

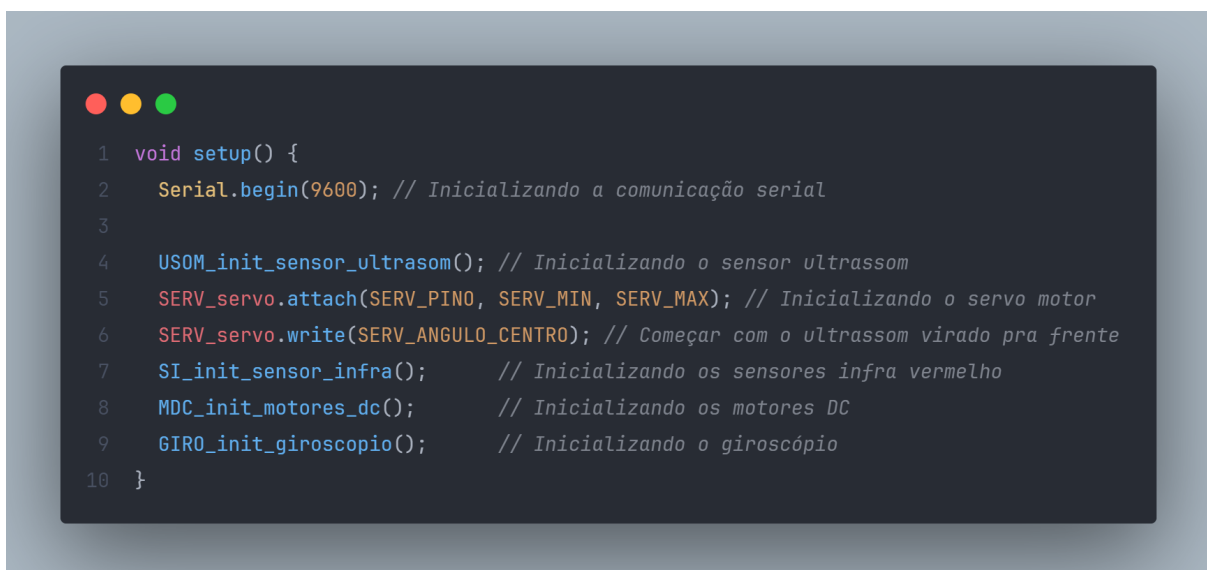
SISTEMAS EMBARCADOS I - ELE-PROP-00020

Projeto Final

Grupo: Caio Alves Fiorotti, Matheus Meier Schreiber, Vinicius Cole de Amorim

O projeto final da disciplina de Sistemas Embarcados I do semestre de 2023/2 foi desenvolver um código em C++ para o embarcado ESP32 no robô que segue linha com o objetivo dele sair de um labirinto. Tudo foi feito em apenas um arquivo, visto que não havia necessidade de separar em vários.

Depois de importar as bibliotecas do giroscópio e do servo motor, declarar todas constantes e cabeçalhos das funções, buscou-se inicializar a serial na função *setup* (apenas para debug) e também inicializar todos os periféricos que serão necessários (sensor ultrassom, servo motor, sensores infravermelho, motores DC e giroscópio), como mostra a Figura 1 abaixo.



```
1 void setup() {
2     Serial.begin(9600); // Inicializando a comunicação serial
3
4     USOM_init_sensor_ultrasom(); // Inicializando o sensor ultrassom
5     SERV_servo.attach(SERV_PIN0, SERV_MIN, SERV_MAX); // Inicializando o servo motor
6     SERV_servo.write(SERV_ANGULO_CENTRO); // Começar com o ultrassom virado pra frente
7     SI_init_sensor_infra(); // Inicializando os sensores infra vermelho
8     MDC_init_motores_dc(); // Inicializando os motores DC
9     GIRO_init_giroscopio(); // Inicializando o giroscópio
10 }
```

Figura 1 - Função *setup*

Dentre essas funções de inicialização, vale a pena dar uma olhada na que inicializa os motores. Depois de configurar os pinos de entrada dos motores e seus respectivos pinos de enable como *output*, foi preciso sincronizar os pinos dos motores com seus respectivos canais de *PWM* com a função *ledcAttachPin*,

depois configurar o próprio *PWM* com a função *ledcSetup*. Essa inicialização pode ser vista na Figura 2 abaixo.

```
1 void MDC_init_motores_dc() {
2     // Configurando pinos de enable dos motores
3     pinMode(MDC_PINO_E_M1, OUTPUT);
4     pinMode(MDC_PINO_E_M2, OUTPUT);
5
6     // Configurando pinos do PWM dos motores
7     pinMode(MDC_PINO_M1, OUTPUT);
8     pinMode(MDC_PINO_M2, OUTPUT);
9
10    // Sincronizando um pino a um canal
11    ledcAttachPin(MDC_PINO_M1, MDC_PWM1_CH);
12    ledcSetup(MDC_PWM1_CH, MDC_PWM_FREQ, MDC_PWM_RES);
13
14    // Sincronizando um pino a um canal
15    ledcAttachPin(MDC_PINO_M2, MDC_PWM2_CH);
16    ledcSetup(MDC_PWM2_CH, MDC_PWM_FREQ, MDC_PWM_RES);
17
18    ledcWrite(MDC_PWM1_CH, 128);
19    ledcWrite(MDC_PWM2_CH, 128);
20
21    MDC_liga_motores();
22 }
23
```

Figura 2 - Função de inicialização dos motores DC

A partir daí começa a lógica de controle do robô de fato, que pode ser simplificada pelo diagrama de fluxo a seguir.

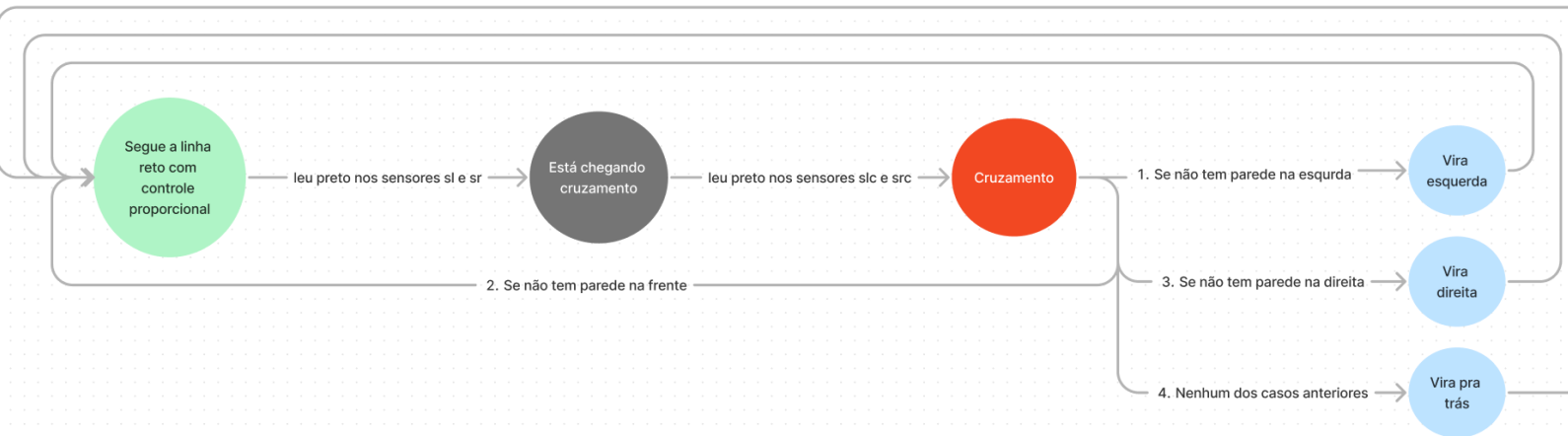


Figura 3 - Fluxograma de decisões ações do robô

A lógica se baseia em utilizar os 3 sensores da frente do robô para fazê-lo seguir a linha por meio de um controle proporcional, e, ao encontrar um cruzamento, escolher uma direção para virar. A escolha da direção do robô será explicada no decorrer do relatório

Os sensores utilizados no algoritmo de segue-linha, são os dois sensores dianteiros e o sensor central, denotados na Figura 4 abaixo.

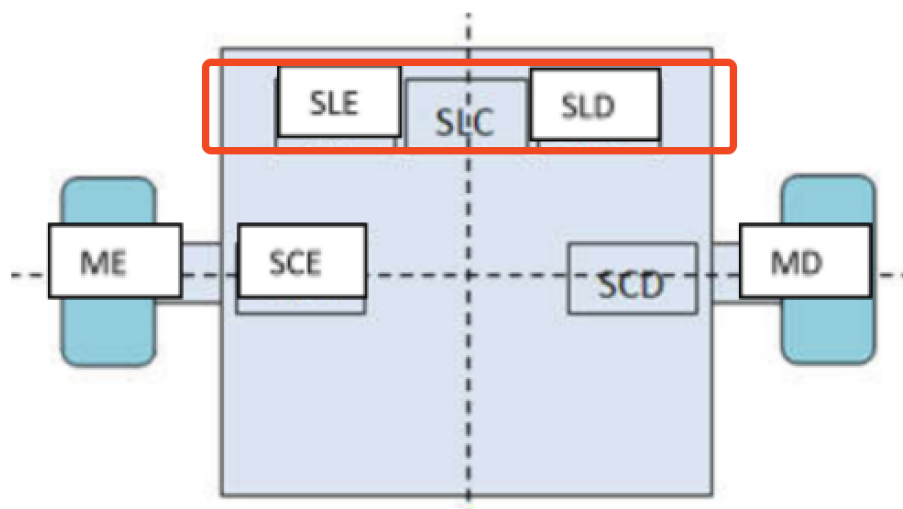
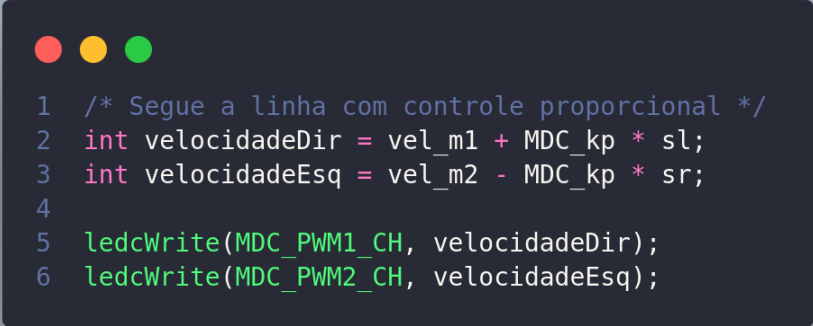


Figura 4. Configuração dos sensores no robô, destacando os sensores utilizado no algoritmo de segue-linha

Para o controlador proporcional, utilizou-se da lógica apresentada na Figura 5 abaixo, que se baseia na seguinte premissa: a velocidade que será transmitida para cada um dos motores (esquerda ou direita) será proporcional ao valor lido no sensor do lado correspondente (esquerda ou direita), considerando uma constante de proporcionalidade K_p , que é usada para controlar a velocidade máxima do robô.



```
1  /* Segue a linha com controle proporcional */
2  int velocidadeDir = vel_m1 + MDC_kp * sl;
3  int velocidadeEsq = vel_m2 - MDC_kp * sr;
4
5  ledcWrite(MDC_PWM1_CH, velocidadeDir);
6  ledcWrite(MDC_PWM2_CH, velocidadeEsq);
```

Figura 5 - Controle proporcional para o algoritmo de segue-linha

Como a resolução do PWM foi configurada para 8 bits, a velocidade dos motores variam de 0 até 255, sendo 0 a velocidade máxima em um dos sentidos, 255 a velocidade máxima no sentido oposto e 128 o estado onde a roda fica parada. Vale ressaltar que o motor da esquerda foi montado invertido no robô, e, por isso, precisamos subtrair $K_p * sr$, de forma que garantimos que ambos os motores estejam funcionando no mesmo sentido.

Em seguida, elaborou-se um algoritmo para detectar quando o robô está se aproximando de um cruzamento, ou seja, quando encontrou a linha ainda nos sensores dianteiros. Se for o caso, o robô reduz a velocidade, para evitar passar o cruzamento direto em alta velocidade, o que pode fazer com que o cruzamento não seja reconhecido.

Agora, lendo os sensores laterais denotados na Figura 6 abaixo, caso se encontre em um cruzamento (duas fitas cruzadas ortogonalmente no chão), o robô deve sempre buscar virar à esquerda, caso não tenha obstáculos, senão, ele utiliza do servo motor para redirecionar o sensor ultrassom montado na parte dianteira, para buscar o caminho livre para seguir.

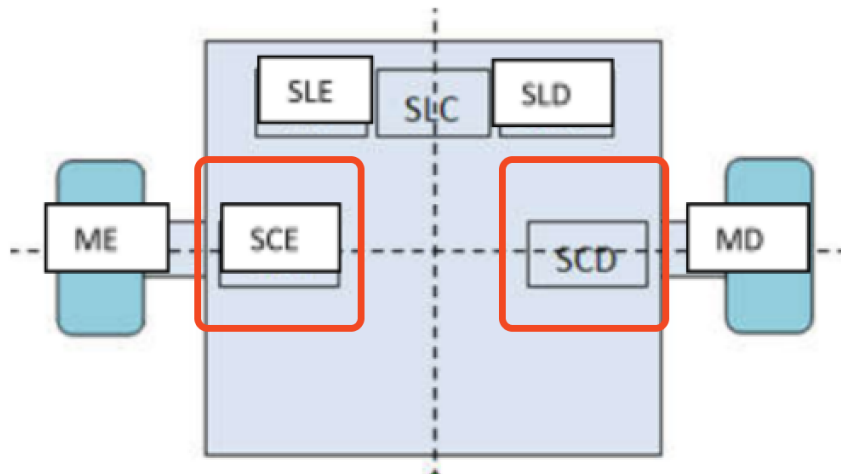


Figura 6 - Sensores de detecção de cruzamento

Virar sempre para uma mesma direção é um algoritmo muito importante para a resolução de labirintos. Repare que ao virar sempre à esquerda, garantimos que o robô sempre chegue ao final, mesmo que precise fazer o caminho mais longo para isso.

Uma versão simplificada do código referente a esse raciocínio, considerando desde a desaceleração até a decisão de direção a seguir, está na Figura 7 abaixo.

```

1  /* Verificação se precisa virar (encontrou cruzamento) */
2  if (!MDC_esta_virando) {
3
4      // Caso detecte cruzamento, desliga motores e le ultrassom
5
6      // decide qual direção virar
7  }
8
9
10 /* Caso não precise virar, apenas siga em linha reta */
11 if (!MDC_esta_virando) {
12
13     // verifica se está fora da linha, se estiver desliga os motores
14
15     // verifica se está chegando no cruzamento, se estiver, diminui velocidade
16
17     // se não estiver chegando em cruzamento, segue a linha com controle proporcional
18
19 } else {
20     // Caso esteja virando, então continue virando até terminar
21 }
22
23
24

```

Figura 7 - Simplificação do algoritmo de decisão de cruzamento e segue-linha

Computado esse raciocínio principal, o robô já consegue concluir o labirinto corretamente. Para complementar, segue abaixo algumas funções utilizadas ao longo do código para realizar as tarefas desejadas em determinadas situações.

Primeiramente, tem-se a função `MDC_vira` de virar, que recebe a direção em que se deseja virar, e então aciona os motores para rotacionar o robô. Tal procedimento é descrito na Figura 8 a seguir.

```
1  /* Função que vira o robô para esquerda, direita ou 180 graus */
2  void MDC_vira(char direcao){
3      // 0 - 127 = trás
4      // 128 - 255 = frente
5      mpu.update();
6      float pos_inicial = mpu.getAngleZ();
7      float pos_atual = pos_inicial;
8
9      switch(direcao) {
10         case 'L':
11             // Vira para esquerda
12             ledcWrite(MDC_PWM1_CH, 170);
13             ledcWrite(MDC_PWM2_CH, 170);
14
15             /* Vira um angulo de 85 graus */
16             while (pos_atual < pos_inicial + GIRO_ANGULO_VIRAR) {
17                 mpu.update();
18                 pos_atual = mpu.getAngleZ();
19             }
20             break;
21
22         case 'R':
23             // Virar para direita
24             ledcWrite(MDC_PWM1_CH, 84);
25             ledcWrite(MDC_PWM2_CH, 84);
26
27             /* Vira um angulo de 85 graus */
28             while (pos_atual > pos_inicial - GIRO_ANGULO_VIRAR) {
29                 mpu.update();
30                 pos_atual = mpu.getAngleZ();
31             }
32             break;
33
34         case 'B':
35             // Vira 180 graus
36             ledcWrite(MDC_PWM1_CH, 170);
37             ledcWrite(MDC_PWM2_CH, 170);
38
39             /* Vira um angulo de 85*2 graus */
40             while (pos_atual < pos_inicial + GIRO_ANGULO_VIRAR*2) {
41                 mpu.update();
42                 pos_atual = mpu.getAngleZ();
43             }
44             break;
45
46         case 'S':
47             // Faz nada
48             break;
49     }
50 }
```

Figura 8 - Função de rotacionar o robô

Em seguida, tem-se a função *USOM_media_das_distancias* de leitura do sensor ultrassom, que faz uma média das últimas 5 distâncias lidas pelo sensor para verificar se há algum obstáculo em sua frente. Tal função pode ser observada na Figura 9 abaixo.

```
1  /* Função que retorna a distancia vista no sensor ultrassom */
2  int USOM_le_distancia(){
3
4      digitalWrite(USOM_TRIGGER_PIN, LOW); // Começa em zero o pulso
5      delayMicroseconds(2);
6
7      digitalWrite(USOM_TRIGGER_PIN, HIGH); // Manda pulso
8      delayMicroseconds(10);
9
10     digitalWrite(USOM_TRIGGER_PIN, LOW); // Volta pra zero o pulso
11
12     // Recebe o tempo que demorou para o pulso percorrer tudo (ida e volta) em ms
13     float tempo = pulseIn(USOM_ECHO_PIN, HIGH);
14
15     /* velocidade do som: 343m/s = 0.0343 cm/us */
16     float distancia = 0.01723 * tempo;
17
18     return distancia;
19 }
20
21 /* Função que retorna a média dos últimos n valores lidos
22 no ultrassom a partir do momento em que foi chamada */
23 int USOM_media_das_distancias() {
24     int n = 5, soma = 0, i = 0;
25     for(i = 0; i < n; i++) soma += USOM_le_distancia();
26     return soma/n;
27 }
```

Figura 9 - Função de leitura do sensor ultrassom

Ainda, implementou-se a função *SERV_decide_para_onde_virar*, para rotacionar o servo motor para uma dada direção específica, sendo essa função ilustrada na Figura 10 a seguir. Essa função retorna a direção em que não há obstáculos detectados pelo ultrassom.

```

1  /* Função que retorna a direção em que não há parede de acordo com o sensor ultrassom */
2  int SERV_decide_para_onde_virar(){
3      SERV_servo.write(SERV_ANGULO_ESQUERDA);          // Aponta pra esquerda do carrinho
4      delay(500);
5      USOM_distancia_parede = USOM_media_das_distancias(); // Lê distancia do ultrassom
6      if (USOM_distancia_parede > USOM_THRESHOLD_PAREDE) return ESQUERDA;
7
8      SERV_servo.write(SERV_ANGULO_CENTRO);            // Aponta pro centro do carrinho
9      delay(500);
10     USOM_distancia_parede = USOM_media_das_distancias(); // Lê distancia do ultrassom
11     if (USOM_distancia_parede > USOM_THRESHOLD_PAREDE) return FRENTE;
12
13     SERV_servo.write(SERV_ANGULO_DIREITA);           // Aponta pra direita do carrinho
14     delay(500);
15     USOM_distancia_parede = USOM_media_das_distancias(); // Lê distancia do ultrassom
16     if (USOM_distancia_parede > USOM_THRESHOLD_PAREDE) return DIREITA;
17
18     return TRAS;
19 }

```

Figura 10 - Função de acionamento do servo motor

Repare na priorização da movimentação do robô. Caso a esquerda esteja livre, nem é necessário checar as outras direções, o robô vira à esquerda e segue o caminho.

Os testes feitos durante o processo de desenvolvimento foram bem satisfatórios, rapidamente atingindo o ponto onde o robô consegue fazer cruzamentos. Boa parte do tempo foi gasto com alguns pequenos ajustes, principalmente no ângulo de rotação do robô e na parte do controle proporcional, com o objetivo de evitar que o robô chegasse em um cruzamento fora da posição correta.