



Instituto Federal da Paraíba - IFPB  
Engenharia de Computação  
Disciplina: Sistemas Embarcados  
Professores: Alexandre Sales Vasconcelos

**Alunos:**

Álisson Brenner, Caio Livio, Lucas Alves, Vinicius Rodrigues, Vinícius Gonzaga

**RELATÓRIO TÉCNICO: SISTEMA DE CONTROLE DE MESA LABIRINTO COM  
MONITORAMENTO IOT**

Campina Grande  
2025

# **RELATÓRIO TÉCNICO: SISTEMA DE CONTROLE DE MESA LABIRINTO COM MONITORAMENTO IOT**

1. Introdução.....	2
2. Diagrama em Blocos do Sistema.....	2
3. Esquemático e Conexões (Hardware).....	2
4. Descrição das Tarefas e Fluxos FreeRTOS.....	3
4.1. Rotina de Inicialização (Main).....	3
4.2. Tarefas (Tasks).....	4
5. Explicação do Funcionamento do Software.....	4
5.1. Mapeamento Inteligente.....	4
5.2. Ponte de Dados (Python Bridge).....	4
6. Componentes e Bibliotecas Utilizados.....	5
Hardware.....	5
Software & Bibliotecas.....	5
7. Capturas de Tela do Grafana.....	6
8. Fotos da Mesa em Operação.....	6
9. Dificuldades e Soluções Encontradas.....	7

# RELATÓRIO TÉCNICO: SISTEMA DE CONTROLE DE MESA LABIRINTO COM MONITORAMENTO IOT

## 1. Introdução

Este projeto consiste no desenvolvimento de uma Mesa Labirinto (Ball and Plate) controlada eletronicamente. O sistema utiliza um microcontrolador ESP32-S3 para processar entradas de um joystick analógico e controlar dois servomotores que alteram a inclinação da mesa nos eixos X (Pitch) e Y (Roll).

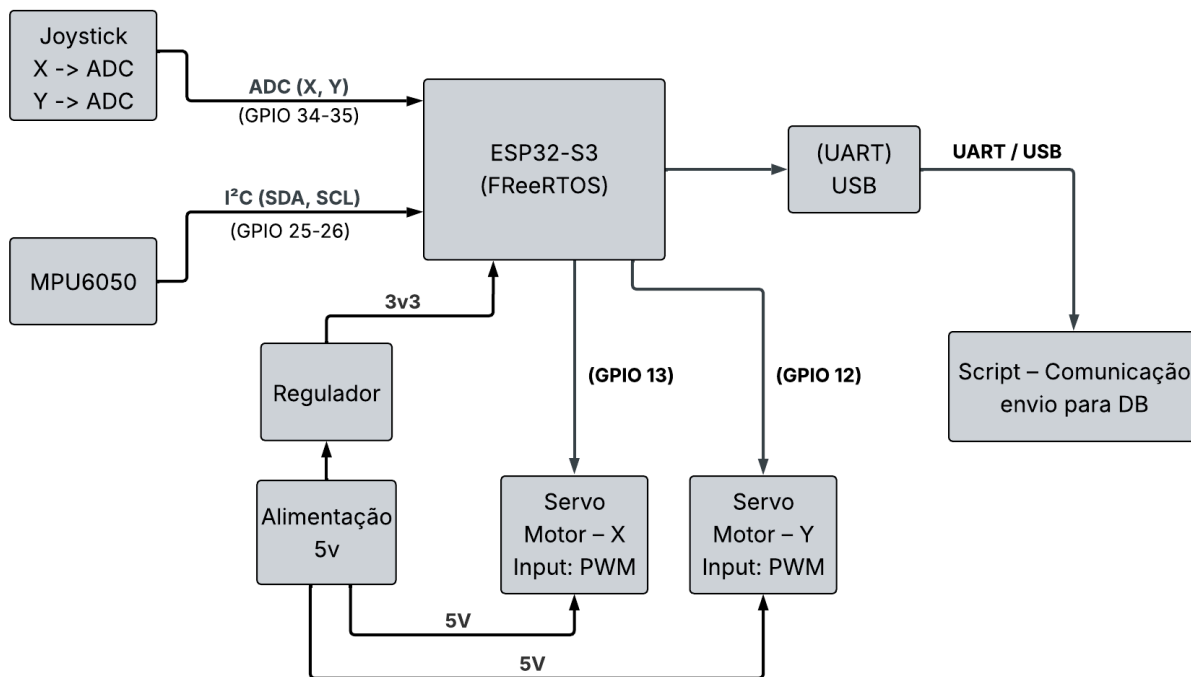
Além do controle mecânico, o sistema integra uma solução de IoT (Internet of Things) local. Um sensor inercial (MPU6050) coleta dados de inclinação em tempo real, que são enviados via Porta Serial para um computador, armazenados em banco de dados temporal (InfluxDB) e visualizados em dashboards interativos (Grafana).

## 2. Diagrama em Blocos do Sistema

O fluxo de informação ocorre da seguinte maneira:

1. **Entrada:** O usuário move o Joystick.
2. **Processamento:** O ESP32 lê o Joystick (ADC) e calcula o ângulo desejado.
3. **Atuação:** O ESP32 envia sinal PWM para os Servos moverem a mesa.
4. **Sensoriamento:** O MPU6050 mede a inclinação real da mesa.
5. **Monitoramento:** O ESP32 envia um pacote JSON via USB -> Python Script -> InfluxDB -> Grafana.

Segue abaixo a imagem do diagrama de blocos:



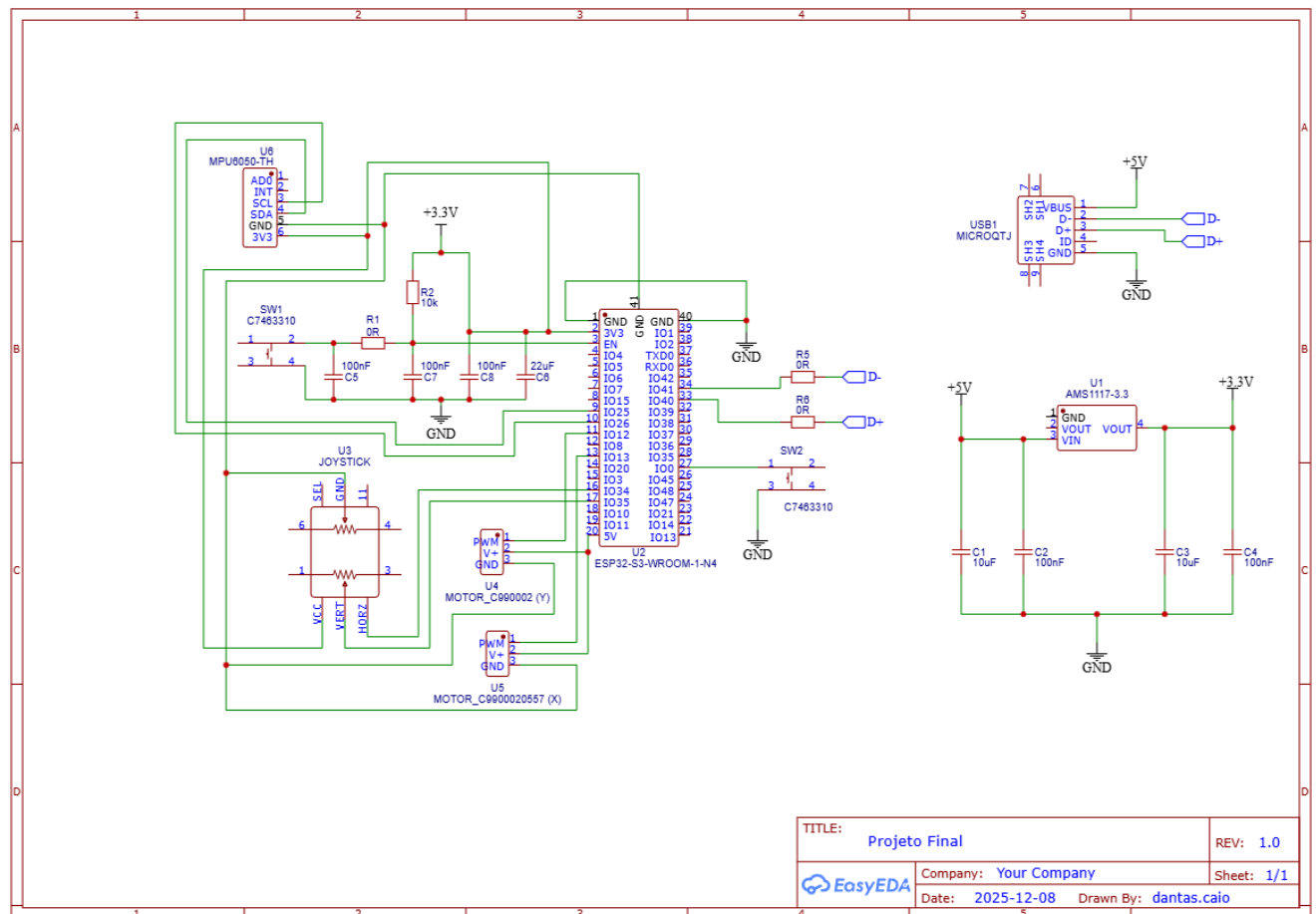
### 3. Esquemático e Conexões (Hardware)

Abaixo estão listadas as conexões físicas realizadas no ESP32-S3.

Componente	Pino Componente	Pino ESP32 (GPIO)	Função
<b>Joystick</b>	VRx (Horiz)	GPIO 4	Leitura Analógica (ADC)
	VRy (Vert)	GPIO 5	Leitura Analógica (ADC)
	SW (Botão)	N/C	Não utilizado
<b>Servos</b>	Sinal X	GPIO 18	PWM (LEDC)
	Sinal Y	GPIO 19	PWM (LEDC)
<b>MPU6050</b>	SDA	GPIO 8	Comunicação I2C
	SCL	GPIO 9	Comunicação I2C
<b>Alimentação</b>	VCC	5V Externo	Alimentação de potência
	GND	GND Comum	Referência

**Nota de Hardware:** Foi utilizada uma fonte de alimentação externa de 5V para os servos, garantindo que a corrente de partida dos motores não reiniciasse o microcontrolador.

Segue abaixo a imagem do Diagrama Esquemático:



## 4. Descrição das Tarefas e Fluxos FreeRTOS

O firmware foi desenvolvido utilizando o **FreeRTOS** (Real-Time Operating System) para garantir determinismo e multitarefa. O sistema foi dividido em 4 tarefas principais e uma rotina de inicialização, protegidas por um Semáforo (Mutex) para acesso seguro às variáveis globais.

## 4.1. Rotina de Inicialização (Main)

Antes de iniciar as tarefas, o sistema executa a função `calibrar_inicial()`.

- **Ação:** Solicita ao usuário que não toque no joystick (mede o centro) e depois que rotacione o joystick (mede os extremos).
- **Resultado:** Gera um mapeamento personalizado para corrigir zonas mortas e desvios mecânicos do joystick.

## 4.2. Tarefas (Tasks)

Tarefa	Prioridade	Frequência	Descrição
<b>task_joystick_read</b>	Alta (2)	50Hz (20ms)	Lê os valores ADC do joystick, aplica filtro exponencial (suavização) e mapeia os valores brutos para ângulos alvo (70° a 110°) usando a calibração.
<b>task_servo_control</b>	Muito Alta (3)	100Hz (10ms)	Responsável por mover os motores. Implementa uma rampa de aceleração suave para evitar movimentos bruscos ("trancos") na mesa.
<b>task_mpu_read</b>	Média (2)	20Hz (50ms)	Lê acelerômetro e giroscópio via I2C. Aplica um <b>Filtro Complementar</b> (98% Giroscópio, 2% Acelerômetro) para obter um ângulo estável sem ruído e sem deriva (drift).
<b>task_monitor</b>	Baixa (1)	5Hz (200ms)	Coleta o "snapshot" atual de todas as variáveis (Joystick, Servos, MPU), formata em uma string <b>JSON</b> e envia pela porta Serial (USB).

## 5. Explicação do Funcionamento do Software

O sistema **Mesa Labirinto** é um projeto embarcado que integra **hardware**, **tempo real**, **processamento de sensores**, **controle de atuadores** e **monitoramento em tempo real**, utilizando três camadas principais:

1. **ESP32 (C / FreeRTOS)** – Aquisição de dados e controle físico.
2. **Python (PC)** – Ponte de comunicação Serial → Banco de Dados.
3. **Grafana + InfluxDB** – Visualização e monitoramento em tempo real.

### 5.1. Código “main.c”

Este arquivo implementa **todo o controle em tempo real da mesa**, rodando sobre o **FreeRTOS** no ESP32-S3.

#### Hardware Controlado

- **Joystick analógico**
  - Entrada do usuário (controle manual)
  - Lido via ADC (eixos X e Y)
- **Dois servomotores**
  - Controlam a inclinação da mesa (eixos X e Y)
  - Controlados via PWM (LEDC)
- **MPU6050**
  - Acelerômetro + Giroscópio
  - Mede inclinação real da mesa

- Comunicação via I2C

## Estrutura do Sistema (FreeRTOS)

O sistema é **multitarefa**, dividido em **quatro tasks principais**, sincronizadas por **mutex**:

TASK	FUNÇÃO
task_joystick_read	Leitura e filtragem do joystick
task_servo_control	Movimento suave dos servos
task_mpu_read	Leitura do MPU6050 + filtro complementar
task_monitor	Envio de dados via Serial em JSON

O **xDataMutex** garante que múltiplas tasks não acessem os mesmos dados ao mesmo tempo, evitando condições de corrida.

## Calibração Inicial do Joystick

Antes de qualquer controle, o sistema executa a função:

**“calibrar\_inicial();”**

Essa calibração é **bloqueante** e ocorre em duas etapas:

### 1. Centro do joystick

- Usuário solta o joystick
- O sistema mede o valor médio (offset real)

### 2. Extremos

- Usuário gira o joystick em círculos



- O sistema detecta mínimo e máximo reais

### **Isso garante:**

- Zona morta precisa
- Movimento simétrico
- Eliminação de erros mecânicos

## **Leitura e Processamento do Joystick**

Na `task_joystick_read`:

1. O ADC lê os **valores brutos**
2. Um **filtro exponencial** remove ruídos elétricos:  
$$\text{“smoothed\_x} = \alpha * \text{raw} + (1-\alpha) * \text{smoothed\_x}”$$
3. O valor passa por um **mapeamento inteligente**, que:
  - Usa os valores calibrados
  - Aplica zona morta
  - Converte para ângulo seguro (70°–110°)
4. O ângulo é **convertido em pulso PWM**.
5. O **duty cycle alvo** é salvo para os servos.

## Controle dos Servos (Movimento Suave)

Na `task_servo_control`:

- O sistema **não move** o servo instantaneamente
- Usa **interpolação gradual**:

**“current += (target - current) \* SERVO\_SMOOTH\_FACTOR”**

**- Resultado:**

- Movimento suave
- Menos vibração
- Maior vida útil dos servos

## Leitura do MPU6050 e Filtro Complementar

Na `task_mpu_read`:

**1. O ESP32 lê:**

- Acelerômetro (ax, ay, az)
- Giroscópio (gx, gy, gz)

**2. Calcula:**

- Pitch e Roll do acelerômetro
- Velocidade angular do giroscópio

**3. Aplica filtro complementar:**

**“ângulo = 0.98 \* (ângulo + gyro \* dt) + 0.02 \* acc”**

**- Combina:**

- Estabilidade do acelerômetro
- Rapidez do giroscópio

## **Envio de Dados via Serial (JSON)**

A `task_monitor` gera mensagens no formato:

```
{  
  "joy_x": 2048,  
  "joy_y": 2100,  
  "servo_x": 95,  
  "servo_y": 88,  
  "pitch": -3.42,  
  "roll": 1.15,  
  "gyro_x": 120,  
  "gyro_y": -98,  
  "gyro_z": 5  
}
```

Esses dados são enviados continuamente pela USB.

## **5.2. Código “envia\_json.py” – Ponte Serial → InfluxDB**

Este script roda no computador e atua como middleware entre o ESP32 e o banco de dados.

## **Comunicação Serial**

- Abre a porta USB (**COM6**)
- Aguarda mensagens do ESP32
- Ignora logs que não sejam JSON

## **Processamento dos Dados**

Para cada linha válida:

1. Converte texto → dicionário Python
2. Valida campos
3. Cria um Point do InfluxDB
4. Envia para o banco MesaLabirinto

Exemplo de campos gravados:

- joy\_x, joy\_y
- servo\_x, servo\_y
- pitch, roll
- gyro\_x, gyro\_y, gyro\_z

Isso desacopla totalmente o ESP32 do banco de dados.

### **5.3. Painel Grafana (Painel-Mesa-Labirinto.json)**

Este arquivo define o dashboard de monitoramento em tempo real.

#### **- O que o Painel Mostra**

**Inclinação da Mesa**

- Gauge de Pitch
- Gauge de Roll
- Ajustes de offset e correção angular via Flux

### **Entrada do Usuário**

- Valor atual do joystick (X e Y)
- Histórico temporal

### **Atuadores**

- Ângulo atual de cada servo
- Visualização rápida de saturação

### **Séries Temporais**

- Evolução da inclinação
- Movimentos do joystick
- Resposta dinâmica do sistema

### **Atualização em Tempo Real**

- Refresh a cada 100 ms
- Ideal para:
  - Debug
  - Ajuste de parâmetros
  - Demonstração do sistema

Desse modo, este sistema demonstra de forma completa:

- ✓ **Uso de FreeRTOS**
- ✓ **Integração ADC + PWM + I2C**
- ✓ **Calibração dinâmica de sensores**
- ✓ **Filtros digitais (exponencial e complementar)**
- ✓ **Comunicação estruturada em JSON**
- ✓ **Pipeline de dados embarcado → PC → Banco → Dashboard**
- ✓ **Monitoramento profissional em tempo real**

## 6. Componentes e Bibliotecas Utilizados

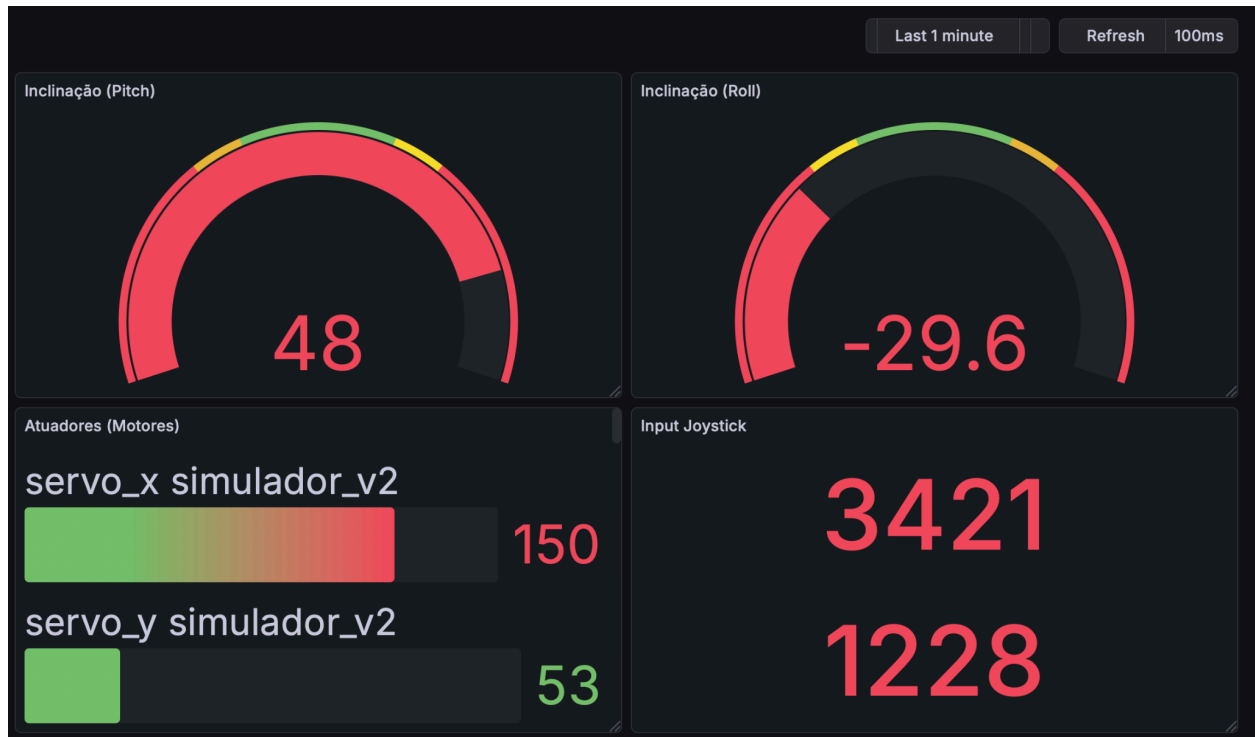
### Hardware

- **Microcontrolador:** ESP32-S3 (Espressif).
- **Atuadores:** Servomotores Padrão (ex: MG996R ou SG90).
- **Sensores:** Joystick Analógico KY-023, IMU MPU6050 (6 eixos).

### Software & Bibliotecas

- **Firmware:** ESP-IDF framework.
  - `driver/adc`: Leitura OneShot de alta precisão.
  - `driver/ledc`: Geração de PWM via hardware.
  - `driver/i2c`: Comunicação com MPU6050.
  - `freertos/semphr`: Gestão de concorrência (Mutex).
- **Backend:**
  - Python 3.10+ (`pyserial`, `influxdb-client`).
  - InfluxDB v2 (Time Series Database).
- **Frontend:**
  - Grafana (Visualização de dados).

## 7. Capturas de Tela do Grafana

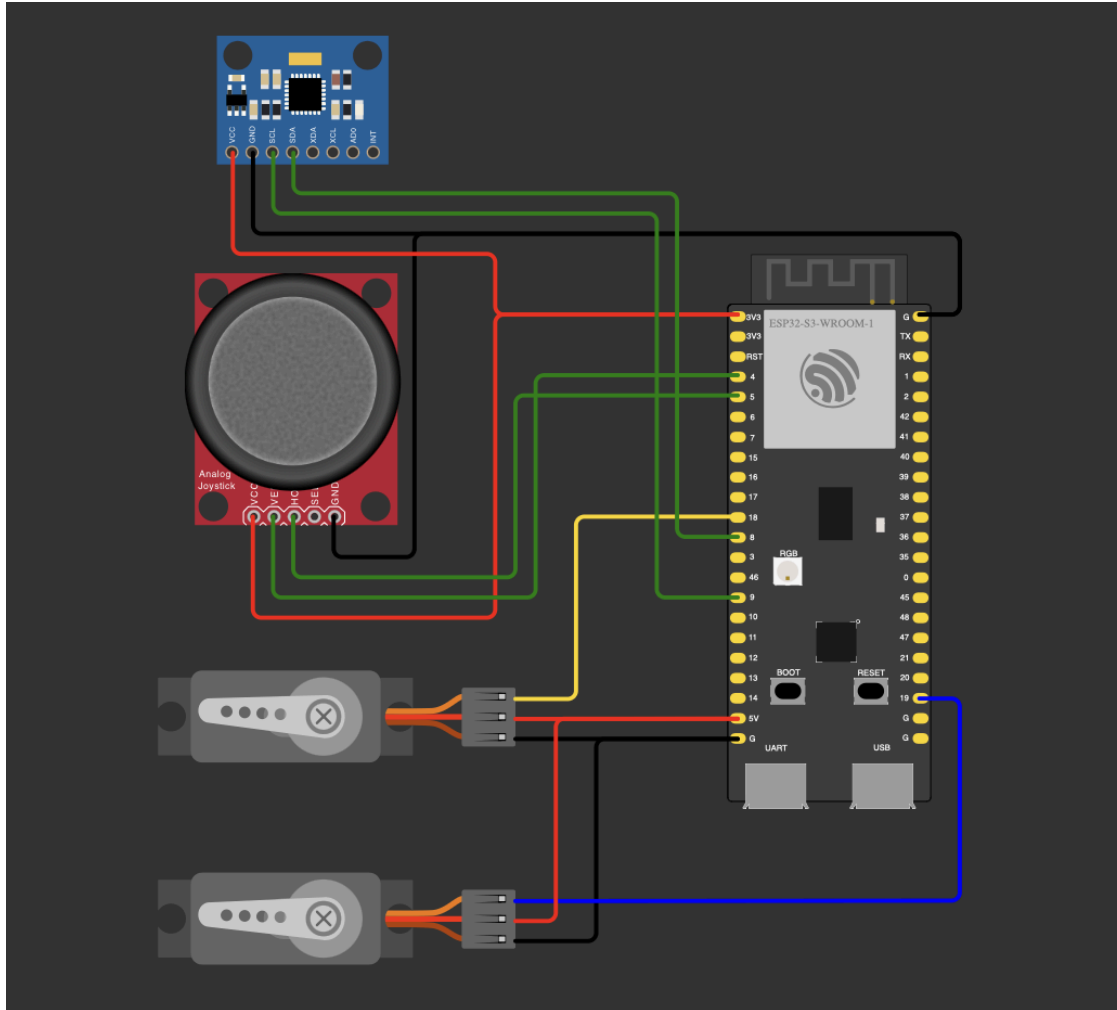


**Painel 1: Horizonte Artificial (Gauge)** Mostra a inclinação real da mesa (Pitch e Roll) lida pelo MPU6050. Configurado com zonas de cor (Verde = Nivelado, Vermelho = Inclinado).

**Painel 2: Monitor de Atuadores (Bar Gauge)** Barras horizontais que indicam a força/posição aplicada aos motores X e Y em tempo real.

**Painel 3: Histórico de Comandos (Time Series)** Gráfico de linhas mostrando a correlação entre o comando do Joystick (entrada) e a resposta da Mesa (saída) ao longo do tempo.

## 8. Fotos da Mesa em Operação



(Espaço reservado para fotos do seu projeto)

- **Foto 1:** Simulação no Wokwi.
- **Foto 2:** Detalhe da eletrônica (ESP32 e conexões).
- **Foto 3:** Visão geral do sistema (Mesa ao lado do Monitor com Grafana).

## 9. Dificuldades e Soluções Encontradas

Durante o desenvolvimento, os seguintes desafios técnicos foram superados:

### 1. Zona Morta e Descentralização do Joystick

- *Problema:* O joystick físico não retornava exatamente para o valor 2048, fazendo a mesa inclinar sozinha.



- *Solução:* Implementação de uma rotina de calibração no boot e criação de uma função de mapeamento assimétrico que considera uma "Deadzone" de software.

## 2. Ruído nas Leituras do Acelerômetro

- *Problema:* A vibração dos servos causava ruído na leitura do ângulo pelo MPU6050.
- *Solução:* Implementação de um **Filtro Complementar**. O filtro confia 98% na integração do giroscópio (preciso, mas tem deriva) e 2% no acelerômetro (ruidoso, mas não tem deriva), resultando em um sinal limpo.

## 3. Movimento Brusco dos Servos

- *Problema:* Mover o joystick rapidamente causava trancos na estrutura mecânica.
- *Solução:* Aplicação de suavização via software (Filtro Exponencial com fator 0.10) na task de controle dos servos, criando uma aceleração fluida.

## 4. Latência na Visualização de Dados

- *Problema:* Enviar muitos dados via Serial travava o processador.
- *Solução:* A tarefa de monitoramento (task\_monitor) foi configurada com prioridade baixa e executa apenas a cada 200ms, enviando dados formatados em JSON leve, garantindo que o controle dos motores (10ms) nunca seja interrompido.