UNIVERSIDADE ESTADUAL DO MARANHÃO
CENTRO DE CIÊNCIAS TECNOLÓGICAS
CURSO DE ENGENHARIA DA COMPUTAÇÃO

LAYS CAVALCANTE RIBEIRO

DESENVOLVIMENTO DE UM ROBÔ AUTÔNOMO PARA EXPLORAR E MAPEAR UM LOCAL DESCONHECIDO

São Luís – MA
2019

LAYS CAVALCANTE RIBEIRO

DESENVOLVIMENTO DE UM ROBÔ AUTÔNOMO PARA EXPLORAR E MAPEAR UM LOCAL DESCONHECIDO

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia da Computação da Universidade Estadual do Maranhão, como registro para obtenção do grau de Bacharel em Engenharia da Computação.

Orientador: Prof. MSc. Pedro Brandão Neto.

Coorientador: Prof. Elton de Sousa e Silva.

São Luís – MA

2019

Ribeiro, Lays Cavalcante.
Desenvolvimento de um robô autônomo para explorar e mapear um local desconhecido / Lays Cavalcante Ribeiro. – São Luís, 2019.

50 f.

Monografia (Graduação) – Curso de Engenharia da Computação, Universidade Estadual do Maranhão, 2019.

Orientador: Prof. MSc. Pedro Brandão Neto.
Coorientador: Prof. Elton de Sousa e Silva.

1.Detecção de obstáculos. 2.Robô autônomo. 3.Raspberry Pi. 4.Mapeamento. I.Título

CDU: 004.896

LAYS CAVALCANTE RIBEIRO

DESENVOLVIMENTO DE UM ROBÔ AUTÔNOMO PARA EXPLORAR E MAPEAR UM LOCAL DESCONHECIDO

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia da Computação da Universidade Estadual do Maranhão, como registro para obtenção do grau de Bacharel em Engenharia da Computação.

Aprovada em: de agosto de 2019

BANCA EXAMINADORA

_____

Prof. MSc. Pedro Brandão Neto (Orientador)
Universidade Estadual do Maranhão

_____

Prof. Elton de Sousa e Silva (Coorientador)
Universidade Estadual do Maranhão

_____

Prof. Dr. Lúcio Flávio de Albuquerque Campos (Examinador)
Universidade Estadual do Maranhão

_____

Prof. Dr. Luís Carlos Costa Fonseca (Examinador)
Universidade Estadual do Maranhão

AGRADECIMENTOS

Agradeço aos meus pais Clézia Cavalcante Ribeiro e Luís de Andrade Ribeiro, que sempre estiveram ao meu lado e batalharam para me dar a melhor educação possível. Ao meu irmão Luís de Andrade Ribeiro Júnior que sempre me animou e a minha irmã Lianna Cavalcante Ribeiro que além de me apoiar, ajudou a editar algumas imagens da monografia.

Agradeço ao meu orientador Pedro Brandão, que com muita paciência cedeu seu tempo para me orientar durante todas as etapas da monografia. Ao meu coorientador Elton de Sousa, que forneceu quase todos os equipamentos necessários para a execução do projeto proposto na monografia e me auxiliou durante os problemas encontrados durante o desenvolvimento do mesmo.

Agradeço aos meus amigos Eduardo Andrade, João Francisco, Lorena Tavares, José Pedro, Rodrigo Albuquerque e Lucas Ferreira pela amizade e apoio ao longo dos anos durante o curso. A Claudyane Araújo, que me ajudou com algumas disciplinas ao longo do curso. Um agradecimento especial ao Italo Feitosa e a Paola Lauande, que além da amizade e apoio me ajudaram em algumas etapas da monografia.

“A ciência nunca resolve um problema sem criar pelo menos outros dez”.

(George Bernard Shaw)

RESUMO

Este trabalho apresenta a montagem e programação de um protótipo robótico autônomo que consegue explorar um local desconhecido e gerar um mapa baseado nos dados coletados. A montagem foi feita com um chassi de acrílico, uma Raspberry Pi e componentes eletrônicos como motores DC, Ponte H e módulos sensores ultrassônico, magnetômetro e acelerômetro. O controle da movimentação foi feito usando a técnica de PWM e a conversão dos dados obtidos pelo módulo acelerômetro foi feita através de integração numérica. O robô foi programado em linguagem Python e as bibliotecas math, time, gpiozero, matplotlib e i2c foram utilizadas para alcançar os objetivos desejados. Um algoritmo foi desenvolvido para escolher um lado para seguir, detectar obstáculos e coletar dados. Os dados coletados alimentaram o algoritmo de geração de mapas desenvolvido, que foi programado para gerar uma planta baixa semelhante à uma matriz binária, explicada na metodologia deste trabalho. A forma como o algoritmo foi planejado e implementado fez com que os mapas gerados tivessem apenas alguns obstáculos com relação ao mapa real. Apesar de algumas limitações físicas, o trabalho gerou resultados satisfatórios para um protótipo.

Palavras-chave: Detecção de obstáculos, robô autônomo, raspberry pi, mapeamento.

ABSTRACT

This work presents the assembly and programming of an autonomous robotic prototype that can explore an unknown location and generate a map based on the collected data. The assembly was made with an acrylic chassis, a Raspberry Pi, and electronics such as DC motors, an H Bridge and ultrasonic, magnetometer and accelerometer modules. The movement control was made using the PWM technique and the conversion of the data obtained by the accelerometer module was done through numerical integration. The robot was programmed in Python language and the math, time, gpiozero, matplotlib and i2c libraries were used to achieve the desired objectives. An algorithm was developed to choose a side to follow, detect obstacles and collect data. The collected data fed the developed map generation algorithm, which was programmed to generate a blueprint like a binary matrix, explained in the methodology of this work. The way the algorithm was designed and implemented meant that the generated maps had only a few obstacles to the actual map. Despite some physical limitations, the work yielded satisfactory results for a prototype.

Keywords: Obstacle detection, autonomous robot, raspberry pi, autonomous robot, mapping.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

CPU	Central Processing Unit (Unidade Lógica de Processamento)
DC	Direct Current (Corrente Direta)
E/S	Entrada e Saída
GPIO	General Purpose Input/Output (Entrada e Saída de Propósito Geral)
IDLE	Integrated Development and Learning Environment (Ambiente Integrado de Desenvolvimento e Aprendizagem
LED	Light Emitting Diode (Diodo Emissor de Luz)
PWM	Pulse-Width Modulation (Modulação por Largura de Pulso)
RAM	Random Access Memory (Memória de Acesso Aleatório)
USB	Universal Serial Bus (Porta Universal)
Wi-Fi	Wireless Fidelity (Fidelidade sem Fio)

SUMÁRIO

1 INTRODUÇÃO

Edifícios são construídos através de normas de segurança que contribuem para a prevenção e mitigação de desastres. Quando não é possível eliminar completamente a possibilidade de acontecimento de tais desastres, são elaborados também planos de emergência que consistem em organizar as possíveis respostas para tais acontecimentos (ARAÚJO, 2011, p. 52).

Durante situações de desastres como incêndios ou desmoronamentos de edifícios, as pessoas podem encontrar dificuldades para evacuar o local, devido a inúmeros obstáculos que podem estar bloqueando passagens ou corredores do ambiente. Tais obstáculos podem ser causados por labaredas, partes da estrutura interna como forros ou paredes desabadas ou até mesmo pelo grande fluxo de pessoas passando por um único local, o que exige a busca de rotas alternativas para a evacuação.

Dados os perigos existentes diante de desastres, o auxílio de robôs durante evacuações torna-se viável e recomendado quando o ambiente apresenta riscos excessivos para a ação de resgate dos bombeiros. No geral, os robôs construídos para auxílio de bombeiros são máquinas robustas móveis, construídas com material resistente a altas temperaturas, podendo ser controlados pelos bombeiros ou atuando de forma autônoma. Nesse segundo caso, os robôs podem ser programados usando técnicas que possibilitam captar informações interagindo com um ambiente e tomar decisões (BALTZAN; PHILLIPS, 2012, p. 39).

Um robô pode ser projetado para percorrer e mapear um local de planta baixa desconhecida, desviando de obstáculos e calculando melhores rotas de acesso entre dois pontos para auxiliar no processo de evacuação de um edifício após a ocorrência de um desastre.

Por essas razões, neste trabalho é proposto o desenvolvimento de um protótipo robótico para o auxílio em situações de evacuação através da exploração e mapeamento de um local desconhecido. Para isto, o protótipo deve ser capaz de locomover-se de forma autônoma em um local plano, identificando e desviando de obstáculos. Após percorrer uma determinada sequência de ações, o robô deve enviar os dados coletados para um computador, onde eles serão processados e utilizados para criação de uma nova planta baixa do local em questão, representando a estrutura do local após o desastre.

1.1 Estrutura do Trabalho

O presente trabalho está dividido em cinco capítulos. O primeiro é a fundamentação teórica, onde serão conceituados tópicos importantes para a compreensão do trabalho, como as etapas de um desastre, robótica, sistemas embarcados, bibliotecas, sensores ultrassônico, magnetômetro e acelerômetro. No capítulo seguinte será descrita a metodologia usada no projeto desenvolvido, com especificações do sistema embarcado e sensores utilizados, além da explicação da abordagem utilizada para executar uma busca por obstáculos e efetuar o mapeamento do local percorrido.

Em sequência, no capítulo 4, serão abordados os resultados obtidos durante o projeto, além de explicações sobre os problemas encontrados. Por fim o último capítulo apresentará as conclusões obtidas e propostas para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Desastres

Um desastre é um evento destrutivo não planejado que se apresenta em diferentes proporções e que pode ter causa natural ou humana, como incêndios, inundações, desabamentos, entre outros. De acordo com a Coordenadoria de Defesa Civil (2018), desastres são eventos adversos que acontecem pela existência das vulnerabilidades decorrentes do desenvolvimento global, causando prejuízos econômicos e sociais intensos.

Atualmente são feitos estudos que medem a probabilidade e impactos dos desastres, analisando as características das vulnerabilidades existente para os cenários estudados. Baseando-se nestes estudos os desastres podem ser divididos em ciclos compostos pelas etapas de prevenção, preparo, resposta e reconstrução (Política Nacional de Defesa Civil, 2007).

A primeira etapa do ciclo é a prevenção. Esta tem como objetivo evitar o acontecimento do desastre ou, quando não houver como evitá-lo, desenvolver formas de reduzir a intensidade dos seus danos. A Defesa Civil cita algumas das atividades preventivas mais comuns como:

- Conservação de bacias hidrográficas;
- Sistemas de irrigação;
- Programas de prevenção a incêndios;
- Programas de investigação de fenômenos perigosos;
- Elaboração de mapas de risco e vulnerabilidades.

Uma vez que a prevenção nem sempre é possível, medidas de mitigação também devem ser consideradas. Dentre estas medidas estão o reforço de construções, conservação de solos, aplicação de normas de saúde pública e outros.

Após prevenir, deve ser analisado como preparar o ambiente para o acontecimento do desastre. Nesta segunda etapa são desenvolvidos um conjunto de ações a serem tomadas por indivíduos e organizações no caso da ocorrência de determinado desastre. Aqui é onde são elaboradas estratégias de resposta tais como planos de emergência, além da capacitação dos indivíduos, que pode ser feita por meio de exercícios de simulação.

Na etapa de resposta são executadas as ações planejadas durante o preparo, com objetivo principal de salvar vidas e proteger bens. Aqui é colocado em prática planos de emergência e contingência. Dentre as atividades executadas nesta etapa estão a busca e resgate, além de assistência médica e evacuações (ARAÚJO, 2011, p. 56).

Reconstrução é a última etapa do ciclo de um desastre. Nesta etapa ocorrem os reparos aos danos causados, buscando reativar a economia da região afetada. Durante o processo de reconstrução várias instituições podem se envolver, incluindo as de setor privado. Ao fim desta etapa devem ser reimplantadas as medidas de prevenção e mitigação para possibilidade de repetição do desastre (ARAÚJO, 2011, p. 58-59).

Após a concretização de um desastre é preciso contar com todos os recursos tecnológicos possíveis para atuar principalmente na etapa de resposta durante a execução dos planos de emergência e contingência (TOASSI; STOLF; OLIVEIRA, 2006). Algumas das tecnologias já utilizadas pelo corpo de bombeiros são sistemas de detecção e alarme de incêndios, caracterizados como sistemas embarcados, além aplicação da robótica no auxílio de extinção de incêndios. Neste trabalho será aplicada a robótica para o mapeamento de locais atingidos por desastres, com a proposta de complementar os planos de emergência já utilizados atualmente.

2.2 Introdução a Robótica

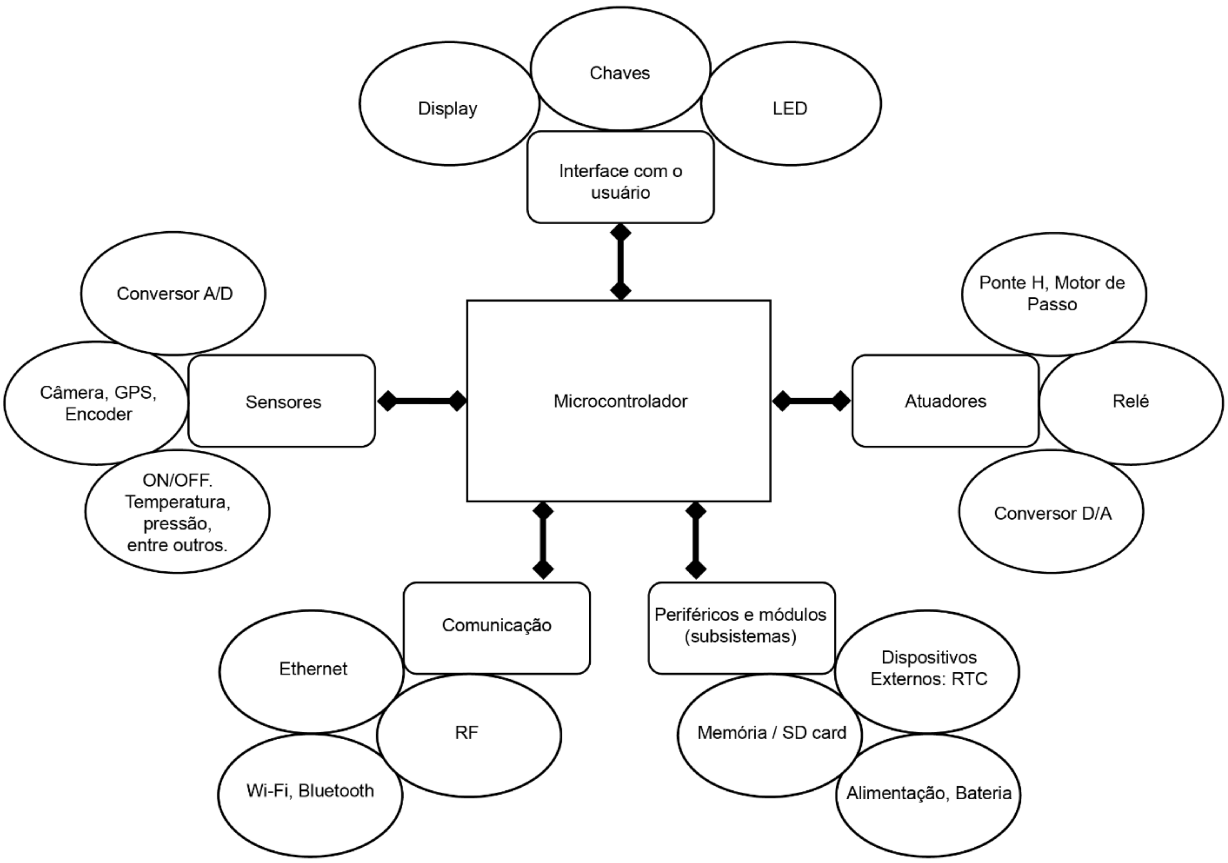
Robótica é o termo dado ao ramo de estudo e uso de robôs. O termo robô veio da palavra tcheca *robota*, que significa trabalho forçado, conceituando um robô como um trabalhador que realiza tarefas que humanos não podem ou querem executar. Portanto, os robôs são máquinas automáticas que podem ser programadas para realizar tarefas como manipulação de objetos, exploração de lugares, processamento de materiais e outras (PAZOS, 2002).

Os robôs estão presentes no setor industrial, com braços robóticos para manipulação de objetos pesados, no ambiente doméstico, com brinquedos e aspiradores de pó autônomos, e no ambiente de trabalho, principalmente os que envolvem manuseio de materiais pesados ou apresentam riscos para humanos. Nos subcapítulos a seguir serão apresentados alguns conceitos e componentes envolvidos no processo de construção de um robô.

2.3 Sistemas Embarcados

Sistemas embarcados são sistemas compactos desenvolvidos para objetivos específicos. Por se tratar de algo portátil, seu consumo de energia deve ser menor que o de sistemas tradicionais e, consequentemente, sua capacidade de processamento também será reduzida. Sendo assim este tipo de sistema requer equipamentos de hardware especiais, além de versões de softwares mais leves.

Figura 1 - Arquitetura de Sistemas Embarcados.



Fonte: Garcia (2018), com adaptações.

A arquitetura de um sistema embarcado tem como base um microcontrolador, que possui uma CPU, memória e dispositivos de E/S (Figura 1), como periféricos de interface com usuário, comunicação, sensores, atuadores e subsistemas. Dentre os exemplos de sistemas embarcados pode-se citar sistemas internos de carros e aviões, sistemas de controle, sistemas relacionados à IoT (*Internet of Things*), sistemas de uso doméstico como aspiradores de pó autônomos e até mesmo brinquedos. Um microcontrolador muito conhecido utilizado em variados projetos como sistema embarcado é a Raspberry Pi.

2.4 Raspberry e Raspbian

Raspberry Pi é um pequeno computador, semelhante a um computador comumente utilizado que deve ser conectado a dispositivos de E/S tais como mouse, teclado e monitor, funcionando com sistemas operacionais específicos para sistemas embarcados. O sistema Raspbian é um sistema criado especificamente para uso na Raspberry Pi. Ele foi desenvolvido com base no sistema Debian, baseado em Linux, e dado como funcional em junho de 2012. Ele já vem com vários pacotes e bibliotecas instaladas, além da IDLE própria para desenvolvimento em Python.

2.5 Python e Bibliotecas

Python é uma linguagem de programação orientada a objetos, tornando possível deixar códigos mais modulares e organizados. Uma biblioteca é um script com um conjunto de instruções chamados de métodos. Os métodos de uma biblioteca são organizados de forma que cada biblioteca contém métodos para um determinado uso, como operações matemáticas, manipulações de tempo ou geração de gráficos.

Ao escrever um código com auxílio de uma biblioteca, o programador não precisa criar e repetir determinadas instruções, podendo apenas chamá-la de dentro da biblioteca, gerando códigos mais compactos e organizados. A linguagem Python tem um conjunto de bibliotecas conhecido como bibliotecas padrão, encontradas nativamente na linguagem, e outras criadas pela comunidade da linguagem. Neste projeto foram utilizadas as bibliotecas:

- Time – Esta é uma biblioteca de manipulações relacionadas ao tempo. Dela é possível adquirir o tempo atual com o método time(), além de pausar uma thread em execução como método sleep();
- Math – Uma biblioteca voltada para operações matemáticas que não envolvem números complexos. Com ela é possível calcular fatoriais, módulos, mmc, exponencial, logaritmo, raiz quadrada, funções trigonométricas, conversão de graus e radianos e outros;
- Numpy – Trata-se de uma biblioteca de computação científica, isto é, uma biblioteca de modelagem aplicação de técnicas matemáticas, como transformadas de Fourier e operações com matrizes, além de ter métodos para interação com arquivos de texto. Na biblioteca numpy existem métodos como: pad(), que preenche uma matriz com determinados números ao redor da mesma, originando uma nova matriz; shape(), que retorna o número de linhas e colunas de uma matriz; rot90(), usado para rotacionar matrizes; savetxt(), que salva um vetor em um arquivo de texto; load(), que lê um arquivo de texto e o insere em um vetor de inteiros; e genfromtxt(), que lê um arquivo de texto e o insere em um vetor de strings;
- GPIOZERO – É instalada por padrão no sistema operacional Raspbian (NUTTALL, 2019). Seus métodos são responsáveis pela interação com equipamentos de hardware como LED's, motores e sensores. Dois dos módulos desta biblioteca são: Motor e DistanceSensor. O módulo Motor faz o controle dos motores DC, com os métodos forward() e backward(), que podem controlar a

rotação pra frente e pra trás das rodas de um robô, enquanto o módulo DistanceSensor realiza o cálculo da distância medida por um sensor ultrassônico;

- I2C – Esta biblioteca permite controlar a interface i2c da Raspberry Pi. A interface i2c se baseia na possibilidade de conectar mais de um dispositivo em um único par de barramento, um de clock e um de dados, de forma que todos os dispositivos conectados à essa interface compartilharão os mesmos pinos GPIO (VALDEZ; BECKER, 2015);
- Matplotlib – É uma biblioteca de criação de gráficos em duas dimensões. Ela possui um módulo chamado pyplot, que permite adaptar o estilo das linhas do gráfico, fonte, propriedades dos eixos, entre outros (HUNTER, 2007). Dentre seus métodos estão o figure() e o imshow(), os quais criam uma figura vazia e geram um gráfico à partir de uma matriz, respectivamente.

2.6 Sensores

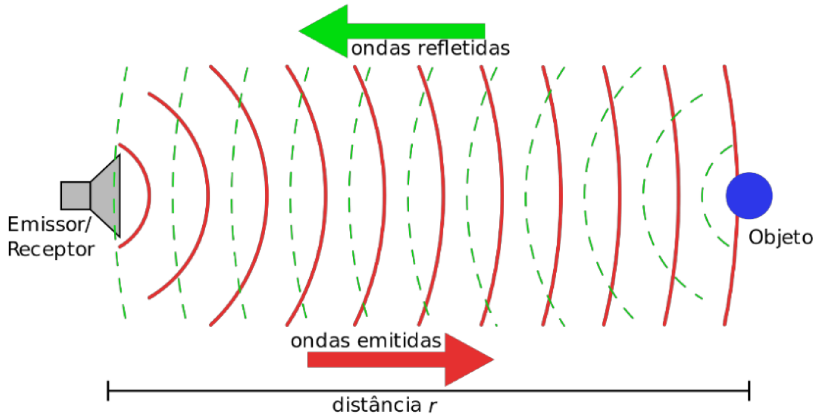
Sensor é um dispositivo que recebe um estímulo do ambiente e emite uma saída a ser convertida ou interpretada. Esses estímulos, que podem ser elétricos, magnéticos, ópticos, mecânicos, acústicos ou térmicos, são processados e utilizados como parâmetros para a aplicação desejada.

Uma forma de classificar os sensores é com relação à saída gerada por eles, de forma que eles podem ser discretos ou de medição. Os sensores discretos são aqueles que ao receberem um estímulo geram uma saída binária, enquanto os de medição geram uma saída em forma de amplitude de sinal. Se esses sensores não só gerarem uma resposta, mas também converterem o sinal recebido, eles são chamados de transdutores. Neste trabalho foram usados três sensores: ultrassônico, magnetômetro e acelerômetro.

2.6.1 Ultrassônico

O funcionamento do sensor ultrassônico consiste na emissão de uma onda mecânica, inaudível para o ser humano, que é refletida por objetos e capturada por um receptor (Figura 2). Com a variação de tempo existente do momento que a onda é enviada até o momento em que o receptor a recebe, é possível calcular a distância até o determinado objeto que refletiu a onda.

Figura 2 - Sensor Ultrassônico.



Fonte: Wikipédia (2010).

Através da medição da distância até um objeto, este sensor pode ser usado na detecção de objetos, rompimento de fios, detecção de presença, medidor de dimensões de objetos, monitoramento de nível de líquidos e posicionamento (SILVEIRA). A tecnologia ultrassônica não tem seu resultado afetado pela luminosidade do local de funcionamento, então, uma vez que o cenário proposto é de iluminação indeterminada por não se saber exatamente como está a estrutura interna do local, foi escolhido trabalhar com este tipo de sensor.

2.6.2 Magnetômetro

Um magnetômetro é um dispositivo que mede o campo magnético na terra ou no espaço. Os magnetômetros produzidos para medir campo magnético terrestre podem ser calibrados com constantes internas ou com referências a um determinado campo terrestre conhecido (MAGNETOMETER, 2009).

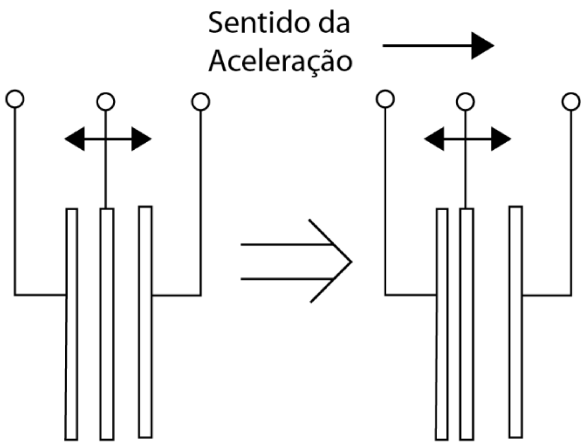
Uma das tecnologias usadas em magnetômetros é a magnetorresistência, que se baseia na capacidade que alguns materiais têm de mudar sua resistência elétrica quando expostos a diferentes intensidades e/ou orientações de campo magnético (SELIGMAN, 2002, p. 31). Quando aplicada uma rotação ao magnetômetro, o material muda sua resistência elétrica, de forma que a variação sofrida pode ser usada para medir a rotação efetuada.

2.6.3 Acelerômetro

Um sensor de aceleração, ou acelerômetro, é capaz de medir a força de aceleração em relação à força de gravidade. Esta aceleração é capturada por um mecanismo de detecção de força e, dependendo do modelo do acelerômetro, a medição pode ser feita em apenas um ou nos três eixos X, Y e Z.

Uma das formas de funcionamento do mecanismo de detecção de força é pelo efeito piezoelétrico. O efeito piezoelétrico trata-se da capacidade que alguns materiais, como o quartzo, têm de gerar tensão quando pressionados (REZENDE, 2004, p. 473). Quando sujeito a uma aceleração, esse material é comprimido por uma massa não fixa dentro do sensor, gerando uma tensão, que por sua vez é usada para medir a força de aceleração aplicada sobre o sensor.

Figura 3 - Acelerômetro capacitivo.



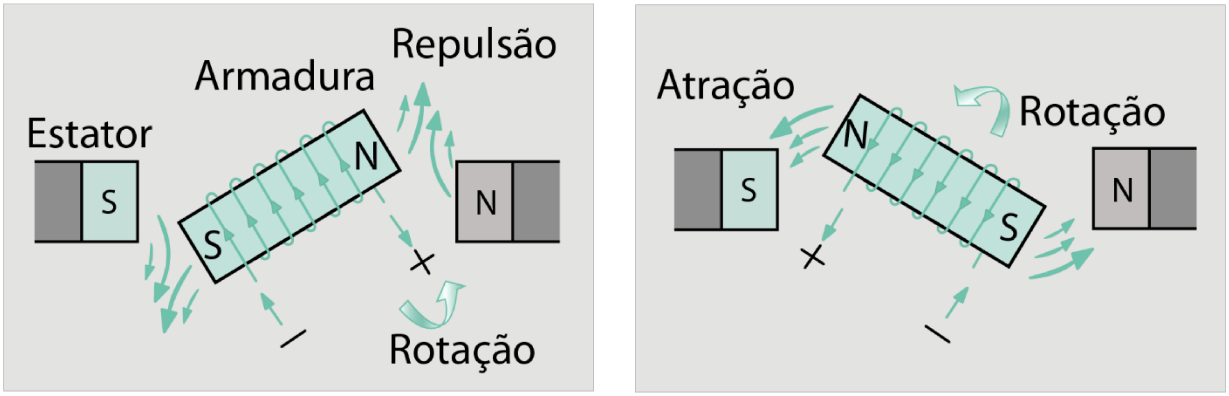
Fonte: Eckelberg (2013), com adaptações.

Outra forma de funcionamento é pela detecção da capacitância. Nesta configuração, duas placas metálicas fixas cercam uma placa central móvel, denominada massa de prova, formando dois capacitores. Quando é aplicada uma aceleração, a massa de prova se move, aumentando ou diminuindo a capacitância dos capacitores, que é utilizada para calcular a força aplicada, equivalente à aceleração (ROCHA; MARRANGHELLO, 2013).

2.7 Motores DC e Ponte H

Motor DC é um motor que funciona alimentado por uma corrente contínua e tem sua rotação gerada pela interação entre dois campos magnéticos. O motor é composto basicamente pelo estator e pela armadura (rotor), que são respectivamente, a parte fixa e a parte móvel que efetua as rotações do motor. Ambas as partes são energizadas de forma a criar um campo magnético para o estator e um para o rotor, os dois com polos positivos e negativos. Quando os dois polos iguais estão próximos, eles se repelem, fazendo o rotor girar até que os polos opostos se encontrem (Figura 4).

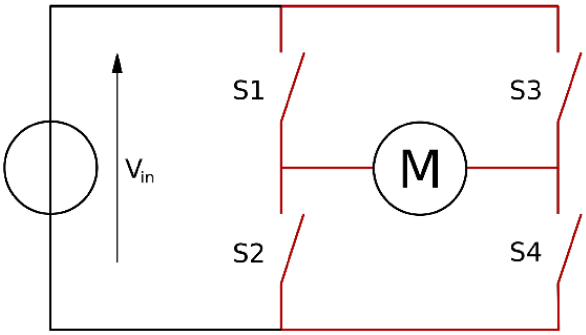
Figura 4 - Funcionamento do motor.



Fonte: PETRUZELLA (2013, p. 115), com adaptações.

Pela atração dos polos opostos o motor pararia de girar, por isso o sentido da corrente do rotor é invertido, invertendo sua polaridade e fazendo com que o rotor torne a girar. Esse processo fica se repetindo, fazendo com que o motor continue girando enquanto for alimentado (PETRUZELLA, 2013, p. 115).

Figura 5 - Representação das chaves de uma Ponte H.



Fonte: Wikipédia (2006).

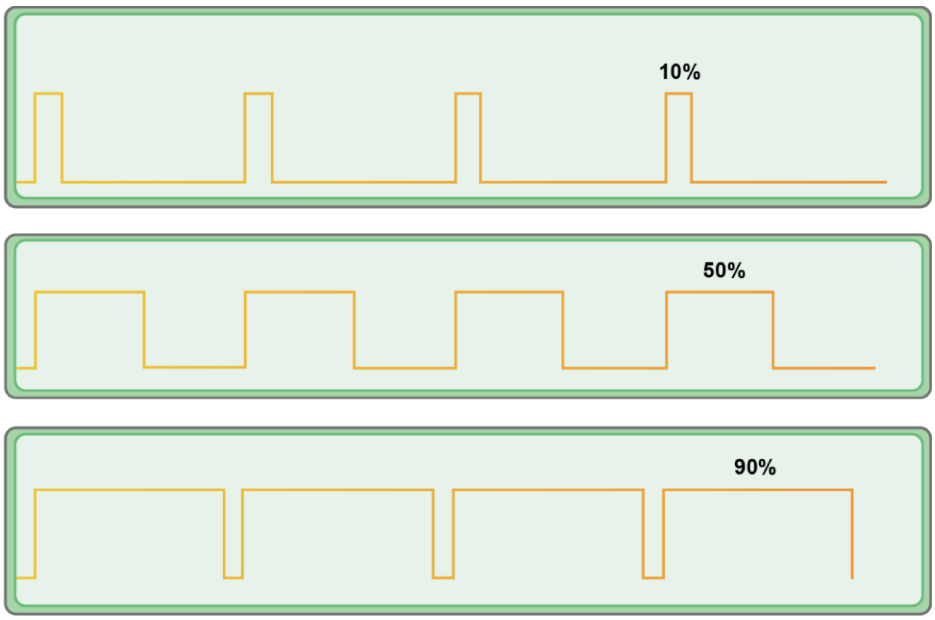
Os motores DC podem ser utilizados em aplicações em que seja necessário mudar o sentido da rotação do motor, que pode ser feito com o auxílio de uma Ponte H. A ponte H é um circuito construído por quatro chaves acionadas de forma alternada, que variam o sentido da corrente e o valor da tensão aplicados ao motor DC.

2.8 PWM

O PWM (Pulse Width Modulation) é uma técnica utilizada para converter digitalmente a amplitude de um sinal com oscilações rápidas do mesmo entre sua amplitude máxima e zero, de forma que o sinal parece estar constante (BARR, 2001). As oscilações podem ser configuradas para oscilar com amplitude máxima por mais, ou menos tempo, para que a amplitude média seja a desejada. O PWM é muito utilizado em aplicações de controle de

voltagem ou corrente de dispositivos eletrônicos como motores DC, podendo ser feita a regulagem das rotações do motor.

Figura 6 - Variação da amplitude com PWM.



Fonte: Barr (2001), com adaptações.

A Figura 6 ilustra três diferentes pulsações de um mesmo sinal, controlados por PWM. O primeiro permanecesse ligado apenas 10% do tempo total de sua frequência, o segundo 50% e o terceiro 90%, gerando um sinal de saída com respectivamente 10%, 50% e 90% da amplitude do sinal original.

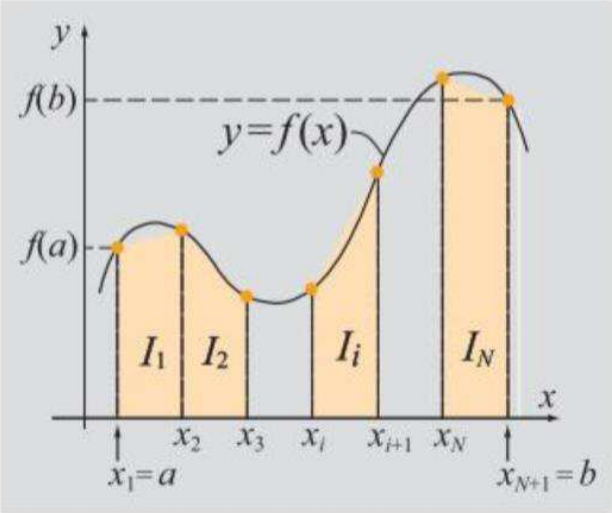
2.9 Integração Numérica

A integração numérica consiste em métodos de aproximação da resolução de uma integral. Em alguns casos o integrando é dado como um conjunto discreto de pontos que torna difícil ou impossível resolver a integral da forma padrão (GILAT; SUBRAMANIAM, 2008, p. 288). Um dos métodos de aproximação existentes é o método trapezoidal, ou método dos trapézios.

2.9.1 Método Trapezoidal e Trapezoidal Composto

O método trapezoidal consiste em considerar a área da integral equivalente à área de um trapézio. Então uma integral $I(f)$ no intervalo de a até b é dada por $I(f) \approx \frac{[f(a)+f(b)]}{2} * (b-a)$.

Figura 7 - Método Trapezoidal Composto.



Fonte: Gilat e Subramaniam (2008, p. 293).

O método trapezoidal composto consiste em dividir a área da integral em n amostras trapezoidais de bases iguais no intervalo [a, b], onde o resultado da integral dá-se pela soma das áreas dos n trapézios (Figura 7). Desta forma, o resultado de uma integral $I(f)$ dividida em N amostras é dado pela equação $I(f) \approx \frac{h}{2}\sum_{i=1}^{N}[f(x_{i+1}) + f(x_i)]$, onde x_i e x_{i+1} são os intervalos das amostras.

3 DESENVOLVIMENTO DE UM PROTÓTIPO ROBÓTICO

A proposta deste trabalho é desenvolver um protótipo de um sistema embarcado capaz de auxiliar o trabalho de busca e resgate dos bombeiros em casos de desastres. Esse protótipo baseia-se em um pequeno robô autônomo capaz de percorrer um ambiente desconhecido, desviando de obstáculos e coletando informações para gerar um mapa do local explorado. A planta baixa gerada é uma imagem gerada por uma matriz, onde cada elemento da matriz representa uma área quadrada de 900 cm² (30cmx30cm).

O primeiro passo para a execução do projeto que está sendo proposto foi a montagem do protótipo robótico. A montagem foi feita com um chassi de acrílico resistente a pequenas colisões, com duas rodas controladas por motores DC e uma roda de apoio, que apenas gira e se direciona de acordo com a movimentação do robô. Os dois motores DC são controlados com o auxílio de uma Ponte H conectada à Raspberry Pi. A conexão da Raspberry Pi com a Ponte H e os sensores foi feita com intermédio de uma protoboard posicionada na parte superior do chassi. Optou-se pelas conexões feitas através da protoboard para facilitar alterações no circuito caso necessário. A Raspberry foi alimentada com um carregador portátil de 5V (Powerbank).

3.1 Raspberry Pi

O sistema embarcado escolhido para execução deste projeto foi a placa **Raspberry Pi**. A versão da placa utilizada foi a Raspberry Pi 2B (Figura 8), que possui um processador quad-core e 1GB de memória RAM, o que possibilita a boa execução dos códigos necessários para o funcionamento do protótipo proposto. Além disso a placa possui entrada para um microSD, que é a mídia de armazenamento da placa, armazenando o sistema operacional e demais arquivos. Outras características importantes são as portas USB, utilizadas para conexão do módulo Wi-Fi, além dos pinos GPIO, onde são feitas as conexões com os sensores e motores utilizados no projeto.

Figura 8 - Raspberry Pi 2B.

Fonte: Raspberry Pi (2015).

O sistema operacional utilizado foi o Raspbian e a linguagem de programação escolhida foi o Python. Os motivos da escolha desta linguagem no projeto proposto foram: já estar instalada no sistema operacional escolhido; ser possível criar códigos em forma de scripts a serem executados; e linguagem possuir bibliotecas que facilitam a programação planejada.

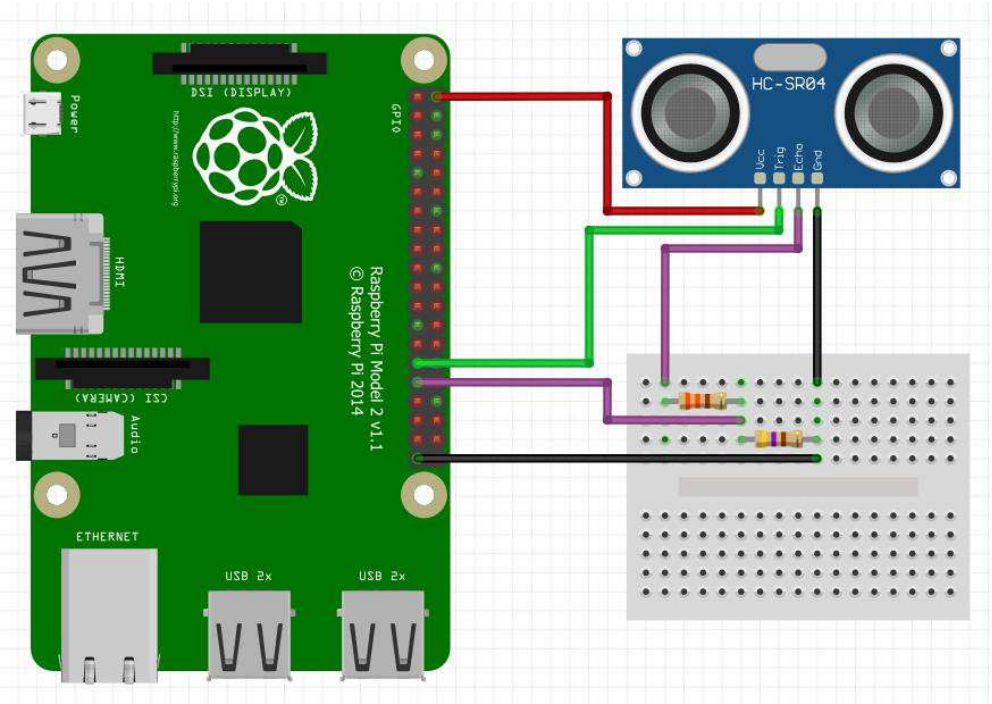
3.2 Módulos

Módulos são pequenos transdutores usados em sistemas de menores dimensões, como sistemas embarcados. O protótipo robótico desenvolvido usou um módulo ultrassônico HC-SR04, o módulo magnetômetro HCM5883L e o módulo acelerômetro ADXL345.

3.2.1 HC-SR04

O módulo sensor ultrassônico HC-SR04 é capaz de detectar objetos de 2 cm até 400 cm e é alimentado por uma tensão de 5V (CYTRON TECHNOLOGIES, p. 3), se encaixando nas necessidades e condições estabelecidas para este projeto, pois seu alcance de detecção é aceitável e sua alimentação pode ser fornecida pelo pino de saída de 5V da Raspberry Pi. Este módulo possui transmissor e receptor separados, emitindo ondas em uma frequência de 40Hz, com um ângulo de detecção de 30 graus.

Figura 9 - Conexão da Raspberry Pi com o módulo HC-SR04.



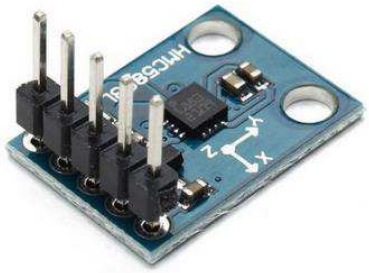
Fonte: Autor.

Dentre os pinos da Raspberry Pi nenhum recebe sinal de tensão superior à 3.3V. Uma vez que o sinal de recepção (Echo) do módulo HC-SR04 tem tensão de 5V, é preciso criar um circuito para dividir a tensão desse sinal antes de enviá-lo à Raspberry, senão a alta tensão queimará o pino da placa. Esse circuito consiste no uso de resistores de 330 ohms e 470 ohms, de forma que apenas aproximadamente 3V saía até o pino configurado para ler o sinal do receptor, sendo o restante da voltagem (2V), também inferior à 3.3V, conectado a um dos pinos terra da placa (Figura 9). Por fim o sensor foi posicionado centralizado na parte superior da frente do protótipo, a fim de detectar objetos à sua frente.

3.2.2 HCM5883L

O módulo HCM5883L (Figura 10) é projetado para captar pequenos campos magnéticos, podendo ser usado como bússola ou em aplicações de medições magnéticas. Este módulo foi escolhido para funcionar como uma bússola de auxílio para a movimentação e exploração de ambientes. O HCM5883L foi posicionado na parte superior frontal do protótipo, logo atrás do módulo ultrassônico

Figura 10 - Módulo HCM5883L.



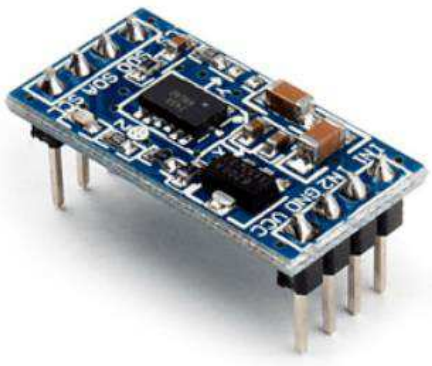
Fonte: Robu.in (2018).

O módulo foi projetado pela empresa Honeywell com base em magnetômetros magnetorresistivos, apresentando boa sensibilidade. Ele pode ser alimentado com uma voltagem entre 2.16V e 3.6V (HONEYWELL, 2013, p. 1-2), possível de ser fornecida pela Raspberry Pi. Além disso, o módulo HCM5883L utiliza a interface I2C da placa, podendo ser programado de forma mais simples com o uso da biblioteca i2c.

3.2.3 ADXL345

O módulo acelerômetro ADXL345 (Figura 11) consegue medir a aceleração nos três eixos, conseguindo medir até 16G, isto é, até aproximadamente 157 m/s². Este módulo é indicado para aplicações móveis, conseguindo medir aceleração estática e dinâmica e sendo capaz de detectar inclinações e queda livre. A alimentação aceita por este módulo está entre 2V e 3.6V e seu controle pode ser feito através da interface i2c da placa Raspberry Pi (ADXL345, 2015).

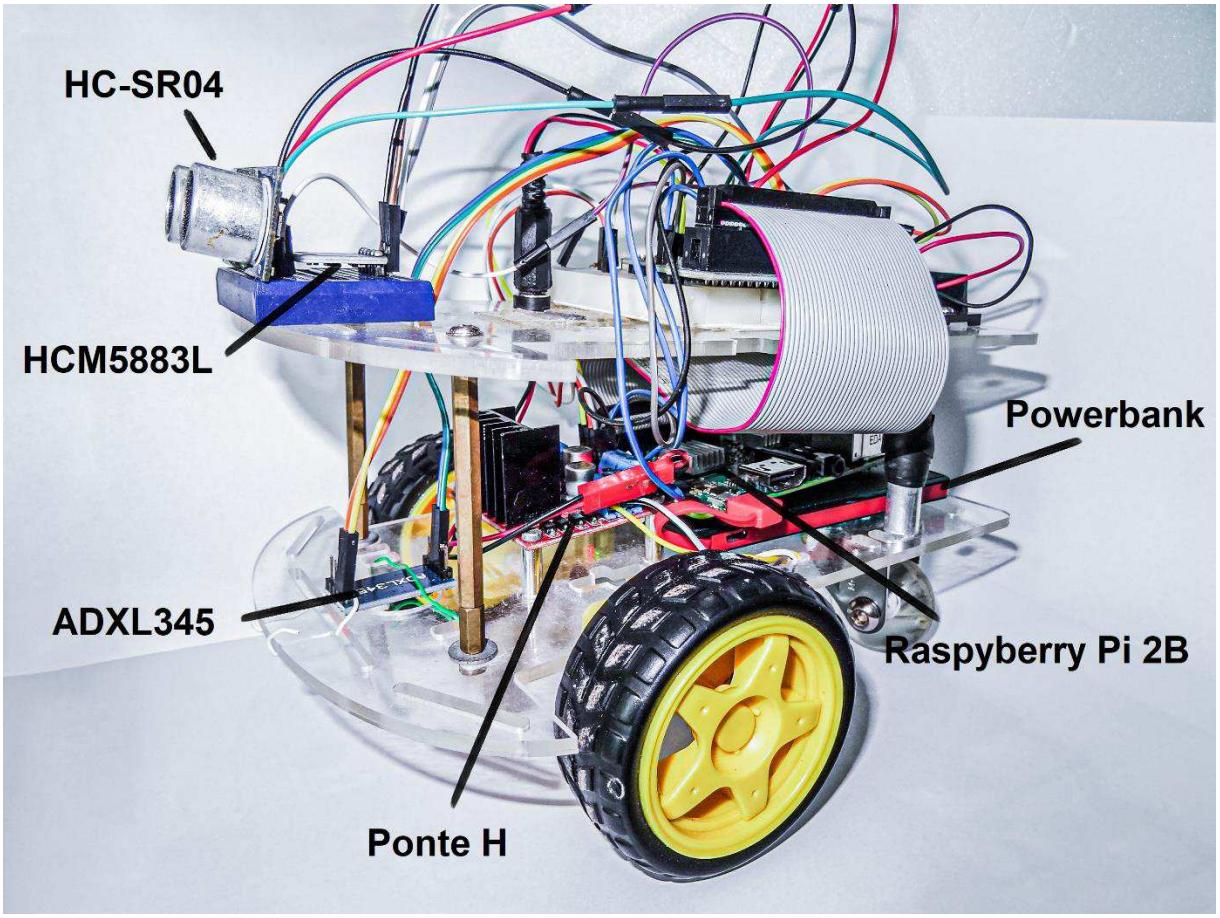
Figura 11 - Módulo ADXL345.



Fonte: Llamas (2016).

Como o protótipo tem pequeno porte e limitações de velocidade, o este módulo foi calibrado e configurado para trabalhar com medições de até 2G, equivalente a 20 m/s². O ADXL345 foi fixado na parte inferior frontal do chassi do protótipo.

Figura 12 - Protótipo robótico montado.



Fonte: Autor.

A Figura 12 mostra o robô montado, com seus principais itens destacados. Os módulos utilizados foram posicionados na parte fronteira, a ponte H entre as rodas e a Raspberry Pi na parte posterior, em cima do carregador portátil que fornece sua alimentação.

3.3 Busca por Obstáculos

A segunda parte da execução do projeto foi a programação do protótipo para explorar um ambiente desconhecido, detectando e desviando de obstáculos à sua frente. Esse processo foi feito com a programação do robô para se locomover de um ponto até outro, detectar obstáculos e selecionar uma nova direção para prosseguir, o que deve se repetir até que sejam satisfeitas determinadas condições de parada.

3.3.1 Movimentação do Robô

Com auxílio da biblioteca GPIOZERO, foi feita a programação dos motores/rodas do robô para seguir em frente, com o intuito de fazer o robô se locomover em linha reta de um ponto até outro. Devido a diferença de torque existente entre os motores, sendo o torque do motor

direito maior que o do esquerdo, o robô, ao ser colocado para se movimentar para frente, anda curvando-se para esquerda.

Para resolver este problema, a velocidade das rodas teve que ser controlada individualmente através de um PWM, que alterna a tensão de alimentação das rodas, fazendo-as rodas mais rápido ou mais devagar. Então, o robô foi reprogramado para se locomover entre dois pontos, agora baseando-se na angulação inicial em que se encontra. Isso quer dizer que o robô mede com o sensor de bússola, programado com a biblioteca i2c, para que ângulo ele está direcionado, e então segue em frente fazendo pequenas curvas alternadas entre esquerda e direita, de forma a se manter na angulação desejada.

Enquanto o robô estiver seguindo em frente, a aceleração é medida através do módulo ADXL345 e da biblioteca i2c. Essa aceleração é convertida em velocidade, e então em distância percorrida, pelo método de integração numérica dos trapézios. Por fim a distância percorrida entre os dois pontos é salva em uma lista, que será usada posteriormente para fazer o robô retornar à sua posição original.

3.3.2 Detecção de Obstáculos

A detecção de obstáculos é realizada com o módulo sensor ultrassônico e programada com auxílio da biblioteca GPIOZERO e acontece periodicamente enquanto o robô estiver seguindo em frente. Assim que um obstáculo for detectado numa determinada distância do robô, ele para de se movimentar para frente e chama um método de seleção de nova direção. A distância escolhida como condição de parada do movimento para frente foi de 30 centímetros, isso porque de acordo com a velocidade do robô, distâncias menores poderiam ocasionar colisões com o obstáculo.

3.3.3 Seleção da Direção

Quando o robô detecta um obstáculo e não pode mais seguir em frente, ele deve procurar uma nova direção para seguir e continuar explorando o ambiente desconhecido. Para isto, foi programado um método que consiste na verificação da distância até os obstáculos tanto da direita como da esquerda do robô. Esse método aciona os motores das rodas para rotacionar o robô em torno de seu eixo, movendo alguns graus para esquerda e para direita, com o objetivo de medir a distância de possíveis obstáculos com o sensor ultrassônico. As distâncias de ambos os lados são então comparadas e o robô segue para o lado de maior distância, onde será capaz de seguir

em frente por mais tempo. Sempre que o robô calcular um novo lado para seguir, a escolha do lado é salva em uma lista, usada posteriormente para gerar o mapa.

3.3.4 Condições de Parada

Se o lado escolhido pelo robô tiver um obstáculo numa distância igual ou menor que 30cm, significando que ambos os lados têm um obstáculo muito próximo do robô, ele encontra sua primeira condição de parada. Quando isso acontece, o rotaciona 180° para qualquer um dos lados e se posiciona em direção a sua posição anterior. Ele então segue em frente, mas em vez de parar ao encontrar um obstáculo, ele para quando percorrer última distância percorrida salva no vetor de deslocamento.

Ao chegar na posição anterior, o robô torna a rotacionar 180°, então ele busca na lista de lados selecionados qual foi sua última escolha e rotaciona para o lado oposto, ou seja, se anteriormente o robô escolheu virar para a esquerda, agora ele irá seguir pela direita. Se a distância deste novo lado selecionado for maior que 30cm ele inicia um novo ciclo de exploração do local, seguindo em frente e selecionando novos lados até que reencontre a condição de parada. Caso a distância do novo lado selecionado seja menor ou igual a 30cm, o robô retorna mais uma posição, e continua retornando posições até que seja possível prosseguir para o lado oposto de uma delas. Se o robô já tiver percorrido todos os lados possíveis e retornar para sua posição original, a exploração para e as informações obtidas são transferidas para o computador que fará a geração do mapa.

3.4 Mapeamento

A terceira e última etapa do projeto foi gerar um mapa do local explorado pelo robô. Aqui os dados obtidos durante a exploração são convertidos e usados para gerar um mapa em forma de uma matriz de blocos pretos e brancos, onde os blocos pretos representam obstáculos e os brancos representam um caminho livre.

3.4.1 Conversão dos Dados

Durante toda a exploração o robô armazena, em duas listas, todas as distâncias percorridas e lados escolhidos. Uma vez que o uso de ferramentas gráficas é bloqueado se a Raspberry Pi não estiver conectada a um monitor e visto que o mapa gerado é feito com ferramentas gráficas, não é possível gerar este mapa na própria Raspberry Pi. Por este motivo as duas listas são salvas em arquivos de texto e enviadas a um computador. No computador é

executado o método de geração de mapa, que inicialmente lê os dois arquivos de texto enviados pela Raspberry e os converte novamente para as duas listas, de distâncias e lados escolhidos. Essas conversões são feitas com a biblioteca numpy.

3.4.2 Geração do Mapa

O mapa gerado consiste em uma matriz, portanto para converter os movimentos do robô em uma matriz foi usada uma abordagem que considera cada posição da matriz como uma posição existente no ambiente real. Cada posição da matriz, dada por uma linha m e coluna n, representa uma área livre ou ocupada de aproximadamente 30 cm de largura e 30 cm de comprimento. Uma posição da matriz, chamada a partir de agora de bloco, será representada como um espaço livre quando tiver valor igual a 0 (zero) e como obstáculo quando seu valor for 1 (um). A área escolhida para representação dos blocos foi de 30cmx30cm por se adequar melhor ao local explorado durante a etapa de testes, podendo esse valor ser alterado para se adequar a locais maiores.

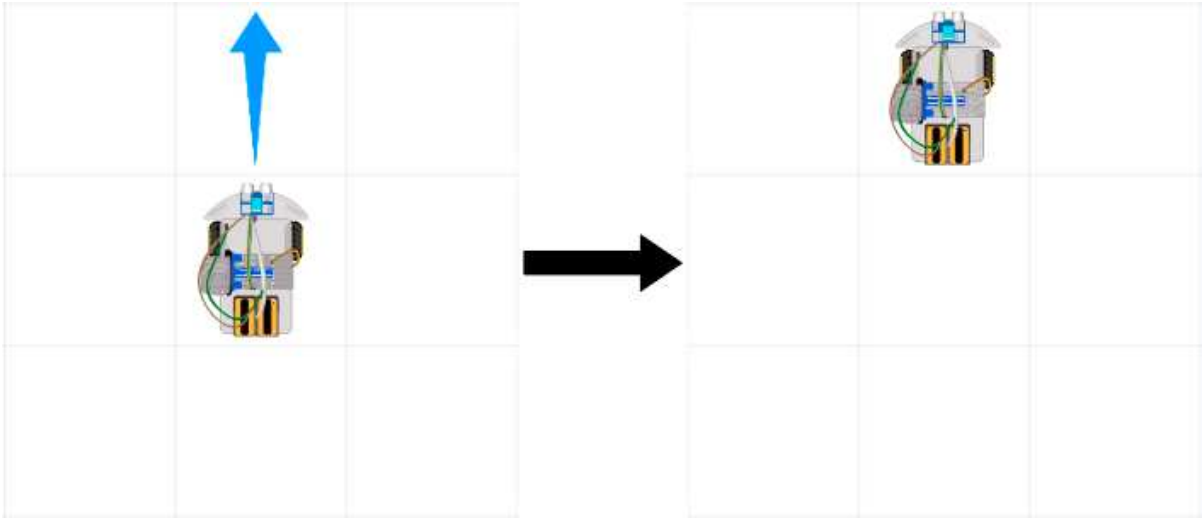
Antes de começar a percorrer os passos feitos pelo robô através da lista de distâncias e lados escolhidos, é criada uma matriz inicial que representa o robô posicionado no centro da matriz e a região que o cerca, sendo esta região inicial um conjunto de blocos livres, visto que o robô ainda não começou a busca por obstáculos, portanto não existindo obstáculos no mapa conhecido até então. Sendo assim, a matriz inicial consiste em uma matriz 3x3 (3 linhas e 3 colunas), considerando a posição inicial do robô o centro desta matriz (linha 1 e coluna 1).

Figura 13 - Matriz inicial.

Fonte: Autor.

Com a matriz inicial criada começa o processo de transcrição dos passos percorridos pelo robô para a matriz. Para entender a transcrição dos passos percorridos, é necessário visualizar a matriz do mapa como uma visão de cima para baixo do local explorado, onde o robô se encontra inicialmente no meio (Figura 13). Visto que possíveis movimentos executados pelo robô são ir para frente, e rotacionar em torno do seu próprio eixo, seja para esquerda ou para direita, três ações devem ser consideradas na montagem da matriz: subir um determinado número de blocos (Figura 14), representando o movimento para frente; rotacionar a matriz para a esquerda quando o robô girar para a direita; e rotacionar a matriz para a direita quando o robô girar para a esquerda. O movimento de rotação da matriz é feito para que a matriz sempre represente o robô visto de cima para baixo com o robô voltado para frente, a parte superior da matriz, a fim de facilitar a manipulação com os índices da matriz.

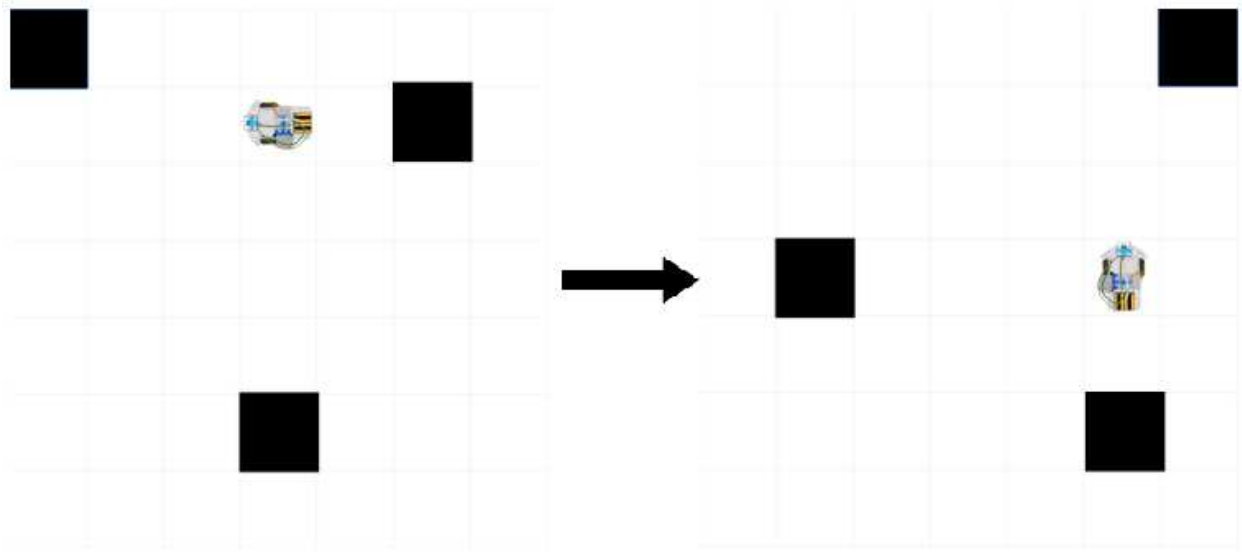
Figura 14 - Movimento para frente dentro da matriz.



Fonte: Autor.

O processo de transcrição começa com o cálculo de quantos blocos serão percorridos na matriz. Este cálculo consiste em pegar o valor da distância percorrida, encontrada na lista de distâncias, e dividir pelo tamanho representado por cada bloco, neste caso um quadrado de 30cm de lado. Desta forma, supondo que o robô se locomove 95 cm, na matriz serão percorridos 3 blocos e caso o robô se desloque menos que 30 cm, será considerada a locomoção de um bloco na matriz.

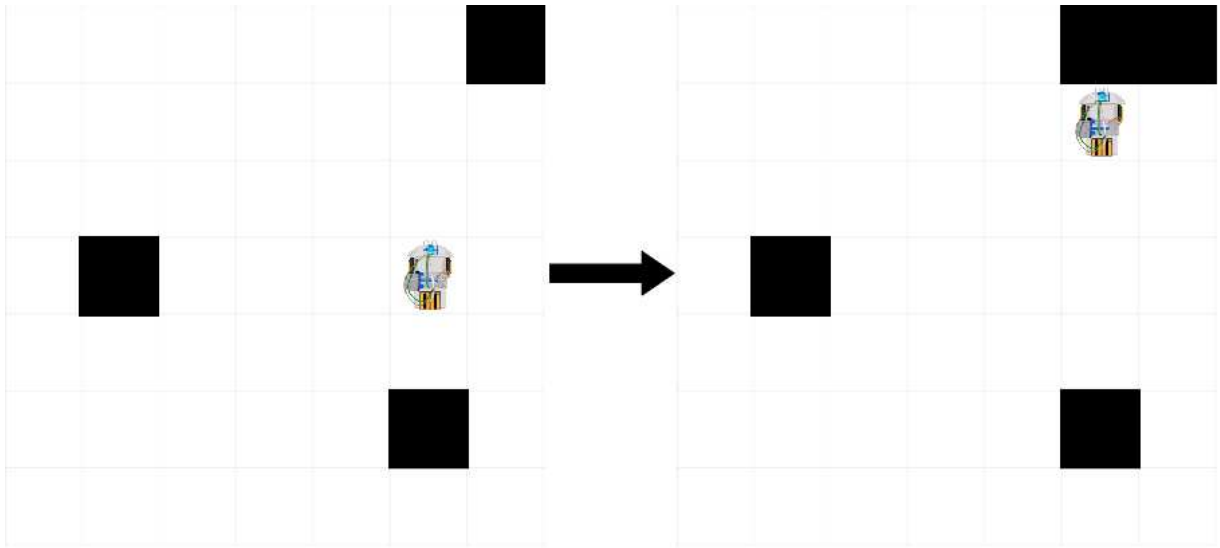
Figura 15 - Rotação para esquerda.



Fonte: Autor

Sabendo quantos blocos serão percorridos, é verificado para qual lado o robô escolheu se locomover essa distância, uma informação que consta na lista de lados escolhidos. Se o primeiro lado escolhido pelo protótipo robótico foi o esquerdo, a matriz será rotacionada para a direita (Figura 15) e então andará a quantidade de blocos calculada, logo sua nova posição será a subtração da quantidade de blocos percorridos da linha atual em que se encontra. Por exemplo, se o robô se encontra na terceira linha da matriz e deve andar dois blocos, sua nova posição na matriz será a linha 1 (um) da mesma coluna e o bloco a sua frente será marcado como um obstáculo (Figura 16).

Figura 16 - Exemplo de movimentação do robô dentro da matriz.

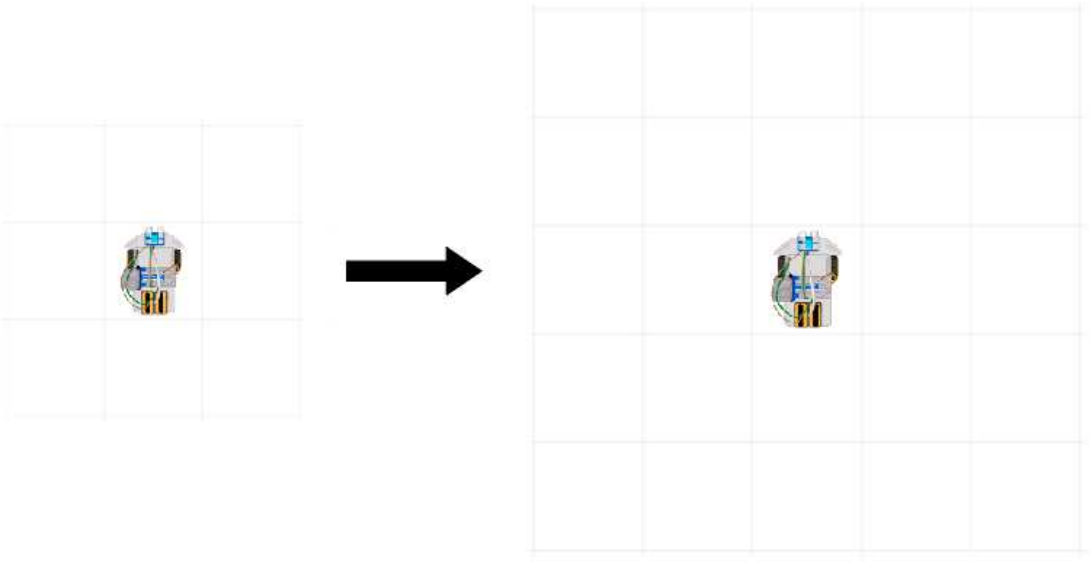


Fonte: Autor.

Em alguns casos a nova posição do robô na matriz será uma posição fora dos limites da matriz. Essa condição irá acontecer pela primeira vez no início da transcrição, quando mesmo

que o robô só precise andar um bloco, não haverá espaço na matriz para marcar o bloco que representa o obstáculo encontrado.

Figura 17 - Aumento da matriz.



Fonte: Autor.

Quando isso acontece o tamanho da matriz é aumentado baseado na adição de linhas e colunas de zeros ao redor da matriz existente, simbolizado um aumento na área conhecida até então (Figura 17). Sempre que o robô efetuar uma rotação e andar um determinado número de blocos, será marcado o bloco em sequência como obstáculo encontrado.

3.5 Principais Métodos Desenvolvidos

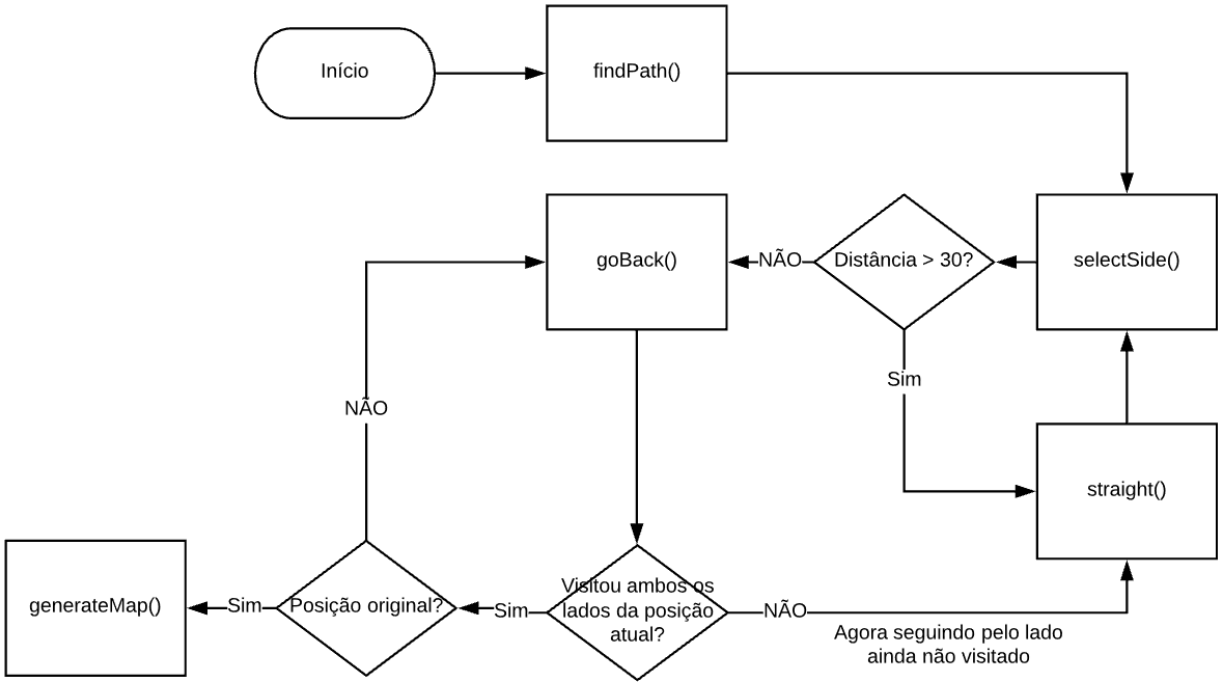
Todo o processo se dá pela chamada de cinco métodos principais, encontrados na seção de apêndices deste documento. Esses cinco métodos são:

- selectSide() – Método que faz a seleção da direção explicada no subtópico 3.3.3 deste documento. Onde é chamado o método de movimentação das rodas para esquerda, feita uma leitura com o módulo ultrassônico e com o módulo magnetômetro, chamado o método de movimentação das rodas para direita e feita novamente a leitura com os dois sensores. As duas leituras do sensor ultrassônico são comparadas, se a leitura da esquerda for de maior valor, o método retorna o ângulo medido pelo magnetômetro na esquerda e a string ‘Left”, caso contrário o método retorna o ângulo medido pelo magnetômetro na direita e a string “Right”;
- straight() – Este é o método que faz o robô se mover em frente até que um obstáculo seja detectado, explicado no subtópico 3.3.1 do documento. O método recebe como parâmetro o ângulo da direção, inicia um laço de repetição que

chama o método de movimentação das rodas responsável pelo movimento para frente e faz a leitura do módulo sensor ultrassônico após andar. Ainda dentro do laço é feita a leitura da aceleração do robô durante o movimento para frente em função dos eixos X e Y. É calculada então a resultante dessa aceleração e em seguida é feito o cálculo da velocidade e do deslocamento efetuado, através dessa aceleração. Quando o laço se quebra, com a leitura do sensor ultrassônico sendo menor que 30, a função que para os motores é chamada o método retorna o ângulo da direção que ele seguiu e a distância percorrida calculada;

- findPath() – É o primeiro método chamado pela execução do programa. Ele chama dentro de si os métodos selectSide() e straight(). Dentro de um laço de repetição é chamado o método selectSide(), que retorna o ângulo e o lado escolhido. Uma leitura no sensor ultrassônico é feita para verificar se o lado escolhido tem um obstáculo logo em frente, se sim ele quebra o laço, se não ele chama o método straight();
- goBack() – Método chamada após o findPath(). Ele recebe como parâmetros uma distância e uma angulação e inicia um laço de repetição que chama o método de movimentação das rodas para frente e faz a leitura do módulo sensor acelerômetro. Da aceleração calculada, nos eixos X e Y, é calculado o módulo da aceleração resultante e em sequência são calculadas a velocidade e deslocamento efetuado durante o movimento para frente. Quando o deslocamento encontrado for maior que o passado como parâmetro pelo método, o laço de repetição é quebrado e a função de parar os motores é chamada;
- generateMap() – Esse é o método de geração do mapa abordado no subtópico 3.4.2. Aqui uma matriz de zeros é inicializada e inicia-se um laço de repetição que irá aumentar o tamanho da matriz caso seja necessário, rotacionar a matriz e atualizar a posição atual do robô na matriz com a mudança dos índices na matriz.

Figura 18 - Fluxograma de correlação entre os principais métodos desenvolvidos.



Fonte: Autor.

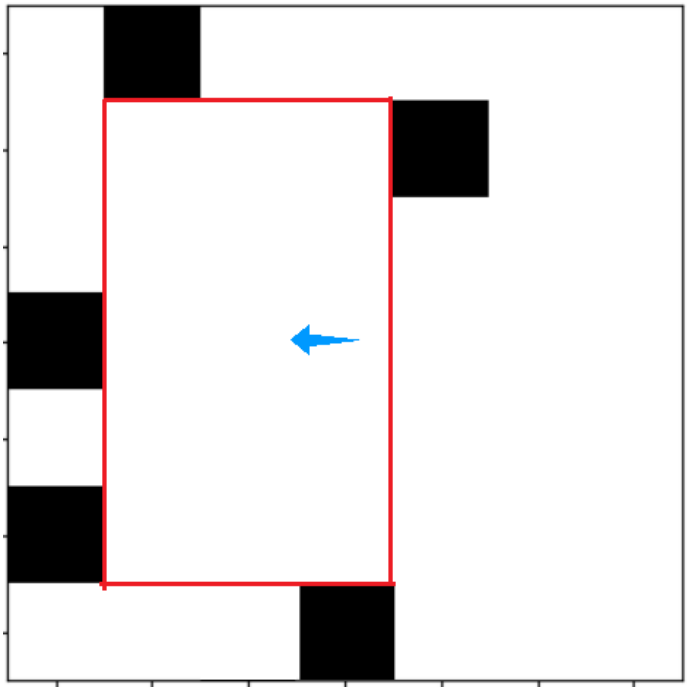
O fluxograma mostrado na Figura 18 apresenta uma correlação entre esses principais métodos. Na função principal do código desenvolvido o método findPath() é chamado. Então um lado é selecionado pelo selectSide() e, se não houver um objeto a menos de 30cm do robô, é chamado o método straight(), caso contrário o método goBack(). Se ambos os lados da posição atual já tiverem sido visitados é verificado se o robô está na posição onde iniciou a exploração e, caso esteja, o método generateMap() é chamado.

4 RESULTADOS E DISCUSSÕES

4.1 Cenários Testados

Uma vez concluída a montagem e programação do protótipo, foi iniciada a etapa de testes, a fim de validar sua eficiência. Foram montados alguns cenários para serem explorados pelo robô, todos com tamanho definido e sem caminhos que levassem o robô a sair da área de exploração desejada. Foram montados dois tipos de cenários: com obstáculos apenas na delimitação da área a ser explorada; e com obstáculos das delimitações e dentro da área a ser explorada.

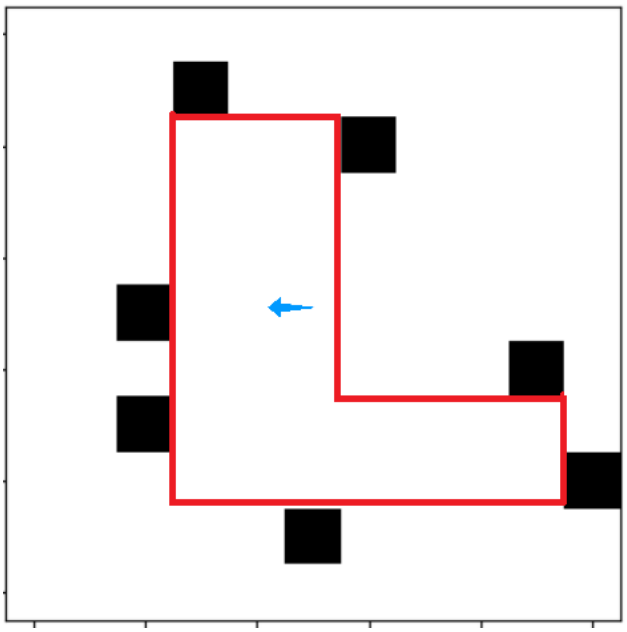
Figura 19 - Mapa gerado em um local retangular sem obstáculos.



Fonte: Autor.

O primeiro cenário testado foi uma área retangular com 2m de comprimento e 1m de largura, sem nenhum obstáculo dentro. Como um obstáculo só é inserido no mapa se ele for encontrado pelo robô enquanto ele anda para frente, neste cenário foram encontrados apenas cinco obstáculos diferentes, o obstáculo logo em frente e os demais cada um em um dos cantos da área retangular. A Figura 19 mostra o mapa gerado para este cenário, onde a linha vermelha marca a delimitação da área e a seta azul marca a posição e direção inicial do robô.

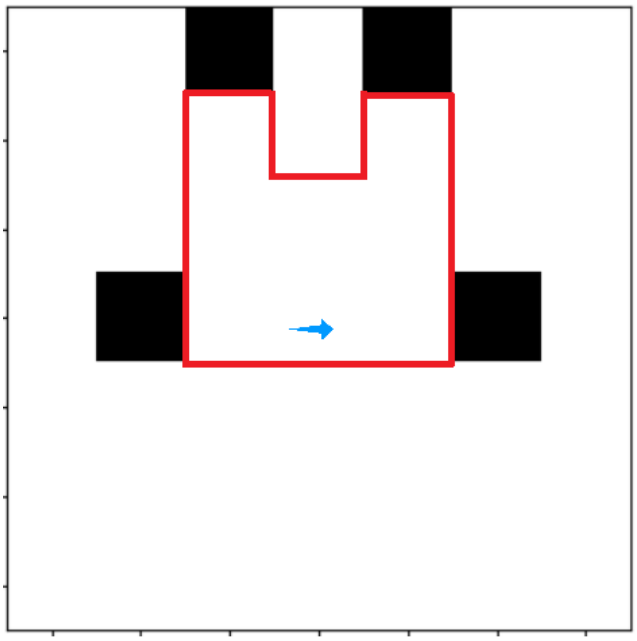
Figura 20 - Mapa gerado em uma área de formato "L".



Fonte: Autor.

O segundo cenário testado foi uma área em formato de “L”, onde o protótipo também foi capaz de mapear alguns pontos existentes no cenário (Figura 20), encontrando cinco obstáculos distintos durante a exploração do ambiente. Tanto no cenário retangular como neste, a condição de parada foi um número limite de iterações.

Figura 21 – Mapa de área quadrada com parede no meio.

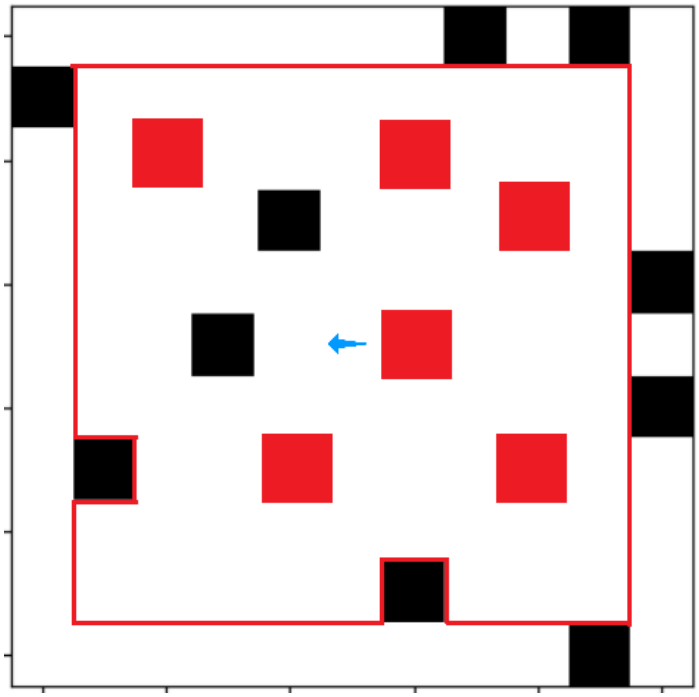


Fonte: Autor.

O próximo cenário testado foi uma área quadrada com uma parede dividindo metade da área (Figura 21). A parede foi posicionada de forma que o robô não tivesse como ir para os lados,

tendo que retornar à posição anterior, repetindo o processo para ambos os lados e encontrando quatro obstáculos no ambiente explorado.

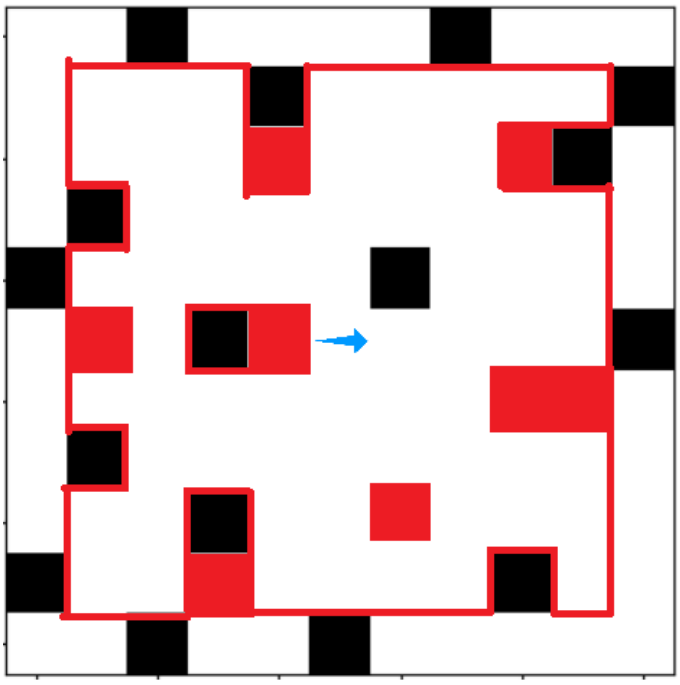
Figura 22 - Mapa de área quadrada com alguns obstáculos.



Fonte: Autor.

O cenário seguinte trata-se de uma região quadrada com alguns obstáculos quadrados espalhados pelo local, onde os blocos vermelhos representam os obstáculos não encontrados (Figura 22). Neste caso somente uma pequena parcela do mapa real foi encontrado.

Figura 23 - Mapa de área quadrada com obstáculos juntos à parede.



Fonte: Autor.

Percebendo que o protótipo tende se se posicionar próximo às paredes no cenário, foi montado um último cenário, também retangular, porém com mais obstáculos junto aos lados de limitação da área (Figura 23). Aqui houve uma detecção de 50% dos obstáculos dentro da área.

4.2 Limitações

A alimentação dos motores foi baseada em baterias de 9V. Quando a bateria está com a carga completa ela proporciona a rotação máxima para os motores, porém após alguns minutos de rotação, com o descarregar da bateria, a voltagem de alimentação diminui, reduzindo a rotação dos motores. Esse fornecimento de voltagem não constante para os motores tornou difícil a calibragem do torque das rodas com a técnica de PWM, de forma que não foi possível fazer uma sincronização precisa entre os dois motores.

Devido a variação de voltagem na alimentação dos motores que controlam as rodas, a velocidade das rodas não se mantém constante. Por isso, tornou-se necessário calibrar regularmente algumas variáveis de movimentação do protótipo, como o tempo em que os motores devem permanecer girando, para que o robô se movimentasse da maneira como foi projetado.

Na etapa de testes foi observado que o protótipo tende a se locomover em direção às paredes que limitam a área explorada. Se o robô estiver explorando uma área retangular sem obstáculos junto às paredes, por conseguir detectar apenas obstáculos à sua frente, ele encontrará apenas quatro obstáculos, que serão os cantos da área explorada. Nesta situação ele ficará encontrando os mesmos obstáculos até atinja um número máximo de iterações e, uma vez que a velocidade do protótipo tende a diminuir ao longo do descarregar da bateria que o alimenta, o protótipo começaria a fazer rotações com ângulos menores que 90°, implicando em grandes inconsistências no mapa gerado.

Durante a exploração do ambiente o protótipo percorre um local até que não haja um lado livre para escolher, então ele volta uma posição e continua explorando o local. Em alguns cenários o robô não encontra essa condição e fica andando em círculos pelo ambiente, até que atinja um número máximo de ações estipulados na programação. Neste caso, como ele fica identificando o mesmo obstáculo várias vezes, o mapa gerado apresenta inconsistências com o local real explorado. Esta situação ocorre por não ter sido implementado uma forma de identificar os locais já percorridos pelo robô.

5 CONCLUSÕES E TRABALHOS FUTUROS

O projeto proposto foi a criação de um protótipo robótico autônomo capaz de mapear uma área desconhecida, a fim de auxiliar no processo de evacuação e resgate de um local após a concretização de um desastre.

Foi montado um sistema embarcado em forma de um pequeno robô capaz de se locomover por um local e extrair informações dele, gerando um mapa. Esse robô foi controlado por uma Raspberry Pi e baseou-se na leitura dos sensores ultrassônico, acelerômetro e magnetômetro para percorrer e mapear o local. Todos os periféricos foram programados corretamente, entretanto não foi possível fazer uma leitura correta em alguns casos, por interferências no sinal da leitura.

O algoritmo desenvolvido conseguiu efetuar o deslocamento do protótipo entre dois pontos de maneira aceitável. Esse algoritmo fez a detecção de todos os obstáculos encontrados na frente do robô durante o movimento para frente. E o algoritmo criado para geração do mapa consegue gerar um mapa exato do local explorado, desde que sejam passados todos os obstáculos existentes no local.

Dentre os testes efetuados sob condições favoráveis com relação às limitações encontradas ao longo do projeto, percebeu-se que o protótipo consegue encontrar partes do mapa real percorrido. Ainda que o protótipo desenvolvido não gere um mapa completo do local, ele detecta e marca corretamente os obstáculos encontrados a sua frente, executando corretamente a tarefa proposta.

Sendo assim foi possível concluir que o protótipo robótico desenvolvido pode ser utilizado para auxílio em situações de evacuação em edifícios após a ocorrência de um desastre. Esta conclusão dá-se pelo robô ter sido capaz de percorrer um local desconhecido, coletando dados necessários para a gerar um mapa desse local. Este mapa, mesmo que não represente o local explorado fidedignamente, apresenta características reais do local, servindo como uma base a ser utilizada por vítimas e equipes de resgate durante um dado desastre.

No futuro propõe-se acoplar sensores ultrassônicos nas laterais do robô, para que seja possível detectar obstáculos laterais, aumentando a precisão dos mapas gerados pelo protótipo. Além disso, pode ser desenvolvida uma forma de identificar os locais já percorridos pelo robô, excluindo a possibilidade de detecção do mesmo obstáculo várias vezes, e garantindo que mais áreas serão exploradas.

REFERÊNCIAS

[1] DEFESA CIVIL. O que é um desastre?. **Coordenadoria Estadual da Defesa Civil**. 2018. Curitiba. Disponível em: http://www.defesacivil.pr.gov.br/Pagina/O-que-e-um-desastre. Acesso em: 17 Jul. 2019.

[2] BRASIL. Ministério da Integração Nacional. **Política Nacional de Defesa Civil**. Brasília: Secretaria Nacional de Defesa Civil, 2007. 82p. Disponível em: <http://www.defesacivil.gov.br/publicacoes/publicacoes/pndc.asp>. Acesso em: 17 Jul. 2019.

[3] DEFESA CIVIL. **Manual de Gerenciamento de Desastres**. 2009. 74 p. Disponível em:http://www.ceped.ufsc.br/wp-content/uploads/2014/09/Manual-de-Gerenciamento-de-Desastres.pdf. Acesso em: 17 Jul. 2019.

[4] ARAÚJO, Sérgio B.. **Administração de Desastres**: Conceitos e Tecnologias. 3. ed. 2011. Disponível em: http://www.defesacivil.pr.gov.br/sites/defesa-civil/arquivos_restritos/files/documento/2018-12/AdministracaodeDesastres.pdf. Acesso em: 17 Jul. 2019.

[5] GARCIA, Fernando Deluno. Introdução aos sistemas embarcados e microcontroladores. **Embarcados**. 2018. Disponível em: https://www.embarcados.com.br/sistemas-embarcados-e-microcontroladores/. Acesso em: 17 Jul. 2019.

[6] SILVEIRA, Cristiano Bertulucci. 10 Aplicações Para o Sensor Ultrassonico na Indústria. **Citisystems**. Disponível em: https://www.citisystems.com.br/sensor-ultrassonico/. Acesso em: 18 Jul. 2019.

[7] SONAR. **Wikipédia**. 2010. Disponível em: https://pt.wikipedia.org/wiki/Sonar#/media/Ficheiro:Sonar_Principle_pt-BR.svg. Acesso em: 18 Jul. 2019.

[8] REZENDE, Sérgio M.. **Materiais e Dispositivos Eletrônicos**. 2. ed. São Paulo: Livraria da Física, 2004.

[9] PETRUZELLA, Frank D.. **Motores Elétricos e Acionamentos**. Tradução José Lucimar do Nascimento. 1. ed. São Paulo: AMGH, 2013. Tradução de: Electric Motors and Control Systems. Disponível em: https://books.google.com.br/books?id=4xw4AgAAQBAJ&pg=PA105&redir_esc=y#v=onepage&q&f=false. Acesso em: 18 Jul. 2019.

[10] RASPBERRY Pi 2 Model B. **Raspberry Pi**. Disponível em: https://www.raspberrypi.org/products/raspberry-pi-2-model-b/. Acesso em: 19 Jul. 2019.

[11] HUNTER, J. D.. Matplotlib: A 2D graphics environment. **Computing in Science \& Engineering**, IEEE COMPUTER SOC, p. 90--95, 2007.

[12] VALDEZ, Jonathan; BECKER, Jared. **Understanding the I2C Bus**. 2015. 8 p. Disponível em: http://www.ti.com/lit/an/slva704/slva704.pdf. Acesso em: 19 Jul. 2019.

[13] NUTTALL, Ben. **Gpiozero Documentation**. 2019. 239 p. Disponível em: https://buildmedia.readthedocs.org/media/pdf/gpiozero/stable/gpiozero.pdf. Acesso em:19 Jul. 2019.

[14] ROCHA, Fábio Saraiva; MARRANGHELLO, Guilherme Frederico. **Propriedades de um acelerômetro eletrônico e possibilidades de uso no ensino da mecânica**. 2013. 10 p. Disponível em:http://www.lajpe.org/march13/6_LAJPE_739_Fabio_Saraiva_preprint_corr_f.pdf. Acesso em:19 Jul. 2019.

[15] CYTRON TECHNOLOGIES. **User's Manual**. 10 p. Disponível em: https://datasheetspdf.com/pdf-file/1291829/Cytron/HC-SR04/1. Acesso em: 19 Jul. 2019.

[16] MAGNETOMETER - the History. **CT Systems**. 2009. Disponível em:https://web.archive.org/web/20070930141937/http://www.ctsystems.eu/gauss.htm. Acesso em: 19 Jul. 2019.

[17] SELIGMAN, Luiza. **Eletrodeposição de Multicamadas Metálicas em Silício**. Florianópolis, f. 118. 108 p. Dissertação (Engenharia Mecânica) - UNIVERSIDADE FEDERAL DE SANTA CATARINA, 2002. Disponível em: https://core.ac.uk/download/pdf/30364472.pdf. Acesso em: 20 Jul. 2019.

[18] HONEYWELL. **3-Axis Digital Compass IC HMC5883L**. 2013. Disponível em: https://cdn-shop.adafruit.com/datasheets/HMC5883L_3-Axis_Digital_Compass_IC.pdf. Acesso em: 20 Jul. 2019.

[19] GY-271 HMC5883L 3-axis Electronic Compass Module Magnetic Field Sensor – Original Chip. **Robu.In**. 2018. Disponível em: https://robu.in/product/gy-271-hmc5883l-3-axis-electronic-compass-module-magnetic-field-sensor/. Acesso em: 20 Jul. 2019.

[20] ADXL345. **Analog**. 2015. 40 p. Disponível em: https://www.analog.com/media/en/technical-documentation/data-sheets/ADXL345.pdf. Acesso em: 20 Jul. 2019.

[21] LLAMAS, Luis. **Usar un acelerómetro ADXL345 con Arduino**. 2016. Disponível em:https://www.luisllamas.es/arduino-acelerometro-adxl345/. Acesso em: 20 Jul. 2019.

[22] BARR. Introduction to Pulse Width Modulation (PWM). **Barr Group**. 2001. Disponível em:https://barrgroup.com/Embedded-Systems/How-To/PWM-Pulse-Width-Modulation. Acesso em: 22 Jul. 2019.

[23] GILAT, Amos; SUBRAMANIAM, Vish. **Métodos Numéricos para Engenheiros e Cientistas**: Uma introdução com aplicações usando MATLAB. Tradução Alberto Rezende. Porto Alegre: Bookman, 2008. Disponível em: https://issuu.com/menezsezs/docs/m_todos_num_ricos_para_engenheiros_. Acesso em: 22 Jul. 2019.

[24] TOASSI, Andresa Jaqueline; STOLF, Michele Caroline; OLIVEIRA, Micheline Ramos de. **Inserção tecnológica no trabalho: etnografia das significações profissionais de bombeiros**. 2006. Pepsic. Santa Catarina. Disponível em: http://pepsic.bvsalud.org/scielo.php?script=sci_arttext&pid=S1414-98932006000200010. Acesso em: 23 Jul. 2019.

[25] ECKELBERG, Rudolf Copi. **DESENVOLVIMENTO DE UM ACELERÔMETRO DE BOLHA DE SENSIBILIDADE VARIÁVEL**. Curitiba, f. 87, 2013. Dissertação (Engenharia e Ciência dos Materiais) - UNIVERSIDADE FEDERAL DO PARANÁ. Disponível em:http://www.pipe.ufpr.br/portal/defesas/dissertacao/256.pdf. Acesso em: 23 Jul. 2019.

[26] BALTZAN, Paige; PHILLIPS, Amy. **Sistemas de Informação**. Tradução Rodrigo Dubal. 1. ed.Porto Alegre: AMGH, 2012. Tradução de: Information Systems. Disponível em:https://books.google.com.br/books?id=NJkR83DSkPYC&pg=PA39&lpg=PA39&dq=IA+em+robo+bombeiro&source=bl&ots=G4NsvINReU&sig=ACfU3U0zotR8gRSUDysELGZVAzMDBd-nGw&hl=pt-BR&sa=X&ved=2ahUKEwj08r_4vc7jAhUBuVkKHWN9CRU4ChDoATABegQICBAB#v=onepage&q=IA%20em%20robo%20bombeiro&f=false. Acesso em: 24 Jul. 2019.

[27] PAZOS, Fernando. **Automação de Sistemas e Robótica.** Rio de Janeiro: Axcel Books, 2002. Disponível em: https://issuu.com/blogengprod/docs/automa____o_de_sistemas_e_rob__tica. Acesso em: 24 Jul. 2019.

APÊNDICE A – CÓDIGO DO MÉTODO SELECTSIDE()

```
# Seleciona um lado para virar (esquerda ou direita)
def selectSide():

    # Verifica a distância para o possível obstáculo na esquerda
    motors.turnLeft(0.6)            # Vira para esquerda
    angleLeft = compass.getAngle()  # Lê a direção atual
    #Lê a distância até o obstáculo na esquerda
    distanceLeft = ultrasonicSensor.distance*100

    # Retorna para a direção inicial
    motors.turnRight(0.5)

    # Verifica a distância para o possível obstáculo na direita
    motors.turnRight(0.5)           # Vira para a direita
    angleRight = compass.getAngle() # Lê a direção atual
    #Lê a distância até o obstáculo na direita
    distanceRight = ultrasonicSensor.distance*100

    # Retorna para a direção inicial
    motors.turnLeft(0.6)

    # Seleciona o lado de maior distância
    if distanceLeft > distanceRight:
        motors.turnLeft(0.6)    # Vira para esquerda
        # Retorna o ângulo da esquerda (angleLeft) e o lado escolhido (Left)
        return angleLeft, 'Left'
    else:
        motors.turnRight(0.5)   # Vira para direita
        # Retorna o ângulo da direita (angleRight) e o lado escolhido (Right)
        return angleRight, 'Right'
```

APÊNDICE B – CÓDIGO DO MÉTODO STRAIGHT()

```
# Segue em frente e detecta obstáculos
def straight(angle):

    speedLeft = 0.6      # Porcentagem inicial da rotação do motor esquerdo
    speedRight = 0.4     # Porcentagem inicial da rotação do motor direito
    DeltaS = 0           # Deslocamento total inicial igual a 0
    minDistance = 30     # Distância mínima aceitada até um obstáculo
    A0 = 0               # Aceleração inicial igual a 0
    V0 = 0               # Velocidade inicial igual a 0

    distance = ultrasonicSensor.distance*100
    while distance > minDistance:

        start = time.time()
        # Movimenta o robô para frente de acordo com a porcentagem de rotação
        (speedLeft, speedRight) = motors.moveForward(angle, speedLeft, speedRight)
        end = time.time()

        # Lê a nova distância até o obstáculo a frente
        distance = ultrasonicSensor.distance*100

        # Mede e a aceleração resultante entre o ponto anterior e o atual
        (Ax, Ay) = accelerationSensor.getAcceleration()
        A = math.sqrt(Ax * Ax + Ay * Ay)

        t = end - start      # Calcula o tempo de locomoção para frente
        # Calcula a velocidade e o deslocamento entre o ponto atual e o anterior
        V = (A0 + A) * t / 2
        S = (V + V0) * t / 2

        A0 = A           # Atualiza o valor inicial da aceleração
        V0 = V           # Atualiza o valor inicial da velocidade
        DeltaS += S*100 # Soma o deslocamento atual com os anteriores convertendo
para cm

    motors.stop()    # Para a rotação das rodas

    # Retorna a direção do movimento (angle) e o deslocamento total(DeltaS)
    return angle, round(DeltaS, 2)
```

APÊNDICE C – CÓDIGO DO MÉTODO FINDPATH()

```
# Inicia a exploração do ambiente
def findPath():
    i = 0
    while i < 10:
        i += 1      # Contador de iterações

        # Chama o método de seleção de lado e recebe o ângulo e lado escolhido
        (angle, side) = selectSide()

        # Verifica se o lado escolhido tem um obstáculo a menos de 30cm
        if ultrasonicSensor.distance * 100 < 30:
            break

        # Insere o lado escolhido em uma pilha
        sideSelected.append(side)
        # Insere o lado escolhido na lista de todos os lados
        # escolhidos durante a exploração do local
        fullsideSelected.append(side)

        (direction, distance) = straight(angle) # Segue em frente na angulação dada

        # Insere o ângulo atual em uma pilha
        directionVector.append(direction)
        # Insere a distância percorrida em uma pilha
        distanceVector.append(distance)
        # Insere o deslocamento na lista de todos deslocamentos
        # efetuados durante a exploração do local
        fulldistanceVector.append(distance)
```

APÊNDICE D – CÓDIGO DO MÉTODO GOBACK()

```
# Retorna uma posição seguindo em frente por
# uma distância (distance) na angulação (angle)
def goBack(distance, angle):

    speedLeft = 0.6      # Porcentagem inicial da rotação do motor esquerdo
    speedRight = 0.4     # Porcentagem inicial da rotação do motor direito
    DeltaS = 0           # Deslocamento total inicial igual a 0
    A0 = 0               # Aceleração inicial igual a 0
    V0 = 0               # Velocidade inicial igual a 0

    # Segue em frente enquanto o deslocamento atual for menor que a distância
determinada
    while distance > DeltaS:

        start = time.time()
        # Movimenta o robô para frente de acordo com a porcentagem de rotação
        (speedLeft, speedRight) = motors.moveForward(angle, speedLeft, speedRight)
        end = time.time()

        # Mede e a aceleração resultante entre o ponto anterior e o atual
        (Ax, Ay) = accelerationSensor.getAcceleration()
        A = math.sqrt(Ax * Ax + Ay * Ay)

        t = end - start         # Calcula o tempo de locomoção para frente
        # Calcula a velocidade e o deslocamento entre o ponto atual e o anterior
        V = (A0 + A) * t / 2
        S = (V + V0) * t / 2

        A0 = A  # Atualiza o valor inicial da aceleração
        V0 = V  # Atualiza o valor inicial da velocidade
        DeltaS += S * 100  # Soma os deslocamentos e converte para cm

    motors.stop()   # Para a rotação das rodas
    # Adiciona o deslocamento à lista de todos os deslocamentos efetuados
    fulldistanceVector.append(round(DeltaS, 2))
```

APÊNDICE E – CÓDIGO DO MÉTODO GENERATEMAP()

```
def generateMap(self):

    m = 1    # Índice atual na linha
    n = 1    # Índice atual na coluna
    # Inicia matriz de zeros 3x3
    blueprint = np.zeros((3, 3))

    for i in range(0, len(sideSelected)):

        # Calcula o número de blocos a ser percorrido
        steps = round(int(distanceVector[i]) / blockSize)

        # Redimenciona a matriz caso necessário
        if m - steps - 1 < 0:
            # Calcula quantas linhas e colunas serão adicionadas
            increase = abs(m - steps - 1)
            # Aumenta o tamanho da matriz
            blueprint = np.pad(blueprint, pad_width=increase,
                               mode='constant', constant_values=0)
            m += increase   # Atualiza o índice atual da linha
            n += increase   # Atualiza o índice atual da coluna

        # Mede o tamanho da matriz
        (mSize, nSize) = np.shape(blueprint)
        indexRange = mSize - 1

        # Rotaciona as matrizes
        if sideSelected[i] == 'Left':
            # Rotaciona a matriz para direita
            blueprint = np.rot90(blueprint, 3)
            # Atualiza a posição atual na matriz
            aux = m
            m = n
            n = indexRange - aux
        elif sideSelected[i] == 'Right':
            # Rotaciona a matriz para a esquerda
            blueprint = np.rot90(blueprint)
            # Atualiza a posição atual na matriz
            aux = m
            m = indexRange - n
            n = aux
        else:
            # Rotaciona a matriz duas vezes para a esquerda
            blueprint = np.rot90(blueprint, 2)
            # Atualiza a posição atual na matriz
            m = indexRange - m
            n = indexRange - n
            # Percorre os blocos
            m -= steps
            continue
        # Percorre os blocos
        m -= steps
        # Marca os obstáculos na matriz
        blueprint[m - 1, n] = 1

    # Gera uma imagem do mapa criado
    fig, ax = pplot.subplots(1)
    pplot.imshow(blueprint, cmap="binary")
    pplot.suptitle('Mapa Gerado')
    ax.set_yticklabels([])
    ax.set_xticklabels([])
    # Salva a imagem como um arquivo .png
    fig.savefig('Blueprint.png')
    pplot.show()
```