Sistemas Embebidos I – IMT-222

COMUNICACIÓN DIGITAL I²C CON SENSOR •MPU6050

UNIVERSIDAD CATÓLICA BOLIVIANA “SAN PABLO” – TARIJA

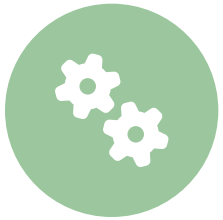
INTEGRANTES

- FLORENCIA FRIGERIO
- BENJAMIN SORUCO
- SUSANN BALDIVIEZO
- ALEJANDRO BEJARANO

01

Arranque del Proyecto

Objetivos del Proyecto



Configurar

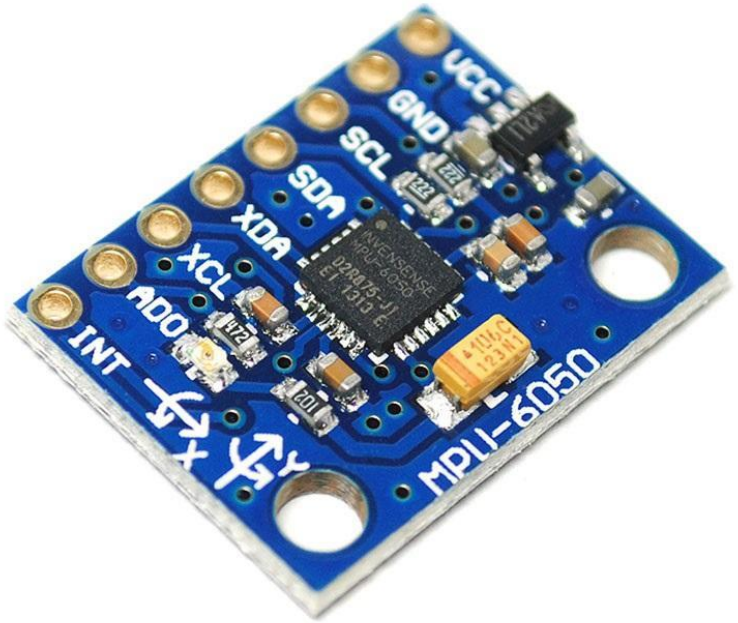
Implementar la interfaz I2C entre Arduino UNO (maestro) y el sensor MPU6050 (esclavo).

Leer y Procesar

Obtener datos del acelerómetro y giroscopio para calcular los ángulos de orientación.

Comprender

Entender el flujo de comunicación a nivel de bit: START, STOP, dirección, ACK, etc.



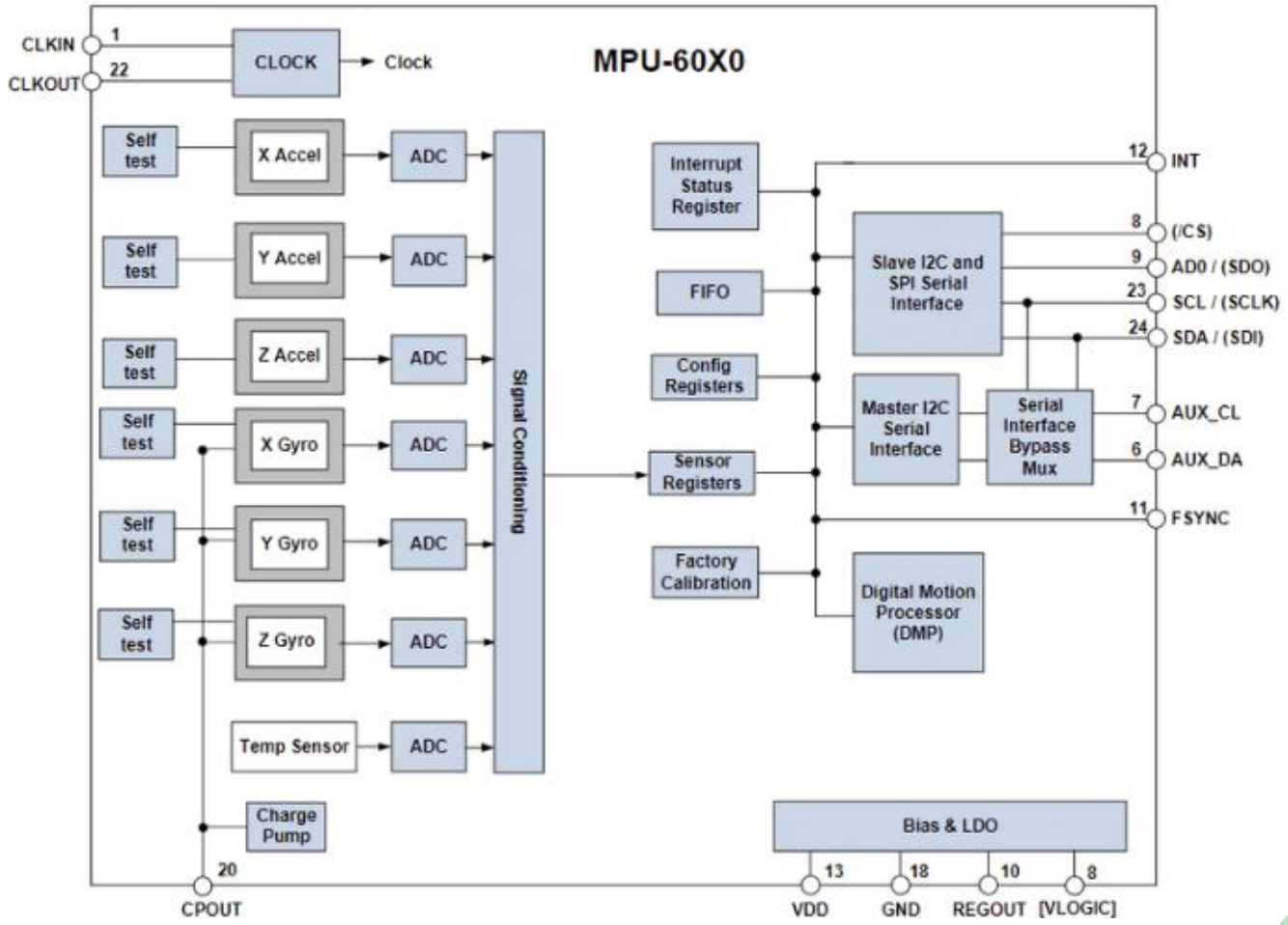
¿Qué es el MPU6050?

Un sensor IMU (Inertial Measurement Unit) de **6 ejes** que integra en un solo chip:

- Un **acelerómetro triaxial** (X, Y, Z) que mide aceleración lineal y gravedad.
- Un **giroscopio triaxial** (X, Y, Z) que mide velocidad angular.
- Un procesador digital de movimiento (DMP) para procesar señales internamente.

Ideal para robótica, drones y sistemas embebidos, comunicándose vía el protocolo I2C .

ARQUITECTURA INTERNA



02

Hardware y Conexión

Materiales y Conexión Física

Hardware

- Arduino UNO (Maestro I2C)
- Sensor MPU6050 (Esclavo)
- Cables de Conexión

Pines de Conexión I2C

SDA (Datos)
Arduino A4 MPU6050 SDA

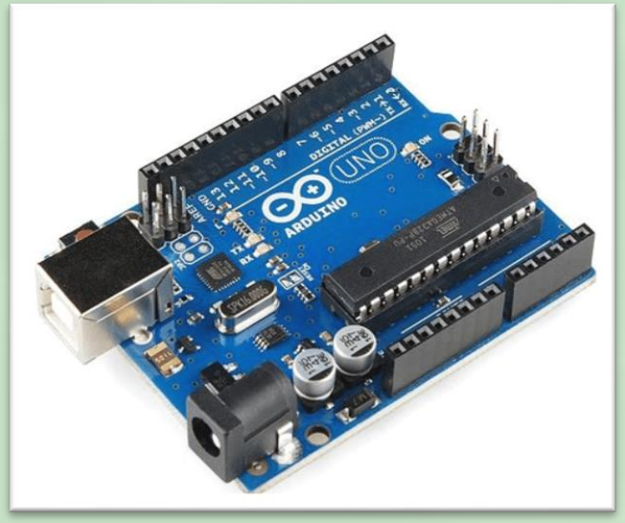
SCL (Reloj)
Arduino A5 MPU6050 SCL

VCC
Arduino 5V MPU6050 VCC

GND
Arduino GND MPU6050 GND

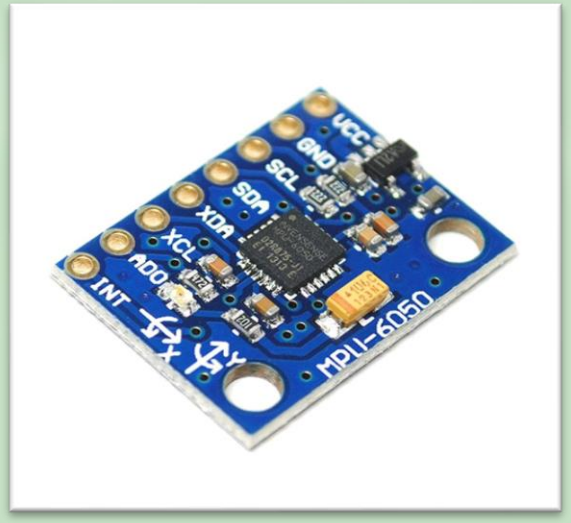
*Las resistencias de pull-up suelen estar integradas en el módulo del sensor.

Diagrama de Conexión Simplificado



Arduino UNO

A4 (SDA)	SDA
A5 (SCL)	SCL
5V	VCC
GND	GND



MPU6050

Una conexión directa y minimalista: solo 4 líneas para establecer el bus de comunicación I2C.

03

PRINCIPIO DE FUNCIONAMIENTO

COMO FUNCIONA?

- Arduino se comunica con el MPU6050 por I^2C.
- Despierta el sensor y lee 14 bytes (acelerómetro + giroscopio).
- Convierte los datos y calcula Roll, Pitch y Yaw.
- Usa un filtro complementario para estabilizar los ángulos.
- Muestra la orientación en tiempo real en el Monitor Serial.

Señales Fundamentales del Protocolo I2C

START / STOP

Indican el inicio y fin de la transmisión. SDA cambia de nivel mientras SCL está en ALTO.

ACK

Confirmación de recepción. El receptor "tira" la línea SDA a nivel BAJO para responder.

NACK

Indica un error o la ausencia del dispositivo, deteniendo la transmisión.

DIRECCIÓN

Identifica al dispositivo esclavo. El MPU6050 responde a la dirección `0x68`.

Secuencia de Lectura de Datos

1. Despertar Sensor: Se escribe `0x00` en el registro `PWR_MGMT_1` (`0x6B`).

2. Seleccionar Registro: Se indica que se comenzará a leer desde el registro `0x3B` (`ACCEL_XOUT_H`).

3. Lectura Secuencial: Se envía un `RESTART` y se leen 14 bytes consecutivos.

4. Datos Obtenidos: Aceleración (X,Y,Z), Temperatura, y Velocidad Angular (X,Y,Z).

04

Procesamiento de Datos

De Datos Crudos a Magnitudes Físicas

Datos Crudos

14 bytes que se combinan en 7 valores de **enteros de 16 bits (int16_t)**.

Rango aproximado: −32,768 a 32,767

Conversión

Magnitudes Físicas

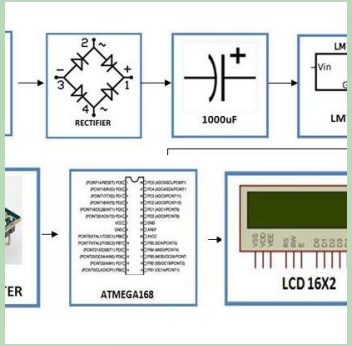
Valores convertidos a unidades utilizables:

Acelerómetro: m/s^2
Giroscopio: °/s (dividiendo por 131)

Estos valores aún no son ángulos, sino aceleración y velocidad angular .

Cálculo de Ángulos: Dos Enfoques

Acelerómetro (Estático)

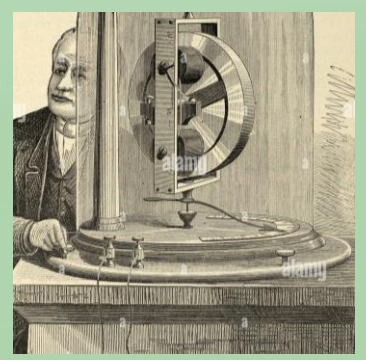


Mide la gravedad para calcular la **inclinación** (Pitch y Roll) usando funciones trigonométricas (arcotangente).

✓ Ventaja: Estable a largo plazo.

✗ Problema: Ruidoso, sensible a vibraciones.

Giroscopio (Dinámico)



Mide la velocidad angular. El **ángulo** se obtiene integrando esta velocidad en el tiempo.

✓ Ventaja: Rápido, preciso en movimiento.

✗ Problema: Deriva (drift) por integración del error.

05

Filtro Complementario

El Problema: Limitaciones de Cada Sensor

Acelerómetro

Es **estable** a largo plazo, pero sus mediciones son **ruidosas** y muy sensibles a vibraciones.

+

Giroscopio

Es **rápido** y preciso, pero la integración del error causa **deriva** (drift) a largo plazo.

Ninguno de los dos, por sí solo, proporciona una medida de orientación perfecta.

La Solución: Filtro Complementario

Combina las fortalezas de ambos sensores para mitigar sus debilidades.

Acelerómetro
(Largo Plazo)

4% 96%

Corrección — Respuesta Rápida

Giroscopio
(Corto Plazo)

Ángulo Filtrado = 0.96 × (Giroscopio) + 0.04 × (Acelerómetro)

06

Software

Arquitectura de Código Modular

`.ino` Principal

Gestiona el ciclo de vida (`setup`, `loop`), inicializa la comunicación y llama a los métodos de la librería.

`.h` Cabecera

Declara la clase `MPU6050`, sus variables privadas (datos crudos) y métodos públicos (inicializar, leer, calcular, mostrar).

`.cpp` Implementación

Contiene la lógica completa: despertar el sensor, leer datos vía I2C, convertir y aplicar el filtro complementario.

Esta estructura mejora la **claridad, reutilización y mantenibilidad** del código.

Código Modular

`.ino` Principal

```
#include "MPU6050.h"

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

Creamos un objeto de la clase MPU6050 que encapsula todo lo relacionado con el sensor: inicialización, lectura y cálculo de ángulos

```
void setup() {
  sensor.inicializar();
}
```

En setup() solo llamamos a sensor.inicializar(). Aquí se configura el bus I²C, se despierta el MPU6050 y se prepara el monitor serial

```
void loop() {
  sensor.leerDatos();
  sensor.calcularAngulos();
  sensor.mostrarDatos();
  delay(100);
}
```

El loop es un ciclo de muestreo

Código Modular

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

La clase tiene fija la dirección I²C del MPU6050 (0x68), que es la que detectamos con el escáner I²C.

```
float roll, pitch, yaw;
```

Exponemos roll, pitch y yaw como variables públicas porque son las que realmente nos interesan para la orientación final.”

`MPU6050.h` Cabecera

```
MPU6050();
void inicializar();
void leerDatos();
void calcularAngulos();
void mostrarDatos();
```

La librería define una interfaz clara: inicializar, leer, calcular y mostrar. Esto hace que el código principal sea legible y modular

`.cpp` IMPLEMENTACIÓN

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

Inicializamos los ángulos y el tiempo en cero. Es el punto de partida para la integración del giroscopio y el filtro complementario

```cpp
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);
  ...
}
```

- Aquí se inicializa el bus I²C con Wire.begin().
- Luego se escribe en el registro PWR_MGMT_1 (0x6B) el valor 0 para despertar el sensor, porque por defecto viene en modo sleep.
- Finalmente se inicia el monitor serial a 115200 baudios para poder ver los datos.

```cpp
void MPU6050::leerDatos() {
  Wire.beginTransmission(direccion);
  Wire.write(0x3B);  // Registro inicial de datos
  Wire.endTransmission(false);
  Wire.requestFrom(direccion, 14, true);

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

  // Leer temperatura
  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();
}
```

- Primero seleccionamos el registro inicial 0x3B (ACCEL_XOUT_H).
- Luego solicitamos 14 bytes seguidos al MPU6050.
- Cada par de bytes (alto y bajo) se combina para formar un entero de 16 bits (int16_t).
- Los primeros 6 bytes son el acelerómetro (X, Y, Z), luego 2 de temperatura y los últimos 6 del giroscopio (X, Y, Z).

```cpp
// Calcular tiempo transcurrido
tiempoAnterior = tiempoActual;
tiempoActual = millis();
tiempo = (tiempoActual - tiempoAnterior) / 1000.0;
```

Calculamos el tiempo entre muestras usando millis(). Esto es importante porque el giroscopio se integra en función del tiempo.

```cpp
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("°");
}
```

Finalmente mostramos los ángulos ya filtrados con dos decimales en el monitor serial. Esto es lo que se puede ver en tiempo real durante la demo

06

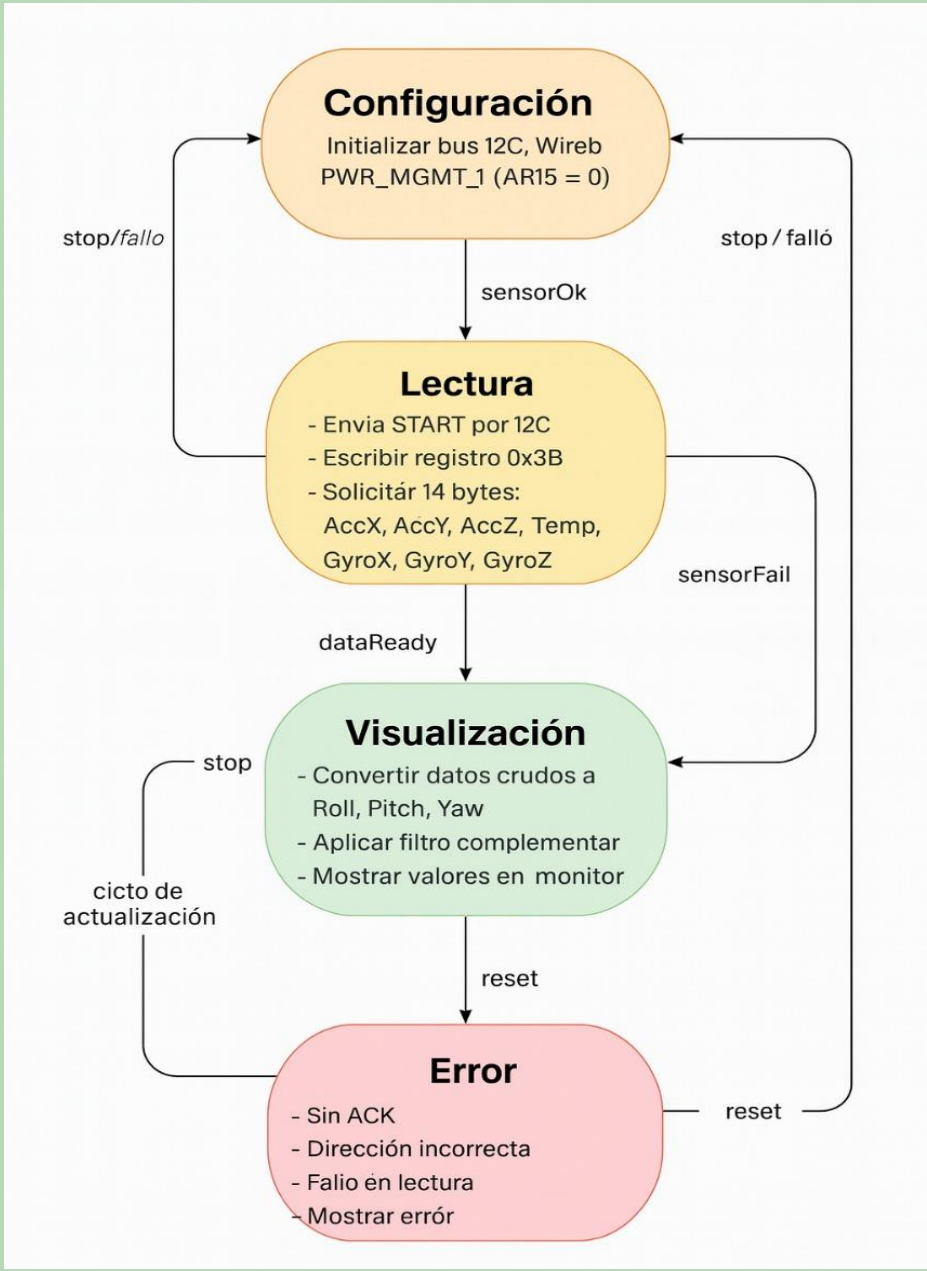
ESTADOS

Máquina de Estados (FSM) del Programa

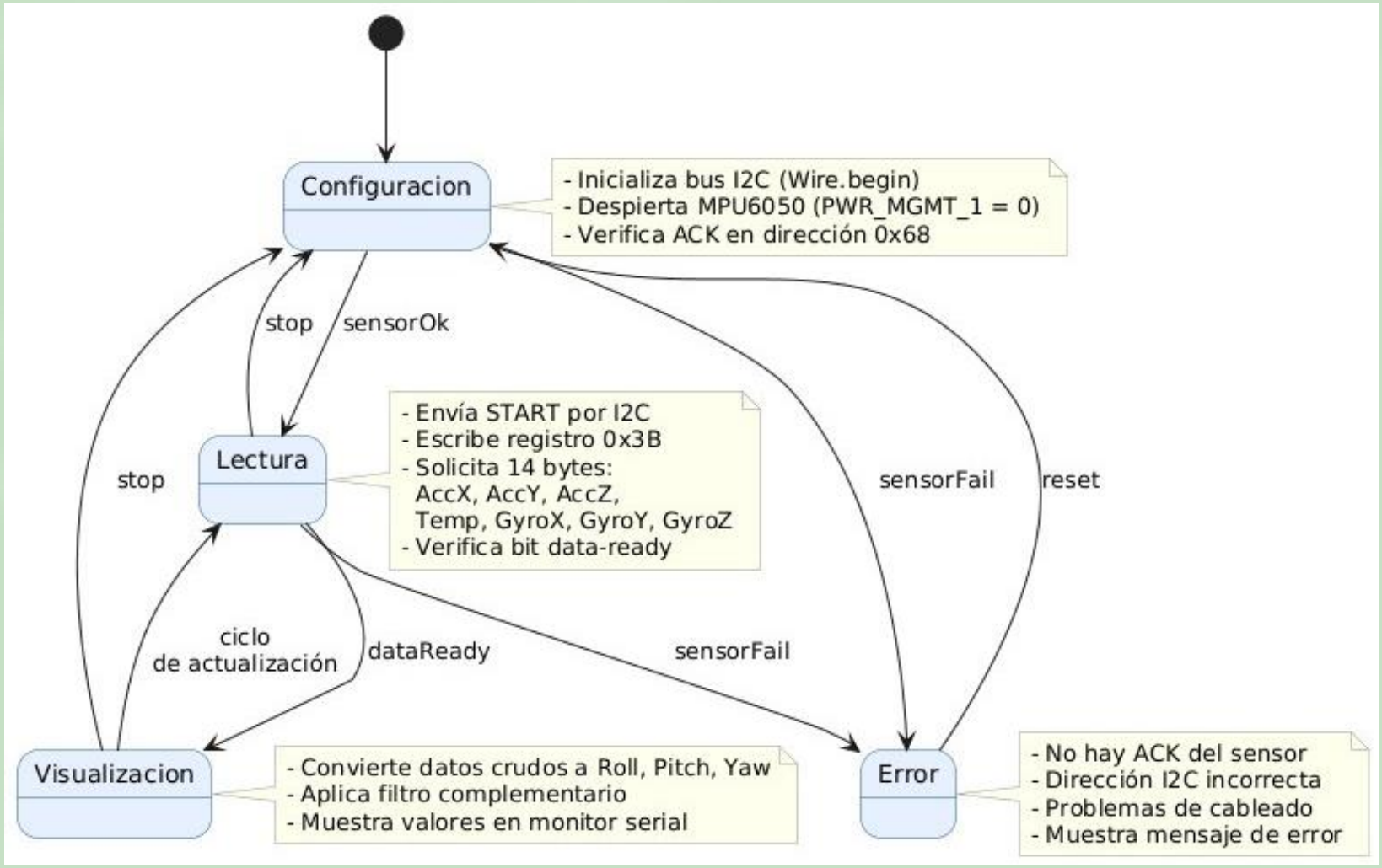
La FSM proporciona una visión clara y estructurada del flujo de ejecución del programa.

Máquina de Estados (FSM)

Este diagrama muestra la lógica de control de nuestro sistema. Diseñamos una Máquina de Estados para garantizar que el sensor se inicialice correctamente antes de intentar leer datos. Además, separamos la lectura física de los datos del procesamiento matemático (fusión de sensores), y añadimos un estado de Error para manejar desconexiones imprevistas, haciendo el sistema más estable



FMS – COMUNICACION I2C CON MPU6050



07

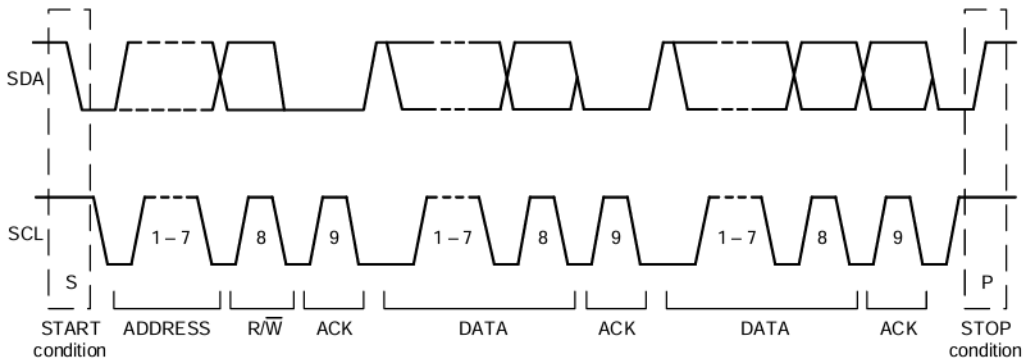
Resultados y Cierre

Demo: Salida en Monitor Serial

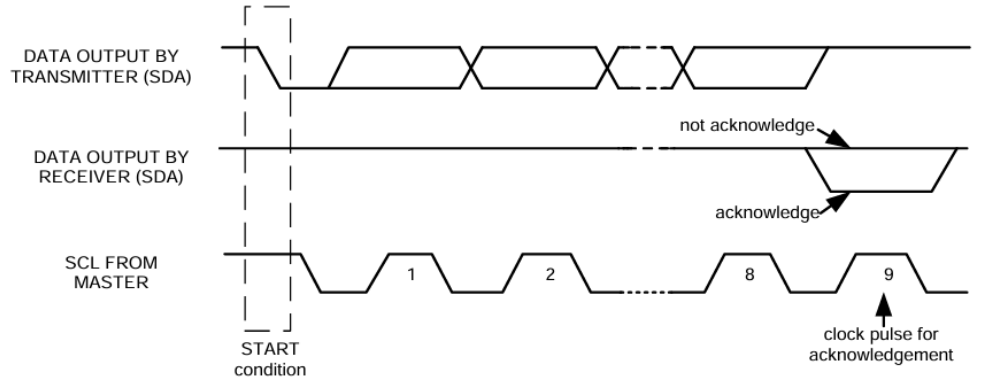
El loop principal ejecuta la FSM, resultando en una salida de datos constante, clara y estable.

- Frecuencia de muestreo: **~10 Hz** (delay de 100ms).
- Velocidad de transmisión: **115,200 baudios**.

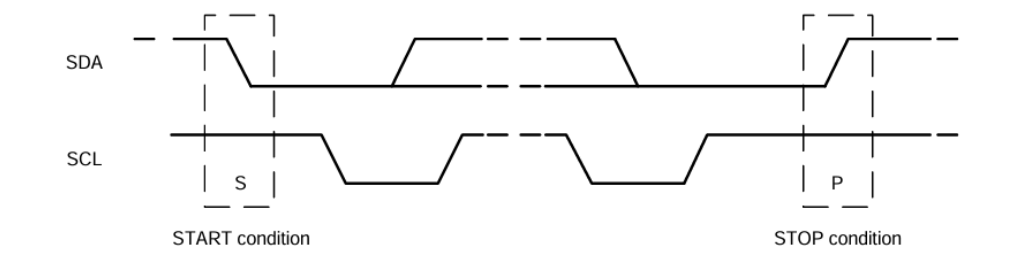
```
> MPU6050 INICIALIZADO CORRECTAMENTE
==========================
> Roll: -2.34° Pitch: 1.56° Yaw: 45.78°
> Roll: -2.31° Pitch: 1.58° Yaw: 45.89°
> Roll: -2.35° Pitch: 1.55° Yaw: 46.01°
> Roll: -2.30° Pitch: 1.57° Yaw: 46.12°
```



Complete I²C Data Transfer



Acknowledge on the I²C Bus



START and STOP Conditions

Conclusiones del Proyecto

- Comunicación I2C implementada con éxito entre Arduino UNO y MPU6050.
- Procesamiento de datos crudos para obtener ángulos de orientación (Roll, Pitch, Yaw).
- Uso del filtro complementario para lograr una medición de orientación estable y precisa.
- Código modular y FSM que mejoran la claridad, reutilización y mantenibilidad del proyecto.

GRACIAS