Aplicação com sensores

Nome: **José Adriano Filho**
Matrícula: **2025101109806**
Unidade_3 - Capítulo_1

1. Leitura de Luminosidade com o Sensor BH1750: configurar o sensor BH1750 para medir a intensidade de luz, utilizando o RP2040. A programação do microcontrolador será feita para coletar os dados de luminosidade e exibi-los diretamente em um monitor serial, permitindo a observação em tempo real das variações na intensidade da luz. Além disso, como um desafio extra, programe o RP2040 para ajustar automaticamente a posição de um Servo Motor 9G SG90 em resposta ao nível de luz captado pelo sensor, de forma que o servo reaja dinamicamente conforme a iluminação detectada.

link: https://github.com/EngAdriano/Residencia/tree/main/Exercicios/U3C1Tarefas/bh1750_servo

R.: O código mostra um exemplo de utilização do sensor de luminosidade BH1750, o display oled, leitura de botão e o controle de posição com o micro servo NG90. Salientamos que o micro servo não tem posição fixa para o 0° e tem rotação de 360°, dificultando seu posicionamento de modo preciso. Para o projeto foi desenvolvido quatro bibliotecas que são: bh1750 para o sensor de luminosidade, flash para armazenamento na memória não volátil da BigDogLab do tempo para calibração, tentativa de melhorar o controle do servo, a biblioteca do servo para controle efetivamente do servo e ssd1306 para controle do display oled.

bh1750_servo.c > ...

```c
/* ----------------------------------------------------------------------------------------------------------------------
 * Projeto: bh1750_servo
 * Descrição: Este projeto utiliza um sensor de luminosidade BH1750 para controlar a posição de um servo contínuo simulado,
 * exibindo informações em um display OLED SSD1306. O servo é calibrado automaticamente no boot se um botão for pressionado,
 * e o tempo de rotação é salvo na memória flash para uso futuro. A luminosidade ambiente determina a posição do servo: menos
 * de 100 lux move o servo para 0 graus, entre 100 e 200 lux para 90 graus, e acima de 200 lux para 180 graus. Os valores dos
 * ângulos são aproximados pois o servo não é um padrão de posição.
 * Bibliotecas: pico-sdk, extras (servo_sim, flash_storage, ssd1306, bh1750).
 * Autor: José Adriano
 * Data de Criação: 06/09/2025
 *----------------------------------------------------------------------------------------------------------------------
 */
#include <stdio.h>
#include "pico/stdlib.h"
#include "hardware/i2c.h"
#include "servo_sim.h"
#include "flash_storage.h"
#include "ssd1306.h"
#include "bh1750.h"

// ==== Pinos ====
#define SERVO_PIN   2       // GPIO do servo contínuo (simulado)
#define BTN_CALIB   5       // Botão de calibração (ativo em nível baixo)

// I2C BH1750 (i2c0)
#define I2C_PORT_SENSOR i2c0
#define SDA_SENSOR 0
#define SCL_SENSOR 1

// I2C SSD1306 (i2c1)
#define I2C_PORT_OLED i2c1
#define SDA_OLED 14
#define SCL_OLED 15

```

Inicialmente fizemos os includes necessários e todos os defines para a utilização e configuração das funções desenvolvidas e de sistema.

```c
int main() {
    stdio_init_all();

    // Botão de calibração (ativo em nível baixo)
    gpio_init(BTN_CALIB);
    gpio_set_dir(BTN_CALIB, GPIO_IN);
    gpio_pull_up(BTN_CALIB);

    // I2C BH1750
    i2c_init(I2C_PORT_SENSOR, 100 * 1000);
    gpio_set_function(SDA_SENSOR, GPIO_FUNC_I2C);
    gpio_set_function(SCL_SENSOR, GPIO_FUNC_I2C);
    gpio_pull_up(SDA_SENSOR);
    gpio_pull_up(SCL_SENSOR);
    bh1750_init(I2C_PORT_SENSOR);
    sleep_ms(200);

    // I2C SSD1306
    i2c_init(I2C_PORT_OLED, 400000);
    gpio_set_function(SDA_OLED, GPIO_FUNC_I2C);
    gpio_set_function(SCL_OLED, GPIO_FUNC_I2C);
    gpio_pull_up(SDA_OLED);
    gpio_pull_up(SCL_OLED);
    ssd1306_init(I2C_PORT_OLED);

    // Tela inicial
    ssd1306_clear();
    ssd1306_draw_string(18, 0, "Embarcatech Servo");
    ssd1306_draw_string(8, 12, "Inicializando...");
    ssd1306_show();
```

Ao executar a main, inicialmente efetuamos as configurações das portas seriais uart/usb, i2c0 para controle do sensor de luminosidade, i2c1 para o display oled e o botão A para permitir a calibração onde o dado será armazenado na flash interna do microcontrolador RP2040. Este dado será utilizado como referência para o ajuste do servo.

```c

    // Servo: carrega calibração da flash (ou usa 1000ms padrão)
    uint32_t rotation_time_ms = 1000;
    bool have_calib = flash_storage_read(&rotation_time_ms);

    servo_sim_t servo;
    servo_sim_init(&servo, SERVO_PIN, (float)rotation_time_ms);

    // Se botão pressionado no boot → calibrar e salvar
    if (!gpio_get(BTN_CALIB)) {
        ssd1306_clear();
        ssd1306_draw_string(20, 24, "Calibrando...");
        ssd1306_show();

        servo_sim_calibrate(&servo);
        rotation_time_ms = (uint32_t)(180.0f / servo.deg_per_ms);
        flash_storage_write(rotation_time_ms);

        ssd1306_clear();
        ssd1306_draw_string(8, 24, "Calibracao salva!");
        ssd1306_show();
        sleep_ms(1200);
    } else if (have_calib) {
        char msg[24];
        snprintf(msg, sizeof(msg), "Calib: %ums", rotation_time_ms);
        ssd1306_clear();
        ssd1306_draw_string(10, 24, msg);
        ssd1306_show();
        sleep_ms(800);
    }

```

Esta etapa do código será responsável por efetuar a calibração, deslocando o servo durante um período de tempo em milissegundos que será armazenado na flash para uso durante o loop infinito.

```
95
96      while (true) {
97          // Lê luminosidade
98          float lux = bh1750_read_lux(I2C_PORT_SENSOR);
99
100         // Mapeia para 0/90/180 graus
101         float angle = (lux < 100) ? 0.0f : (lux < 200) ? 90.0f : 180.0f;
102
103         // Envia pela serial os valores
104         printf("Lux: %.1f, Alvo: %.0f deg, t180: %ums\n", lux, angle, (uint32_t)(180.0f / servo.deg_per_ms));
105
106         // Move (simulado)
107         servo_sim_set_angle(&servo, angle);
108
109         // Display
110         ssd1306_clear();
111         ssd1306_draw_string(18, 0, "Embarcatech Servo");
112         char line1[24], line2[24], line3[24];
113         snprintf(line1, sizeof(line1), "Lux: %.1f", lux);
114         snprintf(line2, sizeof(line2), "Alvo: %.0f deg", angle);
115         snprintf(line3, sizeof(line3), "t180: %ums", (uint32_t)(180.0f / servo.deg_per_ms));
116         ssd1306_draw_string(6, 20, line1);
117         ssd1306_draw_string(6, 36, line2);
118         ssd1306_draw_string(6, 52, line3);
119         ssd1306_show();
120
121         sleep_ms(1200);
122     }
123 }
124
```

Por último temos o loop infinito onde executamos as tarefas pedidas no exercício: ler o sensor de luminosidade em lux, de posse deste dado posicionamos o servo de acordo com o valor lido: menor que 100 lux em uma posição simulando 0°, entre 101 e 200 lux para uma posição simulando 90° e acima disso uma posição simulando 180°.

No loop infinito também enviamos via serial para o terminal, requisito do exercício, como mostramos as informações no oled da placa.

2. Monitoramento de Temperatura e Umidade com o Sensor AHT10: configurar o sensor AHT10 para monitorar tanto a temperatura quanto a umidade do ambiente, utilizando o RP2040. A programação do microcontrolador deverá coletar e exibir esses dados em uma tela LCD 320x240, possibilitando uma visualização em tempo real. Como um desafio adicional, ajuste o código para que o LCD exiba um aviso caso a umidade ultrapasse 70% ou a temperatura fique abaixo de 20°C, criando uma interface que alerta o usuário sobre condições específicas do ambiente.

link: https://github.com/EngAdriano/Residencia/tree/main/Exercicios/U3C1Tarefas/AHT10_temp_umidade

R.: Neste programa agora, mostramos o uso de sensor AHT10 que monitora a temperatura e a umidade do ambiente. Utilizamos duas bibliotecas: AHT10 e ssd1306, que são responsáveis por abstrair o funcionamento do hardware do sensor e do display Oled utilizado na placa BitDogLab.

```c
/*
 * Projeto: Sensor AHT10 com OLED SSD1306
 * Descrição: Este código lê a temperatura e umidade do sensor AHT10 e exibe os dados em um display OLED SSD1306.
 * Bibliotecas: aht10, ssd1306
 * Autor: José Adriano
 * Data de Criação: 26/07/2025
 */

#include <stdio.h>
#include "pico/stdlib.h"
#include "hardware/i2c.h"
#include "aht10.h"
#include "ssd1306.h"

// I2C usado: I2C0 com SDA=GPIO4, SCL=GPIO5
#define I2C_PORT0 i2c0
#define I2C_SDA0 0
#define I2C_SCL0 1

#define I2C_PORT1 i2c1
#define I2C_SDA1 14
#define I2C_SCL1 15

// Prototipos das funções I2C
int i2c_write(uint8_t addr, const uint8_t *data, uint16_t len);
int i2c_read(uint8_t addr, uint8_t *data, uint16_t len);
void delay_ms(uint32_t ms);
```

Da mesma forma que o programa anterior, fizemos os includes necessários e as definições para utilizar nas funções necessárias, neste caso também abrimos uma área para os protótipos de funções já que criamos algumas que estão no final do arquivo principal, sendo necessário indicar para o compilador a existência das mesmas.

```c
int main() {
    stdio_init_all();

    // Inicializa I2C sensor
    i2c_init(I2C_PORT0, 100 * 1000); // 100 kHz
    gpio_set_function(I2C_SDA0, GPIO_FUNC_I2C);
    gpio_set_function(I2C_SCL0, GPIO_FUNC_I2C);
    gpio_pull_up(I2C_SDA0);
    gpio_pull_up(I2C_SCL0);

    // Inicializa I2C OLED
    i2c_init(I2C_PORT1, 400000);
    gpio_set_function(I2C_SDA1, GPIO_FUNC_I2C);
    gpio_set_function(I2C_SCL1, GPIO_FUNC_I2C);
    gpio_pull_up(I2C_SDA1);
    gpio_pull_up(I2C_SCL1);

    ssd1306_init(I2C_PORT1);
    ssd1306_clear();
    ssd1306_draw_string(32, 0, "Embarcatech");
    ssd1306_draw_string(20, 10, "Inicializando...");
    ssd1306_show();

    // Define estrutura do sensor
    AHT10_Handle aht10 = {
        .iface = {
            .i2c_write = i2c_write,
            .i2c_read = i2c_read,
            .delay_ms = delay_ms
        }
    };

    printf("Inicializando AHT10...\n");
    if (!AHT10_Init(&aht10)) {
        printf("Falha na inicialização do sensor!\n");
        ssd1306_clear();
        ssd1306_draw_string(32, 0, "Embarcatech");
        ssd1306_draw_string(23, 30, "Falha no AHT10");
        ssd1306_show();
        while (1) sleep_ms(1000);
    }

```

No main executamos todas as inicializações necessárias como a i2c0 para a leitura do sensor AHT10, a i2c1 para o oled. Inicializamos também a comunicação com o sensor afim de verificar a sua presença.

```c
    while (1) {
        float temp, hum;
        if (AHT10_ReadTemperatureHumidity(&aht10, &temp, &hum)) {
            printf("Temperatura: %.2f °C | Umidade: %.2f %%\n", temp, hum);
        } else {
            printf("Falha na leitura dos dados!\n");
        }
            while(hum > 70){
                ssd1306_clear();
                ssd1306_draw_string(32, 0, "Embarcatech");
                ssd1306_draw_string(30, 10, "AHT10 Sensor");
                ssd1306_draw_string(0, 20, "Umidade");
                char hum_str[16];
                snprintf(hum_str, sizeof(hum_str), "%.2f %%", hum);
                ssd1306_draw_string(85, 20, hum_str);
                ssd1306_draw_string(22,40, "Acima de 70 %");
                ssd1306_draw_string(40,50, "ATENCAO");
                ssd1306_show();
                sleep_ms(500);
                ssd1306_draw_string(40,50, "       ");
                ssd1306_show();
                sleep_ms(500);
                AHT10_ReadTemperatureHumidity(&aht10, &temp, &hum);
            }

        while(temp < 20){
                ssd1306_clear();
                ssd1306_draw_string(32, 0, "Embarcatech");
                ssd1306_draw_string(30, 10, "AHT10 Sensor");
                ssd1306_draw_string(0, 20, "Temperatura");
                char hum_str[16];
                snprintf(hum_str, sizeof(hum_str), "%.2f C", temp);
                ssd1306_draw_string(85, 20, hum_str);
                ssd1306_draw_string(20,40, "Abaixo de 20 C");
                ssd1306_draw_string(40,50, "ATENCAO");
                ssd1306_show();
                sleep_ms(500);
                ssd1306_draw_string(40,50, "       ");
                ssd1306_show();
                sleep_ms(500);
                AHT10_ReadTemperatureHumidity(&aht10, &temp, &hum);
            }
```

No loop infinito fazemos a leitura do sensor, indicando erro se houver, enviamos para a serial. Outro requisito satisfeito neste início do loop é o de testar se a umidade está acima dos 70% ou se a temperatura está abaixo de 20°, enviando uma alerta no display oled.

```
        ssd1306_clear();
        ssd1306_draw_string(32, 0, "Embarcatech");
        ssd1306_draw_string(30, 10, "AHT10 Sensor");
        ssd1306_draw_string(0, 30, "Temperatura");
        char temp_str[16];
        snprintf(temp_str, sizeof(temp_str), "%.2f C", temp);
        ssd1306_draw_string(85, 30, temp_str);
        ssd1306_draw_string(0, 50, "Umidade");
        char hum_str[16];
        snprintf(hum_str, sizeof(hum_str), "%.2f %%", hum);
        ssd1306_draw_string(85, 50, hum_str);
        ssd1306_show();

        sleep_ms(1000);
    }
}
```

Fechamos o loop infinito com a apresentação dos dados no display oled. As atualizações ocorrem a cada 1s. Abaixo as funções de leitura para i2c.

```
// Função para escrita I2C
int i2c_write(uint8_t addr, const uint8_t *data, uint16_t len) {
    int result = i2c_write_blocking(I2C_PORT0, addr, data, len, false);
    return result < 0 ? -1 : 0;
}

// Função para leitura I2C
int i2c_read(uint8_t addr, uint8_t *data, uint16_t len) {
    int result = i2c_read_blocking(I2C_PORT0, addr, data, len, false);
    return result < 0 ? -1 : 0;
}

// Função para delay
void delay_ms(uint32_t ms) {
    sleep_ms(ms);
}
```

3. GPS e Display de Localização: configurar o módulo GY-NEO6MV2 (GPS) para coletar dados de localização e exibi-los na tela LCD 320x240, conectada ao RP2040. A programação deve mostrar, em tempo real, as coordenadas de latitude e longitude na tela, possibilitando o monitoramento da posição. Como desafio extra, programe o sistema para registrar os dados de localização em um cartão SD conectado ao sistema via SPI IDC, permitindo a criação de um histórico de coordenadas armazenado para consultas posteriores.

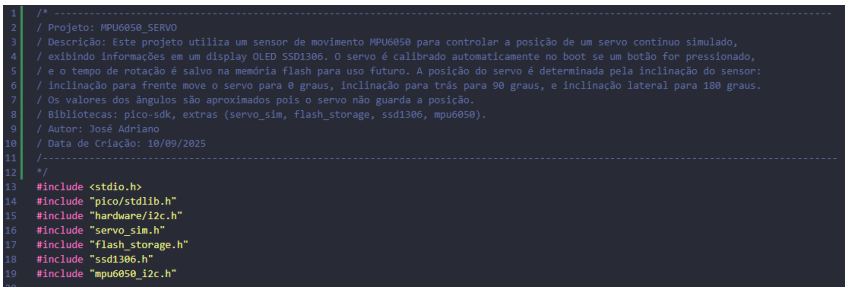
R.: Esta questão não foi desenvolvida pela razão de não termos recebido os materiais necessários como o módulo GPS GY-NEO6MV2 e o display LCD 320x240.

4. Controle e Monitoramento de Movimento com MPU6050: configurar o sensor MPU6050 para captar dados de movimento, como aceleração e rotação, e exibir as leituras de inclinação no monitor serial. Além disso, programe o RP2040 para ajustar a posição de um Servo Motor 9G SG90 em função do ângulo de inclinação detectado, promovendo um controle dinâmico de movimento em resposta à orientação do sensor. No desafio extra, você deverá adicionar um alerta visual na tela LCD 320x240 que

indique quando o sistema ultrapassa um determinado ângulo de inclinação, criando um sistema de monitoramento visual das mudanças de posição.

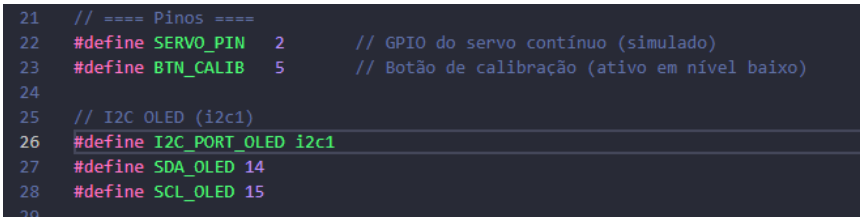
link: https://github.com/EngAdriano/Residencia/tree/main/Exercicios/U3C1Tarefas/MPU6050_Servo

R.: Este programa trabalha com o sensor MPU6050 – acelerômetro e giroscópio, totalizando 6 graus de liberdade para medir movimento e orientação. Detecta aceleração linear nos eixos X, Y e Z (acelerômetro) e a taxa de rotação angular nos mesmos eixos (giroscópio). Comunicando com o microcontrolador RP2040 pela interface serial I2C.



```c
/* ------------------------------------------------------------------------------------------------------------------------
 * Projeto: MPU6050_SERVO
 * Descrição: Este projeto utiliza um sensor de movimento MPU6050 para controlar a posição de um servo contínuo simulado,
 * exibindo informações em um display OLED SSD1306. O servo é calibrado automaticamente no boot se um botão for pressionado,
 * e o tempo de rotação é salvo na memória flash para uso futuro. A posição do servo é determinada pela inclinação do sensor:
 * inclinação para frente move o servo para 0 graus, inclinação para trás para 90 graus, e inclinação lateral para 180 graus.
 * Os valores dos ângulos são aproximados pois o servo não guarda a posição.
 * Bibliotecas: pico-sdk, extras (servo_sim, flash_storage, ssd1306, mpu6050).
 * Autor: José Adriano
 * Data de Criação: 10/09/2025
 *------------------------------------------------------------------------------------------------------------------------
 */
#include <stdio.h>
#include "pico/stdlib.h"
#include "hardware/i2c.h"
#include "servo_sim.h"
#include "flash_storage.h"
#include "ssd1306.h"
#include "mpu6050_i2c.h"
```

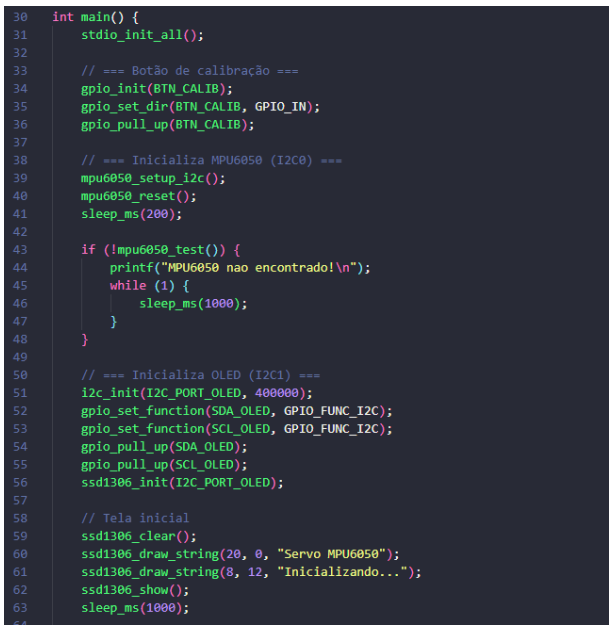
Nesta tela temos informações sobre o projeto e os includes necessários para o funcionamento correto do programa. Devemos salientar que o micro servo utilizado não é o mais apropriado para o projeto pois não tem as características necessárias para uma boa execução, mas serve para testes.



```c
// ==== Pinos ====
#define SERVO_PIN   2       // GPIO do servo contínuo (simulado)
#define BTN_CALIB   5       // Botão de calibração (ativo em nível baixo)

// I2C OLED (i2c1)
#define I2C_PORT_OLED i2c1
#define SDA_OLED 14
#define SCL_OLED 15
```

Gpios utilizados para acesso ao Oled, ao botão de calibragem e o acionamento do servo.



```c
int main() {
    stdio_init_all();

    // === Botão de calibração ===
    gpio_init(BTN_CALIB);
    gpio_set_dir(BTN_CALIB, GPIO_IN);
    gpio_pull_up(BTN_CALIB);

    // === Inicializa MPU6050 (I2C0) ===
    mpu6050_setup_i2c();
    mpu6050_reset();
    sleep_ms(200);

    if (!mpu6050_test()) {
        printf("MPU6050 nao encontrado!\n");
        while (1) {
            sleep_ms(1000);
        }
    }

    // === Inicializa OLED (I2C1) ===
    i2c_init(I2C_PORT_OLED, 400000);
    gpio_set_function(SDA_OLED, GPIO_FUNC_I2C);
    gpio_set_function(SCL_OLED, GPIO_FUNC_I2C);
    gpio_pull_up(SDA_OLED);
    gpio_pull_up(SCL_OLED);
    ssd1306_init(I2C_PORT_OLED);

    // Tela inicial
    ssd1306_clear();
    ssd1306_draw_string(20, 0, "Servo MPU6050");
    ssd1306_draw_string(8, 12, "Inicializando...");
    ssd1306_show();
    sleep_ms(1000);
```

Ao iniciar o programa pela função principal main, temos que fazer as configurações necessárias: botão, gpio do servo, sensor MPU6050 e o display Oled, bem como criar a tela inicial.

```
    // === Servo: carrega calibração ===
    uint32_t rotation_time_ms = 1000;
    bool have_calib = flash_storage_read(&rotation_time_ms);

    servo_sim_t servo;
    servo_sim_init(&servo, SERVO_PIN, (float)rotation_time_ms);

    if (!gpio_get(BTN_CALIB)) {
        ssd1306_clear();
        ssd1306_draw_string(20, 24, "Calibrando...");
        ssd1306_show();

        servo_sim_calibrate(&servo);
        rotation_time_ms = (uint32_t)(180.0f / servo.deg_per_ms);
        flash_storage_write(rotation_time_ms);

        ssd1306_clear();
        ssd1306_draw_string(8, 24, "Calibracao salva!");
        ssd1306_show();
        sleep_ms(1200);
    } else if (have_calib) {
        char msg[24];
        snprintf(msg, sizeof(msg), "Calib: %ums", rotation_time_ms);
        ssd1306_clear();
        ssd1306_draw_string(10, 24, msg);
        ssd1306_show();
        sleep_ms(800);
    }

    float current_angle = 90.0f;  // posição inicial

```

Nesta parte do programa podemos fazer a calibração necessária para tentar controlar melhor o servo.

```
    while (true) {
        int16_t accel[3], gyro[3], temp_raw;
        mpu6050_read_raw(accel, gyro, &temp_raw); // ainda lemos temp_raw mas ignoramos

        // Escolha de ângulo: simples → aceleração no eixo X
        float ax = accel[0] / ACCEL_SENS_2G; // normalizado em "g"
        float target_angle = (ax < -0.5f) ? 0.0f : (ax < 0.5f ? 90.0f : 180.0f);

        // Movimento suave → incrementa gradualmente
        if (current_angle < target_angle) current_angle += 2.0f;
        else if (current_angle > target_angle) current_angle -= 2.0f;

        // Serial debug
        printf("AX=%.2fg AY=%.2fg AZ=%.2fg | GX=%d GY=%d GZ=%d | Alvo=%.0f deg | Atual=%.0f deg\n",
               accel[0]/ACCEL_SENS_2G, accel[1]/ACCEL_SENS_2G, accel[2]/ACCEL_SENS_2G,
               gyro[0], gyro[1], gyro[2], target_angle, current_angle);

        // Servo simulado
        servo_sim_set_angle(&servo, current_angle);

        // Display
        ssd1306_clear();
        ssd1306_draw_string(20, 0, "Servo MPU6050");

        char line1[24], line2[24];
        snprintf(line1, sizeof(line1), "AX: %.2fg", accel[0]/ACCEL_SENS_2G);
        snprintf(line2, sizeof(line2), "Ang: %.0f/%0.f", current_angle, target_angle);

        ssd1306_draw_string(6, 20, line1);
        ssd1306_draw_string(6, 36, line2);
        ssd1306_show();

        sleep_ms(80); // controle da velocidade do movimento
    }
}

```

Por fim temos o loop infinito onde efetuamos a leitura dos dados do sensor, executamos o tratamento dos dados brutos para podermos tirar o ângulo correto, atualizamos o display e setamos o ângulo para movimentar o servo.

O programa completo está no github do qual o link está no início da resposta desta questão.

5. (**Questão desafio**) - Comunicação LoRa e Alerta de Proximidade: configure o sensor de proximidade VI530X em conjunto com o módulo LoRa 915MHz para detectar a presença de objetos próximos e enviar alertas por meio de comunicação LoRa. A programação no RP2040 deve ativar o envio de um alerta ao detectar a presença de um objeto a menos de 10 cm, transmitindo a informação por LoRa. Como desafio adicional, simule a recepção do alerta em outro dispositivo e registre todas as ocorrências de proximidade em um cartão SD, criando um registro histórico de detecções de proximidade para análise futura.

Link: https://github.com/EngAdriano/Residencia/tree/main/Exercicios/U3C1Tarefas/vl53l0x_lcd

R.: Esta questão não foi desenvolvida pela razão de não termos recebido os materiais necessários como o módulo LoRa de 915 MHz.

Mesmo assim coloquei o link para os testes com o sensor VL53l0X – distância a laser.