CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS - UNIDADE DE DIVINÓPOLIS

DEPARTAMENTO DE ENGENHARIA MECATRÔNICA

Víctor Bernardes de Morais

Relatório Final do Projeto e Desenvolvimento de um Scanner de Baixo Custo para Aferição de Distâncias em Aplicações de Robótica Móvel

Orientador: Dr. Adriano Nogueira Drumond Lopes

Coorientador: Dr. Lucas Silva de Oliveira

Divinópolis

Dezembro, 2023

CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS - UNIDADE DE DIVINÓPOLIS

DEPARTAMENTO DE ENGENHARIA MECATRÔNICA

Víctor Bernardes de Morais

Relatório Final do Projeto e Desenvolvimento de um Scanner de Baixo Custo para Aferição de Distâncias em Aplicações de Robótica Móvel

Relatório final apresentado como requisito para conclusão do Projeto de Iniciação Científica 'Projeto e Desenvolvimento de um Scanner de Baixo Custo para Aferição de Robótica Móvel' do Centro Federal de Educação Tecnológica de Minas Gerais – Campus Divinópolis.

Divinópolis

Dezembro, 2023

SUMÁRIO

1. INTRODUÇÃO

Em problemas de robótica móvel, o uso de um sistema capaz de fazer o sensoriamento do espaço de trabalho do robô é essencial para o seu bom funcionamento. Dessa forma, vários são os fenômenos físicos que podem ser utilizados para proporcionar um conhecimento sobre o local em que o dispostivo está localizado.

Dentre esses fatores, a luz é um dos mais interessantes. Isso deve-se a sua grande velocidade, o que permite uma medição extremamente rápida. Assim, são vários os tipos de equipamentos e sensores que usam da luz para aferir distâncias. Entre eles, o LiDAR (*Light Detection and Ranging*) apresenta um funcionamento bastante interessante.

Para seu funcionamento, dispositivos desse tipo utilizam um motor que faz um sensor que emissor-receptor de luz girar 360º, o que produz um mapa de pontos, como visto na Figura 1. Um exemplo de sensor LiDAR é RPLIDAR A1 [1], da Slamtec. Apesar dessa funcionalidade, o uso de LiDAR é, muitas vezes, inviável devido ao seu elevado preço, como pode ser visto na Figura 2.

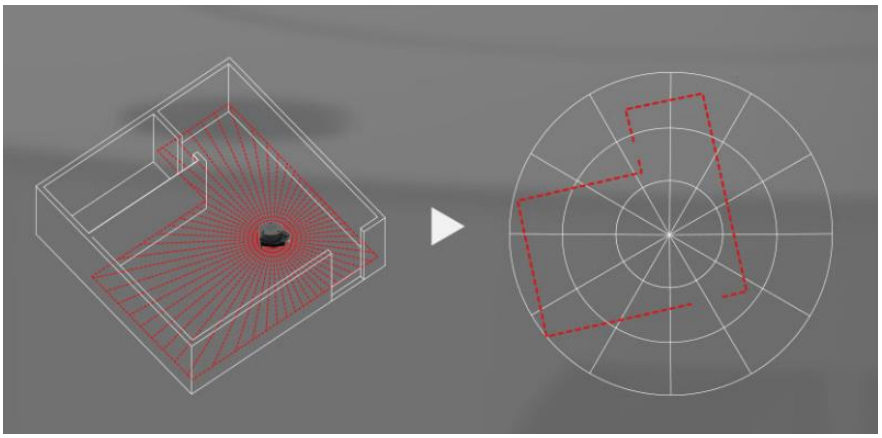

Figura 1 – Mapeamento de pontos

Fonte: Slamtec

Figura 2 – Preço do RPLIDAR A1

Fonte: Perfil oficial da Slamtec no site de vendas Alibaba

Dessa forma, surgiu a ideia de reproduzir o funcionamento do LiDAR, mas almejando um baixo custo. Isso é capaz de permitir a utilização constante desse tipo de equipamento para aferição de distâncias em robótica móvel.

Para isso, foi necessário fazer um estudo de caso em busca dos componentes mais adequados para o projeto em suas partes eletrônicas e mecânicas e da programação mais simples e funcional para o uso facilitado desse dispositivo. Assim, ao longo desse relatório, o desenvolvimento do projeto será apresentado, assim como suas partes constituintes e o modo como conectam-se.

2. ELETRÔNICA

Em um primeiro momento, foi necessário definir os componentes que seriam utilizados nesse projeto. Em especial, foi de suma importância estudar sobre sensores ópticos de distância. Assim, uma série de sensores foram selecionados para que fossem analisados (Figura 3).

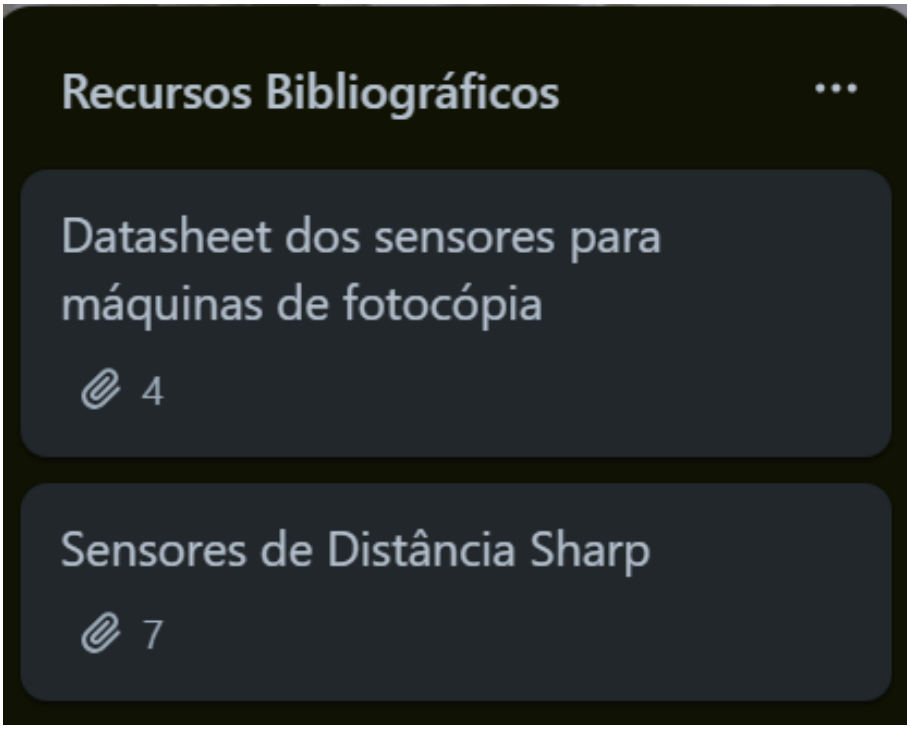

Figura 3 – Sensores a serem pesquisados ao longo dos estudos

Fonte: Trello do projeto

Logo após isso, os outros componentes eletrônicos que foram utilizados ao longo da pesquisa foram determinados. Entre eles estão microcontroladores e microservos, além de motores e pontes H que já haviam sido pré-definidos.

2.1.1. Sensores Sharp

Inicialmente, sensores de máquinas de fotocópia foram pesquisados para descobrir se eles poderiam ser utilizados para aferir distâncias. Assim, foi possível determinar que eles não atenderiam às expectativas do projeto, uma vez que sua faixa de operação é baixa demais. Além disso, sensores Sharp foram analisados para conferir suas diversas características e quais seriam as mais úteis para aferir distâncias. Para facilitar na compreensão dos resultados das pesquisas, tabelas que reúne todas essas informações acerca dos sensores foi criada (Tabela 1 e Tabela 2).

Sensores Sharp/ Características	GP2Y0A51SK0F [2]	GP2Y0D02YK0F [3]	GP2Y0D21YK0F [4]	GP2Y0E03 [5]
Faixa de operação (cm)	2 - 15	80	24	4 - 50
Tipo de Saída	Analógica	Digital	Digital	Digital/Analógica (I²C)
Eletrônica	PSD; IR – LED; variação de tensão de saída de acordo com a distância medida	PSD; IR – LED; variação de tensão de saída de acordo com a distância medida	PSD; IR – LED; variação de tensão de saída de acordo com a distância medida	Sensor de imagem CMOS; IR – LED; variação de tensão de saída de acordo com a distância medida
Tensão de Entrada (V)	4,5 – 5,5	4,5 – 5,5	4,5 – 5,5	2,7 – 5,5 (I/O: 1,8 – 3,3)
Tensão de Saída (V)	0,25 – 0,55	(Vcc – 0,3) - 0,6	(Vcc – 0,3) - 0,6	(-0,3) – 2,8 (I²C: (-0,3) – 0,3)
Exemplo de Aplicações	Robô limpador	Robôs, touchless switch	Robôs, touchless switch	Robô limpador, robôs em geral, touchless switch

Tabela 1 – Sensores Sharp e suas características – Parte 1

Fonte: Elaborado pelo autor

Sensores Sharp/ Características	GP2A25J0000F [6]	GP2Y0A02YK0F [7]	GP2Y0A21YK0F [8]	GP2Y0A41SK0F [9]
Faixa de operação (cm)	0,1 – 0,9	20 - 150	10 - 80	4 - 30
Tipo de Saída	OPIC	Analógica	Analógica	Analógica
Eletrônica	Fotointerruptores com emissor e detector	PSD; IR – LED; variação de tensão de saída de acordo com a distância medida	PSD; IR – LED; variação de tensão de saída de acordo com a distância medida	PSD; IR – LED; variação de tensão de saída de acordo com a distância medida
Tensão de Entrada (V)	4,75 – 5,25	4,5 – 5,5	4,5 – 5,5	4,5 – 5,5
Tensão de Saída (V)	30	0,25 – 0,55	0,25 – 0,55	0,25 – 0,55
Exemplo de Aplicações	Impressoras	Robôs, touchless switch	Robôs, touchless switch	Robô limpador

Tabela 2 – Sensores Sharp e suas características – Parte 2

Fonte: Elaborado pelo autor

Desse forma, foi possível definir quais sensores seriam utilizados para aferir distâncias e mapear distâncias. Uma das ideias iniciais era de fundir dois sensores, mas devido a suas características, foi constatado que isso não era possível.

Além disso, os sensores cujo tipo de saída era digital foram descartados, uma vez que seu funcionamento era como o de sensores de proximidade, o que torna-os inadequados ao projeto. Nesse sentido, os sensores GP2Y0A02YK0F e GP2Y0A21YK0F foram escolhidos devido ao seu funcionamento, sua resposta analógica e sua faixa de operação. Para escolher a faixa de operação, foi decidido que não era necessário um alcance muito grande. No entanto, usar sensores com uma faixa de operação baixa demais não seria bom, uma vez que a aferição de distâncias ficaria limitada.

Ademais, o sensor GP2YE03 foi escolhido por sua boa faixa de operação e por usar a comunicação I²C, o que possibilita usos interessantes para a aferição de distâncias que serão abordados de forma mais aprofundada na seção específica desse sensor

2.1.2. GP2Y0A02YK0F

No estudo desse componente, foi possível perceber que, apesar que sua faixa de operação está entre 20 centímetros e 150 centímetros, a resposta em tensão varia muito em certas faixas de medição. De 0 centímetros a 20 centímetros, há um crescimento acentuado até atingir o pico no centímetro 20. Após isso, de 20 centímetros até, aproximadamente, 80 centímetros há uma descida na curva do gráfico de forma não linear que torna-se mais suave com o crescimento da distância. De 80 centímetros a 150 centímetros, o gráfico comporta-se de forma linear e decrescente. Além disso, a resposta em tensão varia de acordo com o material medido. Esse gráfico está representado na Figura 4.

Esse comportamento dificulta nas análises de distância durante o funcionamento do mecanismo, o que faz com que seja necessário uma lógica de programação que divida essas respostas de tensão em diferentes seções.

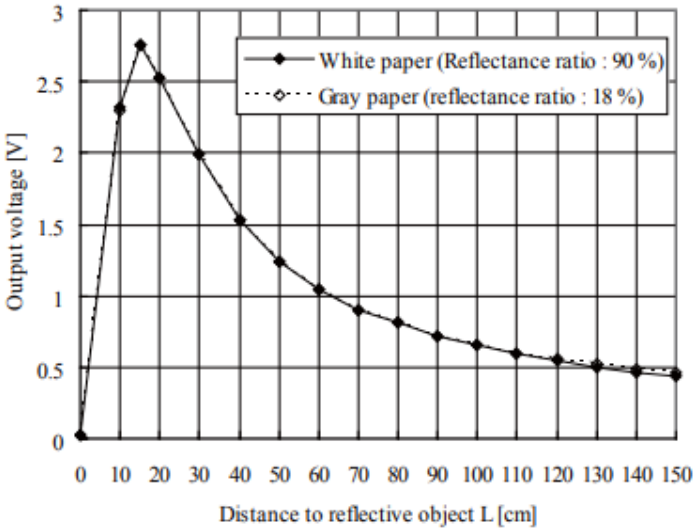

Figura 4 – Gráfico tensão x distância – GP2Y0A02YK0F

Fonte: Datasheet do GP2Y0A02YK0F [7]

2.1.3. GP2Y0A21YK0F

No estudo desse componente, foi possível perceber que, apesar que sua faixa de operação está entre 10 centímetros e 80 centímetros, a resposta em tensão varia muito em certas faixas de medição. De 0 centímetros a, aproximadamente, 8 centímetros, há um crescimento acentuado até atingir o pico no centímetro 8. Após isso, de 8 centímetros até, aproximadamente, 60 centímetros há uma descida na curva do gráfico de forma não linear que torna-se mais suave com o crescimento da distância. De 60 centímetros a 80 centímetros, o gráfico comporta-se de forma linear e decrescente. Esse gráfico está representado na Figura 5.

Assim como no sensor GP2Y0A21YK0F, esse comportamento dificulta nas análises de distância durante o funcionamento do mecanismo, o que faz com que seja necessário uma lógica de programação que divida essas respostas de tensão em diferentes seções.

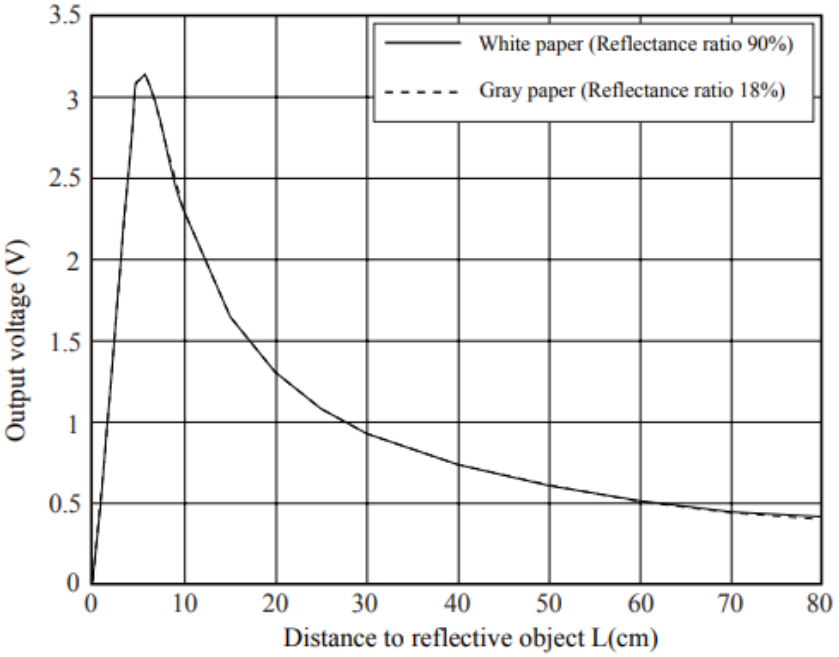

Figura 5 – Gráfico tensão x distância – GP2Y0A21YK0F

Fonte: Datasheet do GP2Y0A21YK0F [8]

2.1.4. GP2Y0E03

Diferente dos outros sensores apresentados, o GP2Y0E03 apresenta uma resposta linear durante todo o seu funcionamento, como visto na Figura 6. Além disso, essa resposta não varia de acordo com o material observado. Isso permite que um mapa de coordenadas seja criado de forma mais fácil, uma vez que a lógica de programação torna-se mais simples. Apesar de sua faixa de operação não ser tão grande quanto a dos outros sensores, ela é o suficiente para ser útil no funcionamento do robô

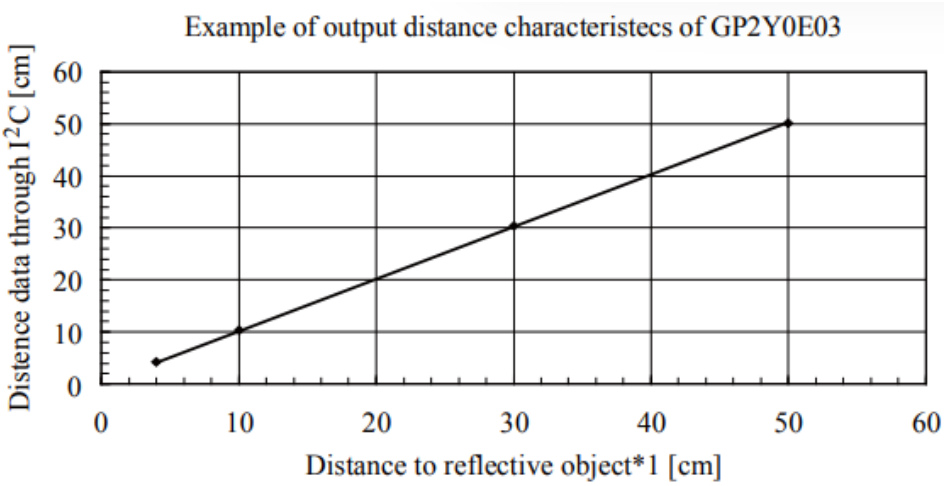

Figura 6 – Gráfico tensão x distância – GP2Y0E03

Fonte: Datasheet do GP2Y0E03 [5]

2.2. Outros Componentes

Além dos sensores, existem outros componentes fundamentais para o projeto em específico e outros que são necessários para qualquer projeto que envolva robôs móveis. Esses últimos foram pré-definidos, uma vez que eles já estavam disponíveis para uso de projetos anteriores. Eles são:

- Uma ponte H Pololu Dual MC33926 Motor Driver Shield for Arduino (Figura 7);

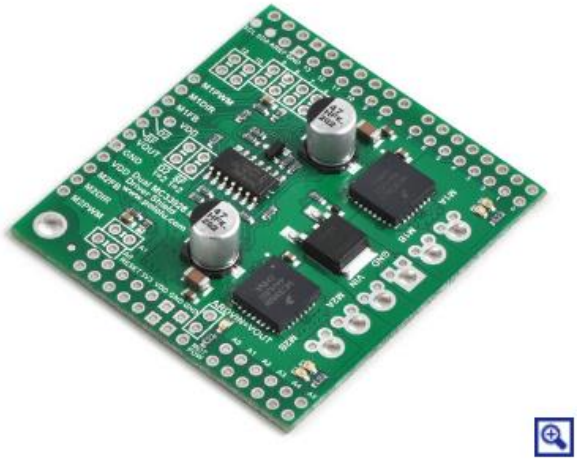

Figura 7 – Ponte H Pololu Dual MC33926 Motor Driver for Arduino [10]

Fonte: Pololu

- Dois motores de engrenagem CC, 24V e 620RPM (Figura 8);

Figura 8 – Motor de engrenagem CC, 24V e 620RPM [11]

Fonte: AliExpress

- Três sensores ultrassônicos Hc-Sr04 (Figura 9).

Figura 9 – Hc-Sr04 [12]

Fonte: Eletrogate

Os outros componentes fundamentais serão especificados nas suas seções próprias.

2.2.1. ESP32 S3 WROOM

Esse microcontrolador foi utilizado para fazer o processamento de dados e análise das distâncias devido a suas diversas funcionalidades e fácil utilização (Figura 10).

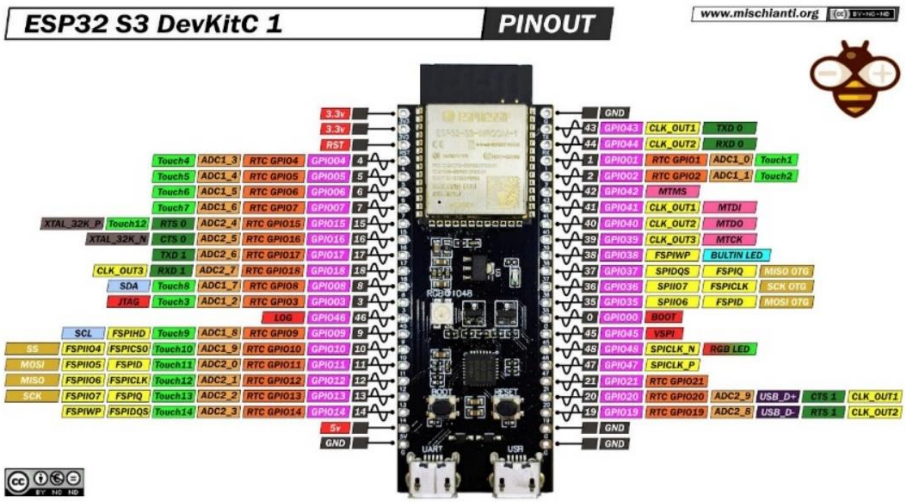

Figura 10 – Pinos do ESP32 S3 WROOM [13]

Fonte: Mischianti

2.2.2. Arduino MEGA 2560

Além dos ESP32, esse Arduino foi utilizado em certos momentos do projeto apenas em momentos que não foi possível usar o ESP32 por problemas logísticos. Isso porque esse Arduino já estava disponível para uso, enquanto foi necessário esperar a encomenda do ESP32 chegar.

Esse microcontrolador é bem parecido com o ESP32, mas não possui tantas funções quanto esse último (Figura 11).

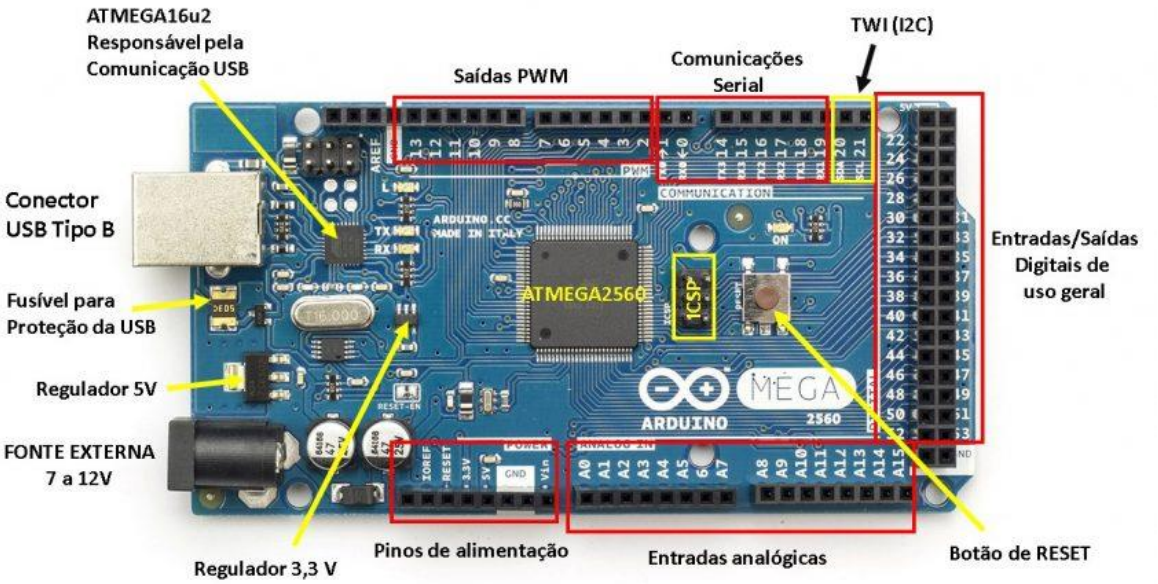

Figura 11 – Pinos do Arduino MEGA 2560 [14]

Fonte: Embarcados

2.2.3. Micro Servo 9G SG90

Um dos componentes de extrema importância é o que faz com que o mecanismo rotacione, o que simula o funcionamento do LiDAR. Para isso, esse servo (Figura 12) foi escolhido por ser simples e com força o suficiente para sustentar o suporte em que os sensores estão. Em relação a isso, o estudo na parte mecânica que virá a seguir foi essencial para determinar o servo que seria utilizado.

Figura 12 – Micro Servo 9G SG90 [15]

Fonte: Eletrogate

3. MECÂNICA

A segunda parte constituinte desse mecanismo é a mecânica. Em especial, essa parte abrange tanto o robô completo em si, mas com um enfoque especial no suporte dos sensores. Esse suporte é essencial para integrar os sensores Sharp com o servo, o que faz com que o funcionamento do LiDAR tradicional seja simulado.

Para isso, o *software* Inventor da AutoDesk foi usado para modelar o robô. Depois disso, o Lab Maker do CEFET – Divinópolis foi utilizado para construir o robô. Em relação ao suporte, ele foi feito em impressão 3D e com o uso de uma microrretifica para dar o acabamento.

3.1.1. Suporte de Sensores

Para a criação desse suporte foi necessário criar um sulco para o encaixe da haste do servo e alocações nos quais os sensores foram encaixados (Figura 13).

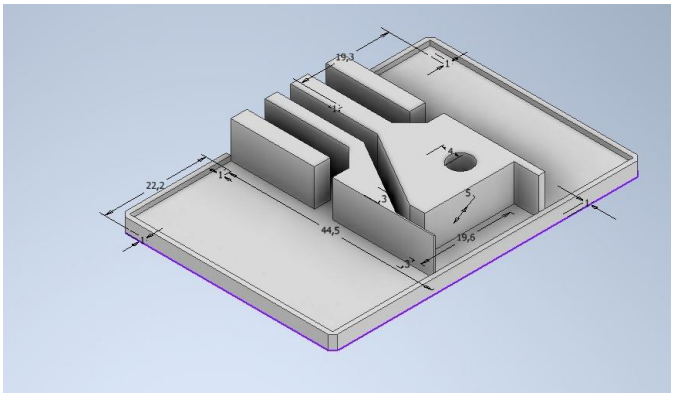

Figura 13 – Suporte de sensores

Fonte: Elaborado pelo autor

Após a elaboração do modelo, ele foi impresso em 3D no Lab Maker, como visto nas Figura 14 e 15.

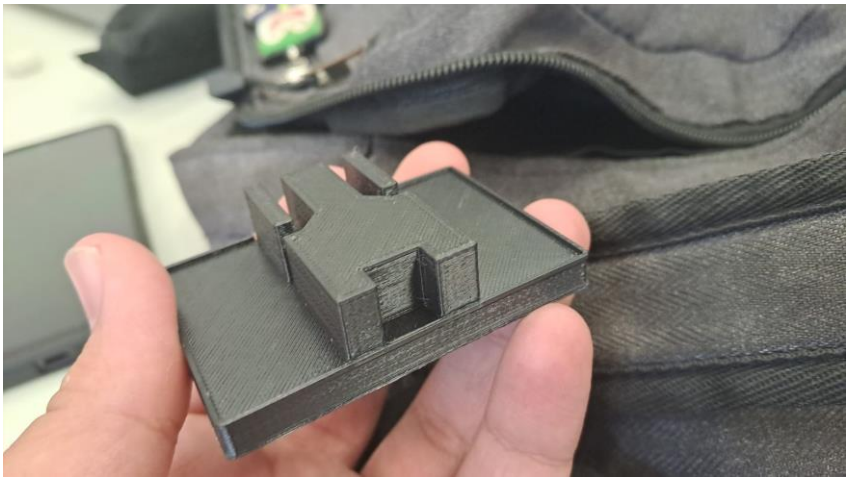

Figura 14 – Impressão 3D do suporte de sensores

Fonte: Elaborado pelo autor

Figura 15 – Encaixe do servo na impressão 3D do suporte de sensores

Fonte: Elaborado pelo autor

Dessa forma, o dispositivo foi montado por completo, com a integração do sensores e do servo, o que possibilita os testes práticos desse LiDAR, como visto na Figura 16.

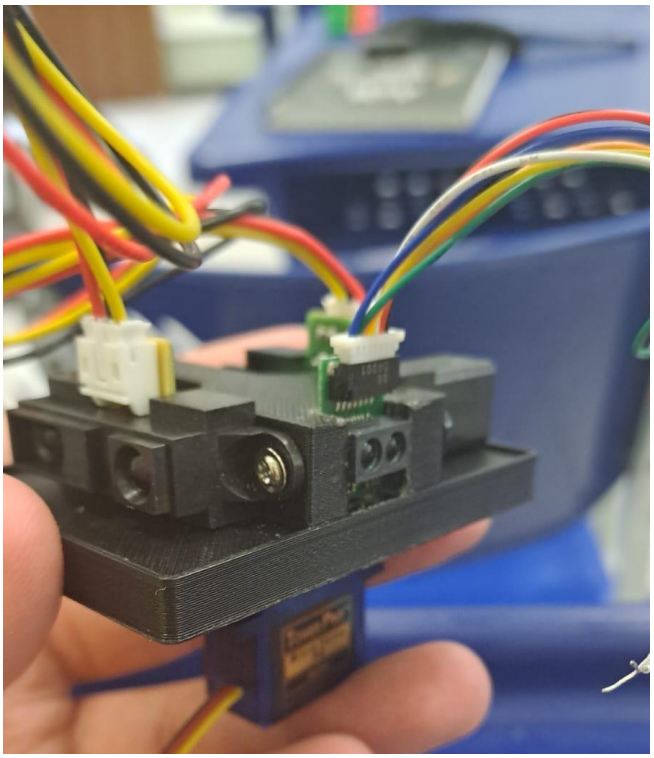

Figura 16 – Dispositivo montado

Fonte: Elaborado pelo autor

3.2. Robô

Outras partes mecânicas que compõem o robô também foram feitas no Inventor separadamente. Posteriormente, elas foram montadas para simulações. No entanto, essas outras partes não foram criadas na prática, uma vez que não havia a necessidade para os testes com o LiDAR. Entre essas partes estão (Figura 17):

- Suporte para os sensores ultrassônicos;

- Suporte para uma caster wheel;
- Suporte para os motores e rodas;
- Bases para microcontroladores, ponte H e integração de todas as partes.

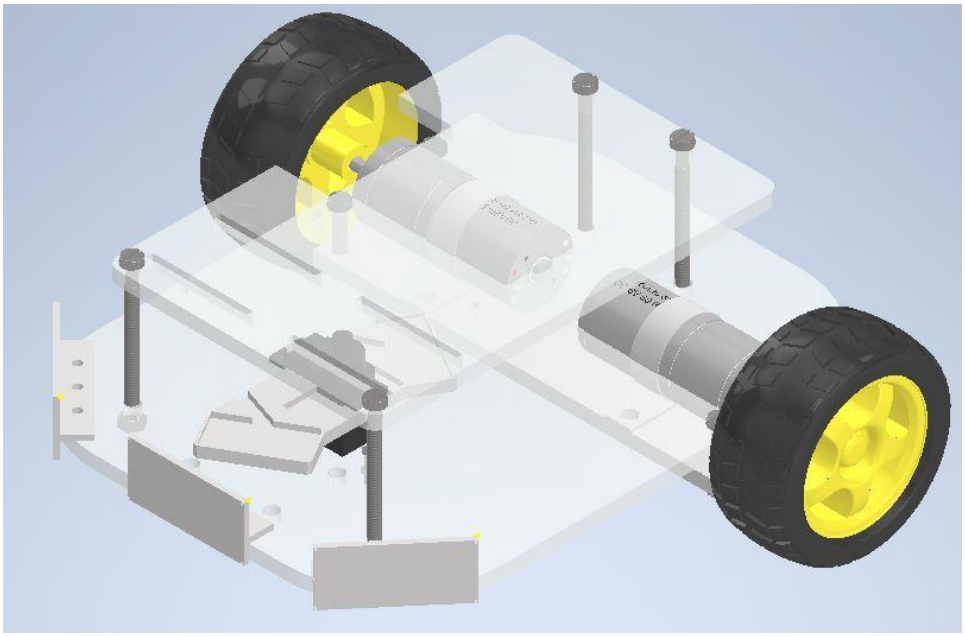

Figura 17 – Robô modelado

Fonte: Elaborado pelo autor

4. PROGRAMAÇÃO

A parte que integra e torna possível a criação de um sensor LiDAR é a programação. Para isso, foi necessário calibrar os sensores antes. Depois disso, um código foi feito que faz com que o servo rotacione e leituras sejam feitas. Assim, as análises podem ser feitas para detectar distâncias e utilizar em uma matriz que representa um mapa de coordenadas. Isso possibilita o uso em robótica móvel e uma fácil análise por usuários.

4.1. Calibração de Sensores

Para calibrar os sensores, uma fita métrica graduada em centímetros foi utilizada para testar os sensores. Assim, objetos brancos, como uma borracha, foram posicionados de centímetro a centímetro e as resposta medidas pelos sensores foram anotadas em tabelas.

Para os sensores GP2Y0A02YK0F e GP2Y0A21YK0F um mesmo código simples foi utilizado para obter as respostas em tensão. Esse código foi feito na IDE do Arduino, como visto no Código 1.

```
float tensao;
float dado;
void setup() {
  pinMode (A8, INPUT);
  Serial.begin (9600);
}
void loop() {
  dado = analogRead(A8);
  tensao = dado*5/1023;
```

```
  Serial.print ("Dado: ");
  Serial.println (dado);
  Serial.print ("Tensão: ");
  Serial.println (tensao);
  delay (2000);
}
```

Código 1 – Calibração de sensores Sharp analógicos

Fonte: Elaborado pelo autor

Assim, os resultados foram registrados na Tabela 3 e na Tabela 4.

GP2Y0A02YK0F		
CM	DADO	TENSÃO (V)
1	188	0,918866
2	220	1,075269
3	245	1,197458
4	269	1,314761
5	284	1,388074
6	316	1,544477
7	344	1,681329
8	359	1,754643
9	410	2,00391
10	478	2,336266
11	522	2,55132
12	544	2,658847
13	551	2,69306
14	555	2,71261
15	544	2,658847
16	536	2,619746
17	529	2,585533
18	522	2,55132
19	511	2,497556
20	503	2,458456
21	492	2,404692
22	484	2,365591

23	473	2,311828
24	462	2,258065
25	451	2,204301
26	439	2,14565
27	428	2,091887
28	414	2,02346
29	403	1,969697
30	395	1,930596
35	351	1,715543
40	289	1,412512
45	249	1,217009
50	216	1,055718
55	204	0,997067
60	192	0,938416
65	176	0,860215
70	164	0,801564
75	152	0,742913
80	144	0,703812
85	140	0,684262
90	136	0,664712
95	131	0,640274
100	126	0,615836
105	122	0,596285
108	120	0,58651

Tabela 3 – Tabela da calibração do sensor GP2Y0A02YK0F

Fonte: Elaborado pelo autor

GP2Y0A21YK0F		
CM	DADO	TENSÃO
1	374	1,827957
2	358	1,749756
3	562	2,746823
4	624	3,049853
5	648	3,167155
6	649	3,172043
7	641	3,132942
8	594	2,903226
9	540	2,639296
10	501	2,44868
11	460	2,248289

12	426	2,082111
13	397	1,940371
14	374	1,827957
15	351	1,715543
16	331	1,617791
17	311	1,520039
18	296	1,446725
19	280	1,368524
20	269	1,314761
21	253	1,236559
22	247	1,207234
23	233	1,138807
24	226	1,104594
25	218	1,065494
26	208	1,016618
27	205	1,001955
28	196	0,957967
29	193	0,943304
30	185	0,904203
35	169	0,826002
40	153	0,747801
45	149	0,72825
50	146	0,713587
55	145	0,7087
60	137	0,669599
65	137	0,669599
70	169	0,826002
75	169	0,826002
80	176	0,860215
85	185	0,904203
90	196	0,957967
95	208	1,016618
100	225	1,099707
105	230	1,124145
108	233	1,138807

Tabela 4 – Tabela da calibração do sensor GP2Y0A21YK0F

Fonte: Elaborado pelo autor

Desse modo, foi possível criar gráficos para confirmar os resultados e comparar dados, como visto nas Figuras 18, 19, 20 e 21.

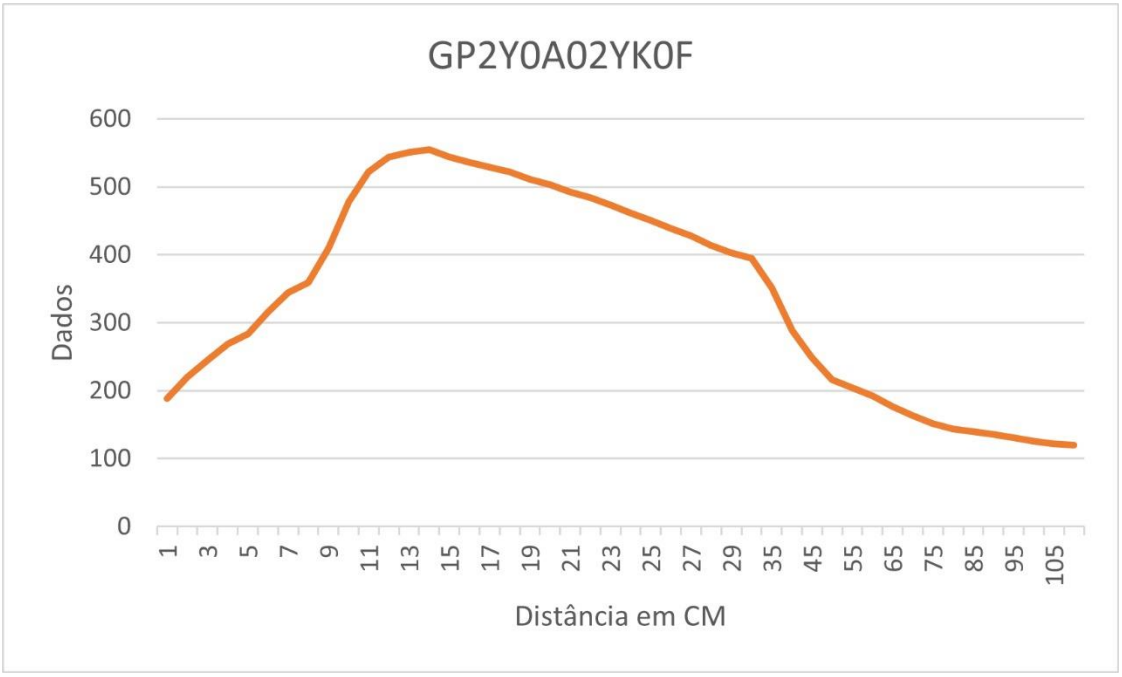

Figura 18 – Gráfico distância por valor em bits do sensor GP2Y0A02YK0F

Fonte: Elaborado pelo autor

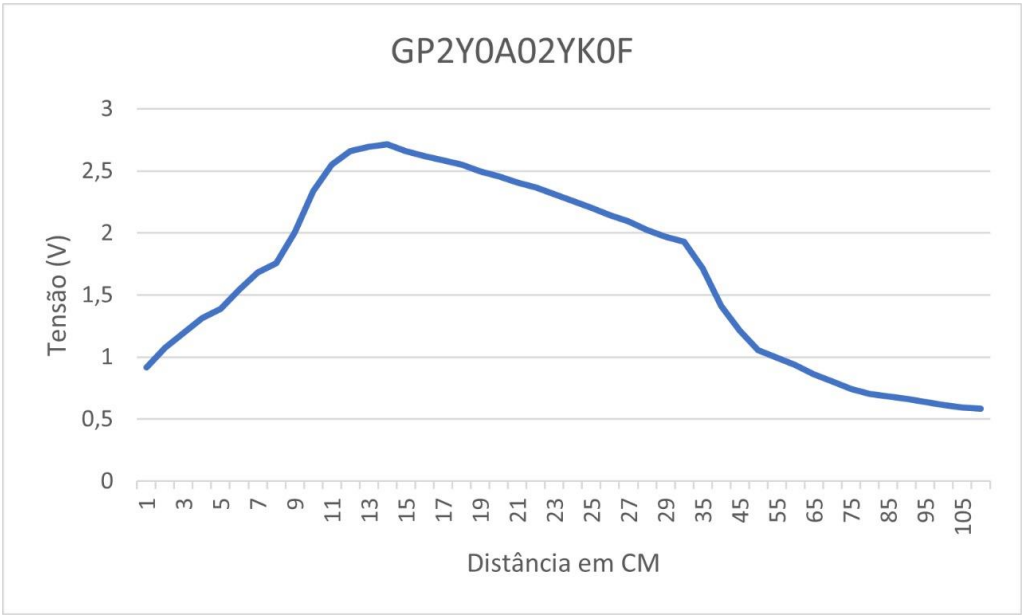

Figura 19 – Gráfico distância por tensão do sensor GP2Y0A02YK0F

Fonte: Elaborado pelo autor

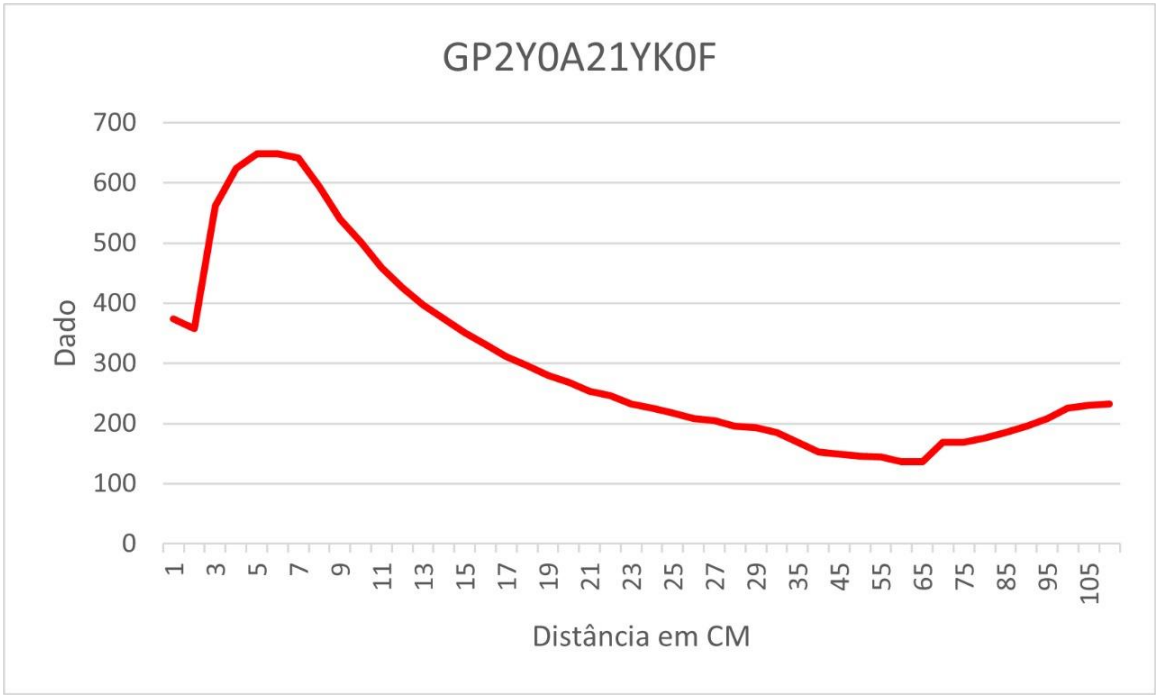

Figura 20 – Gráfico distância por valor em bits do sensor GP2Y0A21YK0F

Fonte: Elaborado pelo autor

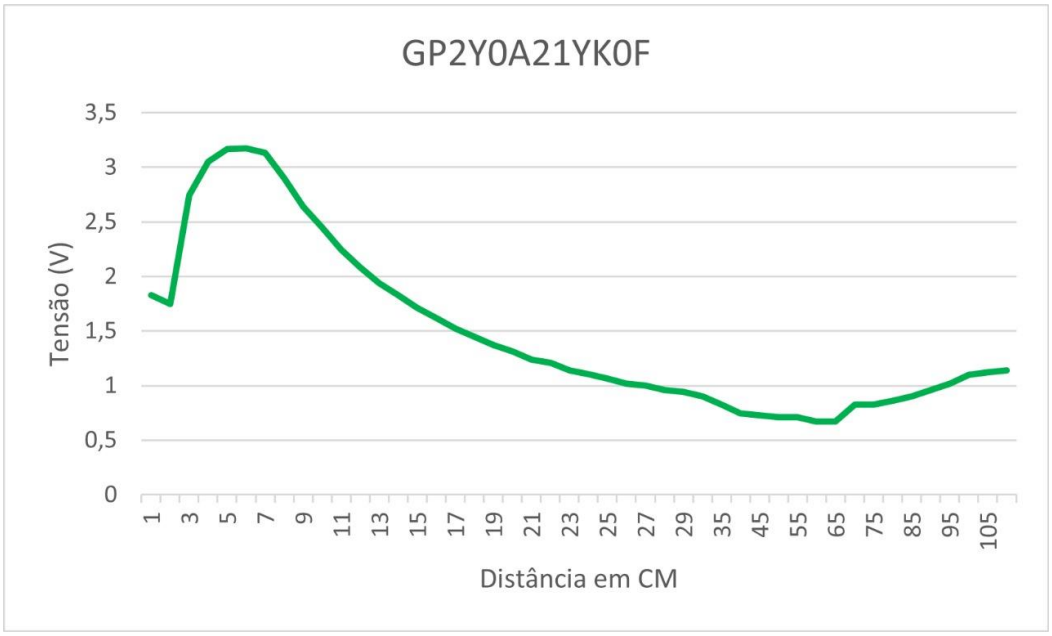

Figura 21 – Gráfico distância por tensão do sensor GP2Y0A21YK0F

Fonte: Elaborado pelo autor

Além disso, essa calibração também foi realizada para o sensor GP2Y0E03, porém foi necessário montar um circuito específico operando em 3.3V para isso devido às características do I²C e, consequentemente, um código específico. Esse circuito está representado na Figura 22, seu esquemático na Figura 23 e o programa no Código 2.

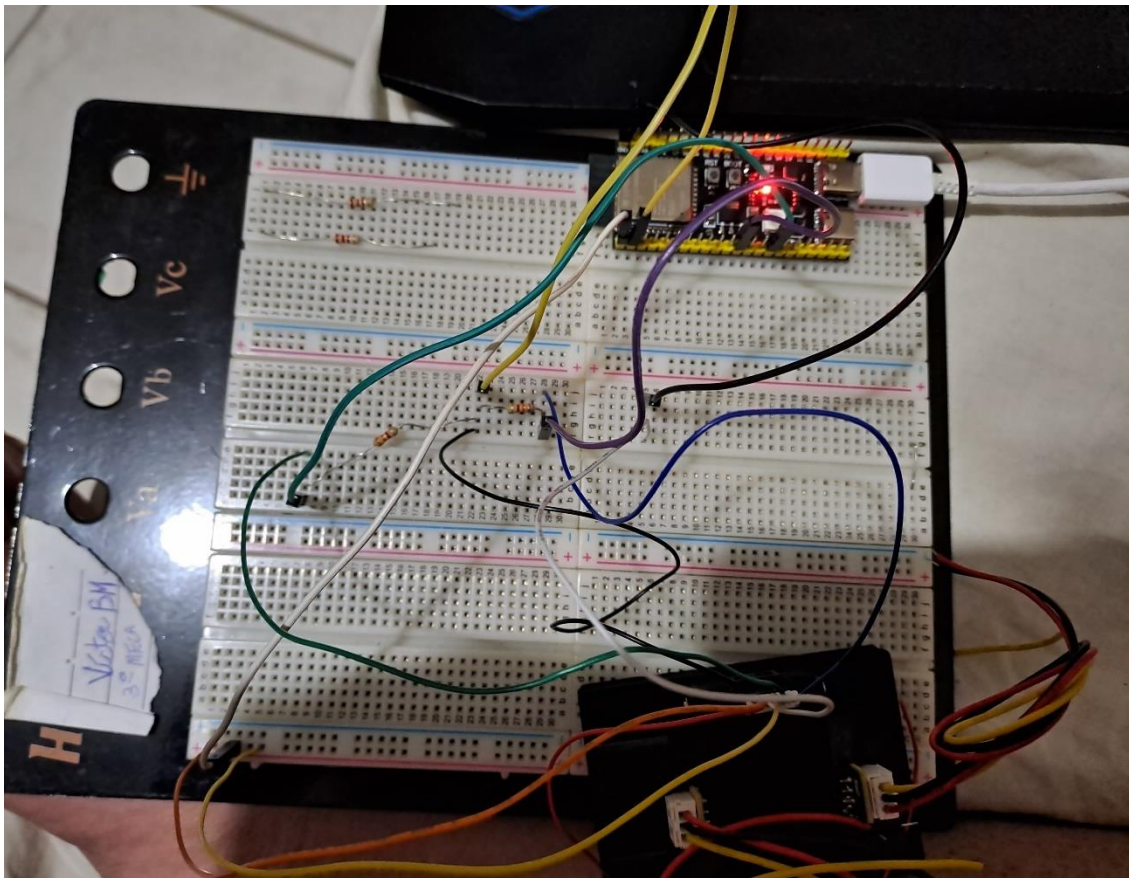

Figura 22 – Circuito para a calibração do GP2Y0E03

Fonte: Elaborado pelo autor

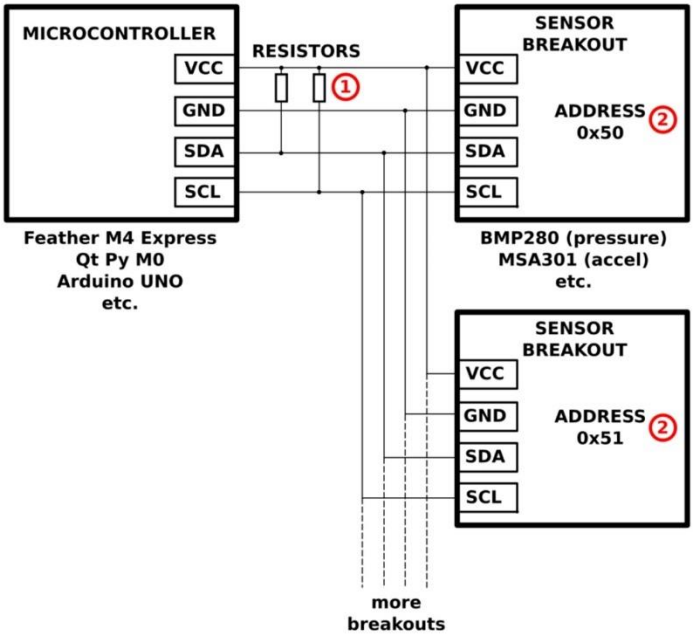

Figura 23 – Circuito esquemático com I²C

Fonte: Elaborado pelo autor

```
#include <I2C_16Bit.h>

#include <Wire.h>


#define I2C_ADDR 0x40

#define DATA_REGISTER_ADDR 0x5E


void setup() {

    Serial.begin(9600);

    I2C_16Bit_begin();

}


void loop() {
```

```
    uint16_t Dado = I2C_16Bit_readFromModule(I2C_ADDR, DATA_REGISTER_ADDR);
    Serial.print("Resposta: ");
    Serial.println(Dado);


    delay(1000);
}
```

Código 2 – Calibração do sensor Sharp com I²C

Fonte: Elaborado pelo autor

Nesse contexto, foi possível calibrar o sensor, obter resultados e compará-los, como visto na Tabela 5 e na Figura 24.

GP2Y0E03	
CM	DADO
1	2306
2	2307
3	65295
4	3855
5	5120
6	5901
7	6924
8	7948
9	8972
10	9999
11	11016
12	12046
13	13318
14	14351
15	15628
16	16904
17	18191
18	19464
19	20495
20	22272
21	23302

22	25101
23	26383
24	29185
25	30988
26	33026
27	36864
28	39431
29	40969
30	43274
31	45327
32	51467
33	54537
34	59909
35	64008
36	65295
37	65295
38	65295

Tabela 5 – Tabela da calibração do sensor GP2Y0E03

Fonte: Elaborado pelo autor

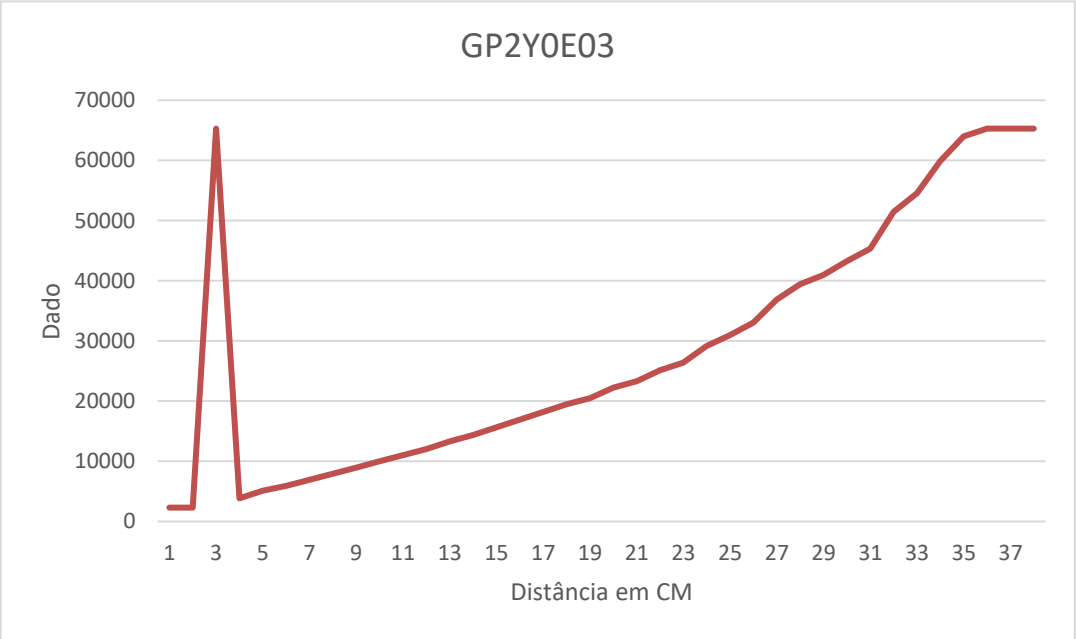

Figura 24 – Gráfico distância por dados da resposta do sensor GP2Y0E03

Fonte: Elaborado pelo autor

Ao comparar todos esses resultados, é perceptível que o sensor GP2Y0E03 possui uma resposta linear ideal para a formulação de um mapa de coordenadas.

No entanto, antes disso, foi necessário utilizar a função do método dos mínimos quadrados para traçar uma função quadrática que será utiliza para mapear e aferir distâncias (Figura 25 e Equação 1). Essa função está disponível no Excel e foi feita com as respostas do centímetro 1 ao centímetro 29, ignorando o centímetro 3. Ignorar o centímetro 3 foi uma escolha definida pelas características do sensor, uma vez que é um valor muito distoante da linearidade e que, devido às características mecânicas do projeto, é um valor que não precisa ser lido. Isso porque o suporte de sensores está 4 centímetros dentro da base, ou seja, os valores para a aferição de distância são relevantes após o centímetro 4. Já escolher terminar no centímetro 29 foi uma escolha motivada pelo tamanho da matriz que simboliza o mapa de coordenadas (Figura 26).

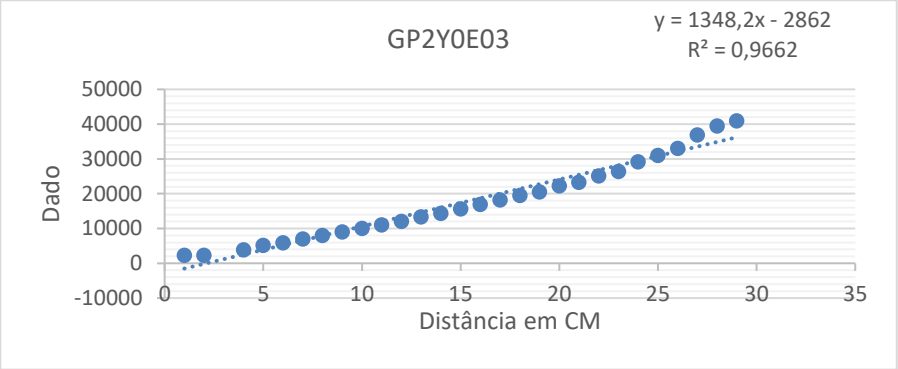

Figura 25 – Gráfico MMQ do GP2Y0E03

Fonte: Elaborado pelo autor

$$x = \frac{y + 2862}{1348,2}$$

Equação 1 – Equação MMQ da distância do GP2Y0E03

Fonte: Elaborado pelo autor

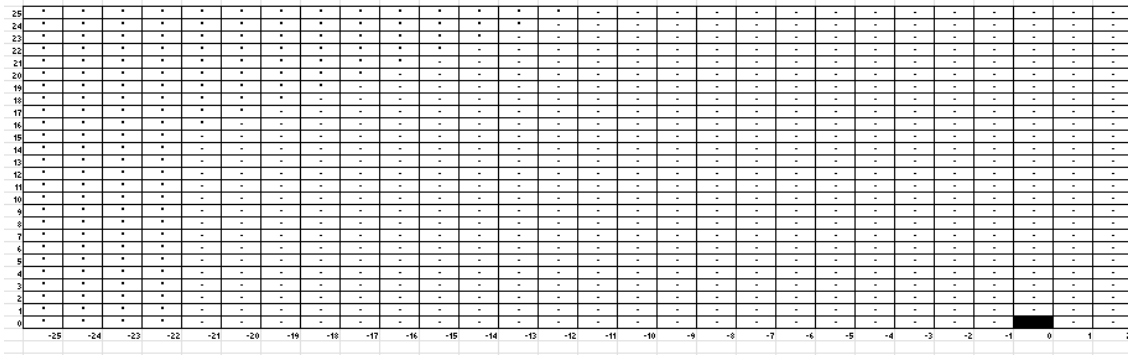

Figura 26 – Exemplo de mapa de coordenadas com 26 linhas e 51 colunas

Fonte: Elaborado pelo autor

Legenda:

* : Há obstáculos

- : Não há obstáculos

Bloco preto: Origem do mapa de coordenadas – posição do sensor

4.2. Mapeamento

Seguindo os parâmatros apresentados na calibração, foi possível realizar um código que afere distâncias, cria um mapa de coordenadas e movimenta o robô com o objetivo de desviar dos obstáculos.

Esse código é composto de diversas funções que possibilitam que seja um código mais polido. É válido ressaltar que os comandos para movimentar o robô não foram implementados de fato, mas sua lógica foi desenvolvida no programa.

Assim, o código de mapeamento foi desenvolvido para funcionar no *Arduino IDE*, ou seja, em C++. No entanto, para desenvolver a lógica, também foram usados o *TinkerCad* da *AutoDesk*, a linguagem C [16] e o *Visual Studio Code*.

Além da lógica desenvolvida previamente, foi necessário pensar em como a o sensor funcionaria no sistema rotativo. Isso está exemplificado na Figura 27.

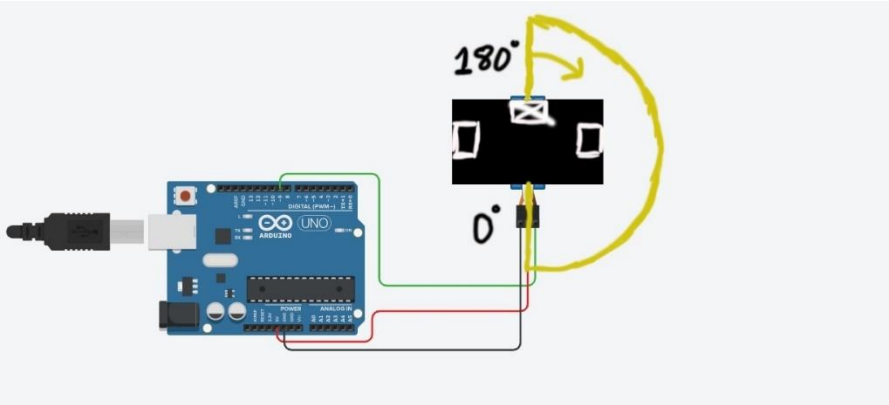

Figura 27 – Exemplo do funcionamento do sistema rotativo

Fonte: Elaborado pelo autor

Além das bibliotecas já presentes no *Arduino IDE,* foi necessário o uso de bibliotecas extras essenciais para o projeto. Elas são:

- *I2C_16Bit.h* [17];
- *Wire.h* [18];
- *Servo.h* [19].

Dessa forma, foi possível elaboarar o código de mapeamento, que conta com diversas funções e está presente em Código 3.

```
#include <I2C_16Bit.h> // Biblioteca de funções do sensor I2C
#include <Wire.h> // Biblioteca de comunicação I2C
#include <Servo.h> // Biblioteca do servo

#define I2C_ADDR 0x40 // Define o sensor
#define DATA_REGISTER_ADDR 0x5E // Define o registrador de dados do sensor
#define PI 3.141592654 // Define o valor de PI

int coord [26][51]; // Matriz
int i = 0; // Variável de linhas usada para printar zerar a matriz e para imprimí-la
int j = 0; // Variável de colunas usada para printar zerar a matriz e para imprimí-la
int pos = 0; // Variável usada para rotacionar o servo
double ang = 0; // Variável usada para calcular o ângulo em relação ao plano
cartesiano do mapa de pontos
```

```
int dist = 0; // Variável que armazena a distância, em centímetros, até o osbtáculo medido
int dado = 0; // Variável que armazena o OUTPUT do sensor I2C
float argx = 0; // Variável usada para medir o cosseno da distância aferida
float argy = 0; // Variável usada para medir o seno da distância aferida
int x = 0; // Variável usada para medir o valor no eixo X da distância até o obstáculo
int y = 0; // Variável usada para medir o valor no eixo Y da distância até o obstáculo

void setup(){
    servo_9.attach(9, 500, 2500); // Definindo o servo
    Serial.begin(9600);
    I2C_16Bit_begin(); // Inicializa o sensor
}

void loop(){
    zerador(); // Função para definir o default da matriz
    move(); // Função que faz com que o robô se movimente e desvie de obstáculos
    while (PI == 3.141592654){ // Função usada para que o robô funcione em malha fechada, funcionando até ser desligado
        for (pos = 0; pos <= 180; pos += 1) { // Sensor girando no sentido horário
            servo_9.write(pos);
            aquisic(); // Função que afere distâncias
            move(); // Função que faz com que o robô se movimente e desvie de obstáculos
            delay(10); // Dessa forma, um ciclo de 180º de sensoriamento demora 1,8 segundos
        }
        //imprime(); // Função para printar a matriz. Nesse caso, não estou utilizando para não poluir o terminal de programação
        for (pos = 180; pos >= 0; pos -= 1) { // Sensor girando no sentido anti-horário
            servo_9.write(pos);
            aquisic(); // Função que afere distâncias
            move(); // Função que faz com que o robô se movimente e desvie de obstáculos
            delay(10); // Dessa forma, um ciclo de 180º de sensoriamento demora 1,8 segundos
        }
        //imprime(); // Função para printar a matriz. Nesse caso, não estou utilizando para não poluir o terminal de programação
    }
}

void aquisic(){ // Função que afere distâncias
    uint16_t dado = I2C_16Bit_readFromModule(I2C_ADDR, DATA_REGISTER_ADDR); // Variável que armazena o OUTPUT do sensor I2C e realiza leitura de distâncias
    dist = (dado + 2862)/1348.2; // Função MMQ para aferir a distância
```

```
    dist = dist - 4; // Considerando que o sensor está 4cm no interior da base
        if (dist > 0){ // Desconsidera qualquer obstáculo dentro da base
            ang = 180 - pos; // Ângulo entre o sensor e a horizontal definida
            ang = (ang*PI)/180; // Converte o ângulo em graus para radianos
            argx = cos (ang); // Calcula o cosseno do ângulo em radianos
            x = argx * dist; // Calcula a distância no eixo X do obstáculo
            x = 25 + x; // Converte a distância no eixo X do obstáculo para colunas
da matriz
            argy = sin (ang); // Calcula o seno do ângulo em radianos
            y = argy * dist; // Calcula a distância no eixo Y do obstáculo
            y = 25 - y; // Converte a distância no eixo Y do obstáculo para linhas da
matriz
            if (x >= 0 && x <= 50 && y >= 0 && y <= 25){ // Permite apenas distâncias
válidas para a matriz
                coord [y][x] = 1; // Define esse ponto da matriz como 1, o que
significa a existência de obstáculos
            }
        }
}

void move(){ // Função que faz com que o robô se movimente e desvie de obstáculos
    if ((coord [25][21] == 1) && (coord [24][21] == 1) && (coord [23][21] == 1) &&
(coord [25][29] == 1) && (coord [24][29] == 1) && (coord [23][29] == 1) && (coord
[21][23] == 1) && (coord [21][24] == 1) && (coord [21][25] == 1) && (coord [21][26]
== 1) && (coord [21][27] == 1)){ // Se existem obstáculos na esquerda, na frente e na
direita do robô, faz com que ele vá para trás
        // -- COMANDO PARA GIRAR PARA A ESQUERDA -> aumenta a velocidade no motor
esquerdo e diminui a velocidade no motor direito
        // -- DELAY DE ACORDO COM o TEMPO QUE O ROBÔ LEVA PARA GIRAR 180º
        zerador(); // Função para definir o default da matriz
        // -- COMANDO PARA MOVIMENTAR PARA FRENTE -> mesma velocidade em ambos os
motores
    }
    else if ((coord [25][21] == 1) && (coord [24][21] == 1) && (coord [23][21] ==
1)){ // Se existem obstáculos à esquerda do robô, ele vira para a direita
        // -- COMANDO PARA GIRAR PARA A DIREITA -> aumenta a velocidade no motor
esquerdo e diminui a velocidade no motor direito
        // -- DELAY DE ACORDO COM o TEMPO QUE O ROBÔ LEVA PARA GIRAR 90º
        zerador(); // Função para definir o default da matriz
        // -- COMANDO PARA MOVIMENTAR PARA FRENTE -> mesma velocidade em ambos os
motores
    }
    else if ((coord [25][29] == 1) && (coord [24][29] == 1) && (coord [23][29] ==
1)){ // Se existem obstáculos à direita do robô, ele vira para a esquerda
        // -- COMANDO PARA GIRAR PARA A ESQUERDA -> aumenta a velocidade no motor
esquerdo e diminui a velocidade no motor direito
        // -- DELAY DE ACORDO COM o TEMPO QUE O ROBÔ LEVA PARA GIRAR 90º
        zerador(); // Função para definir o default da matriz
```

```
        // -- COMANDO PARA MOVIMENTAR PARA FRENTE -> mesma velocidade em ambos os
motores
    }
    else if ((coord [21][23] == 1) && (coord [21][24] == 1) && (coord [21][25] == 1)
&& (coord [21][26] == 1) && (coord [21][27] == 1)){ // Se existem obstáculos à frente
do robô, ele vira para a esquerda
        // -- COMANDO PARA GIRAR PARA A ESQUERDA -> aumenta a velocidade no motor
esquerdo e diminui a velocidade no motor direito
        // -- DELAY DE ACORDO COM o TEMPO QUE O ROBÔ LEVA PARA GIRAR 90º
        zerador(); // Função para definir o default da matriz
        // -- COMANDO PARA MOVIMENTAR PARA FRENTE -> mesma velocidade em ambos os
motores
    }
    else{ // Se não existem obstáculos, o robô movimenta-se para frente
        // -- COMANDO PARA MOVIMENTAR PARA FRENTE -> mesma velocidade em ambos os
motores
    }
}

void imprime(){ // Função para printar a matriz
    Serial.println();
    Serial.println();
    Serial.println();
    for (i = 0; i < 26; i++){
        for (j = 0; j < 51; j++){
            Serial.print (coord [i][j]);
        }
        Serial.println();
    }
}

void zerador(){ // Função para definir o default da matriz
    for (i = 0; i < 26; i++){ // For usado para zerar todos os valores da matriz.
Para esse sistema, 1 significa existência de obstáculos e 0 significa ausência de
obstáculos
        for (j = 0; j < 51; j++){
            coord [i][j] = 0;
        }
    }
    coord [25][25] = 2; // Define o centro da matriz como 2. Assim, o 2 significa a
origem do plano cartesiano que mapeia os pontos
}
```

Código 3 – Código de mapeamento para um sistema rotativo LiDAR

Fonte: Elaborado pelo autor [20]

Como pode ser visto, esse código é composto por 6 funções distintas. A função *“void setup”* está presente em qualquer código que utilize o *Arduino IDE*, assim como a função “*void loop”*. Essas funções garantem o funcionamento básico do código, seja por definições como na primeira, ou por manter o código básico funcionando ininterruptamente como na segunda.

No entanto, o diferencial desse projeto são as funções “*void aquisic”, “void move”, “void imprime”* e *“void zerador”.*

A função “*void aquisic”* é responsável por aferir distâncias com base na equação encontrada na calibração de sensores e gravá-las na matriz que representa o mapa de coordenadas.

A função “*void move”* é responsável por decidir como o robô deve movimentar-se, desviando de obstáculos. Para isso, foi considerado a distância de 5 centímetros da origem do mapa de pontos. Esse valor foi escolhido para garantir uma margem maior de distância entre o robô e o obstáculo, o que garante que não haja colisões.

A função “*void imprime”* é responsável por exibir a matriz de pontos caso o usuário queira ver. Nesse caso, essa função apresenta os dados registrados pela função *“void aquisic”*. A Figura 28 apresenta um caso de funcionamento dessa função. Nesse sentido, um obstáculo situado a 5 centímetros de distância do robô movimenta-se junto com Servo, o que proporciona um arco.

Figura 28 – Exemplo da matriz do mapa de coordenadas

Fonte: Elaborado pelo autor

Legenda:

1: Há obstáculos

0 : Não há obstáculos

2: Origem do mapa de coordenadas – posição do sensor

5. CONCLUSÕES

Esse relatório teve o objetivo de construir um *scanner* de baixo custo para aferir distâncias aplicável a robótica móvel.

No decorrer desse projeto de pesquisa, foi possível simular o funcionamento do *LiDAR* tradicional, criando um mapa de coordenadas que permite a aferição de distâncias. No entanto, o mais fator mais importante que deve ser ressaltado é que foi possível diminuir os custos para o uso desse tipo de tecnologia.

Apesar de todos os dispositivos utilizados na pesquisa, os únicos itens essenciais para esse *scanner* são:

- Suporte de sensores, cuja impressão 3D foi no valor de 5 reais e que o arquivo está disponível em [20];
- Sensor sharp GP2Y0E03, cujo preço é R$19,70. Isso pode ser visto na Figura [21];
- Micro Servo 9G SG90, cujo preço é R$18,90. Isso pode ser visto na Figura [22];
- Código de Mapeamento, disponível em [20].

Assim, considerando apenas o sensor, o preço total é de R$43,60, enquanto o RPLIDAR A1 custa R$437,89 considerando a cotação do dólar no dia 13 de dezembro de 2023.

Desse modo, foi possível realizar todos os objetivos do projeto, proporcionando um dispositivo útil e que integra diversas áreas do curso técnico de mecatrônica.

6. PERSPECTIVAS FUTURAS

XXX

7. REFERÊNCIAS BIBILOGRÁFICAS

[1] **RPLIDAR A1.** Disponível em: < https://www.generationrobots.com/media/rplidar-a1m8-360-degree-laser-scanner-development-kit-datasheet-1.pdf> . Acesso em: 13 de dez. de 2023.

[2] **GP2Y0A51SK0F.** Disponível em: <https://pdf1.alldatasheet.com/datasheet-pdf/view/1243993/SHARP/GP2Y0A51SK0F.html> . Acesso em: 13 de dez. de 2023.

[3] **GP2Y0D02YK0F.** Disponível em: <https://pdf1.alldatasheet.com/datasheet-pdf/view/412638/SHARP/GP2Y0D02YK0F.html> . Acesso em: 13 de dez. de 2023.

[4] **GP2Y0D21YK0F.** Disponível em: <https://pdf1.alldatasheet.com/datasheet-pdf/view/412641/SHARP/GP2Y0D21YK0F.html> . Acesso em: 13 de dez. de 2023.

[5] **GP2Y0E03.** Disponível em: <https://pdf1.alldatasheet.com/datasheet-pdf/view/1243994/SHARP/GP2Y0E03.html> . Acesso em: 13 de dez. de 2023.

[6] **GP2A25J0000F.** Disponível em: <https://pdf1.alldatasheet.com/datasheet-pdf/view/190720/SHARP/GP2A25J0000F.html> . Acesso em: 13 de dez. de 2023.

[7] **GP2Y0A02YK0F.** Disponível em: <https://pdf1.alldatasheet.com/datasheet-pdf/view/412633/SHARP/GP2Y0A02YK0F.html> . Acesso em: 13 de dez. de 2023.

[8] **GP2Y0A21YK0F.** Disponível em: <https://pdf1.alldatasheet.com/datasheet-pdf/view/412635/SHARP/GP2Y0A21YK0F.html> . Acesso em: 13 de dez. de 2023.

[9] **GP2Y0A41SK0F.** Disponível em: <https://pdf1.alldatasheet.com/datasheet-pdf/view/506283/SHARP/GP2Y0A41SK0F.html> . Acesso em: 13 de dez. de 2023.

[10] **Pololu Dual MC33926 Motor Driver Shield for Arduino.** Disponível em: <https://www.pololu.com/product/2503> . Acesso em: 13 de dez. de 2023.

[11] **Motor de engrenagem CC, 24V e 620RPM.** Disponível em: <https://motorlinkto.com/product/lgm25-370-25mm-dc-geared-motor/> . Acesso em: 13 de dez. de 2023.

[12] **Hc-Sr04.** Disponível em: <https://pdf1.alldatasheet.com/datasheet-pdf/view/1132204/ETC2/HCSR04.html> . Acesso em: 13 de dez. de 2023.

[13] **ESP32 S3 WROOM.** Disponível em: <https://www.espressif.com/sites/default/files/documentation/esp32-s3-wroom-1_wroom-1u_datasheet_en.pdf> . Acesso em: 13 de dez. de 2023.

[14] **Arduino Mega 2569.** Disponível em: <https://docs.arduino.cc/resources/datasheets/A000067-datasheet.pdf> . Acesso em: 13 de dez. de 2023.

[15] **Micro Servo 9G SG90.** Disponível em: <https://datasheetspdf.com/pdf/791970/TowerPro/SG90/1> . Acesso em: 13 de dez. de 2023.

[16] **Curso de Linguagem C da UFMG.** Disponível em: <http://paginapessoal.utfpr.edu.br/lalucas/disciplinas/el71e-s01-algoritmos/C_UFMG.pdf/view> . Acesso em: 13 de dez. de 2023.

[17] **Arduino-I2C.** Disponível em: <https://github.com/Wh1teRabbitHU/Arduino-I2C> . Acesso em: 13 de dez. de 2023.

[18] **Wire.h.** Disponível em: <https://www.arduino.cc/reference/en/language/functions/communication/wire/> . Acesso em: 13 de dez. de 2023.

[19] **Servo.h.** Disponível em: <https://www.arduino.cc/reference/en/libraries/servo/> . Acesso em: 13 de dez. de 2023.

[20] **Código de mapeamento LiDAR.** Disponível em: < https://github.com/Victor-BM/PIBIC-CEFET> . Acesso em: 13 de dez. de 2023.

[21] **GP2Y0E03.** Disponível em: < https://pt.aliexpress.com/item/1005003376196913.html> . Acesso em: 13 de dez. de 2023.

[22] **Micro Servo 9G SG90.** Disponível em: < https://www.eletrogate.com/micro-servo-9g-sg90-towerpro> . Acesso em: 13 de dez. de 2023.