UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
PRÓ-REITORIA DE GRADUAÇÃO
CENTRO DE CIÊNCIAS EXATAS E NATURAIS
CURSO DE CIÊNCIA DA COMPUTAÇÃO

FELIPE CÉSAR PINHEIRO LEÃO

CONTROLE DE UM ROBÔ EM UM AMBIENTE MONITORADO PARA RESOLUÇÃO DE LABIRINTOS

MOSSORÓ
2017

FELIPE CÉSAR PINHEIRO LEÃO

CONTROLE DE UM ROBÔ EM UM AMBIENTE MONITORADO PARA RESOLUÇÃO DE LABIRINTOS

Monografia apresentada a Universidade Federal Rural do Semi-Árido como requisito para obtenção do título de Bacharel em ciência da computação.

Orientador: Prof. Dr. Marcelo Roberto Bastos Guerra Vale

MOSSORÓ

2017

L433c Leão, Felipe Cesar Pinheiro.
Controle de um robô em um ambiente monitorado para resolução de labirintos. / Felipe Cesar Pinheiro Leão. - 2017.
49 f. : il.

Orientador: Marcelo Roberto Bastos Guerra Vale.
Monografia (graduação) - Universidade Federal Rural do Semi-árido, Curso de Ciência da Computação, 2017.

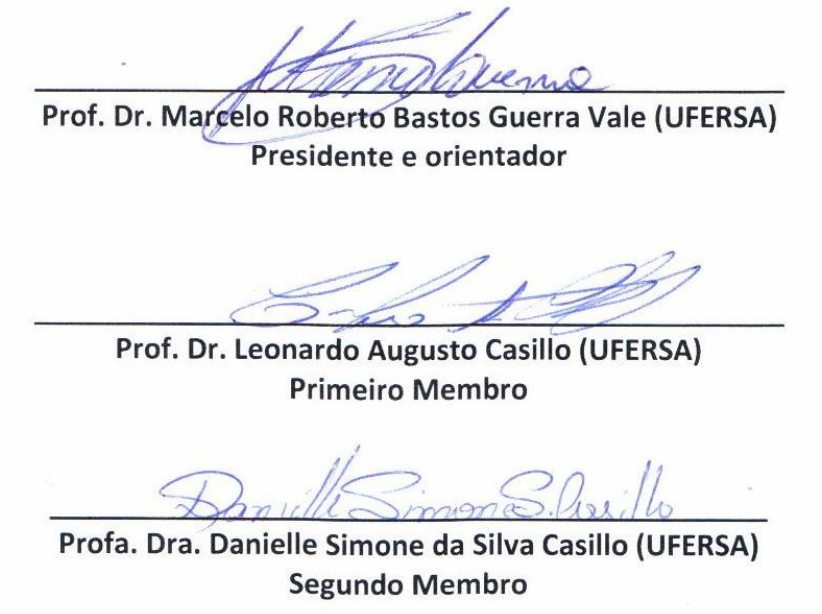
1. AGV. 2. Teoria dos Grafos. 3. Segmentação de Imagens. 4. Arduino. 5. Automação. I. Vale, Marcelo Roberto Bastos Guerra, orient. II. Título.

CONTROLE DE UM ROBÔ EM UM AMBIENTE MONITORADO PARA RESOLUÇÃO DE LABIRINTOS

Monografia apresentada a Universidade Federal Rural do Semi-Árido como requisito para obtenção do título de Bacharel em ciência da computação.

Defendida em: 22 /05/ 2017.

BANCA EXAMINADORA



Prof. Dr. Marcelo Roberto Bastos Guerra Vale (UFERSA)
Presidente e orientador

Prof. Dr. Leonardo Augusto Casillo (UFERSA)
Primeiro Membro

Profa. Dra. Danielle Simone da Silva Casillo (UFERSA)
Segundo Membro

AGRADECIMENTOS

Primeiramente desejo agradecer a toda minha família por todo apoio e suporte nessa minha gloriosa batalha, que a foi a graduação em ciência da computação – UFERSA. Meus amigos de longa data também merecem todos os meus agradecimentos por estarem ao meu lado sempre que eu precisei me ajudando de diversas formas possíveis, sendo uma saída para espairecer, uma carona para universidade ou uma palavra de apoio. Os novos amigos que fiz nessa jornada também merecem destaque, pois eles sabem de tudo que nós passamos por aqui e o tanto que lutamos para conseguir o tão sonhado título de bacharel. Minha namorada merece menção honrosa por sempre me apoiar e me dá os melhores conselhos. Por fim, quero dedicar toda essa minha jornada ao meu irmão que é a joia mais preciosa em minha vida.

RESUMO

Esse projeto tem como proposta apresentar estudos, conceitos e técnicas de automação, robótica, teoria dos grafos e processamento digital de imagens para elaboração de um Veículo Guiado Automaticamente (AGV) capaz de encontrar o menor caminho em um labirinto. Essa locomoção é realizada por um robô que recebe instruções que designa a rota a ser seguida. Essas instruções advêm do uso de algoritmos de grafos aliado a técnicas de segmentação de imagens. No desenvolvimento da aplicação é utilizada a plataforma de prototipagem Arduino e a ferramex'nta de programação MATLAB para a implementação dos algoritmos usados. O projeto desenvolvido mescla conceitos de um robô seguidor de linhas com detecção de objetos em uma cena através do uso de uma câmera posicionada na parte superior do labirinto.

Palavras-chave: AGV. Teoria do grafos. Segmentação de imagens. Arduino. Automação.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1. INTRODUÇÃO

A automação industrial e residencial está em crescimento, com aumento na projeção de 12% entre 2014 e 2016 (BITMAG, 2017). Junto com esse crescimento vêm à baixa no custo dos projetos em diversos setores, entre eles o sistema de Veículos Guiados Automaticamente (*Automatic Guided Vehicle* – AGV). O AGV consiste em um veiculo com capacidade de transitar em um ambiente através de sensores, bastante utilizado na indústria para transporte de cargas, em locais inóspitos que trazem riscos para o ser humano, e até mesmo em hospitais e supermercados (NOGUEIRA, TEIXEIRA e RANGEL, 2015).

Em um âmbito acadêmico, trabalhar com robótica acarretava em dificuldades devido custo envolvido na compra dos componentes, com o passar do tempo esses desafios foram diminuindo com o surgimento de novas tecnologias, como Lego Mindstorns da empresa Lego®. O Arduino® é uma plataforma para prototipagem de computação física ou embarcada de baixo custo onde é possível interagir com o ambiente com uso de *software* e hardware. Por exemplo, medindo a temperatura ou umidade de um ambiente.

Alguns dos desafios encontrados na área da robótica é a construção de sistemas capazes de analisar imagens e encontrar informações relevantes, de forma ágil e pertinente. Um sistema de visão computacional deve ser capaz de coletar informações importantes em uma cena a partir de uma imagem e desconsiderar informações irrelevantes (NEU, 2014). A utilização de Processamento Digital de Imagens (PDI) junto à robótica tem elevado potencial, como por exemplo, o uso para inspeção visual de forma on-time e repetitiva para guiar robôs em suas tarefas (GONÇALVES, RODRIGUES, *et al.*, 2014).

Além das técnicas de PDI e o uso de robóticas, também se mostra importante estruturar os dados de forma eficiente para um melhor desempenho. Para essa estruturação, algoritmos de teoria dos grafos pode ser a solução. De acordo com Santos (2013), a teoria dos grafos é uma ferramenta fundamental que pode ser usada em diversas área como matemática, engenharia, química, entre outras. Ainda segundo Santos (2013) é possível a resolução de problemas práticos como a representação de mapas de estradas ou encontrar o menor caminho entre dois pontos de um labirinto. Esse último ponto merece destaque, pois a resolução de um labirinto é um dos problemas abordados nesse trabalho.

1.1 Objetivo Geral

Neste trabalho são estudados e apresentados os conceitos e as técnicas necessárias para elaborar um AGV capaz de encontrar o menor caminho em um labirinto. Técnicas de PDI e algoritmos de teoria dos grafos são utilizados para alcançar o objetivo do trabalho, juntamente com a ferramenta MATLAB®.

1.2 Objetivos Específicos

Os objetivos específicos deste trabalho de pesquisa foram definidos e divididos conforme os tópicos abaixo:

- Implementar algoritmos de teoria dos grafos para resolução de labirintos;
- Desenvolver um robô usando Arduino®;
- Criar um sistema de visão computacional para guiar o robô em um labirinto.

1.3 Estrutura do Texto

O capítulo 2 aborda áreas e trabalhos relacionados com o objetivo desse projeto. O capítulo 3 explana os conceitos teóricos necessários para um melhor entendimento desse trabalho, juntamente com sua implementação. O capítulo 4 demonstra os resultados obtidos no desenvolvimento desse trabalho. Por fim, o capítulo 5 tem as conclusões, as dificuldades encontradas e os trabalhos futuros.

2. REVISÃO DA LITERATURA

2.1 Automação

A automação pode ser definida como um conjunto de técnicas destinada a realizar tarefas automaticamente, substituindo a energia humana por energia eletromecânica controlada por computadores (SILVEIRA e LIMA, 2003). O termo automação foi criado em 1940 pelo engenheiro da Ford Motor Company. Nessa época os dispositivos usados na industria eram eletromecânicos, com sua parte lógica controlada atráves de relés e temporizadores intervalados, utilizando intervensão humana na tomada de decisões (LAMB, 2015).

Com o avanço dos computadores e microcontroladores as tomadas de decisões foram cada vez mais recaindo sobre as máquinas, e para criar a interface para controle dessas máquinas surgiram os CLP's (*Programmable Logic Controller)*, que são computadores industriais conectados fisicamente aos instrumentos de campo (interruptores, sensores e etc), com programas desenvolvidos na linguagem Ladder, que é uma das linguagens utilizadas em CLP's.

A automação já é vista na indústria como um todo. Porém ela se expandiu para fora do ambiente industrial chegando as residênciais, hospitais, mercados e etc. O termo domótica surgiu da palavra latina *domus*, que significa casa, junto com a palavra robótica. Esse termo é empregado para sinalizar a melhoria de vida, bem estar e redução de afazeres domésticos considerados repetitivos, através do uso de automação residencial (RAMOS e SANTOS, 2015).

2.2 Robótica

O estudo da robótica é a ciência que se ocupa do desenvolvimento e das aplicações com robôs. O escritor Isaac Asimov tornou popular o termo robô em sua obra publicada em 1950 intitulada de "Eu, robô". Ele define um robô como uma máquina de aparência humana sem sentimentos e guiada através de comandos programados por um ser humano (ROMANO e DUTRA, 2002). Asimov também criou três leis básicas para os robôs:

- 1ª Lei: Um robô não pode ferir um humano ou, por inação, permitir que um ser humano sofra algum mal;
- 2ª Lei: Um robô deve obedecer às ordens que lhe sejam dadas por seres humanos exceto nos casos em que tais ordens entrem em conflito com a Primeira Lei.

- 3ª Lei: Um robô deve proteger sua própria existência desde que tal proteção não entre em conflito com a primeira ou segunda lei.

A robótica é uma área multidisciplinar, unindo diversas áreas do conhecimento, como por exemplo, engenharia mecânica, engenharia elétrica, engenharia de automação, ciência da computação, entre outras. Segundo Nogueira, Teixeira e Rangel (2015) a utilização de robôs na industria é uma tendência, devido a capacidade das máquinas em realizar trabalhos com precisão e em locais que trariam riscos para o ser humano. Ainda de acordo com os mesmo autores, as máquinas vieram para subistiruir o trabalhador humano nas atividades produtivas dentro da indústria, já sendo utilizadas em diversas áreas, como na exploração espacial, subaquática e em busca e resgate.

A utilização da robótica vai muito além da área industrial, como por exemplo na educação. Zilli (2004) cita a robótica educacional como uma forma de ensino que permite ao professor ensinar ao aluno de forma prática e didática os conceitos que muitas vezes, são de difícil compreensão nos métodos tradicionais de ensino. Ainda segundo Zilli (2004), a robótica educacional motiva o aluno a construir o conhecimento através de observação própria, e dessa forma prática, o ensino tem melhor aproveitamento.

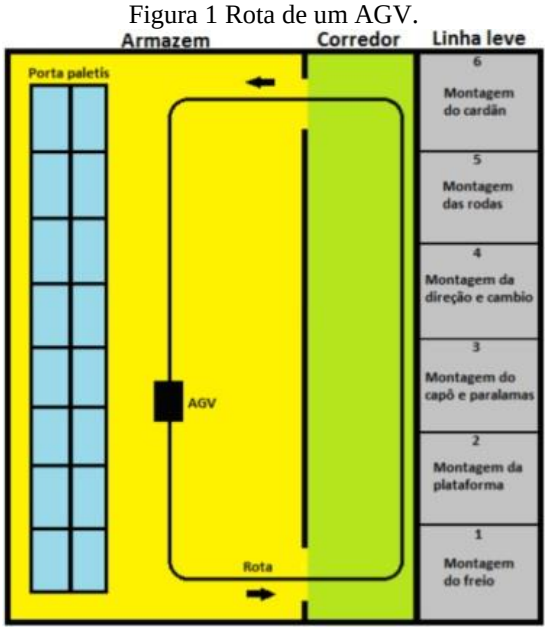
Em termos de robótica educacional, merece destaque a linguagem de programação Logo desenvolvida por Seymourt Papert e o MIT (Massachusetts Institute Technology) na década de 60. A Logo tinha o objetivo de ser uma linguagem de programação para crianças (CHIELLA, 2002). A Logo foi o ponta pé inicial para o que hoje é conhecido como os *kits educacionais de robótica*. Esses kits utilizam linguagens de programação própria ou linguagens já encontradas no mercado, e os kits são desenvolvidos por empresas como a Lego® ou Super Robby, que é uma empresa brasileira (ZILLI, 2004). Até o momento a automação e a robótica foram tratados de forma geral, nas seções 2.2.1, 2.2.2 e 2.2.3 serão abordados temas com maior relação ao projeto desenvolvido nesse trabalho.

2.2.1 Veículo Guiado Automaticamente

Um AGV é um veículo elétrico programável, que pode ser guiado através de trilhos, sensores e diversas outras formas. Com uma locomoção autônoma através do uso de baterias, podendo trabalhar de forma quase interrupta se comparado a um humano ou através de equipamento manuais que necessitam de intervenção humana (KIM, TANCHOCO e KOO, 1999). No estudo de caso de Souza e Royer (2013) sobre a implementação de um AGV em uma empresa do ramo agrícola, eles tiveram como objetivo apresentar as melhorias

proporcionadas aos processos logísticos da empresa, que antes era composta basicamente por empilhadeiras. A primeira parte do estudo foi definir a rota a qual o AGV irá percorrer como mostra a Figura 1.

Figura 1 Rota de um AGV.



Fonte: De Souza e Royer, 2013.

A segunda etapa foi a montagem dos kits necessários para que o carrinho precisasse de apenas uma viagem para suprir todas a etapas dos processo. Após o kit estabelecido, a empresa adquiriu o AGV de uma empresa especializada e por fim definiram o circuito do AGV, marcando uma rota no piso da fábrica com painéis marcadores. Após a instalação do sistema notou-se um redução no número de acidentes e uma redução nos custos, como mostram a Tabela 1 e Tabela 2.

Tabela 1 Custos com o uso de empilhadeiras.

Custo com empilhadeiras	Considerações por ano	Total anual
Mão de obra	Duas pessoas	R$ 93.600,00
Locação	Dois equipamentos	R$ 43.200,00
Causa trabalhista	Três processos	R$ 20.000,00
Erro humano	Indisciplina e colisões	R$ 87.600,00
Combustível	Gasolina	R$ 7.200,00
		R$ 251.600,00

Fonte: De Souza e Royer, 2013.

Tabela 2 Custo com o uso do sistema AGV

Custo com AGV	Considerações por ano	Total anual
Equipamento	Leasing	R$ 199.200,00
Manutenção	Preventiva	R$ 14.400,00
Combustível	Baterias	R$ 8.400,00
Infraestrutura	Instalação	R$ 20.000,00
		R$ 242.000,00

Fonte: De Souza e Royer, 2013.

As tabelas consideram apenas os ganhos econômicos, porém os ganhos relacionados à segurança e produtividade também devem ser considerados. As aplicações com robôs são as mais diversas, na seção 2.2.2 trará um trabalho de um robô com foco em competições.

2.2.2 Robô Seguidor de Linha

Um robô seguidor de linha é capaz de detectar uma linha desenhada no chão através do contraste entre a cor da linha com a do piso, e essa técnica é bastante utilizada em AGV's (GOMES *et al.*, 2015). A RoboCore (ROBOCORE, 2017) é uma empresa brasileira responsável pela competição de robôs autônomos seguidores de linha, onde o robô vencedor será o robô que conseguir vencer o percurso em menor tempo. Para estar apto para competir o robô precisa se enquadrar nas seguintes regas:

- Os robôs devem ser totalmente autônomos e com todos os componentes embarcados. Não pode ser controlado externamente por fio ou por rádio, com exceção para ser iniciado.
- Nenhuma adição, remoção ou alteração de hardware ou software poderá ser feita durante a tomada de tempo. Porém pequenos reparos serão permitidos.
- O Robô não pode exceder 250mm de comprimento, 250mm de largura e 200mm de altura, não podendo alterar suas dimensões durante a partida.
- O Robô não poderá possuir nenhum tipo de mecanismo de sucção para aumentar a força normal em relação ao solo.

Com essas regras em mente, Gomes et al (2015) utilizou um Arduino® para programar um robô seguidor de linha. A entrada dos dados é feita através dos sensores ligados a placa Arduino® e os motores são as saídas de dados. A partir da captura de dados pelos sensores, os motores poderão executar de forma correta as curvas e corrigir seu

posicionamento. Além dos robôs seguidores de linhas, outra linha de pesquisa onde também é possível encontrar competições é a de futebol de robôs, que será explanada no tópico 2.2.3.

2.2.3 Futebol de robôs

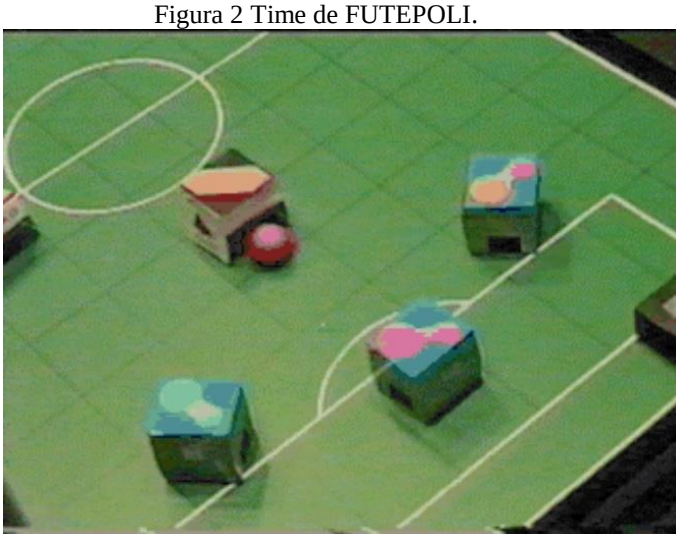
Como mostrado na seção 2.2.2, o robô seguidor de linha trabalha em um ambiente estático e sem cooperatividade, diferente dos robôs jogadores de futebol. Segundo Bianchi e Reali-Costa (2000), os robôs jogadores de futebol precisam realizar processos de reconhecimento visual em tempo real, trabalhar em um espaço dinâmico, rastrear objetos e acertar a bola de forma correta. Para atingir esses objetivos é necessário cooperatividade, eficiência, raciocínio e aprendizado. Ainda de acordo com Bianchi e Reali-Costa (2000), partidas de futebol entre robôs é uma atividade que possibilita experimentos na área de robótica, testando a inteligência artificial dos robôs e a cooperação entre eles.

Para a Copa Brasil de Futebol de Robôs, realizada na Escola Politécnica da Universidade de São Paulo, realizada em 1998, Bianchi e Reali-Costa (2000) desenvolveram o FUTEPOLI, um sistema de visão computacional, juntamente com os robôs para disputarem a copa. O sistema foi dividido em quatro partes, que são:

- Sistema de visão computacional: Responsável pela obtenção das imagens a partir de uma câmera disposta sobre o campo, a qual determina a posição e orientação dos robôs através de cores.
- Sistema tático: Irá utilizar as informações geradas pelo sistema de visão para definir as táticas de ataque e defesa, além de alimentar o módulo seguinte com as informações de velocidade e ângulo para os robôs.
- Sistema de Comando e Comunicação: Sistema de comunicação de rádio que atua entre os robôs e um computador. Também é responsável por calcular as velocidades do robô com as informações recebidas do módulo anterior.
- Sistema Embutido nos Robôs: Responsável por receber as informações de velocidade dos robôs e controlar os seus motores.

E essas informações irão trabalhar dentro de um loop para que o sistema funcione corretamente. A Figura 2 ilustra os robôs utilizados na competição.

Figura 2 Time de FUTEPOLI.



Fonte: Bianchi e Reali-Costa, 2000.

Outra competição de futebol de robôs é a *RoboCub Small Size Size (F180)* (ROBOCUP, 2017) que é disputada por países latino-americanos. A competição é constituída por dois times com seis robôs, cada equipe disputando uma partida de futebol. Com o intuito de competir Costa, Gomes e Bianchi (2003) desenvolveram um robô guiado através de visão local, ou seja, uma câmera é inserida no robô e é através dessa câmera que o robô será guiado. O sistema de visão local irá enviar as imagens para um microcomputador padrão IBM-PC, para que ele processe a imagem e retorne a posição do robô com relação à pista. A Figura 3 ilustra o mini robô desenvolvido pelos autores do trabalho.

Figura 3 Mini robô de visão local.

Fonte: Costa, Gomes e Bianchi, 2003.

Ainda segundo Costa, Gomes e Bianchi (2003) foram necessários criar três sistemas para o robô, que são: sistema visual, sistema de controle de direção e o sistema de controle de

velocidade. O sistema de visão é composto por uma câmera localizada na parte superior do robô responsável por detectar a pista e enviar para o microcomputador. O sistema de direção foi criado a partir de cinco regras, essas regras irão ditar a velocidade dos motores de passo presentes no robô, de acordo com o centro da pista e o centro da imagem. O sistema de velocidade utiliza um motor de passo híbrido, esse motor trabalha recebendo pulsos em uma determinada ordem para seu funcionamento.

2.3 AGV desenvolvido.

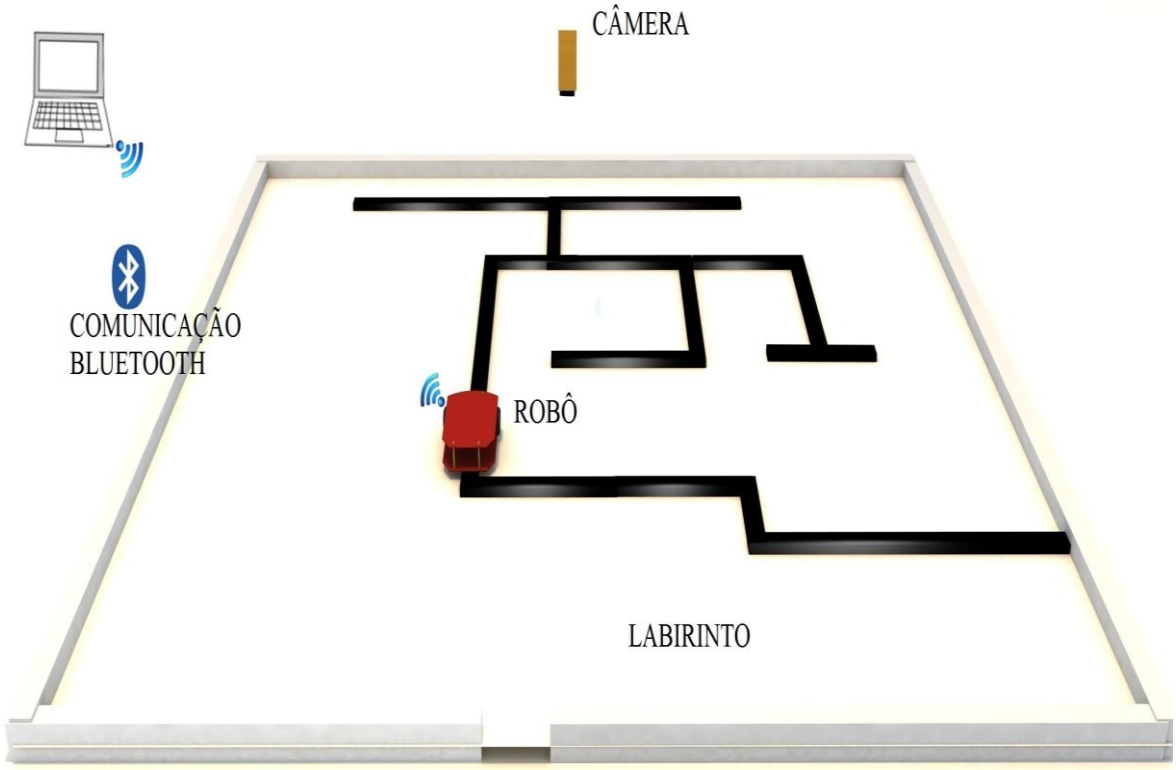
O projeto tem como objetivo desenvolver um AGV seguidor de linha, o qual utilizará técnicas de PDI para a detecção dos objetos e algoritmos de teoria dos grafos para a resolução de problemas, que no caso será a encontrar a saída de um labirinto pelo menor caminho possível. O objetivo dessa pesquisa é criar um AGV mesclando as metodologias usadas do robô seguidor de linha com o de futebol de robôs.

O robô seguidor de linha apresentado na seção 2.2.2 tem sua programação embarcada, sem o uso de recursos externos, diferentemente do robô jogador de futebol apresentado na seção 2.2.3, onde é utilizada uma câmera externa para detectar os robôs e os algoritmos estão contidos em um computador, que se comunicará com os robôs através de sinais de rádio. Mais detalhes da implementação são encontrados nos capítulos 3 e 4.

3. MATERIAIS E MÉTODOS

Nesse capítulo será demonstrado o desenvolvimento do projeto e todo o embasamento teórico para sua conclusão. As principais partes abordadas são: Robô, comunicação computador-robô, labirinto e aquisição de imagens, os quais podem sem vistos na Figura 4 e explanados em detalhes nas seções 3.1, 3.2 e 3.3.

Figura 4 Ilustração do projeto.



Fonte: Autoria própria.

3.1 Robô

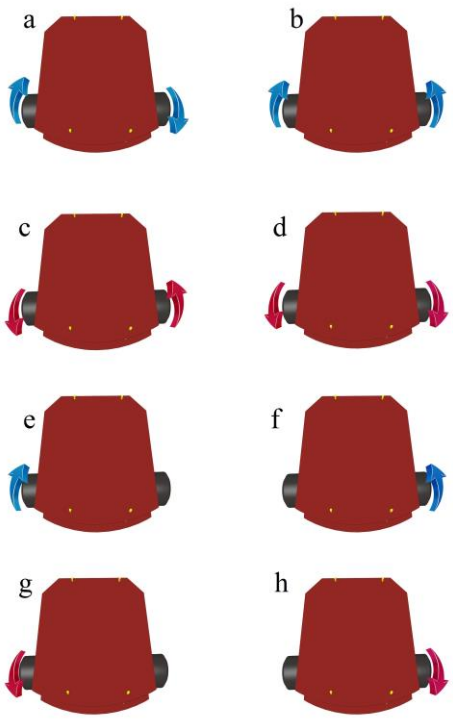
Um robô móvel é um dispositivo mecânico montado sobre uma base móvel, que é controlado por um sistema computacional, o qual irá interagir com o ambiente através de atuadores e sensores (PIERI, 2002). Os robôs móveis podem ser classificados de diversas formas, como por exemplo, a sua forma de controle, funcionalidade ou forma de locomoção. Quanto à locomoção podemos ainda dividir em aéreos, aquáticos e terrestres, podendo ainda dividir os terrestres nos que se movem através de rodas, esteiras ou pernas. Os mais populares ainda são os robôs terrestres, mesmo com o crescente uso dos drones, tanto para lazer quanto

para uso profissional. Esse projeto utiliza um robô com locomoção terrestre, por tanto só se aprofundará nesse tipo.

- Robô com rodas: São os robôs mais simples os quais não necessitam de um hardware complexo para sua execução. Principal desvantagem está na dificuldade de trabalhar em terrenos irregulares.
- Robô com esteiras: Mais sofisticados que os robôs com roda e bastante usados em terrenos irregulares, mas com um gasto maior de energia.
- Robô com pernas: Robô mais complexo que os anteriores, onde pode se trabalhar em terrenos que os anteriores teriam dificuldades. A desvantagem está justamente em sua complexidade.

O robô a ser utilizado nesse trabalho funciona como uma cadeira de rodas, onde a tração está em duas rodas na parte traseira do robô. Os giros são feitos usando apenas uma ou as duas rodas como mostra o esquema da Figura 5.

Figura 5 Esquema de movimentação do robô.



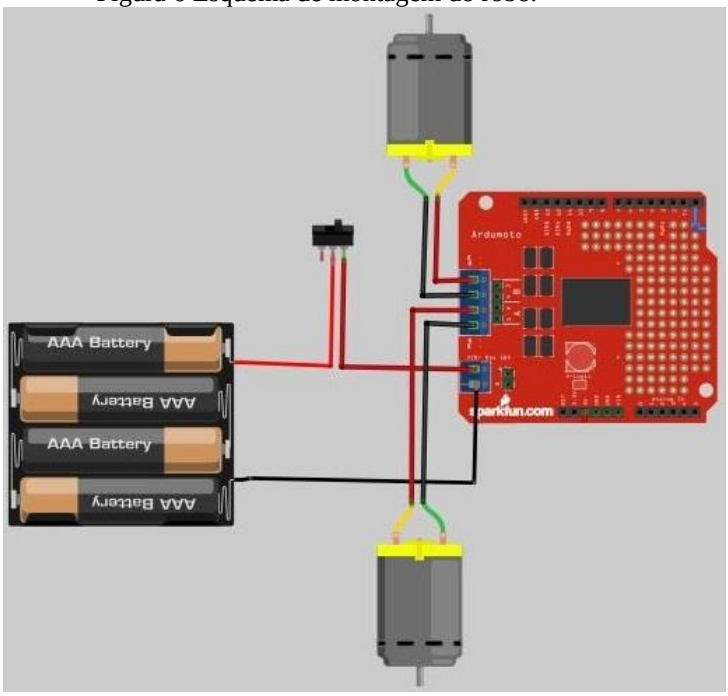
Fonte: Autoria própria.

Na Figura 5, é possível visualizar todas as movimentações possíveis do robô. A Figura 5a representa o movimento de giro para a esquerda, onde a roda esquerda deve se

movimentar para frente e a roda direita para trás. A Figura 5b mostra o movimento para frente, com as duas rodas girando para frente. A Figura 5c ilustra o movimento para direita, com a roda esquerda girando para trás e a direita para frente. A Figura 5d demonstra o movimento para trás, com as duas rodas girando para trás. A Figura 5e representa o movimento para esquerda com o giro para frente apenas da roda esquerda, enquanto a roda direita está parada. Já a Figura 5f é o movimento contrário do mostrado na Figura 5e, com a roda esquerda parada e a direita girando para frente, resultando em um movimento para direita. Por fim as figuras Figura 5g e Figura 5h ilustram o movimento para direita, com apenas o giro da esquerda para trás e o movimento para direita, com apenas a roda esquerda girando para trás, respectivamente.

A tração é controlada por dois servos motores DG1D-A130GEARMOTOR[1] e alimentada por um sistema de baterias de quatro pilhas AA. Para controle do robô foi utilizado um Arduino® Leonardo que será explicado na seção 3.1.2 e o esquema de montagem do robô pode ser visto na Figura 6.

Figura 6 Esquema de montagem do robô.



Fonte: Autoria própria.

[1] *Datasheet* disponível em < http://cdn.sparkfun.com/datasheets/Robotics/DG01D.pdf >. Acesso em jan. 2017

3.1.2 Microcontrolador Arduino

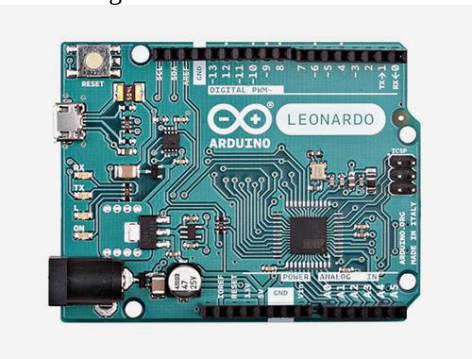
De forma geral o Arduino® é composto por um microprocessador Atmel AVR, um oscilador e um regulador de cinco Volts podendo ser conectado a *displays*, sensores, led (*light emitting diode*), módulos ethernet entre outros, abrindo um leque de possibilidades. Segundo Banzi (2011), a vantagem do Arduino® sobre outras plataformas está na sua facilidade de aprendizagem em um curto período de tempo, dando a possibilidade de o usuário criar seus próprios projetos.

A linguagem usada para programação das placas é a *Wiring* (WIRING, 2017) que foi desenvolvida em baseado no projeto de código aberto *Processing* (PROCESSING, 2017), que é essencialmente C/C++. Por se tratar de um projeto de código aberto a comunidade pode criar ou melhorar as funcionalidades da linguagem. Para auxiliar os programadores existe o *The Arduino Integreted Development Environment* ou *Arduino Software (IDE)*, é uma IDE responsável por upar os códigos (*Scratch)*, acompanhar a execução do código a partir de um monitor serial, entre outras funcionalidades. A programação para Arduino® está basicamente dividida em duas funções:

- *Setup*: Irá configurar as variáveis iniciais do programa e será executada apenas no início;
- *Loop*: Conterá a principal parte do programa, podendo ter instruções para tomada de decisões, laços de repetições, chamadas de funções, declarações de variáveis e etc. Essa função estará em execução enquanto a placa estiver recebendo energia.

Atualmente existem diversas placas Arduino® diferentes, sendo o Arduino® UNO o mais usado e o Arduino® Mega o com maior capacidade de processamento e armazenamento. Para esse projeto foi utilizado o Arduino® Leonardo por ele já atender todos os requisitos para a aplicação. A Figura 7 mostra a placa do Arduino® Leonardo.

Figura 7 Arduino Leonardo.



Fonte: Arduino[4].

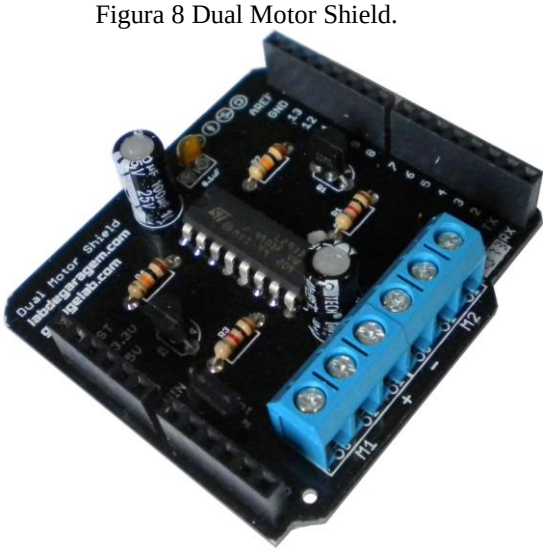
Além do Arduino Leonardo também é necessário o uso do *Dual Motor Shield*[6] (mostrado na Figura 8) e a biblioteca *DualMotor.h*[7], ambos desenvolvidos pelo Laboratório de Garagem (GARAGEM, 2017) para controlar os dois servos motores. A biblioteca *DualMotor.h* possui quatro funções para controle dos servos motores, que são:

- dualMotor.M1move(velocidade,sentido);
- dualMotor.M2move(velidade,sentido);
- dualMotor.M1parar();
- dualMotor.M2parar().

A velocidade do motor será definida com um valor entre 0-255 e o sentindo entre 0-1, onde 0 (zero) é sentido horário e 1 (um) o anti-horário. As portas digitais para controle dos motores são as cinco, seis, sete e oito, as quais a biblioteca é responsável pelo controle.

[4] Disponível em: < https://www.arduino.cc/en/Main/ArduinoBoardLeonardo >. Acesso em Fev, 2017
[6] Tutorial disponível em: < http://labdegaragem.com/profiles/blog/show?id=6223006%3ABlogPost%3A301711&commentId=6223006%3AComment%3A473586 >. Acesso em fev. 2017
[7] Link para download da biblioteca disponível em: < http://api.ning.com/files/MN6ZNqVOIKPbUHq42lc0iBs8Nx0AGXeAe-KTEySr5CZ4LWNMqEASbQak6Mww8nwDyf0KL3J4Vc*4FHwQthNIAYjmjlKcAK3M/DualMotor.zip >. Acesso em fev. 2017

Figura 8 Dual Motor Shield.

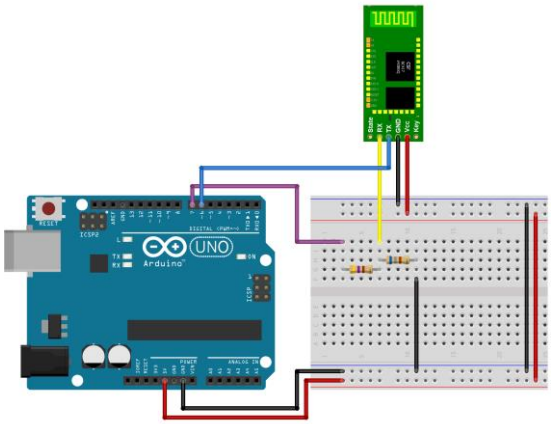


Fonte: Laboratório Garagem.[8]

A comunicação entre o computador e o Arduino® será feita através de um adaptador *bluetooth hc06* [9] acoplado a placa. Este módulo funciona apenas no formato escravo, ou seja, permite apenas que outros dispositivos se conectem a ele, enviando as informações para Ardruino® via porta serial. O alcance é de 10 metros, valor padrão encontrado em dispositivos *bluetooth*. O módulo possui quatro pinos: Vcc para alimentação de 3.6 a 6 volts, GND, RX e TX, os últimos são utilizados para comunicação e trabalham com 3.3V. Como o Arduino® Leonardo trabalha com cinco volts, é necessário utilização de resistores para o funcionamento do módulo. A Figura 9 exemplifica a montagem do módulo em um Arduino® e a Figura 10 o seu esquema final.

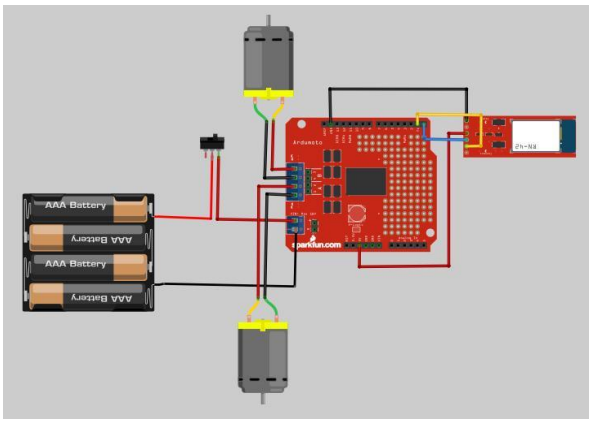
[8] Disponível em: < http://labdegaragem.com/profiles/blog/show?id=6223006%3ABlogPost%3A301711&commentId=6223006%3AComment%3A473586 >. Acesso em fev. 2017.
[9] *Datasheet* disponível em < https://www.olimex.com/Products/Components/RF/BLUETOOTH-SERIAL-HC-06/resources/hc06.pdf >. Acesso em fev. 2017.

Figura 9 Módulo bluetooth.



Fonte: Autoria própria.

Figura 10 Esquema final do robô.



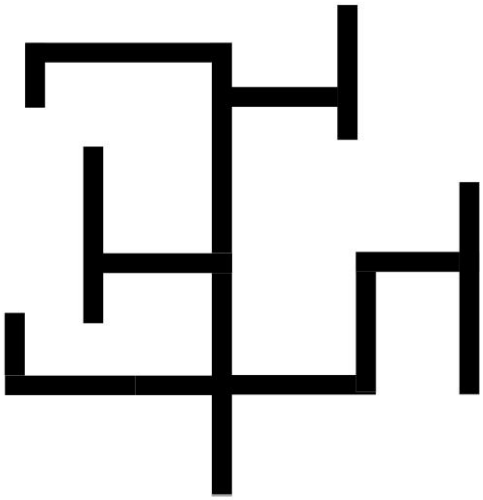
Fonte: Autoria própria.

O robô apresentado nessa seção será colocado em um labirinto, o qual será explicado na seção 3.2. O robô receberá as informações necessárias para encontrar a saída entre dois pontos do labirinto a partir de instruções enviada por um computador.

3.2 Labirinto

O labirinto físico utilizado no projeto é uma folha de papel de 1.20m de comprimento por 1.20m de largura com faixas pretas horizontais e verticais, conforme mostrado na Figura 11. As faixas pretas representarão a pista em que o robô irá se locomover.

Figura 11 Labirinto digital do projeto.



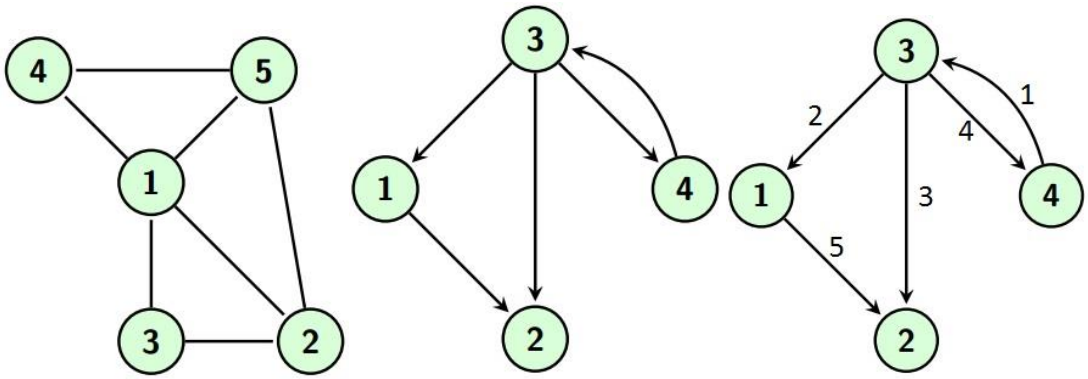
Fonte: Autoria própria.

Para que o computador saiba quais instruções enviar para o robô, alguns algoritmos de teoria dos grafos e técnicas de PDI serão utilizados. As técnicas de PDI serão usadas para poder criar a representação digital do labirinto na forma de um grafo, para que então os algoritmos de teoria dos grafos possam ser utilizados para encontrar o menor caminho entre dois pontos do labirinto.

3.2.1 Teoria dos Grafos

Teoria dos grafos é um ramo da matemática que estuda a relação entre objetos em um determinado conjunto. Esse conjunto (G) é composto por vértices (V) e arestas (A), comumente tratado por G(V,A). As arestas estão associadas a um ou mais vértices, fazendo a ligação entre outras arestas. Os vértices podem ser dirigidos (possuem sentido) ou não dirigidos (não possuem sentido), no caso dos grafos dirigidos não existirão arestas e sim arcos, e ainda podem conter pesos em cada uma de suas arestas ou arcos (FEOFILOFF, KOHAYAKAWA e WAKABAYASHI, 2011). A Figura 12 mostra alguns exemplos de grafos.

Figura 12 Grafos não dirigido – dirigido – dirigido com custos.



Fonte: Bonates, 2013B.

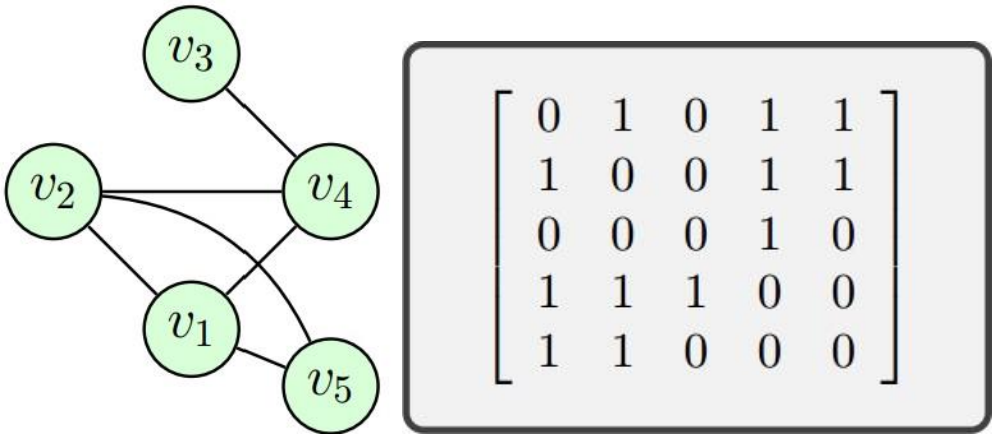
3.2.2 Representação de grafos

Na Figura 12 os grafos foram mostrados de maneira esquemática, bastante útil para entender a estrutura do grafo de forma didática. Porém, é necessária uma forma matemática para sua representação em memória, ou seja, para que o computador entenda seu funcionamento. Como exemplo, temos a matriz de adjacência, matriz de incidência e lista de adjacência. As três opções tem funcionamento semelhante, com a diferença de um melhor desempenho para a implementação através de listas de adjacência quando o grafo possui uma pequena quantidade de arestas em relação ao maior número possível de vértices, que não é caso desse trabalho. Dessa forma, neste trabalho, é utilizada a matriz de adjacência, devido a sua implementação simples e bom desempenho para a aplicação.

3.2.3 Matriz de Adjacência

Esta matriz é chamada de adjacência, pelo fato de cada valor da matriz informar se dois vértices são adjacentes ou não. Será necessário uma matriz M = $[m_{ij}]_{ij}$ de inteiros com dimensão *n x n*. Cada entrada da matriz estará associada com uma aresta do grafo de acordo com a seguinte lógica: se $m_{ij} = 1$, então a aresta $\{v_i, v_j\}$ irá existir; se $m_{ij} = 0$, então $\{v_i, v_j\}$ não irá existir. A Figura 13 mostra um grafo e sua respectiva matriz de adjacência, onde a primeira linha da matriz é referente aos vértices adjacentes de v1, a segunda referente ao vértice v2 e assim sucessivamente.

Figura 13 Grafo e matriz de adjacência.



Fonte: Bonates, 2013C.

Com a matriz de adjacência implementada será computacionalmente possível aplicar algoritmos de busca em grafos para encontrar a menor rota entre dois pontos. Esses algoritmos serão discutidos na seção 3.2.4.

3.2.4 Busca em Grafos

Uma parte fundamental do projeto é encontrar o menor caminho entre um dado ponto inicial e final do labirinto. Segundo Hillier e Lieberman (2013), o problema de caminho mínimo é o maior grupo de problemas na área de pesquisa operacional e é considerado um dos mais importantes. As aplicações do problema de caminho mínimo estão relacionados com a otimização de atividades como, por exemplo, o tráfego em estradas, conexão em redes de computadores, planejamento do movimento de um robô e etc. (EPPSTEIN, 1998).

Para solucionar o problema enunciado no inicio da seção 3.2.4 serão utilizados algoritmos de busca em grafos. Os algoritmos de busca em grafos podem trabalhar de diversas formas, como por exemplo, quando não é de conhecimento prévio que a informação a ser encontrada realmente está inserida no grafo, para isso temos os algoritmos de busca em largura e busca em profundidade, porém eles não funcionam quando temos pesos em suas arestas.

Para resolver a questão de um grafo com pesos em suas arestas existem algoritmos de caminho mínimo, onde são informados dois vértices, querendo descobrir o caminho com o menor custo possível entre esses vértices. Para resolver esse problema é possível usar o algoritmo de Djikstra que trabalha apenas com pesos positivos e o de Floyd-Warshall que trabalha com valores positivos e negativos. O algoritmo de Djisktra foi escolhido para o

projeto pelo motivo do labirinto utilizado não possuir arestas com pesos negativos, e seu custo computacional ser mais baixo em relação ao de Floyd-Warsall.

3.2.5 Algoritmo de Djikstra

O algoritmo do holandês Edsger Djikistra soluciona o problema de caminho com o menor custo em grafo ponderado de valores positivos com um tempo computacional de O([m+n]log n) onde m é o número de arestas e n o número de vértices, essa é uma complexidade considerada boa em questão de desempenho de algoritmos de busca. O algoritmo segue uma estratégia gulosa, partindo de um vértice inicial e sempre buscando o vértice seguinte com o menor peso possível. A Figura 14 mostra um pseudocódigo do algoritmo de djikistra e a Figura 15 mostra o funcionamento e o resultado final de sua execução em um grafo.

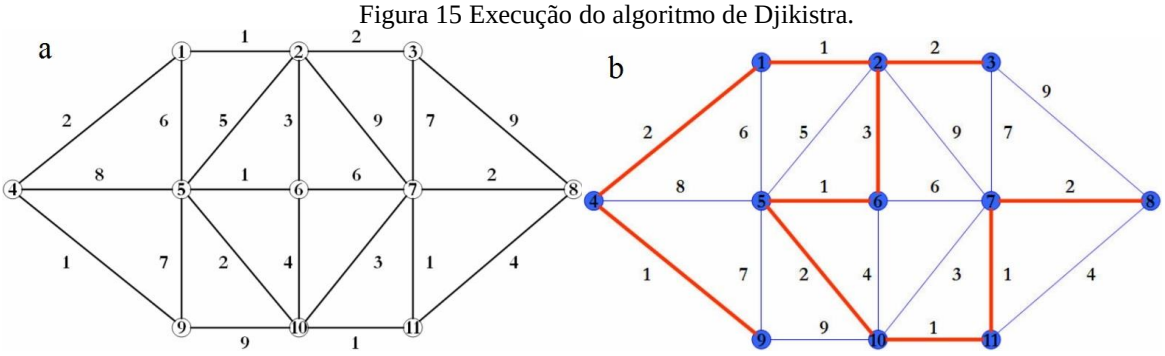
Figura 14 Algoritmo de Djikistra.

```
   ENTRADA: Grafo dirigido e ponderado G = (V, A, c), vértice origem s ∈ V.
   SAÍDA  : Árvore de caminhos mínimos relativa a s.
 1 d(v) := ∞, para todo vértice v ∈ V
 2 d(s) := 0, pred[s]=s
 3 S := V
 4 while |S| > 1 do
 5 |   Seja w ∈ S um vértice com d(w) = min{d(v) : v ∈ S}
 6 |   S := S \ {w}
 7 |   for (w, v) ∈ A com v ∈ S do
 8 |   |   if d(v) > d(w) + c(w, v) then
 9 |   |   |   d(v) := d(w) + c(w, v)
10 |   |   |   pred[v] := w
```

Fonte: Bonates, 2013A.

É possível notar que o algoritmo não solicita o nó destino em nenhum momento, apenas o nó de origem, isso se dá pelo fato do algoritmo retornar uma arvore de caminho mínimo, ligando o nó origem a todos os demais vértices encontrados no grafo. Com essa informação é possível encontrar o menor caminho entre o ponto inicial informado e qualquer outro ponto, basta verificar os predecessores do vértice de destino. Outro ponto que merece destaque é a necessidade do grafo ter ligação entre todos os vértices, caso não possua é necessário cria-la com um valor alto suficiente para representar a sua “não existência”.

Figura 15 Execução do algoritmo de Djikistra.



Fonte: Barros, Pamboukian e Zamboni, 2007.

A Figura 15a mostra um grafo não dirigido com pesos nas arestas, e a Figura 15b demonstra a arvore contendo o caminho mínimo entre todos os vértices do grafo. Para conseguir montar o grafo utilizado nesse projeto foi necessário utilizar técnicas de PDI, e essas técnicas serão explicadas na seção 3.3.

3.3 Aquisição de Imagens

As imagens do projeto foram capturadas a partir da câmera de um smartphone posicionado em um suporte como mostrando na Figura 4. O grafo será montado através das imagens capturadas pela câmera e técnicas de PDI serão utilizadas para analisar essas imagens. A seção 3.3.1 irá explicar quais técnicas de PDI são utilizadas no projeto.

3.3.1 Processamento Digital de Imagens

Uma imagem pode ser definida como uma função bidimensional, *f(x, y)*, em que *x* e *y* são coordenadas espaciais (plano), e a amplitude de intensidade ou nível de cinza da imagem nesse ponto. Quando *x*, *y* e os valores de intensidade de *f* são quantidades finitas e discretas, chamamos de imagem digital (GONZALEZ e WOODS, 2010).

Para representar imagens coloridas, como por exemplo, imagens em *Red, Green* e *Blue* (RGB), é necessário armazenar a intensidade das cores vermelha (R ou *red)*, verde (G ou *green*) e azul (B ou *blue*) em uma determinada posição (x,y) da imagem. Esse ponto (x,y) junto com a intensidade de cores será tratado como *pixel* (GONZALEZ e WOODS, 2010).

O processo de extrair informações de uma imagem é conhecido como processamento digital de imagens. Comumente o processo irá gerar uma nova imagem e são conhecidos como processos de baixo nível. O processo de extração de informações de uma imagem,

como a segmentação, é um processo de médio nível. Por fim os processos de alto nível costumam aplicar operações mais complexas na imagem (SIQUEIRA e VERGARA, 2016). Nesse projeto serão utilizadas técnicas de segmentação de imagem para detecção do robô.

3.3.2 Segmentação de imagens

Segmentar uma imagem é dividi-la em várias partes, como em regiões ou objetos (GONZALEZ e WOODS, 2010). O seu principal objetivo é capturar um objeto em uma imagem para analisá-lo como, por exemplo, encontrar um robô em um labirinto. Será abordado nesse projeto a segmentação através da binarização da imagem e detecção de bordas.

3.3.3 Binarização

A binarização de imagens é uma técnica de segmentação de imagens que trabalha com imagens em escala de cinza, ou seja, cada pixel da imagem irá variar entre o valor zero representando a cor preta, e 255 representando a cor branca (MARENGONI e STRINGHINI, 2009). A partir do nível de cinza da imagem é possível definir um limiar, como mostra a equação 1.

$$g(x,y) = \begin{cases} 1, se\ f(x,y) \geq k \\ 0, caso\ contrário \end{cases} \qquad (1)$$

No projeto esse limiar é utilizado para que a pista (faixas pretas) possuam o valor 0 (zero) e todo o resto o valor 1 (um).

3.3.4 Detecção de Bordas

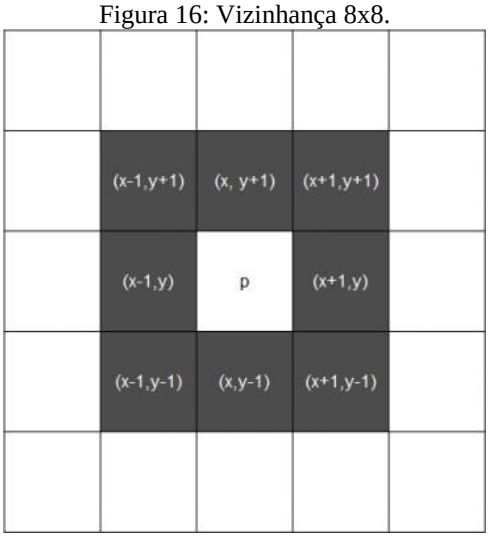
A borda de uma imagem digital é a fronteira em que uma significativa mudança ocorre em algum aspecto físico da imagem, como em sua superfície ou iluminação (SIQUEIRA e VERGARA, 2016). Detecção de bordas ou detecção de contornos trata de extrair as bordas de uma imagem digital. Com o contorno de um determinado objeto é possível extrair informações do mesmo e usa-las posteriormente. Quanto melhor for a detecção do contorno melhores serão as informações a serem extraídas do objeto. Outro fator importante em utilizar

a detecção de bordas, é o ganho computacional devido ao número de pixels que serão analisados ser bem menor que o total de pixels da imagem (ROCHA, 2012). Apenas será necessário verificar os pixels que se encontrão no contorno da imagem, e não todos os pixels da imagem. Esse projeto utilizará o algoritmo de Moore Neighbor que será explicado na seção 3.3.5.

3.3.5 Algoritmo de Moore-Neighbor Tracing

O método *Moore-Neighbor Tracing* trabalha com vizinhança de um determinado pixel, onde essa vizinhança serão os oito pixels que cercam o pixel em questão como mostram a Figura 16. Essa vizinhança também é conhecida como Vizinhança de Moore.

Figura 16: Vizinhança 8x8.



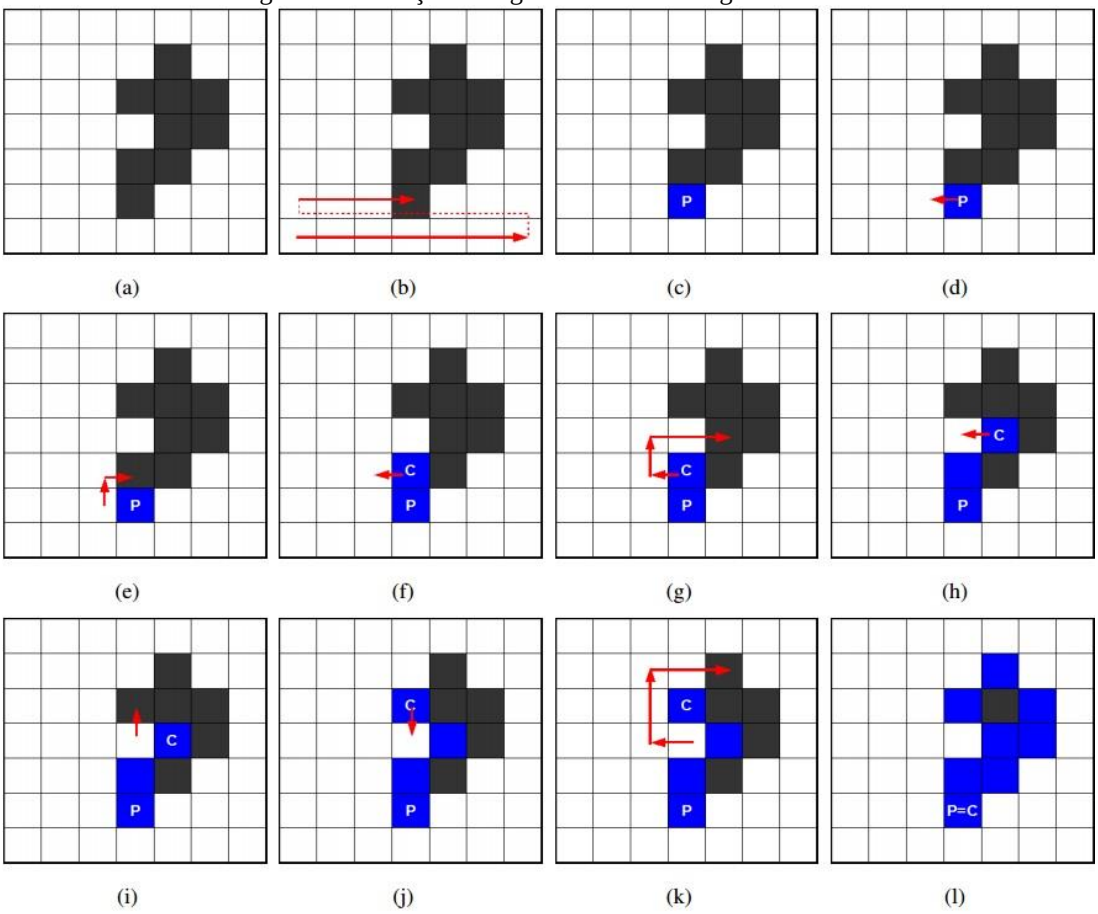
Fonte: Rocha, 2012.

Dado uma imagem binária com objetos da cor preta e fundo da cor branca, o primeiro passo do algoritmo de Moore é encontrar um pixel preto e inicializa-lo como pixel de partida. Este pixel é localizado varrendo a imagem do ponto superior esquerdo até o ponto inferior direito, varrendo todos os pixels das linhas e colunas da imagem. O segundo passo é encontrar os pixels que completam o contorno da imagem, através da analise de suas vizinhanças (ROCHA, 2012).

A Figura 17 exemplifica a execução do algoritmo seguindo os seguintes passos: Encontrar pixel de partida (Figura 17b) e (Figura 17c), buscar o pixel na vizinhança do pixel atual (Figura 17d) e (Figura 17e) e concluir o contorno seguindo os passos anteriores como

mostra (Figura 17f), (Figura 17g), (Figura 17h), (Figura 17i), (Figura 17j) e (Figura 17k) até encontrar novamente o pixel de partida em (Figura 17l).

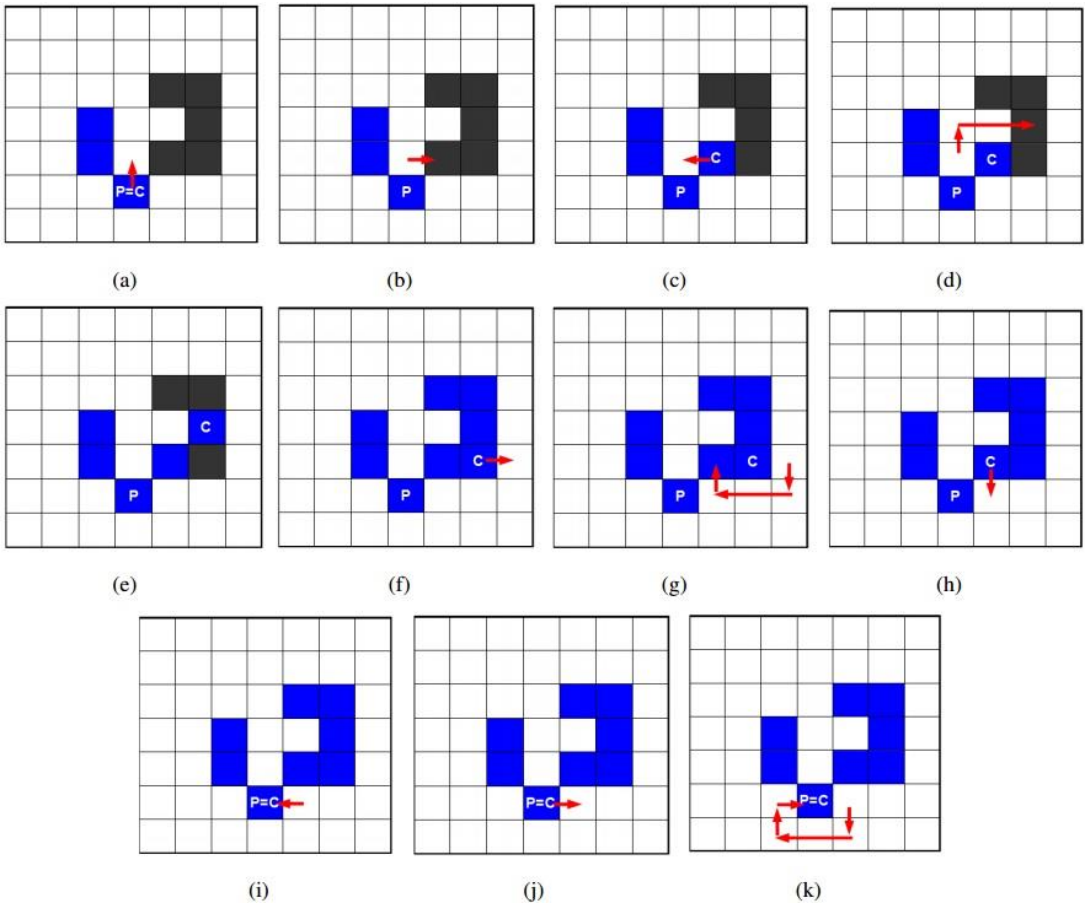
Figura 17 Execução do algoritmo Moore-Neighbor.



Fonte: Rocha, 2012.

O principal problema do algoritmo *Moore-Neighbor Tracing* está em seu critério de parada, onde o pixel de partida é visitado pela segunda vez. Para exemplificar o problema, imagine o objeto da Figura 18, nele o pixel de partida mostrado na Figura 18a pode seguir duas direções diferentes, e independente de qual direção seja tomada não será possível detectar o objeto por completo. Para contornar esse problema é utilizado o critério de parada de Jacob, conhecido como *Jacob's stopping criterion*. Este critério de parada faz com que o algoritmo de Moore varra o pixel inicial uma segunda vez em busca de algum outro pixel preto não marcado, como mostra a Figura 18.

Figura 18 Execução do algoritmo *Moore-Neighbor* com o critério de parada de Jacob.



Fonte: ROCHA, 2012.

Em Figura 18a, caso o critério de Jacob não fosse utilizado, o algoritmo irá parar sua execução nesse ponto e não detectaria o resto da imagem. Com a utilização nota-se uma segunda varredura no pixel inicial como mostra em Figura 18b e o reinício das buscas pela borda da imagem em Figura 18c, Figura 18d, Figura 18e, Figura 18f, Figura 18g, Figura 18h, Figura 18i e Figura 18j até encontrar novamente o pixel inicial e encerrar sua execução.

Para criar todos os algoritmos apresentados nas seões desse capítulo foram utilizados dois ambientes de programação distintos, o primeiro foi o desenvolvimento do carrinho com Arduino®, apresentando na seção 3.1.2 e o segundo é o ambiente de programação MATLAB® que será apresentado na seção 3.4.

3.4 Ambiente de Programação

Os algoritmos do projeto foram desenvolvidos utilizando a ferramenta MATLAB *MathWorks*® para trabalhar com as imagens, os algoritmos de PDI e os de grafos. O MATLAB® é um software interativo de alto desempenho voltado para cálculos numéricos. O

MATLAB® permite que o usuário interaja com o programa através de uma janela conhecida como Janela de Comando. Através dessa janela o usuário fornece os comandos para sua utilização.

Além da janela de comandos, também é possível utilizar o MATLAB® com scripts. Eles são arquivos onde é possível colocar uma lista de comandos em sequência e executá-los sempre que for necessário, sendo possível utilizar laços de repetição, instruções de desvio, invocar funções, declarar variáveis e etc. Todos os algoritmos do MATLAB® utilizados nesse projeto estão contidos em um script e em funções.

3.5 Aplicação

O algoritmo foi dividido em cinco módulos distintos, como ilustra a Figura 19. O desenvolvimento de cada módulo também seguiu a ordem mostrada na Figura 19, devido à necessidade do item seguinte depender da saída do item anterior. Essa divisão tornou simples verificar o quanto faltava para conclusão de projeto, em questão de tempo, auxiliando em um melhor planejamento.

Figura 19 Módulos do projeto

Fonte: Autoria própria.

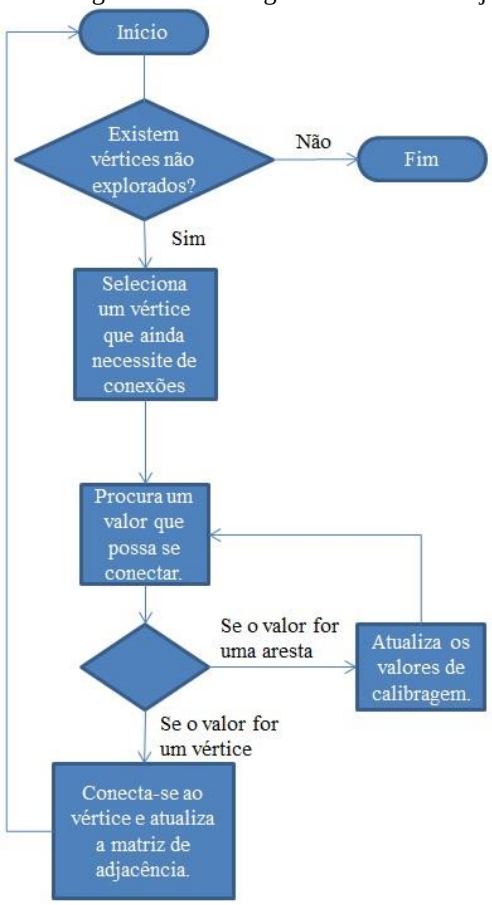
A fase de preparação da imagem inicia com a transmissão do labirinto pela câmera para que o usuário possa ajustar a posição da câmera, do labirinto e o nível de luminosidade. Após os ajustes, é solicitado que o usuário defina uma região de corte na imagem que está sendo transmitida pela câmera. Esse corte é necessário para que a imagem enviada pelo algoritmo seja apenas da área do labirinto.

A fase de pré-processamento irá receber a imagem da fase anterior como entrada e retornará um conjunto de pontos representando os vértices e as arestas de um grafo. Para tanto é necessário passar a imagem para escala de cinza, verificar os níveis de cinza em todos os pontos da imagem, e de acordo com um limiar, converter o ponto verificado para preto ou branco, processo conhecido como binarização, o qual foi tratado na seção 3.3.3. Com a imagem em preto e branco é possível começar uma verificação para encontrar o que é ou não pista.

Com a imagem em preto e branco o algoritmo irá começar uma varredura para detectar a pista do labirinto. Essa varredura inicia no pixel no extremo superior esquerdo e segue varrendo os pixels da imagem, através de um passo que é um valor informado pelo usuário, até o pixel na posição inferior direita. O passo é utilizado para acelerar o algoritmo, pois varrer todos os pixels da imagem é desnecessariamente demorado.

No processo de varredura, sempre que é encontrado um pixel com o valor zero (preto), é iniciada uma verificação na vizinhança desse pixel para descobrir se esse pixel é uma reta horizontal, vertical, ou um ponto de bifurcação. Todas as informações coletadas na fase pré-processamento sobre os pixels é armazenada em uma estrutura, essas informações são números inteiros que variam de 0-13, onde esses valores informam que direções podem ser tomadas nos pontos verificados. O próximo passo é gerar a matriz de adjacência a partir da estrutura fornecida pela fase anterior. O fluxograma da Figura 20 ilustra o funcionamento do algoritmo de ligações dos vértices.

Figura 20 Fluxograma de montagem da matriz de adjacência.



Fonte: Autoria Própria.

O algoritmo irá iniciar no primeiro vértice na estrutura que contém os pontos da fase anterior e fará a ligação com o(s) vértice(s) em sua vizinhança. Quando uma ligação é feita, é inserida uma marcação na direção do vértice marcado para evitar verificações redundantes.

O processo de calibragem é feito para garantir que o vértice que está sendo explorado no momento não se ligue com o vértice que esteja fora de sua aresta. Essa calibragem é o número de pixels o qual ele deve ir a uma determinada direção em busca de um vértice para efetuar a ligação. O valor de calibragem é incrementado sempre que encontra um valor correspondente a uma aresta.

Com todos os vértices explorados a função irá retornar para o algoritmo de djkistra (o qual é explicado na seção 3.2.4) a matriz de adjacência, então o usuário poderá informar os vértices de origem e destino para o que o algoritmo retorne a rota com o menor custo possível.

Por fim, o algoritmo irá guiar o robô da posição inicial até a final informada pelo usuário na etapa anterior. Para localizar o robô no labirinto é utilizada a *toolbox* do MATLAB

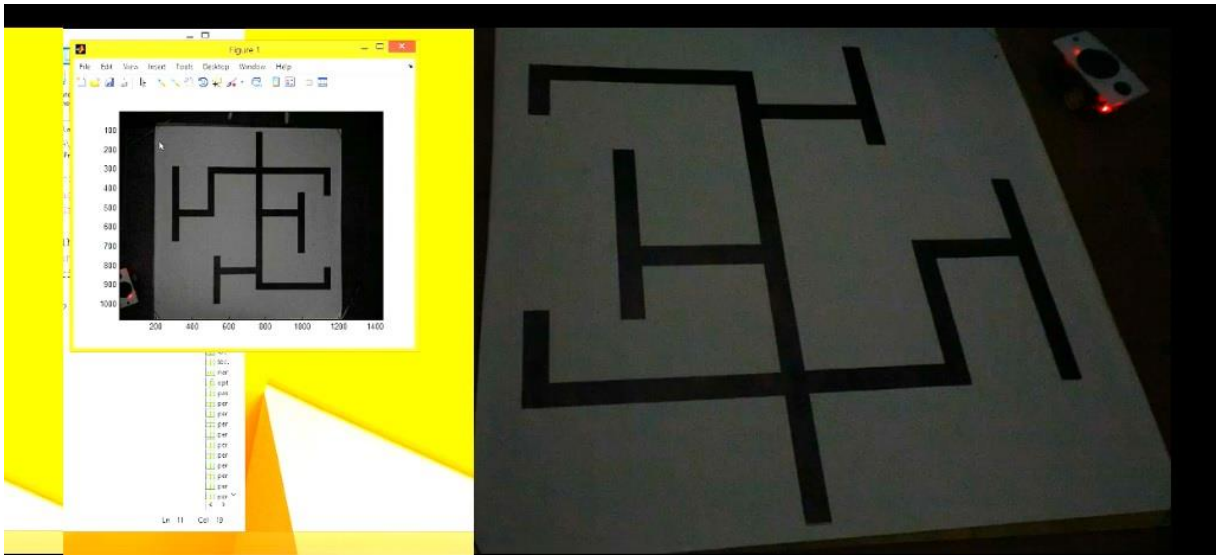
bwboundaries[11], esta função implementa o algoritmo de *Moore-Neighbor* com o critério de parada de Jacob explicado na seção 3.3.5. A *bwboundaries* pode encontrar diversas informações com relação a objetos presentes em uma cena, como sua área, perímetro, diâmetro, entre outras. Mais detalhes da implementação são mostrados no capítulo 4.

[11] Especificações em: < https://www.mathworks.com/help/images/ref/bwboundaries.html >

4 RESULTADOS

Nesse capítulo são apresentados os resultados obtidos no desenvolvimento do projeto, demonstrando todas as fases da aplicação através de figuras e explicações do seu funcionamento. A explanação seguirá a ordem do fluxograma da Figura 20 apresentado na seção 3.5. A primeira fase é a de preparação, onde o usuário tem a possibilidade de preparar os dispositivos para melhor atender sua cena, como mostra a Figura 21. Do lado esquerdo da imagem é possível visualizar qual visão a câmera que é usada na aplicação tem do labirinto, e a imagem da direita foi captura por uma câmera externa, para efeito de comparação.

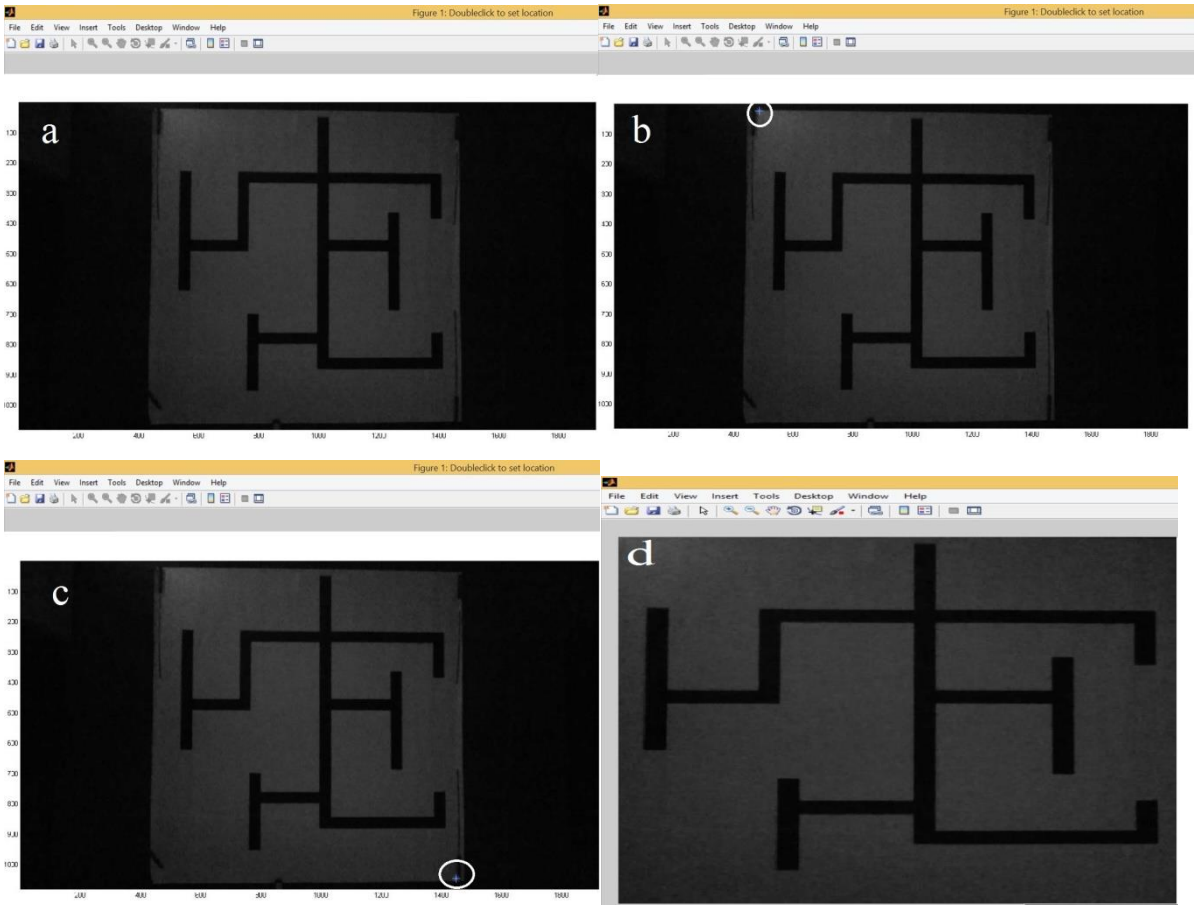
Figura 21 Transmissão da imagem.



Fonte: Autoria própria.

A próxima etapa, ainda dentro da fase de preparação, é o corte da imagem que é detalhado na Figura 22 demostrando o seu passo-a-passo. Na Figura 22a é ilustrada a imagem original, no ponto superior esquerda da Figura 22b é selecionado o primeiro ponto de corte, no ponto inferior direito da Figura 22c é ilustrado o segundo ponto de corte, e por fim a Figura 22d ilustra a imagem final recortada.

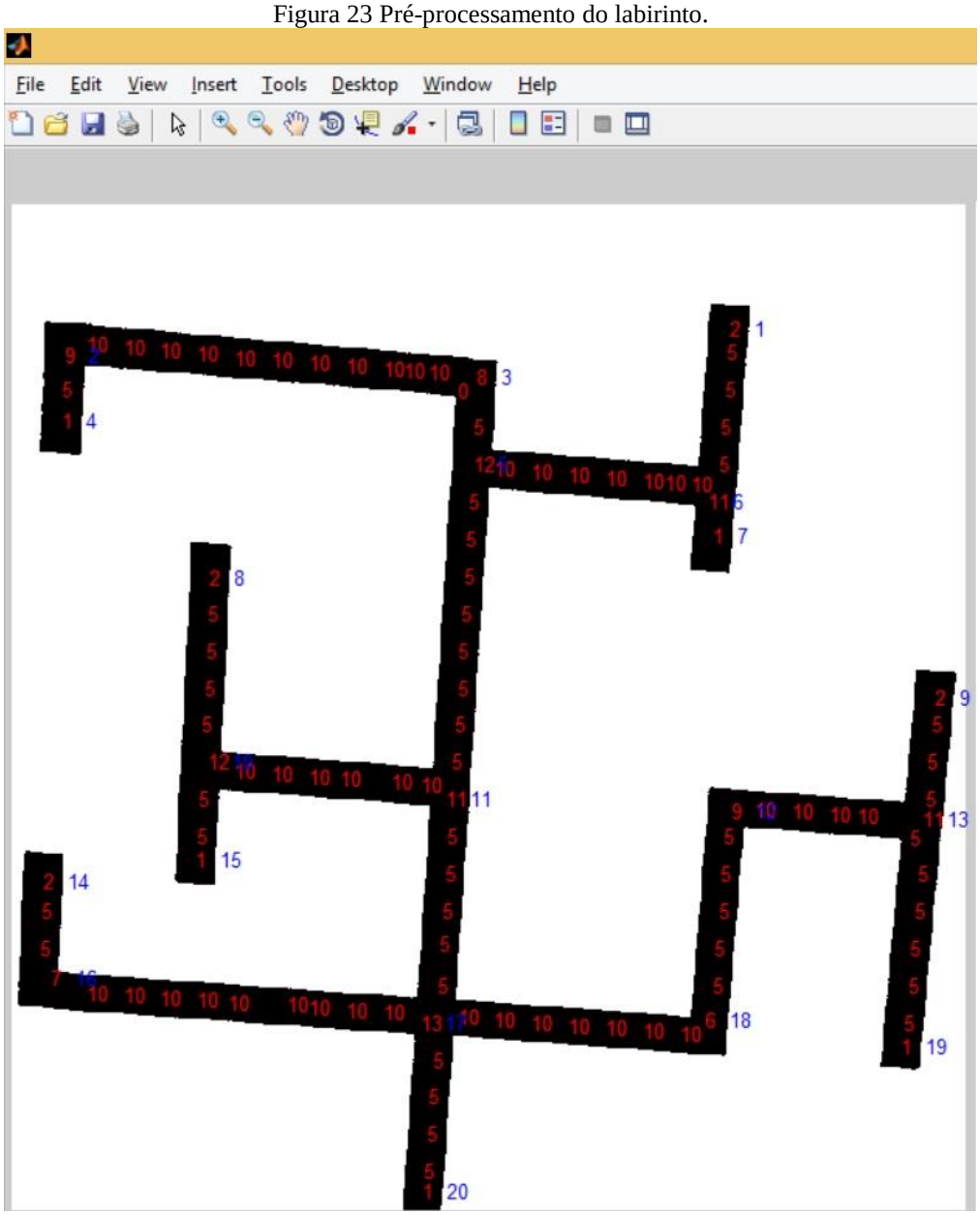
Figura 22 Recorte da imagem.



Fonte: Autoria Própria.

A segunda fase do projeto é a de pré processamento da imagem, onde a imagem resultante da imagem anterior é passada para preto e branco, binarizada e tem os vértices da imagem descobertos, que serão os mesmos do grafo. A Figura 23 mostra todos os vértices encontrados na imagem.

Figura 23 Pré-processamento do labirinto.



Fonte: Autoria Própria.

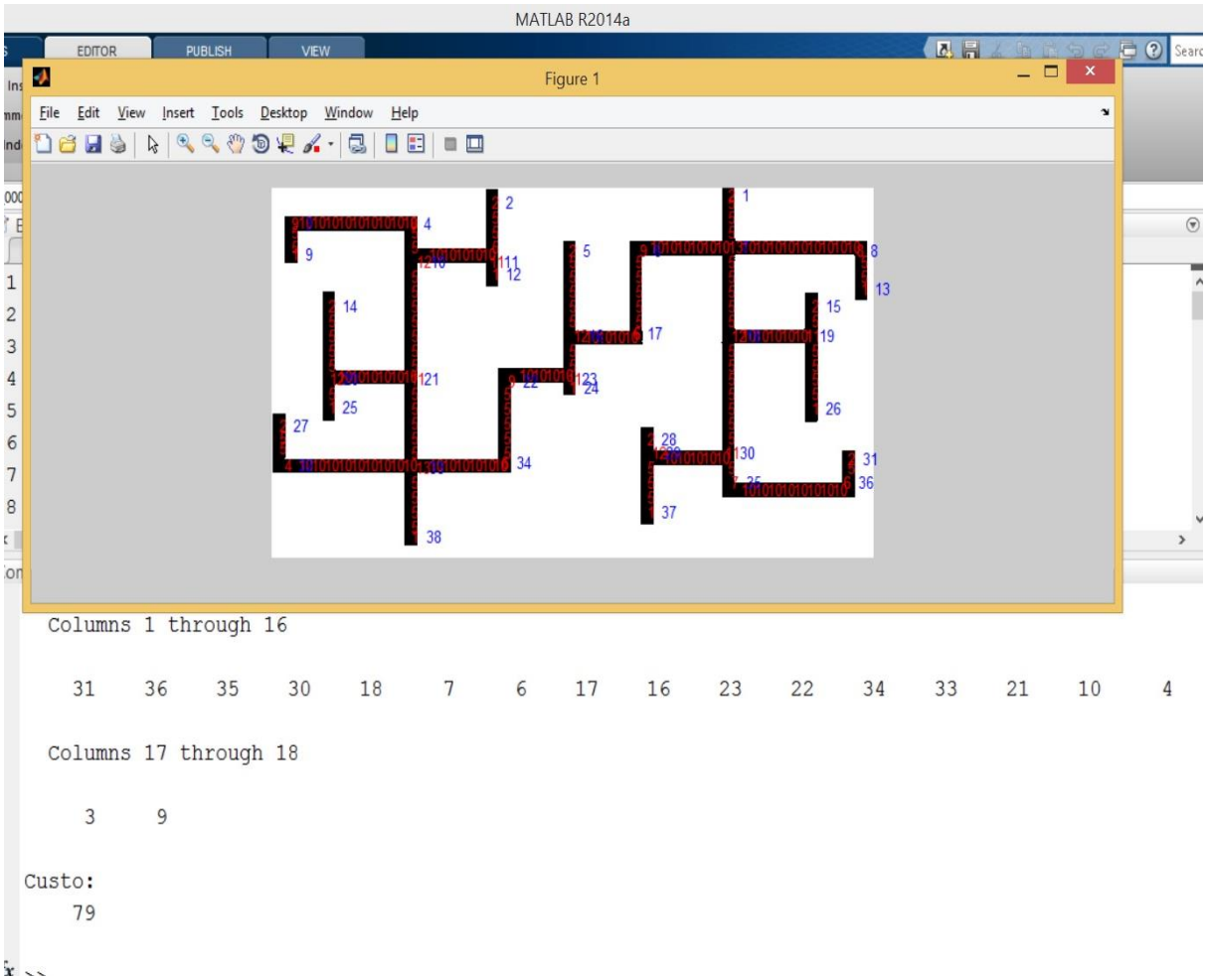
Os números em azul representam os vértices do grafo, os números em vermelho representam as direções que podem ser tomadas nesses pontos. Os números das direções possíveis que o robô poderá se dirigir foram codificados da seguinte forma:

- 1 Cima
- 2 Baixo
- 3 Esquerda
- 4 Direita
- 5 Cima e baixo
- 6 Cima e esquerda
- 7 Cima e direita
- 8 Baixo e esquerda
- 9 Baixo e direita
- 10 Esquerda e direita
- 11 Cima, baixo e esquerda
- 12 Cima, baixo e direita
- 13 Todas as direções

O próximo passo do algoritmo é a geração da matriz de adjacência, como exemplo do seu funcionamento, o vértice 1 (um), que tem direção número 2 (dois), ou seja, só pode ir para baixo, irá buscar um vértice que esteja posicionado abaixo dele, que no caso é o vértice 6 (seis) de direção 11 (onze), o qual tem direções cima, baixo e esquerda. Quando o algoritmo for verificar o vértice 6 (seis), ele não precisará verificar a ligação dele para cima, pois uma flag é setada para que o algoritmo saiba que a ligação já foi realizada.

O próximo passo é encontrar o menor caminho entre dois vértices do grafo, para exemplificar o funcionamento do algoritmo de djikistra, a Figura 24 mostra um labirinto digital e a rota entre os vértices 31 (trinta e um) e o vértice 9 (nove), com custo 79 (setenta e nove). Não foi utilizado um labirinto físico tão grande no projeto devido ao custo da impressão e a estrutura para transmitir o labirinto precisaria ficar em uma altura inviável para o desenvolvimento.

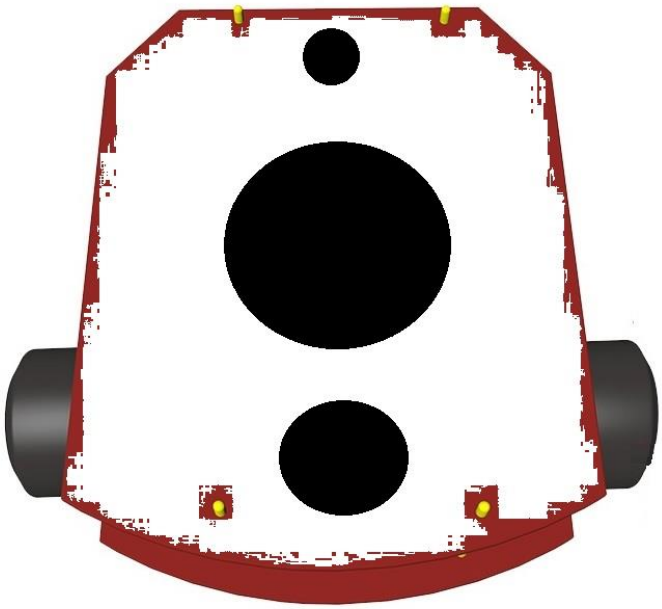
Figura 24 Execução do algoritmo de djikistra.



Fonte: Autoria própria.

Após o algoritmo de djikistra tem início a fase de guiar o carrinho, para facilitar a detecção é posicionada uma folha em branco com três círculos pretos de tamanhos distintos desenhados na parte de cima do carrinho, como demostra a Figura 25. O circulo maior no centro é usado para detectar a posição do carrinho e os outros dois círculos nas extremidades são utilizados para verificar sua direção. O início dessa fase é feita uma calibragem para determinar a área e o perímetro desses círculos para que a função *bwboundaries* detecte o carrinho de forma correta. Também será traçada uma *bounding box* ao redor dos círculos para detectar a posição (x,y) do centro deles.

Figura 25 Carrinho com marcações para detectação.

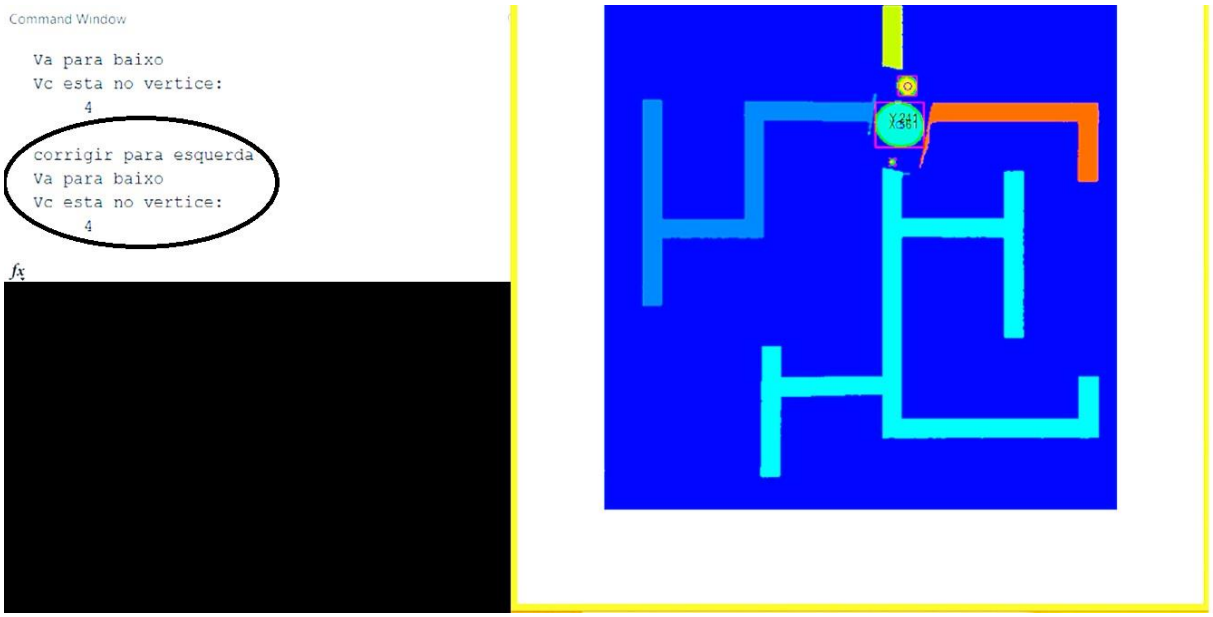


Fonte: Autoria Própria.

Para verificar a posição do carrinho, é processada a posição dos dois círculos nas extremidades do carrinho. Se eles estiverem na mesma coluna, ou com uma margem de erro previamente estabelecida, é considerado que se encontra na posição vertical, caso estejam na mesma linha, com a margem de erro dentro do aceitável, estará na posição horizontal. Caso as posições dos círculos estejam com uma diferença de valor maior que a permitida, então o carrinho estará torto e será necessário ajustar sua posição. A Figura 26 mostra a execução do algoritmo de detecção de carrinho. É possível notar que o algoritmo informa em qual vértice o

carrinho se encontra, qual sentido ele deve seguir, e para qual direção deve corrigir o seu posicionamento.

Figura 26 Detectando a posição e sentido do carrinho.



Fonte: Autoria própria.

Essa fase é finalizada quando o algoritmo detecta que o carrinho chegou ao vértice no destino. Foram realizados inúmeros testes com labirintos virtuais mais complexos, como o ilustrado na Figura 24, e labirintos físicos menores que é ilustrado na Figura 23, para todos os labirintos foram testados vários vértices de partida e chegada diferentes. Também foram feitos testes para verificar a detecção do carrinho em vários pontos do labirinto, para verificar a capacidade do algoritmo em detectar o carrinho em várias situações diferentes.

5 CONCLUSÕES

Os resultados obtidos com os algoritmos de teoria de grafos foram satisfatórios, alcançando todos os objetivos de forma esperada. O algoritmo de djikstra coube na resolução do problema do labirinto. O algoritmo desenvolvido para montagem da matriz de adjacência também funcionou bem em diversos labirintos diferentes.

O carrinho usado no trabalho apresentou problemas na sua locomoção, devido ao fato de suas rodas serem controladas por dois motores diferentes, a potência enviada para as rodas não eram as mesmas, fazendo com que o carrinho não andasse de forma reta, mas sempre tendendo para algum dos lados. Porém esse problema foi resolvido acrescentando uma verificação da posição do carrinho na parte do algoritmo responsável por detectar o carrinho.

Algumas dificuldades também foram encontradas na parte de binarização da imagen devido à iluminação do ambiente, mas foi solucionado com a diminuição da luminosidade do local escolhido para os testes. A função *bwboundaries()* não se mostrou a mais adequada para a detecção do carrinho, apresentando alguns problemas, como por exemplo, confundindo trechos da pista com o os círculos que identificam o carrinho, esse problema foi parcialmente resolvido, e uma melhor solução para o problema será abordada na seção 5.1.

Em todos os testes feitos no trabalho com relação ao labirinto, tanto os virtuais como o físico, o algoritmo conseguiu detectar todos os vértices do algoritmo e descobrir a menor rota entre quaisquer dois pontos informados. Com relação a detecção do carrinho, apesar dos problemas encontrados, o algoritmo apresentou resultado satisfatório e conseguiu guiar o carrinho até a saída em todos os testes.

5.1 Trabalhos futuros

Como trabalhos futuros será considerada a implementação de uma interface gráfica para tornar a aplicação mais amigável. Uma outra forma de detectar o carrinho também poderá ser estudada, visando aliar a função *bwboundaries* com uma detecção de cores, reforçando ainda mais a detecção de objetos na cena. Também será estudada a criação de um novo carrinho, que seja menor e mais robusto, com o intuito de resolver o problema de locomoção encontrado no decorrer do desenvolvimento.

REFERÊNCIAS

ARDUINO, 2017. Disponivel em: <https://www.arduino.cc/en/Main/ArduinoBoardLeonardo>.

ASIMOV, I. **Eu, robô**. [S.l.]: Aleph, 2015.

BANZI, M. **Primeiros passos com o Arduino**. São Paulo: Novatec, 2011.

BARROS, E. A.; PAMBOUKIAN, S. V.; ZAMBONI, L. C. Algoritmo de Dijkstra: apoio didático e multidisciplinar na implementação, simulação e utilização computacional. **INTERNATIONAL CONFERENCE ON ENGINEERING AND COMPUTER EDUCATION**, São Paulo, 2007.

BIANCHI, R. A.; REALI-COSTA, A. H. O Sistema de Visão Computacional do time FUTEPOLI de Futebol de Robôs. **Congresso Brasileiro de Automática**, 2000. 2156 - 2162.

BITMAG, 2017. Disponivel em: <http://www.bitmag.com.br/2016/04/mercado-de-automacao-segue-em-forte-crescimento/>. Acesso em: 24 Janeiro 2017.

BONATES, T. O. Teoria dos Grafos - Caminhos Mínimos - Parte 1, 12 julho 2013A. Notas de Aula.

BONATES, T. O. Teoria dos Grafos - Conceitos Básicos, 10 maio 2013B. Notas de Aula.

BONATES, T. O. Teoria dos Grafos - Representação Computacional de Grafos, 22 maio 2013C. Notas de Aula.

BONDY, J. A.; MURTY, U. S. R. **Graph theory with applications**. Londres: Macmilian, 1976.

CHIELLA, M. T. Ambiente de robótica educacional com logo. **XXII Congresso da Sociedade Brasileira de Computação-SBC2002**, Florianópolis, 2002.

COSTA, E. R.; GOMES, M. L.; BIANCHI, R. A. Um mini robô móvel seguidor de pistas guiado por visao local. **VI Simpósio Brasileiro de Automação Inteligente, Bauru, Brasil**, p. 175-186, 2003.

DE SOUZA, J.; ROYER, R. IMPLANTAÇÃO DE UM SISTEMA AGV-VEÍCULO GUIADO AUTOMATICAMENTE UM ESTUDO DE CASO. **XXXIII Encontro Nacional de Engenharia do Produto,(Salvador, BA, Brasil)**, 2013.

EPPSTEIN, D. Finding the k shortest paths. **SIAM Journal on computing**, v. 28, p. 652-673, 1998.

FEOFILOFF, P.; KOHAYAKAWA, Y.; WAKABAYASHI, Y. Uma introdução sucinta à teoria dos grafos. **http:** //www.ime.usp.br/~pf/teoriadosgrafos, 2011.

FLIPFLOP. **2017**. Disponivel em: <http://blog.filipeflop.com/arduino/tipos-de-arduino-qual-comprar.html>. Acesso em: 2017 mar. 03.

GARAGEM, L. D., 2017. Disponivel em: <http://labdegaragem.com/>. Acesso em: 05 mar. 2017.

GOMES, O. S. M. et al. Robô seguidor de linha para competições. **ForScience**, v. 2, n. 2, p. 07-11, 2015.

GONÇALVES, M. et al. Desenvolvimento de uma solução de processamento de imagem em ambiente industrial. **ENEGICOIMBRA2014: inovação, trcnologia e excelência: atas do 3º Encontro Nacional de Engenharia e Gestão Industrial**, 2014. 28-29.

GONZALEZ, R. C.; WOODS, R. C. **Processamento Digital de Imagens**. 3. ed. São Paulo: Pearson, 2010.

HILLIER, F. S.; LIEBERMAN, G. J. **Introdução à pesquisa operacional**. [S.l.]: McGraw Hill Brasil, 2013.

KIM, C.; TANCHOCO, J.; KOO, P.-H. AGV dispatching based on workload balancing. **International Journal of Production Research**, v. 37, n. 17, p. 4053-4066, 1999.

LAMB, F. **Automação Industrial na Prática-Série Tekne**. [S.l.]: AMGH Editora, 2015.

MARENGONI, M.; STRINGHINI, S. Tutorial: Introdução à visão computacional usando opencv. **Revista de Informática Teórica e Aplicada**, v. 16, p. 125-160, 2009. ISSN 1.

MICHAEL, M. **Arduino Básico**. São Paulo: Novatec, 2011. 456 p.

NEU, C. V. Desenvolvimento de um sistema de reconhecimento de sinais de trânsito utilizando processamento de imagens com OpenCV para um robô humanóide, 2014.

NOGUEIRA, A. D.; TEIXEIRA, V. D. F.; RANGEL, W. APLICAÇÃO DE ROBÓTICA EM PROCESSOS DE TRANSPORTE INDUSTRIAL. **COGNITIO/PÓS-GRADUAÇÃO UNILINS**, v. 1, 2015.

PIERI, E. R. D. **Curso de Robótica Móvel**. Florianópoilis: [s.n.], 2002.

PROCESSING, 2017. Disponivel em: <https://processing.org/>. Acesso em: 2017 maio 01.

RAMOS, A. L. C.; SANTOS, J. E. L. D. **Sistema integrado de automação residencial com comunicação sem fio**. Trabalho de Conclusão de Curso. Universidade Tecnológica Federal do Paraná. 2015.

ROBOCORE, 2017. Disponivel em: <www.robocore.net>. Acesso em: 24 abr. 2017.

ROBOCUP, 2017. Disponivel em: <http://www.cbrobotica.org/?page_id=104&lang=pt>. Acesso em: 24 abr. 2017.

ROCHA, T. D. Uma proposta para a classificação de ações humanas baseada nas carcterísticas do movimento e em redes neurais artificiais. , 2012.

ROMANO, V. F.; DUTRA, M. S. Introdução à Robótica Industrial. **Robótica Industrial– Aplicação na Indústria de Manufaturas e de Processos**, v. 1, 2002.

RUI. **Diretorio Artigos**. Disponivel em: <http://diretoriodeartigos.net/surgimento-e-evolucao-da-robotica/>. Acesso em: 12 abr. 2017.

SANTOS, D. D. C.; FREIRE, A. L. D. S. **Sistemas de Automação e Robótica com Arduino:** Utilizando Dispositivos Móveis Inteligente. Mostra Nacional de Robótica. Macapá: [s.n.]. 2015. p. 3.

SANTOS, S. M. P. D. **Aplicações da teoria dos grafos**. Universidade de Aveiro. Aveiro. 2013.

SILVEIRA, L.; LIMA, W. Q. Um breve histórico conceitual da automação industrial e redes para automação industrial. **Redes para Automação Industrial. Universidade Federal do Rio Grande do Norte**, 2003.

SIQUEIRA, P. G.; VERGARA, R. F. Segmentação e parametrização de imagens de ressonância magnética do cérebro: método semi-automático de extração de características para apoio a diagnóstico de pacientes com esquizofrenia, 2016.

WIRING, 2017. Disponivel em: <http://wiring.org.co/reference/>. Acesso em: 2017 maio 01.

ZILLI, S. D. R. A robótica educacional no ensino fundamental: perspectivas e prática, 2004.