INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DA PARAÍBA
CAMPUS CAMPINA GRANDE
BACHARELADO EM ENGENHARIA DA COMPUTAÇÃO

ARYEL SOUZA
KEVIN RYAN
PLÍNIO LIMA
THIAGO BARBOSA

RELATÓRIO TÉCNICO PROJETO SISTEMAS EMBARCADOS

Campina Grande - PB
2025

Diagrama de Blocos:

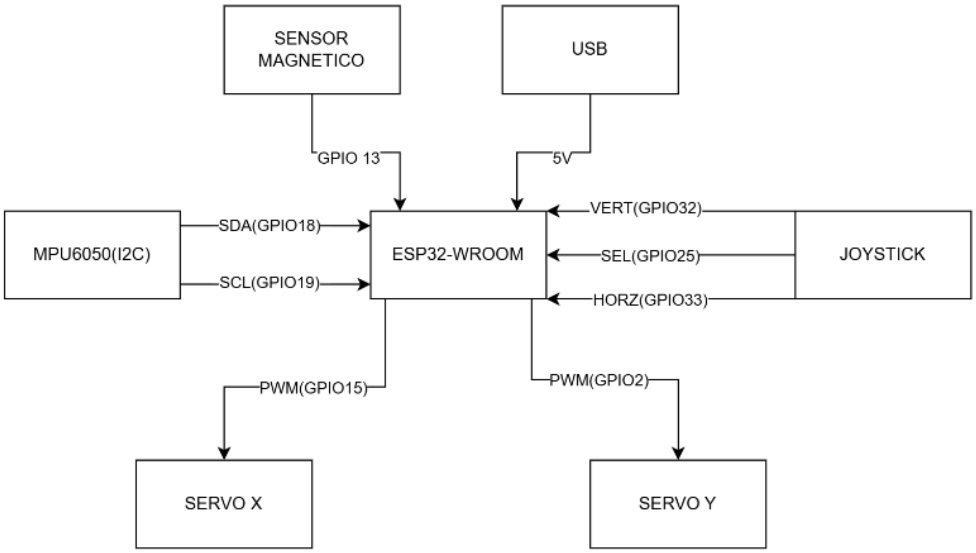
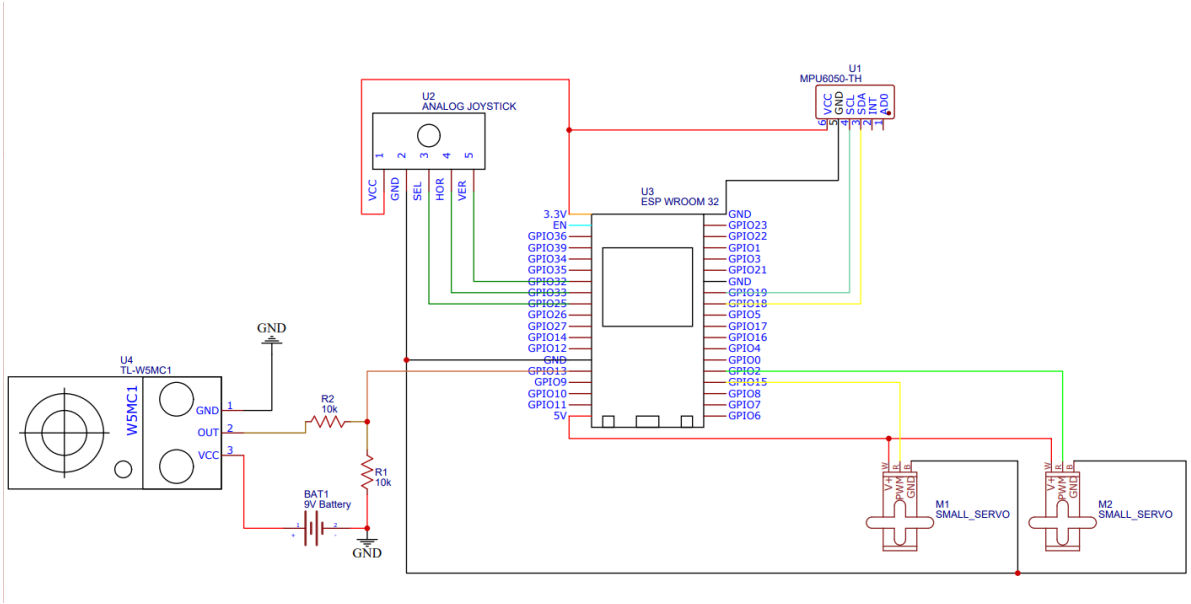


Diagrama Esquemático:



Tarefas e fluxos FreeRTOS:

Para atender todos os requisitos de avaliação do projeto do labirinto, foram desenvolvidas 6 tarefas para gerenciar toda a lógica e as funcionalidades dos sensores e atuadores, sendo elas:

- **mpu_reading_task:** Esta é a task responsável por estabelecer a comunicação I2C com o sensor mpu6050, sendo assim, também é responsável por realizar a leitura do acelerômetro e do giroscópio. Com relação ao processamento feito na task, executa a fusão sensorial

utilizando um filtro Complementar (98% Giroscópio, 2% Acelerômetro) para calcular os ângulos de inclinação (*Pitch* e *Roll*) estáveis, eliminando o *drift* do giroscópio e o ruído do acelerômetro.
A task foi classificada com prioridade alta (4), já que há várias funcionalidades ao decorrer do projeto que dependem dos dados vindos do mpu.

- **Joystick_read_task:** Tarefa responsável por realizar a leitura dos eixos X e Y do joystick, utilizando a o driver ADC, além disto, também monitora o botão do joystick. No processamento, a task aplica um filtro de média móvel (janela de 16 amostras) para suavizar o ruído do sinal elétrico do potenciômetro e implementa lógica de *debounce* para o acionamento do botão.
  A task tem prioridade alta (3), ficando abaixo do mpu.
- **win_check_task:** Responsável por monitorar o sinal enviado pelo sensor indutivo, responsável por indicar se o jogo foi ganho, ou não. No processamento implementa temporizadores lógicos. Para confirmar a vitória, a bola deve ser detectada continuamente por 2 segundos. Para reiniciar o jogo, a bola deve estar ausente por 2 segundos após a vitória. Essa histerese temporal evita falsos positivos.
  Também tem prioridade alta (3), para ficar sempre alerta do status do jogo.
- **servo_control_task:** responsável por converter os valores lidos no joystick em valores pwm para controlar os servos motores. Com relação ao processamento realiza:
  - Aplica uma Zona Morta (*Deadzone*) central para ignorar variações irrelevantes do joystick em repouso.
  - Mapeia os valores do ADC para o intervalo de *Duty Cycle* calibrado dos servos.
  - Aplica um Filtro Exponencial (fator 0.15) na saída, criando um comportamento de movimento suave ("efeito hidráulico"), evitando solavancos mecânicos bruscos na mesa.

  Esta task possui prioridade média (2), já que depende primeiramente de outra task.
  - **auto_calibration_task:** Task que pode ser ativada pelo usuário ao pressionar o botão do analógico 2 vezes consecutivas. Quando acionada assume o controle dos servos e realiza a calibração dos limites PWM que devem ser enviados aos servos. No processamento: realiza uma varredura automática dos eixos, monitorando o acelerômetro para detectar quando a mesa atinge seus limites mecânicos físicos, redefinindo dinamicamente os limites de PWM máximos e mínimos do software. Esta também possui uma prioridade média (2).

- **status_monitor_task:** Responsável por coletar os dados das outras tasks e formatar em uma string JSON e envia por UART. Ela é a responsável por permitir visualizações externas dos dados, como pelo Grafana, por exemplo, sem impactar na funcionalidade das outras tasks. Por depender de outras tasks e não poder “atrapalhar” as demais tasks possui prioridade baixa (1).

Para armazenar os dados de todas estas tasks foi criada uma struct global “system_data_t” e um semáforo binário “data_mutex”.

A struct centraliza os dados de entrada do joystick, estado dos sensores, parâmetros de calibração e estado lógico do jogo (ganho, ou não). Já o mecanismo do Mutex funciona da seguinte maneira: antes de qualquer task escrever ou ler da struct global, ela deve tomar o mutex, prevenindo assim problemas de condições de corrida e garantindo a integridade dos dados.

Funcionamento do sistema:

O sistema da mesa do labirinto possui diversos âmbitos, mas podemos resumir em três principais: Estrutura e Hardware, Software e sistema de visualização.

A estrutura da mesa é composta por um sistema de “tilt” e uma mesa em formato de labirinto. Vinculado a estrutura de tilt há dois servos motores, responsáveis por rotacionar a mesa. Acoplado a mesa há dois sensores, um MPU6050, responsável por coletar os dados de aceleração e giroscópio da mesa e um sensor indutivo, que é responsável por saber quando a esfera metálica chegar ao fim do labirinto, indicando vitória. Com relação ao software foi criada uma estrutura seguindo o padrão do ESPIDF, criando uma pasta “components” e dentro dela criando uma biblioteca para cada sensor ou atuador usado, estas serão melhor explicadas mais a frente no relatório. Para visualização dos dados foi feito um script python, responsável por coletar os dados JSON enviados pelo esp e salvar no InfluxDB, após isto o servidor Grafana acessava o InfluxDB e exibia os dados.

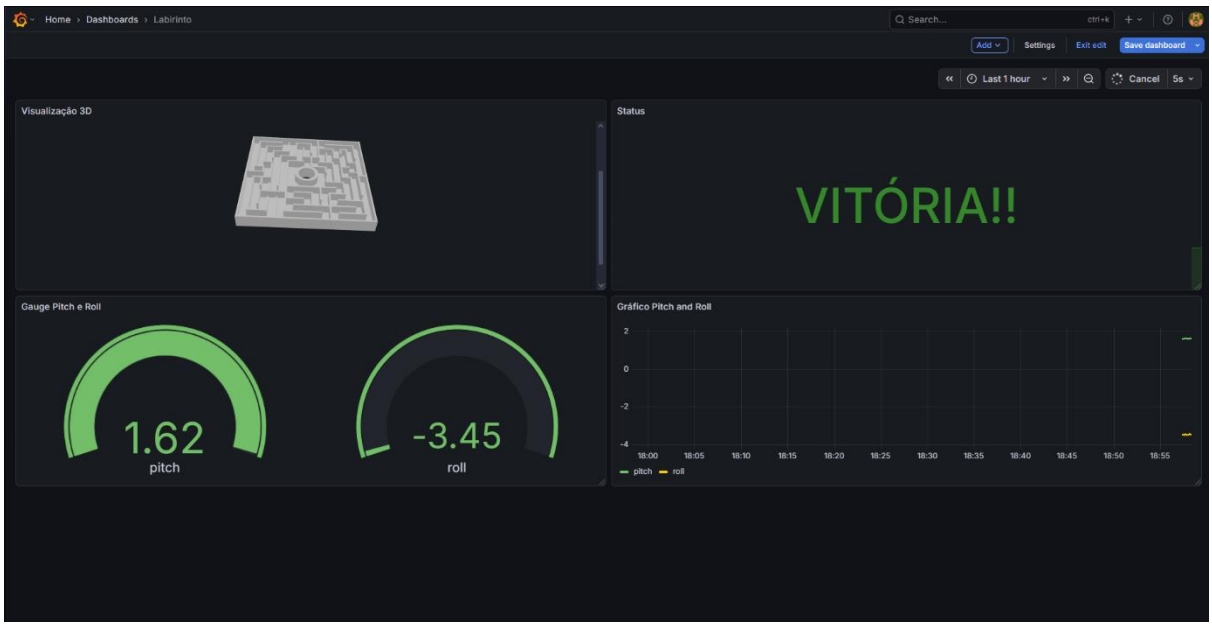
Descrição das bibliotecas:

- **system_commom:** Responsável por armazenar a struct global e o mutex, fornecendo dados para todas as outras bibliotecas e o controle das tasks.
- **joystick_driver:** Usando a biblioteca “esp_adc/adc_oneshot.h”, configura o adc para coletar os dados do joystick, além da implementação de um filtro de média móvel. Também monitora o botão do joystick e proporciona uma lógica de debounce para o mesmo.
- **servo_driver:** Realiza o controle dos servos, no caso, gerencia todo o movimento da mesa. Usando a biblioteca “driver/ledc.h” envia pulsos pwm para os motores, isso enquanto suaviza os movimentos para não ter um “tranco” na mesa. Nesta biblioteca também há a task de

autocalibração da mesa, onde por meio dos motores e dos dados do sensor MPU, encontra-se os máximos e mínimos físicos da mesa.

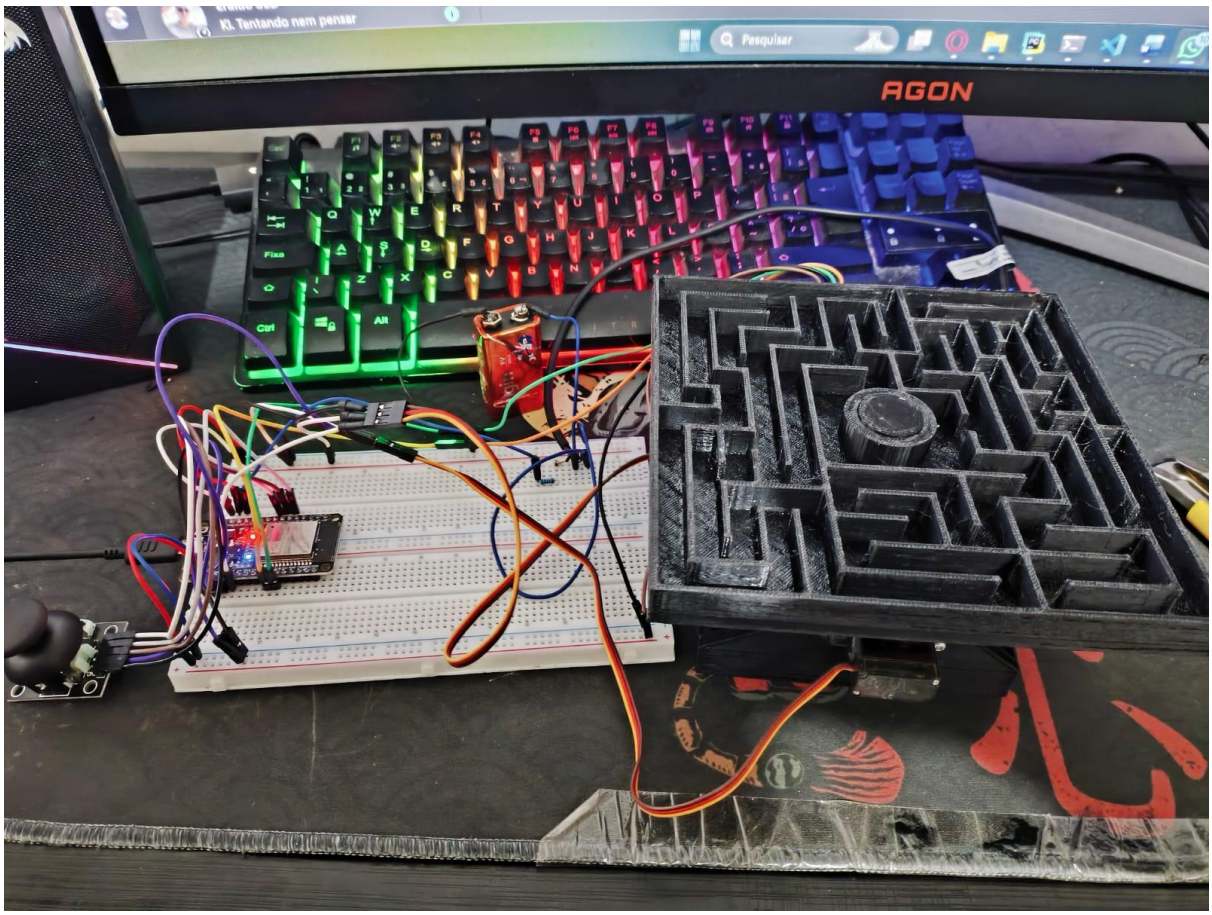
- **mpu_driver:** Utilizando a biblioteca “driver/i2c.h” é responsável por iniciar a comunicação I2C com o MPU. Aqui são feitos todos os cálculos para ajustar os dados enviados pelo MPU e reduzir ruídos e coisas do tipo.
- **game_logic:** Esta é a biblioteca responsável para indicar a vitória ou não no sistema. Utilizando a “driver/gpio.h” para ler o que for enviado pelo sensor indutivo e indicar se houve vitória, ou não.
- **main:** Aqui é onde tudo se concatena. A main, utilizando “freertos/FreeRTOS” e “task.h” inicia as tasks das bibliotecas e é responsável pela task de coleta de dados e exportação dos mesmos via JSON.

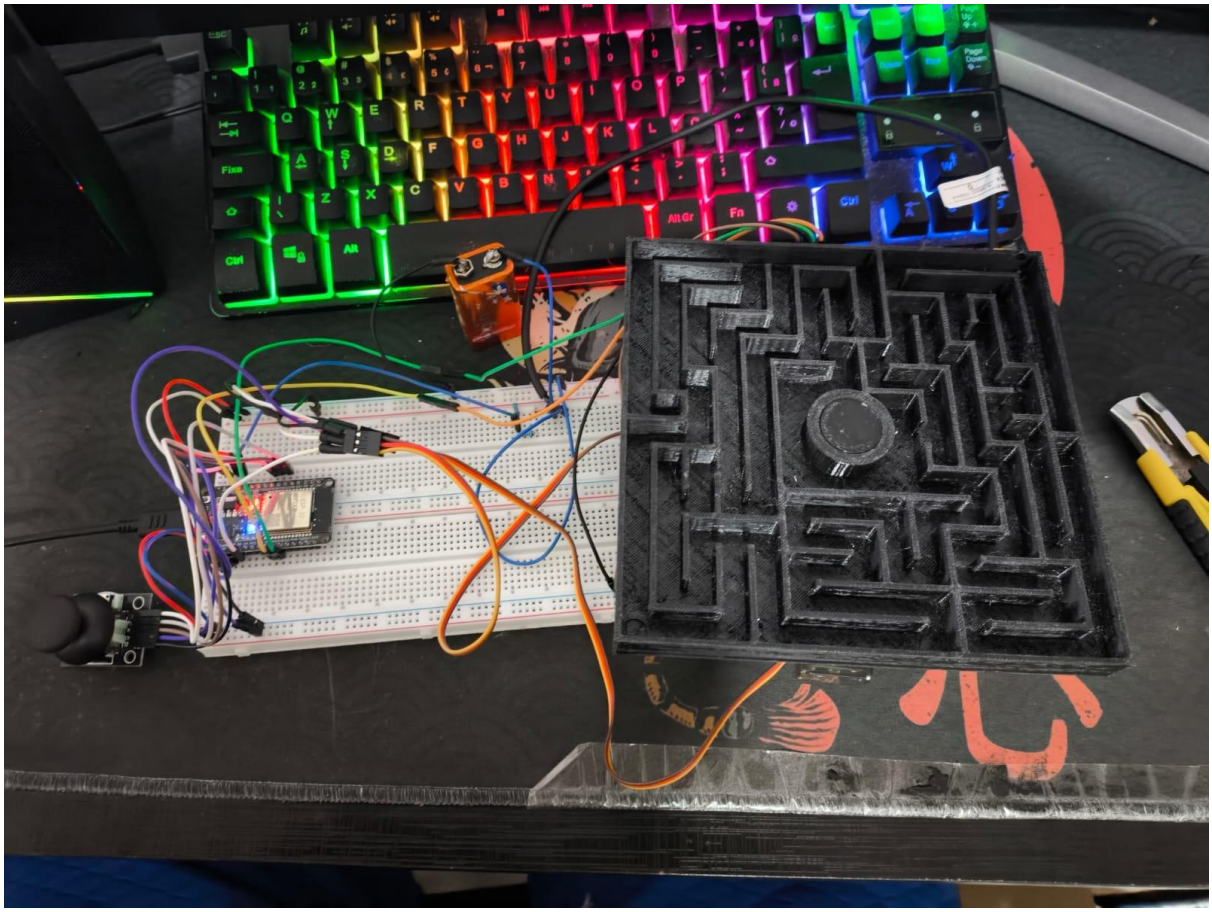
Grafana:

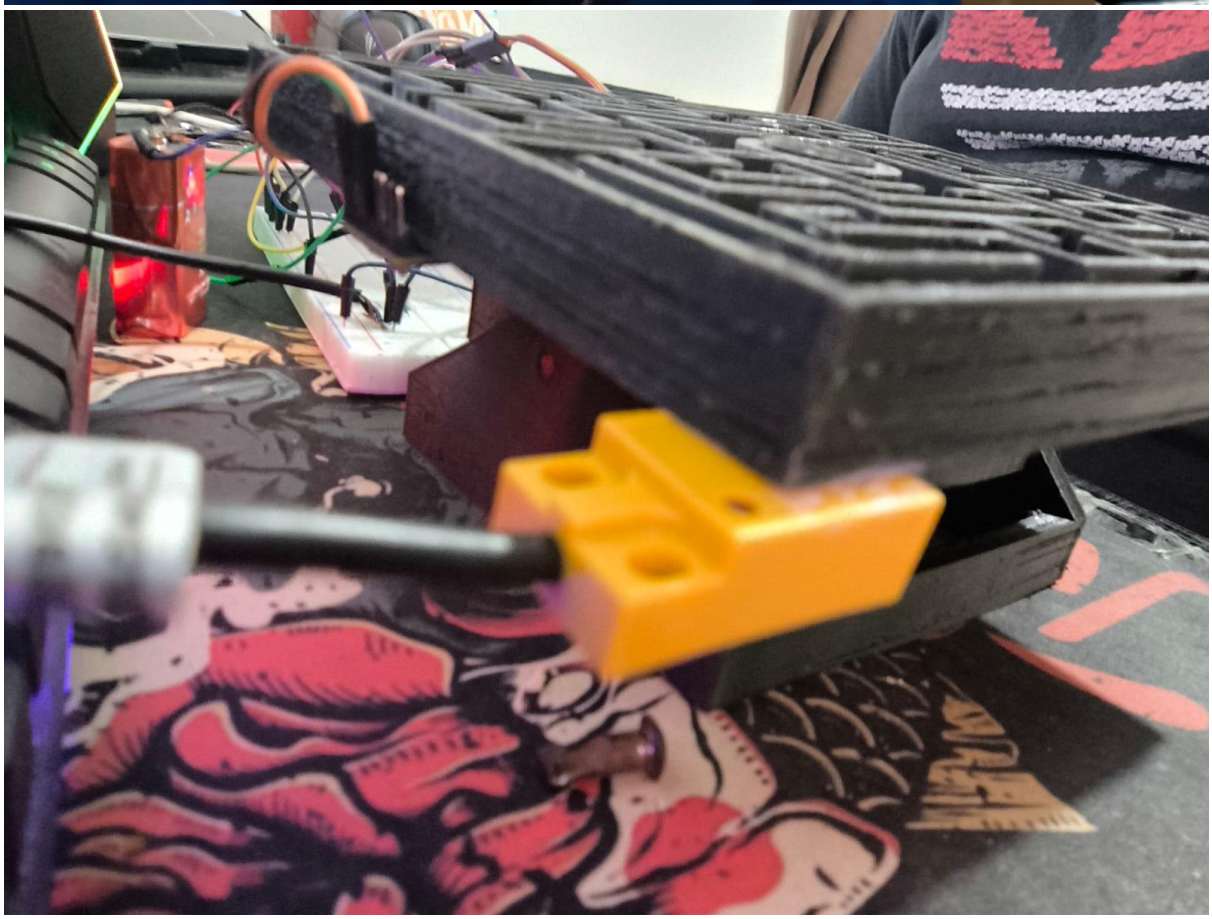


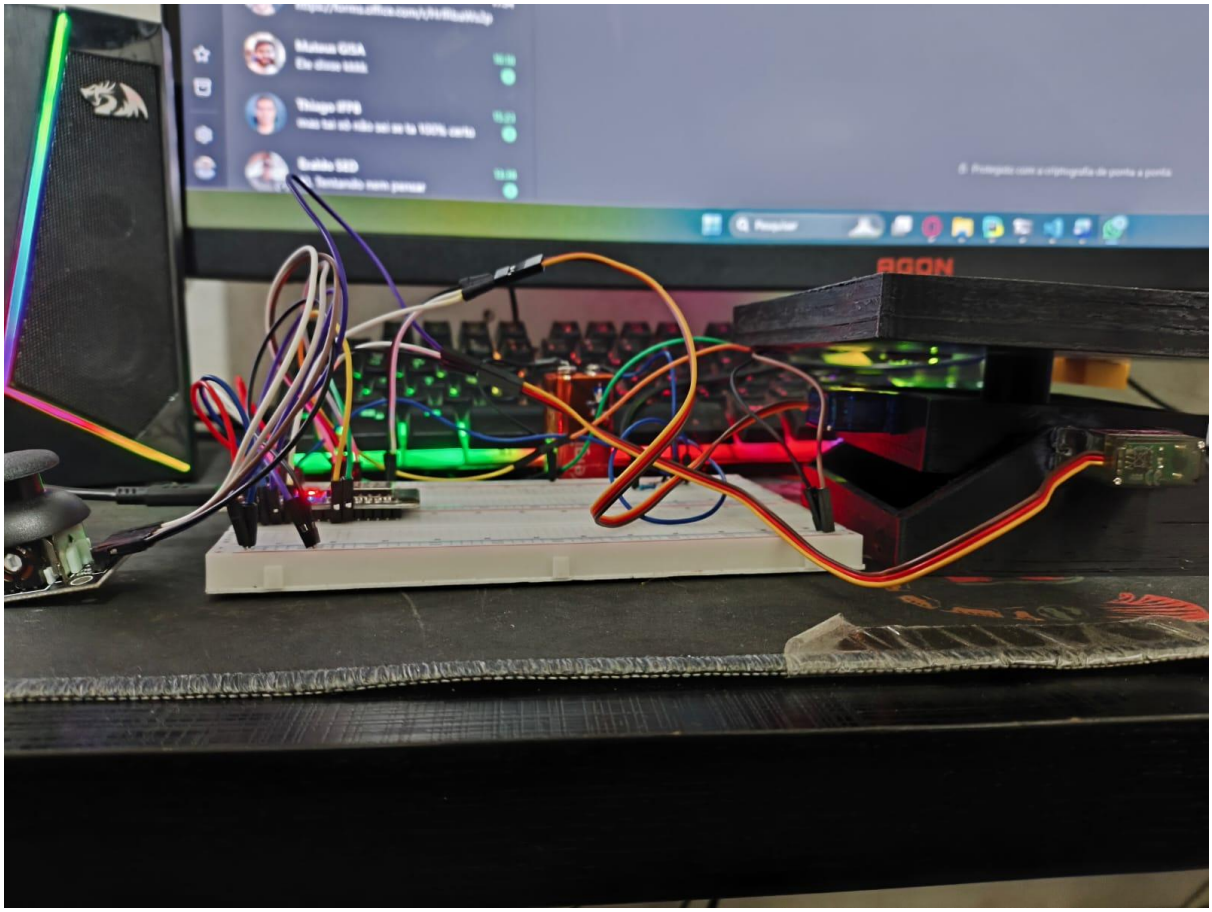
Fotos e vídeos da mesa:

Vídeo da mesa no Youtube: https://youtu.be/qgV5ViZv-Yw









Dificuldades e soluções:

As maiores dificuldades encontradas pelo nosso grupo estão separadas na parte física e na parte de visualização.

Na parte física o que mais gerou problemas e frustrações foram os mal contatos e as interferências dos fios utilizados. Talvez se houvesse uma PCB, ou algo do tipo, poderíamos ter evitado alguns problemas e incertezas nas horas de testagem/calibração.

Já na parte de visualização a maior dificuldade foi nunca ter trabalhado com o InfluxDB e o Grafana, tendo que aprender e enteder toda lógica do zero. Em contra partida, conseguimos ficar satisfeito com a exploração deste recurso e descobrir as possibilidades que podiam ser feitas, como a visualização 3D da mesa usando JavaScript, por exemplo.