

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.

```
C bh1750_servo.c > ...
1  /* -----
2  / Projeto: bh1750_servo
3  / Descrição: Este projeto utiliza um sensor de luminosidade BH1750 para controlar a posição de um servo contínuo simulado,
4  / exibindo informações em um display OLED SSD1306. O servo é calibrado automaticamente no boot se um botão for pressionado,
5  / e o tempo de rotação é salvo na memória flash para uso futuro. A luminosidade ambiente determina a posição do servo: menos
6  / 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
7  / ângulos são aproximados pois o servo não é um padrão de posição.
8  / Bibliotecas: pico-sdk, extras (servo_sim, flash_storage, ssd1306, bh1750).
9  / Autor: José Adriano
10 / Data de Criação: 06/09/2025
11 /-----
12 */
13 #include <stdio.h>
14 #include "pico/stdlib.h"
15 #include "hardware/i2c.h"
16 #include "servo_sim.h"
17 #include "flash_storage.h"
18 #include "ssd1306.h"
19 #include "bh1750.h"
20
21 // ==== Pinos ====
22 #define SERVO_PIN 2 // GPIO do servo contínuo (simulado)
23 #define BTN_CALIB 5 // Botão de calibração (ativo em nível baixo)
24
25 // I2C BH1750 (i2c0)
26 #define I2C_PORT_SENSOR i2c0
27 #define SDA_SENSOR 0
28 #define SCL_SENSOR 1
29
30 // I2C SSD1306 (i2c1)
31 #define I2C_PORT_OLED i2c1
32 #define SDA_OLED 14
33 #define SCL_OLED 15
34
```

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

```

35 int main() {
36     stdio_init_all();
37
38     // Botão de calibração (ativo em nível baixo)
39     gpio_init(BTN_CALIB);
40     gpio_set_dir(BTN_CALIB, GPIO_IN);
41     gpio_pull_up(BTN_CALIB);
42
43     // I2C BH1750
44     i2c_init(I2C_PORT_SENSOR, 100 * 1000);
45     gpio_set_function(SDA_SENSOR, GPIO_FUNC_I2C);
46     gpio_set_function(SCL_SENSOR, GPIO_FUNC_I2C);
47     gpio_pull_up(SDA_SENSOR);
48     gpio_pull_up(SCL_SENSOR);
49     bh1750_init(I2C_PORT_SENSOR);
50     sleep_ms(200);
51
52     // I2C SSD1306
53     i2c_init(I2C_PORT_OLED, 400000);
54     gpio_set_function(SDA_OLED, GPIO_FUNC_I2C);
55     gpio_set_function(SCL_OLED, GPIO_FUNC_I2C);
56     gpio_pull_up(SDA_OLED);
57     gpio_pull_up(SCL_OLED);
58     ssd1306_init(I2C_PORT_OLED);
59
60     // Tela inicial
61     ssd1306_clear();
62     ssd1306_draw_string(18, 0, "Embarcotech Servo");
63     ssd1306_draw_string(8, 12, "Inicializando...");
64     ssd1306_show();
65

```

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.

```

65
66     // Servo: carrega calibração da flash (ou usa 1000ms padrão)
67     uint32_t rotation_time_ms = 1000;
68     bool have_calib = flash_storage_read(&rotation_time_ms);
69
70     servo_sim_t servo;
71     servo_sim_init(&servo, SERVO_PIN, (float)rotation_time_ms);
72
73     // Se botão pressionado no boot + calibrar e salvar
74     if (!gpio_get(BTN_CALIB)) {
75         ssd1306_clear();
76         ssd1306_draw_string(20, 24, "Calibrando...");
77         ssd1306_show();
78
79         servo_sim_calibrate(&servo);
80         rotation_time_ms = (uint32_t)(180.0f / servo.deg_per_ms);
81         flash_storage_write(rotation_time_ms);
82
83         ssd1306_clear();
84         ssd1306_draw_string(8, 24, "Calibracao salva!");
85         ssd1306_show();
86         sleep_ms(1200);
87     } else if (have_calib) {
88         char msg[24];
89         snprintf(msg, sizeof(msg), "Calib: %ums", rotation_time_ms);
90         ssd1306_clear();
91         ssd1306_draw_string(10, 24, msg);
92         ssd1306_show();
93         sleep_ms(800);
94     }
95

```

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, "Embarcotech 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.

```

src > C AHT10_temp_umidade.c > I2C_SDA0
1  /* -----
2  / Projeto: Sensor AHT10 com OLED SSD1306
3  / Descrição: Este código lê a temperatura e umidade do sensor AHT10 e exibe os dados em um display OLED SSD1306.
4  / Bibliotecas: aht10, ssd1306
5  / Autor: José Adriano
6  / Data de Criação: 26/07/2025
7  /-----
8  */
9  #include <stdio.h>
10 #include "pico/stdlib.h"
11 #include "hardware/i2c.h"
12 #include "aht10.h"
13 #include "ssd1306.h"
14
15 // I2C usado: I2C0 com SDA=GPIO4, SCL=GPIO5
16 #define I2C_PORT0 i2c0
17 #define I2C_SDA0 4
18 #define I2C_SCL0 5
19
20 #define I2C_PORT1 i2c1
21 #define I2C_SDA1 14
22 #define I2C_SCL1 15
23
24 // Prototipos das funções I2C
25 int i2c_write(uint8_t addr, const uint8_t *data, uint16_t Len);
26 int i2c_read(uint8_t addr, uint8_t *data, uint16_t Len);
27 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.

```

29 int main() {
30     stdio_init_all();
31
32     // Inicializa I2C sensor
33     i2c_init(I2C_PORT0, 100 * 1000); // 100 kHz
34     gpio_set_function(I2C_SDA0, GPIO_FUNC_I2C);
35     gpio_set_function(I2C_SCL0, GPIO_FUNC_I2C);
36     gpio_pull_up(I2C_SDA0);
37     gpio_pull_up(I2C_SCL0);
38
39     // Inicializa I2C OLED
40     i2c_init(I2C_PORT1, 400000);
41     gpio_set_function(I2C_SDA1, GPIO_FUNC_I2C);
42     gpio_set_function(I2C_SCL1, GPIO_FUNC_I2C);
43     gpio_pull_up(I2C_SDA1);
44     gpio_pull_up(I2C_SCL1);
45
46     ssd1306_init(I2C_PORT1);
47     ssd1306_clear();
48     ssd1306_draw_string(32, 0, "Embarcatech");
49     ssd1306_draw_string(20, 10, "Inicializando...");
50     ssd1306_show();
51
52     // Define estrutura do sensor
53     AHT10_Handle aht10 = {
54         .iface = {
55             .i2c_write = i2c_write,
56             .i2c_read = i2c_read,
57             .delay_ms = delay_ms
58         }
59     };
60
61     printf("Inicializando AHT10...\n");
62     if (!AHT10_Init(&aht10)) {
63         printf("Falha na inicialização do sensor!\n");
64         ssd1306_clear();
65         ssd1306_draw_string(32, 0, "Embarcatech");
66         ssd1306_draw_string(23, 30, "Falha no AHT10");
67         ssd1306_show();
68         while (1) sleep_ms(1000);
69     }
70 }

```

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.

```

71 while (1) {
72     float temp, hum;
73     if (AHT10_ReadTemperatureHumidity(&aht10, &temp, &hum)) {
74         printf("Temperatura: %.2f °C | Umidade: %.2f %%\n", temp, hum);
75     } else {
76         printf("Falha na leitura dos dados!\n");
77     }
78     while(hum > 70){
79         ssd1306_clear();
80         ssd1306_draw_string(32, 0, "Embarcatech");
81         ssd1306_draw_string(30, 10, "AHT10 Sensor");
82         ssd1306_draw_string(0, 20, "Umidade");
83         char hum_str[16];
84         snprintf(hum_str, sizeof(hum_str), "%.2f %%", hum);
85         ssd1306_draw_string(85, 20, hum_str);
86         ssd1306_draw_string(22,40, "Acima de 70 %");
87         ssd1306_draw_string(40,50, "ATENCAO");
88         ssd1306_show();
89         sleep_ms(500);
90         ssd1306_draw_string(40,50, " ");
91         ssd1306_show();
92         sleep_ms(500);
93     }
94     AHT10_ReadTemperatureHumidity(&aht10, &temp, &hum);
95
96     while(temp < 20){
97         ssd1306_clear();
98         ssd1306_draw_string(32, 0, "Embarcatech");
99         ssd1306_draw_string(30, 10, "AHT10 Sensor");
100        ssd1306_draw_string(0, 20, "Temperatura");
101        char temp_str[16];
102        snprintf(temp_str, sizeof(temp_str), "%.2f °C", temp);
103        ssd1306_draw_string(85, 20, temp_str);
104        ssd1306_draw_string(20,40, "Abaixo de 20 °C");
105        ssd1306_draw_string(40,50, "ATENCAO");
106        ssd1306_show();
107        sleep_ms(500);
108        ssd1306_draw_string(40,50, " ");
109        ssd1306_show();
110        sleep_ms(500);
111    }
112    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.

```

112 ssd1306_clear();
113 ssd1306_draw_string(32, 0, "Embarcatech");
114 ssd1306_draw_string(30, 10, "AHT10 Sensor");
115 ssd1306_draw_string(0, 30, "Temperatura");
116 char temp_str[16];
117 snprintf(temp_str, sizeof(temp_str), "%.2f °C", temp);
118 ssd1306_draw_string(85, 30, temp_str);
119 ssd1306_draw_string(0, 50, "Umidade");
120 char hum_str[16];
121 snprintf(hum_str, sizeof(hum_str), "%.2f %%", hum);
122 ssd1306_draw_string(85, 50, hum_str);
123 ssd1306_show();
124
125 sleep_ms(1000);
126 }
127 }

```

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.

```

129 // Função para escrita I2C
130 int i2c_write(uint8_t addr, const uint8_t *data, uint16_t len) {
131     int result = i2c_write_blocking(I2C_PORT0, addr, data, len, false);
132     return result < 0 ? -1 : 0;
133 }
134
135 // Função para leitura I2C
136 int i2c_read(uint8_t addr, uint8_t *data, uint16_t len) {
137     int result = i2c_read_blocking(I2C_PORT0, addr, data, len, false);
138     return result < 0 ? -1 : 0;
139 }
140
141 // Função para delay
142 void delay_ms(uint32_t ms) {
143     sleep_ms(ms);
144 }

```

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, programar 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.

```
13 #include <stdio.h>
14 #include "pico/stdlib.h"
15 #include "hardware/i2c.h"
16 #include "servo_sim.h"
17 #include "flash_storage.h"
18 #include "ssd1306.h"
19 #include "mpu6050_i2c.h"
20
```

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.

```
21 // ==== Pinos ====
22 #define SERVO_PIN 2 // GPIO do servo contínuo (simulado)
23 #define BTN_CALIB 5 // Botão de calibração (ativo em nível baixo)
24 #define BUZZER1_PIN 10 // Buzzer 1
25 #define BUZZER2_PIN 27 // Buzzer 2
26
27 // I2C OLED (i2c1)
28 #define I2C_PORT_OLED i2c1
29 #define SDA_OLED 14
30 #define SCL_OLED 15
31
32 // ==== Configurações ====
33 #define ALERT_THRESHOLD 90.0f // Ângulo limite para alerta
```

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

```

35 int main() {
36     stdio_init_all();
37
38     // === Botão de calibração ===
39     gpio_init(BTN_CALIB);
40     gpio_set_dir(BTN_CALIB, GPIO_IN);
41     gpio_pull_up(BTN_CALIB);
42
43     // === Buzzers ===
44     gpio_init(BUZZER1_PIN);
45     gpio_set_dir(BUZZER1_PIN, GPIO_OUT);
46     gpio_put(BUZZER1_PIN, 0);
47
48     gpio_init(BUZZER2_PIN);
49     gpio_set_dir(BUZZER2_PIN, GPIO_OUT);
50     gpio_put(BUZZER2_PIN, 0);
51
52     // === Inicializa MPU6050 (I2C0) ===
53     mpu6050_setup_i2c();
54     mpu6050_reset();
55     sleep_ms(200);
56
57     if (!mpu6050_test()) {
58         printf("MPU6050 nao encontrado!\n");
59         while (1) sleep_ms(1000);
60     }
61
62     // === Inicializa OLED (I2C1) ===
63     i2c_init(I2C_PORT_OLED, 400000);
64     gpio_set_function(SDA_OLED, GPIO_FUNC_I2C);
65     gpio_set_function(SCL_OLED, GPIO_FUNC_I2C);
66     gpio_pull_up(SDA_OLED);
67     gpio_pull_up(SCL_OLED);
68     ssd1306_init(I2C_PORT_OLED);
69
70     // Tela inicial
71     ssd1306_clear();
72     ssd1306_draw_string(20, 0, "Servo MPU6050");
73     ssd1306_draw_string(8, 12, "Inicializando...");
74     ssd1306_show();
75     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.

```

77 // === Servo: carrega calibração ===
78 uint32_t rotation_time_ms = 1000;
79 bool have_calib = flash_storage_read(&rotation_time_ms);
80
81 servo_sim_t servo;
82 servo_sim_init(&servo, SERVO_PIN, (float)rotation_time_ms);
83
84 if (!gpio_get(BTN_CALIB)) {
85     ssd1306_clear();
86     ssd1306_draw_string(20, 24, "Calibrando...");
87     ssd1306_show();
88
89     servo_sim_calibrate(&servo);
90     rotation_time_ms = (uint32_t)(180.0f / servo.deg_per_ms);
91     flash_storage_write(rotation_time_ms);
92
93     ssd1306_clear();
94     ssd1306_draw_string(8, 24, "Calibracao salva!");
95     ssd1306_show();
96     sleep_ms(1200);
97 } else if (have_calib) {
98     char msg[24];
99     snprintf(msg, sizeof(msg), "Calib: %ums", rotation_time_ms);
100     ssd1306_clear();
101     ssd1306_draw_string(10, 24, msg);
102     ssd1306_show();
103     sleep_ms(800);
104 }
105
106 float current_angle = 90.0f; // posição inicial
107 bool alert_active = false;
108 int frame = 0;
109 bool use_buzzer1 = true;
110
111 char alert_msg[32];
112 snprintf(alert_msg, sizeof(alert_msg), "!!! ALERTA >%.0f !!!", ALERT_THRESHOLD);
113

```

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

```
114 while (true) {
115     int16_t accel[3], gyro[3], temp_raw;
116     mpu6050_read_raw(accel, gyro, &temp_raw);
117
118     // Escolha de ângulo: simples → aceleração no eixo X
119     float ax = accel[0] / ACCEL_SENS_2G;
120     float target_angle = (ax < -0.5f) ? 0.0f : (ax < 0.5f ? 90.0f : 180.0f);
121
122     // Movimento suave
123     if (current_angle < target_angle) current_angle += 2.0f;
124     else if (current_angle > target_angle) current_angle -= 2.0f;
125
126     // Atualiza flag do alerta
127     alert_active = (current_angle > ALERT_THRESHOLD);
128
129     // Debug serial
130     printf("AX=%.2fg AY=%.2fg AZ=%.2fg | GX=%d GY=%d GZ=%d | Alvo=%.0f deg | Atual=%.0f deg\n",
131         accel[0]/ACCEL_SENS_2G, accel[1]/ACCEL_SENS_2G, accel[2]/ACCEL_SENS_2G,
132         gyro[0], gyro[1], gyro[2], target_angle, current_angle);
133
134     // Servo simulado
135     servo_sim_set_angle(&servo, current_angle);
136
137     // === Display ===
138     ssd1306_clear();
139     ssd1306_draw_string(20, 0, "Servo MPU6050");
140
141     char line1[24], line2[24];
142     snprintf(line1, sizeof(line1), "AX: %.2fg", accel[0]/ACCEL_SENS_2G);
143     snprintf(line2, sizeof(line2), "Ang: %.0f/%.0f", current_angle, target_angle);
144
145     ssd1306_draw_string(6, 20, line1);
146     ssd1306_draw_string(6, 36, line2);
```

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.

```
147
148     // Alerta piscante + beep alternado
149     if (alert_active && (frame % 2 == 0)) {
150         ssd1306_draw_string(6, 52, alert_msg);
151
152         if (use_buzzer1) {
153             gpio_put(BUZZER1_PIN, 1);
154             sleep_ms(100);
155             gpio_put(BUZZER1_PIN, 0);
156         } else {
157             gpio_put(BUZZER2_PIN, 1);
158             sleep_ms(100);
159             gpio_put(BUZZER2_PIN, 0);
160         }
161         use_buzzer1 = !use_buzzer1; // alterna buzzer
162     }
163
164     ssd1306_show();
165     frame++;
166
167     sleep_ms(200);
168 }
169 }
170
```

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.