UNIVERSIDADE FEDERAL DO TRIÂNGULO MINEIRO
INSTITUTO DE CIÊNCIAS TECNOLÓGICAS E EXATAS
MESTRADO PROFISSIONAL EM INOVAÇÃO TECNOLÓGICA

WILLIAN BAUNIER DE MELO

Robô explorador térmico de ambientes utilizando rede neural e algoritmo Bug

Uberaba
2020

WILLIAN BAUNIER DE MELO

Robô explorador térmico de ambientes utilizando rede neural e algoritmo Bug

Dissertação apresentada ao Curso de Mestrado Profissional em Inovação Tecnológica da Universidade Federal do Triângulo Mineiro como requisito parcial para obtenção do título de mestre.

Orientador: Prof. Dr. David Calhau Jorge.

Uberaba

2020

Catalogação na fonte: Biblioteca da Universidade Federal do Triângulo Mineiro

D32r

De Melo, Willian Baunier
Robô explorador térmico de ambientes utilizando rede neural e algoritmo Bug / Willian Baunier de Melo. -- 2020.
110 f. : il., graf., tab.

Dissertação (Mestrado Profissional em Inovação Tecnológica) -- Universidade Federal do Triângulo Mineiro, Uberaba, MG, 2020
Orientador: Prof. Dr. David Calhau Jorge
Coorientador: Prof. Dr. Vinicius Abrão da Silva Marques

1. Robôs. 2. Raspberry Pi (Computador). 3. Redes neurais (Computação). 4. Navegação autônoma. I. Jorge, David Calhau. II. Universidade Federal do Triângulo Mineiro. III. Título.

CDU 007.52:004.032.26

WILLIAN BAUNIER DE MELO

ROBÔ EXPLORADOR TÉRMICO DE AMBIENTES UTILIZANDO REDE NEURAL E ALGORITMO BUG

Trabalho de conclusão apresentado ao Programa de Mestrado Profissional em Inovação Tecnológica da Universidade Federal do Triângulo Mineiro, como requisito para obtenção do título de mestre.

Uberaba, 04 de fevereiro de 2020

Banca Examinadora:

Prof. Dr. David Calhau Jorge
Orientador – UFTM

Prof. Dr. Virgílio de Melo Langoni
Membro titular – UFTM

Prof. Dr. Leandro Aureliano da Silva
Membro Titular – FACTHUS

AGRADECIMENTOS

Agradeço a Deus por tudo na vida e principalmente por mais esta oportunidade de enriquecimento intelectual e por me ajudar nos momentos difíceis.

Aos meus pais Silvia e Carlos e ao meu irmão, Gabriel, por dar o suporte que precisei e incentivar incansavelmente pela busca do conhecimento.

A minha namorada Ana Flávia por sempre me dar conselhos e me apoiar nos momentos difíceis e nos problemas que enfrentei nessa jornada.

Ao Prof. Dr. David Calhau Jorge por ensinar-me, orientar-me, ter paciência, e transferir parte de seu extenso conhecimento que foi primordial para conclusão deste trabalho.

Ao Prof. Dr. Vinicius Abrão da Silva Marques, por ter a disponibilidade de coorientar, e pela grande ajuda, principalmente nos testes em laboratório, além de vários conselhos de grande importância para o trabalho.

Aos membros da banca pela disponibilidade de ler este trabalho e prontificar imediatamente após o convite.

A Universidade Federal do Triângulo Mineiro por disponibilizar laboratórios e por disponibilizar horário flexível para cumprimento da carga horária obrigatória.

RESUMO

A navegação autônoma necessita que os agentes artificiais consigam entender e adaptar ao ambiente, principalmente na ação de desvios de obstáculos tanto estáticos quanto dinâmicos. O robô explorador realiza o sensoriamento do ambiente ao seu redor e aprende com as experiências bem-sucedidas de exploração para traçar as melhores rotas e desviar dos obstáculos. O presente trabalho descreve um robô explorador de ambientes que possui um sistema de câmera e sensores de temperatura para identificação de zonas térmicas dentro do mapa explorado. Como o robô visa minimizar custos de fabricação para possíveis missões destrutivas a imagem térmica é composta por uma solução de processamento de uma câmera e um sensor térmico matricial. O robô é controlado por um Raspberry Pi modelo 3 b+, que possui poder de processamento adequado à tarefa, principalmente pelo custo computacional do processamento de imagens da câmera. Os sensores de ultrassom foram responsáveis por “sentir” o ambiente, e a locomoção autônoma é realizada baseada numa solução combinada de algoritmo inteligente Bug e redes neurais artificiais. Uma interface acessada pela internet via IP foi utilizada para controlar o robô tanto em modo manual e modo automático. As aplicações do robô podem ser diversas, desde incêndios a exploração de lugares inóspitos.

Palavras-chave: Robô explorador. Raspberry Pi. Redes neurais. Navegação autônoma.

ABSTRACT

Autonomous navigation requires that artificial agents be able to understand and adapt to the environment, especially in the action of both static and dynamic obstacle diversions. The explorer robot senses the surrounding environment and learns from successful exploration experiences to plot the best routes and avoid obstacles. The present work describes an environment explorer robot that has a camera system and temperature sensors to identify thermal zones within the map explored. Because the robot aims to minimize manufacturing costs for possible destructive missions, the thermal image is comprised of a camera processing solution and a matrix thermal sensor. The robot is controlled by a Raspberry Pi model 3 b +, which has adequate processing power for the task, mainly due to the computational cost of camera image processing. Ultrasound sensors were responsible for "feeling" the environment, and autonomous locomotion is performed based on a combined solution of intelligent Bug algorithm and artificial neural networks. An interface accessed over the internet via IP was used to control the robot in both manual mode and automatic mode. Robot applications can be diverse, from fires to exploring inhospitable places.

Keywords: Explorer robot. Raspberry Pi. Neural networks. Autonomous Navigation.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

ABS – Acrilonitrila Butadieno Estireno

API – *Aplication Programing Interface*

ASCII – *American Standard Code for Information Interchange*

CA – Corrente Alternada

CAD – *Computer Aided Design*

CC – Corrente Contínua

CCN – Complete Coverage Navigation

CSI – *Camera serial interface*

CSS – *Cascading Style Sheets*

FPS – *Frames Per Second*

GND – *Ground*

HV – *High Voltage*

HTML – *Hypertext Markup Language*

I2C – *Inter-Integrated Circuit*

IA – Inteligência Artificial

IDE – *Integrated Development Environment*

IBA – *Intelligent Bug Algorithm*

IOT – *Internet of Things*

IP – Internet Protocol

LIDAR – *Light Detection and Ranging*

LiPo – *Lithium-ion Polimer*

LV – *Low Voltage*

MLL – *Machine Learning Library*

mA – MiliAmpère

PWM – *Pulse Width Modulation*

SDA – *Serial Data*

SCL – *Serial Clock*

REST – *Representational State Transfer*

Wi - Fi – Wireless Fidelity

RELU – *Reticitified Linear Unit*

V – *Volts*

VCC – Tensão em corrente contínua

SUMÁRIO

1 INTRODUÇÃO

A robótica é um ramo da ciência que integra várias áreas de conhecimento e faz parte da grade curricular de alguns cursos como: engenharia mecânica, elétrica, automação, dentre outros. A tecnologia advinda deste conhecimento já é utilizada há séculos, porém de uma maneira um pouco diferente, menos automatizada e com sistemas mais rudimentares, como os autômatos, máquinas mecânicas antigas com certo nível de automação utilizadas para aplicações específicas. Essa busca pela solução das necessidades básicas foi o gatilho para a busca do uso da robótica: “Os cientistas da época já haviam começado e continuado a elaboração de projetos de implantação da substituição do homem pela máquina, obtendo conhecimentos tecnológicos para suprir a demanda de projetos ainda mais audaciosos”. (ROMANO, 2002, p. 1).

Porém, no século XX, a expansão dos métodos computacionais e da eletrônica adicionaram funcionalidades antes inimagináveis e expandiram as barreiras dos sistemas autônomos.

Os robôs são planejados e construídos para um objetivo específico e dependem basicamente de três características, a parte mecânica, responsável pelo movimento e habilidades físicas da máquina, a parte elétrica, composta por sistemas de atuadores, sensores e controladores que executarão os comandos e transmitirão para a parte mecânica, e, finalmente, a parte de controle, que pode ser um algoritmo de inteligência artificial (IA), controle remoto ou um controle híbrido.

Os robôs de navegação autônoma são sistemas interessantes, pois utilizam sua própria capacidade de tomar decisões para deslocar-se de maneira independente. Por isso, esses sistemas que utilizam sua tomada de decisão baseados no sensoriamento do ambiente e possuem visão computacional são significativamente explorados em pesquisas na área da robótica, automação e computação, aumentando ainda mais suas aplicações (ARAÚJO; LIBRANTZ, 2006).

A exploração de ambientes é uma das áreas em grande expansão nas aplicações de robótica. Uma aplicação dessa área que está em crescimento é o mapeamento de ambientes hostis, onde o robô consegue navegar por ambientes com altos níveis de gases tóxicos ou com temperatura muito elevada, o que representa um risco alto para a segurança de uma pessoa. Assim, todos os dados são aquisitados pelo robô conforme a programação do operador, com grande

adaptabilidade para diferentes tipos de ambientes apresentando resultados mais satisfatórios. O trabalho baseia-se na utilização de um robô autônomo para exploração de ambientes e identificação de regiões térmicas e fontes de calor, baseado em uma combinação de redes neurais Perceptron de multicamadas e *Bug* inteligente para locomoção.

O projeto vislumbra a possibilidade da utilização do robô em incêndios, para a identificação das zonas quentes sem a necessidade de adentrar as construções, e a característica do baixo custo também possibilita o uso para explorações destrutivas. Outra aplicação é a sua utilização em indústrias, em locais onde seja inóspito para o trabalhador realizar a coleta de dados, seja pela presença de gases nocivos ou ambiente confinado, o que seria simplesmente contornado pela utilização do robô.

O robô foi construído no formato de um carro, composto por: sensores ultrassônicos, que serão responsáveis por indicar a distância dos obstáculos, uma bússola digital e sensores encoders ópticos, para localização do robô, motores elétricos, que farão a locomoção e movimentos do carro, uma câmera térmica, que foi utilizada para captar a temperatura e as zonas de calor do ambiente explorado, e o Raspberry Pi 3 B+, um “minicomputador” que fará o processamento do algoritmo e tratará as entradas dos elementos sensores e definirá a ação a ser realizada, o que refletirá no sinal de saída aos motores.

No trabalho de ZOHAIB et al (2013), foi proposto um novo tipo de estratégia de locomoção para os algoritmos Bug, com tomada de decisões visadas ao objetivo e a partir deste algoritmo foi utilizado uma integração com as redes neurais artificiais como melhor forma de tomada de decisões para convergência dos valores dos sensores ultrassônicos. Alguns outros trabalhos similares de exploração serão apresentados no capítulo de referencial teórico. A navegação autônoma poderá ser usada em lugares onde a visão da câmera estiver comprometida, seja por fumaça ou muita pouca iluminação, para navegação manual.

O projeto de um robô autônomo deve ser totalmente integrado, ou seja, deve possuir uma parte mecânica, elétrica e um algoritmo bem estruturados e que “conversem” bem entre si, pois qualquer alteração mecânica pode diminuir o rendimento do robô, ou um algoritmo pouco adaptável pode fazer com que o robô perca-se no ambiente ou não chegue a convergir para um ponto ótimo de operação.

1.1 OBJETIVOS

1.1.1 Objetivo geral

Projetar um sistema de navegação autônomo capaz de identificar as zonas de calor utilizando a placa Raspberry Pi como processador central.

1.1.2 Objetivos específicos

- Criar um algoritmo capaz de explorar o ambiente, utilizando uma metodologia combinada de redes neurais e algoritmo Bug;
- Criar uma solução de câmera térmica de baixo custo;
- Desenvolver uma interface simples e intuitiva para o operador;
- Criar sub-rotinas para aquisição dos dados dos sensores utilizados.

2 REFERENCIAL TEÓRICO

A robótica moderna é uma área em grande expansão científica devido à facilidade gerada por seus produtos autônomos e em consequência da revolução tecnológica das últimas décadas. Devido ao alto valor agregado, gerado pela fusão de sistemas elétricos, mecânicos e ferramentas de programação, os robôs tornaram-se símbolo da evolução e da capacidade de criação do ser humano (ROMANO, 2002).

Os diferentes tipos de robôs estão intrinsecamente ligados aos seus objetivos finais. Assim, existem robôs com diferentes aspectos e usos, que variam desde braços robóticos encontrados na indústria a humanoides que simulam a fala e a forma de andar de humanos. Segundo Perez et al. (2013), há alguns anos, a robótica era associada a braços manipuladores, responsáveis pela criação de algum bem de consumo. Porém, nos últimos anos, esse conceito vem se modificando e, tornou-se comum a interação com robôs no dia a dia, um exemplo habitual é o aspirador de pó Roomba ou o robô cão Aibo (SONY, 2017).

Os robôs exploradores são dotados de visão, força e mobilidade que ajudam também em operações de busca ou até para identificar situações de perigo ou explosivos. Um exemplo bem imponente é o jipe-robô explorador *Curiosity Rover* (NASA,[d.a. 2011]), responsável pela exploração de Marte, que se tornou o veículo mais avançado tecnologicamente em território marciano e possui grande variedade de equipamentos de coleta de dados.

A inteligência artificial é um dos ramos da ciência em larga expansão nas últimas décadas, principalmente por ser uma ferramenta que tenta simular a capacidade humana de pensar, raciocinar e tomar decisões em situações específicas ou gerais. Este conceito surgiu a partir da segunda guerra mundial com a criação de computadores e máquinas avançadas que faziam cálculos.

Existem dois tipos de abordagens principais em algoritmos de inteligência artificial: a cognitiva e a conexionista. A cognitiva, ou descendente, dá prioridade na simulação de como o ser humano raciocina. Esse tipo de pensamento tenta achar uma relação entre comportamentos inteligentes baseados em aspectos psicológicos e processos algorítmicos. Já a abordagem conexionista, ou também denominada ascendente, baseia-se no estudo do funcionamento do cérebro, dos neurônios e suas conexões. (MARROQUIN, A. et al, 2017).

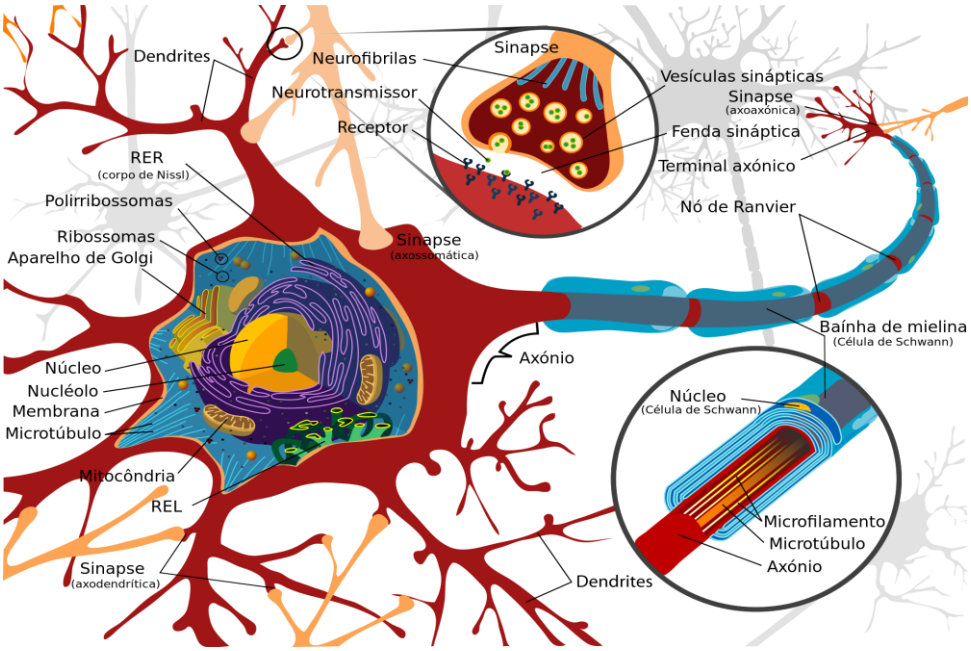
2.1 REDES NEURAIS

As redes neurais artificiais surgiram inspiradas no modelo de sistema nervoso central de seres biológicos, sendo uma importante ferramenta de inteligência artificial. Além disso ela pode ser usada para reconhecimento de padrões e aprendizado por máquina. (HAYKIN, 2001).

Este tipo de IA surgiu na década de 40 com o trabalho de McCulloch e Pitts e tinha como ideia principal fazer uma analogia entre os neurônios biológicos e sistemas elétricos. (HAYKIN, 2001).

Para entender melhor a rede neural artificial é importante conhecer o próprio sistema nervoso humano. O cérebro humano possui bilhões de neurônios que estão conectados entre si e cada neurônio possui dendritos e axônios. Pela Figura 1 pode-se ver exatamente a estrutura de um neurônio biológico. Os dendritos recebem as informações de outros neurônios, por impulsos elétricos, essa informação é tratada no corpo do neurônio e mais tarde é enviada a outros neurônios passando pelos axônios. Essa troca de informações entre os neurônios constituem a rede neural.

Figura 1 – Modelo de neurônio biológico



Fonte: Ruiz, 2007

O neurônio a todo o momento recebe sinais de vários neurônios e essas informações podem ser atenuadas, bloqueadas ou transmitidas para os dendritos,

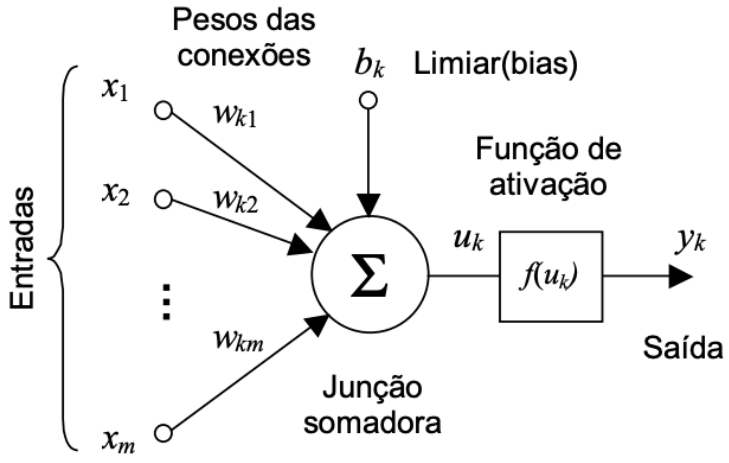
processo em que atuam os estímulos sinápticos, dependendo da importância e da relevância das informações para aquele neurônio.

2.1.1 Topologia

O modelo matemático do neurônio artificial foi proposto em analogia a essa estrutura humana, onde os impulsos elétricos foram substituídos pelas variáveis de entrada e os estímulos sinápticos, serão representados por pesos, que estimularão o neurônio dependendo de seu valor e da entrada.

A Figura 2 representa a estrutura básica de um neurônio artificial. Os sinais de entrada (x_1, x_2, x_3, … , x_n) são sinais externos normalizados para aumentar a eficiência dos cálculos, os pesos sinápticos (w_{k1}, w_{k2}, w_{k3} , … , w_{kn}) são os valores ponderados que demonstram o nível de importância de cada entrada do neurônio. O somador agrega todos os valores de estímulos produzidos pelo produto de cada entrada x pelo seu respectivo peso w_k a fim de produzir um nível de ativação. O bias apenas é acrescentado ao modelo para garantir que o neurônio apresente saída não-nula, mesmo que possua todas as entradas nulas. Esse potencial de ativação, subtraído do elemento b_k gerará um resultado que será a base para a função de ativação (LIMA, 2015). Esse elemento b_k representa uma polarização externa (bias), e tem por finalidade o aumento ou decréscimo da função de ativação. A função de ativação gera a saída e limita os valores a intervalos pré-determinados.

Figura 2 – Estrutura matemática do neurônio artificial



Fonte: Adaptado de Zambiasi, 2011

A função de ajuste é o processo de treinamento de uma rede neural em um conjunto de entradas para produzir um conjunto associado de saídas de destino. Uma vez que a rede neural se encaixe nos dados, ela forma uma generalização da relação de entrada-saída e pode ser usado para gerar saídas para insumos que não foram treinados.

A saída do neurônio artificial (Y) é dado pelo somatório do valor das entradas, onde (x_i) representa a i-ésima entrada, multiplicados pelos respectivos pesos (W_i) e adicionado o valor do bias (b), conforme Equação 1.

$$Y = \sum^{n} x_i W_i + b \tag{1}$$

Os pesos das redes neurais geralmente são ajustados conforme padrões pré-determinados dependendo do modelo escolhido.

2.1.2 Função de ativação

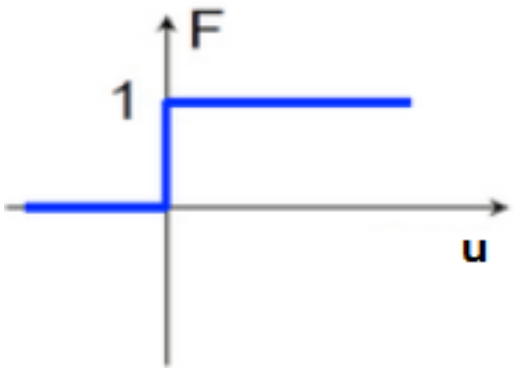
As funções de ativação são utilizadas para dar mais poder na resolução de problemas complexos introduzindo um componente não-linear. Sem essa função, o que acontece basicamente é uma transformação linear do sinal, tornando o modelo uma simples regressão linear. Através dessas funções é definido o quanto a informação daquele neurônio é importante ou se ela pode ser descartada.

Diferentes tipos de funções de ativação podem ser utilizados para resolução de problemas, cada uma com suas características que as tornam mais adequadas em determinadas aplicações ou menos eficientes em outras.

2.1.2.1 Função degrau

Essa função é baseada em um limiar de ativação, ou seja, o neurônio assume valor lógico “1” se o valor do neurônio é não-negativo, caso contrário, se o neurônio for negativo, ele recebe valor lógico “0” deixando de o ativar. Essa função é muito simples e pode ser usada para um classificador binário. A Figura 3 mostra a função ativada para valores maiores e iguais a zero e não ativada para o restante dos valores.

Figura 3 – Função de ativação do tipo degrau



Fonte: Naranjo, 2014

2.1.2.2 Função sigmóide

A função sigmóide que, graficamente, tem em sua resposta um comportamento em formato de S, é a uma das formas mais comuns de ativação. Ela possui uma equação que é estritamente balanceada nos componentes lineares e não-lineares. (HAYKIN, 2001). A Equação 2 representa matematicamente a função sigmóide.

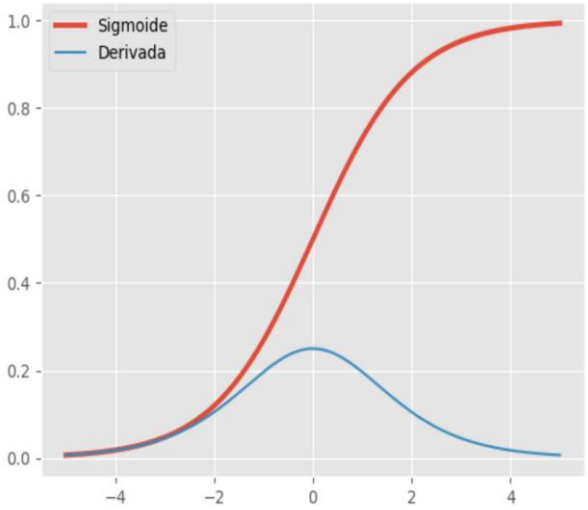
$$F(x) = \frac{1}{(1 + e^{-x})} \quad (2)$$

A função possui claramente limites entre 0 e 1, pois o menor valor possível seria quando x tender ao infinito negativo, como o expoente já era negativo, resulta no número neperiano elevado ao infinito e consequentemente o resultado final da função tenderá a zero. Quando o valor de x tende ao infinito positivo, o número neperiano assume valor nulo e a função tenderá para o valor máximo de um.

A função possui a característica de ser continuamente diferenciável. Ela apresenta a vantagem de possuir componentes não lineares, existindo uma saída que também não é linear. Na Figura 4 o formato em S da curva da função é evidente. Um problema encontrado neste tipo de função é a saturação da derivada perto dos valores extremos, fazendo com que eles tendam a zero. Com isso

acontece o desvanecimento do gradiente nessas regiões e a rede deixa de aprender.

Figura 4 – Função de ativação do tipo sigmóide



Fonte: Alves, 2017

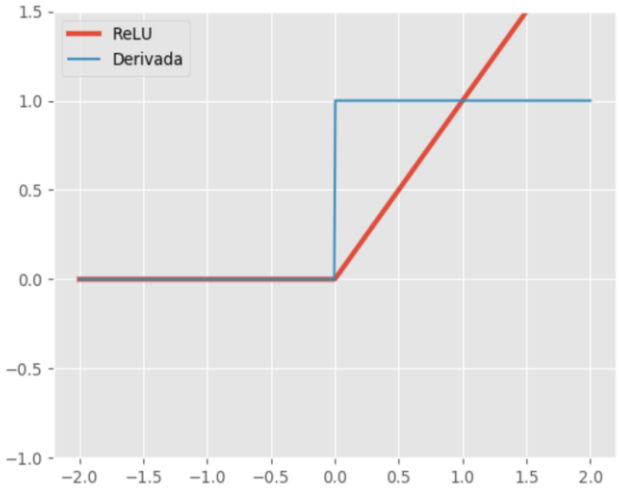
2.1.2.3 Função RELU

Essa função é a ativação linear retificada (RELU), que funciona como uma função de ativação identidade para valores maiores que zero e, caso contrário, para valores negativos, sua saída assume valor zero. Essa função é representada numericamente pela equação 3. (YAROTSKY, 2017).

$$RELU(x) = \max[0, x] \qquad RELU\ (x) = \begin{cases} 1, se\ x \geq 0 \\ 0, se\ x < 0 \end{cases} \tag{3}$$

A função RELU é não linear, e sua principal vantagem é que ela não ativa todos os neurônios ao mesmo tempo. Pela Figura 5 percebe-se que metade do domínio é 0. Logo se a entrada for negativa ela será transformada em zero, desativando o neurônio. Essa seletividade da função traz uma eficiência maior que a sigmoidal, e garante uma maior velocidade de processamento na computação. Essa função foi utilizada para ativação das camadas intermediárias das redes neurais propostas.

Figura 5 – Função de ativação do tipo linear retificada



Fonte: Alves, 2017

2.1.2.4 Função SOFTMAX

A função Softmax transforma as funções de saída de cada classe para valores entre 0 e 1, mesmo que anteriormente existisse componentes negativos ou maiores que 1, e, em seguida, divide pela soma das saídas, sendo assim muito útil para problemas de classificação de várias classes distintas. (WANG, 2016). Sua função matemática é dada pela equação 4:

$$\sigma(z)_i = \frac{e^{zi}}{\sum_{j=1}^{k} e^{z_j}} \quad para\ i = 1, \dots, K \ \ e \ \ z = (z_1, \dots, z_k) \ \epsilon \ \mathbb{R}^K \tag{4}$$

Na Equação 4, para cada elemento z_i do vetor z é aplicado uma função exponencial e depois são normalizados dividindo-os pela soma dos seus exponenciais, a normalização garante que a soma dos componentes da função $\sigma(z)$ seja 1. (WANG, 2016).

A Softmax é constantemente utilizada na camada de saída da rede neural, pois é onde deve ser gerado as probabilidades para os valores de entrada. Logo para ativação da camada de saída da rede neural do robô foi utilizada essa função, visto que a função principal da rede proposta é a classificação.

2.1.3 Perceptron de uma camada

As redes neurais mais tradicionalmente usadas são as chamadas Perceptron, desenvolvidas em 1958 por Frank Rosemblat, que possuem sinapses com pesos variáveis, ajustados na fase de treinamento. Essa rede é bem simples e, basicamente, é um modelo que transforma várias entradas em uma saída binária.

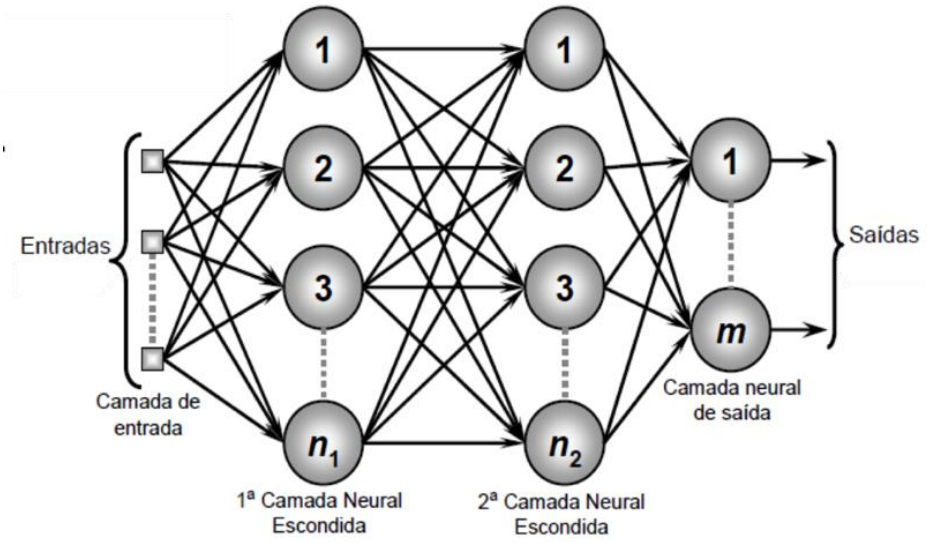
Considerando um modelo com entradas x_1, x_2, x_3 e seus pesos w_1, w_2, w_3 que são números reais expressando a relevância de cada entrada. A saída de cada neurônio será binária e é representada pela soma ponderada da multiplicação desses valores, $\sum x_i\, w_i$, de forma que eles ultrapassem um valor mínimo de ativação do neurônio, uma espécie de gatilho, que pode ser simbolizado pelo bias. Assim quando a soma das entradas e pesos é maior que o bias, o Perceptron dispara. Caso contrário, ele não atuará. (SILVA; SPATTI; FLAUZINO, 2010).

Esse tipo de rede possui três estágios: o primeiro recebe as entradas exteriores e possui conexões fixas, o segundo recebe o sinal da primeira camada que é transmitido segundo os pesos ajustáveis, e envia os sinais para o terceiro estágio que é a saída do sistema, ou seja, a resposta do sistema. O sistema de uma camada consegue resolver problemas lineares, porém, caso não seja linear, deve-se então utilizar um Perceptron de mais camadas. (SILVA; SPATTI; FLAUZINO, 2010).

2.1.4 Perceptron multicamadas

As redes Perceptron multicamadas consistem em várias camadas de neurônios conectados por ligações sinápticas e cada neurônio de uma camada tem conexão com o neurônio da próxima. Essas conexões ou pesos transformam o sinal de entrada transferindo os neurônios da camada de entrada à camada de saída, onde se obtém a resposta desejada (MEDEIROS, 2003). A camada intermediária da rede é importante pela caracterização da não linearidade, o que expande significativamente o campo de aplicações em problemas reais.

Figura 6 – Esquema do perceptron multicamadas



Fonte: Biscaro, 2017

A Figura 6 mostra a estrutura básica da rede. A camada de entrada é composta por sinais de entrada que são transmitidos às camadas intermediárias, ou também conhecidos como camadas ocultas. Durante as conexões os sinais sofrem adaptações pela ponderação de seus pesos, até serem transmitidos a camada de saída.

Segundo Silva et al. (2010), as redes multicamadas possuem várias aplicações, em diferentes tipos de problemas. As aplicações básicas são: reconhecimento de padrões, identificação e controle de processos, previsão de séries temporais, otimização de sistemas e aproximação de funções. Por ser amplamente utilizada em reconhecimento de padrões e classificação, a arquitetura da rede neural deste trabalho foi baseada nesta topologia.

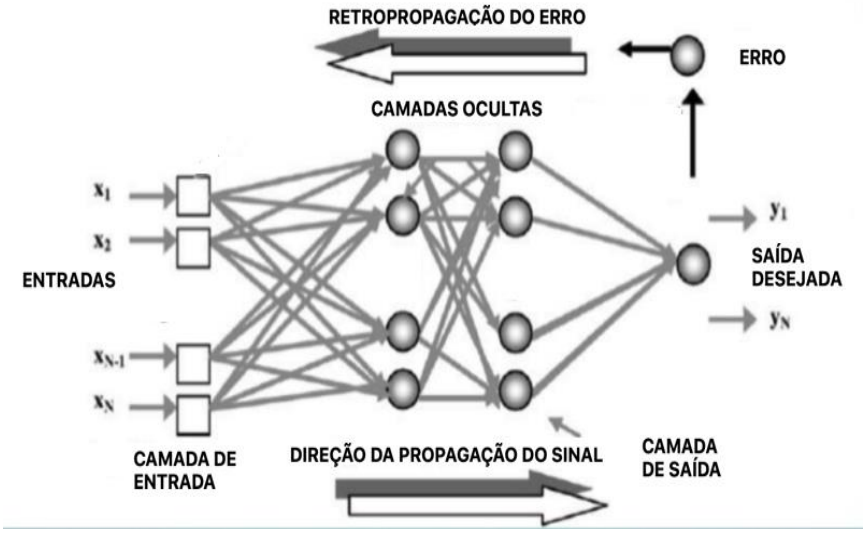
O aprendizado da rede divide-se em: supervisionado e não supervisionado. No supervisionado considera-se que o operador conheça a resposta desejada e indique à rede o padrão de resposta para a entrada inserida. Já na não supervisionada não existe um padrão externo para saída, a própria rede analisa e interpreta os dados para definir padrões encontrados. No caso do Perceptron multicamadas utiliza-se um treinamento supervisionado.

Esta rede multicamadas pode usar diferentes tipos de aprendizado, sendo o mais comum o *backpropagation*. Este tipo de aprendizado é muito utilizado na resolução de problemas não lineares (LUO ; YANG , 2008).

2.1.4.1 Algoritmo Backpropagation

O algoritmo backpropagation é utilizado para aperfeiçoar os pesos para que se alcance o valor ótimo de forma que as entradas sejam corretamente mapeadas para as saídas. Este algoritmo é dividido em duas fases, a *forward,* onde os dados de entrada são propagados camada a camada até as saídas da rede neural, enquanto que a *backward* utiliza o gradiente da função de correção do erro para atualizar o peso das conexões. O sinal do erro é propagado da saída para entrada camada por camada. Na Figura 7 é visível o sinal de propagação ou *forward* e, a fase *backward*, onde o erro ajusta os pesos das camadas anteriores.

Figura 7 – Representação do algoritmo de *backpropagation*



Fonte: Adaptado de Spacagna, 2017

Para aplicar o algoritmo de *backpropagation* a função de ativação deve ser diferenciável pois ele aplica uma correção ao peso sináptico w_{ij} que é proporcional à derivada parcial do erro em relação ao peso w_{ij} e aos nós das camadas subsequentes:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial e_j}\frac{\partial e_j}{\partial y_j}\frac{\partial y_j}{\partial v_j}\frac{\partial v_j}{\partial w_{ij}} \tag{5}$$

A equação 5 representa o gradiente da propagação dos erros onde a variável E representa a somatória dos sinais de erro em cada saída, o sinal e_j representa a saída de apenas um neurônio, o sinal y_j é o sinal de saída do neurônio j e o sinal v_j é o sinal da camada intermediária da rede. (HAYKIN, 2001)

Como visto na equação 5 esse método utiliza os erros das saídas para atualizar seus pesos, utilizando métodos iterativos para modificá-los, através da regra da cadeia, até chegar aos seus valores ótimos. O cálculo da derivada do erro e aplicação do gradiente descendente para ajustar os pesos é o ponto chave da otimização convexa. Para treinamentos bem-sucedidos, estes valores de erros tendem a diminuir com o número de iterações do sistema.

O modelo matemático do *backpropagation* é demonstrado pela Equação 6 utilizando a regra delta:

$$\Delta w_j = -\eta \frac{\partial E}{\partial w_j} \tag{6}$$

Na Equação 6 a atualização dos pesos a taxa de aprendizagem η é atribuída com um valor pequeno para que o peso não varie de forma abrupta. O sinal negativo indica que os valores seguem no sentido contrário do gradiente pois ele fornece o sentido de maior crescimento de uma função e aqui queremos o menor decréscimo possível dessa atualização.

O algoritmo de *backpropagation* possibilita a resolução em problemas de classificação e não lineares. A característica de minimização do erro pelo ajuste de pesos da rede faz com que o treinamento supervisionado assimile-se a um problema de otimização linear não restrito (SILVA et al, 2010).

Apesar deste tipo de aprendizado ser um dos mais usados, ele possui algumas limitações, como o longo período de treinamento para problemas complexos e mínimos locais. Além disso, os pesos podem ser ajustados para valores muito grandes de modo que a rede não consiga chegar a um valor ótimo satisfatório (VIEIRA, 2000).

2.2 PROCESSAMENTO DE IMAGENS

O processamento digital de imagens é um ramo da computação que tem por objetivo o tratamento de imagens, de modo que o pesquisador utilize métodos e modelos para conseguir uma imagem de saída adequada aos seus objetivos. Segundo Silva (2001), a principal função do processamento de imagens de sensoriamento remoto é o fornecimento de metodologias para facilitar a extração e identificação de informações contidas na imagem para posterior avaliação. O resultado é a produção de outras imagens, contendo informações específicas e aprofundadas.

A visão humana é um dos sentidos mais formidáveis da biologia. O sistema de visão em conjunto com o cérebro humano é capaz de reconhecer uma enorme quantidade de padrões, porém existem algumas limitações principalmente porque nem todos os detalhes da imagem podem ser processados ao mesmo tempo. O processamento de imagens serve de ferramenta para captar essas peculiaridades da imagem, ou traços específicos, que complementam a visão. (SILVA, 2001).

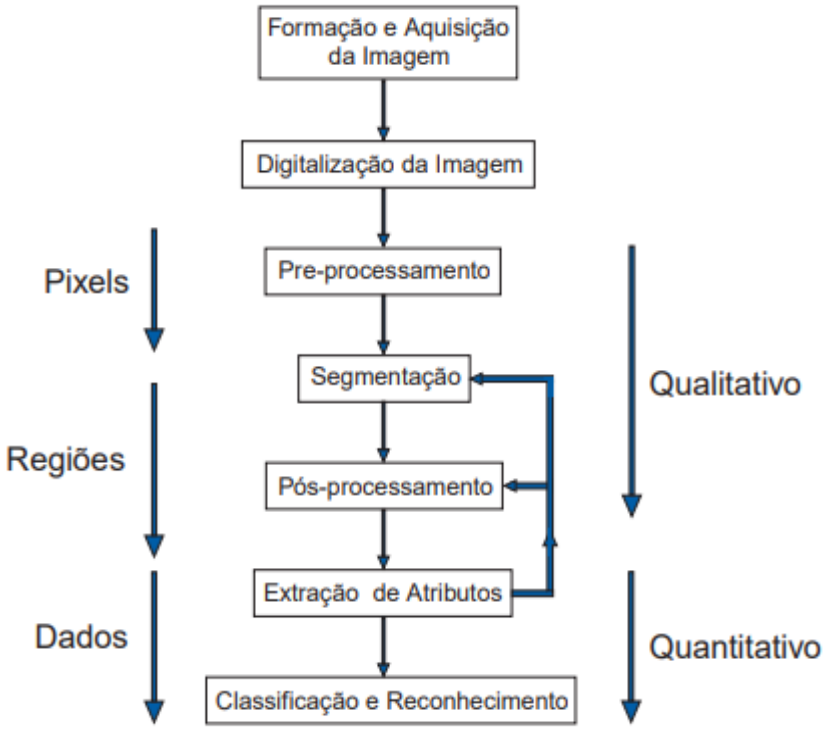
Uma imagem pode ser considerada uma função bidimensional contínua, que pode ser representada por uma função no formato $F(x,y)$, onde x e y são as coordenadas das dimensões e o valor da função é a luminosidade no ponto determinado. Essa representação é importante, pois os computadores enxergam as imagens como vetores de números, e não de forma contínua. Cada ponto de coordenadas bidimensionais das figuras é chamado de *pixel* (*Picture Element*). (FILHO; NETO, 1999).

Os sensores de imagens são utilizados para captar a composição de cores do meio externo e traduzi-las em linguagem computacional de forma que o computador possa processá-las e fazer o tratamento necessário. Esses sensores podem ser câmeras fotográficas ou câmeras de infravermelho que fornecem todas as informações necessárias na imagem adquirida. Outro fator importante é a resolução das imagens captadas, que simplesmente podem ser expressas pela quantidade de pixels, ou seja, o número de linhas por colunas. Quanto maior a resolução, maior a quantidade de *pixels* na imagem.

Algumas etapas são necessárias para transformar o que é visível no ambiente físico em imagens digitalizadas e trabalhadas pelo computador. Segundo Filho e Neto (1999), a maioria das funções de processamento de imagens é realizada por

software, com exceção apenas da aquisição e exibição. O esquema da Figura 8 lista os processos necessários e sua sequência para processamento da imagem.

Figura 8 – Etapas para processamento digital de imagem



Fonte: Esquef et al, 2003

A sequência de processos depende tanto de técnicas qualitativas como quantitativas. Primeiramente os pixels são tratados no pré-processamento, e sequencialmente são divididos em regiões agrupados em características semelhantes. Finalmente os atributos são extraídos e rotulados conforme os parâmetros de classificação. Para melhor entendimento de cada processo, os próximos subitens conterão breves descrições dos mesmos.

2.2.1 Aquisição de imagens

Para a aquisição da imagem são necessários dois elementos básicos, um elemento sensor, que irá captar os sinais eletromagnéticos e de espectro de luz visível, ultravioleta, infravermelha ou raio-x. Esse elemento fará uma conversão do espectro detectado para níveis de tensão e corrente. O segundo elemento é um conversor analógico-digital que converterá essa entrada analógica do transdutor em

um sinal digital com níveis proporcionais a potência de dois dos bits de resolução (ESQUEF et al, 2003).

Uma imagem digital é discretizada no espaço (x, y) em níveis de luminância (escala-cinza) e escala de cores, representada pela matriz de cores vermelho, verde e azul, armazenados separadamente em cada coluna.

A câmera de vídeo é o elemento sensor mais utilizado na captação de imagens e algumas já possuem mecanismos para correção de contraste e luminosidade.

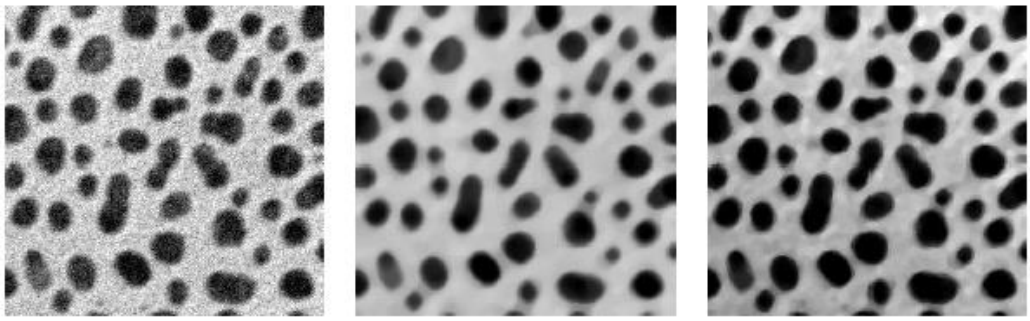
Após a imagem passar pelo conversor analógico-digital, é a vez da alteração da intensidade segundo uma tabela de conversão, a *Look up table*, ou LUT, que nada mais é que memórias com valores pré-fixados para correção instantânea da intensidade luminosa captada pelos elementos sensores.

2.2.2 Pré-Processamento

No pré-processamento da imagem acontece a eliminação de ruídos. São utilizados filtros no âmbito do sinal no tempo e espaço como técnica de atenuação de ruídos e para manipulação do plano da imagem. Outra técnica importante é a filtragem no espectro do sinal, agindo dessa maneira no aspecto da frequência. No processamento de imagens normalmente utilizam-se das duas técnicas para alcançar um melhor resultado.

Para uso do filtro no campo da frequência podemos utilizar uma matriz n x n, e escolher entre filtro passa-baixas (filtra as altas frequências), filtro passa-altas (filtra as baixas frequências) e filtro passa faixa (é a junção dos outros dois tipos de filtros, deixando uma região específica sem filtragem). Na imagem o gradiente mais forte nas escalas de cinza representa as frequências altas. Nesse caso para realçar as bordas de objetos deve ser usado filtro passa alta. Já quando ocorre variações menores na escala de cinza, frequências baixas são observadas. Para evidenciar variações de luz de fundo pode ser usado um filtro passa-baixas. (DE ALBUQUERQUE; DE ALBUQUERQUE, 2000).

Figura 9 – Processo de pré - processamento de imagem: (a) Sem filtros (b) Filtro de mediana (c) Filtro passa-altas.



(a) (b) (c)

Fonte: De Albuquerque, 2000

A Figura 9 mostra a utilização de dois tipos de filtros, (a) representa a imagem original, sem nenhum tratamento, já (b) representa um filtro de mediana responsável por reduzir o ruído causada pela gaussiana, e a Figura 4 (c) é obtida após o uso de um filtro passa-altas para realçar os contornos.

2.2.3 Segmentação

A segmentação consiste na divisão da imagem em diferentes regiões, que possuem características peculiares entre si, e os algoritmos de processamento buscam informações de um tipo “alto-nível”. Logo, a separação pode se dar classificando a região de pixels que delimitam um objeto e descartando o “fundo”, ou seja, o restante dos pixels classificados fora do contorno desejado. (DE ALBUQUERQUE, 2000).

No campo de processamento de imagens a segmentação é considerada talvez o processo mais importante no processamento da informação, pois é nesta etapa que é definido a região desejada para análise e pós-processamento. Assim qualquer erro na melhor forma de segmentar a imagem para escolha do objeto estudado pode resultar em distorções não desejadas no resultado final.

A segmentação é um processo adaptativo e requer análise de cada tipo específico de imagem para emprego de determinada técnica. Em uma mesma imagem há diferentes objetivos sendo viável utilizar diferentes técnicas conhecidas, ou criar uma nova técnica para aquela segmentação específica. No projeto da visão térmica do robô, a etapa de segmentação será importante por separar o objeto quente do “fundo frio” da imagem, assim, identificando as zonas de maior calor.

Uma das técnicas mais utilizadas é a binarização de imagens ou *image threshold*, que é baseada na similaridade entre os pixels e é utilizada quando os níveis de cinza possuem amplitude suficiente para destacar o objeto de estudo. Assim teremos uma saída binária baseada na luminosidade da imagem e destacará o objeto do fundo. (DE ALBUQUERQUE, 2000). A Figura 10 mostra a segmentação de uma imagem.

Figura 10 – Processo de segmentação: (a) imagem original (b) detecção de bordas aplicada

Fonte: Leão, 2018

A imagem original é mostrada na Figura 10 (a) e, em (b), a imagem após aplicação de uma técnica de segmentação conhecida por detecção por bordas. Ela é baseada no estudo da descontinuidade entre pixels, ou seja, baseada nas variações bruscas nos níveis de cinza, e luminância delimitando assim os prováveis contornos para um objeto.

2.2.4 Pós-processamento

Nesta etapa os defeitos e erros provenientes da segmentação da imagem serão corrigidos. Essa correção é feita através de técnicas de morfologia matemática.

A morfologia matemática é baseada em duas linhas: o elemento estruturante, que é a noção da forma básica e os operadores booleanos de conjuntos. Essas operações booleanas são realizadas entre o elemento estruturante e a imagem.

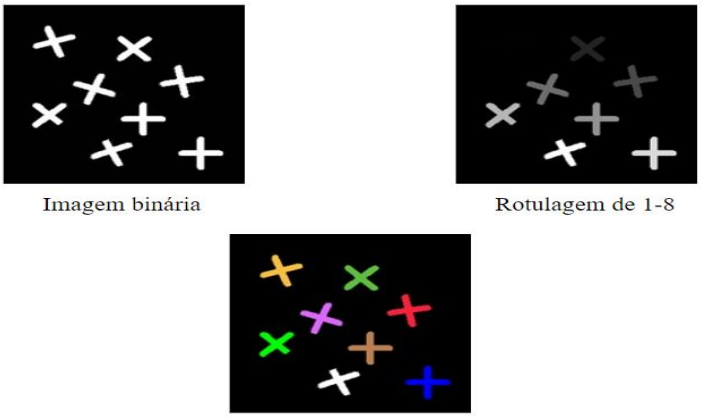
Na morfologia matemática são comumente utilizadas duas técnicas: erosão e dilatação. Na erosão acontece a divisão dos objetos que se tocam. Já na dilatação é possível ligar ou preencher furos dentro do objeto. Essas técnicas são iterativas, resultando em uma sequência de operações de erosão ou dilatação que podem ser sucessivas ou alternadas. (DE ALBUQUERQUE; DE ALBUQUERQUE, 2000).

2.2.5 Extração de atributos

Nesta etapa serão processadas as informações julgadas importantes da imagem processada. A principal técnica do processo de extração de atributos é a rotulação. Após a segmentação a imagem está dividida em regiões onde contém o objeto e o fundo da imagem. Após a segmentação é preciso criar rótulos para cada grupo de pixels agrupados, pois assim é possível parametrizar essas regiões e criar regras específicas.

Portanto, a rotulação cria "*Labels*" para identificar as regiões criadas na segmentação, que separa as regiões pertencentes a célula e as regiões entre células, para tratamento e parametrização utilizado no próximo processo. Na Figura 11 é apresentada uma imagem binária de oito objetos inseridos e rotulados numericamente em forma crescente para posterior tratamento. Depois de rotulados os objetos receberam uma cor diferente cada, referente ao seu rótulo atribuído.

Figura 11 – Rotulação de uma imagem e atribuição de características



Fonte: Fisher et al, 2003

2.2.6 Classificação

O objetivo da etapa de classificação é identificar os objetos segmentados. Das técnicas existentes nesse processo destacam-se duas: o reconhecimento e o aprendizado.

No processo de reconhecimento existe o espaço de medidas que nada mais é que um espaço de dimensões onde cada dimensão corresponde a um atributo. A escolha de cada atributo ou parâmetro, e a sua qualidade, influencia diretamente na capacidade de reconhecimento do objeto. Muitos parâmetros acabam por tornar o processo lento e complicado, porém poucos parâmetros podem não ser suficientes para atingir uma qualidade adequada. (DE ALBUQUERQUE; DE ALBUQUERQUE, 2000).

No processo de aprendizagem existem dois métodos principais já abordados no tópico de redes neurais artificiais: o supervisionado e o não supervisionado. No supervisionado o classificador informa os parâmetros de uma determinada classe e identifica a classe em questão. A partir de vários dados de entrada o aprendizado se torna mais preciso. Ao final é achado uma função de correlação entre as entradas aplicadas e as classes identificadas. Já para o aprendizado não supervisionado, os dados de entrada não terão classes definidas, sendo assim o software de aprendizagem tentará encontrar padrões que forneçam alguma relação de agrupamento entre as classes.

2.3 ALGORITMO BUG DE NAVEGAÇÃO

Um dos desafios recentes da navegação de robôs é a autonomia necessária para que ele possa completar diversos tipos de tarefas como vigilância, transporte, exploração, mapeamento, entre outros. O algoritmo de navegação autônoma deve proporcionar o caminho mais otimizado que consiga desviar de obstáculos que encontre. (ZOHAIB et al, 2013).

O primeiro algoritmo *Bug* surgiu no final dos anos 80, e esse termo foi escolhido devido a inspiração na movimentação de insetos e na maneira como desviavam de obstáculos. Estes tipos de algoritmos são completos e cada uma de suas variações possui algumas vantagens e limitações. Os tipos abordados serão: *Bug* 1, *Bug* 2, *Dist-Bug* e *intelligent Bug Algorithm (IBA)*. (YUFKA, 2009).

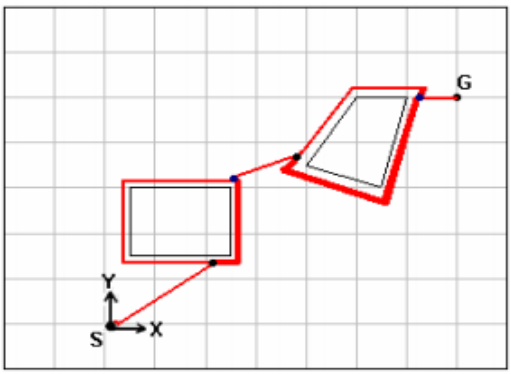
Este tipo de algoritmo possui dois comportamentos principais: o desvio e contorno ao objeto e o movimento em direção ao alvo. No processo de desvio, o robô deve seguir as bordas do obstáculo até que a condição de parada seja alcançada. Já no processo de movimento para o alvo, o robô move-se diretamente ao objetivo pelo caminho mais curto.

2.3.1 *Bug* 1

O algoritmo *Bug* 1 é o mais antigo e relativamente simples quanto os demais. Este tipo de algoritmo não se preocupa em otimizar a distância percorrida pelo robô em muitos cenários, e sim garantir que o robô chegue ao seu destino.

O processo do algoritmo basicamente inicia-se com o robô movendo-se em direção ao alvo traçando uma determinada trajetória. Caso algum obstáculo esteja no trajeto, o robô desvia seguindo as bordas do obstáculo e a cada ponto calcula a distância entre o ponto atual e o alvo salvando todos os pontos. Ao completar uma volta, o ponto de saída será aquele que apresentar o menor valor de distância. Após desviar do obstáculo o robô calcula novamente a sua trajetória e move-se diretamente ao objetivo até encontrar outro obstáculo. A Figura 12 demonstra a trajetória percorrida por um robô de um ponto qualquer ao objetivo G utilizando o algoritmo *Bug* 1.

Figura 12 – Trajetória utilizando algoritmo BUG 1



Fonte: Yufca, 2009

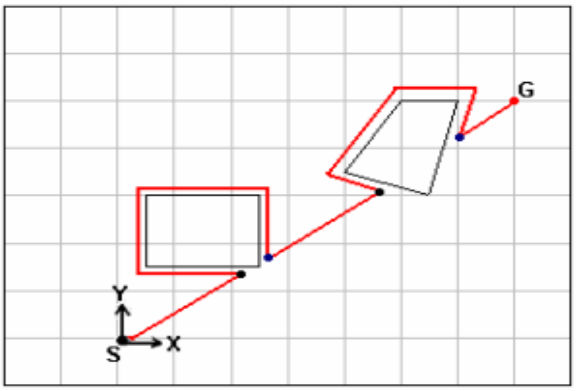
Na Figura 12 fica claro que o robô faz uma volta completa em cada obstáculo antes de chegar ao destino. Isso pode ocasionar um tempo adicional e um maior

caminho percorrido, além de ter algumas limitações em determinados formatos de obstáculos.

2.3.2 *Bug* 2

O algoritmo *Bug* 2 cria uma linha reta entre o ponto inicial do robô e o objetivo final, armazenando assim a menor distância ao alvo. O robô então move-se ao longo da reta até que um obstáculo obstrua sua passagem. Ao detectar um obstáculo, inicia-se o procedimento de desvio, sendo que o robô seguirá as paredes dos obstáculos até ele entrar na trajetória da reta traçada no começo do movimento. Após atingir essa reta o robô volta a seguir a direção do alvo. Este algoritmo otimiza o tempo gasto e o caminho percorrido em diversos cenários, em comparação com o algoritmo anterior. A Figura 13 apresenta o robô movendo-se ao objetivo G usando o algoritmo *Bug* 2.

Figura 13 – Trajetória utilizando algoritmo *Bug* 2



Fonte: Yufca , 2009

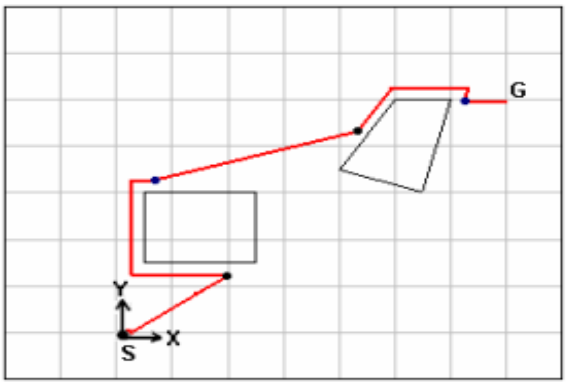
Na Figura 13 a linha reta entre a origem e o alvo na trajetória é visivelmente perceptível. O robô permanece na linha da trajetória até encontrar o obstáculo, então contorna-o até atingir o ponto pertencente a reta inicial, e continua até atingir o próximo obstáculo ou o destino final. Esse algoritmo é mais otimizado que o primeiro em questão de tempo e distância, porém ainda percorre distâncias desnecessárias. Além disso ele possui alguns problemas por não conseguir contornar obstáculos em formato de U. (YUFCA, 2009).

2.3.3 Dist-*Bug*

Este algoritmo é uma espécie de evolução dos outros dois algoritmos e, comparativamente, alcança o objetivo percorrendo uma distância menor e gastando menos tempo. (ZOHAIB et al, 2013).

No *Dist-Bug* o robô move-se em direção ao objetivo e caso encontre algum obstáculo impedindo a passagem ele começará a rotina de desvio e contornará as bordas do objeto. A diferença em relação aos outros algoritmos é no cálculo do ponto de saída. A cada movimento, a distância do ponto atual e do próximo ponto ao destino são calculados. Caso a distância do ponto atual seja maior que o do próximo ponto o robô continuará contornando. Caso contrário o robô moverá diretamente ao alvo ($D_{atual} < D_{proximo}$). A Figura 14 apresenta o trajeto de um robô movendo-se em direção ao ponto G utilizando o algoritmo *Dist-Bug*.

Figura 14 – Trajetória utilizando o algoritmo *Dist-Bug*



Fonte: Yufca, 2009

A Figura 14 demonstra o caminho utilizando a estratégia de contorno até a distância mínima ao próximo destino. Pode-se notar que a distância percorrida e o tempo foram otimizados nesse algoritmo em relação aos dois primeiros.

2.3.4 *Intelligent Bug*

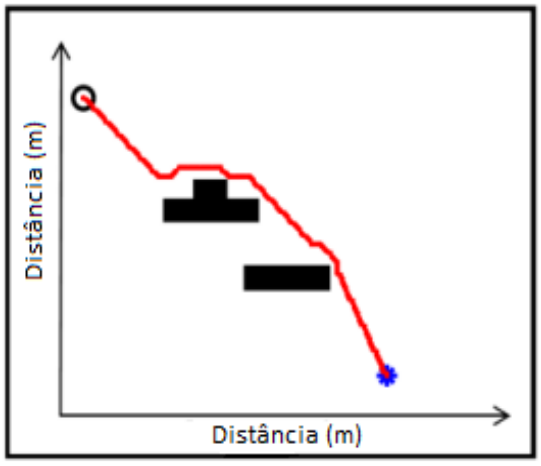
O IBA é uma variação do algoritmo dist-*Bug*, sendo a principal diferença na maneira como a tomada de decisão gira em torno do objetivo, pois o primeiro é totalmente orientado ao melhor resultado, considerado um algoritmo com tomada de

decisão gananciosa, enquanto o segundo possui algumas tomadas de decisão não orientadas totalmente aos resultados. (ZOHAIB et al, 2013).

O IBA também possui dois movimentos principais. A trajetória direcionada ao alvo e a rotina de desvio do obstáculo. Primeiramente calcula-se a menor rota entre o ponto inicial e o objetivo, em seguida o robô segue a trajetória até encontrar um obstáculo ou atingir o objetivo. Assim começa o movimento de desvio e contorno do objeto, porém o robô contornará o objeto apenas até detectar um caminho livre até o destino. Assim o ponto de saída não leva apenas em consideração a mínima distância ao destino, o caminho livre em direção ao obstáculo também é considerado. Logo o robô não precisará contornar o objeto até o ponto de mínima distância ao destino.

A Figura 15 demonstra a trajetória do robô utilizando o algoritmo em questão, e é possível perceber que o comportamento orientado ao objetivo traz uma otimização no gasto de tempo e distância percorrida.

Figura 15 – Trajetória utilizando o algoritmo intelligent-BUG



Fonte: Adaptado de (ZOHAIB,2013)

O IBA foi o algoritmo de navegação escolhido para ser utilizado no protótipo junto as redes neurais. Sua tomada de decisões orientada ao objetivo e a otimização de tempo e espaço percorrido foram os principais motivos para sua escolha.

2.4 TRABALHOS RELACIONADOS

Alguns trabalhos na área de navegação autônoma de robôs baseados em inteligência artificial são considerados importantes para a pesquisa e de alguma forma contribuíram como conhecimento para a implementação do projeto.

O trabalho de Ortiz (2012) ataca a dificuldade para a adaptação do trajeto do robô nas diferentes situações. O robô será guiado por aprendizagem de máquina utilizando uma estratégia baseada em reforço e um mapa de estado dinâmico do sensoriamento. Esse método diminui ajustes e habilita o robô a navegar em diferentes ambientes. O algoritmo é baseado no cálculo da matriz espaço de estados para maximizar o tempo antes de uma falha e maximizar a aprendizagem.

Já Vian (2007) propõe o desenvolvimento de um robô com um sistema de visão omnidirecional e sensores de ultrassom para adquirir informações do meio externo e fornece-las ao sistema de mapeamento. A parte de classificação dos diferentes obstáculos no ambiente explorado é feito por uma rede neural hierárquica, com resultados satisfatórios e eliminando possíveis confusões.

No trabalho de Luo ; Yang (2008) foi apresentado um método de cobertura completa de navegação (CCN), de cobertura total de ambiente, no qual os robôs deveriam passar por todos os caminhos do espaço para realizar o mapeamento. O modelo proposto é comparado com um método de planejamento de cobertura de trajeto completo comparado a um mapa baseado em células triangulares (Oh et al., 2004) que combina transformação da distância do caminho, algoritmo de seguimento de parede e técnica baseada em modelamento.

Para Quíntía et al (2010) a maior dificuldade da navegação foi a adaptação do trajeto do robô nas diferentes situações. O robô foi guiado pela aprendizagem de máquina utilizando uma estratégia baseada em aprendizado por reforço e um mapa de estado dinâmico do sensoriamento. O algoritmo é baseado no cálculo da matriz espaço de estados para maximizar o tempo antes de uma falha e maximizar a aprendizagem.

Um exemplo de montagem de baixo custo pode ser observado em Marroquin et al (2017) que projeta um robô explorador que utiliza s*oftware* e *hardware* abertos, cujo design objetiva a incorporação de câmeras, sensores outros periféricos para serem controlados remotamente. O robô possui locomoção a três rodas, sensores, e a visão do ambiente é feito pela câmera.

Em Zohaib et al (2013) foi proposto o uso de uma nova estratégia de algoritmo *Bug*, o *intelligent Bug Algorithm (IBA)*. Esse algoritmo mostrou-se mais otimizado quanto a distância percorrida quanto aos algoritmos *Bug* anteriores.

Outro trabalho baseado em navegação utilizando redes neurais foi apresentado por Singh (2014), em que foram treinadas diversas situações possíveis de configurações que o robô poderia vir a encontrar durante o trajeto. A rede neural respondeu muito bem ao desvio de objetos estáticos e desempenho satisfatório diante de objetos dinâmicos.

3 METODOLOGIA

O principal desafio do trabalho é o desenvolvimento e integração entre *hardware* e *software*. O *hardware* consiste na montagem da estrutura do robô, baseado em um chassi em forma de carro, composto por motores, processador central, sensores e câmera. Já a parte de *software* é composta por rotinas de aquisição dos dados de entrada dos sensores, processamento dos dados da câmera térmica, utilização das redes neurais e algoritmo *Bug*. Os algoritmos responsáveis por executar as funções descritas foram desenvolvidos em Python, uma linguagem de alto nível, orientada a objetos, que possui o desenvolvimento aberto, e para o desenvolvimento do aplicativo web também foi usado JavaScript e estrutura HTML (*Hypertext Markup Language*).

3.1 CONSTRUÇÃO DO ROBÔ

No chassi do robô foram instalados os sensores, câmera e demais componentes. Foram necessários vários ajustes para garantir a funcionalidade dos diversos sensores do protótipo. Para isso também foram utilizados suportes, muitos deles feitos em impressora 3D, para dar sustentação as peças. A câmera e o sensor térmico foram alocados na parte frontal do robô de forma que as imagens possam ser captadas do melhor ângulo de visão. A parte de alimentação ainda conta com conversores Buck e uma bateria LiPo (*Lithium-Ion Polimer*).

3.1.1 Raspberry Pi

O Raspberry Pi é um microcontrolador muito complexo, considerado um computador de bolso, devido ao alto poder de processamento e quantidade de memória RAM. Além disso o Raspberry Pi modelo 3 B+ conta com um alto poder de processamento, um processador com quatro núcleos e clock de 1,4 GHz. A memória RAM é de 1 GB, facilitando e tornando-a ideal para tratamento e processamento de imagens (RASPBERRY FOUNDATION, 2018).

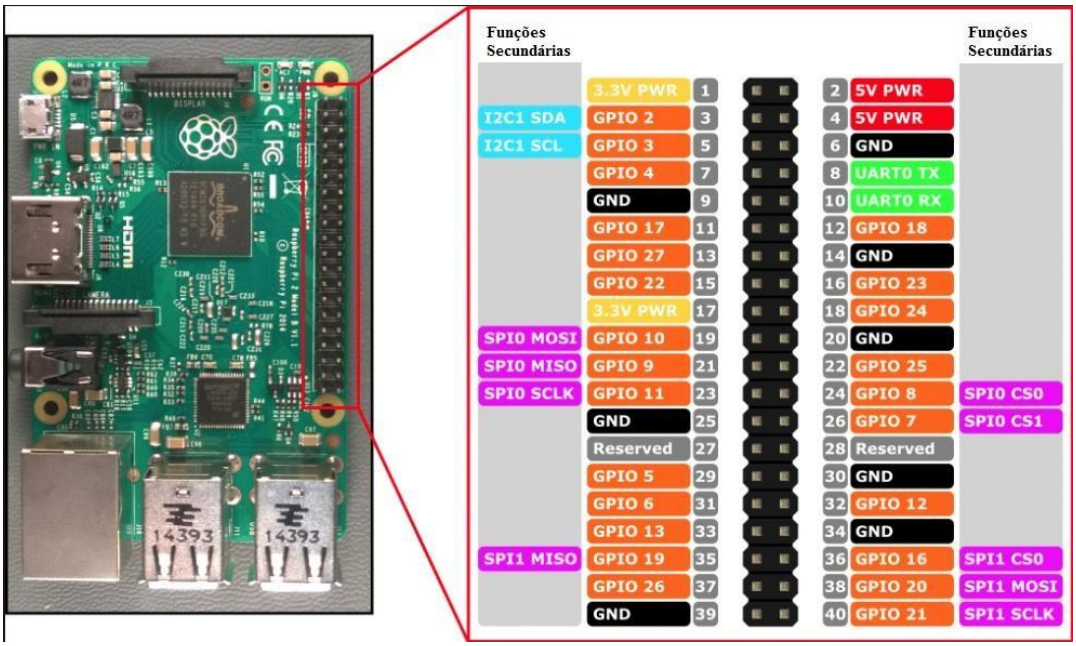
O Raspberry Pi possui um sistema operacional Linux Ubuntu, e existem várias versões compatíveis com a placa. A versão que melhor se adapta ao hardware é a

MATE, que é uma versão mais leve e processa rapidamente suas tarefas. A programação dentro do Raspberry pode ser feita em Python.

Para enviar os dados para o monitoramento externo de um computador ou celular, foi utilizado a função *Wi-Fi (Wireless Fidelity)* do *Raspberry Pi* (padrão 802.11.n), que não necessita de adaptadores e também possui *bluetooth* 4.1, para conexão em celulares.

Na Figura 16 é apresentado o Raspberry Pi 3 B+. Note que do lado direito da figura aparece um detalhamento dos pinos onde serão conectados os sensores e motores.

Figura 16 – Raspberry Pi modelo 3 B+



Fonte: Microsoft Windows IOT Core, 2017.

No projeto, o Raspberry Pi recebe as entradas dos sensores ultrassônicos da câmera, encoder, bússola, sensor termal matricial e os motores CC (Corrente contínua). O algoritmo central proposto irá receber as entradas, processá-las e definir valores para as ações necessárias. Esses valores serão transmitidos para os motores, que serão o elemento atuante do robô. Os dados ainda serão transmitidos via *Wi-Fi* para um computador ou celular. Na Tabela 1 vemos as portas que serão utilizados para conexão dos dispositivos periféricos citados anteriormente.

Tabela 1 – Mapeamento das funções do GPIO utilizados no projeto

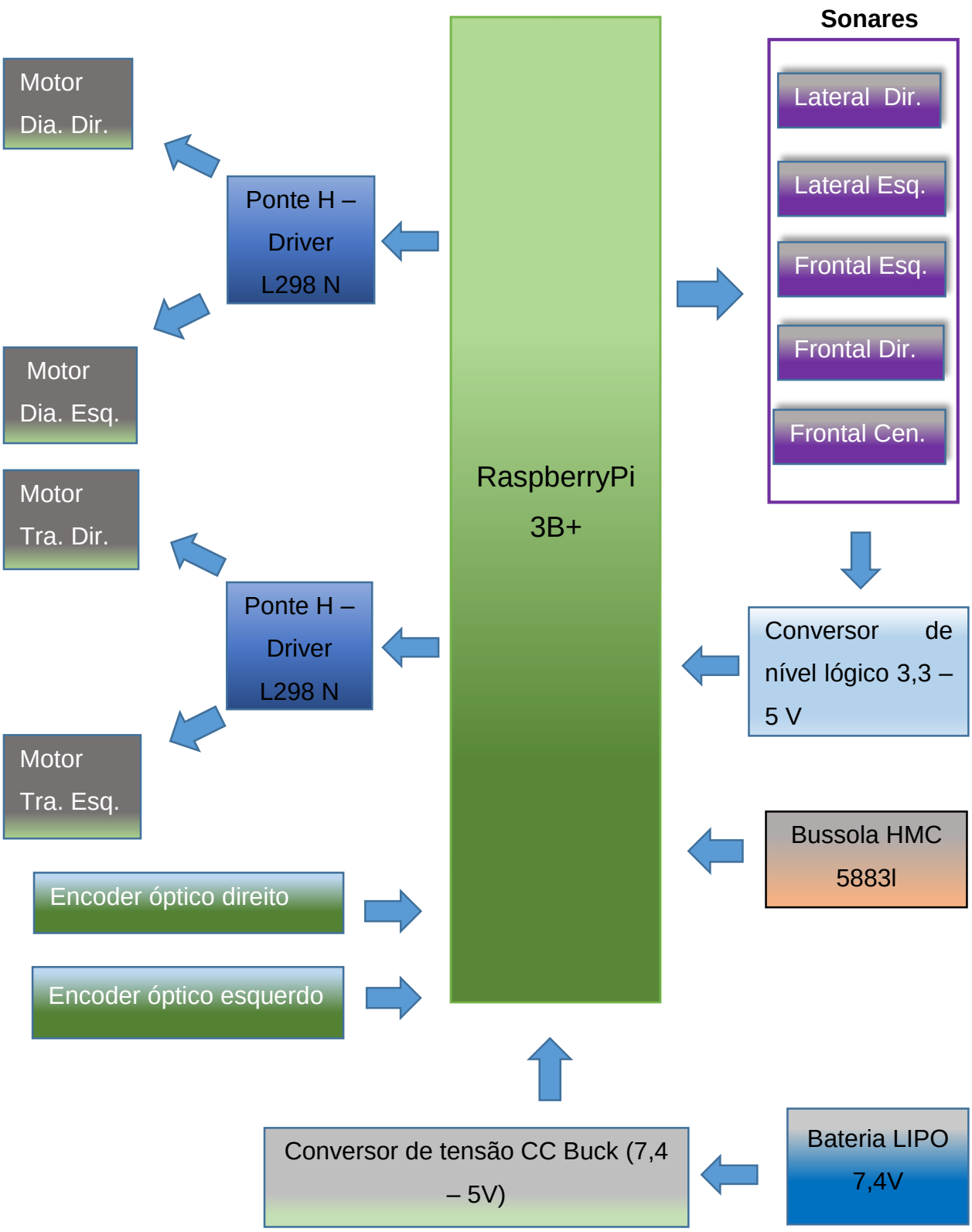
Pinos utlizados (GPIO)	Característica	Equipamento
19,11	Saída digital	Motor dianteiro direito
22,27	Saída digital	Motor dianteiro esquerdo
12,16	Saída digital	Motor traseiro direito
21,13	Saída digital	Motor traseiro esquerdo
5,6	Comunicacão I2C	Bussola HMC5883
2,3	Comunicação I2C	Sensor Termal AMG 8833
23	Entrada Digital	Encoder óptico direito
26	Entrada Digital	Encoder óptico esquerdo
24,25	Entrada,saída digital	Sensor ultrassônico lateral direito
20,14	Entrada,saída digital	Sensor ultrassônico lateral esquerdo
10,9	Entrada,saída digital	Sensor ultrassônico frontal direito
14,21	Entrada,saída digital	Sensor ultrassônico frontal esquerdo
8,7	Entrada,saída digital	Sensor ultrassônico frontal central

Fonte: Do autor, 2019

Os pinos de 5V (Volts), 3.3V e o terra (GND) foram expandidos para um barramento pois a placa Raspberry possui apenas alguns pinos de alimentação e todos os sensores necessitam de algum tipo de alimentação. Para melhor ilustrar a montagem, a Figura 17 mostra um desenho esquemático do Raspberry e a relação entre alguns componentes da montagem. Os sensores sonares foram agrupados e nomeados segundo a posição no protótipo. De maneira semelhante foram agrupados os motores.

Na Figura 17 é apresentado um esquema de blocos mostrando o fluxo da ligação dos componentes ao Raspberry. Os sentidos das setas indicam o sentido das informações transmitidas, definindo assim as entradas e saídas dos sensores. Os sensores ultrassônicos possuem duplo sentido dos dados (entrada e saída) na comunicação.

Figura 17 – Esquema de montagem do hardware em blocos



Fonte: Do autor, 2019

3.1.2 Motores CC e controle PWM.

Para locomoção do robô foram utilizados dois motores de corrente contínua alimentados por uma tensão nominal de 6 V. Neste tipo de motor o estator é composto por imãs e o comutador tem função de transferir energia da fonte ao rotor, no qual circula uma corrente elétrica. Logo o comutador serve como uma ligação entre a fonte de tensão e o rotor do motor CC. O rotor quando conduz corrente elétrica dentro de um campo magnético (estator) sofre a ação de uma força perpendicular à bobina e ao campo magnético. O torque do motor, que impulsionará o rotor, é influenciado pela corrente de indução e o campo magnético entre os imãs.

A comutação para este tipo de motor geralmente é feita por escovas e a variação da velocidade da rotação do motor pode ser controlada apenas variando a tensão de alimentação, diferentemente dos motores de corrente alternada, onde a velocidade é controlada pela frequência da alimentação e números de polos do motor.

Uma limitação deste tipo de motor são as baixas potências, motores CC com alta potência possuem um custo relativamente superior se comparados aos motores CA (corrente alternada). (HART, 2011).

Os motores conectados ao eixo da roda possuem alguns transmissores de velocidade acoplados para aumentar o torque e conseguir vencer obstáculos mais complexos, como rampas ou terrenos escorregadios. Esse aumento de torque em contrapartida reduz o valor de velocidade do movimento do robô. Na Figura 18 é representado o motor CC utilizado já acoplado com o redutor e a roda do robô.

Figura 18 – Conjunto motor/roda utilizados no protótipo

Fonte: Do autor, 2019

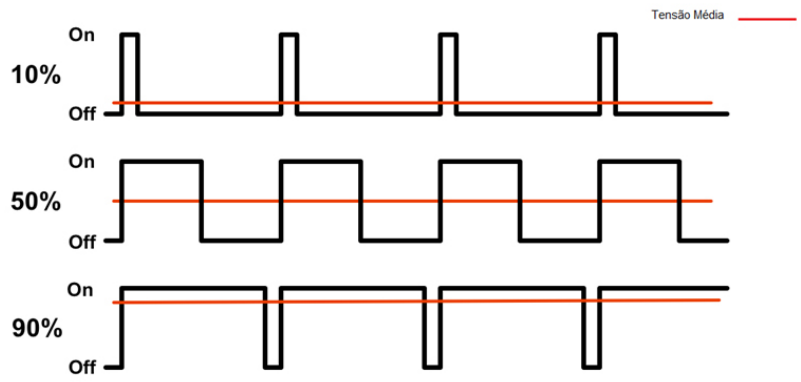
Esse transmissor é muito usado em motores com alta velocidade e pouco torque, e no caso do projeto foi usado uma redução de 48:1, ou seja, o torque é aumentado 48 vezes e a velocidade cai inversamente proporcional.

Além disso o consumo máximo de corrente desse motor é de 250 mA com carga e uma rotação de até 140 rpm sem carga.

O controle da velocidade foi realizado por um sinal de PWM (*Pulse Width Modulation*), ou modulação por largura de pulso, enviado do Raspberry Pi, assim, através da largura de pulso de uma onda quadrada foi possível controlar a potência elétrica da saída, traduzindo na velocidade angular dos motores. Esta técnica é bastante utilizada para simular sinais analógicos de tensão, mesmo o PWM sendo um sinal totalmente digital, pois em qualquer dado tempo a alimentação ou está ligada (nível lógico 1) ou está desligada (nível lógico 0).

A tensão média do sinal varia de acordo com o tempo de ativação e tempo de desativação. Essa relação entre períodos é denominada de ciclo de trabalho (*dutycycle*). Nela quanto maior o ciclo de trabalho maior será a tensão média de saída. (HART, 2011). A Figura 19 apresenta três valores de PWM distintos. O primeiro valor é um sinal com 10% de ciclo em nível alto e 90% em nível baixo, o que representa uma saída de 10% da tensão nominal do sinal. O segundo valor apresenta um ciclo de 50%, o que representa metade do tempo em nível alto e metade em nível baixo totalizando 50% da tensão aplicada. Por último, o maior valor de PWM, um ciclo de 90% em nível alto e por consequência uma tensão média de 90% do sinal de entrada.

Figura 19 – Diferentes ciclos de onda de PWM e tensão média



Fonte: Moraes, 2018

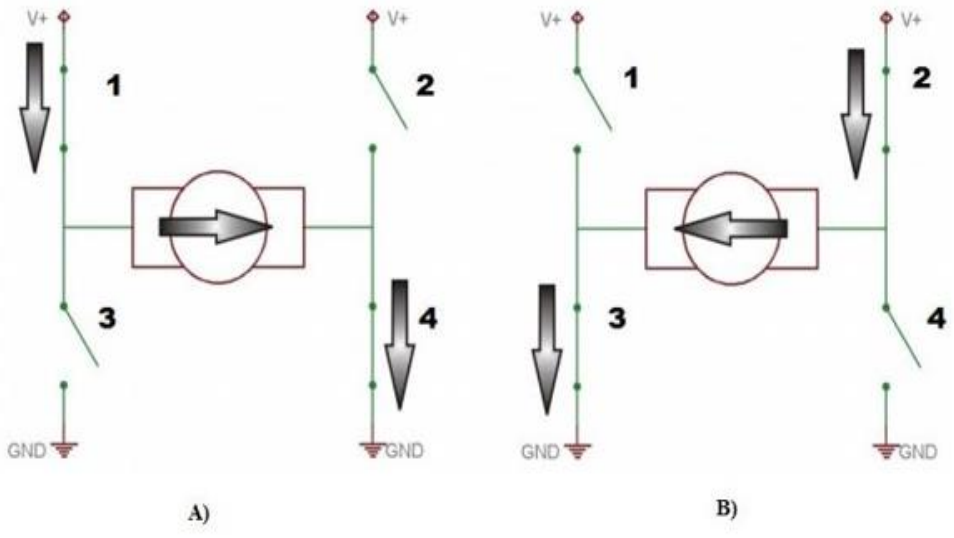
O PWM foi utilizado no projeto para variar a velocidade dos motores já que a tensão média aplicada a cada motor é proporcional à velocidade angular gerada no eixo. Porém esse sinal não foi transmitido diretamente aos motores, ele foi tratado antes em uma ponte H.

3.1.3 Ponte H para motores

Os sinais de comando (PWM) das portas do Raspberry Pi precisam ser conectadas a um circuito (*driver)* que irá acoplar a parte de alta e baixa potência do protótipo. Isso ocorre, pois, os motores drenam até 250 mA (miliAmpères) de corrente e o Raspberry fornece apenas, até uma margem segura, 16 mA nos pinos de entrada e saída (BCM 2835 ARM Peripherals). Portanto, para que o sinal transmitido seja traduzido em velocidade para os motores de forma segura, foi utilizado um driver em ponte H modelo mini L298n.

A ponte H é um circuito de eletrônica de potência que pode controlar o nível de tensão em um motor e a polaridade da tensão e corrente. Em motores CC a polaridade da tensão invertida significa a mudança no sentido de giro, conforme Figura 20. (HART, 2011).

Figura 20 – Ponte H para motores. a) Corrente circulando no sentido direto. b) corrente circulando no sentido reverso.



Fonte: Repositório da automação, 2014

A ponte H possui quatro transistores que atuam como chaves. Quando 1 e 4 estiverem acionados, a corrente circula no sentido direto e a tensão depende do pulso do PWM que é responsável pelo chaveamento dos transistores. Quanto maior o pulso de PWM maior a tensão no motor. Quando os transistores 2 e 3 forem acionados, o sentido da corrente e tensão no motor se inverte mudando o sentido de giro do motor.

Em cada circuito ponte H mini L298n foram conectados dois motores independentes e necessárias quatro entradas de PWM, duas para cada motor. A Tabela 2 apresenta a tabela verdade de acionamento dos motores pelas entradas do circuito. Para um sinal de nível alto na primeira entrada e nível baixo na segunda o motor gira no sentido horário e, caso contrário, o motor gira no sentido anti-horário. Quando as duas entradas estiverem em zero não haverá nenhuma ação e se as duas entradas comutarem para nível alto o motor para instantaneamente.

Tabela 2 – Relação entre sentido de giro e entradas da ponte H

MOTOR	IN1	IN2
HORÁRIO	5v	GND
ANTI-HORÁRIO	GND	5v
PONTO MORTO	GND	GND
FREIO	5v	5v

Fonte : Thomsen, 2013

No protótipo foram necessários dois circuitos pontes H mini l298n para os quatro motores, como citado anteriormente, sendo o consumo dos motores baixo, as especificações do circuito garantem a operação com até 1.500 *mA* de consumo médio por circuito e picos de no máximo 2.500 *mA*.

3.1.4 Sensor Encoder

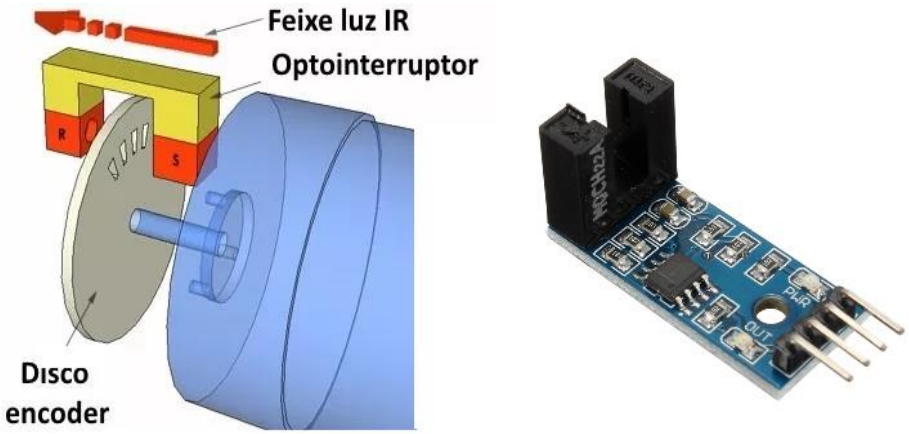
O encoder óptico ou sensor de velocidade, foi adicionado ao robô para medir as distâncias percorridas e velocidade linear necessárias para navegação do protótipo. O sensor utilizado foi um encoder óptico MH-series, que possui um circuito

com um foto-transistor atuando como receptor e um LED de luz infravermelha atuando como emissor.

O disco posicionado ao centro do sensor dispõe de diversas marcações que desempenham o papel de bloquear e desbloquear o feixe de infravermelho que chegará ao receptor. Assim, com o giro da roda, o sinal será bloqueado tantas vezes quanto o disco possuir furos e transmitirá uma onda com um sinal quadrado proporcional a essas marcações. Sabendo o número de furos da roda e o comprimento da roda é possível calcular a velocidade e distância incremental percorrida pelo robô.

Na Figura 21 (a) o desenho esquemático do sensor mostra o diodo emissor de infravermelho, a distância entre emissor e receptor onde foi posicionado o disco furado acoplado a roda, e a direção do feixe de luz, sinalizado do diodo emissor para o foto-transistor. Em (b) é observado o modelo do sensor óptico utilizado no protótipo.

Figura 21 – Encoder óptico: (a) esquema de funcionamento (b) Modelo utilizado.



Fonte: Arduino e Cia, 2018

O sensor MH - series possui um pino de VCC (Tensão em corrente contínua), que pode operar de 3,3 à 5V, um pino de terra, um pino de saída digital e um pino de saída analógica. No projeto utilizou-se dois sensores encoders, um na roda direita e outro na esquerda, conectados aos canais 23 e 26 do Raspberry Pi, para calcular com melhor precisão a distância e velocidade percorridas.

3.1.5 Bússola digital

A bússola é um instrumento utilizado para navegação e determinação das direções. A orientação da bússola é baseada nas propriedades magnéticas de alguns materiais e do campo magnético gerado pelo planeta Terra.

A bússola digital utilizada é uma HMC5883l, aplicada para mensurar campos magnéticos de baixa intensidade. Essa bússola possui um circuito interno baseado em uma ponte resistiva compostos com materiais que sofrem o efeito anisotrópico magneto-resisitivo (AMR), que ocorre em materiais paramagnéticos. Quando um campo magnético é aplicado a uma fina camada de material ferroso ocorre uma mudança da impedância na ponte resistiva, e através dessa variação da impedância é possível mensurar diferentes níveis de campo. Este tipo de bússola é também chamado de magneto resistiva. (CARUSO,1998).

Após sensoriar o sinal do campo magnético, a bússola processa a informação dos eixos x, y ,z e envia os dados, contendo a posição real da bússola em relação ao norte terrestre. Lembrando que a Terra possui um norte magnético localizado no hemisfério sul e um sul magnético na região norte. Na região do Equador a bússola opera sem desvios, pois a direção do campo é um vetor horizontal, e, à medida que se distancia do Equador, movendo-se em sentido aos pólos geográficos, a bússola apresenta desvios, dependendo do hemisfério localizado. Esses desvios, causados pela inclinação magnética, devem ser considerados nos cálculos para melhorar a precisão da localização.

A bússola com 3 eixos magnéticos possui elementos sensores nos três eixos ortogonais, o vetor horizontal, vertical, e a gravidade terrestre. Para equilibrar a bússola os dois sensores magnéticos são complementados a esse elemento de medição gravitacional. Logo esse elemento que mensura a força gravitacional pode também ser usado para balancear e equilibrar os eixos, aumentando assim, a precisão da bússola para diferentes posições.

A princípio a bússola foi fixada junto ao chassi do robô, porém, foi observado que os motores geram um forte campo magnético e assim influenciam nos valores medidos e não garantem confiabilidade. O nível da interferência comprometeu a leitura do sensor chegando ao ponto que um giro de 360º do protótipo variava apenas 100º os valores da bússola. A solução encontrada pode ser vista na Figura 22.

Figura 22 – Bússola HMC 5883 acoplada a haste de madeira

Fonte: Do autor, 2019

Para resolver o problema, foi necessário afastar a bússola do protótipo usando uma espécie de “antena”, uma haste de madeira, fixada na parte traseira do chassi, com altura de 25 cm da base, segura da influência magnética. Os valores que antes variavam de 210 a 310º após o implemento da antena passaram a ter a escala original projetada variando de 0 a 360º. Para enviar os dados ao dispositivo de controle a bússola utiliza o protocolo I2C (*Inter-Integrated Curcuit*).

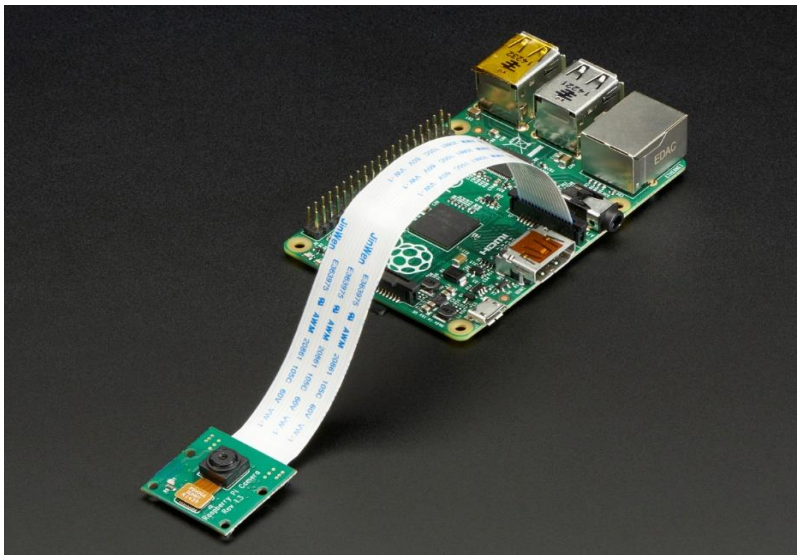
3.1.6 Câmera Raspberry Pi

A Câmera Raspberry Pi Versão 1.3 com 5 megapixels de resolução foi escolhida para compor o projeto da câmera térmica. O dispositivo é conectado diretamente ao plug CSI (*camera serial interface*) do Raspberry. A câmera é compacta e leve, com peso aproximado de 3 g, possui resolução de 2592 X 1944 pixels além de gravar vídeos em até 1080p (1920 X 1080).

A conexão é realizada através de um cabo *flat* de 15 pinos ao terminal CSI do Raspberry projetado exclusivamente para a conexão de câmeras. Esta conexão oferece alta velocidade de comunicação e consegue uma taxa de 30 fps (frames per second) na resolução de 1080p e uma taxa de até 60 fps numa resolução de 720p

(Adafruit). A Figura 23 apresenta a câmera acoplada ao Raspberry Pi através do cabo flat que oferece uma flexibilidade para posicionamento no protótipo.

Figura 23 – Camera Raspberry acoplado a placa central



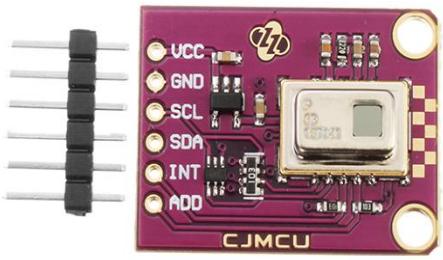
Fonte: Adafruit®

3.1.7 Sensor de temperatura infravermelho AMG 8833

O sensor de infravermelho AMG 8833 possui uma matriz de 8 linhas de 8 pixels totalizando 64 células, sendo que, cada célula conta com uma termopilha de infravermelho que realiza a medição passiva da radiação emitida pelo corpo do objeto, geralmente numa faixa entre 8 e 15 mícrons. Esse número de células de infravermelho é importante para visualizar o gradiente da distribuição de temperatura no espaço, o que não é muito comum para sensores de tão baixo custo. (Hrisko, 2018).

O sensor possui uma precisão de ± 2,5 ºC, com característica de baixo consumo de energia (4,5mA) e boa eficiência. A comunicação com o Raspberry Pi é realizada através do canal I2C, garantindo uma rápida taxa de transferência de dados e um diagnóstico de falhas simples e eficaz. Na figura 24, o canal I2C é representado pelos pinos SDA (*Serial Data*) e SCL (*Serial Clock*), o pino VCC é a alimentação positiva de 3,3V e o GND é o terra do sensor. (PANASONIC, 2019).

Figura 24 – Sensor térmico AMG 8833



Fonte: Do autor, 2018

O sensor mede temperatura a distâncias de até 7 metros. A taxa de aquisição do sensor é de 10 Hz, satisfatória para transientes de temperatura, porém, ao desenvolver a câmera térmica, a taxa utilizada foi consideravelmente menor, devido ao tempo perdido no processamento de alguns dados dentro do Python. (PANASONIC, 2019).

O princípio de funcionamento desse sensor é baseado na medição da radiação de calor emitida por um corpo cinza. Essa temperatura é calcula pela lei de Stefan-Boltzmann representada na Equação 7. (PANASONIC, 2019).

$$\Pi = \varepsilon A \sigma T^4 \tag{7}$$

Onde ε é a emissividade (valor entre 0 e 1), σ é a constante de Boltzmann (5,67 x 10^{-8} $\frac{w}{m^2 K^4}$), A é a área da superfície , T é a temperatura do corpo e Π é a potência emitida. Essa lei estabelece que a energia total radiada por unidade de área superficial de um corpo negro na unidade de tempo (fluxo radiante), é diretamente proporcional à quarta potência da sua temperatura termodinâmica T. Em um corpo negro a emissividade é considerada igual a 1 , em corpo cinza varia de 0 a 1. Como o sensor mede a potência da irradiação, a Equação 8 é adicionada para mensurar a temperatura do corpo:

$$T_{corpo=} \left(\frac{V}{k} + T_s^4\right)^{1/4} \tag{8}$$

Na Equação 8 a letra V representa a tensão medida pelo sensor, k representa uma variável que engloba a constante σ, ε e A da equação de Boltzmann. A variável

T_S indica o valor de temperatura do sensor, e T_{corpo} a temperatura do corpo a ser mensurado. A presença da variável T_S na equação se dá pela sua influência na medição da temperatura do corpo.

O código responsável por desenhar o mapa de calor gerado pelo sensor foi escrito e compilado em Python 3. Os mapas de calor são a melhor representação nesta aplicação pois conseguem representar a distribuição espacial das fontes de calor. O mapa de temperatura é representado por colorações avermelhadas onde o calor é mais intenso e coloração mais azulada onde a temperatura é menor. A interpolação da câmera do Raspberry com o sensor AMG 8833 de 64 células gerará a imagem térmica utilizada para classificar o ambiente.

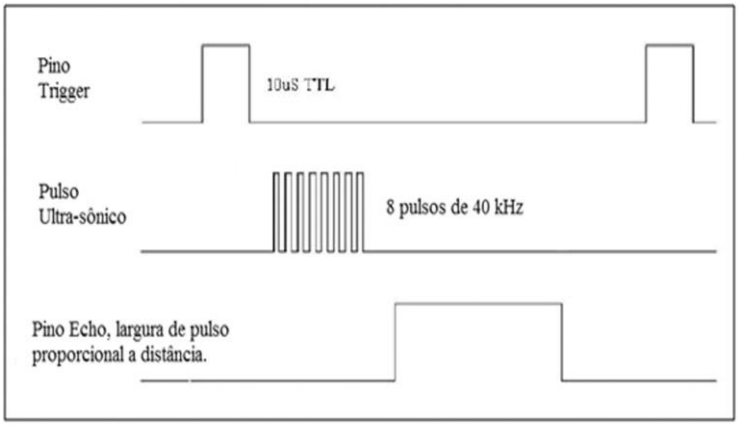
3.1.8 Sensores Ultrassônicos HC-SR04

Os sensores ultrassônicos são compostos basicamente por dois componentes: o emissor e o receptor. O sinal sonoro é emitido e, ao colidir com um obstáculo, dentro dos limites do sensor, retorna, cronometrando o tempo de ida e volta. Sabendo a velocidade do som no ar, que é aproximadamente 343 m/s, fica simples calcular a distância que o sensor está do objeto. Caso o objeto esteja fora do alcance dos limites do sensor provavelmente ele mostrará um valor fora de escala.

Estes tipos de sensores trabalham com frequências a partir de 40KHz. Essas frequências estão fora da faixa de audição humana que varia de 20Hz a 20KHz. No projeto foi utilizado um sensor ultrassônico muito comum que é usado em conjunto com o Raspberry Pi e também Arduino, o HC-SR04.

O HC-SR04 trabalha enviando um pulso de gatilho (trigger) com duração de dez microssegundos e, logo após, o transmissor envia 8 pulsos ultrassônicos na frequência de 40kHz que o seu receptor receberá em forma de eco. Após isto, o pino ECHO irá a nível alto por um tempo proporcional à distância a que o sensor se encontra do obstáculo. A Figura 25 mostra o processo de funcionamento e disparo dos pinos TRIGGER e ECHO.

Figura 25 – Relação entre os pulsos nos pinos TRIGGER E ECHO



Fonte: Dworakowski et al, 2016

A partir da largura pulso de saída é possível calcular a distância, pois esse tempo do pino ECHO é exatamente o tempo de duração entre emissão e recepção do sinal, sendo assim:

$$Distância = \frac{Pulso\ nível\ alto\ ECHO(\mu s) * Velocidade\ do\ som\ \ no\ ar}{2} \qquad (9)$$

Onde a velocidade do som no ar é 343 m/s, a duração do pulso em nível alto é dada em segundos e proporcional à distância ao objeto, e a divisão por 2 é necessária pois só será computada metade da distância total de propagação. A Tabela 3 traz algumas características do sensor como tensão de operação, distância de medição, e o baixo consumo de corrente do sensor.

Tabela 3 – Características do sensor HC-SR04

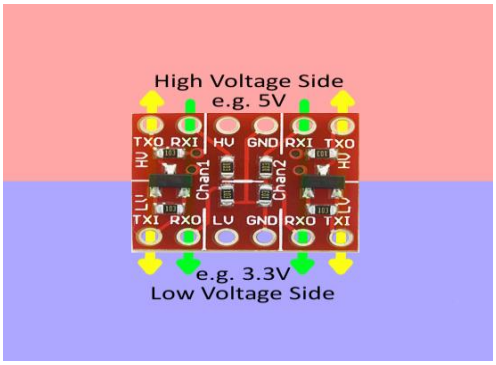
Especificação	Descrição
Tensão de operação	5V
Corrente Quiescente	2mA
Corrente em funcionamento	15mA
Ângulo de medida	15°
Distância de medição	2 a 400 cm
Resolução	3mm
Frequência ultrassônica	40kHz

Fonte: Adaptado de Sparkfun®

Um dos problemas encontrados na montagem do circuito foi a tensão de operação do sensor ultrassônico operar em uma faixa diferente dos pinos de entrada/saída do Raspberry Pi, sendo que o primeiro opera com 5V e o segundo com 3,3V. O sinal em questão é a largura de pulso emitida do pino ECHO do sensor HCSR04 para o Raspberry Pi.

Desta maneira, se o sensor enviar sinal de 5V direto em um pino da placa pode danificar o circuito. A solução encontrada foi utilizar um conversor de nível lógico bidirecional para diminuir a tensão de 5V para 3,3V e transmitir o novo sinal ao Raspberry Pi. Por ser bidirecional, este conversor pode tanto operar como conversor *step-down* ou como conversor *step-up*. Nesse caso ele funcionará como *step down* pois diminuirá a tensão. O circuito possui 2 canais de 4 pinos que podem ser utilizados independentes. A Figura 26 traz um esquema de fluxo do sinal do conversor utilizado.

Figura 26 – Circuito de um conversor bidirecional de tensão

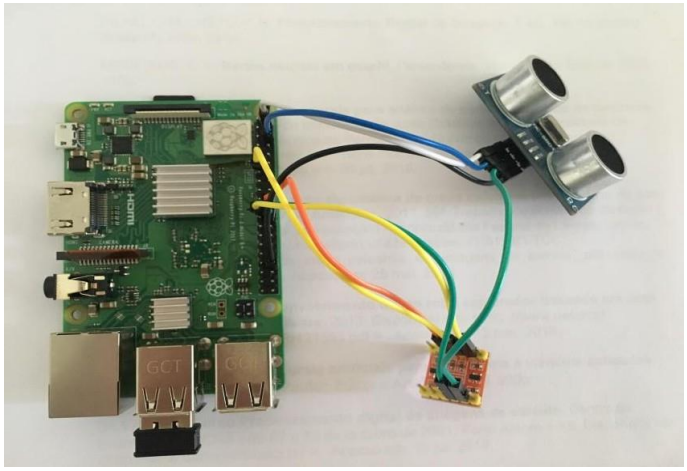


Fonte: Saravana Electronics

Os pinos denotados por HV (*High Voltage*), são os pinos alimentado por 5V, os pinos LV (*Low Voltage*) são alimentados por 3,3V e GND (*Ground*) foi ligado ao terra. Os pinos TX são bidirecionais, ou seja, funcionam tanto como amplificador como abaixador de tensão. Já os pinos RX são unidirecionais, ou seja, só operam no sentido de abaixar a tensão de 3,3V para 5V.

A conexão entre o Raspberry, o conversor de nível lógico e o sensor ultrassônico é mostrado na Figura 27. No protótipo foram utilizados dois conversores de nível lógico, sendo um deles responsável pela conversão dos sinais de dois sensores ultrassônicos e o outro pela conversão dos demais.

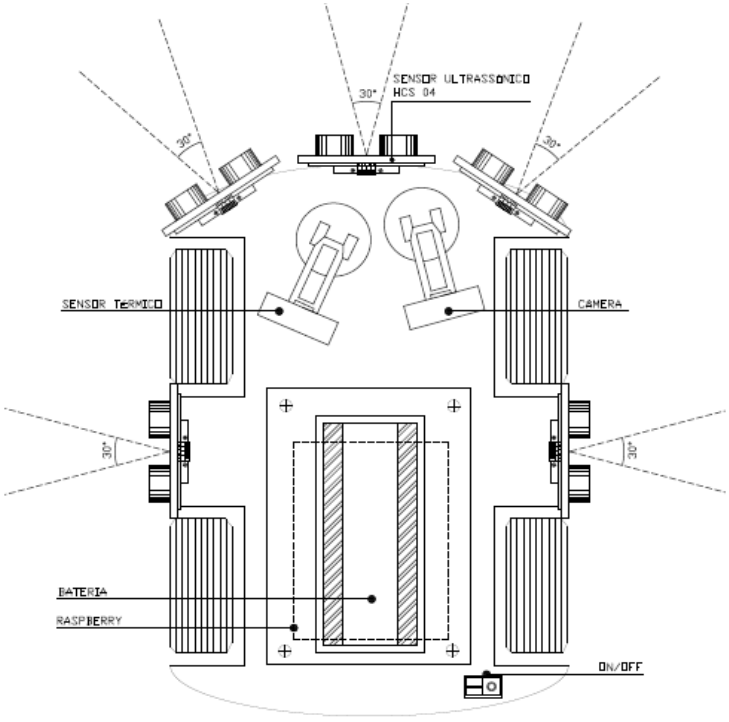
Figura 27 – Conexão entre Raspberry e sensor



Fonte: Do autor, 2018

Para o treinamento da rede neural foram utilizados 5 sensores ultrassônicos em posições estratégicas no protótipo. A Tabela 3 indica uma boa resposta do sensor HC-SR04 para ângulos de até 15º de abertura a esquerda e 15º a direita totalizando 30º. A Figura 28 apresenta o chassi do protótipo contendo os sensores ultrassônicos e os ângulos de abertura do feixe de cada sensor.

Figura 28 – Protótipo com sensores ultrassônicos e ângulo de visão



Fonte: Do autor, 2019

No trabalho de Dworakowski et al (2016), foi observado a eficiência do sensor em ângulos de medida oblíquos e foi observado maior precisão em objetos situados num ângulo de visão de até 30º. Sendo assim, como foi visto na Figura 28, foi posicionado um sensor na posição central do chassi, um 30º a direita e outro 30º a esquerda. Também foram utilizados dois sensores laterais, um do lado esquerdo e um do lado direito para o algoritmo de navegação. Assim o protótipo conseguirá “enxergar” obstáculos à frente, retos ou oblíquos, além de obstáculos laterais.

3.1.9 Desenhos em impressora 3d

A impressão por deposição de material termoplástico ou fdm (*fused depositing moddeling*) é uma técnica de fabricação do objeto camada por camada, a partir de um material termoplástico, chamado de filamento, que possui características mecânicas distintas uns dos outros. A impressora possui um bico por onde é injetado o filamento, este é aquecido até sua temperatura de fusão, onde é depositado em camadas em uma mesa aquecida formando a geometria do objeto, pré-definida em um programa tipo CAD (*Computer Aided Design*). (SHIMANO et al., 2018).

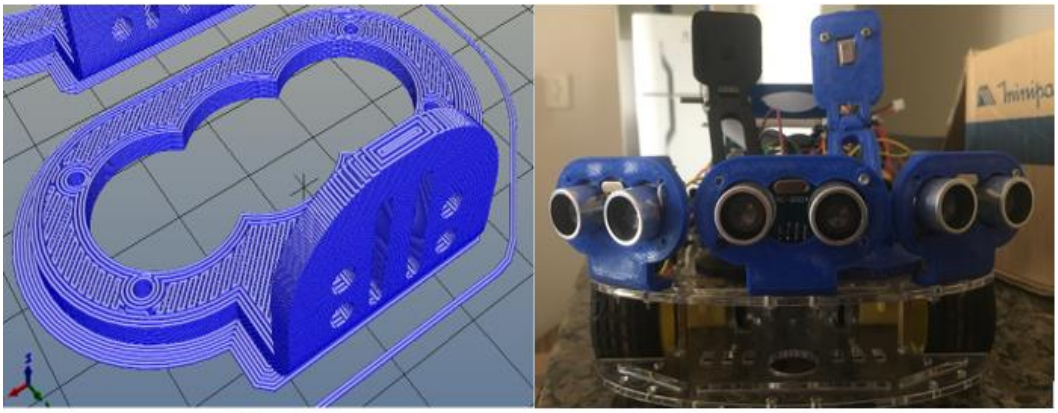
A impressão por extrusão tem características de baixo custo comparados a outros tipos de impressão 3D, e são eficientes para objetos que não sofreram esforços mecânicos significativos. Para o projeto do protótipo foram adaptados alguns desenhos da plataforma Thingverse, sendo um deles os suportes dos sensores ultrassônicos, já o suporte do sensor AMG8833 foi inteiramente projetado para o protótipo utilizando o *software* de desenho SketchUp.

A impressão foi executada em uma impressora fechada da marca GTMax® 3D - modelo core A1. Esta impressora possui autonivelamento da mesa e gabinete totalmente fechado, o que possibilita a impressão com materiais que sofrem contração, como o ABS (Acrilonitrila Butadieno Estireno).

O ABS foi utilizado como filamento, pois possui um menor custo e com características mecânicas compatíveis com a aplicação e esforços nas peças. O preenchimento utilizado foi de 30%, o que já possibilita uma boa resistência mecânica para várias aplicações. O software empregado na configuração da impressora foi o REPETIER – HOST, responsável pelo ajuste dos parâmetros e controle das ações da impressora. Para gerar o código G (*G-code*), que é a

sequência de movimentos necessários para imprimir o objeto, utilizou-se a extensão Slic3r na etapa conhecida por fatiamento. Dentro dessa extensão são inseridas as informações de porcentagem de preenchimento, temperatura de fusão do material, altura da camada, entre outros. A Figura 29 (a) apresenta o desenho fatiado do suporte dos sensores ultrassônicos, já (b) destaca o suporte já inserido no protótipo encaixado nos sensores ultrassônicos. A figura 30 apresenta o sensor AMG 8833 impresso (a) e em desenho (b).

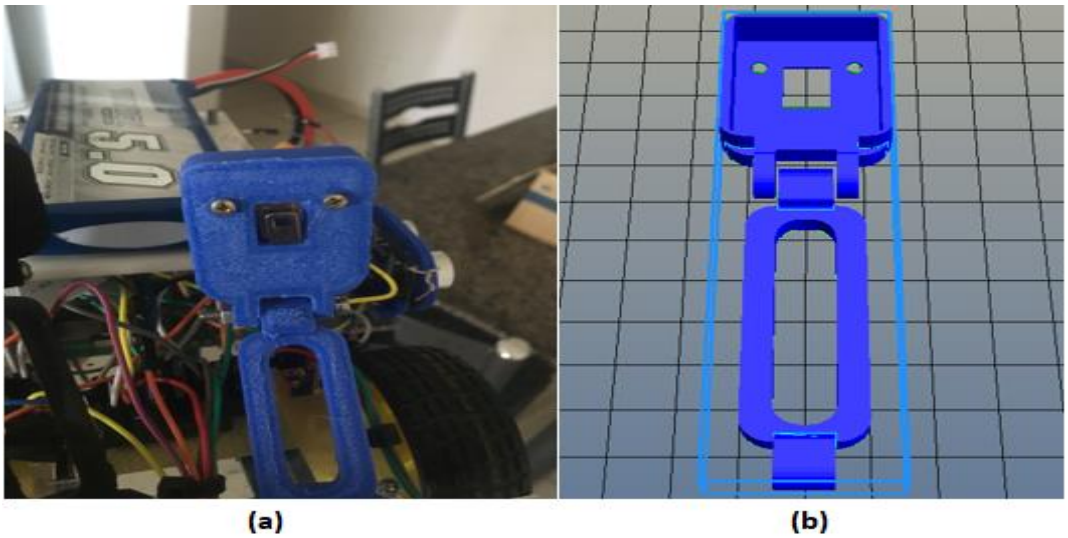
Figura 29 – Suporte para sensor ultrassônico: (a) Desenho fatiado visto pelo software REPETIER-HOST (b) Montado no chassi do protótipo dando sustentação aos sensores.



(a) (b)

Fonte : Adaptado do site Thingverse, 2019

Figura 30 – Suporte do sensor AMG 8833 : (a) montado no protótipo e (b) Desenho aberto no software REPETIER-HOST.



(a) (b)

Fonte: Do autor, 2019

O suporte para o sensor AMG 8833 é representado na Figura 30 (a) com o modelo já impresso e acoplado ao sensor no próprio protótipo, já em (b) observa-se o desenho no ambiente do REPETIER-HOST antes de ser fatiado.

Os suportes depois de finalizados foram fixados no protótipo e utilizados para sustentação dos sensores. Os sensores foram fixados através de parafusos de 1,5 mm de diâmetro por 10 mm de largura. No suporte do sensor térmico, especificamente, foram utilizados parafusos de 3 mm de diâmetro por 30 mm de largura nas junções das partes móveis do suporte. A impressão do suporte articulado por parafusos foi de grande ajuda para ajustes na altura e direção do sensor térmico. O suporte da câmera Pi foi impresso também em impressora 3D, em material ABS, porém o desenho utilizado foi extraído da plataforma Thingverse e não foram realizadas grandes modificações, pois as medidas eram idênticas a câmera utilizada no protótipo.

3.1.10 Sistema de alimentação

3.1.10.1 Bateria

O Raspberry Pi 3 B+ necessita, para o correto funcionamento, um suprimento de 5V, comum a todos os modelos da marca, e fonte de corrente de 2,5 A, recomendado pelo fabricante. (RASPBERRY PI FOUNDATION, [201-]). Logo a fonte de tensão foi uma bateria de 5 V e 2,5Ah ou corrente superior.

Ademais, uma outra parte do robô necessita de alimentação externa: o driver da ponte H, que alimenta e controla os motores CC. A alimentação dos motores varia de 3 a 6 V, em 6V observa-se o maior torque, com consumo máximo de 200 mA por canal. Sendo assim a bateria que atende as especificações deve fornecer uma tensão na faixa de 3 a 6V e uma corrente mínima de 1 A.

Com base nas informações prévias de consumo mínimo de tensão e corrente, tanto para o Raspberry quanto para os motores, utilizou-se uma bateria de Lítio-Polímero (LiPo) de 7,4 V, duas células com corrente de 5000 mAh. Essa bateria ficou responsável por alimentar todos os dispositivos presentes no protótipo.

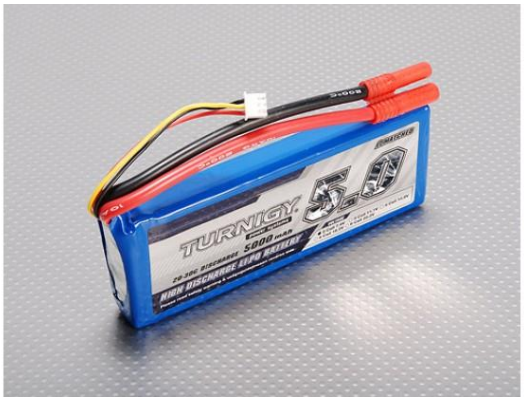
As baterias do tipo LiPo são recarregáveis e baseadas na tecnologia de íon-Lítio, que usa um eletrólito de polímero ao invés de um eletrólito líquido. Essa

tecnologia permite criar baterias com estruturas variadas, com peso leve e com características de alta descarga.

Cada célula de uma bateria LiPo possui 3,7V, e são denominadas pela letra S. A corrente associada quantifica a carga que pode ser fornecida da bateria ao circuito alimentado. A capacidade de carga e descarga das baterias é dada pela letra C, ou seja, caso apareça o valor de 1C, equivale a dizer que o pico de descarga é uma vez o valor de sua corrente. Algumas baterias possuem capacidade de descarga que varia até 100C.

A capacidade de descarga da bateria utilizada é 20C, isso representa uma descarga de pico até 20 vezes maior que a corrente nominal da bateria, se a bateria do protótipo possui corrente de 5000mAh o pico da capacidade de descarga é 20*5000 = 100000mA. A Figura 31 apresenta a bateria de LiPo utilizada no protótipo.

Figura 31 – Bateria do tipo LiPo de 2 células(7,4V) e 5000mAh



Fonte: Do autor, 2018

Caso a bateria seja forçada a fornecer uma carga além dos limites, pode ocorrer um “inchaço”, formado pelo desprendimento de gases no processo químico da conversão de energia. Essa deformação da bateria danifica sua estrutura permanentemente.

Além disso, no processo de carregamento das baterias, deve-se tomar o cuidado de balancear as células, ou seja, certificar que as células carreguem por igual, e não sobrecarregue alguma pois o sobre ou subcarregamento pode danificar irreversivelmente a bateria.

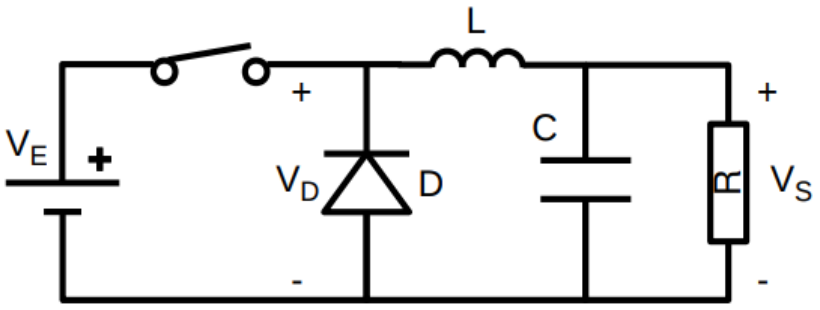
3.1.10.2 Conversor Step-down (Buck)

Previamente foi abordado que o Raspberry Pi necessita de 5V para seu correto funcionamento, porém a bateria utilizada fornece 7,4 V, uma tensão maior do que o necessário. Os motores operam com 6 V, tensão também menor que a fornecida pela bateria.

Para solucionar este problema utilizou-se um circuito capaz de reduzir os valores de tensão contínua. Esse circuito é conhecido como conversor *Buck* ou *Step Down.* O conversor *Buck* é uma fonte chaveada que reduz o valor de tensão de entrada, trabalhando com corrente contínua e normalmente possui dois elementos semicondutores e pelo menos um elemento para armazenar energia, como um capacitor ou indutor. Ele também conta com filtros capacitivos na saída do sinal para diminuir o fator de *ripple*. (POMILIO, 2013)

Este tipo de dispositivo é muito eficiente, tornando-o mais viável do que reguladores de tensão linear que reduzem as tensões por divisores e dissipam o calor. A Figura 32 é um circuito *buck* simples chaveado, no qual o chaveamento normalmente é feito por transistores ou mosfets, que exercem o papel de aumentar ou diminuir a tensão média fornecida a carga. O diodo D e o indutor L são responsáveis por diminuir a oscilação da corrente de saída e de concatenar a variação da tensão média com o ciclo de trabalho.

Figura 32 – Circuito conversor Buck



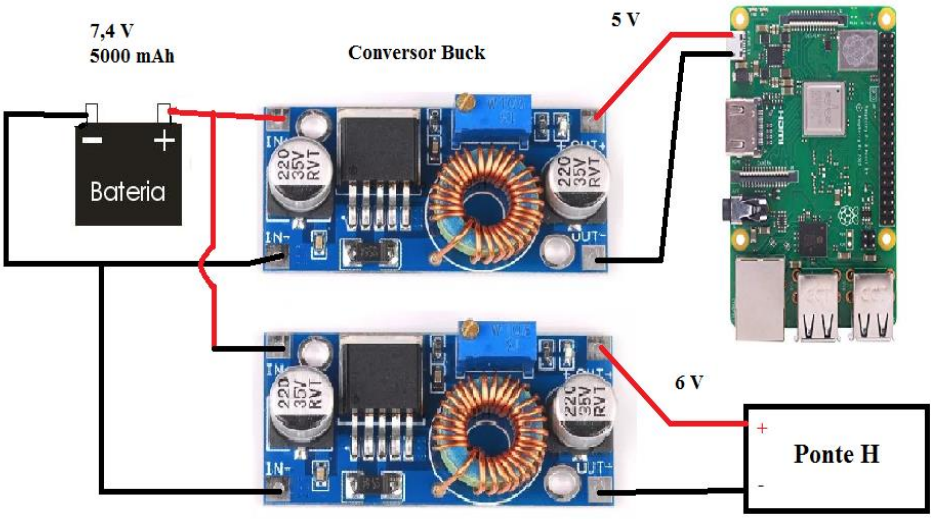
Fonte: Liberti

O conversor utilizado foi um *Buck* modelo Xl 4015, que comporta uma tensão de entrada variável de 7 a 36 volts e uma tensão de saída de 1,25 a 32V, suporta uma corrente de saída de até 5A, o que é suficiente visto que o consumo máximo do

Raspberry é 2,5A e dos motores são 1A (250 mA por motor), além disso dispõe de proteção contra curto circuito. As características desse módulo vieram ao encontro das necessidades quanto a alimentação dos motores e Raspberry Pi.

O sistema ficou composto pela bateria LiPo fornecendo 7,4 V e um circuito conversor *Buck* na sua saída abaixando a tensão para 5 V, para alimentar o Raspberry e um outro conversor para alimentar os motores com 6 V como pode ser visto na Figura 33.

Figura 33 – Esquema de alimentação do circuito.

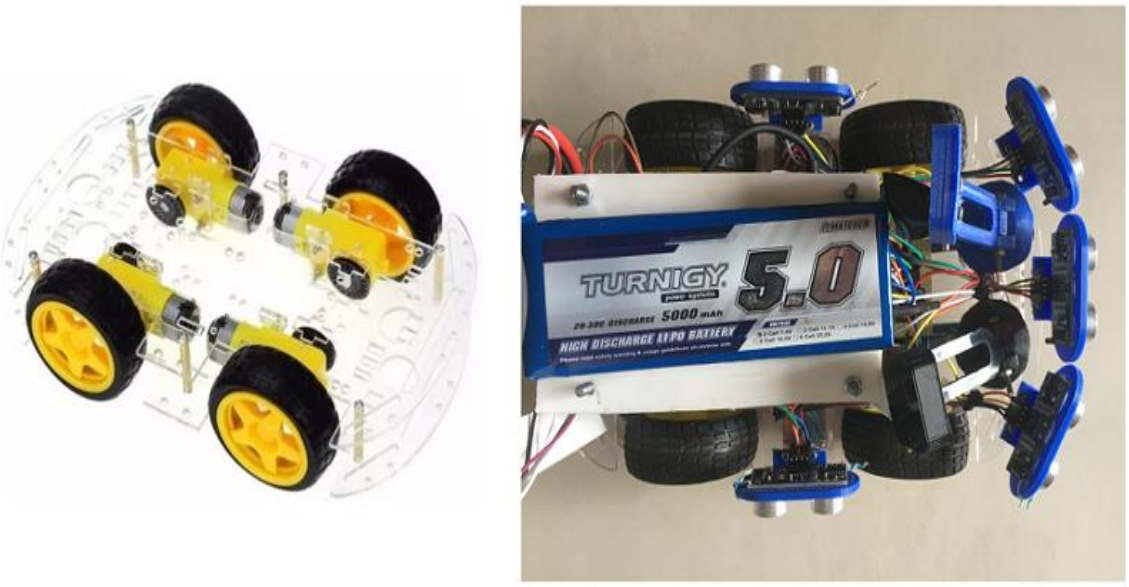


Fonte: Do autor, 2018

3.1.11 Chassi

O protótipo foi montado com base em um chassi de acrílico e foi adquirido juntamente com os motores e rodas. Trabalhando na proposta de redução de custos para montagem do protótipo foram utilizados motores e sensores facilmente encontrados no mercado e que cumpriam as exigências necessárias para execução das funções. O chassi possui 6 parafusos de sustentação e diversos furos e aberturas para fixação e passagem dos fios. A Figura 34 (a) mostra apenas o do robô com os motores e rodas, sem sensores e Raspberry. A Figura 34 (b) apresenta o chassi finalizado pronto para testes.

Figura 34 – Protótipo: (a) Chassi livre de componentes (b) Montagem final



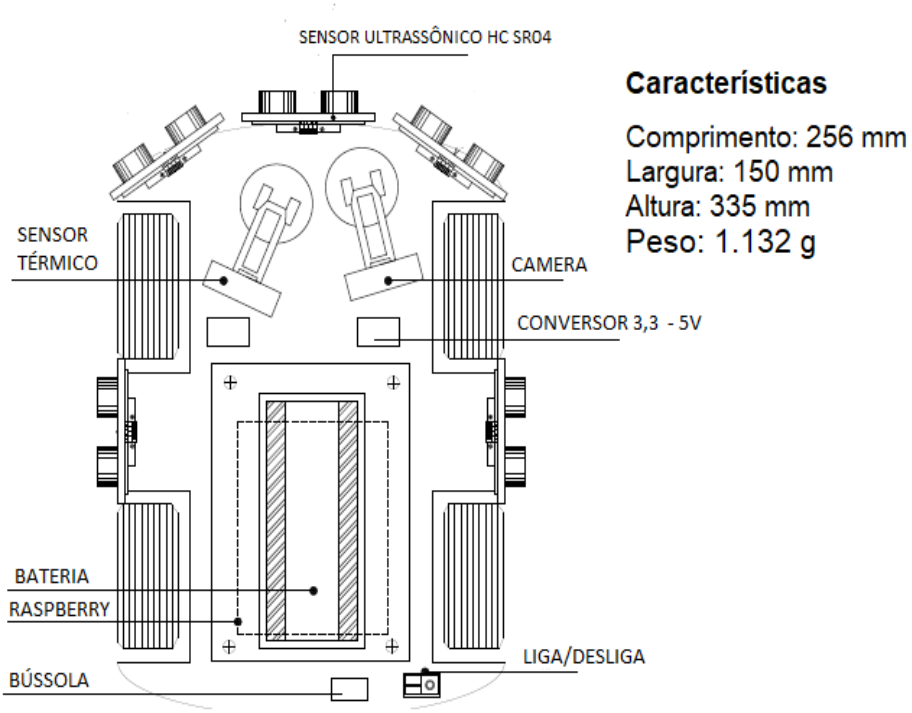
(a) (b)

Fonte: Do autor, 2019

A partir da estrutura presente na Figura 34 (a) foram adicionados elementos na parte inferior e superior até a montagem final do protótipo. Na primeira camada de acrílico foram fixados os circuitos conversores Buck e os dois circuitos L298n, que ocuparam um espaço pequeno comparado aos componentes da segunda camada, porém o espaço ficou bastante limitado pela ocupação dos 4 motores elétricos. Na porção central da segunda camada de acrílico foi posicionado o Raspberry Pi, de onde foram derivadas as ligações aos demais equipamentos. Na parte frontal foram fixados a câmera Pi, o sensor térmico AMG 8833 , três sensores ultrassônicos e dois conversores lógicos de tensão.

Na parte lateral direita e esquerda foram posicionados o sensor ultrassônico em cada lado. Na parte de trás foi apenas fixado o comutador de acionamento do robô e a haste com a bússola digital. Para fixar a bateria foi necessário construir uma terceira camada, pois suas dimensões não possibilitaram encaixá-la no quadro original, desse modo foi colocado uma pequena plataforma na região central da segunda camada de acrílico e colada a bateria. A Figura 35 traz um desenho estrutural da vista superior do protótipo, trazendo também características importantes como medidas e peso.

Figura 35 – Visão superior do protótipo em estágio final de montagem



Fonte: Do autor, 2019

3.2 *SOFTWARE* E ALGORITMOS

Nesta parte do trabalho foram introduzidos as bibliotecas e algoritmos necessários para atingir os objetivos do projeto. Foi escolhido como padrão, a linguagem Python 3, de alto nível e compatível com diversas bibliotecas utilizadas pelo Raspberry Pi. Para cada aplicação específica foi utilizada uma biblioteca compatível auxiliando no desenvolvimento do algoritmo como, por exemplo, o uso do OpenCV para processamento de imagens, e o Keras e TensorFlow para montar a arquitetura das redes neurais artificiais.

Além do Python ainda foi utilizado a sintaxe HTML e linguagem JavaScript para a aplicação Web. Nos próximos tópicos cada linguagem e biblioteca serão descritas individualmente.

Outro ponto importante a ser observado é o sistema operacional próprio do Raspberry: o Raspbian, Esse sistema já possui softwares desenvolvedores para programar em Python, C, C++, além de possuir mais de 35.000 bibliotecas (RASPBERRY PI FOUNDATION, [201-]). Assim o acesso e programação do

Raspberry se tornam mais interativa, facilitando a programação do operador. No protótipo está sendo usado a versão *Stretch*.

3.2.1 Linguagem Python

O Python 3 foi a principal linguagem utilizada para programação tanto do algoritmo de navegação quanto do processamento das imagens dentro do Raspberry Pi. O Python é uma linguagem orientada ao objeto, funcional, de alto nível e tipagem forte. É uma linguagem de protocolo aberto e gerenciado pela Python Software Foundation. Foi criada em 1991 por Guido Van Rossum (ALVES, 2013).

Nesta linguagem o esforço do programador é menor que o esforço computacional. A linguagem prioriza a estrutura e entendimento do código sobre velocidade computacional e processamento. A composição de suas bibliotecas e sintaxes são claras e concisas.

O Python está difundido no cenário mundial e já é usado em algumas empresas como o site Youtube, o buscador Google, Yahoo e até a agência NASA. O sistema de gerenciamento de reservas da Air Canada também usa Python em alguns de seus componentes.

A versão recente que foi utilizada no trabalho, o Python 3, lançada em 2009, alterou algumas funções de programação em relação ao Python 2 mas conservou a estrutura e facilidade no uso. O Raspberry possui um IDE (*Integrated Development Environment*) gratuito para desenvolvimento de códigos, o IDLE.

A Figura 36 mostra um *script* de aquisição do sinal do sensor ultrassônico usando Python. A partir da duração do pulso em nível alto do pino ECHO (termpototal) foi possível calcular a distância do objeto utilizando a Equação 9. O código para leitura dos sensores ultrassônicos foi criado em um arquivo auxiliar para que o código principal ficasse mais enxuto. De maneira semelhante foi feito com o sensor encoder óptico, onde os cálculos da distância pelos pulsos foram executados em um código salvo em um arquivo auxiliar e chamado por funções específicas no código principal.

Figura 36 – Algoritmo utilizado para aquisição de sinal dos sensores HC SR04

```
import RPi.GPIO as gpio
import time


def distance1():

    gpio.setmode(gpio.BCM)
    gpio_TRIGGER1 = 10
    gpio_ECHO1 = 9
    gpio.setup(gpio_TRIGGER1, gpio.OUT)
    gpio.setup(gpio_ECHO1, gpio.IN)

    #Colocar Trigger en níver alto
    gpio.output(gpio_TRIGGER1, True)

    # Setar Trigger após 0.01ms para 0
    time.sleep(0.00001)
    gpio.output(gpio_TRIGGER1, False)

    StartTime = time.time()
    StopTime = time.time()

    # Salvar o tempo inicial
    while gpio.input(gpio_ECHO1) == 0:
        StartTime = time.time()

    # Salvar o tempo de chegada do pulso
    while gpio.input(gpio_ECHO1) == 1:
        StopTime = time.time()

    # Calcular a diferença de tempos
    Tempototal = StopTime - StartTime
    # Multiplicar pela velocidade do som (34300 cm/s)
    # Dividir por 2
    distance1 = (Tempototal * 34300) / 2
    gpio.cleanup()
    return distance1
```

Fonte : Do autor utilizando Python, 2019

3.2.2 Aplicação Web

3.2.2.1 Sintaxe HTML

O HTML tem por função a criação da estrutura necessária para organizar as páginas web ou aplicativos. Essa estrutura inclui cabeçalho, parágrafos, links para outras páginas e rodapé. O HTML é caracterizado por ser um conjunto de rótulos (*tags)* que irão demarcar o conteúdo do interior da página a ser criada. Cada rótulo possui uma funcionalidade específica e normalmente aparece no formato “<rótulo>”, como por exemplo “<title>” que irá especificar o título da página ou seção. O nome da *tag* vem acompanhada dos caracteres < e >. Caso encontre a estrutura “<rótulo/>”, significa que a função em questão está sendo fechada. (MAZZA, 2014)

Tim Berners-Lee criou a sintaxe HTML com o intuito de disseminar as pesquisas feitas por ele e seus amigos. Sua criação veio ao encontro da expansão da internet pública em todo o mundo. Atualmente ela é padronizada pela W3C (World Wide Web Consortium), uma organização mundial para padrões da internet.

A organização estrutural de uma página HTML pode ser observada na Figura 37. Essa figura representa a página inicial do aplicativo *web* desenvolvido. Por ser padronizado, a estrutura pode ser processada pela maioria dos navegadores conhecidos.

Figura 37 – Exemplo da estrutura de uma página html

```
<!DOCTYPE HTML>
<html lang="pt-br">
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

    <meta name="viewport" content = "height = device-height,width=device-width ,initial-scale=1"

    <title>Robot Menu</title>

    <link rel="stylesheet" type="text/css"  href="estilo.css" />
    <link rel="stylesheet" href="source.css">

<style type="text/css">
body{

    background: url(fundo_2.jpg) no-repeat center top fixed;

    -webkit-background-size: cover;
    -moz-background-size: cover;
    -o-background-size: cover;
    background-size: cover;
    font-family: arial, helvetica, sans-serif;
    font-size: 18px;
    }

.cloud-sidebar {
                background: white;
                font-family: monospace

              }

</style>
</head>
<body>
<div id="principal">
    <div id="cabecario">
        <nav class="w3-bar w3-black">
            <a href="index.html" class="w3-button w3-bar-item">Home</a>
            <a href="Modo_Manual.html" class="w3-button w3-bar-item">Modo Manual</a>
            <a href="Modo_Automatico.html" class="w3-button w3-bar-item">Modo Automático</a>
            <a href="contato.html" class="w3-button w3-bar-item"> Contato</a>
```

Fonte: Do autor utilizando HTML, 2019

Na primeira linha da programação observa-se a instrução DOCTYPE, que basicamente avisará para os navegadores qual o tipo de código que eles receberão e terão que decodificar. Nesse caso o browser irá receber um documento do tipo HTML, mas poderia ser, por exemplo, um código CSS (*Cascading Style Sheets*) .

Após essa linha de comando, o código receberá o rótulo “html”. O atributo Lang, posicionado de forma subsequente ao HTML, indica o idioma do documento.

Após a abertura da *tag*, até seu fechamento o documento será estruturado em HTML.

O rótulo *"*head" define o que será incluído no cabeçalho. No cabeçalho é definido o título do documento, e o conteúdo dentro dessa região não é visível para o usuário, mas é definido o comportamento da página e funções. Nessa parte pode ser definido estilos de escrita, botões, parágrafos e também podem ser inseridos *scripts*, atuando como funções. (MAZZA, 2014)

A *tag* "*Title"* define o nome do título do documento. Já "Meta" define o tipo de carácter que será usado como fonte para o texto da página. O UTF-8 possui caracteres Unicode e ASCII (*American Standard Code for Information Interchange*), o mais comumente usado.

O conteúdo visual da página, como o texto, botões, imagens, gráficos, entre outros, vem entre o fechamento e abertura do rótulo "body". Para adição de comentários é utilizado a sintaxe "<!--comentário-->".

A página web muitas vezes não é só estruturada no HTML, também é usado o CSS e algumas vezes funções em JavaScript, para configuração do estilo da página e das funções que regulamentam o comportamento de botões dentro da página. Para uma página ser bem organizada é importante separar e identificar o CSS ou JavaScript utilizado dentro do HTML.

4.2.2.2 Servidor WebIOPi

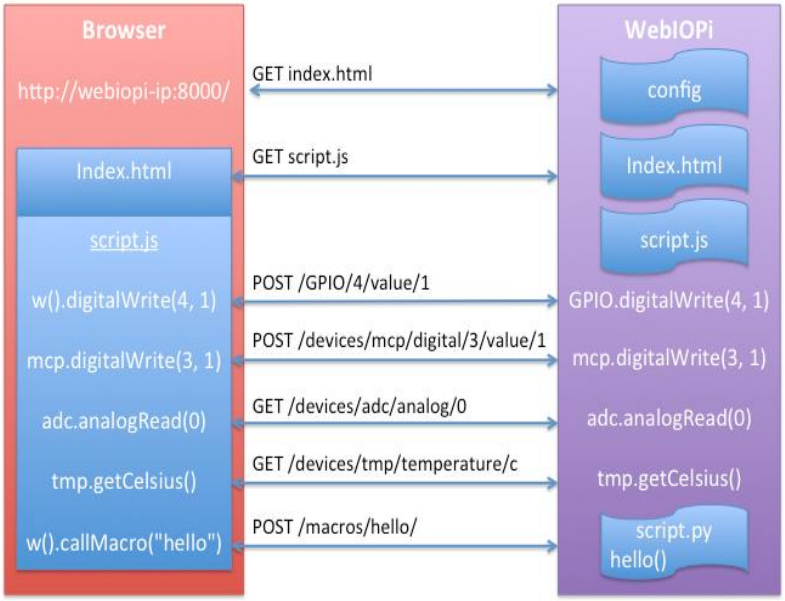
O WebIOPi é uma poderosa ferramenta para aplicações IOT *(Internet of Things*) do Raspberry Pi. Ele normalmente é usado para controlar e utilizar as portas gpio, além de sensores e conversores para aplicações *web*.

O servidor é escrito em Python e tem opção de carregar e executar scripts externos. Para construir a interface do usuário podem ser utilizadas funções em JavaScript e HTML, sendo que essa aplicação é compartilhada utilizando um IP dentro da rede.

O WebIOPi possui um servidor HTTP que fornece os recursos HTML necessários, uma REST (*Representational State Transfer*) e um API (*Application Programing Interface*) para controlarem a aplicação. O HTML é primeiramente carregado na aplicação e logo após o script escrito em JavaScript fará a ponte entre o REST API e a interface do usuário. A Figura 38 exemplifica o papel do WebIOPi

como servidor e sua relação com o usuário. A porta de comunicação padrão do servidor é a porta 8000 no endereço de IP do dispositivo servidor. (WEBIOPI.ORG, [201-]).

Figura 38 – Esquema de funcionamento da ferramenta WebIOPi



Fonte: Webiopi.org, [201-]

Para o desenvolvimento da aplicação Web, foi utilizado um script central em Python de onde o WebIOPi carrega as funções, e as páginas em HTML carregam a interface que o usuário estará apto a navegar. Dentro dessas páginas ainda temos algumas funções em JavaScript para controle das mesmas. Na primeira tentativa de conexão do aplicativo com o servidor é necessário informar um nome de usuário e senha definidos pelo administrador. Após validação, a aplicação poderá ser usada normalmente.

Através da barra de menus da aplicação é possível acessar uma das quatro opções: página inicial, controle manual, controle automático e contato. A página inicial aborda apenas uma breve descrição do projeto, com uma foto atual do robô e os logotipos da UFTM e do programa do mestrado no rodapé. Essa página é representada pela Figura 39.

Figura 39 – Página inicial do aplicativo web acessada por notebook

Fonte: Do autor, 2019

A parte escrita e a imagem foram todos centralizados para obter melhor ajuste quando for usado por celulares e *tablets,* que possuem uma tela de dimensões e resolução menores.

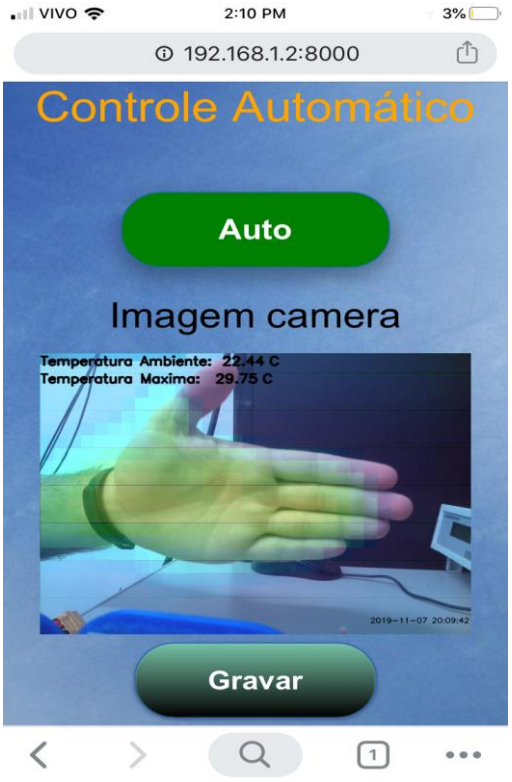
O modo manual apresenta uma interface simples, contendo os botões para comando do robô, a imagem térmica em tempo real e um comando para gravação de vídeo. Algumas funções em JavaScript na página HTML foram adicionadas para atualização dos botões e chamada das funções principais. A Figura 40 apresenta a página de navegação do controle manual do robô.

Figura 40 – Página da navegação manual aberta em notebook

Fonte: Do autor, 2019

Na Figura 40 os botões de navegação do protótipo estão posicionados na porção inferior, totalizando 5 botões de navegação, a imagem térmica, de resolução 640 x 480 *pixels,* foi posicionado ao centro da página. A função gravar muda o estado ao ser clicada iniciando a gravação e mudando a escrita do botão para “parar”. O servidor web também pode ser acessado de celulares, na Figura 41 mostra a aba de modo automático a partir de acesso de um celular Iphone modelo 6S.

Figura 41 – Página da navegação automática acessada por celular



Fonte: Do autor, 2019

Na WebIOPi um arquivo em Python é carregado na interface, e nele são definidas as ações das entradas e saídas que a página HTML utilizará. Assim são criadas funções, chamadas macros, que serão usados na página HTML para chamadas das funções em JavaScript. Para que uma função seja definida como um macro ela deve conter o marcador “@WebIOPi.macro”. É deste modo que ao clicar em um botão para um comando de seguir em frente o comando chega aos pinos de

saída do Raspberry Pi. A Figura 42 mostra um fragmento do cabeçalho da página HTML onde são inseridas as funções JavaScript.

Figura 42 - Funções em JavaScript dentro da página HTML

```
function init() {
    // automatically refresh UI each seconds
    setInterval(updateUI, 1000);
    $("#bt_mode").css("background-color", "Red");
}
function updateUI() {
     webiopi().callMacro("getMode", [], macroCallback);
     webiopi().callMacro("getMode2", [], macroCallback2);
     }

function macroCallback(macroName, args, data) {
    $("#bt_mode").text(data);
}
 function toggleMode() {
    if ($("#bt_mode").text() == "Auto") {
        webiopi().callMacro("setMode","Manual", macroCallback);
        $("#bt_mode").css("background-color", "Red");
    }
    else if ($("#bt_mode").text() == "Manual") {
        webiopi().callMacro("setMode", "Auto", macroCallback);
        $("#bt_mode").css("background-color", "Green");
    }
}
```

Fonte: Do autor, 2019

Na figura a função “callMacro”, dentro das funções JavaScript, faz a chamada das funções macro dentro do arquivo em Python. Logo a cada execução dessas funções dentro da página HTML as funções macro do arquivo em Python são executadas. No trabalho em questão foram utilizados sete macros, cinco deles para navegação (frente, direita, esquerda, parar e reverso) um para habilitar ou desabilitar gravação da câmera térmica, e um para habilitar navegação automática ou manual.

3.2.3 Processamento e transmissão de imagens

A visão computacional é uma área que está expandindo rapidamente devido ao acesso a câmeras com melhores resoluções e recursos a preços cada vez mais acessíveis. A visão computacional nada mais é do que a transformação dos dados advindos de uma câmera de vídeo em uma decisão ou uma representação. (BRADSKI; KAEHLER, 2008).

No protótipo foi utilizado a câmera Pi em conjunto com o sensor térmico matricial AMG 8833 para identificar fontes de calor. Essa foi uma alternativa

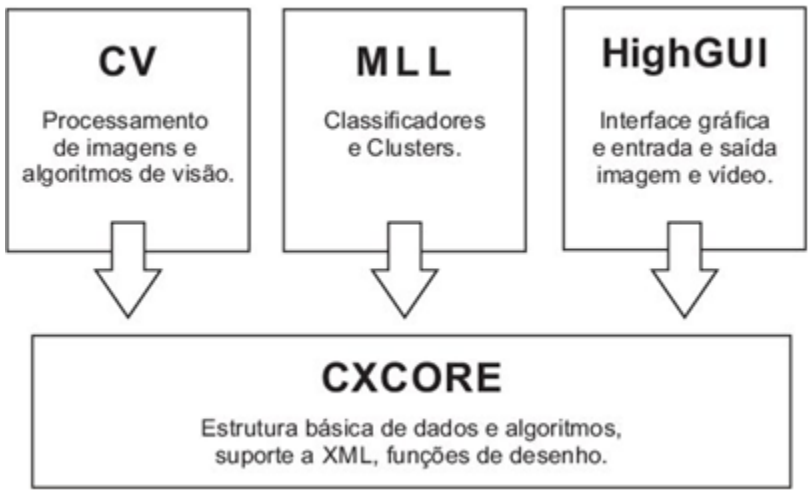
encontrada para baixar o custo ao utilizar uma câmera térmica visto que, as câmeras térmicas presentes no mercado possuem custos superiores como será visto nos resultados. Para processamento das imagens foi utilizado a biblioteca OpenCV, que é uma ótima ferramenta no desenvolvimento de projetos no campo da visão computacional.

3.2.3.1 Biblioteca OpenCV

OpenCV é uma biblioteca aberta de visão computacional que possui módulos na área de processamento de imagens, estrutura de dados, álgebra linear, entre outros. A biblioteca é desenvolvida em C e C++ e pode ser usada em interfaces como Python, Matlab, Ruby entre outros. (OPENCV, 2019).

Como a plataforma foi criada com o objetivo de ajudar o usuário a criar aplicações complexas em tempo hábil menor, o OpenCV contém mais de 500 funções em diversas áreas da visão, como imagens médicas, segurança, interface de usuário e robótica. O aprendizado de máquina também é uma parte importante da visão computacional, principalmente no segmento de classificação de imagens. Logo a OpenCV também desenvolveu uma biblioteca própria de uso geral de aprendizado de máquina, a MLL (*Machine Learning Library*). (BRADSKI; KAEHLER, 2008). O OpenCV é estruturado em cinco partes, como pode ser visto na Figura 43.

Figura 43 – Diagrama básico da biblioteca OpenCV



Fonte : Bradski; Kaehler, 2008

O componente CV possui as informações básicas do processamento de imagem e algoritmos de alto nível. MLL é responsável pelo aprendizado de máquina, e ainda possui classificadores estatísticos. HighGUI contém algumas rotinas para armazenar e rodar vídeos e imagens. O CXCORE contém as estruturas básicas do programa.

O OpenCV iniciou-se de uma pesquisa da Intel para aplicações avançadas de CPU. A biblioteca foi concebida no intuito de fazer a visão computacional universalmente disponível. Os principais objetivos idealizados para a criação do OpenCV foi a disseminação do conhecimento propondo uma estrutura de desenvolvimento comum e uma visão de pesquisa avançada por promover um software aberto e otimizado para estruturas básicas.

3.2.3.2 Formação da imagem

A primeira etapa necessária para que a imagem térmica seja montada é processar a imagem recebida pela câmera Pi. Para isso foi definido trabalhar com uma resolução de 640 por 480 pixels, pois apesar da câmera pi possuir até cinco megapixels de resolução, a resolução do sensor térmico é menor adicionado ao fato de que a transmissão será mais veloz comparado à alta resolução. A taxa de quadros por segundo (fps) foi definido em 24, para melhor ajuste na interpolação. Para adquirir os valores da imagem para um vetor do estilo RGB (Red, Green, Blue) foi utilizada a função "*PiRGBArray*", responsável por transformar os valores dos *pixels* em vetores RGB facilitando as operações matriciais que serão posteriormente realizadas no código.

Outro ponto importante é a utilização da biblioteca nativa do sensor AMG8833, a Adafruit_AMG88xx, que permite utilizar funções de alto nível para adquirir e manipular os dados do sensor. A temperatura de cada célula é fornecida na escala Celsius, em uma lista bidimensional sendo o primeiro indexador a linha e o segundo a coluna. Toda a comunicação do sensor foi feita pelo barramento I2C.

A biblioteca utilizada para formação da imagem do sensor é conhecida como *Pygame*. Esta biblioteca é de código aberto, geralmente utilizada em conjunto com Python no desenvolvimento de jogos, sendo portátil para um número alto de plataformas. Em poucas palavras o *Pygame* funciona como uma "*game engine*", que facilita o desenvolvimento dos jogos. Seus processos básicos são redenrização

para gráficos 2D e 3D, detecção de colisão, suporte à sons, inteligência artificial, entre outros. (PYGAME, [20-]).

A Figura 44 indica a parte do código responsável pela leitura e atribuição de cores aos pixels da imagem que foi formada a partir dos dados do sensor. Algumas funções e parte da estrutura foi adaptada da biblioteca Adafruit_AMG 88xx.

Figura 44 – Fragmento de código para montagem da imagem térmica

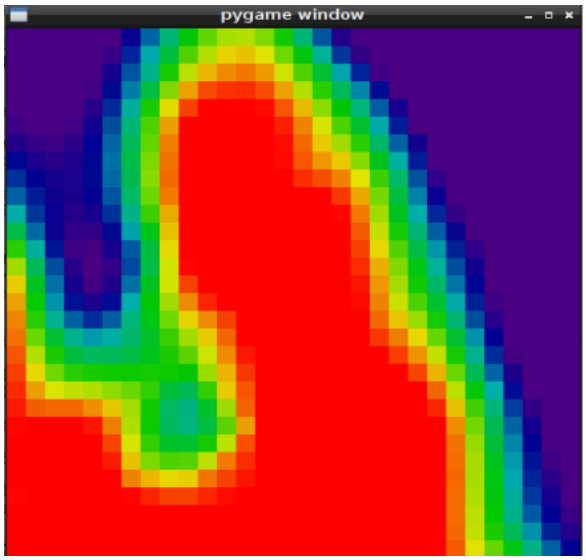
```
MINTEMP = 27

# Range com temperatura minima e maxima do sensor
MAXTEMP = 32

#Quantos valores de cores serão utilizados
COLORDEPTH = 1024

os.putenv('SDL_FBDEV', '/dev/fb1')
pygame.init()

#inicializando o sensor
sensor = Adafruit_AMG88xx()

points = [(math.floor(ix / 8), (ix % 8)) for ix in range(0, 64)]
grid_x, grid_y = np.mgrid[0:7:32j, 0:7:32j]


height = 640
width = 480

#A lista de cores que pode-se escolher
blue = Color("indigo")
colors = list(blue.range_to(Color("red"), COLORDEPTH))

#Criando o vetor de cores
colors = [(int(c.blue * 255), int(c.green * 255), int(c.red * 255)) for c in colors]

displayPixelWidth = width / 24
displayPixelHeight = height / 30

lcd = pygame.display.set_mode((width, height))

lcd.fill((255,0,0))

pygame.display.update()
pygame.mouse.set_visible(False)

lcd.fill((0,0,0))
```

Fonte: Do autor, 2019

No código apresentado, as temperaturas mínima e máxima são definidas pelo operador. *Pixels* com valores de temperatura mais próximos do mínimo, 27 °C no código, recebem uma cor mais azulada. Já para temperaturas altas, perto do máximo, são atribuídos cores avermelhadas aos pixels. As temperaturas intermediárias receberam tons mais esverdeados, amarelados, que vão depender da composição RGB atribuída ao pixel. O *display*, definido como “lcd” dentro do código, foi responsável por montar a imagem final dentro de funções da biblioteca Pygame. Cada pixel pode receber atributos de RGB nos valores de 0 a 255.

A imagem gerada pela interpolação da temperatura das células e matriz de cores pode ser visualizada na Figura 45. Nela os elementos só podem ser destacados quando há um gradiente de temperatura. Corpos com a mesma temperatura não são identificados separadamente.

Figura 45 – Imagem térmica gerada pelo sensor AMG 8833



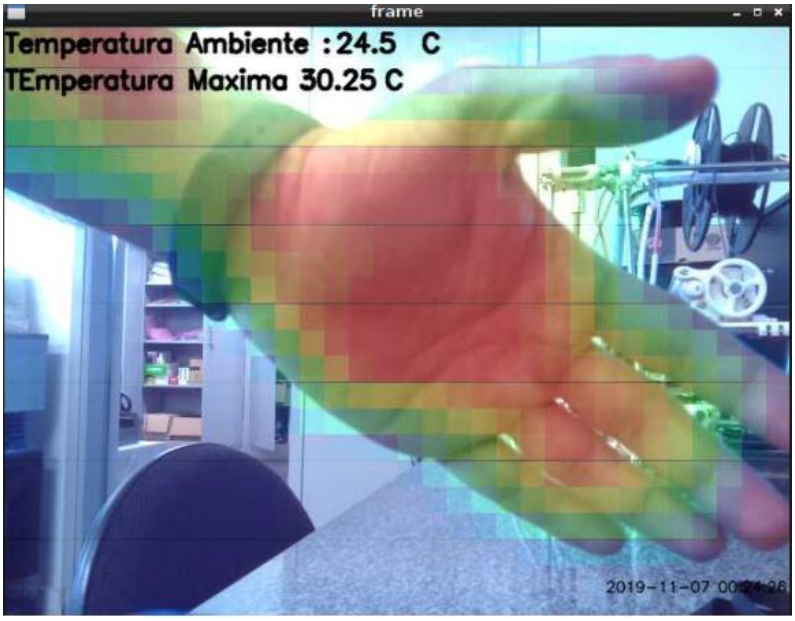
Fonte: Do autor, 2019

No desenvolvimento da imagem térmica foi utilizado sobreposição, ajuste de posição e escrita de valores de temperatura na tela. Primeiramente, as imagens foram convertidas de RGB para BGRA (Blue, Green, Red, Alpha). O espaço de cores BGRA possui um quarto canal, chamado de alpha, que representa a opacidade das cores no pixel. Com essa característica é possível combinar as imagens, pois a dimensão Alpha é capaz de atribuir tons de transparência diferentes para imagens.

Após a conversão para o espaço BGRA ser concluída é realizada uma transposição entre a imagem do sensor AMG 8833 e a câmera Pi. Na imagem produzida pelo sensor térmico foi atribuído uma opacidade maior que o da câmera Pi, assim, o fundo se sobressaiu sobre o gradiente de cores da imagem térmica. A proporção de transposição foi de 30% para a imagem térmica e 70% para a imagem da câmera Pi. Além disso, foi executado um ajuste de posições entre as duas imagens, para que a justaposição representasse com maior fidelidade a imagem real.

No final foi adicionado também os valores da temperatura ambiente e temperatura máxima detectada para que o usuário possa identificar em termos quantitativos os tons de cores. A Figura 46 apresenta a imagem térmica resultante da justaposição da câmera e sensor térmico.

Figura 46 – Imagem resultado da sobreposição de imagem térmica a imagem da câmera do Raspberry Pi



Fonte: Do autor, 2019

A imagem térmica apresentada demonstra bem o contraste de cores resultante do gradiente de temperaturas. A imagem ao fundo da mão, apresenta uma temperatura menor que o limiar mínimo mencionado anteriormente, por isso essa parte da imagem possui uma coloração azulada, representando provavelmente a temperatura ambiente. A mão, posicionada à frente do sensor, representa uma fonte de calor, com temperatura maior no centro, assim as cores dos pixels alteram-se seguindo o gradiente de temperatura proveniente do cálculo da matriz RGB. Para temperaturas mais altas, perto do limiar máximo, a coloração se torna avermelhada.

Além disso foi adicionado uma rotina para notificar o usuário com mensagens de alerta caso um objeto muito quente esteja próximo. Outro ponto é a possibilidade da gravação de vídeos em formato “.avi”, vídeo gerado pela junção de todos os quadros processados no código.

3.2.4 Algoritmo de navegação

O algoritmo de navegação proposto pode ser dividido em duas partes. Para realização das tarefas de desvio e contorno do obstáculo foi utilizado uma rede neural utilizando as ferramentas TensorFlow e Keras. Já a estratégia responsável por determinar o caminho a seguir e cálculo de trajetória foi o algoritmo Intelligent *Bug* introduzido por Zohaib et al (2013). Esse algoritmo foi escolhido por ser de baixo custo computacional e, entre os demais algoritmos *Bug,* é o que objetiva principalmente diminuir o tempo e distância percorrida.

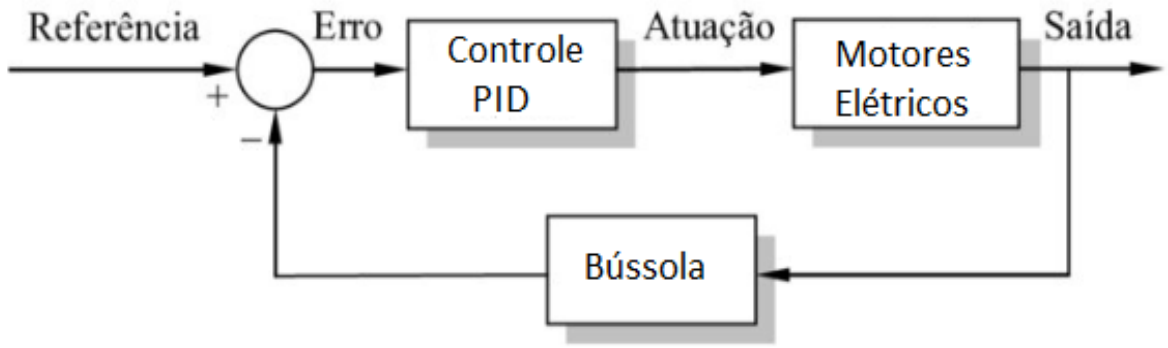
Para garantir que o robô siga a trajetória determinada, e para corrigir possíveis diferenças de rotação entre os motores, e evitar que aconteça desvios fora do esperado, foi utilizado um controlador PID (proporcional, integral, derivativo). Nos próximos subtópicos serão abordadas as estratégias de controle.

3.2.4.1 Controle PID

No algoritmo desenvolvido, a distância total da trajetória a ser percorrida e o ângulo a ser tomado como referência são informados e a cada ação são atualizados após correções da trajetória. Sabendo o valor do ângulo a seguir o robô utiliza a bússola como principal elemento sensor e atua no PWM dos motores acoplados as rodas para dar a direção correta. Esse tipo de controle é necessário pois os motores, conjunto eixo e rodas, mesmo sendo do mesmo modelo, não garantem ser idênticos em performance, assim, em alguns casos, um motor tem maior rotação utilizando a mesma faixa de tensão. Para corrigir esses erros, foi utilizado o controle PID, que busca a minimização ou eliminação do erro, dentro de uma malha, utilizando o valor retroativo (*feedback*) do sinal recebido pelo sensor. O erro é calculado pela diferença entre o valor objetivo (*setpoint*) e o mensurado.

A Figura 47 traz o diagrama do controle PID. A malha utilizada é fechada, pois nela é possível receber os dados de sensores e mensurar a qualidade do ajuste do controlador através do erro gerado. O erro é gerado pela diferença entre o valor objetivo e a variável de processo. O controle PID atuará no intuito de minimizar o erro. Após o ajuste do controle, o PID também atuará em casos onde há algum tipo de distúrbio não planejado.

Figura 47 – Diagrama de controle em malha fechada



Fonte: Adaptado de CARRARA, 2015

Em termos matemáticos, esse tipo de controle é definido pelas Equações 9 e 10:

$$u(t) = Ke(t) + K_i \int_0^t e(t)dt + K_d \frac{de}{dt} \quad (9)$$

$$e = \hat{a}ngulo(objetivo) - \hat{a}ngulo(medido) \quad (10)$$

Na Equação 10, u é o sinal de controle, e é o sinal do erro dado pela equação 2, K é o termo de ganho proporcional, K_i é o termo de ganho integral e K_d é o termo de ganho derivativo. O sinal final é dado pela soma dos três termos, o termo P (proporcional ao erro), o termo I (proporcional a integral do erro) e o termo D (proporcional a derivada do erro). (ASTROM; MURRAY, 2008).

O controle proporcional é definido pela multiplicação do erro do processo pelo valor do ganho proporcional. O controle é simples, e quanto maior o valor do ganho K, mais rápido será a resposta do sistema. Porém ao elevar-se de maneira exagerada o valor do ganho pode ocorrer instabilidade no sistema ou amplificação de ruídos indesejados. No caso do protótipo, foi verificado que apenas o ganho proporcional não foi satisfatório para uma resposta rápida e estável de correção.

O controle integral é utilizado na tentativa de eliminar os erros de regime permanente, aquele onde o sinal estabiliza em um valor não desejado, através da integração dos valores de erro no tempo. O controlador integral é geralmente utilizado juntamente com o controlador proporcional ou proporcional-derivativo.

Valores muito altos de ganho integral podem ocasionar aumento do sobressinal e maior tempo de estabilização. (ASTROM; MURRAY, 2008). A parte integral do controle do protótipo foi necessária para evitar erros permanentes nos ângulos da trajetória, otimizando o caminho a ser percorrido e diminuindo erros acumulados.

O controle derivativo prediz a ocorrência de determinada ação, corrigindo um erro que ainda está por vir, observando a curva tendência dos erros anteriores. Esse controle tende a estabilizar a tangente da curva do erro a zero. Essa oposição a variação do sinal ajuda na estabilização e combate grandes oscilações no sistema. No protótipo o controle derivativo foi importante para evitar grandes desvios nas correções da trajetória e otimizar o tempo gasto.

O controle PID foi utilizado para modificar os valores de PWM nos motores esquerdo e direito. Os valores de PWM foram inicializados em 35 % do valor máximo, que representará a velocidade média dos motores. A cada leitura da bússola, os valores do controlador sofrem mudanças e consequentemente ocorre o ajuste de PWM. O ajuste é inversamente proporcional aos lados, ou seja, enquanto o PWM dos motores do lado direito cresce, os motores do lado oposto têm seus valores reduzidos. Os valores utilizados de ganho proporcional, integral e derivativo foram encontrados após testes de desvios da trajetória e também levando em consideração o atrito entre as rodas e o material do terreno local. Não foram utilizados controles e metodologias mais complexos nesta parte do trabalho pois o PID foi utilizado apenas como forma de correção de pequenos erros no giro dos motores e dificuldades do terreno. Os valores adotados no código estão na Tabela 4.

Tabela 4 – Valores obtidos de PID

Parâmetros	**Valores**
Proporcional	0.4
Integral	0.01
Derivativo	0.1
Tempo de amostra	0.2

Fonte: Do autor, 2019

Os valores apresentados na Tabela 4 foram ajustados após a realização de mais de 50 testes de navegação, observando os valores que obtiveram a melhor resposta para o movimento de correção em um menor tempo de oscilação. O controle do PWM das quatro rodas utilizou esse PID, sendo que o PWM dos motores

do lado direito do robô estava inversamente proporcional ao lado esquerdo, como já citado, para obter um melhor ajuste do movimento de curva para correção do ângulo.

3.2.4.2 Rede neural

A rede neural artificial foi aplicada no projeto para prever e controlar as ações de desvio e contorno de obstáculos encontrados que pudessem causar colisões caso o robô não tomasse as ações corretas. A arquitetura escolhida foi a rede Perceptron Multicamadas, e a estrutura da rede consiste em quatro camadas de neurônios, sendo uma camada de entrada, duas camadas ocultas e uma camada de saída. Nos primeiros testes foi utilizado apenas uma camada oculta, porém após a introdução da segunda camada oculta, a acurácia do modelo aumentou cerca de 7% no primeiro teste, justificando a necessidade das duas camadas. Foi utilizado o aprendizado supervisionado, ou seja, foij fornecido a rede uma resposta desejada a determinado condição conhecida do ambiente.

No desenvolvimento da rede foram empregadas as bibliotecas TensorFlow e Keras, que são plataformas de código aberto para aplicações em aprendizado de máquina. O TensorFlow é a segunda geração de sistemas projetados para aplicação em aprendizagem profunda da empresa Google. Esta biblioteca oferece vários níveis de abstração para que o usuário molde o código a sua necessidade. O Keras é uma Interface de Programação de Aplicações em rede neurais de alto nível, desenvolvida em Python, que surge com o objetivo de facilitar a construção do modelo pelo usuário para que o mesmo possa economizar esforços na programação da estrutura da rede e focar na estratégia de resolução do problema. O Keras pode ser utilizado dentro da biblioteca Tensoflow através do comando “tf.Keras”. (GÉRON, 2017).

A primeira camada é responsável por receber os dados de entrada, que no caso da aplicação do projeto, consiste nos cinco sensores ultrassônicos, sendo três posicionados a frente do carro, um a direita e um a esquerda. Os dados levantados para alimentar a rede possuem 5 posições de entrada e uma de saída. A saída, que representará a ação, foi distribuída em 8 valores inteiros conforme tabela 5.

Tabela 5 – Valores correspondente entre o valor e ação

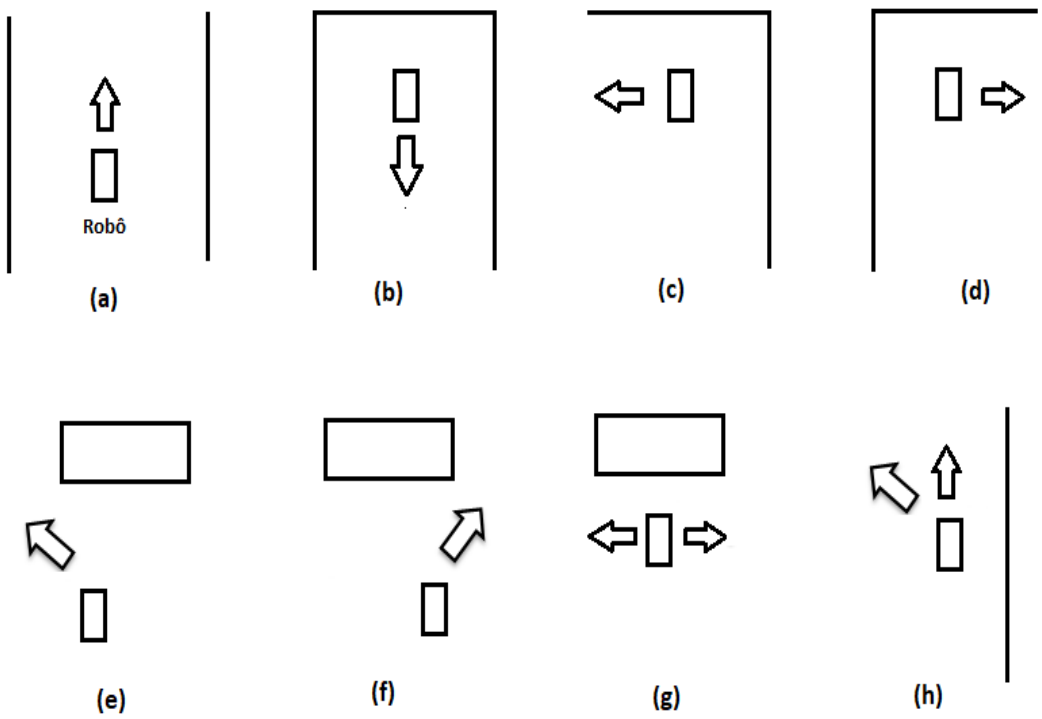
Valor	Ação
1	Desvio de 45 graus a direita
2	Desvio de 90 graus a direita
3	Desvio de 45 graus a esquerda
4	Desvio de 90 graus a esquerda
5	Seguir Reto / Alvo
6	Desvio de 180 graus
7	Desvio de 25 graus a direita
8	Desvio de 25 graus a esquerda

Fonte: Do autor

Na Tabela 5 as ações 1,2 e 7 representam desvios a direita, as ações 3, 4 e 8 são desvios a esquerda, o valor 8 representa o giro de 180 graus para inversão de sentido e a ação de valor 7 representa o movimento de seguir em direção ao alvo em trajetória otimizada segundo algoritmo IBA. Desvios com ângulo de 25 graus são utilizados para situações desvio fraco, apenas para ajustar a rota e evitar colisões laterais, os ângulos de 45 graus são usados para desviar do obstáculo há uma distância média, levando em consideração a rotação do eixo no sentido do comprimento do protótipo. Os desvios fortes (90°) são usados quando o protótipo está muito próximo do obstáculo e o risco de colisão é eminente. A rotação em 180 graus foi utilizada exclusivamente para casos em que o robô esteja cercado por paredes dos três lados e há única opção seja voltar no sentido contrário.

Para treinar a rede foram levantadas aproximadamente 200 configurações de dados diferentes, em possíveis situações que o robô encontraria em sua trajetória e diferentes posições de obstáculos. Na Figura 48 são exibidas algumas das configurações utilizadas para coleta de dados. Nos casos (b), (c), (d) e (g) são observados obstáculos próximos que exigem desvios maiores de 90 graus e bruscos de 180 graus. No caso a o protótipo seguirá reto, em (e) e (f) houve um leve desvio e em (h) o protótipo poderá seguir o alvo.

Figura 48 – Exemplos utilizados para levantamento de dados



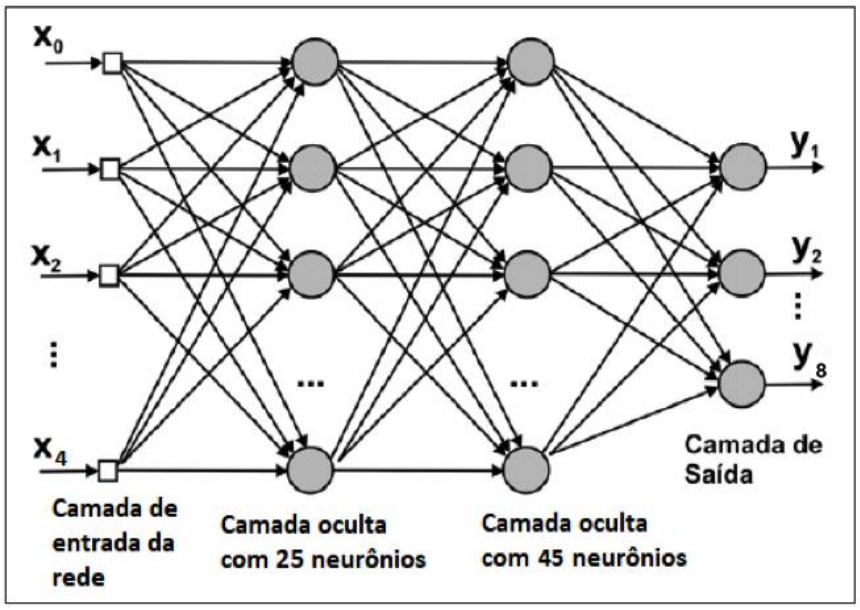
Fonte: Do autor, 2019

As redes neurais necessitam de alguns parâmetros essenciais para um bom funcionamento, chamados de hiperparâmetros, que definem sua arquitetura. Primeiramente foi definido a porcentagem dos dados de entrada que seriam divididos entre treino e teste. O treino é a etapa de ajuste de pesos e bias. A etapa de teste serve para validar o modelo, e verificar um possível caso em que a rede é supertretreinada (*overfitting*). Caso aconteça esse supertreinamento o modelo perderá sua capacidade de generalização, funcionando muito bem para os dados treinados, mas não conseguindo prever as saídas para outros dados. No treinamento da rede observou-se melhores resultados para os dados divididos em 20% para teste e 80 % para treino. Os valores de entrada dos dados, extraídos dos sensores ultrassônicos, foram normalizados para encaixar no espaço de valores [0,1] e facilitar o treinamento.

Em seguinte é definida a dimensão da camada de entrada e definida a inicialização dos pesos aleatoriamente e do bias com valores nulos. O próximo passo foi criar a primeira camada oculta com 25 neurônios utilizando a função de ativação RELU. Na segunda camada oculta foi utilizado 45 neurônios e novamente a função de ativação RELU. Esses valores foram encontrados após diversos

treinamentos, em que se utilizaram valores variando de 10 a 100 neurônios. A última camada foi composta por uma saída de 8 neurônios utilizando a função de ativação Softmax. Na camada de saída foi empregada a técnica de “*one-hot-encoding* “, que consiste em transformar os valores categóricos inteiros em vetores binários. Essa técnica ajuda na melhoria da eficiência e precisão da rede na predição das saídas. A Figura 49 demonstra a arquitetura dos neurônios utilizados na rede e as camadas descritas anteriormente.

Figura 49 – Arquitetura da rede neural



Fonte: Do autor, 2019

O hiperparâmetro de otimização utilizado foi o Adam, que é uma versão adaptada do gradiente descendente e baseia-se na estimativa de momentos de baixa ordem. A função de perda (*loss function*) utilizada foi a entropia categórica cruzada (*categorical cross-entropy*) que é responsável por descrever o desvio do valor da rede treinada em relação a melhor solução utilizada. Essa função medirá o progresso de aprendizado da rede e sua precisão. A função categórica é utilizada principalmente para classificação em treinamento supervisionado utilizando o *one-hot-encoding*.

Para escolha das épocas e tamanho do lote (*batch size*), foi considerado um valor pequeno para conseguir uma melhor proporcionalidade baseado na quantidade dos dados de entrada da rede. O tamanho do lote define o número de valores de entrada que serão treinadas a cada iteração, podendo variar de 1 ao número

máximo de valores de entrada. Uma época é quando todo os dados de entrada passam pela rede neural uma única vez. O tamanho de lote escolhido foi 3. Nos primeiros treinamentos foram utilizados valores de épocas menores que 500 e logo após, valores entre 500 e 1000. O valor que trouxe uma maior precisão para a rede foi na configuração de 600 épocas. A eficiência da rede baseada na arquitetura descrita será apresentada nos resultados.

3.2.4.3 Algoritmo Bug de locomoção

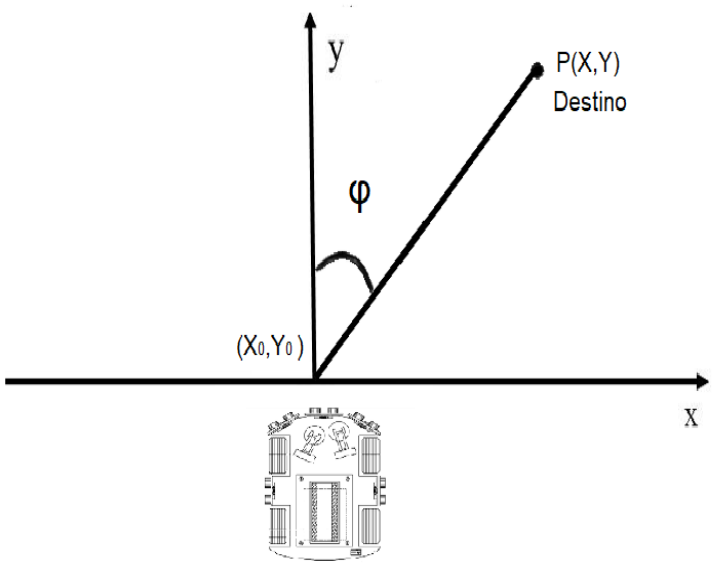
O algoritmo *Bug* utilizado foi responsável pela tomada de decisão final na locomoção do protótipo. Dentre os algoritmos conhecidos foi escolhido o IBA. Esse algoritmo é orientado ao objetivo e toma as ações de forma gananciosa, sempre procurando o menor caminho. Caso o protótipo visualize caminho livre à sua frente ele irá seguir a trajetória de menor distância entre o ponto atual e o ponto de destino. Caso haja algum obstáculo o protótipo irá desviar e contornar o obstáculo até que haja um caminho livre a seguir.

Para identificar a presença de obstáculos e a melhor ação a tomar de acordo com a leitura dos sensores foi necessário utilizar a rede neural. A rede desenvolvida interpreta os valores dos sensores e escolhe a melhor solução com uma precisão muito grande. Assim foi proposto um modelo que concilia o algoritmo IBA com redes neurais para tomada de decisão.

A parte inicial do algoritmo de navegação conta com a inicialização da rede neural artificial, com pesos e parâmetros, e dos sensores ultrassônicos, bússola digital e encoder. Após a primeira aquisição de dados, a rotina é inicializada com o usuário fornecendo a distância que o robô deve percorrer e o ângulo de destino. O ponto inicial do carrinho é considerado como a origem do plano de cartesiano de navegação do protótipo. Assim, caso o destino esteja à esquerda do protótipo, ele irá mover-se no sentido negativo do plano x (abscissas), caso contrário, o protótipo irá seguir para o plano positivo. Para futuros cálculos de trajetória, o ponto P (destino) é dissociado aos seus valores refletidos nos eixos das ordenadas e abscissas, através de cálculo de seno e cosseno da diferença do ângulo atual com o destino (φ). Quando o protótipo se move no plano positivo das abcissas seu ângulo de destino varia de 0 a 180°. Caso mova-se no sentido negativo do eixo X, o ângulo

φ varia de -180 a 0°. A Figura 50 mostra a situação inicial do protótipo, o ponto de destino P, a origem com (X_0 , Y_0) e o ângulo φ.

Figura 50 – Cálculo da trajetória final do protótipo



Fonte: do autor, 2019

Desde que a trajetória esteja livre, sem obstáculos, o protótipo chegará ao destino sem desvios, seguindo apenas as rotas calculadas pelas fórmulas de distância e ângulo. A cada avaliação da rede as coordenadas (x,y) do protótipo são calculadas, usando os valores da distância fornecida pelo sensor encoder e do ângulo fornecido pela bússola, e salvas para que a nova trajetória seja predita. Para cálculo da distância ao alvo e ângulo correto do caminho é utilizado a Equação 11 de distância entre dois pontos e Equação 12 da função arco tangente mostradas a seguir.

$$d = \sqrt[2]{(x - x_0) + (y - y_0)} \tag{11}$$

$$\varphi = \tan^{-1}\left(\frac{x - x_0}{y - y_0}\right) \tag{12}$$

Um cuidado tomado foi quanto aos valores que o ângulo resultante do arco tangente poderia assumir. Dependendo dos valores de x e y ele assumirá valores

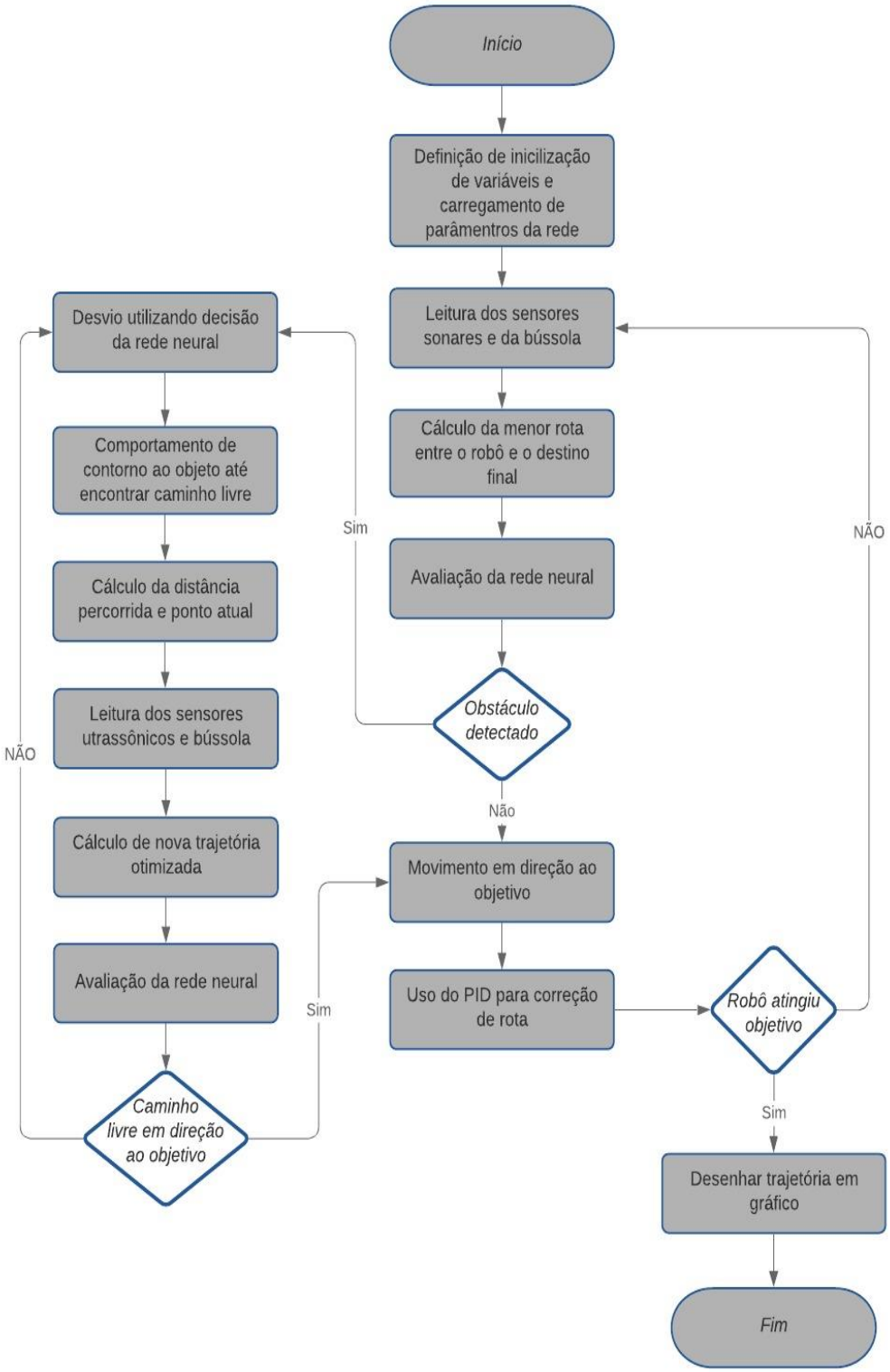
entre -180° e 180°. Logo foi analisado o quadrante de origem e destino dos pontos no plano para definir o ângulo como positivo ou negativo.

Outro ponto importante está observação da escala da bússola (0 a 360°) na inicialização e programação das rotinas de desvio. O primeiro ângulo é tomado como o ângulo de referência e a partir dele todos os cálculos são realizados. Logo no exemplo da Figura 49 o ângulo programado no código seria o valor φ mais o ângulo inicial tomado como referência. Devido a escala fixa da bússola foi necessário realizar algumas conversões de valores, como, por exemplo, garantir que sempre que um ângulo for somado ou subtraído, o valor final não ultrapasse os valores da escala. Outra conversão utilizada foi para garantir que o robô sempre siga no sentido do ângulo mais próximo e não gire mais que o necessário, sendo esse um dos problemas verificados nas transições de 0 para 360° ou de 360 para 0°.

O fluxograma da Figura 51 exemplifica as etapas de funcionamento do algoritmo desenvolvido. A avaliação da rede neural define se existe um obstáculo a frente ou não. Caso exista ele toma uma entre as 7 decisões disponíveis e caso não haja obstáculos ela toma uma decisão apenas. As decisões de desvio variam os graus de giro conforme apresentado anteriormente na Tabela 5. A cada ponto de desvio do objeto os pontos são salvos e a nova trajetória é calculada até que o protótipo obtenha caminho livre novamente. A cada decisão do robô de avanço é chamado uma subrotina de andar reto na trajetória por um percurso de 25 cm, pois a cada iteração existe uma perda de tempo com o processamento, assim esse valor foi utilizado para deixar o movimento mais fluido ao mesmo tempo que assegurasse que o protótipo desviaria dos obstáculos com segurança. Após isso o robô avalia novamente utilizando a rede neural.

No caso de obter caminho livre, o robô calcula o ângulo e a distância na qual ele deve percorrer para chegar ao destino. Após os cálculos o robô segue a trajetória em linha reta até que um novo obstáculo seja detectado. Caso o robô possua caminho livre a frente, mas lateralmente esteja bloqueado por algum obstáculo, irá acionar uma rotina do código que não permitirá ao protótipo virar na direção do obstáculo, evitando a colisão. O robô então assume o último ângulo calculado e segue em linha reta. Após nova avaliação da rede e cálculo da nova trajetória, caso a decisão 5 seja tomada novamente é avaliada as laterais para observar a viabilidade do desvio. O ciclo do algoritmo só termina quando o robô chega ao ponto de destino, definido no início do código pelo operador.

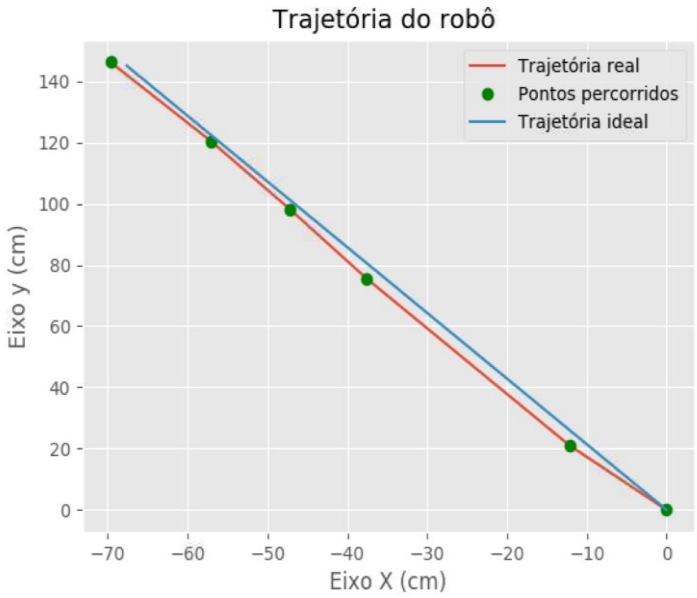
Figura 51 – Fluxograma da tomada de decisões da navegação do protótipo



Fonte: Do autor, 2019

Também foi considerado uma tolerância de 10 cm como critério de chegada ao ponto objetivo, para que o robô chegando próximo ao ponto final evite movimentos curtos, usados apenas para chegar ao ponto exato, economizando tempo e distância percorrida. Ao alcançar o objetivo é desenhado o gráfico mostrado na Figura 52.

Figura 52 - Exemplo de trajetória plotada ao final do percurso



Fonte: Do autor

O gráfico da Figura 52 é plotado ao final do trajeto utilizando os pontos x e y do plano de movimento do robô, salvos após cada ação da rede, onde é comparado a trajetória real com a trajetória calculada para aquela configuração de obstáculos, utilizando as funções da biblioteca Matplotlib, que é específica para geração de gráficos em duas dimensões.

4 RESULTADOS E DISCUSSÕES

Os resultados foram divididos em três partes distintas, primeiramente será discutido os resultados da eficiência da medição da câmera térmica para diferentes valores de temperatura dentro da faixa de valores do sensor. Em segundo será discutido os resultados da rede neural e do algoritmo de navegação proposto. Finalmente será analisado os custos totais do projeto.

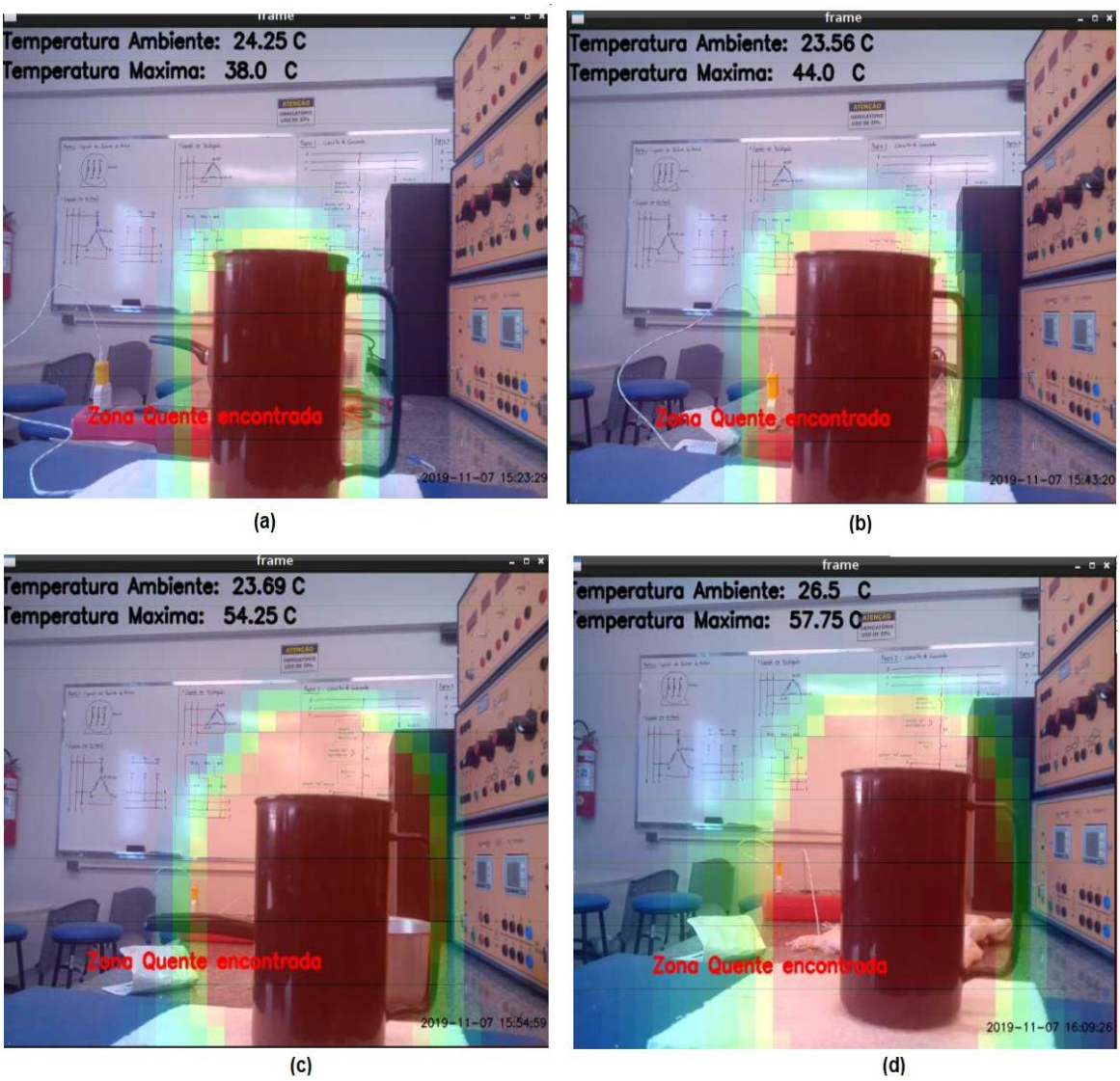
Um ponto importante para abordar é o alcance máximo do sinal de transmissão de informações ao protótipo. O Raspberry Pi 3 modelo b+ possui uma placa integrada de Wi-Fi que trabalha com duas bandas de frequência: 2,4 GHz e 5GHz. Logo o alcance do sinal Wi-Fi, que habilitará o usuário o controle sobre o protótipo, dependerá, principalmente, da potência da fonte transmissora do sinal, e do número e tipo de paredes bloqueando o sinal. O roteador utilizado foi um D-Link DIR-615 com autonomia de 100 m^2 . (D-LINK, 2017).

4.1 CAMERA TÉRMICA

A primeira parte dos resultados foi obtida analisando a leitura do sensor térmico AMG 8833. A imagem térmica apresentada ao usuário contém os valores de temperatura ambiente, fornecido através de um termistor do próprio sensor térmico, e a temperatura da célula da matriz 8x8 do sensor térmico que apresenta maior temperatura, simbolizando uma zona térmica mais crítica.

Para análise da eficiência da leitura das células infravermelhas do sensor, foram realizados testes em diferentes valores de temperatura dentro da escala de leitura da câmera térmica. Os testes consistiram em aquecer água dentro de um recipiente (copo de alumínio) posicionado a 30 cm do protótipo, até a temperatura adequada e colocado sobre isopor para diminuir o processo de perda de calor. Para aferir a precisão das medidas, foi utilizado um termopar tipo k com escala de -200 a 1200°C e erro de 0,4 °C. As temperaturas analisadas variaram aproximadamente de 5 em 5 °C dentro de uma faixa de 25 °C a 80 °C. A Figura 53 apresenta algumas das medições realizadas utilizando a câmera térmica do protótipo bem como o mapa de calor gerado pelo código.

Figura 53 – Teste de precisão da câmera térmica. Valores testados: (a) 40°C (b) 45°C (c) 55° e (d) 60°C.



Fonte: Do autor, 2019

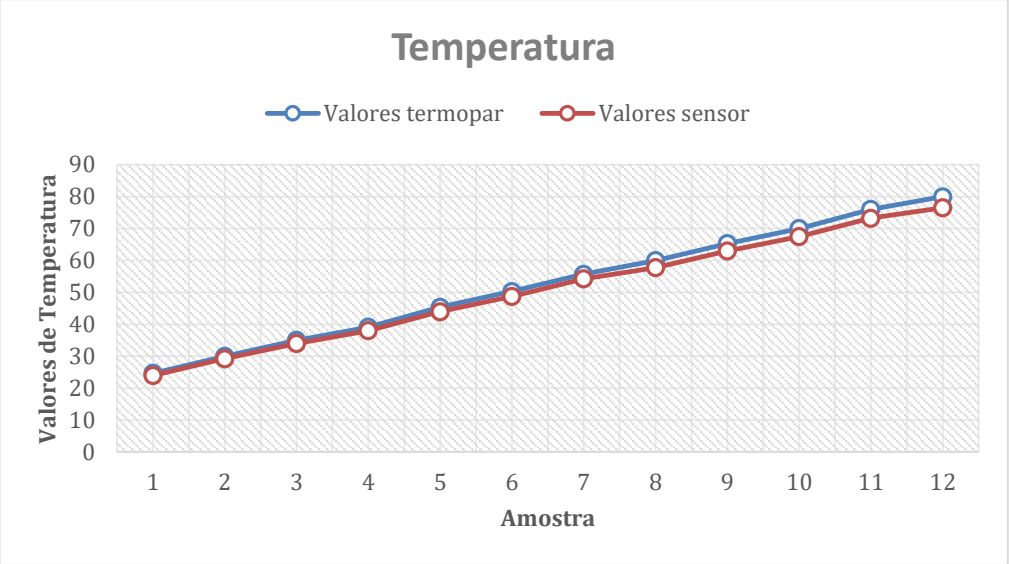
Os resultados foram coletados e apresentados na Tabela 6. Devido a pequenas variações o valor medido pelo termopar também não foi exatamente a variação de 5°C. Também foram calculados os erros absolutos e percentual para cada medida. A partir dos resultados das medidas de temperatura da Tabela 6 foi criado o gráfico apresentado na Figura 54 que relaciona a curva de medição apresentada entre o sensor térmico proposto e o termopar.

Tabela 6 – Valores apresentados no teste da eficiência da câmera térmica

Amostra	Valores medidos pelo temopar (°C)	Valores medidos pela câmera térmica (°C)	Erro absoluto	Erro (%)
1	24,7	24	0,7	2,83
2	30	29,25	0,75	2,50
3	35	34	1	2,86
4	39,1	38	1,1	2,81
5	45,4	44	1,4	3,08
6	50,3	48,75	1,55	3,08
7	55,7	54,25	1,45	2,60
8	60	57,75	2,25	3,75
9	65,3	63	2,3	3,52
10	70	67,5	2,5	3,57
11	76	73,25	2,75	3,62
12	80	76,5	3,5	4,38

Fonte: Do autor, 2019

Figura 54 – Comparação da temperaturas entre termopar e câmera térmica



Fonte: Do autor, 2019

Analisando os dados da Figura 54 percebe-se que com o aumento da temperatura, a precisão da câmera térmica tende a diminuir tanto em valor absoluto quanto percentualmente. O valor da precisão do sensor é ± 2,5% segundo dados do fabricante, logo, para efeito de comparação, utilizou-se o erro absoluto da tabela. Também foram calculados a média do erro absoluto e a medida do desvio padrão de dispersão do erro em relação as médias. Os valores obtidos para média e desvio

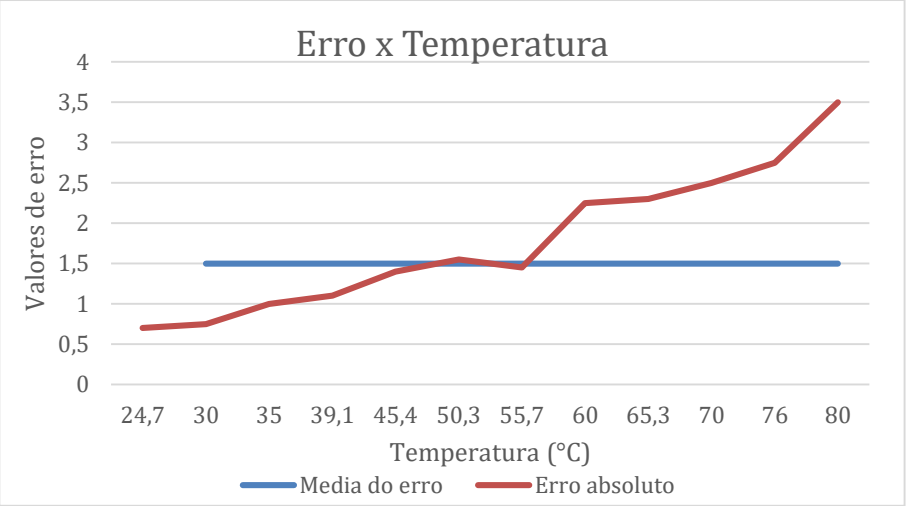
padrão estão na Tabela 7. Para evidenciar o crescimento do erro com o aumento da temperatura, foi plotado um gráfico do erro médio em comparação com os erros absolutos presente na Figura 55.

Tabela 7 – Valores obtidos de média e desvio padrão do erro

Variável	Média	Desvio padrão
Erro absoluto	1,5	0,88

Fonte: Do autor, 2019

Figura 55 – Gráfico de erro absoluto por temperatura medida



Fonte: Do autor, 2019

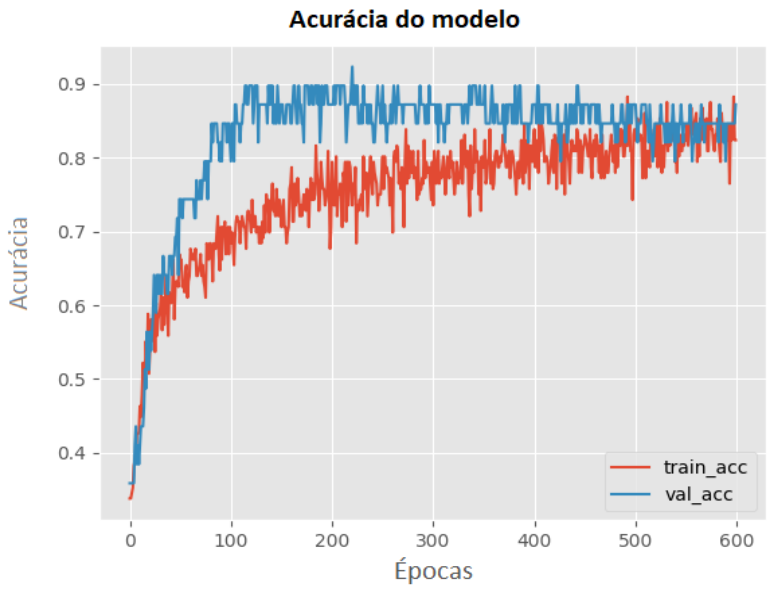
O gráfico da Figura 55 deixa claro a proporção do aumento do erro com o aumento da temperatura, sendo que em um certo momento, perto de 70°C ultrapassa a tolerância fornecida pelo fabricante. Outro ponto importante foi a redução da taxa de quadros por segundo da câmera que, com o tempo necessário para o processamento e sobreposição dos dados, diminuiu para aproximadamente 4 fps, o que teoricamente deixou a câmera mais “lenta”, mas não influenciou na precisão dos dados medidos e ainda foi possível ter uma boa percepção visual dos mapas de calor gerados.

4.2 REDE NEURAL E ALGORITMO DE NAVEGAÇÃO

Em um segundo momento foi analisado a navegação autônoma do robô. A primeira parte dos resultados da navegação advém do treinamento da rede neural. Para conseguir um valor de treinamento com valores de acurácia maior que 80% foi necessário o ajuste dos hiperparâmetros da rede a cada iteração. Os valores apresentados na metodologia foram encontrados após inúmeros testes e treinamentos da rede, e fixado, após o treinamento atingir um valor de 87% de acurácia no teste de validação.

A acurácia mede a proporção do total de acertos do modelo, representando a frequência de valores corretos previstos, avaliando além da precisão a veracidade dos resultados obtidos. Percentualmente ela é dada pela soma dos valores verdadeiros (tanto positivos quanto negativos) divididos pelo total das amostras. A Figura 56 traz o gráfico da acurácia em relação ao número de épocas.

Figura 56 – Gráfico da acurácia do modelo



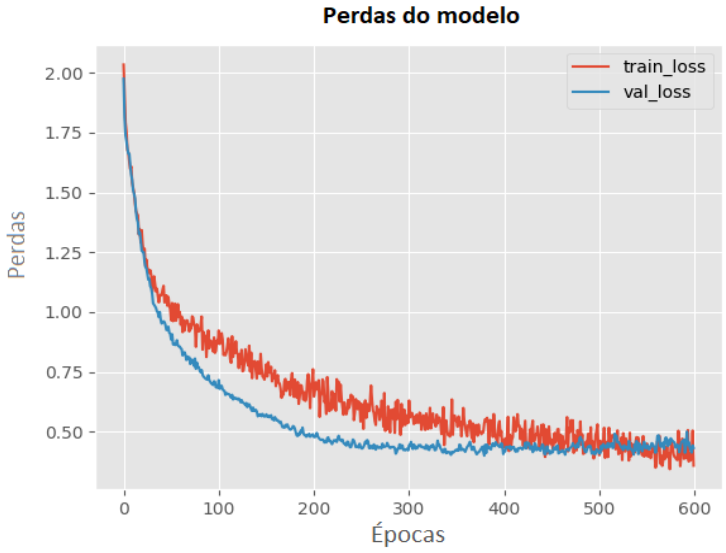
Fonte: Do autor, 2019

No gráfico são mostradas as acurácias de treinamento, em vermelho, e validação, em azul. Até a época 400 a curva de validação obteve valores superiores aos de treinamento. Em casos em que a validação obtém resultados muito superiores aos de treinamento, pode ser um indício de supertreinamento, que, como citado anteriormente, ajusta a rede para os valores treinados diminuindo sua

capacidade de generalização. Porém a partir da época 400 as curvas tenderam em convergir para um valor comum, em torno de 87% para validação e 85% para treinamento.

A função de perda ou custo (*Loss Function*), reduz todos os aspectos da rede, sendo elas boas ou ruins, em um único valor escalar, que habilita o usuário a comparar modelos de funções. Logo essa função deve ser minimizada para se obter um bom modelo. Essa função deve ser minimizada. Na Figura 57 é introduzido o gráfico da função de perda.

Figura 57 – Gráfico das perdas do modelo

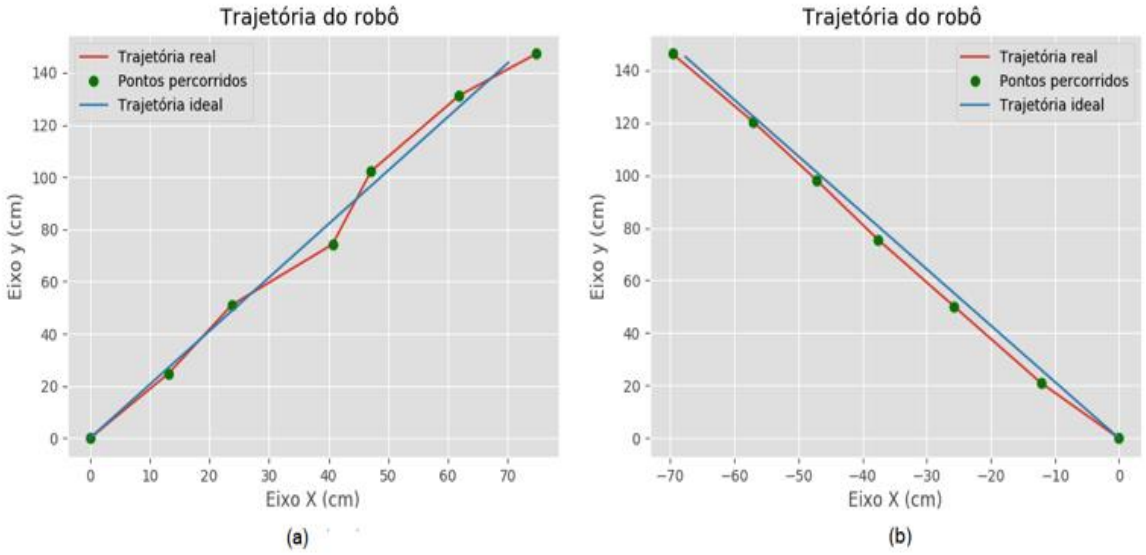


Fonte: Do autor, 2019

As curvas da função de custo apresentaram até a época 50 uma mesma tendência de decaimento, porém a partir desse valor, e até a época 400, a curva de perdas da validação alcançou resultados melhores como visto também na acurácia. A partir de 400 épocas as duas curvas convergiram a um ponto comum em 0,41, como resultado final do modelo treinado.

A partir dos resultados obtidos pela rede treinada foram realizados testes para avaliar o algoritmo proposto. O primeiro teste foi realizado sem presença de obstáculos para analisar a navegação utilizando as coordenadas cartesianas. O destino final foi colocado a uma distância de 180 cm da origem com um ângulo de $\pm$ 25 graus, ou seja, foi testado tanto o destino com giro em ângulo positivo quanto em negativo. A Figura 58 mostra a trajetória do robô nos dois testes.

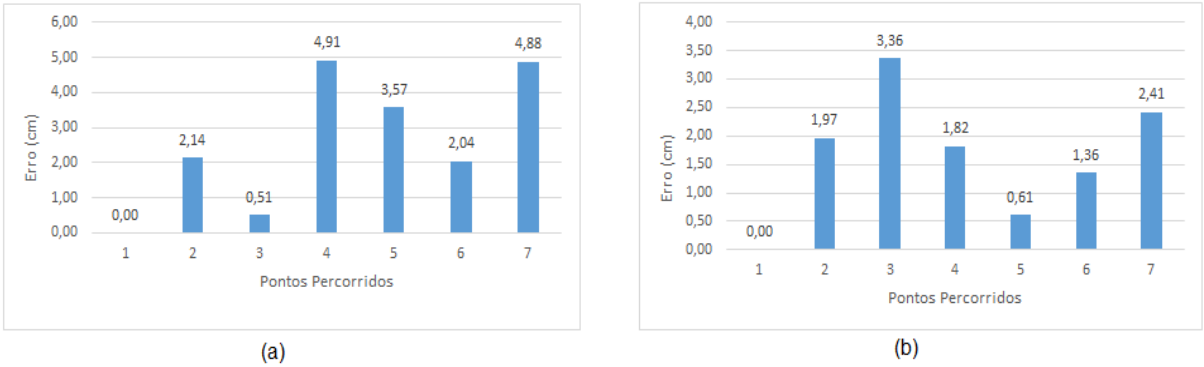
Figura 58 – Trajetória do robô no teste 1 sem obstáculos. (a) ângulo positivo (b) ângulo negativo



Fonte: Do autor, 2019

A trajetória com ângulo positivo e negativo obtiveram um pequeno erro na chegada ao destino, esse erro é devido a algumas variações na leitura da bússola e na contagem de distância do sensor óptico, por isso foi considerado, como mencionado anteriormente, um critério de parada uma variação de 10 cm das coordenadas do destino final. Para avaliar a distância ponto a ponto entre a trajetória prevista e a real foi desenhado o gráfico da Figura 59, onde o primeiro ponto percorrido é o ponto situado na origem, e os pontos seguintes percorrem a trajetória até o último ponto, próximo ao objetivo.

Figura 59 – Gráfico da distância entre os pontos da trajetória ideal e real: (a) desvio com ângulo positivo (b) desvio com ângulo negativo.

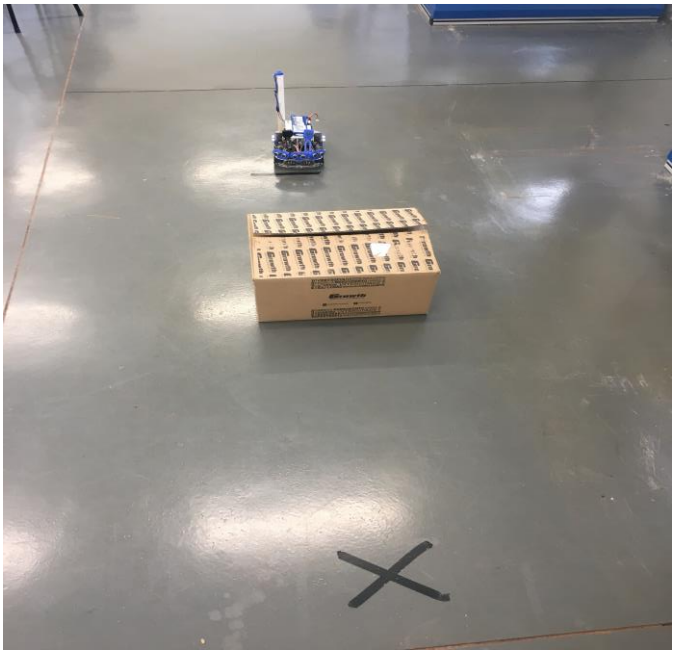


Fonte: Do autor, 2019

Percebe-se que a maioria dos erros ponto-a-ponto da trajetória para o desvio com ângulo positivo foi menor que o ângulo negativo. As oscilações fora da trajetória ocorreram devido a pequenos deslizes das rodas na rotina de giro, e por um tempo morto entre parar realmente os motores no ângulo exato da trajetória. Porém a própria rotina de seguir em linha reta utiliza o controle PID consegue corrigir essas falhas.

No segundo teste foi utilizado um obstáculo para obstruir a passagem do protótipo em direção ao destino. Como obstáculo foi utilizado uma caixa de 44 cm de comprimento por 35 de largura. Ela foi posicionada a 60 cm do ponto de origem. A distância total do ponto inicial ao final foi de 180 cm sem ângulo de desvio. A Figura 60 traz a imagem do teste sendo realizado em laboratório, e o posicionamento do robô, obstáculo e destino.

Figura 60 – Configuração do teste 2 realizado utilizando um obstáculo

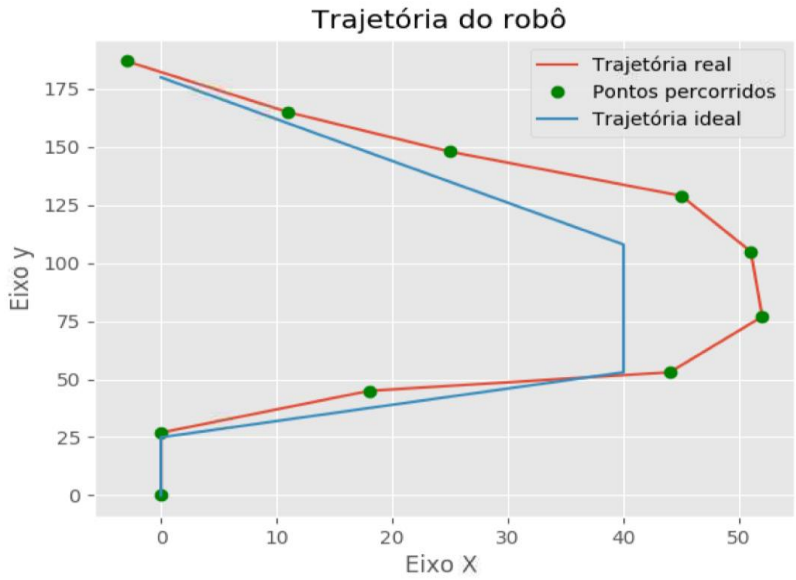


Fonte: Do autor , 2019

Na Figura 61 é abordado o gráfico da trajetória obtido com o segundo teste. Nele é possível perceber que a trajetória do robô foi similar a prevista, porém com uma distância de trajeto superior. Isso ocorreu, pois, ao contornar o obstáculo, o protótipo tomou a ação de afastar da parede para evitar a colisão e as próximas ações foram tomadas objetivando caminhar para o destino sem avançar na direção

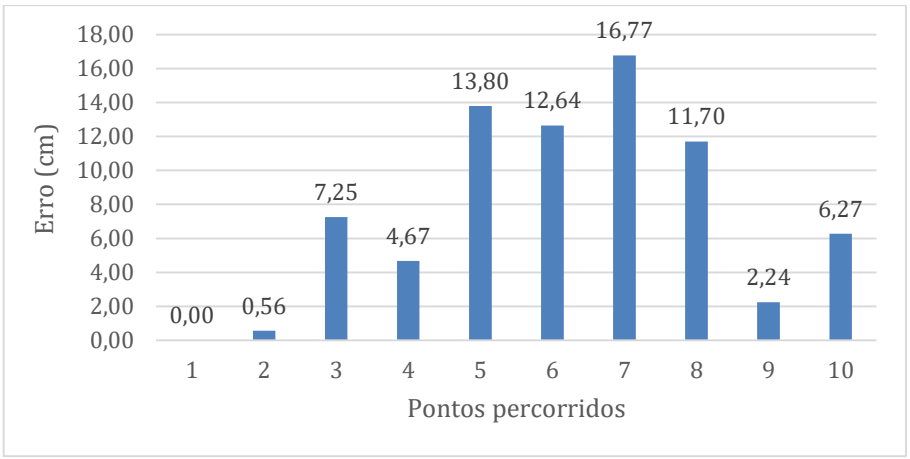
do obstáculo. A rede assumiu decisões tanto no sentido de seguir ao alvo quanto em virar à direita e esquerda. No gráfico da Figura 62, que traz a diferença entre as distâncias dos pontos da trajetória proposta e a real, onde o maior erro percebido foi de 16,7 cm no processo de desvio citado anteriormente e o erro final foi de aproximadamente 6,2 cm, dentro da tolerância estabelecida.

Figura 61 – Trajetória do teste realizado com um obstáculo



Fonte: Do autor, 2019

Figura 62 – Gráfico da distância entre os pontos da trajetória ideal e real para segundo teste.

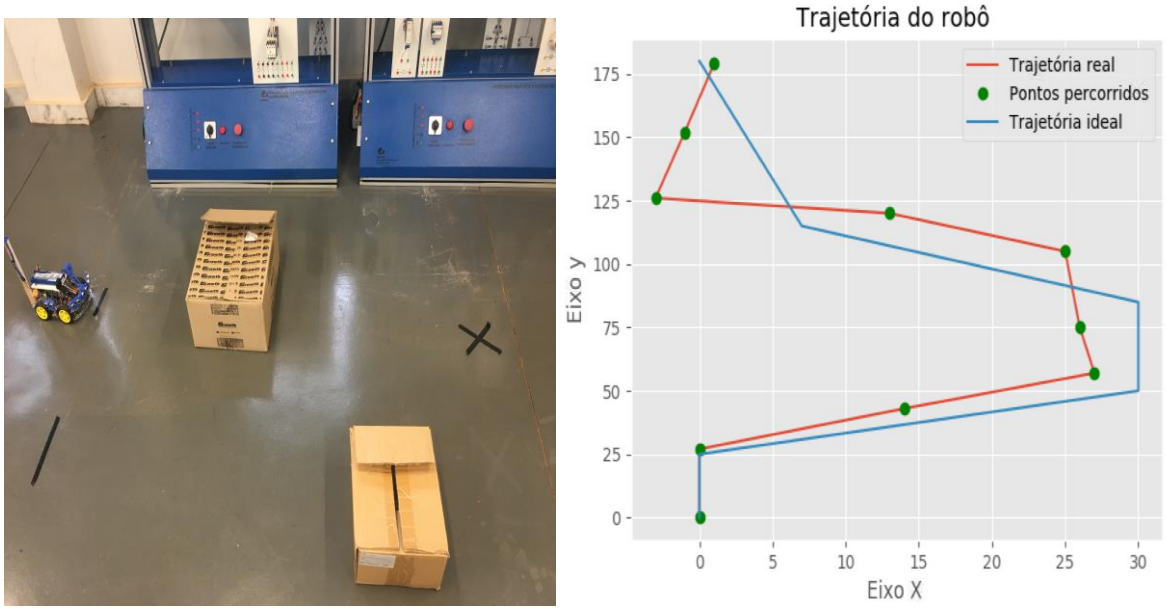


Fonte: Do autor, 2019

No terceiro teste uma caixa de dimensões 31 cm x 28 cm, foi adicionada no trajeto, de forma que a primeira caixa, a mesma usada no teste anterior, foi

posicionada a 60 cm do protótipo e a segundo caixa posicionada a 100 cm. A distância entre o ponto de origem e o ponto final é 180 cm sem ângulo de desvio. A Figura 63 (a) demonstra a configuração das caixas para o teste realizado no laboratório e em (b) o gráfico do trajeto percorrido. A figura 64 exibe os resultados das distâncias dos pontos percorridos entre a trajetória real e a esperada.

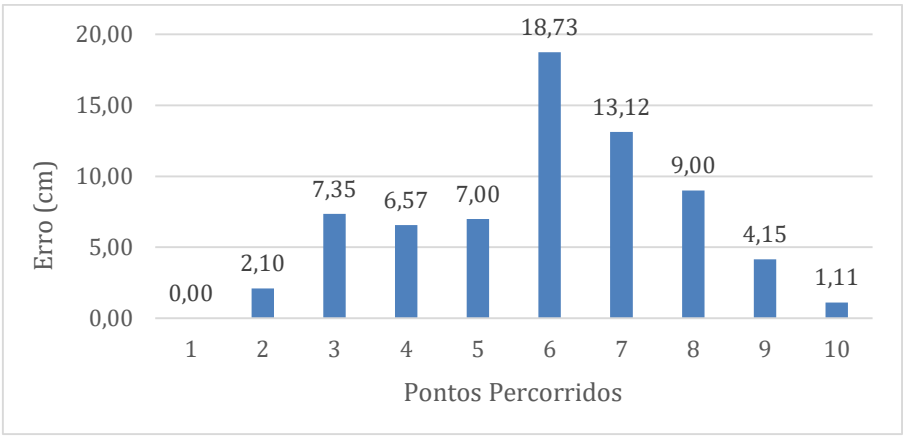
Figura 63 – Terceiro teste realizado com dois obstáculos: (a) Configuração dos obstáculo (b) Gráfico da trajetória percorrida



(a) (b)

Fonte: Do autor, 2019

Figura 64 – Gráfico da distância entre os pontos da trajetória ideal e real para o terceiro teste.

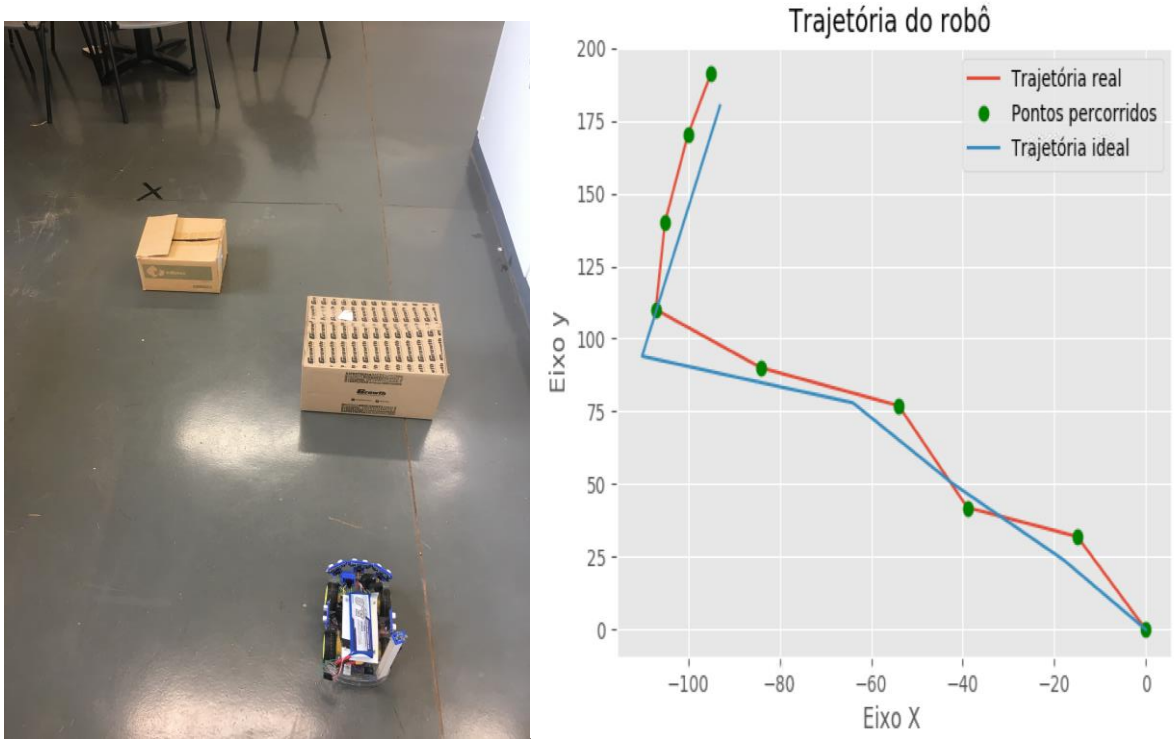


Fonte: Do autor, 2019

A trajetória descrita pelo robô foi, de certa forma, similar ao trajeto calculado até alcançar o ponto de coordenadas (13,123) onde o protótipo não poderia ir diretamente em sentido ao objetivo, pois havia um obstáculo a direita, desse modo, a distância percorrida aumentou e houve um alongamento do trajeto no sentido negativo do eixo das abscissas. Com isso observou-se o maior erro do trajeto no ponto 6 com quase 19 cm. Porém, como esperado, nos próximos pontos a trajetória foi corrigida no sentido da direção ao ponto objetivo, chegando ao menor erro de chegada ao destino dos quatro testes, com valor próximo a 1,1 cm.

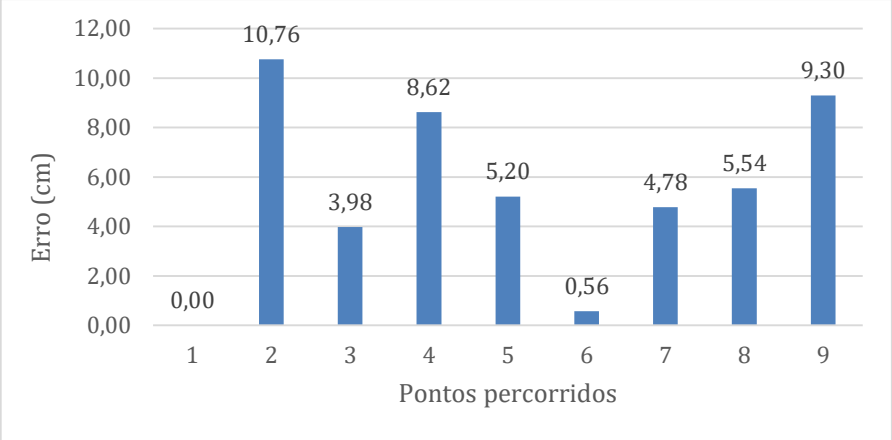
No quarto e último teste foram colocados dois obstáculos, em configuração diferente do teste anterior. Para isso foram utilizados os mesmos obstáculos, sendo o primeiro posicionado a 50 cm do protótipo e o segundo a 110 cm. A distância total entre a origem e o ponto final é 200 cm com desvio de - 26 graus. As Figuras 65 (a) e (b) trazem respectivamente, o posicionamento dos obstáculos utilizados no teste e a trajetória executada. Já a figura 66 demonstra os resultados das distâncias dos pontos percorridos entre a trajetória real e a esperada para este teste.

Figura 65 – Terceiro teste realizado com dois obstáculos: (a) Configuração dos obstáculo (b) Gráfico da trajetória percorrida



Fonte: Do autor, 2019

Figura 66 – Gráfico da distância entre os pontos da trajetória ideal e real para o quarto teste.



Fonte: Do autor, 2019

O último teste foi o que obteve uma melhor precisão, entre os testes com obstáculo, em comparação entre o trajeto real e o trajeto proposto, com o valor máximo de desvio de aproximadamente 10,8 cm. Porém uma das maiores diferenças dos pontos aconteceu na chegada ao destino final, que pode ter ocorrido por erros na totalização do odômetro do sensor óptico. A diferença foi de aproximadamente 9,3 cm, o maior de todos os testes. Apesar disso, a diferença está dentro do critério de tolerância definido anteriormente e, no teste 4, foi observado que as ações tomadas pelo robô foram bem próximas das esperadas para o trajeto.

4.3 CUSTOS DO PROJETO

Nesta parte dos resultados será apresentado os gastos totais do projeto e uma comparação com alguns equipamentos disponíveis no mercado. A maioria dos equipamentos foram adquiridos online e alguns até foram importados por falta de disponibilidade nacional. A Tabela 8 mostra o quantitativo dos equipamentos utilizados e o preço unitário e o preço total do projeto.

Tabela 8 – Custos para montagem do protótipo

Equipamento	Quantidade	Preço unitário	Preço total
Raspberry Pi 3 modelo b +	1	R$ 229,00	R$ 229,00
Cartão de memória 32 gb	1	R$ 34,99	R$ 34,99
Conversor Buck	2	R$ 22,00	R$ 44,00
Bateria LiPo 7,4 V 5000 mAh	1	R$ 150,00	R$ 150,00
Sensor Ultrassônico HC Sr04	5	R$ 15,00	R$ 75,00
Camera Raspberry pi 5 mp	1	R$ 50,00	R$ 50,00
Sensor encoder óptico	2	R$ 13,00	R$ 26,00
Sensor térmico AMG 8833	1	R$ 205,00	R$ 205,00
Chassi de acrílico com 4 rodas e 4 motores	1	R$ 115,00	R$ 115,00
Parafusos e fios	1	R$ 30,00	R$ 30,00
Conversor de nível lógico bidirecional	2	R$ 7,90	R$ 15,80
Mini driver Ponte H L298n	2	R$ 21,50	R$ 43,00
Bússola eletrônica HMC 5883L	1	R$ 33,90	R$ 33,90
	Preço total:		**R$ 1.131,69**

Fonte: Do autor, 2019

Para um robô com navegação autônoma equipado com câmera térmica o preço foi muito abaixo do praticado no mercado. Foram utilizados sensores com custo reduzido, mas que garantiram eficiência na aquisição do sinal. Para efeito de comparação, os gastos com a câmera térmica chegaram a R$ 255,00, preço combinado do sensor AMG 8833 e a câmera térmica. Uma câmera compatível com o projeto seria a FLIR Modelo One, que possui um custo médio de R$ 1899,00 no mercado brasileiro, ou seja, mais de sete vezes maior que o gasto no projeto. Deve-se levar em conta que a resolução da câmera FLIR é maior que a câmera usada no projeto, porém o sensor térmico apresentou um erro de no máximo 3,5 °C, sendo possível utilizá-lo para diversas aplicações dentro da faixa de medida adequada.

Outro ponto foi o custo do protótipo. Existem muitos robôs autônomos sendo desenvolvidos com diferentes metodologias de navegação, mas ainda não há uma grande gama de produtos comerciais nessa área. Um robô autônomo comercial que vem se popularizando é o robô aspirador de pó inteligente ROOMBA modelo 675 com valor de R$ 2036,99 no site oficial da marca. (IROBOT, 2019). O robô explora o ambiente de forma totalmente autônoma. Para comparação com outro protótipo de exploração de incêndios, a literatura traz o trabalho de Chowdhury et al. (2012), que foca no desenvolvimento de um protótipo explorador de incêndios de baixo custo,

com funções autônomas, sem mencionar valor quantitativo de componentes, onde utilizou-se sensores de baixo custo, alguns similares aos do atual trabalho, obtendo uma boa economia na construção do protótipo. Porém o trabalho de Chowdhury não conta com nenhum sistema de transmissão e processamento de imagens, uma ferramenta importante, onde foi observado a maior redução de custo deste trabalho.

5 CONCLUSÃO

Os resultados obtidos vieram ao encontro do proposto no objetivo principal, sendo assim, ao final do projeto, foi possível desenvolver um robô para identificação de zonas térmicas utilizando algoritmo Bug e redes neurais como base para navegação autônoma. Além disso a navegação manual foi oferecida como uma opção para o usuário com o controle realizado através do aplicativo web.

A câmera térmica adaptada para a aplicação, desenvolvida através da justaposição entre uma câmera e um sensor matricial, demonstrou boa eficiência para medir temperaturas dentro da escala do sensor, e o mapa de cores representou o gradiente de temperatura real observado em objetos com temperaturas elevadas.

O algoritmo proposto para navegação autônoma conseguiu chegar ao destino estabelecido nos diferentes testes com uma pequena margem de erro, menos de 5% da distância total, e os desvios aos obstáculos foram realizados sem haver nenhuma colisão com o protótipo. Essas tomadas de decisões corretas foram possíveis devido ao alto grau de acurácia da rede, e também ao fato de a base de dados utilizada no treinamento ser coletada em diferentes situações ambientes e disposição de obstáculos. Os fragmentos do algoritmo responsáveis pelo cálculo e correção da rota foram baseados inteiramente na aquisição dos dados da bússola e sensor encoder óptico, permitindo tanto a navegação no interior (*indoor*) quanto no exterior (*outdoor*) de salas.

A grande vantagem do protótipo em relação aos custos apresentados nos resultados foi o desenvolvimento da visão térmica. Apesar de não produzir uma imagem de alta resolução como outras câmeras comerciais, seu baixo custo viabiliza o uso em testes destrutivos, onde devido a situações ambientes extremas, a recuperação do robô seja incerta.

Durante testes e ao longo do projeto, surgiram alguns problemas e limitações associada a alguns instrumentos e equipamentos. A bússola digital sofre interferência em campos magnéticos fortes, inviabilizando a navegação autônoma em presença de tais campos. Outro ponto importante observado em testes, é possuir pneus de diferentes formatos e materiais disponíveis, pois dependendo do tipo de chão, o atrito pode ser muito pequeno fazendo com que a roda deslize e gere erros no cálculo da distância da trajetória.

O protótipo cumpriu seus objetivos, conforme mostrado nos resultados e metodologia. Porém outros trabalhos podem derivar-se a partir dessa pesquisa. Um trabalho futuro proposto seria a navegação utilizando o sensor LIDAR (Light Detection and Rangig), que emite feixes de laser infravermelho para mapear a superfície do ambiente tridimensionalmente. Utilizando essa ferramenta em conjunto com o *software* ROS (Robot Operating System) é possível trabalhar com algoritmos complexos e com maior precisão de localização comparado com o sensor ultrassônico. Entre esses algoritmos propõe-se trabalhar com o aprendizado por reforço, onde um agente aprende por tentativa e erro a melhor solução, juntamente com redes neurais Perceptron multicamadas, para solução dos problemas de navegação.

REFERÊNCIAS

ADAFRUIT. **Raspberry Pi Camera.** Disponível em: < https://www.adafruit.com/product/1367>. Acesso em: 15 Jan 2019.

ALVES, F J. **Introdução à linguagem de programação Python**. 1. Ed. Rio de Janeiro: Editora Ciência Moderna Ltda, 2013. 104p.

ALVES, M. F. **Funções de ativação**. Disponível em:< https://matheusfacure.github.io/2017/07/12/activ-func/>. Acesso em: 18 Set. 2019.

ARAÚJO, A.S. ; LIBRANTZ, A. F. H. Visão e inteligência computacionais aplicadas a Navegação autonôma de robôs. **Exacta**, São Paulo, v. 4 , p. 343-352. 2006.

ARDUINO e CIA. **Como medir a rotação de um motor utilizando o sensor LM393**. 2016. Disponível em: <https://www.arduinoecia.com.br/sensor-de-velocidade-lm393-arduino/>. Acesso em: 23 Set. 2019.

ASTROM, j. ; MURRAY, R**. Feedback systems: an introduction for scientists and engineers**. 1. Ed. Princeton: Princeton University Press, 2008. 408p.

BISCARO, A. P. **Redes Neurais Artificiais:** Perceptron Multicamadas. 2017. Disponível em :< http://sinop.unemat.br/site_antigo/prof/foto_p_downloads/fot _15225aula_08-pebceptbons_multicamadas_pdf_Aula_08-Perceptrons _Multicamadas.pdf>. Acesso em: 13 Fev. 2019.

BRADSKI, G.; KHAELER, A. **Learning OpenCV**: computer vision with the opencv library. 1. Ed. Editora O'Reilly Media, 2008. 508p.

CARRARA, V. **Introdução à robótica industrial**. 2015. Disponível em: <https://www.researchgate.net/profile/Valdemir_Carrara/publication/282651875_Introducao_a_robotica_industrial/links/5615ebed08ae4ce3cc657269.pdf>. Acesso em: 13 Mai. 2019.

CARUSO , M. J. Applications of Magnetoresistive Sensors in Navigation Systems. **PROGRESS IN TECHNOLOGY**, v. 72, p. 159-168, 1998.

CHOWDBURY, S. et al. A Design of a cost effective Fire Fighting Robot using Intelligent System. In: **The Seventeenth International Symposium on Artificial Life and Robotics 2012 (AROB)**. 2012. p. 1197-1200.

D-LINK. **Manual do usuário**. 2017. Disponível em:< https://www.dlink.com.br/wp-content/uploads/2018/08/DIR-615_T3_Manual_v1.10DI.pdf>. Acesso em: 15 jul. 2019.

DE ALBUQUERQUE, M. P.; DE ALBUQUERQUE, M. P. **Processamento de imagens:** métodos e análises. Centro brasileiro de pesquisas físicas, Rio de Janeiro, 2000.

DWORAKOWSKI et al. Uso da plataforma Arduino e do software PLX-DAQ para construção de gráficos de movimento em tempo real. **Revista Brasileira de Ensino de Físic**a, v. 38, n. 3, 2016.

ESQUEF, I. A. ; ALBUQUERQUE, M. P. ; ALBUQUERQUE, M. P. **Processamento digital de imagens**. Centro brasileiro de pesquisas físicas, Rio de Janeiro, 2003.

FISHER et al. Connected Components Labeling. 2003. Disponível em: <https://homepages.inf.ed.ac.uk/rbf/HIPR2/label.htm>.Acesso em: 23 Mai. 2019.

FILHO, O.M. ; NETO,V. H. **Processamento Digital de Imagens**. 1 ed. Rio de Janeiro: Brasport, 1999. 331p.

GÉRON, A. **Hands-on machine learning with Scikit-Learn and TensorFlow:** Concepts, tools, and techniques to build intelligent systems.1. ed. O'Reilly Media, 2017. 566p.

HART, D. W. **Eletrônica de Potência**: análise e projetos de circuitos. 1. ed. McGraw Hill Brasil, 2011. 504p.

HAYKIN, S. **Redes Neurais**: Princípios e prática. 2. ed. Hamilton: Bookman, 2001. 900p.

HRISKO, J. **Heat mapping with a 64-pixel infrared detector (amg8833), Raspberry Pi, and Python**. 2018. Disponível em: < https://makersportal.com/blog /2018/1/2 5/ heat -mapping-with-a-64-pixel-infrared-sensor-and-raspberry-pi>. Acesso em: 13 Jun. 2019.

IROBOT. **Robô aspirador Roomba 675**. 2019. Disponivel em:< https://www.irobotloja.com.br/roomba-675---robo-aspirador-de-po-inteligente-bivolt-r675400-p-/p>. Acessado em: 15 Out. de 2019.

LEÃO, A.; PETRUCCI G.; CARDOSO, M**. Detecção de bordas da imagem.** 2018. Disponível em : <https://fotodearte.com.br/ilab/tutoriais/deteccao-de-bordas/>. Acesso em: 13 Ago. 2019.

LIBERTI, N. **Eletrônica de potência**. Disponível em: file:///C:/Users/willi/Desktop/Aula%208B%20-%20Conversores%20Buck%20e%20Boost.pdf. Acesso em: 15 Ago. 2019

LIMA, D. **Utilização de redes neurais para análise de séries temporais de potência eólica**. 2015. 55f. Trabalho de conclusão de curso (Bacharelado em engenharia de computação) – Escola politécnica de Pernambuco, Universidade de Pernambuco, Pernambuco, 2015. Disponível em:< https://tcc.ecomp.poli.br/20152 /Danilo LIma_2015_TCC.pdf> . Acesso em: 08 jul. 2018.

LUO, C.; YANG, S. X. A bioinspired neural network for real-time concurrent map building and complete coverage robot navigation in unknown environments. **IEEE Transactions on Neural Networks**, v. 19, n. 7, p. 1279-1298, 2008.

MARROQUIN, A. et al. Design and implementation of explorer mobile robot controlled remotely using IoT technology. In: **Electrical, Electronics Engineering, Information and Communication Technologies (CHILECON), 2017 CHILEAN Conference on**. IEEE, 2017. p. 1-7.

MAZZA, L. **HTML5 e CSS3:** Domine a web do futuro. [S.l].Editora Casa do Código, 2014. 214 p.

MEDEIROS, L. F. **Redes neurais em delphi.** Florianópolis: Visual Books Editora, 2003. 115p.

MICROSOFT WINDOWS IOT CORE . **Raspberry Pi 2 and 3 Pin Mappings**. 2017. Disponível em : < https://docs.microsoft.com/en-us/windows/iot-core/learn-about-hardware/pin mappings/pinmappingsrpi>. Acesso em : 18 Jan. 2019.

MILLER, D. **Adafruit AMG8833 8x8 Thermal Camera sensor.** Disponível em: < https: //cdn-learn.adafruit.com/downloads/pdf/adafruit-amg8833-8x8-thermal-camera-sensor .pdf?timestamp=1549462805.> Acesso em: 23 Dez. 2018.

MORAES, R**. PWM no Raspberry Pi**. 2018**.** Disponível em:<http://blog.baudaeletronica.com.br/pwm-no-raspberry-pi/>. Acesso em: 23 jun. 2019.

NARANJO, J. L**. Inteligência computacional aplicada na geração de respostas impulsivas bi-auriculares e em aurilização de salas.** 2014. Disponível em: < https://www.researchgate.net/figure/Figura-20-Principais-funcoes-de-ativacao_ fig19_300015903 >. Acesso em: 13 Nov. 2019.

NASA. **Curiosity Rover**. 2011. Disponível em: <https://www.nasa.gov/mission_ pages/msl/i ndex.html.> Acesso em: 15 Dez. 2018.

OH, J. S. et al. Complete coverage navigation of cleaning robots using triangular-cell-based map. **IEEE Transactions on Industrial Electronics**, v. 51, n. 3, p. 718-726, 2004.

OPENCV. **AI Courses by OpenCV**. 2019. Disponível em: < https://opencv.org/ courses/>. Acesso em: 12 Abr. 2019.

ORTIZ, J. E.. Visual servoing for an omnidirectional mobile robot using the neural network-Multilayer perceptron. In: **Engineering Applications (WEA), 2012 Workshop on**. IEEE, 2012. p. 1-6.

PANASONIC. **Infrared Array Sensor Grid-EYE (AMG88).** 2019. Disponível em:< https://industrial.panasonic.com/cdbs/www-data/pdf/ADI8000/ADI8000C66.pdf>. Acesso em: 12 Out. 2019.

PEREZ, A. L. F. et al. **Desenvolvimento de um robô explorador baseado em uma arqutetura híbrida de hardware**. 2013. Disponível em: < http://www.natalnet .br/lars2013/WGWR-WUWR/121553.pdf > . Acesso em: 12 mai. 2018.

POMILIO, J. A. **Eletrônica de Potência para Geração, Transmissão e Distribuição de Energia Elétrica.** 2013. Disponível em : < http://www.dsce.fee. unicamp.br/~antenor/pdffiles /it744/cap3.pdf>. Acesso em: 15 Jan. 2019.

PYGAME. **About Pygame**. [20-]. Disponível em: < https://www.pygame.o rg/wiki/about>. Acesso em: 12 Jun 2019.

QUINTIA, P. et al. Simultaneous learning of perception and action in mobile robots. **Robotics and autonomous systems**, v. 58, n. 12, p. 1306-1315, 2010

RASPBERRY PI FOUNDATION. **Raspberry Pi model 3 B+**. [201-]. Disponível em: < https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/>. Acesso em : 30 Set. 2018.

REPOSITÓRIO DA AUTOMAÇÃO. **Motor CC com ponte H.** 2014. Disponível em: <https://au tomacaoifrsrg.wordpress.com/2014/09/20/motor-cc-com-ponte-h/>. Acesso em: 27 Ago. 2018.

ROMANO, Vitor Ferreira. Robótica industrial: aplicação na indústria de manufatura e de processos. São Paulo: Edgard Blücher, 2002.

RUIZ, M. **Anatomia dos Neurônios**. 2007. Disponível em: < https://pt.wikipedia. org/wiki/Ficheiro:Complete_neuron_cell_diagram_pt.svg>. Acesso em: 13 Jul. 2019.

SARAVANA ELECTRONICS. **Level shifter bidirectional 3 – 5V.** Disponível em: http://ww w.alselectro.com/level-shifter-3v-5v-bidirectional.html>. Acesso em: 12 Out. 2019.

SPACAGNA, G. **Anomaly Detection using deep auto-encoding**. 2017. Disponível em:< https://pt.slideshare.net/ds_mi/anomaly-detection-using-deep-autoencoders-gianmario-spacagna>. Acesso em: 16 Out. 2018.

SILVA, I. N. et al. **Redes neurais artificiais para engenharia e ciências aplicadas: curso prático**. 1 ed. São Paulo: Editora Artliber, 2010. 399p.

SINGH, M. K.; PARHI, D. R. Path optimisation of a mobile robot using an artificial neural network controller. **International Journal of Systems Science**, v. 42, n. 1, p. 107-120, 2011.

SILVA, A. M. **Curso Processamento digital de imagens de satélite.** Centro de Eventos da PUCRS - de 07 a 12 de outubro de 2001. Porto Alegre - RS. Disponível em: <http://www.cartografia.org.br/ >. Acesso em: 19 jul. 2018.

SHIMANO et al. Influência do percentual de preenchimento no comportamento mecânico de peças em PLA e ABS obtidas por Impressão 3D por extrusão. **Revista Brasileira de Ciência, Tecnologia e Inovação**, v. 3, n. 2, p. 178-190, 2018.

SONY. **Aibo**. 2017. Disponível em: < https://us.aibo.com/>. Acesso em: 13 Jan. 2019.

SPARKFUN. **Ultrasonic Ranging Module HC - SR04**. Disponível em : <https://cdn. sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf>. Acesso em: 12 Jan. 2019.

THOMSEN, A. **Motores CC com driver ponte H L298**. 2013. Disponível em:< https://w ww.filipeflop.com/blog/motor-dc-arduino-ponte-h-l298n/>. Acesso em : 15 Abr. 2019.

VIAN, H. et al. **Modelagem de ambientes com robô móvel baseado em Rede Neural Hierárquica e visão omnidirecional**. Disponível em: http://www.lbd.dcc.ufmg.br/ colecoes/wvc/2007/0012.pdf>. Acesso em: 03 mai. 2018.

VIEIRA, C. A. O. **Accuracy Of Remotely Sensing Classification Of Agricultural Crops**: **A Comparative Study**. 2000. 396 f. Tese (Doutorado em filosofia)- University of Nottinghann, Notthinghan, Reino Unido. 2000.

WANG, Feng et al. Additive margin softmax for face verification**. IEEE Signal Pro cessing Letters**, v. 25, n. 7, p. 926-930, 2018.

WEBIOPI. **Light control with auto on/off**. [201-]. Disponível em : <https://webiopi.tr ouch.com/Tutorial_Basis.html>. Acesso em : 15 Fev. 2019.

YAROTSKY, D. Error bounds for approximations with deep ReLU networks. **Neural Networks**, v. 94, p. 103-114, 2017.

YUFKA, A.; PARLAKTUNA, O. Performance comparison of the BUG's algorithms for mobile robots. In: **International symp. on Innovations in intelligent Systems and applications: INISTA**. 2009. p. 416-421.

ZAMBIASI, S. P. **Redes Neurais.** Disponível em:< https://www.gsigma.ufsc.br/~popo v/aulas/rna/n euronio_artificial/index.html>. Acesso em : 26 jul. 2018.

ZOHAIB, M. et al. IBA: Intelligent Bug Algorithm - A novel strategy to navigate mobile robots autonomously. In: **International Multi Topic Conference**. Springer, Cham, 2013. p. 291-299.