MEC-SETEC
INSTITUTO FEDERAL MINAS GERAIS - *Campus* Formiga
Curso de Ciência da Computação

DESENVOLVIMENTO DE UM PROTÓTIPO PARA VEÍCULOS AUTÔNOMOS COM LOCALIZAÇÃO E MAPEAMENTO SIMULTÂNEOS

Pedro Henrique Oliveira Veloso

Orientador: Prof. Dr. Otávio de Souza Martins Gomes

Formiga - MG

2019

PEDRO HENRIQUE OLIVEIRA VELOSO

DESENVOLVIMENTO DE UM PROTÓTIPO PARA VEÍCULOS AUTÔNOMOS COM LOCALIZAÇÃO E MAPEAMENTO SIMULTÂNEOS

Trabalho de Conclusão de Curso apresentado ao Instituto Federal Minas Gerais - *Campus* Formiga, como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Otávio de Souza Martins Gomes

Formiga - MG

2019

Veloso, Pedro Henrique Oliveira.
004 Desenvolvimento de um Protótipo para Veículos Autônomos com Localização e Mapeamento Simultâneos / Pedro Henrique Oliveira Veloso. -- Formiga : IFMG, 2019.
84p. : il.

Orientador: Prof. Dr. Otávio de Souza Martins Gomes
Trabalho de Conclusão de Curso – Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais – *Campus* Formiga.

1. SLAM. 2. Navegação de Robôs. 3. Mapeamento. 4. Localização. . I. Título.

CDD 004

Ficha catalográfica elaborada pela Bibliotecária Msc. Naliana Dias Leandro CRB6-1347

PEDRO HENRIQUE OLIVEIRA VELOSO

DESENVOLVIMENTO DE UM PROTÓTIPO PARA VEÍCULOS AUTÔNOMOS COM LOCALIZAÇÃO E MAPEAMENTO SIMULTÂNEOS

Trabalho de Conclusão de Curso apresentado ao Instituto Federal de Minas Gerais - Campus Formiga, como Requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Aprovado em: **14 de Junho de 2019**.

BANCA EXAMINADORA

Prof.º Otávio de Souza Martins Gomes - Orientador

Prof.º Diego Mello da Silva

Prof.º Walace de Almeida Rodrigues

Agradecimentos

Agradeço primeiramente à minha família, pelo apoio, pelo investimento e pela compreensão durante não apenas a realização deste curso, mas durante toda a minha vida. Agradeço especialmente à minha mãe Marilene de Oliveira, à minha irmã Katerine de Oliveira Veloso e ao meu pai Eros Lopes Veloso.

Agradeço ao Professor Otávio Gomes, pela paciência e pelo apoio durante todo o curso. Por me proporcionar não apenas conhecimento, mas também por me ajudar com este trabalho, com atividades de pesquisa e com conselhos para a vida profissional.

Agradeço também aos discentes, docentes e funcionários do Instituto Federal de Minas Gerais, campus Formiga, por proporcionarem um curso de qualidade e oferecerem oportunidades a toda a comunidade.

"É melhor ir devagar na direção correta do que sair acelerando na direção errada."
(Sinek, Simon)

Resumo

Mapeamento e localização são necessários para que veículos autônomos consigam compreender o ambiente em que estão, possibilitando que estes veículos consigam navegar pelo ambiente sem colidir com obstáculos. Entretanto, é preciso de um mapa para calcular a posição de um veículo, e é preciso da posição do veículo para criar uma representação do mapa. O problema de estimar tanto o estado do mapa quanto a posição do veículo ao mesmo tempo é chamado de SLAM (*Simultaneous Localization and Mapping*). Com a popularização de plataformas de prototipagem eletrônicas e dos sistemas microeletromecânicos, é possível desenvolver robôs utilizando componentes de baixo custo. Neste trabalho foi desenvolvido um protótipo de um veículo autônomo, utilizando conceitos de localização e mapeamento simultâneos para auxiliar na navegação e criar uma representação do mapa. O mapa desenvolvido é representado por uma grade de ocupação de duas dimensões que fornece informações sobre a probabilidade de ocupação de regiões no mapa com precisão em centímetros.

Palavras-chaves: SLAM. Navegação de Robôs. Mapeamento. Localização.

Abstract

Mapping and localization are necessary for the comprehension of autonomous vehicles about the environment they are in, allowing these vehicles to navigate without colliding with obstacles. However, a map is necessary to estimate the vehicle's pose, and the pose is necessary for creating a representation of the map. The problem of estimating both the map's state and the robot's pose is called SLAM (Simultaneous Localization and Mapping). With the popularization of electronic prototyping platforms and microelectromechanical systems, it is possible to develop robots using low cost components. In this work, a prototype of a autonomous vehicle was developed using concepts of simultaneous localization and mapping to aid the navigation and to create a representation of the environment. The developed map is represented by a two-dimensional occupancy grid that provides information about the occupancy probability of specific regions in the map with a precision of centimeters.

Key-words: SLAM. Robot Navigation. Mapping. Localization.

Lista de ilustrações

Lista de quadros

Lista de tabelas

Lista de abreviaturas e siglas

SLAM	*Simultaneous Localization and Mapping*
IEEE	*Institute of Electrical and Electronics Engineers*
GPS	*Global Positioning System*
TRL	*Technology Readiness Level*
RANSAC	*Random Sample Consensus*
IMU	*Inertial Measurement Unit*
RXD	*Digital Data Receiver*
TXD	*Digital Data Transmitter*
IDE	*Integrated Development Environment*
GPIO	*General Purpose Input Output*
BCM	*Broadcom System On a Chip Channel*
HTML	*Hypertext Markup Language*
DOM	*Document Object Model*
TCP	*Transmission Control Protocol*
IETF	*Internet Engineering Task Force*
EKF	*Extended Kalman Filter*

Lista de símbolos

s	Conjunto de posições do veículo
u	Conjunto de controles do veículo
z	Conjunto de leituras dos sensores
n	Conjunto de dados extraídos das leituras
Θ	Estado do mapa.
a	Valor de saída de cada neurônio da camada oculta
w	Peso da conexão entre dois neurônios

Sumário

1 INTRODUÇÃO

Localização e Mapeamento Simultâneos (SLAM) é considerado um problema fundamental da robótica, no qual um robô que não possui conhecimento prévio de seu estado precisa se movimentar em um ambiente desconhecido. No SLAM, cabe ao robô construir gradativamente uma representação do ambiente enquanto se localiza em relação ao mapa criado simultâneamente (THRUN; BURGARD; FOX, 2005).

O problema de Localização e Mapeamento Simultâneos é considerado um problema complexo porque para que um veículo consiga se localizar é preciso de uma representação consistente do mapa, enquanto que para adquirir uma representação do mapa é necessário uma estimativa da posição do veículo. De acordo com Kuemmerle R. et. Al, esta dependência mútua entre a posição e o mapa tornam SLAM um problema difícil que requer soluções em um espaço multidimensional (KÜMMERLE et al., 2009).

As primeiras abordagens probabilísticas ao problema, que levam em conta a incerteza dos controles e dos sensores do veículo, foram propostas em 2003 por Durrant-Whyte e Bailey, embora ocorrências de modelagem probabilística tenham ocorrido desde 1986 na Conferência de Robótica e Automação da IEEE (BAILEY; DURRANT-WHYTE, 2006; CADENA et al., 2016). Os modelos probabilísticos foram seguidos por abordagens com filtros, propostas em 2008 por Aulinas et al. (AULINAS et al., 2008). Desde a abordagens de filtros foram estudados novos tópicos envolvendo diferentes tipos de odometria, como odometria visual (2012) (FRAUNDORFER; SCARAMUZZA, 2012; SCARAMUZZA; FRAUNDORFER, 2011) e reconhecimento de lugares (2016) (LOWRY et al., 2015).

O estado do robô é estimado por um conjunto de sensores embarcados. Um estado pode ser representado por sua posição e orientação; entretanto novas características podem ser utilizadas, como a velocidade do robô, a margem de erro dos sensores e parâmetros de calibração (CADENA et al., 2016).

A posição pode ser estimada por métodos de odometria baseados nas rodas do veículo. Nestes métodos são utilizados *encoders* para medir a velocidade de rotação das rodas. Os dados da odometria são utilizados juntos ao modelo de movimentação do robô para estimar sua localização em relação a um sistema de coordenadas cartesianas.

Entretanto este método possui limitações: a odometria é limitada a veículos com rodas; erros de medição são acumulados a cada iteração, aumentando discrepância dos erros de medições futuras, levando o veículo a derivar de sua posição; e as rodas são propensas a erro dependendo da fricção do terreno (YOUSIF; BAB-HADIASHAR; HOSEINNEZHAD, 2015).

Neste trabalho foi desenvolvido um prótotipo de um veículo autônomo e um sistema em *software* para resolver o problema do SLAM utilizando sensores de distância de baixo custo que fornecem apenas uma medida a cada iteração. O mapa é representado por uma matriz de duas dimensões, e o SLAM é feito através do método de mapeamento conhecido como mapeamento com posições conhecidas – *Mapping with Known Poses* (THRUN et al., 2002).

1.1 Justificativa

O conceito de SLAM é necessário para a robótica devido a sua necessidade para aplicações que se beneficiam de mapas consistentes; na realização de inspeções em estruturas, como edifícios e pontes, na qual o robô precisa mapear e criar uma representação consistente da estrutura em 3D. (THRUN; MONTEMERLO, 2006; CADENA et al., 2016).

Robôs também podem utilizar SLAM em aplicações *indoor*, como robôs de limpeza a vácuo, guias de exibições em museus, e automação de serviços farmacêuticos (PARKER, 2000); tipos de aplicações que popularizaram o termo por sua demanda. O uso de GPS para determinar a localização do veículo não é suficiente para garantir navegação autônoma. Assim a navegação autônoma é possível mesmo na ausência de uma infraestrutura de localização preexistente – como os satélites de um GPS. (CADENA et al., 2016).

SLAM é considerado o princípio fundamental de sistemas de navegação autônoma. O problema está presente em qualquer dispositivo móvel com autonomia para se locomover, e foi utilizado também para exploração planetária no robô *Opportunity* da NASA em 2003 (OLSON et al., 2003).

1.2 Objetivos

Este trabalho tem como objetivo desenvolver um protótipo de um veículo autônomo utilizando sensores de baixo custo que possa navegar em um ambiente de duas dimensões com obstáculos, sem conhecimento prévio de sua posição e dos obstáculos presentes.

Com o desenvolvimento deste protótipo foi realizada uma prova de conceito para um sistema de navegação que dispõe de sensores ultrassônicos e infravermelhos para realizar tanto a localização do veículo quanto o mapeamento do ambiente. O protótipo resultante visa atingir os níveis de prontidão tecnológica (TRL) 3 e 4, prova de conceito com validação em ambiente monitorado.

Dentre os objetivos específicos, estão:

- Criação de um sistema de navegação autônoma para competições de robótica, como robô seguidor de linha e resolução de labirintos;

- Aprofundamento dos conhecimentos na área de Robótica Móvel, com foco em navegação e posicionamento de robôs;
- Estudo da comunicação entre *software* embarcado e dispositivos remotos;
- Desenvolvimento de uma estrutura capaz de integrar sensores e atuadores;
- Implementação da incerteza de localização de um veículo autônomo;
- *Design* e implementação de uma estrutura de dados para representação do mapa;
- Desenvolvimento de um algoritmo de planejamento de rotas para veículos autônomos.

1.3 Estrutura do Trabalho

Esta monografia de conclusão de curso é organizada em 7 capítulos, sendo o primeiro uma introdução ao problema estudado, apresentando aplicações de veículos autônomos e a necessidade do estudo de localização e mapeamento simultâneos para o desenvolvimento da robótica. No Capítulo 2 são apresentados os conceitos utilizados neste projeto para o desenvolvimento do veículo autônomo, estes conceitos garantem uma melhor compreensão do trabalho, descrevendo as representações utilizadas em robótica probabilística, os conceitos de diferentes tipos de mapas e sensores, algoritmos populares para a navegação e um modelo matemático para o problema do SLAM. O Capítulo 3 apresenta os equipamentos e ferramentas utilizados no desenvolvimento do projeto, contextualizando cada material com o seu papel para o protótipo. Neste capítulo também são apresentados os métodos utilizados para o desenvolvimento do trabalho. No Capítulo 4 é demonstrado como o protótipo foi projetado e como o projeto foi executado, explicando as decisões tomadas durante a implementação, relacionando-as à fundamentação teórica apresentada neste trabalho. O Capítulo 5 apresenta os resultados obtidos durante a etapa de testes e validação, comparando-os com trabalhos relacionados (Seção 2.14). O Capítulo 6 aborda planos para trabalhos futuros utilizando este como referência. No Capítulo 7 são feitas as considerações finais sobre o projeto como um todo, seguido pelas referências bibliográficas e apêndices.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados os principais conceitos utilizados por técnicas de mapeamento e de localização. Estes conceitos foram observados durante a etapa de estudo do estado da arte e foram aplicados durante o desenvolvimento deste projeto e da escolha dos materiais relevantes à aplicação. Os conceitos utilizados são apresentados junto à justificativa de sua escolha e de alternativas ao método escolhido para cada parte do sistema.

As Seções 2.1 e 2.2 introduzem os principais tipos de mapas, a representação utilizada por este trabalho é discutida na Seção 2.1.2. A Seção 2.3 detalha métodos utilizados para extrair informações do ambiente. A Seção 2.4 apresenta uma definição do problema do SLAM que é utilizada durante o desenvolvimento deste trabalho. As Seções 2.6 e 2.7 apresentam duas áreas de inteligência artificial que foram utilizadas durante o desenvolvimento do sistema de desvio de obstáculos. A Seção 2.8 introduz o conceito de algoritmos de busca, que são utilizados pelo planejamento de rotas do veículo. As demais seções apresentam os principais dispositivos eletrônicos utilizados no sensoriamento do veículo autônomo. No final do capítulo são apresentados dois trabalhos relacionados que fazem uso dos mesmos conceitos utilizados por este trabalho.

2.1 Mapas Métricos

Mapas métricos representam o ambiente através da relação geométrica entre objetos e um ponto de referência. Os objetos são posicionados em coordenadas precisas sobre um espaço de duas dimensões. Esta representação é frequentemente utilizada, porém é sensível a erros devido à precisão requerida para a criação do mapa(YOUSIF; BAB-HADIASHAR; HOSEINNEZHAD, 2015). Para este trabalho foram utilizadas as Grades de Ocupação (Seção 2.1.2), para representar o ambiente como mapas métricos.

2.1.1 Mapas de Característica

Mapas de Característica representam o ambiente através dos marcos (Seção 2.3) encontrados no ambiente. Neste tipo de representação os marcos são modelados como conjuntos de formas geométricas e posições (CHATILA; LAUMOND, 1985).

A localização é feita através da observação e da extração dos marcos do ambiente, que podem ser armazenados e consultados para iterações futuras ou observações repetidas do mesmo marco.

Neste método, uma quantidade limitada de marcos são utilizados para representar o mapa, o que exige pouco esforço computacional. Entretanto, o processo de associação de dados, que consiste em verificar se um marco foi observado anteriormente, é sensível e pode levar a associações incorretas, constituindo uma desvantagem dos mapas de característica (YOUSIF; BAB-HADIASHAR; HOSEINNEZHAD, 2015).

2.1.2 Grades de Ocupação

Grades de ocupação são representadas por matrizes nas quais cada célula se refere a uma região do ambiente. Uma célula contém informação probabilística do estado do ambiente, obtido através de dados dos sensores. A informação é discreta e representa se a célula está ocupada **1**, ou não **0** (ELFES, 1990).

Uma das vantagens deste método de representação é sua utilidade em algoritmos como planejamento de caminhos e exploração, garantido tanto pela redução da complexidade devido à informação de células ocupadas quanto pela facilidade em se manipular matrizes de duas ou três dimensões (YOUSIF; BAB-HADIASHAR; HOSEINNEZHAD, 2015).

As grades de ocupação foram utilizadas neste trabalho como estrutura de dados para representação de mapas métricos. O veículo utiliza esta abstração com resoluções de 10cm por célula, e permite que novas resoluções possam ser utilizadas quando se deseja mapas mais exatos.

2.2 Mapas Topológicos

Mapas topológicos são utilizados para representar ambientes grandes sem precisar de determinar relações geométricas entre um robô e os marcos detectados. Ao invés disto, os mapas topológicos são definidos por grafos em que os vértices são constituídos por marcos do ambiente e os arcos por informações adjacentes a um par de marcos (DUDEK et al., 1991).

Os vértices de um mapa topológico incluem informações de alto nível como portas, janelas, pessoas e outros objetos. Uma das maiores vantagens deste método, devido à sua representação por grafos, é o seu suporte para implementar algoritmos de alta complexidade para planejamento de caminho, como encontrar o caminho mais curto. Porém, depender apenas de informações qualitativas pode não ser adequado para ambientes dinâmicos ou muito aglomerados (YOUSIF; BAB-HADIASHAR; HOSEINNEZHAD, 2015).

2.3 Marcos do Ambiente

Marcos do ambiente (*landmarks*) são características fáceis de serem percebidas pelos sensores de um robô. Estas características são utilizadas no processo de localização, no qual o robô utiliza os sensores disponíveis para descobrir marcos no ambiente e determinar sua distância em relação a estes (RIISGAARD; BLAS, 2003).

Landmarks podem ser medidas de distância, ou detecção de cores e formas conhecidas. Em *SLAM for dummies* (RIISGAARD; BLAS, 2003), o autor enfatiza que o tipo de marco utilizado depende do meio em que o robô está operando. Para determinar o que constitui um bom marco do ambiente são levadas em consideração quatro características:

- *Landmarks* devem ser re-observáveis: podendo ser detectadas facilmente independente da posição e da orientação do robô;
- *Landmarks* devem ser fáceis de serem distinguidas entre si: dois marcos observados podem ser detectados novamente e distinguíveis entre si;
- Devem existir *landmarks* suficientes no ambiente para que o robô não precise navegar por longo período de tempo sem ter com o quê se orientar;
- *Landmarks* devem ser estacionárias: elas não podem mudar de posição enquanto o robô navega pelo ambiente.

Para este trabalho, um marco do ambiente é qualquer obstáculo encontrado no mapa. Como o mapa é representado por uma matriz de duas dimensões, os marcos informam a célula da matriz e a existência de um obstáculo para a respectiva célula. Portanto, para cada obstáculo que os sensores detectarem, é criado um novo marco do ambiente. Os marcos são utilizados junto às grades de ocupação para alterar o estado do mapa de acordo com os novos marcos observados, ou marcos antigos que forem reobservados.

2.3.1 Spike Landmarks

A extração de *Spike landmarks* é feita através do cálculo da diferença entre a distância de duas leituras próximas a fim de encontrar pequenos obstáculos no ambiente — como portas, pés de mesas e curvas em paredes. Para obter estes marcos são utilizados dados de *scanners* a *laser* e sensores sonar, que fornecem leituras em unidades de comprimento, ou que podem facilmente ser convertidas para unidades de comprimento.

A Figura 1 demonstra um mapa obtido durante a navegação de um robô com o auxílio de *spike landmarks*. Os círculos representam marcos detectados.

Estes marcos são encontrados através da análise dos dados de sensores de distância cujos valores diferem por mais do que um valor preestabelecido, como 0.5m. Esta diferença

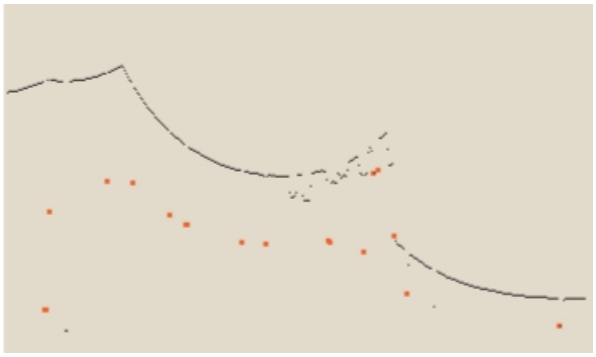


Figura 1 – Spike Landmarks (RIISGAARD; BLAS, 2003).

permite encontrar alterações grandes nos dados dos sensores, o que pode indicar que uma leitura encontrou uma parede, enquanto que a outra leitura mais próxima não encontrou tal obstáculo, por exemplo. Pode-se também utilizar três leituras próximas, ao subtrair a leitura entre duas outras leituras próximas, obtém-se dois valores que, quando somados, fornece uma boa aproximação para encontrar constrastes no ambiente (RIISGAARD; BLAS, 2003).

2.3.2 Consenso de Amostra Aleatória (RANSAC)

RANSAC - do inglês *Random Sample Consensus* - é um método iterativo para obter amostras de um conjunto de dados de modo que as amostras sigam uma determinada função matemática. O RANSAC pode ser utilizado tanto para descobrir parâmetros da função matemática em questão quanto para remover *outliers* (observações discrepantes em relação às demais) do conjunto de dados (FISCHLER; BOLLES, 1981).

Em SLAM, o método RANSAC é utilizado para extração de obstáculos como paredes, inclinações e demais retas ao alcance dos sensores do robô. A Figura 2 exibe as leituras de um scanner a *laser* representadas por pontos (esquerda), e as linhas resultantes encontradas pelo algoritmo RANSAC (direita).

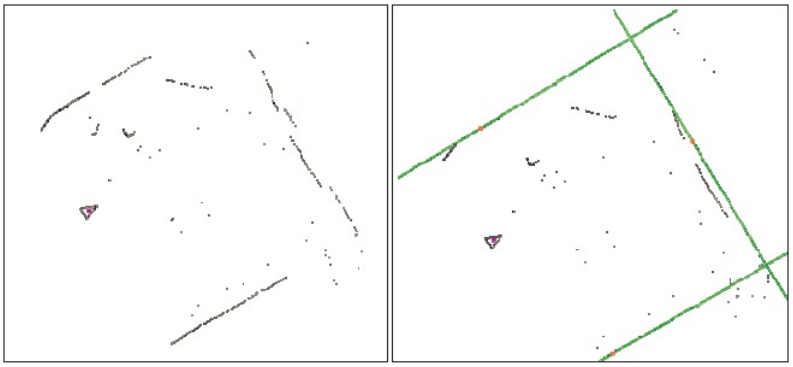


Figura 2 – RANSAC aplicado sobre leituras de um scanner a laser. Adaptado de (RIISGAARD; BLAS, 2003).

Tanto o método RANSAC quanto os marcos do tipo *Spikes* são boas alternativas para sensores de distância a *laser* ou ultrassônicos. Entretanto, estes métodos requerem

uma quantidade razoável de leituras espaçadas em pequenos ângulos umas das outras. Neste trabalho são utilizadas apenas três leituras dos sensores de distância, cada uma espaçada a 90°, portanto estes métodos de extração de marcos, embora importantes para o processo do SLAM, não foram utilizados.

2.4 Definição Formal de SLAM

O problema do SLAM pode ser definido matematicamente por quatro conjuntos de variáveis e o mapa. O modelo matemático apresentado foi definido por Thrun e Montemerlo (MONTEMERLO; THRUN, 2007).

A posição do robô em um dado tempo **t** é denominado s_t, cada posição consiste das coordenadas x e y (para movimento em um plano) e da sua orientação. Assim, durante a iteração **t** tem-se o conjunto de posições:

$$s_t = \{s_1, s_2, ..., s_t\} \tag{2.1}$$

Durante a navegação, o robô deve ser capaz de coletar informações sobre sua própria movimentação, que são obtidas por sensores de velocidade ou ainda pelos próprios controles do robô. Estas informações são denominadas controles (u_t) e são armazenadas num conjunto em função ao tempo **t**:

$$u_t = \{u_1, u_2, ..., u_t\} \tag{2.2}$$

Os marcos observados durante a navegação são descritos em função do tempo pelo conjunto z^t:

$$z_t = \{z_1, z_2, ..., z_t\} \tag{2.3}$$

Em SLAM, é comum que os marcos possam ser reobserváveis, para isto é necessário que cada marco seja decomposto em um conjunto de dados que o torne único, ou o mais próximo de ser distinguível. Cada observação de um marco **z** para uma dada posição **s** é representado por uma variável **n** que se refere à identidade do marco a ser observado. De acordo com Thrun e Montemerlo, as características dos marcos são extraídas uma de cada vez pelos dados dos sensores, o que tem sido usado em implementações com sucesso (MONTEMERLO; THRUN, 2007).

$$n_t = \{n_1, n_2, ..., n_t\} \tag{2.4}$$

Com as definições sobre posição (**s**), controles (**u**), marcos (**z**) identidade dos marcos (**n**), e o mapa Θ, o objetivo do SLAM é estimar a posição do robô (s_t) e o mapa (Θ) em um dado tempo, dado um conjunto de observações (z_t) e controles (u_t), sabendo-se que as observações não são ideais e estão propícias a erro.

O problema pode ser expressado em termos probabilísticos por:

$$p(s_t, \Theta | z_t, u_t) \tag{2.5}$$

Como em algumas instâncias do problema os marcos podem ser observados e diferenciados entre si, o conjunto **n** entra na expressão como variáveis conhecidas:

$$p(s_t, \Theta | z_t, u_t, n_t) \tag{2.6}$$

2.5 Cadeia de Markov

As Cadeias de Markov são modelos que descrevem a probabilidade de um evento ocorrer e alterar o estado atual de um determinado processo. As cadeias são similares a um grafo, em que os vértices constituem os estados e as arestas a probabilidade de acontecer uma mudança de estado (GAGNIUC, 2017).

Em SLAM, as Cadeias de Markov são utilizadas para criar um modelo gráfico utilizando a definição formal de SLAM (Seção 2.4). As Figuras 3 e 5 ilustram como a posição do robô pode ser estimada em duas categorias diferentes de SLAM. Para qualquer iteração **t** tem-se que a posição atual é definida como uma função probabilística da posição anterior (s_{t-1}) e do controle atual (u_t) utilizado pelo robô.

2.5.1 Online SLAM

Em *Online SLAM* o problema se limita a recuperar a última localização s_t do robô. Assim posições e observações anteriores não são levadas em consideração para o cálculo da localização, atingindo significante desempenho computacional.

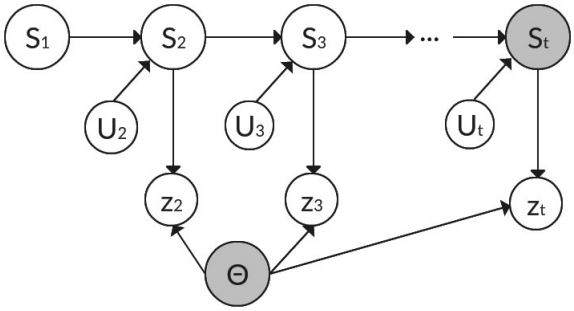
O modelo pode ser descrito como:

$$p(s_t, \Theta | z_{1:t}, u_{1:t}) \tag{2.7}$$

2.5.2 Full SLAM

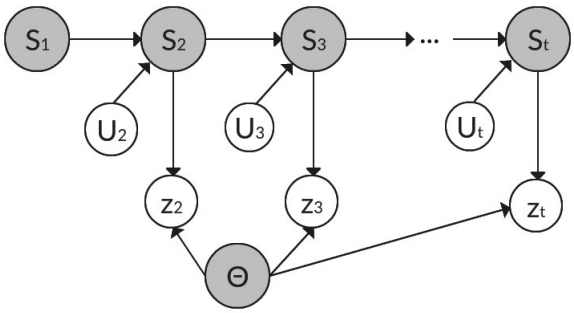
Em contraste ao *Online SLAM*, o *Full SLAM* consiste em encontrar todas as posições para um determinado mapa, de modo a estimar todo o caminho percorrido.

Figura 3 – Modelo gráfico do Online SLAM.



Adaptado de FastSLAM (MONTEMERLO; THRUN, 2007)

Figura 4 – Modelo gráfico do Full SLAM.



Adaptado de FastSLAM (MONTEMERLO; THRUN, 2007)

O modelo pode ser descrito como:

$$p(s_{1:t}, \Theta | z_{1:t}, u_{1:t}) \tag{2.8}$$

2.6 Redes Neurais Artificiais

Redes neurais artificiais são modelos que se baseiam na hipótese de que o comportamento do cérebro consiste em atividade eletroquímica em redes de células do cérebro, neurônios. O interesse em redes neurais artificiais surgiu em pesquisadores de inteligência artificial e estatística devido a sua aplicação em plataformas de computação distribuídas, a capacidade da rede de tratar entradas ruidosas e a capacidade da rede de aprender (RUSSEL; NORVIG, 2003).

Uma rede neural artificial é composta por um conjunto de nós, denominados neurônios, conectados por enlaces, ou *links*. Cada enlace possui um peso associado a ele que é utilizado para determinar se um nó será ativado ou não. A alteração do peso destes enlaces é a maneira com que a rede neural é treinada; com diferentes pesos, diferentes neurônios serão ativados.

A ativação de um nó é dada por uma função de ativação (Equação 2.10) que recebe como entrada a combinação linear da saída dos neurônios da camada anterior

(a_i) multiplicada pelo peso da conexão ($w_{i,j}$) entre o nó da camada anterior e o nó atual (Equação 2.9).

$$input_j = \sum_{i=1}^{n} w_{i,j} a_i \tag{2.9}$$

$$output_j = g(input_j) = g(\sum_{i=1}^{n} w_{i,j} a_i) \tag{2.10}$$

Para criação de uma rede neural artificial tanto a função de ativação quanto a quantidade de camadas e nós são obtidos empiricamente, cabendo ao projetista determinar estes parâmetros.

2.7 Algoritmo Genético

O Algoritmo Genético é um algoritmo evolutivo desenvolvido inicialmente para estudo do comportamento adaptativo e é utilizado como método de otimização. O algoritmo se baseia em mecanismos de evolução como seleção natural, cruzamento e mutação, contando com diferentes estratégias para realizar cada mecanismo (EIBEN; SMITH et al., 2003).

Os algoritmos genéticos foram introduzidos na década de 1970 por John Holland, através dos seus estudos sobre evolução natural, o qual acreditou que poderia ser adaptado solucionar problemas computacionais (GABRIEL; DELBEM, 2008). De acordo com Michalewicz (MICHALEWICZ, 2013), as principais etapas de um algoritmo genético são:

- Inicializar uma população de indivíduos, na qual cada indivíduo representa uma possível solução para o problema;
- Avaliar cada indivíduo da população (determinar o *fitness* dos indivíduos);
- Escolher indivíduos pelos quais serão gerados novos individúos;
- Fazer com que alguns indivíduos possam sofrer alterações por meio de mutações;
- Selecionar os indivíduos sobreviventes para a próxima geração.

Existem diferentes métodos para selecionar os indivíduos que serão utilizados para gerar novos indivíduos. O algoritmo da roleta seleciona os indivíduos de acordo com os melhores valores de *fitness*, quanto melhor o *fitness* de um indivíduo, maior a probabilidade de que ele seja escolhido para cruzamento. Na seleção por torneio, é selecionada uma quantidade de pais aleatoriamente, destes indivíduos, vence aquele que possuir o melhor *fitness* (GOLDBERG; HOLLAND, 1988).

Neste trabalho foi utilizada a seleção elitista. Neste tipo de seleção os melhores indivíduos da geração atual são passados para a próxima, impossibilitando que indivíduos com pior *fitness* possam ser levados à próxima geração (FOGEL, 1994).

O processo de obter novos indivíduos através de dois indivíduos previamente selecionados é chamado de Recombinação. Neste processo uma nova solução é criada a partir da solução representada pelos pais. Existem diferentes tipos de recombinação, como recombinações de n-pontos, nas quais os cromossomos dos pais são divididos em n partições e recombinados para gerar novos filhos; recombinação uniforme, na qual uma lista de variáveis aleatórias seguindo uma distribuição uniforme em [0, 1] é criada, e para cada posição do cromossomo é escolhido o gene de um pai diferente de acordo com a variável aleatória (GABRIEL; DELBEM, 2008).

Neste trabalho foi utilizada a recombinação aritmética simples. Neste tipo de recombinação novos indivíduos são gerados através de uma combinação linear entre dois cromossomos x e y, para gerar o descendente z (Equação 2.11).

$$z = \alpha x + (1 - \alpha)y \tag{2.11}$$

A recombinação pode ser feita em apenas um gene (recombinação individual), através do sorteio de uma posição do alelo pelo qual a recombinação irá começar, ou percorrendo todo o cromossomo (recombinação total) (MICHALEWICZ; JANIKOW, 1991).

2.8 Algoritmos de Busca

Os algoritmos de busca são técnicas que analisam o estado atual de uma dada estrutura e decidem qual a melhor ação possível que pode ser tomada para se aproximar do estado desejado. Estes algoritmos são utilizados no processo de planejamento de rotas em vídeo-games e robôs autônomos(HART; NILSSON; RAPHAEL, 1968).

2.8.1 Algoritmo A*

O algoritmo A* é um algoritmo de busca informada que toma decisões de acordo com uma função de avaliação heurística. Por possuir dados adicionais à definição do problema e ao seu estado, buscas informadas são capazes de encontrar o caminho entre dois estados analisando menos estados. A função heurística fornece informações adicionais ao algoritmo de busca, como a distância entre o objetivo e as demais posições do campo de busca.

Neste trabalho, o algoritmo implementado utiliza o menor caminho entre o estado inicial e o objetivo. Para determinar o caminho, o algoritmo expande o estado inicial em

busca da melhor ação que pode ser tomada para aproximar do objetivo. A ação é decidida pelo custo da heurística acrescido do custo da distância entre os dois estados. Após a ação, é armazenado um conjunto de estados que podem ser expandidos através do novo estado. O algoritmo se repete até chegar ao objetivo, ou até não existirem mais estados para serem expandidos, o que ocorre quando não há caminho entre o estado inicial e o objetivo (RUSSELL; NORVIG, 2016).

2.9 WebSockets

WebSockets é um protocolo de comunicação que opera sobre o protocolo TCP. O protocolo fornece a transferência de mensagens em ambas as direções (*full-duplex*), além de entregar mensagens em ordem e garantir a entrega de qualquer mensagem. O protocolo *WebSockets* foi padronizado em 2011 *Internet Engineering Task Force* (IETF) e fornece uma API para navegadores desenvolvida pelo *World Wide Web Consortium* (W3C).

As *WebSockets* permitem realizar a comunicação entre navegadores de internet e servidores *web*. O protocolo é capaz de transmitir pacotes com cabeçalhos menores do que o protocolo HTTP o que facilita a comunicação de aplicações em tempo real (IETF, 2011).

2.10 Unidade de Medição Inercial

A inclinação de cada eixo é obtida através de uma Unidade de Medição Inercial (IMU) que comporta sensores como acelerômetros, giroscópios e magnetômetros. O acelerômetro é capaz de medir a aceleração (m/s^2) ou em Força-G (g), uma unidade de aceleração em que 1g equivale à gravidade na superfície da Terra, aproximadamente 9.806 m/s^2. O giroscópio mede a velocidade angular com três graus de liberdade. As unidades utilizadas pelos giroscópios podem ser graus por segundo ($^\circ/s$) ou revoluções por segundo (RPM).

Os IMUs não são capazes de medir diretamente a inclinação em graus, portanto é necessário calcular a inclinação através dos dados obtidos do acelerômetro. Com a leitura do acelerômetro pode-se calcular a inclinação do IMU sobre os eixos **x** e **y**. A inclinação sobre o eixo **z** pode ser determinada apenas em relação à sua inclinação inicial, mas não em relação ao pólo norte magnético da Terra, para isto é necessário um magnetômetro. Para obter a inclinação em relação ao eixo **z** é necessário um magnetômetro – sensor capaz de medir intensidade de campo magnético.

2.11 Sensor Ultrassônico

Sensores ultrassônicos utilizam ondas sonoras para detectar obstáculos e medir distâncias. Ondas sonoras são ondas mecânicas que se propagam em diferentes materiais

em forma de energia. A velocidade de propagação do som difere de acordo com o meio de propagação, sendo a velocidade de propagação no ar de aproximadamente 330m/s (POSSANI et al., 2017).

Embora os sensores ultrassônicos utilizem ondas sonoras, estas ondas sonoras são emitidas em uma frequência inaudível pelos humanos, denominada ultrassom. Os primeiros indícios das ondas ultrassom surgiram do estudo da audição apurada de animais como os morcegos, que são capazes de detectar ondas sonoras com frequência superior ao limiar do ouvido humano (20Hz a 20KHz) (POSSANI et al., 2017).

Ondas ultrassom possuem aplicações além da detecção de obstáculos. Ultrassom é utilizado na medicina para exames de ultrassonografia pré-natal, exames do sistema gástrico, e em próteses para portadores de deficiência visual[1] em aplicações industriais ondas ultrassom são utilizadas para inspeção de soldas e estruturas (POSSANI et al., 2017); na área militar são utilizados em submarinos para navegação e comunicação (ELIZABETH; CHARLES; ROBIN, 1975).

Os sensores ultrassônicos possuem um emissor e um receptor responsáveis por enviar ondas ultrassom em uma determinada frequência e captar a reflexão das ondas emitidas, respectivamente. Estes dispositivos se baseiam no conceito da reflexão de ondas. Quando uma onda ultrassom atinge um obstáculo ela retorna na direção oposta permitindo ao receptor detectá-la para que o sensor calcule o tempo gasto entre a emissão e a recepção da onda. A distância é então calculada em função do tempo gasto pela onda para atingir o obstáculo e voltar ao sensor.

A Seção 3.2 apresenta a utilização do sensor ultrassônico HC-SR04, o seu funcionamento e como a distância é obtida em razão do tempo medido.

2.12 Sensor Infravermelho

Os sensores infravermelhos são dispositivos eletrônicos utilizados para obter informações do ambiente ao medir o calor emitido por um objeto ou por sua movimentação (MAHULIKAR; SONAWANE; RAO, 2007). Estes sensores são encontrados comumente no cotidiano, como em controles remotos e sensores de presença para instalações residenciais. Sensores infravermelhos são classificados como sensores passivos e ativos.

Os sensores passivos são utilizados em sistemas de detecção de movimento. Estes dispositivos possuem um receptor de radiação infravermelha capaz de detectar presença através do calor emitido pelos corpos dispostos ao seu alcance. Os sensores ativos entretanto possuem, além do receptor, um emissor de luz infravermelho — como LEDs IR. Sensores ativos são normalmente utilizados para detectar obstáculos.

[1] Próteses para portadores de deficiência visual: <https://www.forbes.com/sites/alexknapp/2013/02/23/prototype-suit-gives-you-real-life-spider-sense/#1032fff17690>. Acessado em Junho/2019.

Assim como ondas ultrassom não são audíveis pelo ouvido humano, ondas infravermelho não são visíveis ao humano, mas podem ser percebidas como calor, características que permitem a estas ondas serem utilizadas para comunicação entre dispositivos eletrônicos e visão noturna. O comprimento das ondas infravermelho se situam entre 700nm a 1mm, comprimento acima do espectro visível que detecta ondas de 750nm a 400nm (DANNO et al., 2001).

2.13 Chave Óptica

As chaves ópticas são sensores capazes de detectar a presença de um objeto entre seus terminais. Os terminais de uma chave óptica são compostos de um LED emissor de infravermelho e um foto-transistor sensível a infravermelho. Quando o feixe do LED emissor é interrompido por algum obstáculo, o sensor altera o nível lógico do pino de sinal. Assim é possível determinar quando um obstáculo está entre os terminais do sensor através de um microcontrolador.

Estes sensores são geralmente utilizados para determinar a velocidade ou o deslocamento de um veículo. Para calcular o deslocamento do veículo, utiliza-se discos *encoder*, que possuem perfurações igualmente espaçadas em torno do eixo que é acoplado ao eixo do motor. Calculando-se experimentalmente a quantidade de vezes que a chave óptica troca de estado em razão da distância medida, pode-se determinar a distância que o veículo percorre por pulso da chave óptica(ENCODER. . . , 2017).

2.14 Trabalhos Relacionados

Existem na literatura projetos similares que envolvem os mesmos sensores utilizados neste trabalho para o problema do SLAM. Estes trabalhos são importantes por provêrem resultados que podem ser comparados com trabalhos futuros sobre navegação autônoma em ambientes planos. O estudo de projetos relacionados também serve como estudo de caso para entender como a fundamentação teórica da navegação autônoma é implementada em casos reais.

Em um trabalho desenvolvido pelos estudantes da Universidade de Auckland, *Autonomous SLAM Robot* para o curso *MECHENG 706*, foi desenvolvido um veículo autônomo com capacidade de navegação similar aos robôs aspiradores, como o *Roomba*. Para que o robô pudesse se localizar e mapear o ambiente sem passar por áreas repetidas foi implementado um sistema de SLAM que conta com um conjunto de sensores de distância de alta precisão. O trabalho descreve todas as etapas do processo de desenvolvimento, da confecção da estrutura do robô à calibração dos sensores e testes.

O trabalho (BUONOCORE; NETO; JÚNIOR, 2012) aproxima o problema do

SLAM com o uso de sensores de baixo custo, com sensor infravermelho, ultrassônico e uma câmera. Os autores justificam a escolha de sensores de baixo custo pelo desafio de se obter medições precisas e rápidas se comparados aos sensores utilizados na literatura. Através da fusão proposta para os dados obtidos pelos sensores, os autores foram capazes de construir veículos autônomos para ambientes fechados sem a necessidade de dispositivos caros. Entretanto o trabalho não entra em detalhes sobre o *hardware* utilizado para navegação, como microcontroladores e estrutura do robô.

2.15 Sumário do Capítulo

Os principais conceitos utilizados para o desenvolvimento deste trabalho foram apresentados neste capítulo. Assim como os termos utilizados no decorrer deste texto foram definidos e serão explorados nos próximos capítulos.

Foi apresentada uma definição formal de SLAM apresentada no livro FastSLAM, que será utilizada neste trabalho como referência para definir quais serão os controles do robô e como as leituras dos sensores serão transformadas em dados relevantes à aplicação.

Também foi introduzido o conceito de incerteza na localização e da robótica probabilística. Estes conceitos demonstram como erros de leitura devem ser levados em consideração para estimar a localização de um veículo com boa fidelidade.

No final do capítulo foram apresentados os sensores utilizados neste projeto, explicando o princípio de funcionamento de cada sensor e quais são suas principais funcionalidades em sistemas embarcados.

3 MATERIAIS E MÉTODOS

Neste capítulo são apresentados os materiais utilizados para o desenvolvimento do protótipo. São explicados os detalhes dos dispositivos necessários à aplicação, como sensores, motores e plataformas de prototipagem, analisando a necessidade destes componentes para o desenvolvimento do trabalho e como eles podem ser integrados com a aplicação desenvolvida. No final do capítulo são discutidos projetos de iniciação científica e a sua importância no aprendizado de novas tecnologias, as quais foram utilizadas para este projeto.

Este trabalho foi desenvolvido em pequenos projetos, cada um focando na implementação de um componente do veículo. Para cada projeto completado foram escritos testes funcionais para verificar se o componente funciona como esperado. Com os componentes aceitos pelos testes funcionais foram desenvolvidos projetos maiores que agregam um ou mais componentes, como a classe *Robot* e *Controller*. Para estes projetos foram criados testes de integração para verificar que os componentes funcionam corretamente quando utilizados em conjunto.

Para gerenciar os projetos em andamento e manter uma visão do sistema em sua completude, foi utilizado o método *Kanban*, criando-se listas de tarefas com prazos definidos e as organizando de acordo com a fase de desenvolvimento em que a tarefa se encontra. Para facilitar o uso deste método, foi utilizada a ferramenta *Trello* [1], uma aplicação *online* que permite gerenciar listas de atividades e compartilhar com outros usuários.

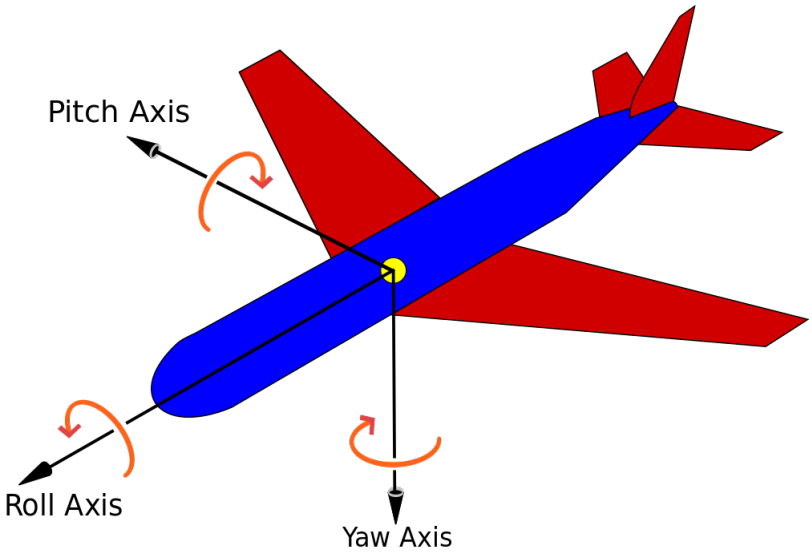
3.1 Unidade de Medição Inercial

Para este projeto foi utilizado o IMU MPU-9250, que possui um magnetômetro além do acelerômetro e do giroscópio. Este IMU é capaz de determinar a orientação de um corpo rígido em três eixos, possibilitando ao veículo se orientar em relação aos eixos latitudinais e longitudinais, necessários para navegação em um ambiente plano.

Por possuir um magnetômetro adicionalmente ao acelerômetro e o gisroscópio de três eixos que a versão MPU-6050 têm, o MPU-9250 possibilita calcular o ângulo de rotação de um corpo rígido qualquer em relação aos três Ângulos de Euler – também conhecidos por *roll*, *pitch*, *yaw*. Os Ângulos de Euler são um modelo matemático proposto por Leonard Euler para expressar a orientação de um objeto em um espaço tridimensional (EDMONDS, 1996).

[1] Trello: <https://trello.com/>. Acessado em Junho/2019.

Figura 5 – Os principais eixos de uma aeronave.



Adaptado de (AEROSPACE. . . , 1990)

Para este projeto é necessário apenas um ângulo para expressar a orientação do veículo. Entretanto esta orientação é paralela à superfície da Terra e serve para representar a orientação do veículo quantos aos pólos magnéticos do planeta. Portanto foi utilizado o MPU-9250 para mensurar o ângulo de *yaw* do veículo.

O MPU-9250 também fornece informação sobre a temperatura do dispositivo, e possibilita que outros dispositivos que utilizem o mesmo protocolo de comunicação sejam conectados a ele. Para abstrair o processo de ler informações do sensor foi utilizada a biblioteca RTIMULib2[2] para *Python*. Esta biblioteca abstrai o processo de leitura de bits pelo protocolo i2C, reconhece diversos tipos de *IMU*, e realiza a fusão de dados do acelerômetro, giroscópio e magnetômetro, retornando os ângulos de *pitch*, *yaw* e *roll* em radianos.

3.2 Sensores Ultrassônicos

Neste projeto foram utilizados os sensores ultrassônicos HC-SR04 e Parallax Ping, que fornecem meios para medir distâncias entre 2cm e 400cm, e 3cm a 300cm, respectivamente. Estes sensores utilizam ondas ultrasom para realizar medições e portanto dispensam contato entre o sensor e o obstáculo. Ambos os sensores operam sobre uma faixa de detecção de 15°.

[2] RTIMULib2: <https://github.com/richardstechnotes/RTIMULib2>. Acessado em Junho/2019.

3.2.1 Sensor HC-SR04

O sensor HC-SR04 possui um receptor e um emissor que podem ser acessados pelos pinos *Trigger* e *Echo*, respectivamente. De acordo com o *datasheet*[3] do HC-SR04, o sensor opera sobre tensão de 5V, com corrente elétrica de até 15mA. Este sensor é adequado para superfícies planas e lisas, possui um baixo custo e é comumente utilizado por *hobbistas* e no desenvolvimento de protótipos. Superfícies não ideais podem afetar negativamente os resultados de leitura do sensor.

Para realizar uma leitura, o *Trigger* do sensor deve ser acionado por 10 µs, após o acionamento são enviados oito pulsos de 40kHz. O módulo aciona o pino *Echo* e o mantém em nível lógico alto até que receba a reflexão do pulso enviado. A distância é então calculada em razão do tempo em que o pino *Echo* permaneceu acionado.

$$d = t_{echo} * \frac{V_{som}}{2} \tag{3.1}$$

Na Equação 3.1 a distância (d) é o produto entre o tempo em que o *Echo* ficou em nível lógico alto (t_{echo}) e a velocidade do som (V_{som}). O resultado é dividido por dois porque é levado em consideração que a onda de ultrasson levou metade de t_{echo} até atingir o obstáculo de reflexão, e metade de t_{echo} para voltar ao receptor do sensor. Portanto o tempo para uma onda ultrassom chegar à superfície refletora é de $t_{echo}/2$.

3.2.2 Sensor Parallax Ping

O sensor ultrassônico *Parallax Ping* possui o mesmo princípio de funcionamento do HC-SR04, porém a pinagem deste sensor é diferente. O *Parallax Ping* possui dois pinos de alimentação (V_{cc} e GND) e um pino digital para troca de informações, ao contrário dos pinos *trigger* e *echo* do HC-SR04. Este pino digital opera tanto como dispositivo de saída, para receber informações de quando emitir ondas ultrasom para medição, quanto como dispositivo de entrada, para informar quando a onda emitida retornou ao sensor.

A escolha deste sensor sobre o HC-SR04 se deve ao fato de precisar de um pino a menos. Como os *GPIO* do *Raspberry Pi* são limitados, o sistema também é limitado pelo número de pinos de entrada e saída, portanto escolher sensores que utilizam menos recursos do microcontrolador permite expandir o sistema com novos dispositivos e funcionalidades.

3.3 Sensor Infravermelho

Para obter informações sobre o ambiente, foi também utilizado um sensor de proximidade infravermelho para auxiliar os sensores ultrassônicos durante a leitura das

[3] HC-SR04 *Datasheet*: <https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf>. Acessado em Junho/2019

distâncias entre o veículo e os obstáculos durante a navegação. O sensor *E18-D80NK* possui uma faixa de detecção entre 3cm a 80cm que pode ser ajustada por um potênciometro. Diferentemente dos sensores ultrassônicos, este sensor não é capaz de medir distâncias variáveis em tempo de execução. A distância ajustada pelo potenciômetro é fixa, cabendo ao sensor informar apenas se existe ou não um obstáculo no alcance.

Assim como os sensores ultrassônicos, o *E18-D80NK* opera sob tensão contínua de 5V, e pode ser integrado facilmente com as plataformas de prototipagem *Arduino* e *Raspberry Pi* pelos pinos de entrada e saída digitais. Estes sensores são também comercializados para estudantes e para a criação de protótipos de produtos eletrônicos, são de baixo custo e fornecem medições estáveis.

3.4 Chave Óptica

Para este trabalho foi utilizada a chave óptica FC-03, junto a um disco de 20 perfurações. O FC-03 possui dois pinos de sinal, um digital e um analógico, ambos fornecendo informações até 3.3V ou 5V. Para este projeto foi utilizado apenas o pino digital, e o circuito foi alimentado com 3.3V devido à limitação de voltagem dos *GPIOs* do *Raspberry Pi*.

Durante o desenvolvimento foi calculado experimentalmente a quantidade de pulsos por centímetro. Este valor foi utilizado na odometria do veículo, possibilitando calcular a distância percorrida em razão da quantidade de pulsos lidos (Seção 4.3.5).

3.5 Plataformas de Prototipagem

Plataformas de prototipagem são dispositivos que facilitam o desenvolvimento de projetos de eletrônica. Estas plataformas são geralmente compostas por um microcontrolador, interface USB, entrada para fonte de alimentação e diversos pinos de entrada e saída que permitem implementar diferentes dispositivos em conjunto com o microcontrolador. Os pinos podem variar quanto a sua funcionalidade, podendo ser utilizados como pinos digitais, analógicos, comunicação i2C, modulação de pulsos, e até como fonte de alimentação. Algumas plataformas de prototipagem são capazes de comportar sistemas operacionais, interface Ethernet e HDMI, e são comumente denominados microcomputadores.

3.5.1 Arduino

O *Arduino* é uma ferramenta de desenvolvimento utilizada por estudantes, engenheiros e *hobbistas* para criar projetos de eletrônica e produtos para Internet das Coisas (IoT). A linguagem utilizada pela ferramenta é escrita em C++ e é capaz de compilar programas na linguagem *Arduino*, C ou C++. Tanto a linguagem quanto a IDE do *Arduino*

– desenvolvida para programar e compilar códigos no microcontrolador – são *open-source* e permitem que a comunidade contribua no seu desenvolvimento com códigos e tutoriais.

Existem diferentes placas *Arduino* que se diferenciam quanto ao poder de processamento, a quantidade de memória disponível e a quantidade de pinos de entrada e saída disponíveis. Todas as placas compartilham da linguagem de programação *Arduino*.

3.5.2 Raspberry Pi

Raspberry Pi é um microcomputador desenvolvido pela *Raspberry Pi Foundation* com o objetivo de disseminar a área da computação e desenvolvimento digital, tanto para resolução de problemas quanto para expressão artística[4]. A Raspberry Pi Foundation oferece também recursos para treinamento e ambientação com a ferramenta, com Python e com programação em geral[5].

O microcomputador é capaz de portar diferentes sistemas operacionais para trabalhos com tabelas, editores de texto, desenvolvimento de *software* e projetos de eletrônica. Para projetos de eletrônica, a placa conta com 40 GPIOs (*General-Purpose Input/Output*), pinos controlados em tempo de execução que enviam e recebem sinais digitais a circuitos conectados à eles.

Existem diferentes versões de placas Raspberry Pi, que se distinguem pelo poder de processamento e funcionalidades adicionais, como conexão Wi-Fi e número de portas USB. Em geral, as placas têm maior poder computacional do que microcontroladores como Arduino, por serem projetadas para servirem como microcomputadores. A Tabela 1 compara diferentes versões da placa com características de destaque.

Versão	**Velocidade**	**RAM**	**USB**	**Wireless**	**Bluetooth**
Raspberry Pi Modelo A+	700MHz	512MB	1	Não Possui	Não Possui
Raspberry Pi Modelo B+	700MHz	512MB	4	Não Possui	Não Possui
Raspberry Pi 2 Modelo B	900MHz	1GB	4	Não Possui	Não Possui
Raspberry Pi 3 Modelo B	1200MHz	1GB	4	802.11n	4.1
Raspberry Pi 3 Modelo B+	1400MHz	1GB	4	802.11ac/n	4.2
Raspberry Pi Zero	1000MHz	512MB	1	Não Possui	Não Possui
Raspberry Pi Zero W	1000MHz	512MB	1	802.11n	4.1
Raspberry Pi Zero WH	1000MHz	512MB	1	802.11n	4.1

Tabela 1 – Diferenças entre os modelos de Raspberry Pi. Raspberry Pi Documentation

Para este projeto foi utilizada a placa Raspberry Pi 3 Modelo B+ devido ao seu maior poder de processamento e à conexão *Wireless*, utilizada para estabelecer conexão entre o robô e a aplicação de controle.

[4] Raspberry Pi Foundation: <https://www.raspberrypi.org/>. Acessado em Junho/2019.
[5] Raspberry Pi Projects: <https://projects.raspberrypi.org/en/>. Acessado em Junho/2019.

3.6 Ponte H

Pontes H são circuitos eletrônicos capazes de controlar a alimentação de uma carga, possibilitando o controle para inverter a polaridade da alimentação ou desabilitar a fonte. Pontes H são comumente utilizadas em robótica para o controle de motores de corrente contínua, permitindo o acionamento em ambos os sentidos.

3.6.1 Ponte-H L298N

A Ponte-H L298N é um *driver motor* utilizado para controlar cargas indutivas como relés, solenoides, motores DC e motores de passo. O módulo é capaz de comportar até dois motores de corrente contínua, operando sobre tensões de alimentação de 4V a 35V, temperaturas entre -20 e 135°C e potência de até 25W[6].

3.6.2 Arduino *Motor Shield*

O *Arduino Motor Shield* é um *shield* para Arduino que permite o controle de motores de corrente contínua e servo motores. Os *shields* de Arduino são componentes que podem ser integrados à placa de desenvolvimento através dos pinos de entrada e saída da placa, conferindo-lhe novas funcionalidades.

O *Motor Shield* é composto por circuitos integrados L293D, que possibilitam à placa controlar a velocidade e a direção de até quatro motores, além de separar a fonte de alimentação dos motores da fonte do Arduino. A tensão de saída do *Motor Shield* é de 4.5V até 36V, fornecendo corrente elétrica de 600mA para cada um dos quatro canais. O módulo ocupa 14 pinos digitais, disponibilizando apenas os pinos analógicos para placas menores como o Arduino Uno[7].

3.7 Divisor de tensão

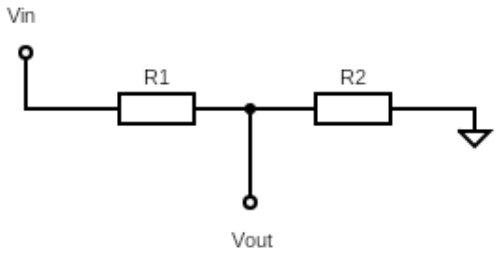
O circuito divisor de tensão é utilizado para reduzir a tensão de entrada em um circuito elétrico. Deste modo pode-se obter a tensão necessária para alimentar um dispositivo que possui tensão de operação menor do que a fornecida. O circuito é composto por dois ou mais resistores associados em série (Figura 6).

Na Figura 6, a corrente elétrica que flui por R_2 com tensão de V_{out} é igual à corrente elétrica que passa pelos dois resistores R_1 e R_2, sob a tensão V_{in}, pois eles estão associados em série. Assim, através da Lei de Ohm, pode-se deduzir a relação entre a tensão de

[6] L298N Datasheet: <https://www.sparkfun.com/datasheets/Robotics/L298_H_Bridge.pdf>. Acessado em Junho/2019.

[7] L293D Datasheet: <https://www.arduino.cc/documents/datasheets/L293D.pdf>. Acessado em Junho/2019.

Figura 6 – Diagrama unifilar de um divisor de tensão.



Fonte: Próprio Autor

entrada (V_{in}) e a tensão de saída (V_{out}), que atuam sobre os resistores R_1 e R_2. Este efeito ocorre devido a característica dos resistores de causarem queda de tensão, diminuindo a tensão dos demais componentes conectados ao resistor em série (TIPLER; MOSCA, 2000).

A Equação 3.2 mostra o processo para deduzir a tensão de saída através da igualdade das correntes elétricas.

$$I_{R1,R2} = I_{R2}$$
$$\frac{V_{in}}{R_1 + R_2} = \frac{V_{out}}{R_2}$$

$$V_{out} = \frac{V_{in} * R_2}{R_1 + R_2} \tag{3.2}$$

3.8 Projetos de Iniciação Científica

Durante o desenvolvimento deste trabalho foram desenvolvidos dois projetos de iniciação científica. Ambos os projetos são da área de sistema embarcados e focam no uso de sensores e na comunicação entre *software* embarcado e computador remoto. Os projetos de iniciação científica garantiram a familizarização com as plataformas Arduino e Raspberry, assim como conhecimento sobre sensores ultrassônicos e IMUs.

3.8.1 Módulo de Localização

Para o Edital 21/2017 foram estudados os conceitos de sensores ultrassônicos, de unidades de medição inercial e de GPS. Após o estudo destes dispositivos, eles foram implementados e testados na plataforma de prototipagem Arduino. Esta iniciação científica propiciou um primeiro contato com os sensores de distância ultrassônicos e as IMUs, que foram essenciais para o desenvolvimento deste trabalho de conclusão de curso.

3.8.2 Sistema de Estabilidade e Controle para Drones

Foi desenvolvido para o Edital 28/2017 um Sistema de Estabilidade e Controle para Drones. Neste projeto foi implementado um *software* embarcado para realizar o controle da estabilidade de um drone quadrirrotor através de um controlador PID. O *software* desenvolvido utilizou o protocolo *WebSocket* para comunicação com um aplicativo de controle remoto, e a unidade de medição inercial MPU-6500.

O protótipo do veículo desde projeto de conclusão de curso faz uso das tecnologias utilizadas no desenvolvimento deste projeto de iniciação científica, como o protocolo *WebSockets*, o microcomputador Raspberry Pi, a unidade de medição inercial e programação de dispositivos de entrada e saída em Python.

3.9 Sumário do Capítulo

Os materiais utilizados para o desenvolvimento deste trabalho foram introduzidos neste capítulo. Introduzindo cada sensor adicionado ao veículo e suas principais características, como os sensores de distância e a unidade de medição inercial. Também foram apresentados os circuitos necessários para integrar os sensores e atuadores aos microcontroladores Arduino e Raspberry Pi, como o circuito divisor de tensão e a Ponte-H.

Foi também apresentado neste capítulo as metodologias utilizadas para desenvolver este trabalho, de modo a diminuir o escopo pequeno, simplificando o desenvolvimento do trabalho como um todo, a partir do desenvolvimento de projetos menores e menos complexos. Um dos fatores que influenciou a escolha dos dispositivos apresentados neste capítulo foram as iniciações científicas desenvolvidas durante o curso, que possibilitaram o desenvolvimento de projetos com Arduino e Raspberry Pi antes do início deste trabalho.

4 PROJETO E DESENVOLVIMENTO

O desenvolvimento do projeto foi realizado após a condução dos estudos bibliográficos sobre localização e mapeamento na área de robótica móvel. O projeto foi desenvolvido através da implementação dos dispositivos necessários à aplicação em duas plataformas de prototipagem diferentes, Arduino e Raspberry Pi. Após a validação dos dispositivos e da realização de testes funcionais foi feito o desenvolvimento do algoritmo de localização e mapeamento, utilizando os sensores e motores previamente testados. Por fim foram feitas simulações dos algoritmos que não puderam ser implementados no *software* embarcado devido a limitações de *hardware*, como os sensores de distância escolhidos e a estrutura do veículo.

4.1 Desenvolvimento Inicial com Arduino

Durante o desenvolvimeno inicial do projeto foi implementada uma rede neural artificial para desvio de obstáculos com Arduino. O treinamento da rede foi feito utilizando conceitos de algoritmo genético, na qual a entrada é composta pelos dados dos sensores ultrassônicos e a saída da rede é o acionamento dos motores que determina se os motores da direita e/ou da esquerda serão acionados.

O desenvolvimento inicial permitiu testar a estrutura, os motores de corrente contínua e os sensores ultrassônicos. Para controle dos motores foi utilizado o *Arduino Motor Shield*. A comunicação foi feita por *bluetooth*, através da interface serial do Arduino e um aplicativo para dispositivos *mobile* enviarem e receberem dados por *bluetooth*.

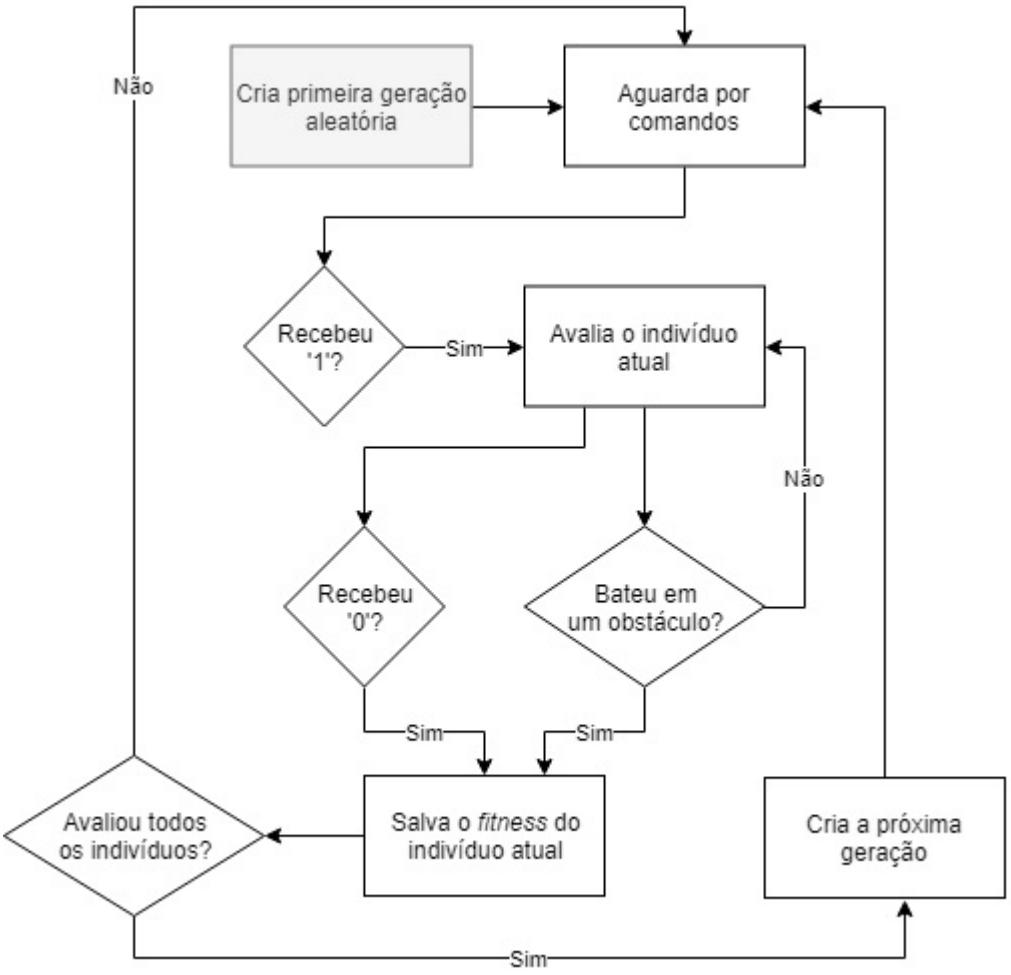
4.1.1 Algoritmo de Treinamento

O treinamento é feito através da avaliação do *fitness* de cada indivíduo. Os indivíduos são avaliados pelo seu tempo de vida — período em que o veículo navegou sem colidir com um obstáculo. Após todos os indivíduos serem avaliados, uma nova geração é criada e o processo é repetido.

A quantidade de gerações não é limitada, permitindo ao treinamento ser realizado por quantas gerações forem necessárias. Ao final do treinamento de cada indivíduo o *firmware* envia uma mensagem ao dispositivo de controle informando a geração e o indivíduo atuais, e o *fitness* do indivíduo.

A Figura 7 ilustra o processo de treinamento. Para avaliar um indivíduo é preciso voltar o veículo a sua posição inicial e enviar o caractere **1**, que é entendido pelo *firmware* como comando para iniciar o treinamento.

Figura 7 – Fluxograma do algoritmo de treinamento da rede neural.



Fonte: Próprio Autor

4.1.2 Desenvolvimento da Rede Neural

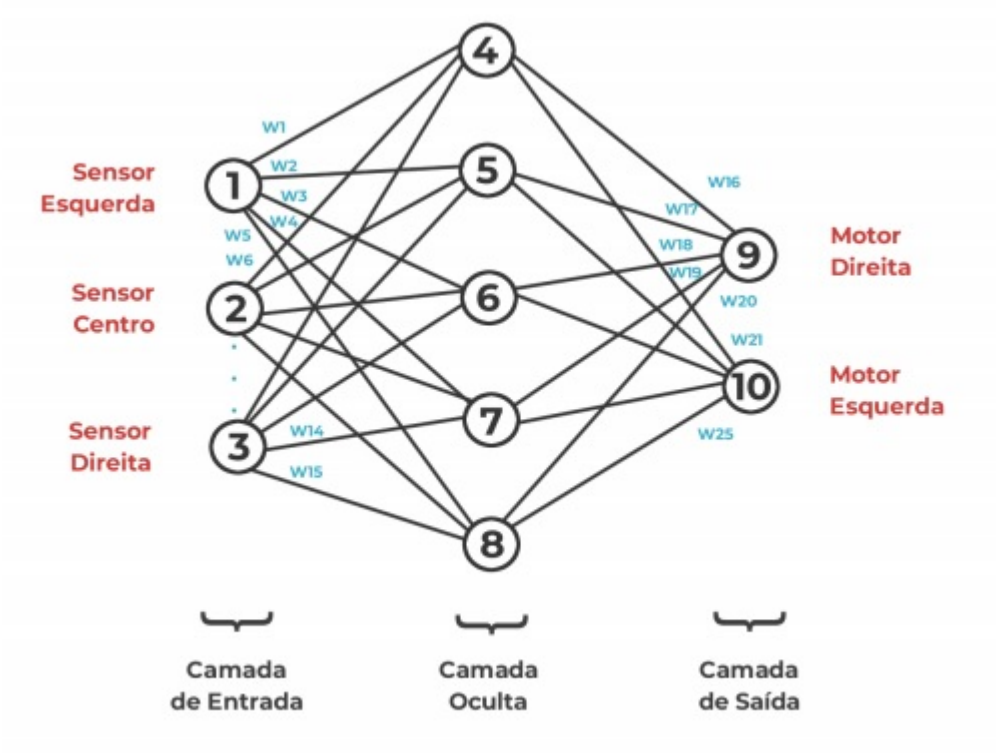
A rede neural implementada (Figura 8) é representada por um vetor de 25 posições. Cada posição contém o peso do enlace entre dois neurônios. Inicialmente as redes são geradas aleatoriamente com pesos aleatórios entre 0 e 1. Os pesos são ordenados no vetor partindo-se da camada de entrada, para a camada de saída.

A primeira camada da rede recebe a distância, em centímetros, entre os sensores e possíveis obstáculos, seguindo a ordem: sensor da esquerda, sensor da frente, e sensor da direita do veículo. A camada de saída é composta por dois neurônios, um para controlar os motores à direita do veículo, e um para controlar os motores à esquerda do veículo. Para esta rede neural foi utilizada uma camada oculta.

4.1.3 Algoritmo Genético

O algoritmo genético consiste na adaptação de seis indivíduos (redes artificiais representadas pelo vetor) gerados inicialmente aleatoriamente e armazenados em uma

Figura 8 – Rede neural artificial proposta para desvio de obstáculos.



Fonte: Próprio Autor

matriz de tamanho 6x25. As linhas da matriz representam os indivíduos e as colunas os pesos sinápticos. O objetivo do algoritmo genético é encontrar a combinação de pesos que leva ao maior tempo de vida.

Os dois melhores indivíduos – que possuem maior tempo de vida – são selecionados por seleção natural para cruzamento, gerando dois filhos. o terceiro e o quarto melhores indivíduos têm uma chance de 10% de sofrerem mutação. A próxima geração é formada pelos dois filhos, os dois pais selecionados e os dois indivíduos que podem ter sofrido mutação.

O cruzamento é feito por recombinação aritmética simples. É sorteada uma posição do vetor na qual os próximos valores serão a média aritmética entre os pesos dos dois pais nas respectivas posições.

Por exemplo, considerando dois pais cujos pesos são:

Pai 01: { 0.30 – 0.55 – 0.49 – 0.65 – 0.24 }

Pai 02: { 0.99 – 0.64 – 0.37 – 0.28 – 0.01 }

Caso ocorra uma recombinação na posição 3, são criados os filhos:

Filho 01: { 0.30 – 0.55 – 0.49 | 0.46 – 0.24 }

Filho 02: { 0.99 – 0.64 – 0.37 | 0.46 – 0.12 }

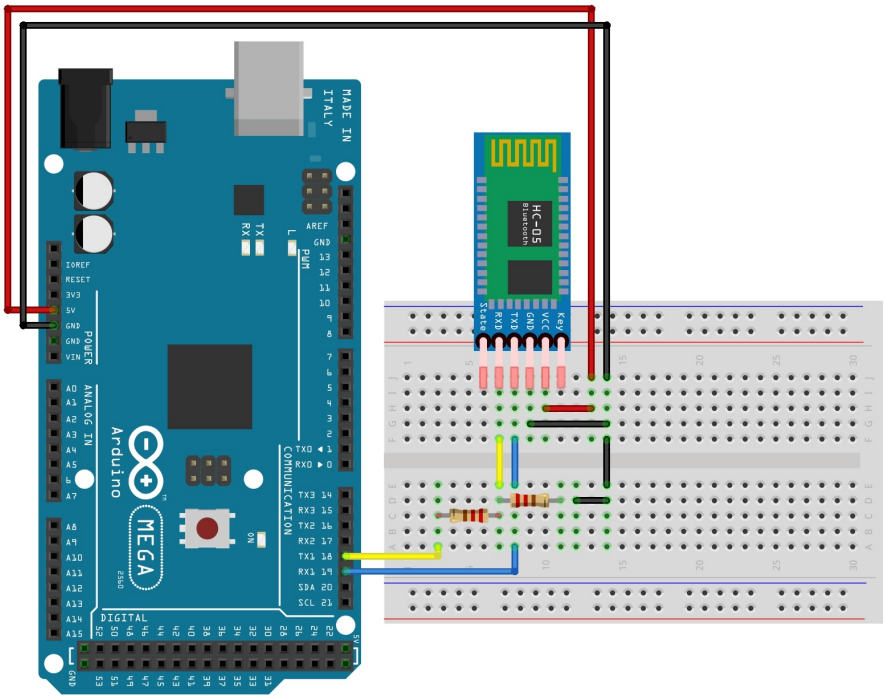
O Filho 01 herda os primeiros valores do Pai 01 até o ponto de recombinação, já o Filho 02 herda do Pai 02. Após o treinamento de cada indivíduo é enviado um log com a geração atual, o indivíduo e o tempo de execução. Se o veículo parar antes de bater contra um obstáculo é subtraído 10.000ms do tempo de execução, como penalidade para diminuir as chances de que ele seja selecionado para a próxima geração.

4.1.4 Módulo *Bluetooth* HC-06

O Módulo Bluetooth HC-06 permite a transmissão de dados por *bluetooth*, comunicação de longa distância sem fio. O módulo pode ser integrado ao Arduino, utilizando portas seriais para entrada/saída de dados, que podem ser tanto disponibilizadas a mais em algumas placas ou criadas em *software* sobre os pinos digitais do arduino com a biblioteca SoftwareSerial.h.

Os pinos *RXD* e *TXD* devem ser conectados a uma entrada *TX* e *RX* do arduino, respectivamente. O módulo deve ser alimentado com 3.3V a 5V. A Figura 9 mostra as conexões entre o Arduino e o módulo *bluetooth*.

Figura 9 – Esquemático da conexão entre o Módulo HC-06 e a placa Arduino.



Fonte: Próprio Autor

No desenvolvimento inicial o módulo foi utilizado para acionar o veículo autônomo dada a população atual do algoritmo genético. Caso o caractere **1** fosse enviado ao Arduino, o veículo inicia a navegação utilizando a rede neural do indivíduo atual. Se o veículo parar antes de atingir um obstáculo o processo de avaliação pode ser interrompido ao enviar o caractere **0** para o Arduino.

4.1.5 Resultados do Desenvolvimento Inicial com Arduino

A Tabela 2 mostra o tempo de vida dos indivíduos da 10ª geração. Os indivíduos 4, 5 e 6 têm tempo de vida negativo porque o veículo optou por não acionar nenhum dos motores antes de colidir. Ao subtrair 10.000ms do seu tempo de vida, a probabilidade destes indivíduos serem escolhidos para a próxima geração e para cruzamento é menor que os indivíduos 1, 2 e 3.

Tabela 2 – Tempo de vida dos indivíduos após 10 gerações.

Indivíduo	**Tempo de Vida**
1	2443 ms
2	3195 ms
3	3081 ms
4	-6913 ms
5	-7218 ms
6	-8097 ms

Fonte: Próprio Autor

Para a rede proposta não houveram resultados satisfatórios, pois não pode ser observado progresso significativo no tempo de vida dos indivíduos a cada geração. Entretanto o comportamento do veículo foi alterado conforme a escolha da função de ativação e da quantidade de neurônios na camada oculta, o que indica que a rede artificial necessita de uma melhor modelagem para o problema de desvio de obstáculos.

Outro fator que influencia no treinamento da rede é a quantidade de indivíduos de cada geração. Indivíduos ruins – com tempo negativo – são frequentemente selecionados devido à população pequena.

4.1.6 Sumário do Capítulo

O desenvolvimento inicial em Arduino possibilitou integrar técnicas de inteligência artificial no contexto de robótica. Podendo-se verificar o funcionamento do algoritmo genético na transformação dos pesos da rede, e do funcionamento da rede neural artificial como método de tomada de decisão para veículos autônomos.

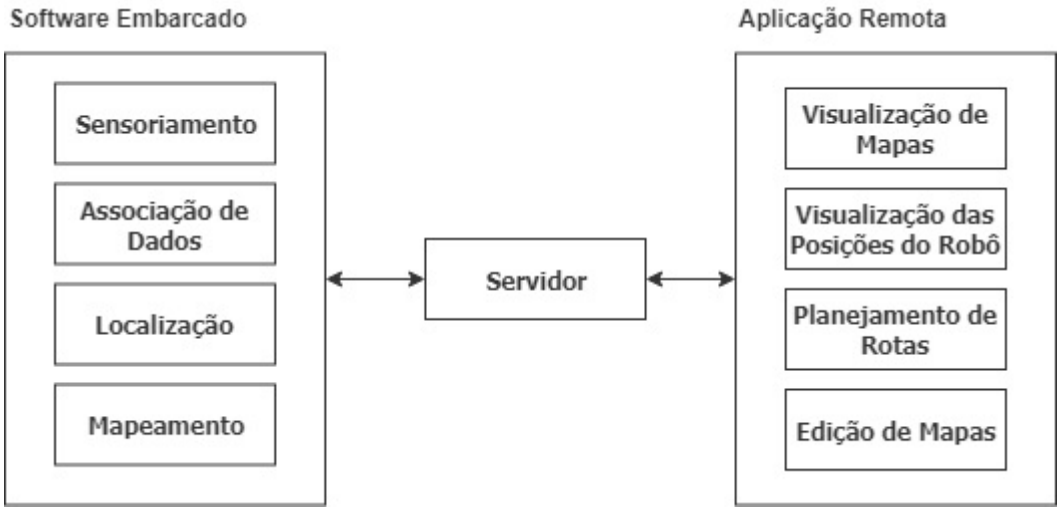
De acordo com Russel e Peter (RUSSEL; NORVIG, 2003), o mundo real é inacessível aos robôs devido a imperfeições no sensores, limitando-os a estímulos propícios a erros. O que pode ser observado neste trabalho de acordo com a dificuldade em se obter leituras constantemente fiéis dos sensores ultrassônicos e remover aquelas que atrapalhariam o treinamento da rede.

Para trabalhos futuros seria interessante o desenvolvimento de uma biblioteca na linguagem Arduino para lidar com redes e populações de tamanho arbitrário, para que se possa testar diversas configurações redes neurais para o mesmo problema.

4.2 Arquitetura do Sistema

O sistema foi desenvolvido em três partes (Figura 10): *software* embarcado; aplicação de controle; e servidor remoto. O *software* embarcado é toda parte lógica presente no veículo autônomo. A aplicação de controle é toda parte lógica feita por dispositivos remotos que se conectam ao veículo. As principais partes do sistema do *software* embarcado são os módulos de localização e de mapeamento, mas para que estes módulos funcionem são necessários os módulos de sensoriamento e de associação de dados. A aplicação remota possibilita visualizar tanto os mapas criados pelo veículo, quanto as posições do veículo. Na aplicação remota também é possível realizar simulações de planejamento de rotas e editar mapas.

Figura 10 – Arquitetura do Sistema.

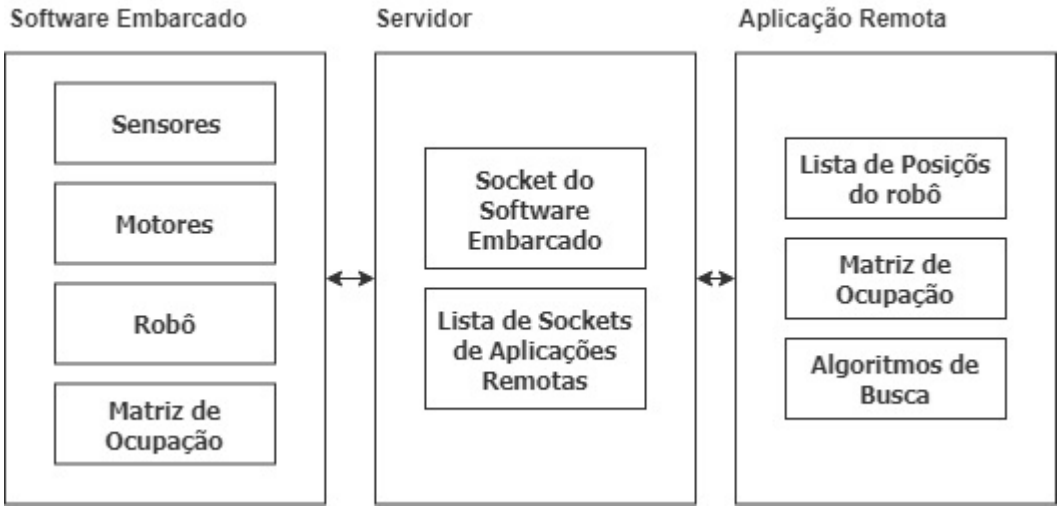


Fonte: Próprio Autor

Os componentes do sistema (Figura 11) foram projetados para implementar todas as funcionalidades do sistema presentes na arquitetura (Figura 10). No *software* embarcado, um conjunto de sensores é responsável pelo sensoriamento do ambiente, enquanto que um conjunto de motores possibilita a navegação em duas dimensões. O componente robô agrega os sensores e motores do sistema, e é capaz de estimar a sua posição de acordo com as leituras dos sensores. A matriz de ocupação, presente tanto no *software* embarcado quanto na aplicação remota é a representação do mapa de duas dimensões, utilizada durante o mapeamento.

O servidor mantém um *socket* para o veículo autônomo e uma lista de *sockets* para as aplicações remotas. Portanto mais de um cliente é permitido para visualizar os mapas criados pelo veículo durante a navegação. Na aplicação remota, o módulo de Algoritmos de Busca fornece a base para o sistema de roteamento.

Figura 11 – Componentes do Sistema.



Fonte: Próprio Autor

4.3 Projeto do Protótipo do Veículo Autônomo

Após o desenvolvimento em Arduino, a plataforma de prototipagem foi alterada para um Raspberry Pi 3. O desenvolvimento inicial em Arduino não gerou resultados positivos quanto ao treinamento da rede neural artificial para desvio de obstáculos, mas garantiu a fundamentação necessária para utilizar motores de corrente contínua e sensores ultrassônicos em plataformas de prototipagem. O Raspberry Pi, além de servir como um microcontrolador como o Arduino, pode ser utilizado como um microcomputador, oferecendo vantagens como sistemas operacionais linux, conexão Wi-Fi e Ethernet, além de possibilitar a programação dos GPIOs em diversas linguagens de programação.

O desenvolvimento do protótipo foi feito através da implementação de pequenos dispositivos, seguido de testes de unidade para cada dispositivo implementado. O *firmware* – composto pelo conjunto de todos os dispositivos implementados – foi desenvolvido após a implementação e os testes de cada componente, seguido de um teste de integração. O teste de integração executa comandos para os motores e obtêm leituras dos sensores simultâneamente para verificar se não existem erros no sistema ao integrar os componentes em um único sistema. O Raspberry Pi foi capaz de suportar a carga de dispositivos conectados ao GPIO com sucesso, sendo limitado apenas quanto à tensão de operação do GPIO de 3.3V.

O único sensor que não é capaz de fornecer nível lógico alto em 3.3V é o sensor ultrassônico HC-SR04. Para isto foi desenvolvido um divisor de tensão para limitar a tensão fornecida pelo sensor a 3.3V (Seção 4.3.2). Os demais componentes puderam ser integrados ao Raspberry sem modificações adicionais ao circuito. Alguns componentes, como a unidade de medição inercial MPU-9250 (Seção 4.3.4) possuem implementações *open-*

source disponíveis em Python[1] que facilitam a integração do sensor ao microcomputador através de APIs.

O veículo foi desenvolvido para comportar três sensores de distância, responsáveis pelo mapeamento e navegação; uma unidade de medição inercial, para detectar a orientação do veículo e realizar manobras simples como girar uma quantidade específica de graus para a esquerda ou para a direita; e uma chave óptica responsável pela odometria do robô, através da contagem do número de voltas das rodas do veículo. Foram utilizadas duas fontes de alimentação separadas: uma para o microcomputador, com 5V e corrente elétrica máxima de 2.1A; e uma fonte de 9V para os motores que fornece uma corrente elétrica de aproximadamente 93.5mA para cada motor. Os sensores são conectados aos pinos de 5V e *Ground* do Raspberry Pi, compartilhando da alimentação de 5V/2.1A e se beneficiam do circuito interno do Raspberry Pi para receber uma alimentação constante de 5V.

A estrutura do veículo é um chassi acrílico de dimensões 256 x 150 x 65mm, peso de 470g, com quatro rodas de plástico, cada uma acoplada a um *encoder* de velocidade e a um motor de corrente contínua de 6V que opera sob correntes elétricas de até 200mA. O *encoder* é utilizado em conjunto com a chave óptica para a contagem das revoluções das rodas (Seção 4.3.5). Este componente é um disco com 20 perfurações igualmente espaçadas, que acionam a chave óptica e incrementam a contagem de voltas do sensor. A velocidade dos motores sem carga informada pelo fabricante é de 200RPM quando operado em 6V; entretanto, para este projeto a velocidade especificada não foi uma boa aproximação para a velocidade real do veículo em tempo de execução.

4.3.1 Circuito de Potência

O veículo utiliza quatro motores de 6V para se movimentar, e é capaz de controlar cada motor independentemente. O controle foi feito com o módulo L298N, que permite o acionamento de até dois motores de corrente contínua ou um motor de passo, separa a alimentação do circuito de potência do circuito lógico e permite alterar o sentido de rotação do motor.

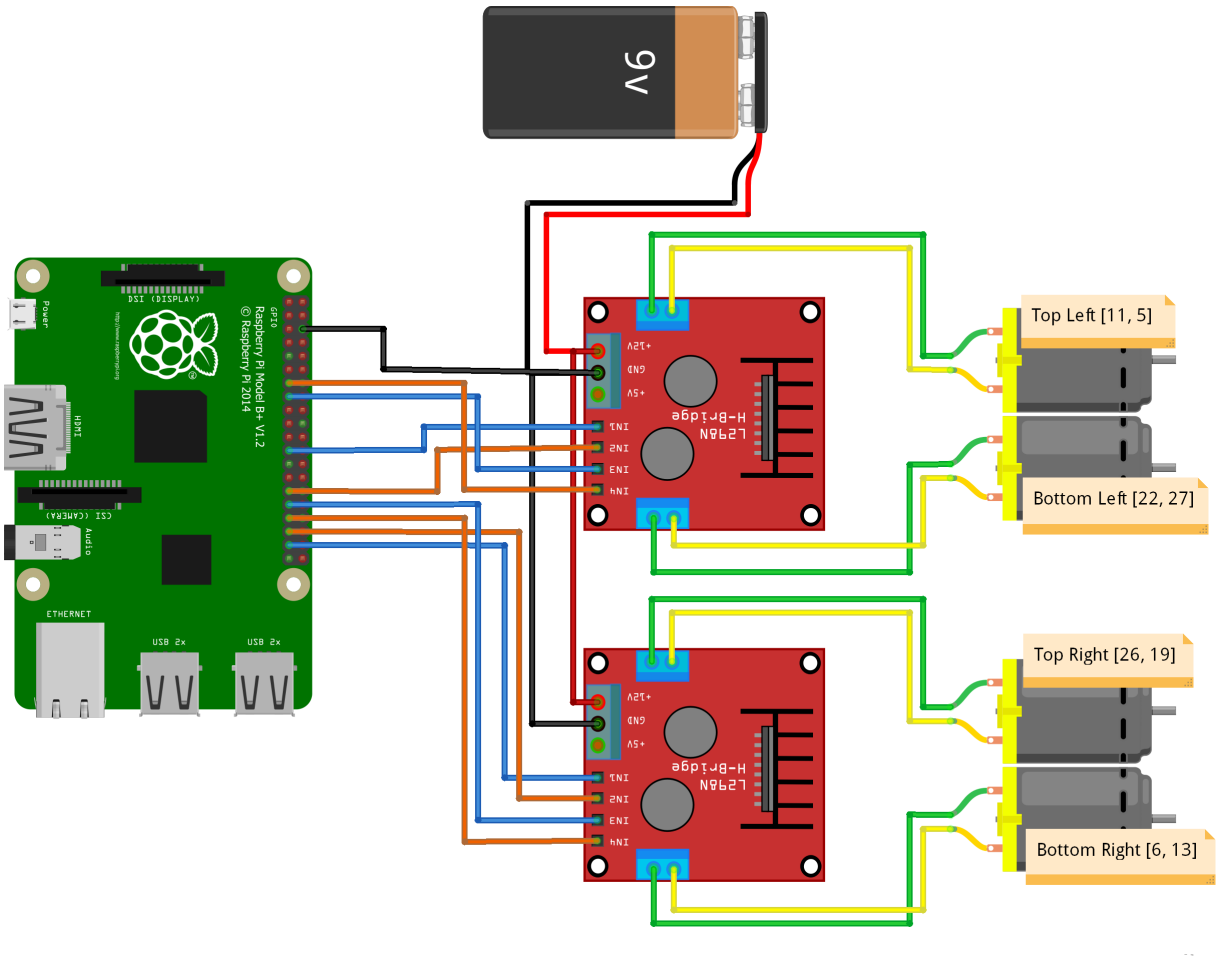
O circuito de potência utiliza dois módulos L298N, uma para controlar os motores da esquerda e outro para os motores da direita do veículo (Figura 12). Cada módulo possui quatro pinos (N1, N2, N3, N4) que controlam o sentido de rotação do motor, sendo os dois primeiros pinos para o Motor A, e os dois últimos para o Motor B.

A Figura 12 mostra o esquema do circuito de potência, exibindo os componentes utilizados no circuito e suas respectivas conexões.

Para simplificar o método de localização do veículo os motores são sempre acionados na mesma velocidade. Assim, para acionar um motor são utilizados os níveis lógicos do

[1] RTIMULib2: <https://github.com/richardstechnotes/RTIMULib2>. Acessado em Junho/2019.

Figura 12 – Esquema do circuito de potência.



Fonte: Próprio Autor

GPIO: *HIGH* ou *True* para o nível lógico alto, e *LOW* ou *False* para nível lógico baixo. A Tabela 3 mostra o controle de um motor de acordo com o nível lógico de cada entrada:

Tabela 3 – Acionamento de um motor de corrente contínua com o módulo L298N.

	Parado	**Sentido Horário**	**Sentido Anti-Horário**
N1	LOW	HIGH	LOW
N2	LOW	LOW	HIGH

Fonte: Próprio Autor

Os motores podem ser ativados e desativados por *software* ou pelos *jumpers* situados ao lado dos pinos de controle do módulo. Como não há necessidade de desativar nenhum motor, os *jumpers* foram conectados para que os motores ficassem ativados enquanto os módulos estiverem conectados à fonte externa.

Os módulos L298N foram implementados em *software* na classe *Motor*. Esta classe define os GPIOs do Raspberry utilizados para um único motor. Foram criados métodos para acionar o motor no sentido horário, anti-horário e para desligar o motor.

Após a implementação do controle de cada motor, foi desenvolvida a classe *Con-*

troller para manipular os quatro motores simultâneamente. Nesta classe foram criados métodos para que o veículo possa andar para frente, para trás, parar os motores, girar para a esquerda e girar para a direita. Estes métodos podem ser utilizados com *timers* (por exemplo: girar para a esquerda por 500ms), ou apenas como comandos, que definem uma ação para os motores até que outro comando seja recebido.

4.3.2 Sensor de Distância Ultrassônico

Foram utilizados dois sensores ultrassônicos, o HC-SR04 e o Ping Parallax. Os sensores captam medidas de distância entre o veículo e obstáculos que são interpretadas tanto para localização quanto para mapeamento.

O sensor HC-SR04 foi colocado no centro à esquerda do protótipo e tem como função principal auxiliar na tomada de decisões do veículo (por exemplo, se há espaço para virar para a esquerda). O sensor Ping foi colocado na frente do veículo para verificar se a próxima célula está livre.

Os dois sensores ultrassônicos possuem o mesmo princípio de funcionamento, porém se diferem quanto à pinagem. No HC-SR04 existem dois pinos de entrada e saída, o *trigger* e o *echo*, enquanto que no Ping há apenas um tanto para entrada quanto para saída. A diferença entre os sensores foi abstraída em *software*, para simplificar o uso independente dos pinos de entrada.

Na implementação dos sensores ultrassônicos em *software* foi utilizado o padrão de projeto *Factory Method*, que abstrai a criação de objetos. Este padrão define uma classe capaz de decidir qual objeto deve ser instanciado para os parâmetros recebidos. Para instanciar um sensor ultrassônico é preciso passar os pinos em que os sensores estão conectados ao Raspberry Pi como parâmetro:

Quadro 1 – *Factory Method* para instanciamento de sonares.

```
1 sonar1 = Sonar.create(10);
2 sonar2 = Sonar.create(4, 17);
```

No exemplo do Quadro 1, o *sonar1* será um sensor HC-SR04, enquanto que o *sonar2* será um Ping. Este padrão de projeto melhora a legibilidade do código, além de tornar as classes dos sensores reutilizáveis para projetos futuros.

Ambos os sensores operam em 5V, entretanto o sensor HC-SR04 envia sinais de até 5V pelo pino *echo* para o microcontrolador. Os GPIO do Raspberry Pi operam em 3.3V, portanto conectar o pino *echo* do HC-SR04 poderia ocasionar problemas futuros no GPIO do Raspberry. Para diminuir a tensão de saída do *echo* foi utilizado um divisor de tensão.

O divisor de tensão é composto por três resistores de $1k\Omega$, um resistor entre o pino *echo* e o GPIO do Raspberry, e dois resistores entre a porta e o *ground*. Utilizando a Equação 3.2, pode-se calcular a razão entre os resistores necessária para transformar os 5V do sensor para os 3.3V utilizados pelo Raspberry Pi.

$$V_{out} = \frac{V_{in} * R_2}{R_1 + R_2} \tag{4.1}$$

$$\frac{R_2}{R_1 + R_2} = \frac{V_{out}}{V_{in}} \tag{4.2}$$

Utilizando um resistor de $1k\Omega$ para R_1, é possível calcular o valor de R_2 necessário para o circuito:

$$\frac{R_2}{1000 + R_2} = \frac{3.3}{5} \tag{4.3}$$

$$R_2 = 0.66 * (1000 + R_2) \tag{4.4}$$

$$R_2 = 660 + 0.66 * R_2 \tag{4.5}$$

$$0.34 * R_2 = 660 \tag{4.6}$$

$$R_2 \approx 1941.18 \tag{4.7}$$

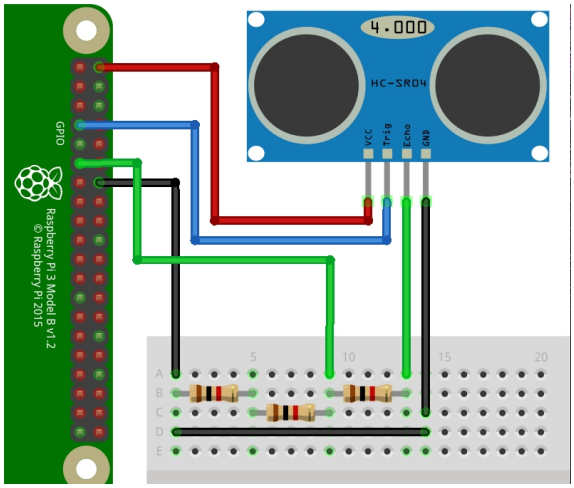
O valor mais próximo de 1941.18Ω (Equação 4.7) disponível é um resistor de $2k\Omega$, que pode também ser obtido conectando-se dois ou mais resistores em série. No circuito do HC-SR04, foram utilizados dois resistores de $1k\Omega$ em série.

4.3.3 Sensor de Proximidade Infravermelho

Foi utilizado um sensor de distância infravermelho E18-D80NK ao lado direito do veículo no centro. A função deste sensor é igual ao do sensor HC-SR04, colocado ao lado esquerdo do veículo, auxiliar na tomada de decisões e na etapa de mapeamento. O alcance do sensor é ajustável através de um potênciometro, o alcance escolhido foi de aproximadamente 30cm, de modo que se o veículo escolher girar para esquerda e ir para frente, há espaço suficiente para ele.

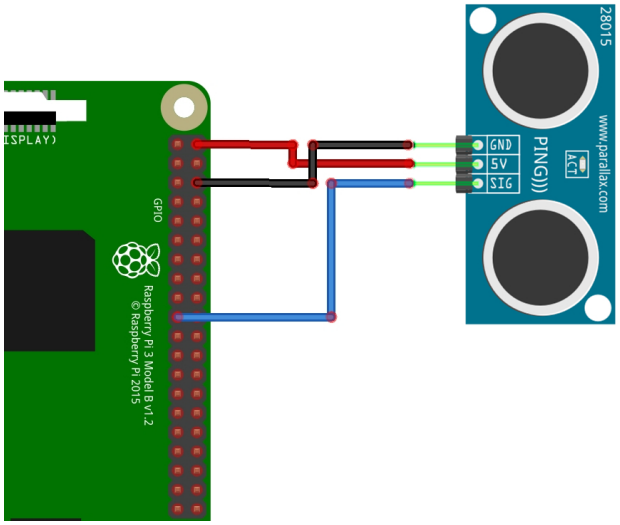
Em constraste com os sensores ultrassônicos, o sensor E18-D80NK não é capaz de fornecer medidas de tempo ou distância. A informação passada para o GPIO do

Figura 13 – Esquema do circuito do sensor ultrassônico HC-SR04 com divisor de potência.



Fonte: Próprio Autor

Figura 14 – Esquema do circuito do sensor ultrassônico Parallax Ping.

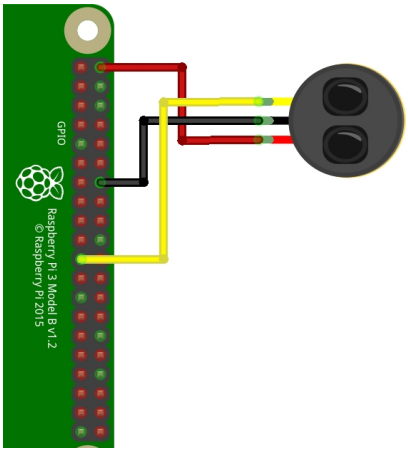


Fonte: Próprio Autor

microcomputador é apenas se existe ou não um obstáculo na faixa de alcance do sensor. Esta informação é passada ao algoritmo de associação de dados (Seção 4.7) junto do alcance do sensor para que o mapa seja corretamente atualizado.

O sensor E18-D80NK opera sob uma tensão de 5V fornecida pelo Raspberry Pi. É necessário apenas um pino digital para transmitir a informação para o microcontrolador. O sinal assume o valor de nível lógico alto se não houver um obstáculo no alcance do sensor, ou de nível lógico baixo caso exista um obstáculo. A Figura 15 mostra o esquema da conexão entre o sensor e o Raspberry Pi, utilizando o mesmo pino digital que foi utilizado no projeto.

Figura 15 – Esquema do circuito do sensor de proximidade infravermelho E18-D80NK.



Fonte: Próprio Autor

4.3.4 Unidade de Medição Inercial

A unidade de medição inercial utilizada neste projeto foi o MPU-9250. Este sensor contém um giroscópio, um acelerômetro e um magnetômetro, o que possibilita calcular a inclinação do sensor em relação a três eixos ortogonais. Nos projetos de iniciação científica foi utilizado o MPU-6500 que possui um giroscópio e um acelerômetro, entretanto com este MPU não é possível medir o ângulo de *yaw*, que é o ângulo que determina a orientação de um objeto em relação ao plano *xy*.

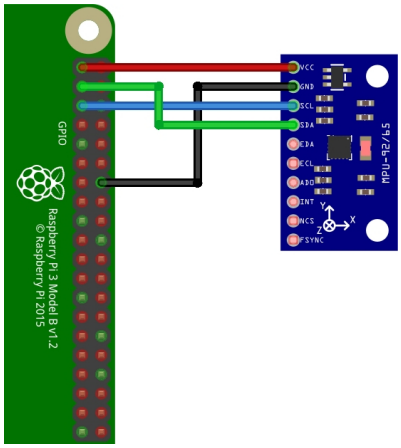
A Figura 16 mostra o esquema de conexão entre o MPU-9250 e o Raspberry Pi. O sensor é alimentado com os pinos de 3.3V e GND do Raspberry, e utiliza os pinos SDA e SDL que possibilitam a comunicação i2C com o microcomputador. Para receber os dados do IMU e calcular o ângulo de *yaw* foi utilizada a biblioteca RTIMULib2 que abstrai o processo de obtenção dos *bytes* através do protocolo i2C e o cálculo dos ângulos de *pitch*, *roll* e *yaw*. Como neste projeto apenas o ângulo de *yaw* é utilizado, foi criada uma classe em Python para mediar os dados obtidos pela biblioteca RTIMULib2 e os dados necessários à aplicação.

O sensor possui outros pinos que fornecem funcionalidades adicionais, como conectar mais sensores através das portas SDA e SDL, medir temperatura, e interromper os pinos de saída digital. Estas funcionalidades não foram necessárias ao projeto, portanto os pinos *EDA*, *ECL*, *AD0*, *INT*, *NCS* e *FSYNC* não precisam ser conectados ao Raspberry Pi.

4.3.5 Chave Óptica

A chave óptica é o sensor responsável pela odometria, com este sensor foi possível estimar a distância percorrida pelo veículo através do número de revoluções de suas rodas. O sensor utilizado foi o FC-03, colocado na roda superior direita do veículo. O sensor possui dois pinos de entrada, um digital e um analógico, que fornecem informação se o

Figura 16 – Esquema do circuito da unidade de medição inercial MPU-9250.



Fonte: Próprio Autor

raio infravermelho da chave óptica está interrompido por um obstáculo ou não. Para o protótipo foi utilizado o pino digital conforme a Figura 17. Ambos os pinos fornecem a mesma informação.

O *software* embarcado foi desenvolvido para que quando a chave óptica trocar de estado, um contador de voltas seja incrementado. Como o disco *encoder* possui mais de uma perfuração, uma revolução da roda do veículo ocorre quando o contador registra a mesma quantidade de furos no *encoder*. As informações sobre a circunferência da roda e o número de perfurações do encoder são fornecidas pelo arquivo de configuração4.3.6.

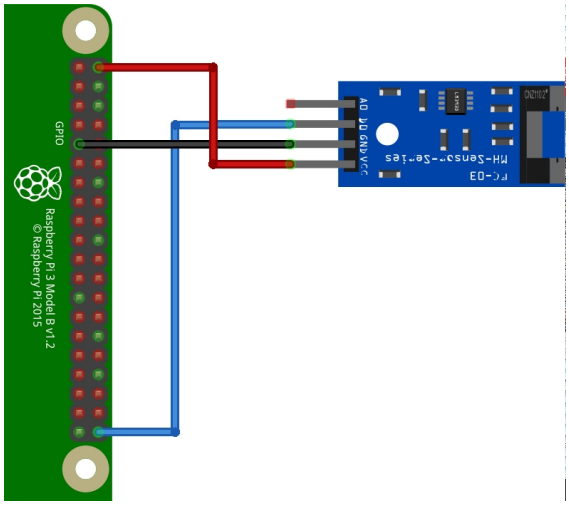
Em *software* a chave óptica foi implementada na classe *Odometer*. Esta classe foi desenvolvida para manipular o contador de acordo com as especificações da roda e do *encoder*. Assim o *software* embarcado pode fazer requisições para iniciar a contagem e para receber a distância percorrida, permitindo ao veículo andar por uma distância específica. Esta função é essencial tanto para a localização quanto para o mapeamento, pois possibilita ao robô estimar qual a sua nova posição após navegar em uma determinada direção.

4.3.6 Arquivo de Configuração

Para que o código do *sofware* embarcado não ficasse sempre restrito aos mesmos GPIOs e aos mesmos componentes utilizados neste projeto, as informações dos pinos e dos componentes foram separadas em um arquivo de configuração. Deste modo, caso seja preciso alterar a conexão de um disposito com o Raspberry, é preciso atualizar apenas o arquivo de configuração, ao invés do código-fonte.

O arquivo de configuração contém um objeto JSON com informação dos sensores, dos motores e dos parâmetros utilizados para a localização e mapeamento, como a resolução do mapa em centímetros e informações sobre odometria (circunferência da roda e número de perfurações do *decoder*). O arquivo informa também qual sistema de numeração dos

Figura 17 – Esquema do circuito da chave óptica.



Fonte: Próprio Autor

GPIO foi utilizado, a dimensão inicial do mapa e o endereço do servidor remoto.

Ao iniciar o programa, o *software* embarcado lê o arquivo de configuração e instancia os sensores com a pinagem correta de entrada e saída, informando no terminal qual dispositivo está sendo configurado. Caso haja algum erro, como formato inválido ou informações insuficientes, a execução do *software* embarcado é interrompida.

4.4 Desenvolvimento do Software de Controle

Foi desenvolvida uma aplicação remota para possibilitar a visualização do mapa em tempo real durante a navegação do robô. A aplição foi desenvolvida com tecnologia *web*, Javascript, HTML e CSS. Para exibir corretamente o mapa, a aplicação recebe informações sobre o estado atual do robô e atualiza o mapa com um algoritmo similar ao algoritmo de mapeamento implementado no *software* embarcado (Seção 4.9). As informações são recebidas através de uma WebSocket conectada ao servidor remoto e identificada como uma aplicação de visualização. *Sockets* reconhecidos como aplicações de visualização pelo servidor são atualizadas sempre que novas informações sobre o robô são obtidas. Para saber quando novos dados chegam pelo *socket*, aplicações em Javascript, como o servidor e o aplicativo de visualização, utilizam o paradigma de programação orientada a eventos. Outros paradigmas suportados por Javascript como orientação a objetos e funcional também foram necessários para a implementação do aplicativo de visualização.

A orientação em eventos se baseia na execução de métodos pré-definidos quando um determinado evento ocorre. Por exemplo, quando o *WebSocket* do robô mantido pelo servidor recebe o sinal que um evento do tipo *message* foi disparado, o servidor sabe que uma nova mensagem está disponível. Assim, para ler a mensagem é preciso apenas atribuir ao evento *message* do socket do robô uma função que recebe como parâmetro uma *string*.

Em Javascript, a orientação a eventos está ligada ao paradigma funcional, pois os eventos são programados pela atribuição de funções dentro do escopo do método do evento.

O trecho de código abaixo demonstra o uso de eventos pelo aplicativo de visualização. A WebSocket do cliente se inscreve nos eventos *onopen*, *onmessage*, *onerror* e *onclose* que são disparados sempre que uma nova conexão é estabelecida com sucesso, quando uma nova mensagem é recebida, quando um erro de conexão ocorre e quando a conexão é encerrada, respectivamente. A implementação de *WebSockets* em HTML/Javascript[2] fornece os eventos necessários para programação de aplicações em rede, cabe ao programador decidir quais eventos utilizar. Outros eventos como *onerror* também são disponíveis.

No aplicativo de visualização desenvolvido, sempre que um evento do tipo *onmessage* é disparado pelo WebSocket, a aplicação tenta transformar os dados da mensagem em um objeto JSON. Se a mensagem for um objeto válido, a aplicação atualiza o mapa e a posição atual do veículo para visualização. Foi também implementado um formulário para especificar os parâmetros da WebSocket do servidor e para iniciar uma nova conexão. Deste modo, o servidor pode ser executado em outra máquina, garantindo maior flexibilidade para o sistema. O endereço de IP e o porto do servidor são utilizados para criar um novo WebSocket que gerenciará a conexão.

A visualização do mapa utiliza o elemento *canvas*, suportado em HTML5[3]. O elemento *canvas* pode ser acessado em Javascript através da manipulação do DOM (*Document Object Model*), fornecendo ferramentas para renderizar elementos como formas geométricas básicas, texto e imagens, além de permitir inserir lógica adicional em Javascript como estruturas condicionais e laços de repetição. No desenvolvimento do aplicativo de visualização, cada célula do mapa é desenhada no canvas como um quadrado. A célula é preenchida de cinza caso o robô esteja incerto do estado da célula; a célula é preenchida de preto caso a área esteja ocupada, ou de branco caso a área esteja livre.

O mapa foi desenvolvido utilizando o padrão de protótipos do Javascript. Protótipos são uma maneira de definir classes de objetos. Ao se criar um mapa utilizando o padrão *prototype*, outros mapas podem ser instanciados utilizando o primeiro como referência. A vantagem deste modo de programação é que os novos clones do protótipo podem ter métodos e atributos exclusivos sem a necessidade de criar uma classe para representar o novo protótipo. Este método foi escolhido porque o conceito de classes e instanciamento de objetos em Javascript é relativamente novo – introduzido em 2015 no ECMAScript 6[4] – portanto existem *browsers* que não suportam o uso de classes em Javascript.

O construtor do protótipo do mapa recebe como parâmetros a matriz inicial, a

[2] HTML Living Standart: <https://html.spec.whatwg.org/multipage/web-sockets.html>. Acesso em Junho/2019.

[3] Canvas API: <https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API>. Acessado em Junho/2019.

[4] ECMAScript 6: <https://www.w3schools.com/js/js_es6.asp>. Acessado em Junho/2019.

resolução do mapa e a referência para o elemento canvas. A matriz inicial é um mapa cujas células são preenchidas por 0.5, pois não há nenhuma informação sobre o mapeamento. A resolução é o tamanho de cada célula em centímetros. A referência do canvas é utilizada para obter o contexto 2D, com o contexto é possível desenhar o mapa na página web.

O mapa possui dois métodos que são chamados sempre que a aplicação recebe novas informações do veículo, o método *update* e o método *draw*. O *update* atualiza o mapa com as novas informações do veículo, modifica o valor das células alcançadas pelos sensores e atualiza a posição atual do robô. No final do método *update* é chamado o método *draw* que desenha o mapa e a posição atual do veículo na página.

Diferentemente do *software* embarcado, a aplicação de visualização mantém todas as posições anteriores do robô salvas, e as utiliza para desenhar o caminho percorrido. O caminho foi representado por linhas vermelhas que vão do centro de uma célula à outra.

Para possibilitar a melhor visualização de mapas grandes, foi criado o método *zoom*. Utilizando a programação orientada a eventos do javascript, pode-se adicionar uma função para quando o evento '*wheel*' é disparado. Este evento permite detectar quando a roda do mouse é utilizada. A função assinalada ao evento pode receber como parâmetro uma referência para o evento disparado, que por sua vez possui o atributo *deltaY*. Se *deltaY* for positivo, a roda do *mouse* foi movida para frente, se for negativo, a roda foi movida para trás. Com esta informação foi possível aumentar e diminuir o tamanho do mapa ao multiplicar o valor de cada célula pela variável *zoom*.

Por fim, o último evento necessário para que a aplicação funcionasse corretamente foi o *onload*. Este evento é disparado quando a página *web* for completamente carregada. Utilizando este evento, erros como referências a variáveis e protótipos indefinidas são evitados, pois todos os *scripts* estarão carregados na memória. Este evento pertence ao objeto *window* que fornece detalhes sobre a página *web* dentro do *browser*, como largura, comprimento, estado da página, informações sobre dispositivos de entrada e saída e diversos eventos.

4.5 Desenvolvimento da Comunicação entre Protótipo e Software de Controle

A comunicação entre o veículo e a aplicação de visualização foi feita através do protocolo *WebSockets*. Este protocolo foi escolhido pela simplicidade da implementação em diferentes linguagens. O *WebSocket* também abstrai a camada de rede, possibilitando a prototipagem rápida de aplicações em rede.

As mensagens enviadas são objetos no formato JSON que são transformados em *string* e posteriormente em *bytes* pelo *WebSocket*. Assim as aplicações trabalham com

objetos com alto nível de abstração (JSON para Javascript e dicionário para Python), enquanto que a camada de transporte é capaz de serializar qualquer tipo de informação recebida.

A linguagem de programação Python possui suporte nativo para objetos JSON, permitindo transformar dicionários em JSON e objetos JSON em dicionários Python. Já em Javascript JSON é a notação utilizada para representar qualquer objeto. Portanto utilizar JSON na comunicação entre os dispositivos garante flexibilidade no formato das mensagens e permitiu desenvolver as aplicações em alto nível, sem a necessidade de desenvolver um *parser* de uma linguagem para outra, ou de um protocolo de comunicação em baixo nível.

Uma mensagem é composta de um comando e um conjunto de dados. O comando é a ação que a aplicação deseja que o servidor execute e os dados são os parâmetros necessários para a execução do comando. Por exemplo, a aplicação de visualização envia o comando *start_robot* para iniciar o processo de mapeamento e navegação, neste comando não é passado nenhum argumento, logo a mensagem é um comando como mostra o Quadro 2.

Quadro 2 – Mensagem enviada pelo *software* de controle remoto para iniciar o SLAM.

```
{ "command": "start_robot" }
```

Fonte: Próprio Autor

Já o veículo precisa de enviar comandos mais detalhados ao aplicativo de visualização. Inicialmente, o comando para atualizar o mapa enviava a posição atual do robô e o mapa atual. A mensagem resultante é semelhante ao objeto do Quadro 3:

Entretanto, enviar o mapa para cada iteração do algoritmo de mapeamento e navegação se torna custoso, uma vez que as dimensões do mapa aumentam durante a etapa de mapeamento. Para representar um mapa de $10m^2$ com uma resolução de 10cm por célula, seria necessário uma matriz com 100 linhas e 100 colunas. Como números em JavaScript são sempre representados com precisão dupla – padrão IEEE 754[5] – cada posição da matriz utiliza 8 bytes. Um mapa de $10m^2$ gastaria pelo menos 0.08 Mb (100 linhas $\times$ 100 colunas $\times$ 8 bytes).

Considerando um mapa dez vezes maior que o anterior, seria necessário uma matriz de 1000 linhas e 1000 colunas para representá-lo com grades de ocupação, considerando uma resolução de 10cm por célula. O tamanho do mapa seria de 8 Mb (1000 linhas $\times$ 1000 colunas $\times$ 8 bytes). Pode-se observar que o tamanho da representação do mapa cresce

[5] Números em Javascript: <https://www.w3schools.com/js/js_numbers.asp>. Acessado em Junho/2019.

Quadro 3 – Mensagem de atualização inicial enviada pelo veículo ao servidor.

```
{
    "command": "new_robot_data",
    "map": [
     [0.5, 0.5, 0.5, 0.5, 0.5],
     [0.5, 0.5, 0.5, 0.5, 0.5],
     [0.5, 0.5, 0.5, 0.5, 0.5],
     [0.5, 0.5, 0.5, 0.5, 0.5],
     [0.5, 0.5, 0.5, 0.5, 0.5],
    ],
    "robot": {
        "x": 4,
        "y": 4,
        "orientation": "X_POS"
    }
}
```

Fonte: Próprio Autor

polinomialmente em relação ao tamanho real do ambiente. Para resoluções menores do que 10cm o tamanho do mapa seria ainda maior.

A Unidade de Transmissão Máxima (MTU) de uma rede Wi-Fi local é de 1500 bytes. O MTU especifica a quantidade máxima de dados que um pacote pode conter na camada de transporte. Logo, para enviar o mapa de $100m^2$ do Raspberry Pi ao aplicativo de visualização seriam gastos no mínimo 5333 pacotes ($\lceil Tamanho\ do\ Mapa/MTU \rceil$).

Como o protocolo WebSockets é implementado sobre o protocolo TCP, uma quantidade grande de pacotes para uma aplicação de tempo real impacta consideravelmente na performance da aplicação. Portanto foi necessário reduzir o tamanho das mensagens enviadas pelo veículo.

O formato anterior foi alterado para enviar apenas as últimas leituras e a posição atual do robô. De modo que o tamanho da mensagem não aumentasse proporcionalmente ao tamanho do mapa.

Para exibir corretamente o mapa na aplicação de visualização, parte do algoritmo de mapeamento e associação de dados foi replicado para interpretar os dados dos sensores e atualizar corretamente o mapa. O campo *map_resize* foi adicionado para que as aplicações remotas atualizem o tamanho do mapa de acordo com o tamanho estabelecido pelo *software* embarcado. Para valores negativos o tamanho permanece o mesmo. A lista informa a quantidade de colunhas e de linhas adicionadas no início da matriz, e a quantidade de colunas e de linhas adicionadas no fim da matriz, respectivamente.

Quadro 4 – Mensagem de atualização enviada pelo veículo ao servidor.

```
{
    "command": "new_robot_data",
    "map_data": [
        [35, 44, true,  90],
        [43, 44, true, -90],
        [42, 49, true,   0]
    ],
    "map_resize": [-4, -3, 0, -18],
    "robot_pose": {
        "x": 44,
        "y": 42,
      "orientation": 90
    }
}
```

Fonte: Próprio Autor

Para estabelecer a comunicação entre o veículo e o aplicativo remoto foi criado um servidor remoto em Javascript utilizando a ferramenta NodeJS. O servidor é responsável por gerenciar as conexões e as mensagens recebidas. As conexões são armazenadas no servidor, cada conexão é distinguida entre *socket* do veículo e *socket* de aplicativo de visualização. Quando uma nova conexão por *WebSocket* é estabelecida o cliente se identifica ao servidor informando se é o robô ou uma aplicação de visualização. O servidor foi desenvolvido para armazenar uma lista de clientes para visualização, de modo a permitir que mais de um dispositivo possa acessar o mapa criado pelo veículo.

Quando o servidor recebe uma mensagem do *socket* do robô, a mensagem é repassada a todos os clientes da lista. Qualquer cliente pode se conectar ao servidor através do endereço IP e do porto utilizado pela WebSocket do servidor. Se o cliente for uma aplicação de visualização (Seção 4.4), o cliente irá interpretar os dados e atualizar o seu mapa. Do mesmo modo, todas as mensagens dos clientes são repassadas ao *socket* do robô. As mensagens devem estar no formato JSON ou serão descartadas pelo servidor antes de encaminhar ao próximo *socket*.

4.6 Projeto da Representação do Mapa

O mapa utilizado pelo veículo é uma grade de ocupação. Cada célula do mapa é representada por uma variável aleatória binária com uma medida real no intervalo [0, 1] que informa a probabilidade da célula estar ocupada por um obstáculo. A probabilidade

de cada célula pode ser expressa por uma Distribuição de Bernoulli (Equação 4.8), que demonstra se uma célula (m_{ij}) está ocupada ou não, dada a probabilidade.

$$f(m_{ij}) = \begin{cases} 1, & \text{se } m_{ij} > 0.5 \\ 0, & \text{se } m_{ij} < 0.5 \end{cases} \tag{4.8}$$

Como as grades de ocupação são mapas métricos, deve ser definida uma escala que represente a proporção entre o mapa exibido e a superfície real. Esta escala é expressa no mapa como o tamanho da célula, em centímetros.

Para este trabalho foi utilizada uma escala de 10cm. Quanto menor o valor de cada célula, maior a fidelidade do mapa à superfície. Entretanto escalas pequenas requerem movimentos e medidas mais precisas do que as disponíveis neste protótipo.

4.7 Desenvolvimento do Algoritmo de Associação de Dados

A associação de dados é o processo de relacionar as informações obtidas pelos sensores aos respectivos marcos do ambiente a fim de reconhecer marcos anteriores e atualizar sua posição em relação ao robô.

No protótipo desenvolvido neste projeto não há extração de marcos do ambiente, pois o mapa utilizado é uma grade de ocupação. Os marcos foram definidos como qualquer obstáculo detectado pelos sensores. Assim o processo de extração de marcos foi simplificado durante o processo de sensoriamento.

Após a extração das informações dos sensores é necessário que as medições sejam corretamente associadas à orientação do veículo. Esta etapa é necessária tanto para a construção do mapa quanto para a navegação.

A orientação do veículo é expressa pela inclinação do veículo em um plano cartesiano, sendo 0° a inclinação cuja frente do veículo está no sentido positivo do eixo das abscissas no mapa, e 90° no sentido negativo do eixo das coordenadas. A orientação é feita com referência ao ângulo inicial de *yaw* do robô, tomando-se qualquer ângulo inicial como 0°.

O algoritmo desenvolvido recebe como parâmetros a localização atual do veículo e as leituras dos sensores. Determina se a leitura encontrou algum obstáculo, e retorna a célula do mapa que corresponde a cada leitura. O algoritmo retorna uma lista de tuplas informando a posição alvo da leitura no mapa, a existência de um obstáculo e o ângulo da leitura. A implementação em Python do algoritmo de associação de dados é apresentada no Apêndice A.

Na associação de dados são então organizadas as leituras para a etapa de mapeamento, que incrementa os novos dados à grade de ocupação. É importante que a associação

de dados seja feita separado da etapa de mapeamento, pois caso existam mais leituras disponíveis, é necessario atualizar apenas como elas são interpretadas pelo robô, e não todo o algoritmo de mapeamento.

4.8 Desenvolvimento do Algoritmo de Localização

Localização é o método de determinar a posição e a orientação do robô em um determinado ambiente. Na robótica móvel este problema é geralmente modelado com base em um mapa no qual o robô escaneia em busca de marcos do ambiente e atualiza sua posição. Para este trabalho, a localização assume que os movimentos do veículo são exatos, utilizando os dados da odometria para atualizar a posição e a orientação do veículo.

No protótipo desenvolvido a localização do robô é baseada na posição anterior do robô e das ações tomadas a cada ponto no tempo. A posição inicial do robô é sempre considerada o centro do mapa, o qual é construído ao seu redor. O tempo foi discretizado com relação a cada iteração do algoritmo SLAM, de modo que o índice t informa a qual iteração determinada variável se refere.

A localização utiliza a posição anterior do robô s_{t-1} e os dados obtidos pelas informações dos sensores z_t para decidir qual controle u_t deverá ser executado. Os controles foram definidos como:

- z_1: Mover uma célula para frente.
- z_2: Girar 90° para a esquerda.
- z_3: Girar 90° para a direita.

As observações z_t são as leituras dos sensores ultrassônicos da frente e da esquerda, e do sensor infravermelho. Cada observação informa a distância medida pelo sensor e se um obstáculo foi detectado. São adicionados juntos à observação as características dos sensores: orientação do sensor em graus, a distância entre o sensor e o centro do veículo, e o alcance mínimo.

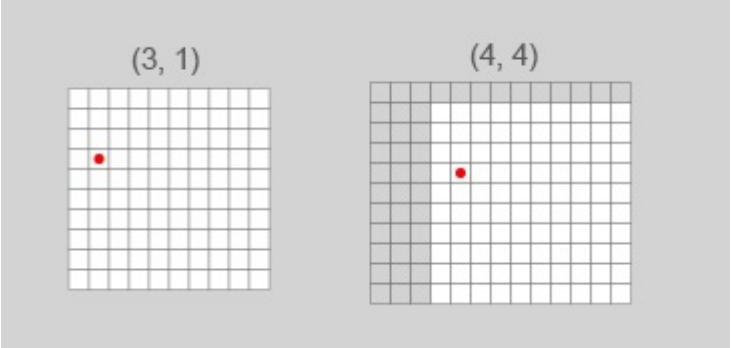
O alcance mínimo é utilizado durante a etapa de desvio de obstáculos. O desvio de obstáculos checa se a leitura do sensor cuja orientação é de 0°, é menor do que o alcance mínimo. Caso o teste seja verdadeiro, o veículo sabe que existe um obstáculo próximo, e escolhe um controle diferente de z_1 (mover para frente), para executar. Foi observado que os sensores ultrassônicos não fornecem boas medidas quando existe um obstáculo muito próximo, cerca de 5cm ou menos. Portanto, estes sensores foram configurados para medir distâncias mínimas de 10cm.

A posição do robô é representada por uma célula do mapa e a orientação em graus. Durante a etapa de localização esta posição é alterada caso o movimento z_1 seja executado.

Caso o movimento z_2 ou z_3 seja executado, a localização atualiza a orientação do robô, mas não sua posição em relação ao mapa. A implementação em Python do algoritmo de localização é apresentada no Apêndice B.

Após atualizar a posição do robô, as dimensões do mapa são redefinidas, de modo a garantir que as leituras não ultrapassem o tamanho atual do mapa. Para implementar este método foi definida a variável de alcance máximo dos sensores, com este alcance e a posição do robô, é possível determinar a célula-alvo para o alcance máximo. Caso a célula-alvo esteja fora dos limites da matriz, são adicionadas novas linhas e colunas, no final da matriz ou no início da matriz, de acordo com a necessidade. Uma correção na posição do veículo é feita caso novas linhas ou novas colunas sejam adicionadas no início da matriz. A correção é feita ao adicionar a quantidade de linhas e colunas novas à posição do robô (Figura 18).

Figura 18 – Correção da posição do robô após ajuste no tamanho do mapa. A correção foi feita após adicionar três colunas e uma linha no mapa.



Fonte: Próprio Autor

4.9 Desenvolvimento do Algoritmo de Mapeamento

O algoritmo de mapeamento desenvolvido consiste em atualizar a matriz de ocupação. O mapeamento recebe os dados processados pelo algoritmo de associação de dados e a posição atual do robô. Para cada célula da matriz presente no conjunto de dados passados é feita a média do valor anterior da célula e do valor lido. O valor lido é **1** se existir um obstáculo nessa célula, ou **0** caso não exista.

Após atualizar a célula-alvo da medição, o algoritmo de mapeamento atualiza todas as céluas entre a célula-alvo e o robô, porém informando que as células estão livres. Assume-se que as células estão livres porque a leitura dos sensores ultrapassaram a posição destas células, se elas estivessem ocupadas a célula-alvo seria mais próxima ao veículo.

A atualização das demais células é feita através da média entre a probabilidade anterior da célula estar ocupada e zero. Para saber qual célula atualizar, o algoritmo

atualiza a linha e a coluna alvo de acordo com as Equações 4.9 e 4.10.

$$column = column + \lfloor \cos(\theta) * -1 \rfloor \tag{4.9}$$

$$row = row + \lfloor \sin(\theta) \rfloor \tag{4.10}$$

O ângulo θ é a orientação da medida em relação ao eixo positivo das abscissas. Como neste projeto foi utilizado apenas um sensor para os ângulos de 0°, 90° e −90°, a orientação da medida é sempre igual à orientação do sensor. Existem outros tipos de sensores infravermelhos que fornecem mais de uma medida, espaçadas por um ângulo fixo umas das outras. Para estes sensores então, o algoritmo de mapeamento deve receber cada medida separada em uma tupla, como se fossem provenientes de diferentes sensores.

A Equação 4.9 irá ignorar medidas cuja inclinação seja 90° ou −90° devido ao cosseno, que é igual a zero para estes ângulos. Portanto, somente irá alterar o valor da coluna quando a inclinação estiver no intervalo [0, 90°), referente a qualquer quadrante do plano ortogonal. Por exemplo, para uma medida cujo valor seja 0° – posicionada na frente do robô, no sentido positivo das abscissas – o algoritmo irá, a cada iteração, diminuir a coluna em uma célula, atualizando cada célula como livre.

Analogamente a Equação 4.10 irá ignorar medidas cuja inclinação seja 0° ou 180°. Como o mapa é uma matriz composta por uma lista, que possui uma lista para cada linha, as células do mapa são atualizadas de acordo com a Equação 4.11, em que *data* pode ser 0 ou 1, de acordo com a nova medição do sensor.

$$map_{ij}^{t+1} = (map_{ij}^{t} + data)/2 \tag{4.11}$$

A implementação em Python do algoritmo de mapeamento é apresentada no Apêndice C.

4.10 Desenvolvimento do Sistema de Navegação Autônoma

A navegação autônoma foi desenvolvida através de um ciclo de desvio de obstáculos, localização e mapeamento. O desvio de obstáculos informa ao robô qual controle executar; a localização atualiza a posição do veículo após a execução do controle; e por fim o mapa é atualizado com a nova posição do veículo e os novos dados obtidos pelos sensores e pelo algoritmo de associação de dados.

O algoritmo de navegação autônoma é executado quando uma das aplicações de controle envia o comando *start_slam* ao servidor, que encaminha o comando para o robô. Durante a navegação, o robô envia novas informações ao servidor, que por sua vez as

encaminham para os aplicativos de visualização. No aplicativo de visualização é executada a etapa de mapeamento novamente, mas em dispositivos diferentes, para atualizar e exibir o mapa corretamente.

O Algoritmo 1 apresenta o processo de navegação autônoma com localização e mapeamento simultâneos. O algoritmo é executado enquanto houver uma possibilidade de movimento para o veículo. Após a execução do controle, o sistema recebe os dados da odometria, como distância percorrida e a nova orientação do veículo. A posição e a orientação do veículo são atualizados de acordo com os dados da odometria, neste processo assumiu-se que a distância medida pela chave óptica é correta, para que se possa construir o mapa em torno da nova posição do veículo.

Algoritmo 1: Algoritmo de Localização e Mapeamento Simultâneos

```
Lê arquivo de configuração e inicializa o veículo;
Determina a posição inicial;
Constrói o mapa inicial;
while Existe um controle que pode ser executado do
    Executa o controle;
    Atualiza a posição do robô;
    Obtêm leituras dos sensores;
    Associa dados;
    Atualiza o estado do mapa;
    Envia dados para o servidor;
    Determina o próximo controle a ser executado;
Anuncia o fim do algoritmo para o servidor;
```

Após a localização, as informações que o robô possui sobre o ambiente não são mais úteis, pois elas se referem a um ponto diferente daquele em que o veículo está. Portanto é feita a etapa de sensoriamento, na qual se obtém as leituras de todos os sensores do robô. O sistema foi desenvolvido para que a quantidade de sensores seja variável, possibilitando ao veículo agregar mais do que três sensores de distância, como utilizado neste projeto.

O sensoriamento é um laço de repetição que armazena a informação de cada sensor em uma lista. As informações contêm a distância entre o sensor e o obstáculo, a distância mínima que o sensor é capaz de medir, a presença de um obstáculo, e o ângulo de medição. A lista de leituras é passada ao algoritmo de associação de dados para preparar as leituras para a etapa de mapeamento.

Durante a associação de dados a distância de cada leitura (em centímetros) é convertida em posições na matriz. A conversão é feita utilizando a resolução da aplicação e o ângulo da medida. Para este projeto a resolução escolhida foi de 10cm, o que significa que cada célula no mapa possui largura e comprimento de 10cm. Assim, a distância entre o veículo e a medição é a razão entre a distância medida e a resolução do mapa. Como a

matriz é acessada apenas com números inteiros, a razão é arredondada para um número inteiro utilizando-se a função teto.

O mapa pode ser atualizado após a etapa de associação de dados. Receber os dados transformados em células da matriz, ao invés das medidas, possibilita ao algoritmo de mapeamento processar apenas o novo estado do mapa. Também é possível fazer a associação de dados junto ao mapeamento, para que não seja preciso iterar sobre todas as leituras mais de uma vez, porém a quantidade de leituras não é suficientemente grande para impactar o desempenho do sistema. Separar a associação de dados do mapeamento é importante caso se queira extrair características do ambiente que não sejam apenas a existência ou não de obstáculos, como é comum em outros projetos SLAM (Seção 2.3).

O mapeamento determina todas as células da matriz que foram afetadas pelas medidas dos sensores e atualiza o estado de cada uma. Para esta estrutura, considerou-se que a probabilidade de uma célula estar ocupada é independente da probabilidade de ocupação das células vizinhas. Para o mapeamento também foi considerado que o mundo é estático, o ambiente não altera durante o processo de navegação.

Uma iteração do processo de localização e mapeamento simultâneos termina quando o robô envia os dados para o servidor e decide qual o próximo controle será executado. O envio dos dados para o servidor serve para exibir o mapa para quaisquer aplicações de visualização que estejam conectadas a ele por *WebSocket*. As aplicações de visualização executam o processo de adicionar novas linhas e colunas à matriz, e ao processo de mapeamento novamente, garantindo que os dados recebidos sejam exibidos do mesmo modo em que o robô os interpreta.

4.11 Simulação do Planejamento de Rotas com Busca A*

No sistema, foram desenvolvidos algoritmos de desvio de obstáculo, localização, mapeamento e comunicação com um servidor. Para complementar as funcionalidades do veículo e auxiliar durante o processo de navegação, foi também necessário desenvolver um método para que o veículo pudesse planejar as rotas entre a sua posição atual e uma posição destino qualquer.

Para o planejamento de rotas, foram desenvolvidos dois algoritmos, um que sempre expande o estado com menor custo, e outro que expande o estado em que o custo mais a distância entre o estado e o objetivo for menor. O segundo é a Busca A*, que utiliza como heurística uma matriz cujos valores é a distância entre a célula e o objetivo.

Existem quatro movimentos possíveis, mover para cima, para baixo, para a esquerda ou para a direita, estes movimentos são armazenados em uma lista e são consultados pela busca para determinar as células vizinhas. As células vizinhas são testadas para garantir

que a área é navegável – probabilidade no intervalo de [0, 0.5) – e que a célula está contida nas dimensões da matriz.

Durante a expansão das células, é marcado em cada posição qual movimento foi realizado. O movimento é armazenado na matriz *ação* e é utilizado para definir a rota resultante. Partindo-se da posição final, o trajeto é calculado somando-se a posição atual ao inverso do movimento realizado. Por exemplo, se foi somado um ao número de colunas e zero ao número de linhas, a célula anterior é igual à atual menos uma coluna.

A função de desenho da aplicação de atualização, foi alterada para exibir tanto o caminho resultante do planejamento de rotas quanto os nós expandidos, possibilitando visualizar a quantidade de nós expandidos pelo algoritmo A* e pela busca não informada.

4.12 Sumário do Capítulo

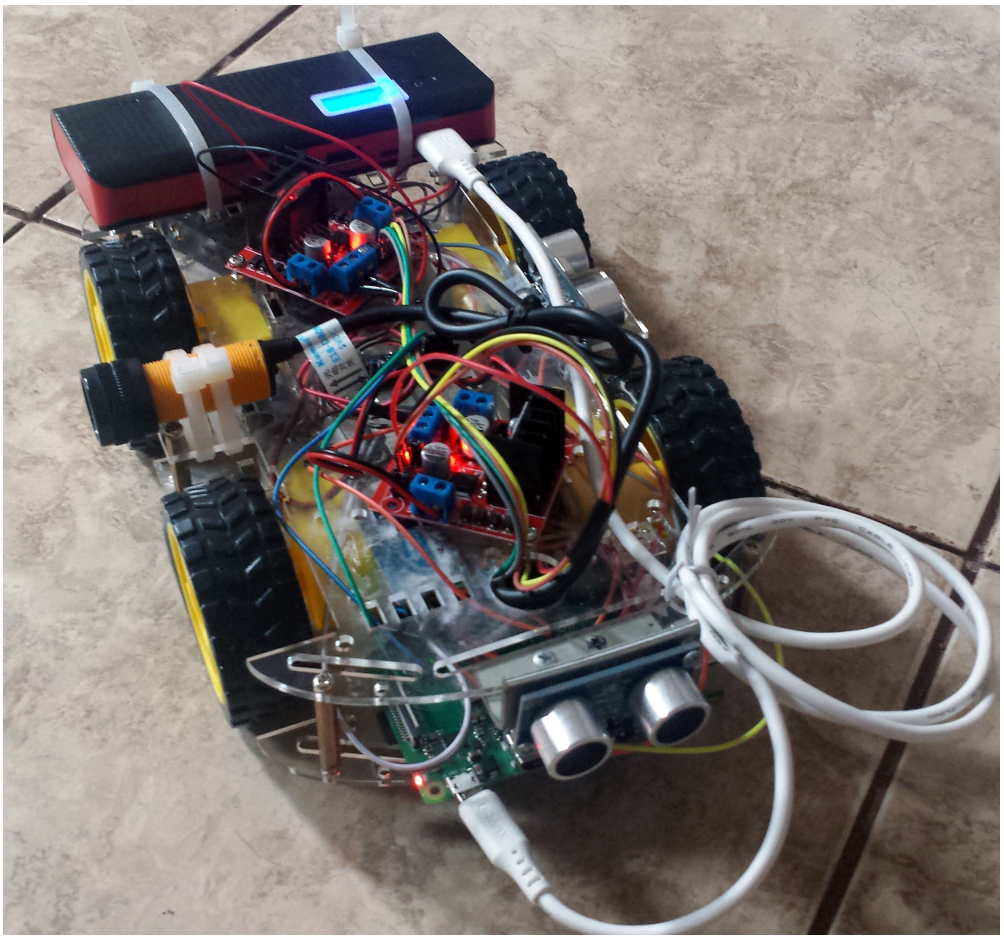
Neste capítulo foram apresentados as etapas necessárias para desenvolver o sistema. Partindo-se da integração dos sensores ao Raspberry Pi à simulação do planejamento de rotas. Durante o desenvolvimento do projeto, foi preciso realizar abstrações e simplificar o escopo do projeto para que o sistema pudesse funcionar corretamente com os componentes disponíveis. Foram também apresentas decisões de projeto que impactaram positivamente no desenpenho do sistema, como o formato da troca de mensagens. No próximo capítulo serão discutidos os resultados obtidos pelo projeto desenvolvido, analisando o desempenho dos sensores, do sistema de navegação, da aplicação de controle e do planejamento de rotas.

5 RESULTADOS E ANÁLISE

Neste capítulo são apresentados os resultados do projeto desenvolvido. Na Seção 5.1 são discutidas as informações sobre a odometria do veículo, do seu movimento e suas limitações quanto à navegação. Na Seção 5.2 são apresentados os resultados do software de controle com ilustrações das telas da aplicação e de sua performance. Na Seção 5.3 são discutidos os resultados do algoritmo de planejamento de rotas. Por fim, discute-se o algoritmo de navegação autônoma, analisando mapas desenvolvidos pela aplicação e a performance do algoritmo.

O prótotipo do veículo desenvolvido é apresentado na Figura 19. Na qual se pode observar o posicionamento dos sensores, da fonte de alimentação, do microcontrolador e das rodas do veículo. As Pontes-H foram dispostas na parte superior do veículo; os circuitos e o microcontrolador foram colocados entre a parte inferior e a parte superior do veículo para proteger os componentes.

Figura 19 – Protótipo do veículo autônomo.



Fonte: Próprio Autor

5.1 Odometria

Foram realizados testes de integração para validar os controles do veículo, mover uma célula para a frente, girar para a esquerda e girar para a direita. A odometria adotada depende da quantidade de pulsos por centímetro, medida experimentalmente, e das leituras do sensor MPU-9250 para estimar a nova postura do veículo, que é dada pelas coordenadas do veículo na grade de ocupação e a sua orientação em graus.

Observou-se que a odometria do veículo é imprecisa. Com uma resolução de 10cm por célula, cada movimento do veículo faz com que o odômetro calcule uma distância de ± 0.25cm. Erro substancial, uma vez que com 40 movimentos para frente, o veículo terá um erro de aproximadamente uma célula. Em aplicações SLAM, existem técnicas para tratamento de erros, como filtros de Kalman e filtros de partículas. Como estes métodos não foram implementados, e foi assumido que o movimento do veículo é preciso, é esperado que exista erros tanto no movimento do veículo, quanto no sensoriamento.

Como a odometria é baseada na quantidade de voltas de uma roda, variações na velocidade não impactam na navegação. Foi observado variações na velocidade porque a tensão de alimentação das baterias diminui à medida em que as baterias são utilizadas. Com tensões menores, o módulo L298N não é capaz de suprir os motores com alimentação suficiente para locomover o veículo.

Notou-se também que os movimentos de girar necessitam de mais esforço do que ir para frente. Portanto, com baterias desgastadas o robô se torna incapaz de realizar os movimentos z_2 e z_3. Estes movimentos foram inicialmente implementados com o deslocamento, em graus, calculado pelo MPU-9250. Entretanto, esta unidade de medição inercial não é capaz de fornecer ângulos de *yaw* confiáveis em tempo real.

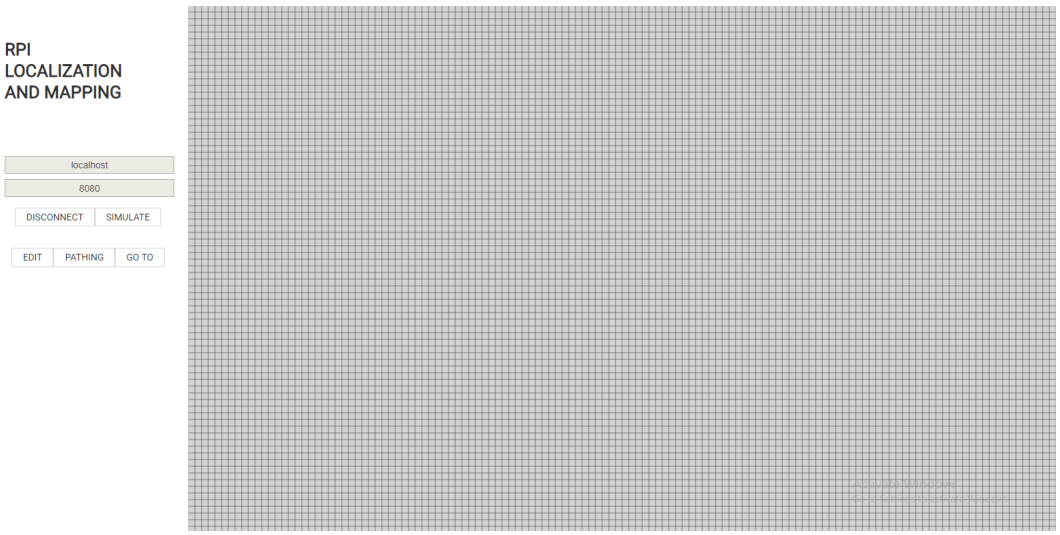
Para garantir que os movimentos de rotação pudessem ser executados, foi medido o tempo necessário para que o veículo movesse 90°, tanto para a esquerda quanto para a direita. As medidas do IMU foram dispensadas e o movimento do robô foi baseado apenas na odometria da chave óptica e do tempo de atuação dos motores.

5.2 Software de Controle Remoto

O *software* de controle remoto foi desenvolvido tanto para exibir o mapa gerado pelo veículo quanto para simular a navegação com algoritmos de busca. A Figura 20 mostra a interface do aplicativo ao ser conectada com o servidor. No navegador à esquerda existem opções para desconectar, editar o mapa, criar um caminho entre duas células do mapa, e fazer com que o robô vá até uma célula específica.

Em cinza é exibida uma matriz que preenche toda a tela do navegador. Quando o veículo inicia o mapeamento, todas as aplicações que estiverem conectadas ao servidor

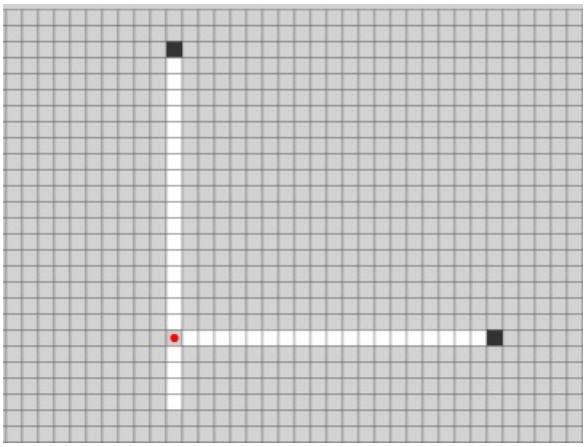
Figura 20 – Interface da aplicação de controle.



Fonte: Próprio Autor

recebem a posição inicial do veículo e as três primeiras leituras. As leituras e a posição são recebidos em células da matriz. Parte do algoritmo de mapeamento é repetido no navegador de cada cliente, atualizando o mapa sem que seja necessário que o robô envie todo o mapa para o servidor, o qual repassa todas as informações para cada cliente conectado.

Figura 21 – Posição inicial do veículo com três leituras.

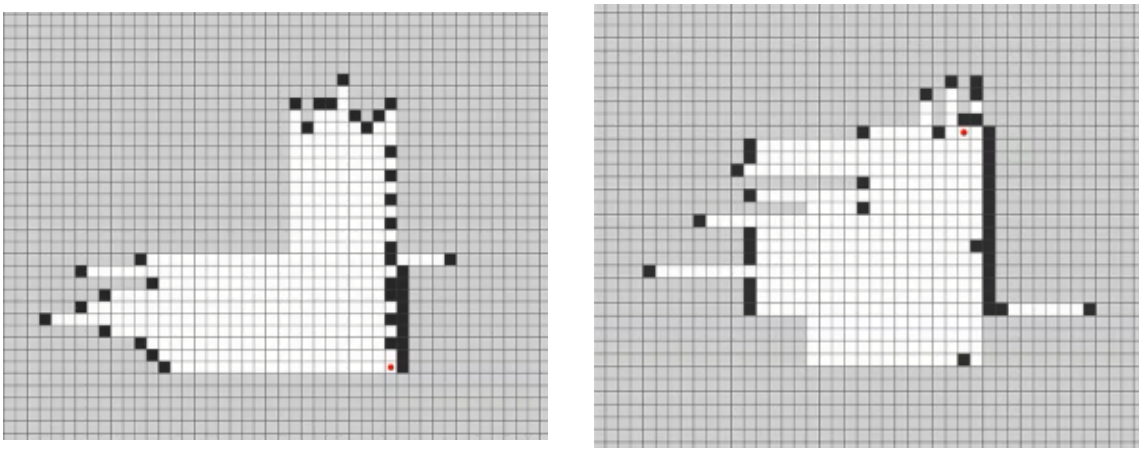


Fonte: Próprio Autor

A Figura 21 mostra o estado do mapa após receber os dados inicais do robô. O ponto vermelho indica a posição atual do robô, as células em preto indicam os obstáculos encontrados pelos sensores de distância, e as células brancas representam área navegável. Neste estado, a orientação do veículo é sempre considerada 0°, portanto o obstáculo à direita do ponto vermelho é a leitura do sensor da frente do veículo, o obstáculo acima do ponto vermelho é a leitura da esquerda, e abaixo a leitura do sensor infravermelho. Como o sensor infravermelho tem alcance fixo, ele detectará um obstáculo imediatamente abaixo do veículo, ou uma área livre.

A Figura 22 exibe o resultado do mapa após a navegação. Durante a navegação o veículo apresentou dificuldade em realizar os movimentos de girar para a esquerda e para a direita, controles u_2 e u_3. Assim, foram introduzidos erros de medição devido ao veículo julgar que sua orientação é 90° a mais do que a orientação real. Portanto a navegação foi interrompiada antes que o algoritmo de SLAM terminasse.

Figura 22 – Resultado dos testes de navegação.



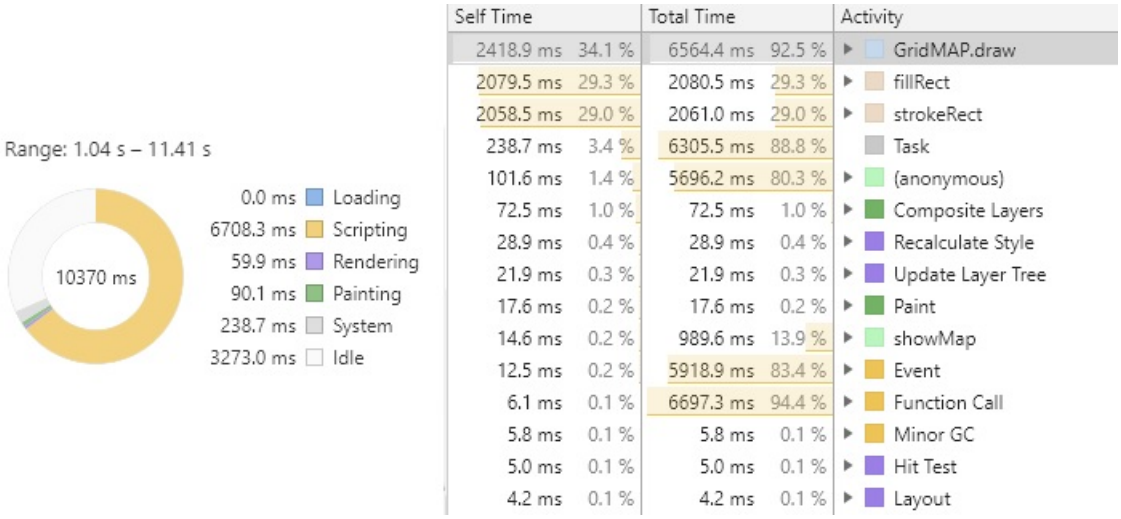
Fonte: Próprio Autor

A troca de mensagens entre o servidor e os sockets conectados a ele é consistente, as mensagens são sempre entregues em ordem e com toda a informação enviada. Isto ocorre porque o protocolo *WebSockets* é implementado sobre o protocolo *TCP*, o que garante vantagens como entrega ordenada de mensagens e confiabilidade de que as mensagens serão sempre entregues. Entretanto, como esta aplicação é executada em tempo real, as aplicações de visualização podem ficar desatualizadas caso o veículo esteja conectado a uma rede congestionada. O algoritmo SLAM é independente da troca de mensagens, portanto continuará a ser executado mesmo quando houver vários dados esperando ser entregues ao servidor.

A performance da aplicação de visualização é limitada pela renderização do mapa na tela. Inicialmente o mapa era renderizado sempre que uma nova mensagem era recebida ou sempre que o mouse movesse sobre o *canvas*. Com a ferramenta de performance disponível no *Google Chrome*, mediu-se o tempo gasto em cada função durante a edição do mapa (Figura 23). O sistema gasta aproximadamente 57% do tempo de execução renderizando o mapa, e deste tempo 34.1% calculando a posição de cada célula a ser desenhada pelo método *GridMAP.draw*.

Durante o processo de navegação, a performance do aplicativo de visualização é melhor do que durante o processo de edição. Isto ocorre porque o mapa é atualizado e renderizado apenas uma vez. A Figura 24 mostra o tempo gasto durante a atualização *GridMAP.update* e renderização *GridMAP.draw*. Pode-se observar também que o método *GridMAP.initialize* é relativamente custoso, porque ele cria a matriz inicial de acordo

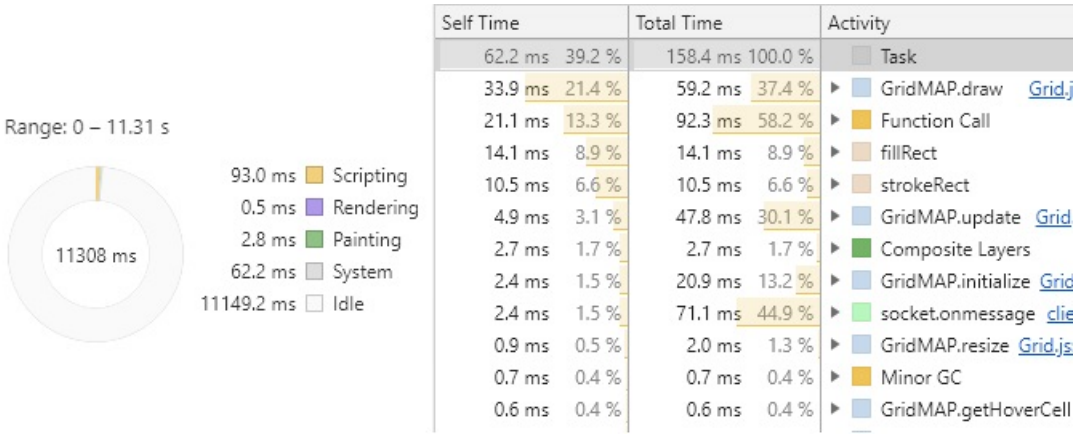
Figura 23 – Performance do algoritmo de renderização do mapa.



Fonte: Próprio Autor

com o alcance máximo dos sensores, assim como o robô faz durante a inicialização do SLAM. Comparando-se com a performance em modo de edição, o aplicativo de visualização durante a navegação é consideravelmente melhor, pois o mapa é desenhado apenas uma vez por cada iteração do algoritmo SLAM do veículo.

Figura 24 – Performance do algoritmo de atualização do mapa.

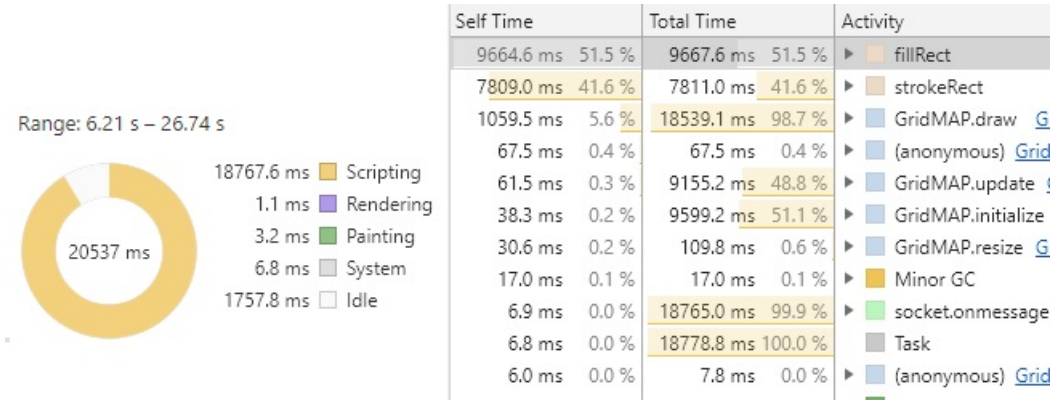


Fonte: Próprio Autor

Os dados das Figuras 23 e 24 são referentes a um mapa com resolução de 10cm por célula. Para o mesmo alcance de sensores, com uma resolução de 1cm por célula, a performance de apenas uma atualização é visivelmente pior (Figura 25). Embora a resolução tenha aumentado em 10 vezes, o tamanho da matriz é aumentado em 100 vezes. Este custo é decorrente da escolha da estrutura de dados que representa o mapa.

Dada a posição do veículo, a matriz cresce de acordo com o alcance do sensor e da resolução do mapa. Como resultado obtém-se uma matriz quadrada. Quando o veículo se move, novas linhas e colunas são adicionadas de acordo com o alcance máximo, tornando a matriz não quadrada. Para renderizar a matriz, o algoritmo passa por cada célula e decide

Figura 25 – Performance do algoritmo de atualização do mapa com resolução de 10cm/célula.



Fonte: Próprio Autor

qual cor preencher. Portanto pode-se aferir que o custo depende da quantidade de linhas e de colunas da matriz, que pode ser representado pela notação big-O por $\mathcal{O}(mn)$, em que m é a quantidade de linhas e n a quantidade de colunas.

5.3 Planejamento de Rotas

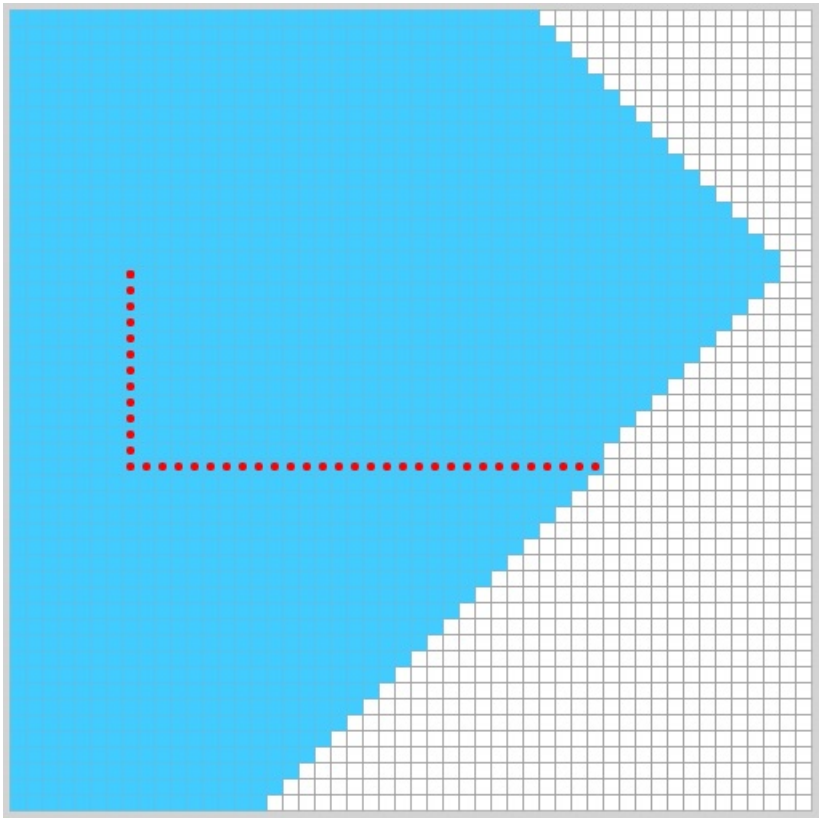
Os algoritmos de busca puderam ser implementados com sucesso como planejamento de rotas. Foram desenvolvidos dois algoritmos, um com informações adicionais à respeito do mapa (Algoritmo A*) e outro sem informações adicionais, que expande sempre a célula com o menor número de expansões. O custo adotado para movimentar entre duas células quaisquer é sempre unitário, e as informações adicionais passadas para o algoritmo A* são dispostas em uma matriz que informa a distância entre todas as células e o objetivo.

A Figura 26 mostra um caminho feito sobre um mapa completamente livre. A área azul representa todas as células que foram visitadas e expandidas pelo algoritmo. Os pontos vermelhos representam a trajetória encontrada. A Figura 27 mostra o mesmo mapa e a rota resultante do algoritmo de busca A*.

Pode-se observar que a quantidade de células expandidas é consideravelmente menor no algoritmo A*. Isto ocorre porque o algoritmo prioritiza células cujo custo mais a distância entre a célula e o alvo são menores. Portanto, quanto mais distante a célula estiver do alvo, menor a chance da célula ser expandida, permitindo ao algoritmo checar células mais próximas ao alvo.

Para mapas pequenos ou com poucas células livres entre obstáculos, pode ser que o veículo não seja capaz de executar a rota. Pois a largura do veículo é maior que o tamanho de uma célula. Por exemplo, a rota simulada para um mapa de tamanho 5x6 células (Figura 28) pode não ser uma rota válida dependendo das dimensões do veículo e da resolução do mapa. Uma possível solução é diminuir a resolução do mapa para ser

Figura 26 – Nós expandidos pelo algoritmo de busca por largura.



Fonte: Próprio Autor

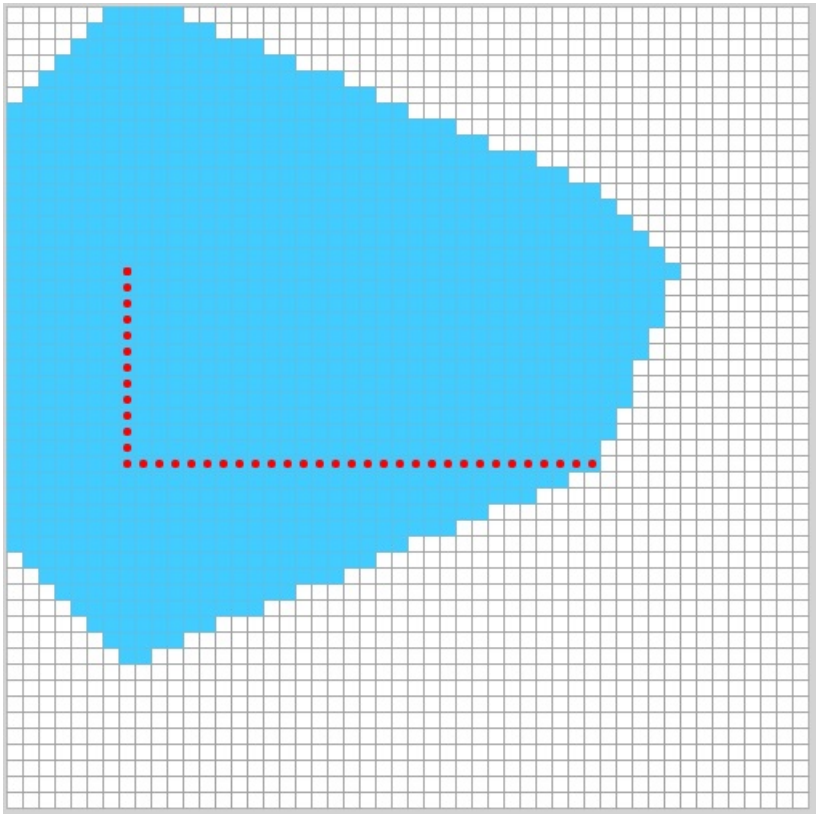
igual ao tamanho do veículo.

Devido à inconsistência dos movimentos do veículo (Seção 5.2), o planejamento de rotas foi implementado apenas nas aplicações de visualização como uma ferramenta de simulação para planejar rotas reais em um mapa representado por grades de ocupação. Este planejamento pode ser enviado para o veículo para que a rota não precise ser calculada novamente, economizando recursos de processamento do *software* embarcado. Como os microcontroladores possuem geralmente menor capacidade de processamento e de memória disponíveis, é interessante que o planejamento de rotas seja feito em um computador remoto.

5.4 Análise do Sistema de Navegação

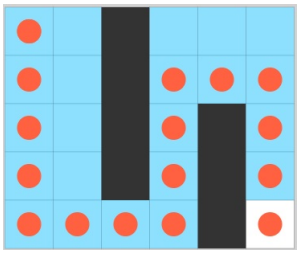
O algoritmo de navegação implementado é capaz de fazer o veículo mover pelo ambiente sem colidir com obstáculos e representar o ambiente com grades de ocupação. Entretanto não é capaz de determinar a posição do veículo e o estado do mapa com exatidão. Pela definição formal de SLAM (Seção 2.4), sabe-se que o problema de mapeamento e localização simultâneos é representado através de probabilidades. Portanto, um sistema que não inclui incerteza de movimento e de representação do mapa, como o desenvolvido neste trabalho, não é capaz de garantir resultados precisos. O código do *software* embarcado, do

Figura 27 – Nós expandidos pelo algoritmo de busca A*.



Fonte: Próprio Autor

Figura 28 – Rota resultante para uma grade de ocupação de tamanho 5x6.



Fonte: Próprio Autor

servidor e da aplicação de controle estão disponíveis no *GitHub*[1].

A etapa de mapeamento é capaz de comportar sensores dipostos nas orientações de 0°, 90°, 180° e 270°. Caso seja adicionado algum sensor em uma orientação diferente, o algoritmo de mapeamento não será capaz traçar a área livre entre o veículo e a célula-alvo. Isto ocorre porque o processo de mapeamento atualiza a posição das células entre o veículo e medição através das Equações 4.9 e 4.10. Nestas equações são utilizados os valores de seno e coseno para atualizar a linha e a coluna alvo. Estes valores são arredondados utilizando a função piso. Como seno e cosseno assumem valores reais no intervalo [0, 1], o valor da célula-alvo nunca irá ser alterado para ângulos não ortogonais.

[1] Código-Fonte: <https://github.com/pedrohov/rpi-localization-mapping>. Acessado em Junho/2019.

5.5 Sumário do Capítulo

Com as discussões realizadas neste capítulo pode-se perceber que é possível desenvolver um sistema de navegação e mapeamento simultâneos utilizando sensores e estruturas de baixo custo. Entretanto, o sistema também se limita de acordo com a capacidade dos sensores de fornecer dados precisos em tempo real, e com a quantidade de informação que o sistema recebe. Pode-se também observar o impacto que a estrutura de dados e que a resolução tem sobre o desempenho do sistema. No próximo capítulo são apresentados os trabalhos que visam aprimorar este sistema e contornar as principais limitações do algoritmo de navegação.

6 TRABALHOS FUTUROS

Espera-se aprimorar o sistema desenvolvido neste trabalho com trabalhos futuros, de modo a diminuir as limitações do veículo e desenvolver um sistema de navegação autônoma mais preciso quanto ao posicionamento do veículo e à criação de mapas.

Neste trabalho o desvio de obstáculos foi inicialmente desenvolvido com redes neurais e algoritmo genético. Entretanto não se obteve resultados positivos durante a etapa de treinamento da rede artificial. Para que o veículo não utilize de controles determinísticos, como a rotina de desvio de obstáculos implementada, pode-se criar um controlador utilizando lógica *Fuzzy* para decidir qual o melhor controle a ser executado, dadas as leituras dos sensores.

Outro trabalho que pode ser desenvolvido é a implementação do algoritmo Linha de Bresenham (DAO, 1996) para calcular a área livre entre o obstáculo e o veículo. O uso deste algoritmo permitirá ao sistema acomodar diversos sensores em orientações diferentes de 0°, 90°, 180° e 270°, o que auxiliaria não só no mapeamento, mas também na navegação, possibilitando extrair marcos que fornecem mais informação do que apenas a ocupação ou não de uma célula em uma grade de ocupação.

Por fim, para tratar das incertezas inerentes às medições dos sensores e do mapeamento, é importante que o sistema possua um modelo para tentar prever erros futuros ou modelar a esperança de ocorrência destes erros para que eles possam ser diminuídos durante a etapa de mapeamento e navegação simultâneos. Uma destas técnicas é o uso do filtro extendido de Kalman (EKF) (HUANG; DISSANAYAKE, 2007), que é capaz de estimar o valor real de uma medição, dada a incerteza e a medida.

7 CONSIDERAÇÕES FINAIS

Com o desenvolvimento do protótipo e a realização dos testes funcionais, pode-se afirmar que é possível desenvolver um sistema de navegação autônoma com localização e mapeamento simultâneos utilizando sensores de baixo custo.

O poder de processamento de plataformas de prototipagem como o Raspberry Pi reduz as lmitações para a representação do mapa, permitindo o uso de estruturas de custo relativamente alto, como matrizes, para representar o ambiente. Mesmo para altas resoluções, o microcomputador é capaz de criar grades de ocupação sem ultrapassar os limites de memória.

O custo de altas resoluções é refletido na incerteza do movimento do veículo. Quanto menor o tamanho de uma célula do mapa, em centímetros, mais precisos precisam ser a odometria e os motores do veículo. Como os motores utilizados foram motores de corrente contínua, o movimento do veículo não pode ser controlado com exatidão, dependendo completamente nas informações do odômetro e de temporizadores.

Após o desenvolvimento ficou aparente a necessidade de um modelo da incerteza durante o sensoriamento e a movimentação do veículo. A movimentação do veículo acumulou erro considerável durante a etapa de localização. Este erro é repassado à etapa de mapeamento, que utiliza a posição do veículo para calcular a posição de cada obstáculo no mapa. Com a posição errada, o mapa é atualizado incorretamente, e não apenas insere medidas erradas, mas também acrescenta erro às medidas anteriores.

O protocolo *WebSockets* facilitou o processo de desenvolvimento, e permitiu a transmissão de diversas estruturas, como listas, dicionários, números e *strings*, sem se preocupar com a conversão destes tipos em um formato único para transmissão. Este protocolo pode ser utilizado em aplicações de tempo real para troca de informações por Wi-Fi, entretanto, para redes congestionadas o tempo de resposta pode demorar. Como o *WebSockets* é implementado sobre o TCP, o protocolo controla o congestionamento, diminuindo a taxa de transmissão para redes muito congestionadas. Este controle de congestionamento pode impactar negativamente em aplicações de tempo real. Uma alternativa ao uso de *WebSockets* seria a comunicação sem fio via rádio, ou a criação de uma rede Wi-Fi privada no Raspberry Pi.

Pode-se verificar a influência que um sistema de controle remoto têm sobre o *software* embarcado. Algoritmos que precisam de maior poder de processamento, como o planejamento de rotas e a renderização do mapa, podem ser delegados a outros computadores conectados ao *software* embarcado ou a um servidor. Os computadores remotos processam as informações necessárias e devolvem resultado, possibilitando que o robô

compute menos tarefas simultâneamente.

O sistema de localização e de mapeamento desenvolvidos utilizam quaisquer leituras que estejam no formato estabelecido durante o projeto do sistema, o que possibilita a trabalhos futuros utilizarem o sistema com quaisquer sensores de distância. Espera-se que este algoritmo possibilite uma representação de fácil compreensão do mapa, que possa ser reutilizada para projetos com necessitem de mapeamento, como robôs domésticos e robôs para competições de resolver labirintos. Espera-se também que a teoria e as decisões de projeto auxiliem no desenvolvimento de novos sistemas de navegação em duas dimensões e estimulem a pesquisa na área de robótica móvel.

Referências

AEROSPACE; concepts, quantities and symbols for flight dynamics; aircraft geometry. [S.l.]: Deutsches Institut fur Normung, 1990. Citado na página 33.

AULINAS, J. et al. The slam problem: a survey. *CCIA*, Citeseer, v. 184, n. 1, p. 363–371, 2008. Citado na página 16.

BAILEY, T.; DURRANT-WHYTE, H. Simultaneous localization and mapping (slam): Part ii. *IEEE Robotics & Automation Magazine*, IEEE, v. 13, n. 3, p. 108–117, 2006. Citado na página 16.

BUONOCORE, L.; NETO, A. D. A.; JÚNIOR, C. L. N. A sensor data fusion algorithm for indoor slam using a low-cost mobile robot. In: *Adaptive Mobile Robotics*. [S.l.]: World Scientific, 2012. p. 762–769. Citado na página 30.

CADENA, C. et al. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics*, IEEE, v. 32, n. 6, p. 1309–1332, 2016. Citado 2 vezes nas páginas 16 e 17.

CHATILA, R.; LAUMOND, J.-P. Position referencing and consistent world modeling for mobile robots. In: IEEE. *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*. [S.l.], 1985. v. 2, p. 138–145. Citado na página 19.

DANNO, K. et al. Near-infrared irradiation stimulates cutaneous wound repair: laboratory experiments on possible mechanisms. *Photodermatology, photoimmunology & photomedicine*, Wiley Online Library, v. 17, n. 6, p. 261–265, 2001. Citado na página 30.

DAO, G. H. *Bresenham/DDA line draw circuitry*. [S.l.]: Google Patents, 1996. US Patent 5,570,463. Citado na página 76.

DUDEK, G. et al. Robotic exploration as graph construction. *IEEE transactions on robotics and automation*, IEEE, v. 7, n. 6, p. 859–865, 1991. Citado na página 20.

EDMONDS, A. R. *Angular momentum in quantum mechanics*. [S.l.]: Princeton university press, 1996. v. 4. Citado na página 32.

EIBEN, A. E.; SMITH, J. E. et al. *Introduction to evolutionary computing*. [S.l.]: Springer, 2003. v. 53. Citado na página 26.

ELFES, A. Occupancy grids: A stochastic spatial representation for active robot perception. In: *Proceedings of the Sixth Conference on Uncertainty in AI*. [S.l.: s.n.], 1990. v. 2929, p. 6. Citado na página 20.

ELIZABETH, W.; CHARLES, S.; ROBIN, G. d. Q. A submarine sonar study of arctic pack ice. *Journal of Glaciology*, Cambridge University Press, v. 15, n. 73, p. 349–362, 1975. Citado na página 29.

ENCODER and Arduino. Tutorial about the IR speed sensor module with the comparator LM393. 2017. <http://androminarobot-english.blogspot.com/2017/03/encoder-and-arduinotutorial-about-ir.html>. Acesso em: 26 de Maio, 2019. Citado na página 30.

FISCHLER, M. A.; BOLLES, R. C. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, ACM, v. 24, n. 6, p. 381–395, 1981. Citado na página 22.

FOGEL, D. B. An introduction to simulated evolutionary optimization. *IEEE transactions on neural networks*, v. 5, n. 1, p. 3–14, 1994. Citado na página 27.

FRAUNDORFER, F.; SCARAMUZZA, D. Visual odometry: Part ii: Matching, robustness, optimization, and applications. *IEEE Robotics & Automation Magazine*, IEEE, v. 19, n. 2, p. 78–90, 2012. Citado na página 16.

GABRIEL, P. H. R.; DELBEM, A. C. B. *Fundamentos de algoritmos evolutivos*. [S.l.]: ICMC-USP, 2008. Citado 2 vezes nas páginas 26 e 27.

GAGNIUC, P. A. *Markov Chains: From Theory to Implementation and Experimentation*. [S.l.]: John Wiley & Sons, 2017. Citado na página 24.

GOLDBERG, D. E.; HOLLAND, J. H. Genetic algorithms and machine learning. *Machine learning*, Springer, v. 3, n. 2, p. 95–99, 1988. Citado na página 26.

HART, P. E.; NILSSON, N. J.; RAPHAEL, B. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, IEEE, v. 4, n. 2, p. 100–107, 1968. Citado na página 27.

HUANG, S.; DISSANAYAKE, G. Convergence and consistency analysis for extended kalman filter based slam. *IEEE Transactions on robotics*, IEEE, v. 23, n. 5, p. 1036–1049, 2007. Citado na página 76.

IETF. *The WebSocket Protocol*. 2011. <https://tools.ietf.org/html/rfc6455>. Acesso em: 26 de Maio, 2019. Citado na página 28.

KÜMMERLE, R. et al. On measuring the accuracy of slam algorithms. *Autonomous Robots*, Springer, v. 27, n. 4, p. 387, 2009. Citado na página 16.

LOWRY, S. et al. Visual place recognition: A survey. *IEEE Transactions on Robotics*, IEEE, v. 32, n. 1, p. 1–19, 2015. Citado na página 16.

MAHULIKAR, S. P.; SONAWANE, H. R.; RAO, G. A. Infrared signature studies of aerospace vehicles. *Progress in aerospace sciences*, Elsevier, v. 43, n. 7-8, p. 218–245, 2007. Citado na página 29.

MICHALEWICZ, Z. *Genetic algorithms+ data structures= evolution programs*. [S.l.]: Springer Science & Business Media, 2013. Citado na página 26.

MICHALEWICZ, Z.; JANIKOW, C. Z. Handling constraints in genetic algorithms. In: *ICGA*. [S.l.: s.n.], 1991. p. 151–157. Citado na página 27.

MONTEMERLO, M.; THRUN, S. *FastSLAM: A scalable method for the simultaneous localization and mapping problem in robotics*. [S.l.]: Springer, 2007. v. 27. Citado 2 vezes nas páginas 23 e 25.

OLSON, C. F. et al. Rover navigation using stereo ego-motion. *Robotics and Autonomous Systems*, Elsevier, v. 43, n. 4, p. 215–229, 2003. Citado na página 17.

PARKER, L. E. Current state of the art in distributed autonomous mobile robotics. In: *Distributed Autonomous Robotic Systems 4*. [S.l.]: Springer, 2000. p. 3–12. Citado na página 17.

POSSANI, D. et al. Ondas ultrassônicas: teoria e aplicações industriais em ensaios não-destrutivos. *Revista Brasileira de Física Tecnológica Aplicada*, v. 4, n. 1, 2017. Citado na página 29.

RIISGAARD, S.; BLAS, M. R. Slam for dummies. *A Tutorial Approach to Simultaneous Localization and Mapping*, v. 22, n. 1-127, p. 126, 2003. Citado 3 vezes nas páginas 8, 21 e 22.

RUSSEL, S.; NORVIG, P. Artificial intelligence: A modern approach. *EUA: Prentice Hall*, v. 178, 2003. Citado 2 vezes nas páginas 25 e 44.

RUSSELL, S. J.; NORVIG, P. *Artificial intelligence: a modern approach*. [S.l.]: Malaysia; Pearson Education Limited,, 2016. Citado na página 28.

SCARAMUZZA, D.; FRAUNDORFER, F. Visual odometry [tutorial]. *IEEE robotics & automation magazine*, IEEE, v. 18, n. 4, p. 80–92, 2011. Citado na página 16.

THRUN, S.; BURGARD, W.; FOX, D. *Probabilistic robotics*. [S.l.]: MIT press, 2005. Citado na página 16.

THRUN, S.; MONTEMERLO, M. The graph slam algorithm with applications to large-scale mapping of urban structures. *The International Journal of Robotics Research*, SAGE Publications, v. 25, n. 5-6, p. 403–429, 2006. Citado na página 17.

THRUN, S. et al. Robotic mapping: A survey. *Exploring artificial intelligence in the new millennium*, v. 1, n. 1-35, p. 1, 2002. Citado na página 17.

TIPLER, P. A.; MOSCA, G. *Física para cientistas e engenheiros. Vol. 2: eletricidade e magnetismo, óptica* . [S.l.]: Grupo Gen-LTC, 2000. Citado na página 38.

YOUSIF, K.; BAB-HADIASHAR, A.; HOSEINNEZHAD, R. An overview to visual odometry and visual slam: Applications to mobile robotics. *Intelligent Industrial Systems*, Springer, v. 1, n. 4, p. 289–311, 2015. Citado 3 vezes nas páginas 16, 19 e 20.

Apêndices

APÊNDICE A – Algoritmo de Associação de Dados

```
import math;

def dataAssociation(map, sensor_data, robot_pose):
    # Transforma as leituras dos sensores em celulas do mapa.
    # A orientacao do robo e do mapa sao feitas no sentido anti-horario.
    # 0 graus  = sentido positivo de 'x'.
    # 90 graus = sentido negativo de 'y'.
    # Retorna uma lista de tuplas com:
    # numero da linha e da coluna alvo,
    # se existe um obstaculo, e orientacao da leitura.

    data = [];
    for key, info in sensor_data.items():
        # Distancia total e igual a medida mais a
        # distancia entre o sensor e o centro do veiculo:
        distance = info['data'] + info['offset'];

        # Transforma a distancia em numero de celulas do mapa:
        offset = math.ceil(distance / map.resolution);

        # Encontra a celula alvo do mapa:
        x, y = robot_pose['x'], robot_pose['y'];
        orientation = info['orientation'];
        x += int(offset * math.cos(math.radians(orientation)));
        y += int(offset * math.sin(math.radians(orientation)) * -1);

        # Cria tupla com os dados necessarios para mapeamento:
        # Posicao da matriz, se existe um obstaculo, direcao do marco.
        data.append((y, x, info['obstacle_found'], orientation));

    return data;
```

APÊNDICE B – Algoritmo de Localização

```
import math;

def localization(robot, map, odometry_data):
    # Atualiza a posicao atual do robo.
    # Retorna a posicao atual e a quantidade de linhas
    # e de colunas adicionadas ao mapa.

    robot_pose = robot.update(odometry_data);
    correct_pose = self.map.resize(robot_pose);
    return (self.robot.correctPose(correct_pose), correct_pose);

# Classe Robot:
def update(self, odometry):
    # Atualiza a posicao do robo de acordo com os dados da odometria.
    # Retorna a nova posicao do robo.

    distance = odometry['distance'];
    orientation = math.radians(self.pose['orientation']);

    # Calcula a nova posicao do robo na matriz:
    if(distance >= 0):
        self.pose['x'] += int(distance * math.cos(orientation) / self.↘
            resolution);
        self.pose['y'] += int(distance * math.sin(orientation) * -1 / self.↘
            resolution);

    # Atualiza a orientacao do robo:
    self.pose['orientation'] = odometry['orientation'];
    for key, sensor in self.sensors.items():
        sensor.updateOrientation(odometry['orientation']);
        print(key, sensor.orientation);

    return self.pose;

# Classe Robot:
def correctPose(self, correction):
    # Ajusta a posicao do robo em relacao ao mapa atualizado.
    if(correction[0] > 0):
        self.pose['x'] += correction[0];
    if(correction[1] > 0):
        self.pose['y'] += correction[1];
    return self.pose;

# Classe Map:
def resize(self, robot_pose):
    # Atualiza o tamanho do mapa de acordo com a posicao do veiculo.
    # Se o alcance maximo definido pelo mapa for menor que a distancia
    # entre as bordas do mapa e a posicao do veiculo, sao adicionadas
    # novas linhas e novas colunas.
    # Retorna a quantidade de linhas e colunas adicionadas antes
    # da posicao do veiculo para corrigir sua pose.
```

```
50
51      # self.grid contem a matriz de duas dimensoes que representa o mapa:
52      cols = len(self.grid[0]);
53
54      # Adiciona linhas no fim da matriz:
55      # self.no_cells represena o alcance maximo
56      # dos sensores em numero de celulas
57      new_lines_end = robot_pose['y'] + self.no_cells;
58      if(new_lines_end >= len(self.grid)):
59          for i in range(new_lines_end - len(self.grid) + 1):
60              self.grid.append([0.5] * len(self.grid[0]));
61
62      # Adiciona linhas no inicio da matriz:
63      new_lines = robot_pose['y'] - self.no_cells;
64      if(new_lines < 0):
65          for i in range(new_lines * -1):
66              self.grid.insert(0, [0.5] * len(self.grid[0]));
67
68      # Adiciona colunas no fim da matriz:
69      new_columns_end = robot_pose['x'] + self.no_cells;
70      if(new_columns_end >= len(self.grid[0])):
71          for row in self.grid:
72              row += [0.5] * (new_columns_end - len(row));
73
74      # Adiciona colunas no inicio da matriz:
75      new_columns = robot_pose['x'] - self.no_cells;
76      if(new_columns < 0):
77          for row in self.grid:
78              row = ([0.5] * (new_columns * -1)) + row;
79
80      new_columns *= -1;
81      new_lines   *= -1;
82      return (new_columns, new_lines, new_columns_end - cols, new_lines_end - ↘
            cols);
```

APÊNDICE C – Algoritmo de Mapeamento

```
import math;

# Classe GridMAP:
def update(self, map_data, robot_pose):
# Atualiza a posicao das celulas do mapa
# que foram recebidas em 'data'.
# A atualizacao e a media entre a nova medida e a medida anterior.
# O resultado e a probabilidade de ocupacao de cada celula.

for data in map_data:
    # Nova medicao:
    new_data = 0;

    # Se existir um obstaculo:
    if(data[2] is True):
        new_data = 1;

    # Atualiza a probabilidade da celula:
    y, x = data[0], data[1];
    self.grid[y][x] = (new_data + self.grid[y][x]) / 2;

    # Atualiza a probabilidade das demais celulas:
    orientation = math.radians(data[3]);
    x += int(math.cos(orientation)) * -1;
    y += int(math.sin(orientation));
    while((x != robot_pose['x']) or (y != robot_pose['y'])):
        self.grid[y][x] = (0 + self.grid[y][x]) / 2;
        x += int(math.cos(orientation)) * -1;
        y += int(math.sin(orientation));

return;
```