontal: Sensor inclinado 45°:



Z = 9.8 X = 6.9

X = 0 Z = 6.9

Y = 0 Y = 0

1. Calcula el tiempo entre lecturas con millis().

2. Usa los valores del acelerómetro para obtener ángulos de inclinación (ángulo desde la gravedad).

3. Convierte los valores crudos del giroscopio a grados/segundo dividiendo por 131 (sensibilidad típica para ±250 °/s).

4. Integra la velocidad angular para obtener ángulos: ángulo nuevo = ángulo anterior + velocidad × tiempo.

5. Aplica un filtro complementario que combina el ángulo del giroscopio y el del acelerómetro.

6. Actualiza los ángulos del giroscopio con los valores filtrados para reducir el "drift".

4.3.6 Método mostrar Datos()

```
void MPU6050::mostrarDatos() {
  Serial.print("Roll: ");
  Serial.print(roll, 2);
  Serial.print("°   Pitch: ");
  Serial.print(pitch, 2);
  Serial.print("°   Yaw: ");
  Serial.print(yaw, 2);
  Serial.println("°");
}
```

Muestra los valores finales de roll, pitch y yaw con dos decimales, en un formato fácil de leer en el Monitor Serial.

4.4 Comentario inicial y conexiones

```
/*
  Practica 04 - Comunicacion I2C con MPU6050
  Universidad Catolica Boliviana "San Pablo"
  IMT-222 Sistemas Embebidos I

  Hardware:
  - Arduino UNO
  - Sensor MPU6050 (I2C)

  Conexiones:
  MPU6050 VCC -> Arduino 5V
  MPU6050 GND -> Arduino GND
  MPU6050 SCL -> Arduino A5
  MPU6050 SDA -> Arduino A4
*/
```

Se documenta el objetivo de la práctica, el curso y las conexiones físicas entre Arduino y el sensor.

4.3.1 Inclusion de a librería y creación del objeto

```
#include "MPU6050.h"

// Crear objeto del sensor
MPU6050 sensor;
```

- Se incluye la cabecera de la librería para poder usar la clase.

- Se crea un objeto global sensor de tipo MPU6050, accesible desde setup() y loop().

4.4.2 Funcion Setup()

```
void setup() {
  // Inicializar sensor MPU6050
  sensor.inicializar();
}
```

setup() se ejecuta una sola vez al inicio. En este caso, solo llama a sensor.inicializar(), que ya se encarga de configurar I2C, despertar el sensor y preparar el Serial.

4.4 Función loop()

```
void loop() {
  // Leer datos del sensor
  sensor.leerDatos();

  // Calcular angulos Roll, Pitch, Yaw
  sensor.calcularAngulos();

  // Mostrar resultados
  sensor.mostrarDatos();

  // Esperar 100ms antes de siguiente lectura
  delay(100);
}
```

En cada iteración del loop se realiza el ciclo completo:

1. leerDatos(): se obtienen los datos crudos del sensor.
2. calcularAngulos(): se procesan y se actualizan roll, pitch y yaw.
3. mostrarDatos(): se imprimen los resultados.
4. delay(100): se espera 100 ms antes de la siguiente lectura (10 lecturas por segundo).

4.5 Interacción entre los tres archivos

- El .h define la estructura de la clase (qué variables y métodos existen).

- El .cpp implementa el comportamiento de esos métodos y maneja toda la lógica del sensor.

- El .ino crea el objeto de la clase y lo usa dentro de setup() y loop(), sin preocuparse por los detalles

internos.

4.6 Errores frecuentes y cómo abordarlos

- Si el sensor no responde, puede ser por cables mal conectados, falta de alimentación o dirección I2C incorrecta.

- Si los valores saltan bruscamente, puede deberse a cables muy largos, ruidos en la alimentación o falta de filtrado.

- Si el Monitor Serial no muestra nada, suele ser un problema con la velocidad configurada (115200) o con el puerto seleccionado.

- Si la compilación falla por “Wire no declarado” o “MPU6050 no es un tipo”, hay que revisar los includes y la organización de archivos.

5. Máquina de Estados (FSM) del Sistema

La siguiente máquina de estados representa el flujo general del sistema implementado para la lectura del sensor MPU6050 mediante el protocolo I²C. Su propósito es organizar el funcionamiento del programa en etapas bien definidas, facilitando la comprensión del ciclo de muestreo, la adquisición de datos y el procesamiento de los mismos.

El sistema avanza de manera secuencial por cada estado, asegurando que la configuración, lectura y visualización de los valores del acelerómetro y giroscopio se realicen de manera ordenada y repetitiva.

La máquina de estados propuesta organiza el funcionamiento del sistema en cuatro etapas principales: Configuración, Lectura, Visualización y Manejo de Errores. Cada una de ellas cumple un rol específico en el proceso de adquisición y procesamiento de datos del MPU6050 mediante el protocolo I²C.

Estado: Configuración

En este estado inicial se prepara todo el sistema para iniciar la comunicación:

- Se inicializa el bus I²C mediante Wire.begin(), habilitando las líneas SDA y SCL.
- Se despierta el sensor escribiendo el valor **0** en el registro PWR_MGMT_1, ya que el MPU6050 arranca por defecto en modo de suspensión.
- Se verifica que el sensor responda correctamente enviando un ACK desde la dirección **0x68**.

Transiciones posibles:

- Si el sensor responde: pasa a **Lectura** (sensorOk).

- Si no responde: pasa al estado **Error** (sensorFail).
- También puede detenerse el sistema antes de continuar (stop).

Estado: Lectura

En esta etapa el sistema realiza la adquisición directa de los datos crudos del sensor:

- Envía una condición START por I²C.
- Selecciona el registro **0x3B**, correspondiente a ACCEL_XOUT_H.
- Solicita la lectura secuencial de **14 bytes**:
    - Acelerómetro: AccX, AccY, AccZ
    - Temperatura interna
    - Giroscopio: GyroX, GyroY, GyroZ
- Verifica el bit interno que indica que los datos están listos (dataReady).

Transiciones posibles:

- Si los datos están listos: pasa a **Visualización** o al siguiente ciclo (ciclo de actualización).
- Si ocurre un fallo en el sensor: pasa a **Error** (sensorFail).
- Puede detenerse el sistema (stop).

Estado: Visualización

Este estado se encarga del procesamiento y presentación final de los datos:

- Convierte los datos crudos del acelerómetro y giroscopio a ángulos de Roll, Pitch y Yaw.
- Aplica un **filtro complementario**, combinando acelerómetro (estabilidad) y giroscopio (respuesta rápida).
- Muestra los valores finales en el Monitor Serial en tiempo real.

Transiciones posibles:

- Retorna al estado de **Lectura** para continuar el ciclo de muestreo.

Estado: Error

Este estado se activa únicamente cuando ocurre un problema en la comunicación o lectura:

Posibles causas:

- El sensor no envía ACK.

- La dirección I²C utilizada es incorrecta.
- Problemas en el cableado físico (SDA/SCL).
- Fallo inesperado durante la lectura o procesamiento.

En este estado se muestra un mensaje de error al usuario para facilitar la depuración.

Transiciones posibles:

- Puede reiniciarse el sistema y volver a **Configuración** (reset).

Estado Inicial (Nodo negro superior)

Representa el punto de entrada del sistema.
Desde aquí el flujo avanza automáticamente hacia **Configuración**, donde comienza todo el proceso.

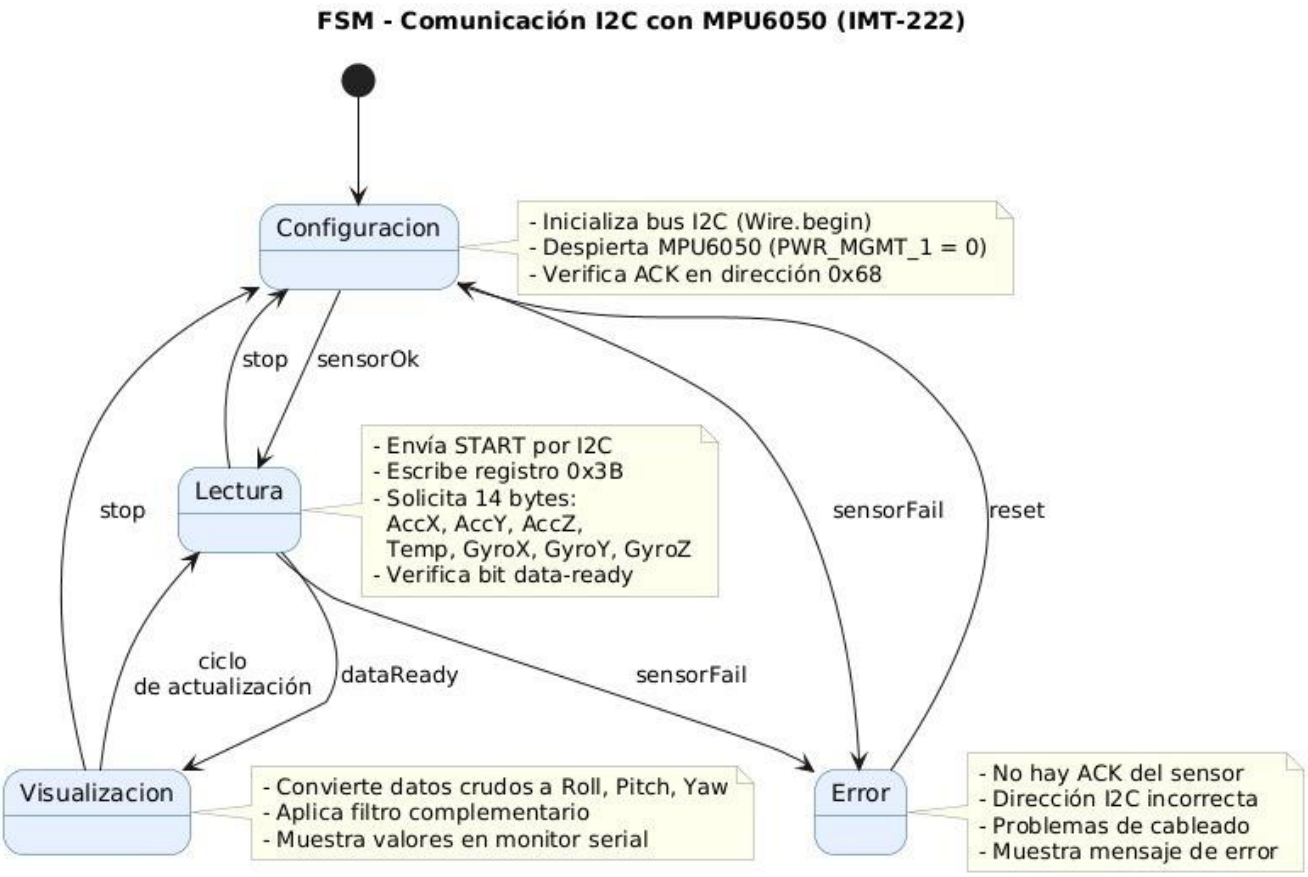


Figura 6: FSM general del proceso de lectura y comunicación entre Arduino UNO y el sensor MPU6050 mediante I²C.
Fuente: Elaboración Propia

Máquina de Estados (FSM) representada con notación inspirada en un Activity Diagram

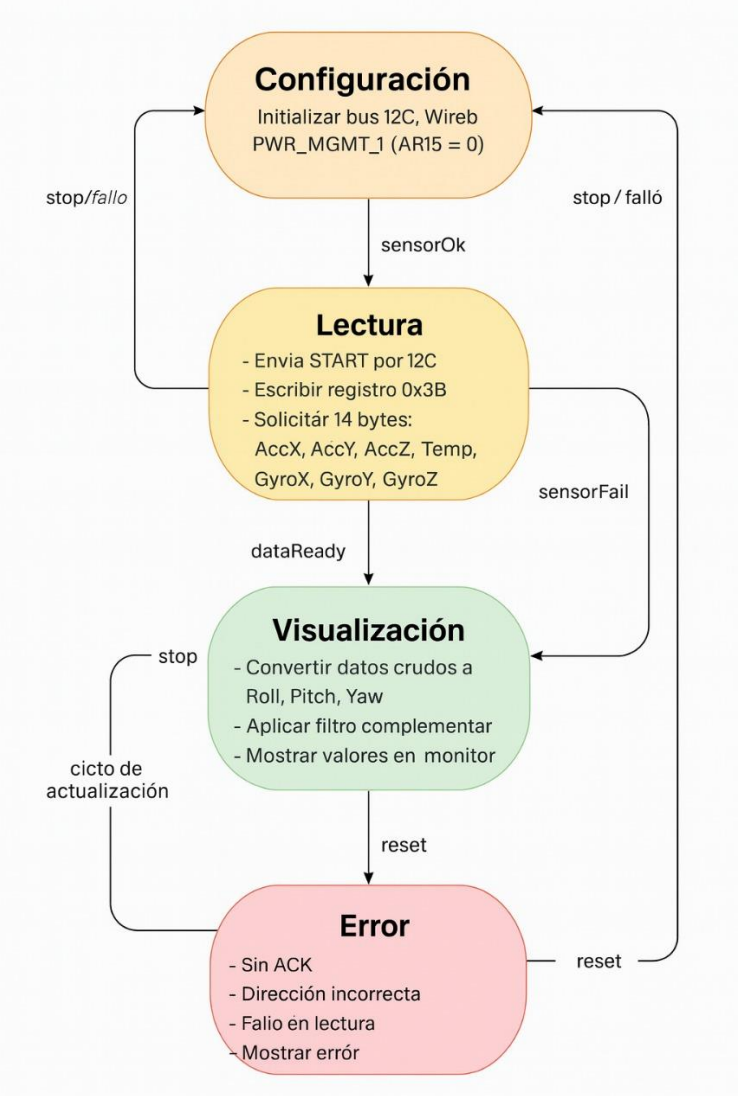


Figura 7: Máquina de Estados (FSM) representada con notación inspirada en un Activity Diagram. Fuente: elaboración propia

La FSM presentada muestra el flujo completo del sistema que interactúa con el sensor MPU6050 mediante I²C, dividido en cuatro estados principales: Configuración, Lectura, Visualización y Error. Cada estado se detalla a continuación.

Estado: Configuración
Este es el primer estado del sistema. Aquí se realizan las tareas necesarias para iniciar la comunicación con el sensor:

- Se inicializa el bus I²C con Wire.begin().
- Se escribe el registro **PWR_MGMT_1 = 0** para despertar el MPU6050.
- Se verifica que el sensor responda enviando un ACK.

Transiciones desde Configuración:

- Si el sensor responde correctamente → pasa a **Lectura** (sensorOk).
- Si hay un error o el usuario detiene el proceso → vuelve a Configuración o va a **Error** (stop/falló).

Estado: Lectura

En esta etapa el sistema obtiene los datos crudos del MPU6050 mediante I²C.
Acciones principales del estado:

- Enviar condición START por el bus I²C.
- Escribir el registro inicial 0x3B (ACCEL_XOUT_H).
- Solicitar **14 bytes** de datos correspondientes a:
  - AccX, AccY, AccZ
  - Temperatura
  - GyroX, GyroY, GyroZ
- Verificar si los datos están listos (dataReady).

Transiciones desde Lectura:

- Si los datos están listos → pasa a **Visualización**.
- Si ocurre un fallo → pasa a **Error** (sensorFail).
- Si se detiene el ciclo → regresa a Configuración o Visualización.

Estado: Visualización

Aquí se procesan y muestran los datos obtenidos en el estado anterior.
Tareas realizadas:

- Conversión de los datos crudos a ángulos **Roll, Pitch y Yaw**.
- Aplicación del filtro complementario para mejorar estabilidad.
- Envío de los valores procesados al **Monitor Serial** para su visualización.

Transiciones desde Visualización:

- Regresa a **Lectura** como parte del ciclo continuo de muestreo.
- Puede detenerse el flujo (stop).

Estado: Error

Este estado gestiona cualquier fallo ocurrido durante la comunicación o lectura del sensor.
Causas posibles de ingresar a "Error":

- El sensor no responde con ACK.
- Dirección I²C incorrecta.
- Problemas físicos en el cableado.
- Lectura incompleta o inválida.

Acciones del estado:

- Mostrar un mensaje de error.
- Esperar a que el usuario reinicie el proceso.

Transiciones desde Error:

- Volver a **Configuración** mediante la señal reset.

La imagen representa el funcionamiento completo del sistema que adquiere y procesa datos del MPU6050 usando una estructura de estados clara y fácil de entender. Aunque conceptualmente es una **FSM**, su estilo visual está inspirado en un **Activity Diagram**, lo que mejora la legibilidad del flujo del programa.

6. Conclusiones

La práctica permitió comprender y aplicar de manera integral el proceso de comunicación digital entre un microcontrolador y un sensor IMU mediante el protocolo I²C, reforzando tanto los conceptos teóricos como las habilidades prácticas necesarias en sistemas embebidos. Se logró establecer correctamente la conexión física y lógica entre el Arduino UNO y el sensor MPU6050, verificando su presencia en el bus y asegurando una inicialización adecuada mediante la configuración del registro PWR_MGMT_1.

Asimismo, la implementación modular del código —separada en archivos .ino, .h y .cpp facilitó la comprensión del funcionamiento interno del sistema, permitiendo un manejo más claro de procesos como la lectura de los 14 bytes del sensor, la conversión de datos crudos a unidades físicas y la aplicación del filtro complementario para la obtención de ángulos de orientación más estables.

La máquina de estados (FSM) desarrollada permitió estructurar el flujo del programa de forma ordenada y repetitiva, asegurando un comportamiento predecible y eficiente del ciclo de muestreo. La visualización en tiempo real de Roll, Pitch y Yaw evidenció el correcto funcionamiento del sistema, así como la capacidad del MPU6050 para medir dinámicamente movimientos y variaciones de orientación.

Finalmente, la práctica permitió afianzar conocimientos clave sobre el protocolo I²C, la gestión de registros internos, la importancia de la temporización y la utilidad de técnicas de fusión sensorial, demostrando la relevancia de estos conceptos en aplicaciones reales de robótica, control y sistemas embebidos.

7. Cuestionario

¿Qué diferencias existen entre los protocolos I²C y SPI en cuanto a número de líneas, velocidad y direccionamiento?

I²C utiliza solo 2 líneas (SDA y SCL), velocidades típicas entre 100 kHz y 400 kHz, y el direccionamiento se hace por direcciones de 7 o 10 bits, asignadas a cada dispositivo del bus (por ejemplo, 0x68 para el MPU6050).
SPI utiliza como mínimo 4 líneas (SCK, MOSI, MISO y CS), puede trabajar a velocidades de varios MHz (hasta 10–50 MHz según el hardware), y su direccionamiento es mediante líneas Chip Select, donde cada esclavo requiere una línea CS independiente. (NXP Semiconductors, 2021; Motorola, 1983)

¿Qué función cumple la señal de ACK (acknowledge) en la comunicación I²C?

El ACK confirma que el receptor (maestro o esclavo) ha recibido correctamente un byte de datos y está listo para continuar la transmisión. Se envía tirando la línea SDA a nivel bajo durante el noveno pulso de reloj. (NXP Semiconductors, 2021)

¿Qué sucede si el esclavo no envía un ACK durante una transmisión I²C?

Si el maestro detecta un NACK, significa que:

- No existe un dispositivo en esa dirección,
- El dispositivo está ocupado, o
- Hubo un error de comunicación.

El maestro normalmente finaliza la transmisión con STOP o reintenta el proceso. (NXP Semiconductors, 2021)

¿Cuál es la función de la línea Chip Select (CS) en el protocolo SPI?

La línea CS (o SS) permite seleccionar qué dispositivo esclavo está activo en un momento dado. Solo el esclavo cuyo CS está en nivel bajo responde; el resto permanece en alta impedancia, evitando conflictos en el bus. (Motorola, 1983)

¿Qué ventajas presenta SPI respecto a I^2C en aplicaciones de alta velocidad?

SPI ofrece velocidades de reloj mucho mayores (en el rango de MHz), comunicación full-duplex y menor overhead de protocolo (sin ACK por byte ni direccionamiento en el propio protocolo), lo que lo hace ideal para pantallas, memorias y sensores de alto rendimiento. (Motorola, 1983)

¿Qué limitaciones tiene cada protocolo al conectar múltiples dispositivos?

En I²C, el número de dispositivos se ve limitado por la capacitancia del bus, las resistencias pull-up y las direcciones disponibles. En SPI, el límite principal es el número de pines CS disponibles en el maestro, ya que cada esclavo requiere una línea de selección independiente. (NXP Semiconductors, 2021; Motorola, 1983)

¿Cómo podría verificarse mediante osciloscopio o analizador lógico la comunicación realizada?

Con un osciloscopio se conectan canales a SDA y SCL para observar condiciones START/STOP, flancos y niveles lógicos, así como la frecuencia del reloj. Con un analizador lógico se puede además decodificar el protocolo I²C, viendo direcciones, datos y bits de ACK/NACK de manera simbólica, lo que facilita la depuración.

8. Anexos

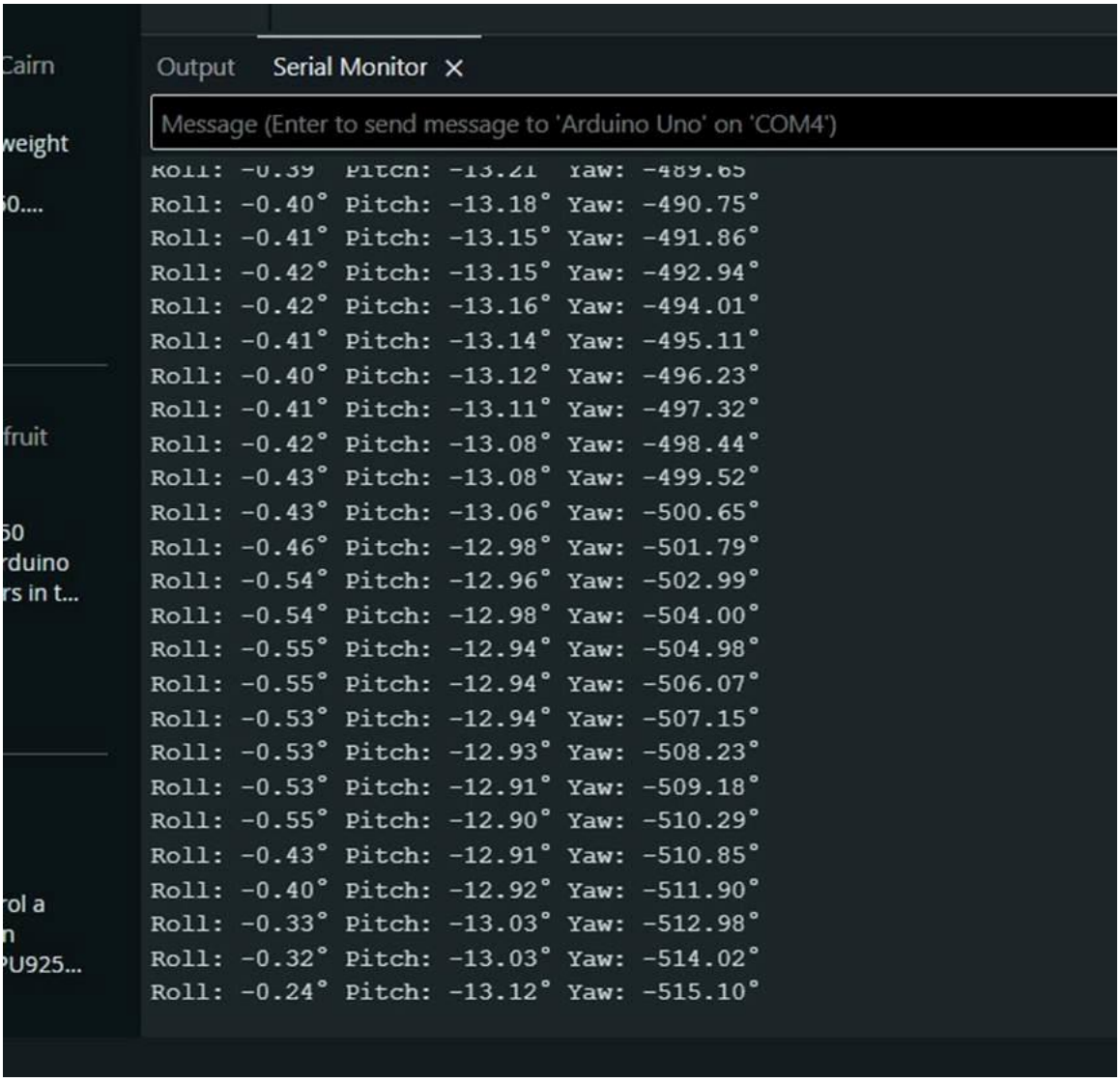


Figura 5. Monitor serial mostrando los ángulos Roll, Pitch y Yaw obtenidos del MPU6050. Elaboración propia.

Registro	Nombre	Función
0x6B	PWR_MGMT_1	Control de energía (despertar/dormir)
0x3B	ACCEL_XOUT_H	Acelerómetro X (byte alto)
0x3C	ACCEL_XOUT_L	Acelerómetro X (byte bajo)
0x3D	ACCEL_YOUT_H	Acelerómetro Y (byte alto)
0x3E	ACCEL_YOUT_L	Acelerómetro Y (byte bajo)
0x3F	ACCEL_ZOUT_H	Acelerómetro Z (byte alto)
0x40	ACCEL_ZOUT_L	Acelerómetro Z (byte bajo)
0x41	TEMP_OUT_H	Temperatura (byte alto)
0x42	TEMP_OUT_L	Temperatura (byte bajo)
0x43	GYRO_XOUT_H	Giroscopio X (byte alto)
0x44	GYRO_XOUT_L	Giroscopio X (byte bajo)
0x45	GYRO_YOUT_H	Giroscopio Y (byte alto)
0x46	GYRO_YOUT_L	Giroscopio Y (byte bajo)
0x47	GYRO_ZOUT_H	Giroscopio Z (byte alto)
0x48	GYRO_ZOUT_L	Giroscopio Z (byte bajo)
0x75	WHO_AM_I	Identificación (siempre devuelve 0x68)

Figura 6. *Principales registros del MPU6050 usados para obtener datos del acelerómetro, giroscopio y temperatura. Elaboración propia.*

```
Bit 7: DEVICE_RESET (1=resetear todo)
Bit 6: SLEEP (1=dormir, 0=despertar)
Bit 5: CYCLE
Bit 4-2: (no importantes)
Bit 1-0: CLKSEL (selección de reloj)
```

Figura 7. *Bits del registro PWR_MGMT_1 del MPU6050 y su función en la gestión de energía. Elaboración propia.*

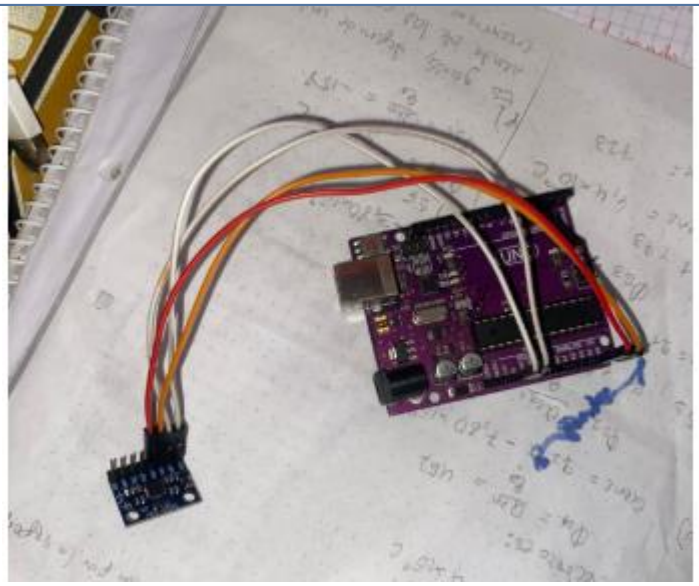


Figura 8. *Conexión física entre el Arduino UNO y el sensor MPU6050 mediante el bus I²C (SDA y SCL). Elaboración propia.*

9. Bibliografia

NXP Semiconductors. (2021). I²C-bus specification and user manual (UM10204).

Motorola. (1983). Serial Peripheral Interface (SPI) Block Guide.

Saleae. (2020). Logic Analyzer Protocol Documentation.

Tektronix. (2019). Fundamentals of the Oscilloscope for Digital Debugging.