

UNIVERSIDADE FEDERAL DE SÃO PAULO - UNIFESP
Instituto de Ciência e Tecnologia
Curso de Engenharia de Computação

RENAN PRADA DOS ANJOS

**SISTEMA CIBERFÍSICO PARA APLICAÇÃO NO PROCESSO DE MANIPULAÇÃO
E TRANSPORTE DE
MATERIAIS NA INDÚSTRIA 4.0**

São José dos Campos, SP
2020

RENAN PRADA DOS ANJOS

**SISTEMA CIBERFÍSICO PARA APLICAÇÃO NO PROCESSO DE MANIPULAÇÃO
E TRANSPORTE DE
MATERIAIS NA INDÚSTRIA 4.0**

Trabalho de Conclusão de Curso apresentado à disciplina de Trabalho de Graduação do curso de Engenharia de Computação do Instituto de Ciência e Tecnologia da Universidade Federal de São Paulo, como requisito parcial para obtenção do título de Engenheiro da Computação.

Orientador: Prof. Dr. Sérgio Ronaldo Barros dos Santos

São José dos Campos, SP

2020

Resumo

Este trabalho apresenta o desenvolvimento de um sistema ciberfísico para realizar as tarefas de manipulação e de transporte de materiais na indústria 4.0. O sistema ciberfísico consiste de um robô autônomo terrestre conectado a um Servidor Local, através do *framework Robot Operating System (ROS)*. O Servidor Local estabelece a ligação do robô autônomo com um Servidor *Web*, que fornece uma página para permitir ao usuário definir remotamente as tarefas que serão realizadas no ambiente proposto. O robô autônomo é equipado com rodas mecanum (suecas) conectadas em quatro motores de passo Nema 17, um braço robótico e uma garra magnética, que são acionados a partir de um sistema embarcado dedicado, baseado no microcontrolador Atmega 2560. Esse microcontrolador é conectado a um popular *Single Computer Board*, conhecido como *Raspberry Pi 3*, a partir de uma comunicação serial assíncrona *USART*, na qual fornece os comandos para executar o acionamento do robô. Por sua vez, o *Raspberry Pi 3* se conecta ao Servidor Local utilizando uma rede *Wifi* e o *framework ROS*. O sistema de localização usado para estimar a posição e a orientação do veículo consiste de um algoritmo de processamento de imagem, desenvolvido em *Python* e *OpenCV*, e uma câmera externa que permite a visualização superior do ambiente de experimentação. A navegação autônoma é realizada a partir de um sistema de controle de seguimento de trajetória, definidos por dois controladores P usados para controlar a posição e a orientação do robô. Duas abordagens para realizar a busca pelo menor caminho em grafos, conhecidas com A-estrela e Dijkstra, foram investigadas para efetuar o planejamento das trajetórias, considerando o ambiente sem nenhum obstáculo e com obstáculos fixos e móveis. O planejamento das trajetórias é realizado a partir da representação métrica (em simulação) e topológica (no cenário experimental) do mapa do ambiente. Foram definidos estudos de casos com 1, 4, 7 e 10 peças a serem manipuladas e transportadas para diferentes locais estabelecidos pelo usuário, tanto no ambiente simulado quanto no ambiente real. Os resultados em simulação demonstram a eficiência dos algoritmos implementados (A-estrela e Dijkstra) para planejar rotas em mapas métricos. Em contrapartida, todos os resultados experimentais mostram que o robô autônomo é capaz de manipular e transportar cargas entre os pontos desejados utilizando-se as rotas ótimas e o desvio de obstáculos com êxito em mapas topológicos.

Palavras-chaves: sistemas ciberfísico, manipulação e transporte, indústria 4.0, rodas mecanum, *framework ROS*, processamento de imagem, sistema de controle automático, planejamento de trajetórias.

Listas de ilustrações

Figura 1 – Fases da revolução industrial	9
Figura 2 – <i>IoT applications</i>	10
Figura 3 – Estrutura de controle em rede	11
Figura 4 – Nova estrutura de produção	11
Figura 5 – Exemplos de sistemas robóticos de transporte	12
Figura 6 – Precisão para manipular uma carga	13
Figura 7 – Detecção de obstrução no caminho	13
Figura 8 – Modelo uniciclo	17
Figura 9 – Modelo diferencial	18
Figura 10 – Roda mecanum	19
Figura 11 – Roda omnidirecional	19
Figura 12 – Carnegie Mellon Uranus, um robô omnidirecional com quatro rodas 45° mecanum motorizadas	20
Figura 13 – (a) Representação de um ambiente não mapeado (b) Representação de um mapa topológico	22
Figura 14 – Decomposição celular	23
Figura 15 – (a) Representação de um ambiente não mapeado (b) Representação de um mapa métrico	24
Figura 16 – Diagrama de blocos de um sistema de malha aberta	27
Figura 17 – Diagrama de blocos de um sistema de malha fechada	27
Figura 18 – Diagrama de blocos de um sistema PID	28
Figura 19 – Tópicos do <i>ROS</i>	30
Figura 20 – Exemplo de didático de funcionamento do <i>ROS</i>	31
Figura 21 – Diagrama de bloco geral do sistema	32
Figura 22 – Diagrama dos componentes do sistema	34
Figura 23 – Componentes do braço robótico	36
Figura 24 – Estrutura principal	36
Figura 25 – Servomotor conectado no efetuador	38
Figura 26 – Fonte e <i>Arduino</i> com a <i>shield</i>	38
Figura 27 – Bateria, <i>Raspberry</i> e sensor ultrassônico	39
Figura 28 – Guias do efetuador	39
Figura 29 – Robô desenvolvido	40
Figura 30 – Terminal do Servidor Local	41
Figura 31 – Terminal do <i>Raspberry</i>	41
Figura 32 – Criar uma tarefa	43

Figura 33 – Tarefa adicionada à lista	43
Figura 34 – Fluxo de dados ao criar uma tarefa	44
Figura 35 – Visão superior do ambiente	44
Figura 36 – Interface <i>Web</i> desenvolvida	45
Figura 37 – Inserindo um novo grafo no Dijkstra	46
Figura 38 – Diagrama planejamento de rota	48
Figura 39 – Mapa métrico	50
Figura 40 – Ocupação do grid	51
Figura 41 – Câmera C920	51
Figura 42 – Visão superior do robô	52
Figura 43 – Representação do parâmetro <i>HSV</i>	53
Figura 44 – Parâmetros <i>HSV</i> no <i>GIMP</i>	53
Figura 45 – Diagrama de blocos - Processamento de imagem	54
Figura 46 – Diagrama de blocos - controlador	56
Figura 47 – Direção de deslocamento	57
Figura 48 – Fluxograma de execução da movimentação do robô	58
 Figura 49 – Ambiente representando por um mapa métrico (grid)	61
Figura 50 – Simulação - Pontos de carga e descarga	62
Figura 51 – Simulação Dijkstra - Tempo total	63
Figura 52 – Simulação Dijkstra - Distância ideal estimada pelo algoritmo Dijkstra	64
Figura 53 – Simulação Dijkstra - Trajetória para manipulação e transporte de 4 cargas	65
Figura 54 – Simulação Dijkstra - Trajetória executada nas direções x e y	66
Figura 55 – Simulação A-estrela - Tempo total	67
Figura 56 – Simulação A-estrela - Distância ideal estimada pelo algoritmo A-estrela	67
Figura 57 – Simulação A-estrela - Trajetória para manipulação e transporte de 4 cargas	68
Figura 58 – Simulação A-estrela - Trajetória executada nas direções x e y	69
Figura 59 – Página <i>Web</i> - Criando tarefa	70
Figura 60 – Página <i>Web</i> - Tarefa concluída	70
Figura 61 – Mapa topológico - Grafo completo	71
Figura 62 – Mapa topológico - Sobreposto no ambiente	71
Figura 63 – Experimental - Tempo total	73
Figura 64 – Experimental - Erro absoluto para descarregar	73
Figura 65 – Experimental - Distância total executada pelo robô	74
Figura 66 – Experimental - Distância total ideal estimada pelo algoritmo Dijkstra	75
Figura 67 – Experimental - Trajetória executada pelo robô para manipulação e transporte de 4 cargas	76
Figura 68 – Experimental - Trajetória executada nas direções x e y, e orientação (<i>theta</i>)	77
Figura 69 – Experimental - Trajetória executada pelo robô com um obstáculo móvel	78

Figura 70 – Experimental - Trajetória executada nas direções x e y, e orientação (theta), com um obstáculo móvel	79
Figura 71 – Experimental - Tempo total com obstáculos fixos	80
Figura 72 – Experimental - Distância total percorrida com obstáculos fixos	80
Figura 73 – Experimental - Distância total ideal estimada pelo Dijkstra com obstáculos fixos	81
Figura 74 – Experimental - Trajetória sem obstáculo fixo no plano x y	82
Figura 75 – Experimental - Trajetória executada nas direções x e y, e orientação (theta), sem obstáculo fixo	83
Figura 76 – Experimental - Trajetória executada pelo robô com um obstáculo fixo	84
Figura 77 – Experimental - Trajetória executada nas direções x e y, e orientação (theta), com um obstáculo fixo	85
Figura 78 – Experimental - Trajetória executada pelo robô com dois obstáculos fixos	86
Figura 79 – Experimental - Trajetória executada nas direções x e y, e orientação (theta), com dois obstáculos fixos	87

Lista de tabelas

Tabela 1 – Servomotores - Braço	37
Tabela 2 – Servomotores - Efetuadores	37
Tabela 3 – Formato da mensagem <i>talker</i>	42
Tabela 4 – Direções que podem ser realizadas pelo robô	46
Tabela 5 – Valores aceitáveis de cores no padrão <i>HSV</i> para identificação do robô	54
Tabela 6 – Condições de controle do robô	59
Tabela 7 – Características dos ambientes utilizados	61
Tabela 8 – Categorias de testes em simulação	62
Tabela 9 – Categorias de testes experimental	72

Sumário

1	Introdução	9
1.1	Motivação	13
1.2	Objetivos	14
1.2.1	Objetivo Geral	14
1.2.2	Objetivos Específicos	14
1.3	Organização do Trabalho	14
2	Fundamentação Teórica	16
2.1	Sistemas Ciberfísicos	16
2.1.1	Robôs Móveis	16
2.1.1.1	Robô uniciclo	17
2.1.1.2	Robô diferencial	18
2.1.1.3	Robô omnidirecional	18
2.2	Navegação Autônoma	21
2.2.1	Tipos de Mapa e Representação do Ambiente	21
2.2.1.1	Mapa topológico	21
2.2.1.2	Mapa métrico	23
2.2.2	Planejamento de Caminho	24
2.2.2.1	Dijkstra	24
2.2.2.2	A-Estrela	25
2.2.3	Localização	25
2.2.4	Sistema de Controle	27
2.2.4.1	Controladores PID	28
2.3	<i>Robot Operating System</i>	29
3	Desenvolvimento do Transportador Autônomo	32
3.1	Arquitetura Geral do Sistema	32
3.1.1	Concepção do Robô Transportador	35
3.1.2	Configuração <i>ROS</i>	40
3.2	Interface <i>Web</i>	42
3.3	Planejador de Trajetória em Mapas Topográficos	45
3.3.1	Dijkstra	45
3.3.2	Desvio dos obstáculos	49
3.3.2.1	Obstáculo móvel	49
3.3.2.2	Obstáculo fixo	49
3.4	Planejador de Trajetória em Mapas Métricos	49

3.5	Localização por Visão Externa	51
3.6	Controlador de Seguimento de Trajetória	55
3.7	Controlador do braço robótico	60
4	Resultados	61
4.1	Avaliação em Simulação	61
4.1.1	Dijkstra	63
4.1.2	A-estrela	66
4.2	Avaliação Experimental	69
4.2.1	Sem obstáculos	72
4.2.2	Obstáculo móvel	78
4.2.3	Obstáculo fixo	80
4.2.3.1	Sem obstáculo fixo	81
4.2.3.2	Um obstáculo fixo	84
4.2.3.3	Dois obstáculos fixos	86
5	Conclusão e Trabalhos Futuros	88
Referências		90
Apêndices		93
APÊNDICE A Codificação do projeto		94

1 Introdução

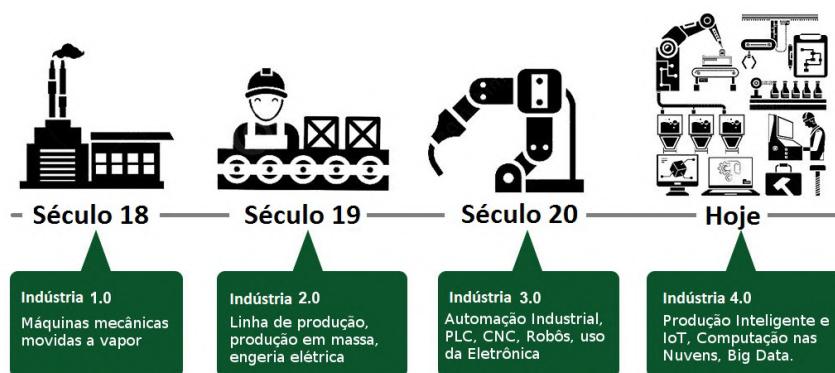
Uma grande parcela dos produtos produzidos e consumidos são industrializados, o que torna a indústria e os processos envolvidos em seus meios de produção, essenciais para a produção mundial. Com o intuito de otimizar esses processos, são feitas diversas melhorias, seja na diminuição do tempo necessário para produzir um determinado produto, na economia de insumos, ou então, na melhoria da qualidade da mercadoria, podendo inclusive possuir características específicas para cada cliente.

A indústria já sofreu inúmeras mudanças, como por exemplo a Revolução Industrial, que se iniciou com Henry Ford, nesse período a máquina a vapor trouxe uma maior mecanização à produção. Porém esse fator diminuiu a personalização de cada produto, uma vez que os artefatos eram produzidos em massa, para diminuir os custos e aumentou a quantidade de mercadorias homogêneas. Portanto a conexão entre o produto e o cliente não era muito valorizada, pois muitas vezes o consumidor não dispunha de opções de escolha para personalizar o produto ([WILSON, 2014](#)).

Agregar características de personalização na linha de montagem pode ser bastante complicado, pois muitas técnicas podem não ser aplicáveis para a produção em massa, o que pode ocasionar um aumento do custo da produção. Dessa forma, com o intuito de desenvolver produtos com características específicas e de maneira automatizada, é preciso desenvolver a tecnologia necessária para essa tarefa, onde é preciso que a linha de produção seja dinâmica e flexível, possibilitando que seus atributos sejam determinados conforme o produto.

Essa demanda do mercado poderá ser suprida pela Indústria 4.0 ([KAGERMANN; WAHLSTER; HELBIG, 2013](#)), que seria a quarta Revolução industrial, bastante relacionada com o uso da informação, como internet das coisas (*IoT - Internet of Things*), o uso de *big data* e de sistemas ciberfísicos (*CPS - Cyber Physical Systems*), como ilustrado na [Figura 1](#).

Figura 1 – Fases da revolução industrial



Fonte: INDÚSTRIA 4.0 NA TEORIA ([CARDOSO, 2018](#))

O advento da tecnologia permitiu a utilização de redes automatizadas, propiciando a comunicação com outros membros da cadeia de produção, resultando em uma automação de toda a cadeia, viabilizando uma operação de maneira descentralizada ([BRETTEL N. FRIEDERICHSSEN; ROSENBERG, 2014](#)), como é retratado na [Figura 2](#).

Figura 2 – *IoT applications*

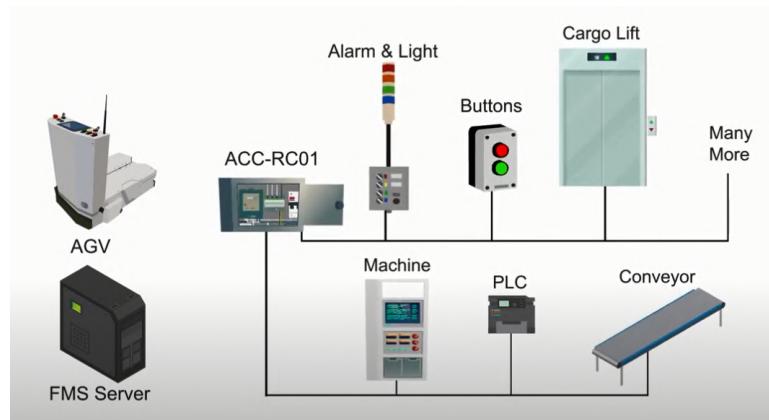


Fonte: INDÚSTRIA 4.0 NA TEORIA ([CARDOSO, 2018](#))

Essas novas possibilidades de inovação na produção acarretam em mudanças nos mercados, modificando o fluxo de produção e serviços, alterando as economias e estruturas de muitos setores, modificando também o mercado de trabalho e os hábitos de consumo ([DRUCKER, 2000](#)).

O uso de *CPS* permite que dispositivos atuem de maneira coordenada, uma vez que estão conectados em rede, é possível aplicar algoritmos capazes de gerir os dispositivos de forma eficiente, com o intuito de atender as necessidades do mercado. Dessa forma é necessário o desenvolvimento desses sistemas, buscando facilitar a integração dos mecanismos utilizados e melhorar a estrutura de coordenação de toda a plataforma, facilitando a programação e gerenciamento. Na [Figura 3](#) vemos diversos dispositivos integrados, o que pode aumentar a complexidade da rede e do controle.

Figura 3 – Estrutura de controle em rede



Fonte: *Autonomous Mobile Robot (AMR) for Industry 4.0 e Smart Manufacturing* ([YOUTUBE, b](#))

O deslocamento e transporte de cargas constituem uma etapa fundamental em qualquer linha de produção, com o advento de novas demandas, que a indústria 4.0 busca suprir, esteiras que eram tradicionalmente utilizadas nesse processo, se tornam insuficientes, sendo necessário o desenvolvimento de meios de transporte versáteis e integrados com outros dispositivos.

Essa lógica também se aplica a casos onde não existia tecnologia suficiente para automatização ou onde a automação não seria viável economicamente, entretanto, novas tecnologias tornaram isso possível, resultando em um processo mais eficiente e, em alguns casos, mais seguro por evitar acidentes e lesões pelo transporte de cargas por pessoas, como é apresentado na [Figura 4](#).

Figura 4 – Nova estrutura de produção



Fonte: *An IIoT based Smart Robotic Warehouse Management System for Industry 4.0* ([YOUTUBE, g](#))

Existe uma variedade muito grande de sistemas robóticos de transportes, de acordo com as necessidades do ambiente e das tarefas a serem executadas em cada caso, na [Figura 5](#) podemos ver alguns exemplos.

Figura 5 – Exemplos de sistemas robóticos de transporte

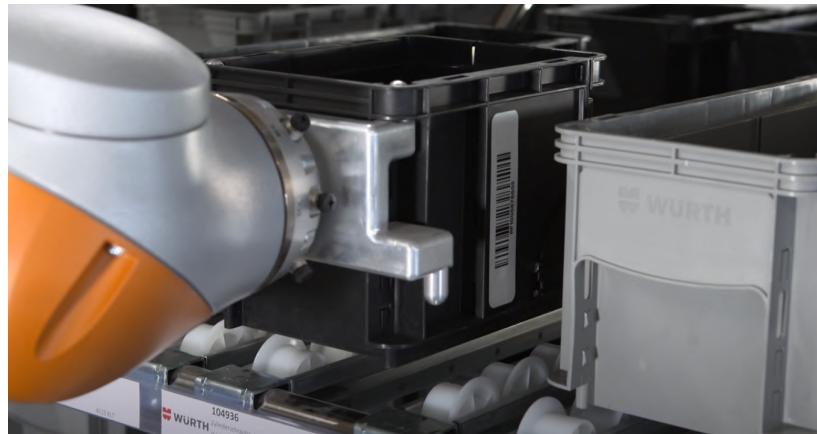


Fontes ([YOUTUBE, d](#)) ([YOUTUBE, c](#)) ([YOUTUBE, e](#)) ([YOUTUBE, j](#)) ([YOUTUBE, i](#)) ([YOUTUBE, k](#)) ([YOUTUBE, h](#)) ([YOUTUBE, a](#)) ([YOUTUBE, f](#))

Mesmo com uma variedade muito grande de tamanhos, formatos, estruturas e funcionalidades, podemos definir algumas características comuns em praticamente todos os robôs apresentados:

- Capacidade de se localizar e se locomover no ambiente: É necessário ser capaz de estimar sua posição e velocidade para realizar a navegação no ambiente e execução da tarefa desejada. Outro ponto fundamental é a capacidade de processamento e comunicação em rede, onde é preciso ter um bom fluxo de dados para obter um bom desempenho.
- Buscar, transportar e descarregar cargas: É fundamental que consiga realizar a navegação pelo ambiente e o posicionamento no local de manipulação de forma precisa para executar a carga e descarga, como visto na [Figura 6](#). Caso ocorra algum erro, além de impossibilitar a execução da tarefa, também pode resultar em danos gerais.

Figura 6 – Precisão para manipular uma carga



Fonte: *Autonomous transportation with mobile robot KMR iiwa* ([YOUTUBE](#), [d](#))

- Percepção do ambiente (uso de sensores embarcados ou externos): De extrema importância em todo sistema de locomoção, além de evitar danos materiais, também pode evitar danos físicos (considerando uma linha de produção compartilhada entre máquinas e pessoas), como visto na [Figura 7](#).

Figura 7 – Detecção de obstrução no caminho



Fonte: *Autonomous transportation with mobile robot KMR iiwa* ([YOUTUBE](#), [d](#))

1.1 Motivação

A indústria 4.0 é um novo modelo de produção que deve se desenvolver conforme novas tecnologias são criadas e integradas, portanto, o sistema descrito nesse trabalho, possui um potencial de inovação e pode servir de base para o desenvolvimento e estudo de melhorias para o sistema de manipulação e transporte de cargas.

O sistema ciberpísico aplicado no processo de manipulação e transporte de materiais na indústria 4.0, pode acelerar a distribuição de mercadorias, diminuir a necessidade de mão de

obra humana e automatizar meios de produção, além do potencial de tornar alguns processos mais seguros e com maior qualidade ([JAZDI, 2014](#)).

1.2 Objetivos

1.2.1 Objetivo Geral

O objetivo geral desse trabalho é desenvolver um sistema ciberfísico dentro dos conceitos da Indústria 4.0 para realizar tarefas específicas de manipulação e transporte de cargas, definidas por um usuário conectado remotamente ao sistema. Os requisitos das tarefas a serem executadas serão estabelecidos pelos próprios clientes através de um sistema disponível em rede. As tarefas serão executadas por uma plataforma robótica móvel que deve permanecer conectada em rede com o Servidor Local, que enviará informações oriundas de diversos sensores e receberá instruções, estabelecendo um sistema com retroalimentação.

1.2.2 Objetivos Específicos

- a) Delinear teoricamente o problema;
- b) Construir a plataforma robótica móvel constituído por hardware e software;
- c) Projetar os controladores de baixo nível para realizar a navegação da plataforma durante a realização das tarefas;
- d) Estabelecer a comunicação entre o Servidor Local, a plataforma móvel e os sensores disponíveis no ambiente, utilizando o *framework ROS (Robot Operating System)*;
- e) Implementar os planejadores de trajetória e o gerenciador da tarefa integrado com o sistema ciberfísico;
- f) Conectar o sistema ciberfísico à rede, a partir de um Servidor *Web*, permitindo a interação do usuário de maneira remota.

1.3 Organização do Trabalho

Os capítulos a seguir estão estruturados da seguinte forma:

- No Capítulo 2 são apresentados os principais conceitos teóricos utilizados na implementação do projeto.
- No Capítulo 3 é apresentado o desenvolvimento de todo o sistema proposto.
- No Capítulo 4 podemos observar os resultados obtidos a partir dos testes propostos e executados.

- No Capítulo 5 é apresentada a conclusão do trabalho e sugestões de aprimoramentos para o projeto.

2 Fundamentação Teórica

2.1 Sistemas Ciberfísicos

Os Sistemas Ciberfísicos, ou *Cyber Physical Systems (CPS)*, buscam criar uma estrutura que seja totalmente conectada em rede, unindo os atributos físicos com os sistemas virtuais, dessa maneira permite que sistemas inteligentes se comuniquem e interajam entre si e com o ambiente, possibilitando que uma determinada tarefa seja realizada em grupo, com todos os seus integrantes em sincronia, propiciando um desenvolvimento e um resultado melhor do que se a tarefa fosse realizada individualmente. Em comparação com os sistemas embarcados, os *CPS* buscam transformar esses sistemas e integrar esses dispositivos em rede, aumentando a gama de atividades que esses sistemas conseguem exercer ([MACDOUGALL, Alemanha, 2014](#)).

Segundo JAZDI ([JAZDI, 2014](#)), os *CPS* são sistemas embarcados com a funcionalidade de se comunicar em rede. Pode-se dizer que possuem uma unidade de controle, constituída por um ou mais microcontroladores, esses microcontroladores enviam e recebem dados de diversos sensores e atuadores, além de serem responsáveis pelo processamento dos dados. Outra característica importante é a necessidade de uma interface para a comunicação em rede entre os microcontroladores e outros sistemas em nuvem ([JAZDI, 2014](#)).

A integração entre esses dispositivos gera dependências, fazendo com que o funcionamento dos dispositivos dependam de outros sistemas, que devem participar do processo e responder de maneira síncrona e rápida, de modo que não atraso o processo.

Apesar da complexidade da implementação desses sistemas, o uso e desenvolvimento dessa tecnologia favorece o surgimento de soluções totalmente inovadoras e com impactos de grande significância, uma vez que viabiliza o surgimento de novos modelos de negócios e soluções inovadoras.

2.1.1 Robôs Móveis

A ideia de um robô móvel se baseia na sua capacidade de se locomover pelo ambiente em que se encontra, uma característica muito importante desse dispositivo são os atuadores (motor de passo, servomotores, caixa de engrenagens, entre outros) utilizados que devem permitir a navegação pelo ambiente, os quais devem ser escolhidos conforme a aplicação desejada ([SIEGWART; NOURBAKHSH, 2004](#)). Existem diversas aplicações possíveis, dependendo do ambiente escolhido, como podemos citar o ambiente aquático, terrestre, aéreo ou uma combinação entre eles e também pode variar conforme o objetivo desejado.

Mesmo restringindo o nosso foco para a classe dos robôs terrestres, ainda sim é possível

citar alguns tipos de robôs, como a utilização de acessórios como pernas ou rodas, podendo ainda variar em quantidade (robôs com 2 ou 4 pernas ou rodas, por exemplo). Nesse trabalho iremos nos concentrar nos robôs terrestres e com rodas.

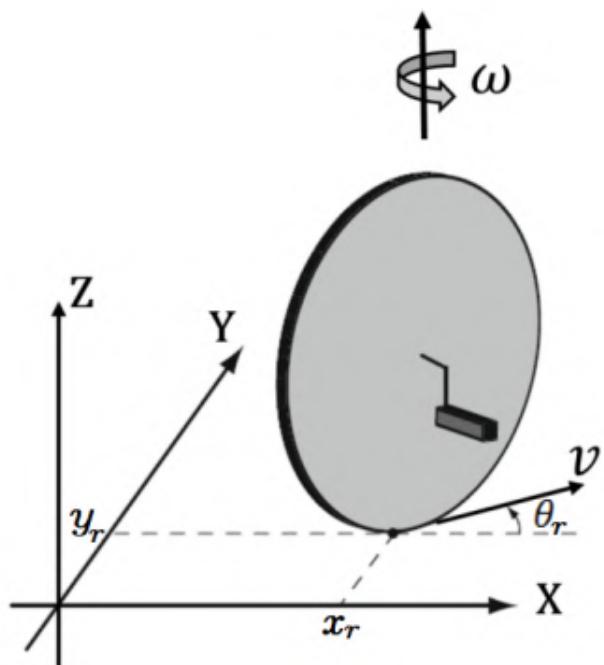
2.1.1.1 Robô uniciclo

Uma forma de classificar os robôs é referente a quantidade de graus de liberdade que ele possui em determinado ambiente. No modelo uniciclo, o robô possui apenas uma roda e dois eixos de liberdade, referente a velocidade linear e a velocidade angular ([SIEGWART; NOURBAKHSH, 2004](#)).

Na [Figura 8](#) vemos um modelo simplificado do robô uniciclo, onde observamos os eixos de translação v e rotação ω sobre o robô representado por uma roda cinza, utilizando como referência o sistema de eixos x,y,z. Nessa representação foram consideradas as premissas abaixo:

- a roda permaneça sempre perpendicular em relação ao chão;
- não ocorre deslizamentos;
- a superfície é plana;
- a roda não sofre deformações.

Figura 8 – Modelo uniciclo



Fonte: FRANCIS e MAGGIORE ([FRANCIS, 2016](#))

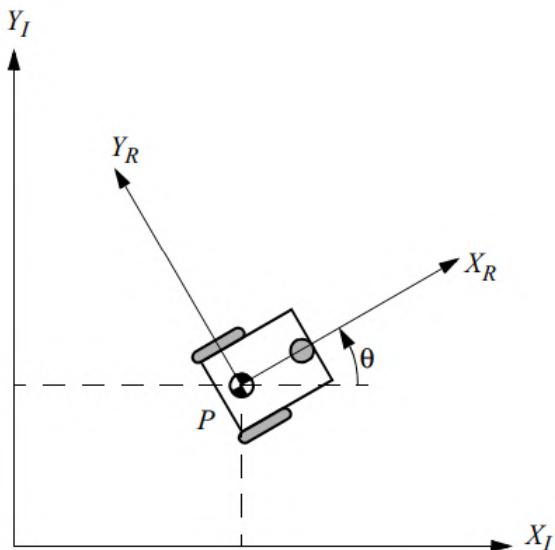
2.1.1.2 Robô diferencial

Esse modelo é bastante comum, sendo composto por duas rodas em um mesmo eixo. Essa estrutura com duas rodas é o mínimo necessário para se obter a estabilidade estática, desde que seu centro de massa esteja logo abaixo do eixo da roda.

Para se obter maior estabilidade durante o movimento, além das rodas, é comum a utilização de um ou dois pontos de contato com o solo utilizando esferas deslizantes, conforme a necessidade da aplicação ([SIEGWART; NOURBAKHSH, 2004](#)).

Com o uso de um motor em cada roda, é possível controlar cada uma independentemente, fazendo com que a variação da velocidade entre as duas rodas faça o robô girar em torno de seu Centro Instantâneo de Rotação (CIR), que fica localizado na metade das distâncias entre as rodas, no eixo. Dessa forma, ambas as rodas rotacionam com uma mesma velocidade angular em torno do CIR. Podemos visualizar essa representação na [Figura 9](#)

Figura 9 – Modelo diferencial



Fonte: SIEGWART e NOURBAKHSH ([SIEGWART; NOURBAKHSH, 2004](#))

2.1.1.3 Robô omnidirecional

Os robôs omnidirecionais podem realizar movimentos longitudinais e laterais por meio do uso de rodas específicas, podemos citar as rodas mecanum (também conhecidas como suecas) e as rodas omnidirecionais.

A roda mecanum é composta por um ponto de fixação (hub), onde o motor é conectado, e apresenta roletes distribuídos igualmente e inclinados em um ângulo de 45° na roda. Além dos roletes rotacionarem junto com a roda, eles também conseguem girar livremente em torno de seu próprio eixo. Sua configuração é vista na [Figura 10](#).

Figura 10 – Roda mecanum



As rodas omnidirecionais possuem um funcionamento semelhante às rodas mecanum, entretanto os roletes possuem um ângulo de 90° em relação ao ponto de fixação, como é visto na [Figura 10](#).

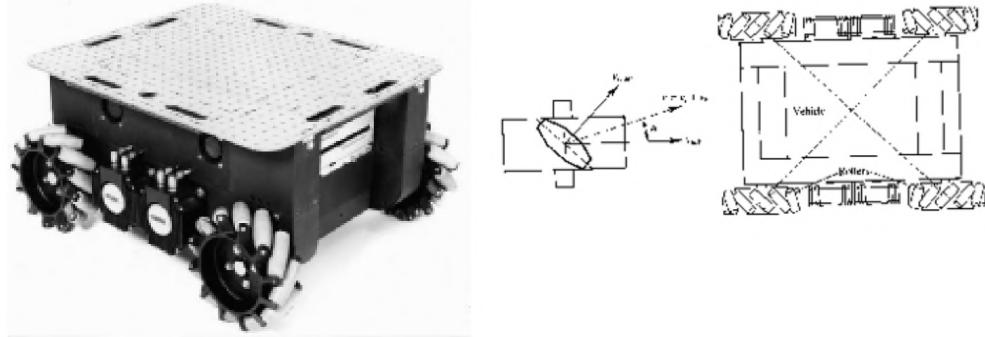
Figura 11 – Roda omnidirecional



Fonte: ALIEXPRESS ([ALIEXPRESS](#),)

Ambas as rodas permitem realizar, além do movimento longitudinal, que é possível realizar com rodas convencionais, também é possível efetuar movimentos nas diagonais, para os lados e curvas (rotação e translação). Podemos ver uma configuração com quatro rodas mecanum na [Figura 12](#).

Figura 12 – Carnegie Mellon Uranus, um robô omnidirecional com quatro rodas 45° mecanum motorizadas



Fonte: SIEGWART e NOURBAKHSH ([SIEGWART; NOURBAKHSH, 2004](#))

Podemos representar a movimentação desse modelo, com a equação da cinemática direta, definida pela igualdade vista na [Equação 2.1](#).

$$\begin{bmatrix} w1 \\ w2 \\ w3 \\ w4 \end{bmatrix} = \frac{1}{r} \begin{bmatrix} 1 & -1 & -(lx + ly) \\ 1 & 1 & (lx + ly) \\ 1 & 1 & -(lx + ly) \\ 1 & -1 & (lx + ly) \end{bmatrix} \begin{bmatrix} vx \\ vy \\ wz \end{bmatrix} \quad (2.1)$$

Onde:

- $w1, w2, w3, w4$ representam a velocidade angular de cada roda;
- r é o raio das rodas utilizadas;
- lx representa a distância entre o centro do robô até sua extremidade em x;
- ly representa a distância entre o centro do robô até sua extremidade em y;
- vx é a velocidade linear em x;
- vy é a velocidade linear em y;
- wz é a velocidade angular do robô.

A [Equação 2.1](#) pode ser reescrita de modo a encontrar o valor da velocidade angular em cada roda, como é visto na [Equação 2.2](#).

$$\begin{cases} w1 = \frac{1}{r}(vx - vy - (lx + ly)wx) \\ w2 = \frac{1}{r}(vx - vy - (lx + ly)wx) \\ w3 = \frac{1}{r}(vx - vy - (lx + ly)wx) \\ w4 = \frac{1}{r}(vx - vy - (lx + ly)wx) \end{cases} \quad (2.2)$$

Desenvolvendo a cinemática inversa, temos como resultado a [Equação 2.3](#) vista a seguir.

$$\begin{bmatrix} vx \\ vy \\ wz \end{bmatrix} = \frac{r}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ -1 & 1 & 1 & -1 \\ -\frac{1}{(lx+ly)} & \frac{1}{(lx+ly)} & -\frac{1}{(lx+ly)} & \frac{1}{(lx+ly)} \end{bmatrix} \begin{bmatrix} w1 \\ w2 \\ w3 \\ w4 \end{bmatrix} \quad (2.3)$$

Do mesmo modo que foi feito para a cinemática direta, ao reescrevermos a [Equação 2.3](#), podemos definir as velocidades lineares e também a velocidade angular do robô. Essa relação é vista na [Equação 2.4](#).

$$\begin{cases} vx = (w1 + w2 + w3 + w4) \frac{r}{4} \\ vy = (-w1 + w2 + w3 - w4) \frac{r}{4} \\ wz = (-w1 + w2 - w3 + w4) \frac{r}{4(lx+ly)} \end{cases} \quad (2.4)$$

2.2 Navegação Autônoma

Por se tratar de um sistema complexo e que necessita de diversas funcionalidades, o sistema de navegação autônoma é composto por diversos subsistemas, que serão apresentados nessa seção, que ao trabalhar de maneira conjunta, conseguem realizar com êxito a locomoção do robô pelo ambiente em que se encontra. Dentre essas funcionalidades podemos citar por exemplo a representação do ambiente, o planejamento de rota, a localização do robô e também o sistema de controle. A seguir iremos ver mais detalhes sobre os subsistemas citados.

2.2.1 Tipos de Mapa e Representação do Ambiente

Existem alguns tipos de mapas que podem ser utilizados, essa escolha deve ser definida de acordo com o ambiente em questão e a funcionalidade (aplicação) do robô, como por exemplo em casos em que é preciso mapear todo o ambiente para limpeza e outros em que é preciso conhecer apenas o trajeto que será utilizado, não sendo preciso ter conhecimento de todo o ambiente (além do trajeto).

Certas características devem ser consideradas na escolha do mapa, de maneira geral os mapas devem buscar ser o mais preciso possível, sendo capaz de representar as características do ambiente e a complexidade da representação deve ser compatível com a capacidade computacional utilizada ([SIEGWART; NOURBAKHSH, 2004](#)).

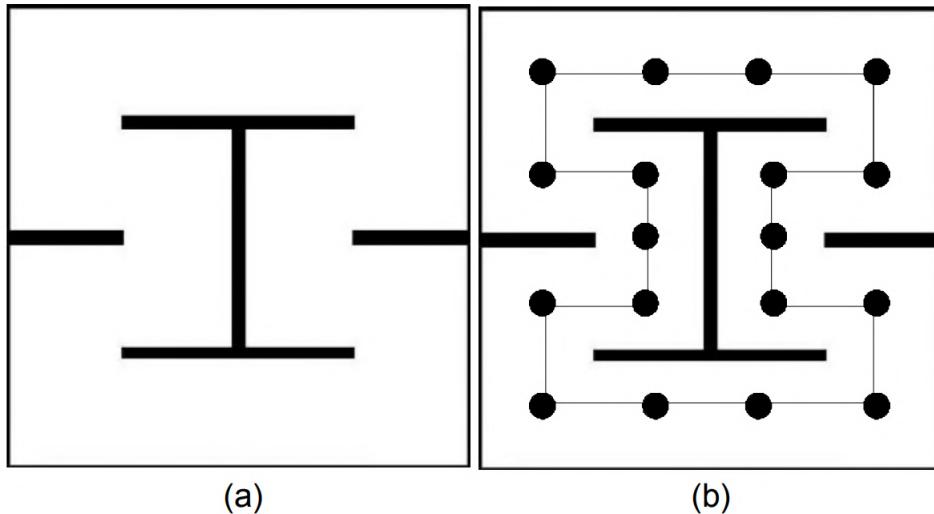
2.2.1.1 Mapa topológico

Este tipo de mapa não possui informações geométricas do ambiente, são representadas as conexões entre os diversos locais ou objetivos desejados e são caracterizados por arestas, que são conectados aos nós. Quando é preciso inferir mais informações sobre o ambiente, podem ser

utilizadas marcações, como por exemplo código QR. Essas marcações são chamadas de marcos ([THRUN, Feb. 2002](#)).

Portanto o mapa topológico é composto por um grafo, cada nó representa um ponto de interesse e, quando existe um caminho entre os nós, eles são conectados por uma aresta, como podemos ver na [Figura 13](#).

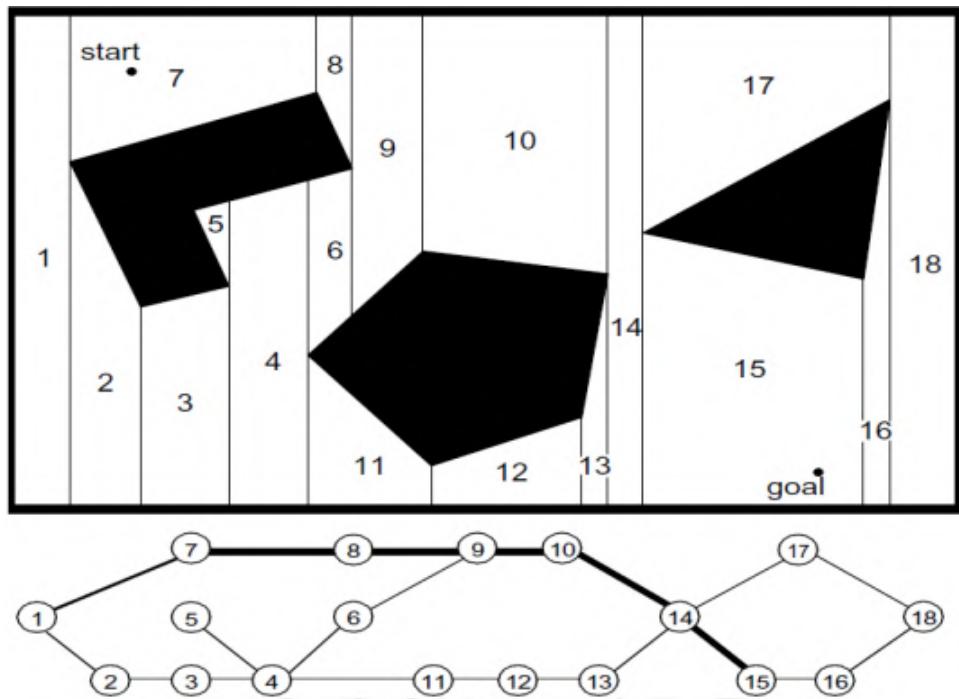
Figura 13 – (a) Representação de um ambiente não mapeado (b) Representação de um mapa topológico



Fonte: OLIVEIRA ([OLIVEIRA, 2010](#))

A representação por grafo pode se derivar da decomposição celular. A partir do ambiente, são definidas as áreas livres por onde o robô móvel pode transitar, essas áreas livres são divididas em setores (retas verticais) e então cada setor é representado por um nó do grafo e então, os setores que fazem fronteira entre si, são conectados por arestas. Dessa forma obtemos o grafo que representa o mapa topológico desse ambiente, como podemos ver o exemplo exposto na [Figura 14](#).

Figura 14 – Decomposição celular

Fonte: LATOMBE ([LATOMBE, 1991](#))

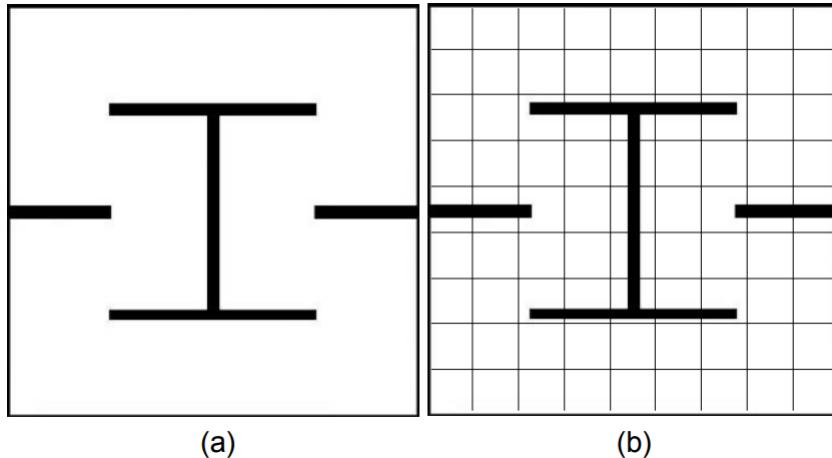
2.2.1.2 Mapa métrico

Diferentemente do item anterior, os mapas métricos transmitem informações referentes a geometria do ambiente ([THRUN, Feb. 2002](#)). Com essa característica, é possível gerar mapas bastante precisos mas, em contrapartida, o planejamento da rota mais eficiente se torna mais complexo e isso pode se agravar em ambientes maiores visto que, em alguns casos, o custo computacional pode aumentar de maneira exponencial.

De maneira semelhante a representação de uma imagem digital, que possui uma definição de acordo com a quantidade de pixels, a representação desse mapa é, na maioria das vezes, feita pelo uso de grades que são compostas por células uniformes que dividem todo o plano. Portanto, quanto mais células, mais detalhes podem ser representados. No caso de uma representação com pouca definição (poucas células por área), o mapa pode gerar ocorrências inconsistentes, como por exemplo eliminar uma passagem estreita.

Cada célula armazena apenas uma informação, se é um ponto por onde o robô pode passar (passagem livre) ou se está obstruído, como é visto na [Figura 15](#). O robô só pode se locomover pelas células que não possuem obstáculo, os obstáculos são representados em preto.

Figura 15 – (a) Representação de um ambiente não mapeado (b) Representação de um mapa métrico



Fonte: OLIVEIRA ([OLIVEIRA, 2010](#))

2.2.2 Planejamento de Caminho

O processo para encontrar um caminho pode ser dividido em duas etapas, a primeira é referente ao mapa que deve representar o ambiente, o qual foi abordado na seção anterior, a segunda etapa envolve o uso de algoritmos que devem encontrar rotas ótimas que permitam ao robô percorrer o ambiente e encontrar seu objetivo, esses algoritmos serão abordados a seguir.

Portanto, a partir do mapa e com pelo menos um destino, o robô deve planejar suas ações a longo prazo, de modo que consiga atingir seu alvo. É necessário identificar a melhor rota (a rota ótima), entre várias outras rotas possíveis, permitindo que o robô, partindo do ponto inicial, chegue em seu objetivo ([SIEGWART; NOURBAKHSH, 2004](#)).

2.2.2.1 Dijkstra

O algoritmo Dijkstra utiliza-se de uma estratégia gulosa. Com essa estratégia ele irá buscar o vértice seguinte mais próximo, o qual deve ser adicionado ao conjunto solução. Entretanto, no caso de arestas com pesos negativos, não é garantido que será encontrado o caminho ótimo ([CORMEN et al., 2009](#)).

De modo geral o funcionamento do algoritmo não é complexo, sempre que um novo vértice é escolhido, ele é definido como um novo ponto de partida para encontrar o vértice seguinte mais próximo, até chegar ao objetivo desejado. Contudo essa é uma busca não-informada, o que pode resultar em um elevado desperdício de processamento do sistema computacional usado para executar o algoritmo, uma vez que todos os possíveis caminhos equidistantes serão testados. Segundo Cormen ([CORMEN et al., 2009](#)) esse algoritmo possui, tradicionalmente, um custo de $O(E \log V)$, sendo E o número de arestas e V o número de vértices.

2.2.2.2 A-Estrela

Esse algoritmo é a solução mais utilizada para buscas de melhor escolha em grafos (RUSSELL; NORVIG, 2003), sua busca consiste em utilizar funções heurísticas (busca informada), baseando-se na distância euclidiana entre o nó atual e o objetivo desejado, para então decidir o próximo nó que será visitado (DJOJO; KARYONO, 2013). A distância euclidiana $d(a, b)$ é definida pela Equação 2.5.

$$d(a, b) = \sqrt{(Xa - Xb)^2 + (Ya - Yb)^2} \quad (2.5)$$

Visto que analisar todos os caminhos possíveis de um grafo pode se tornar algo bastante custoso, a utilização da função heurística possui o intuito de diminuir o tamanho da árvore de pesquisa gerada, fazendo com que exista uma priorização na ordem de escolha dos ramos testados, resultando em uma maior rapidez para encontrar o caminho para o objetivo desejado. Portanto esse algoritmo realiza uma busca guiada.

2.2.3 Localização

A partir de uma representação satisfatória do ambiente, os algoritmos de planejamento de caminho também precisam obter informações a respeito da localização do robô. Dessa forma é possível realizar uma navegação autônoma adequada, percorrendo a trajetória calculada pelo algoritmo. A localização é dada a partir de pontos de interesse em relação a um referencial global (THRUN, Feb. 2002).

De maneira geral, podemos dividir as técnicas para auto-localização em dois grupos:

- Posicionamento relativo: se baseia em informações locais do robô, obtidas a partir de sensores de proximidade, câmeras embarcadas e odômetros (Betke; Gurvits, 1997).
- Posicionamento absoluto: se baseia em informações de sensores como receptor GPS ou câmeras externas usando marcadores coloridos ou reflexivos fixados no corpo do veículo, possuindo como referência localizações conhecidas no ambiente (Betke; Gurvits, 1997).

A escolha sobre qual técnica utilizar, depende das características do ambiente e também do objetivo desejado, o receptor GPS possui um erro de +/- 15 m e pode não funcionar em ambientes internos como em fábricas ou indústrias, o seu uso também pode se tornar impraticável em ambientes sub-aquáticos (Smith et al., 1997), sendo necessário levar o robô até a superfície para obter sua localização. De maneira geral, o uso de câmeras externas proporcionam uma maior precisão. Existem também robôs que se utilizam das duas técnicas em conjunto, obtendo assim uma localização mais precisa.

A seguir iremos destacar três sensores que são bastante aplicados em localização em ambientes internos e externos, também será abordado brevemente o seu funcionamento e características:

- Odômetro: é utilizado para estimar a velocidade do robô a partir da quantidade de rotações feitas por cada roda, é preciso utilizar um sensor com precisão suficiente para contabilizar todas as rotações. Com isso também é possível obter a distância percorrida até o momento e também estimar a localização do robô em relação a sua posição inicial conhecida, essa técnica se chama *dead-reckoning*, entretanto deve-se atentar a alguns problemas que podem afetar a sua precisão, como o alinhamento não satisfatório das rodas, erro de sinal nos sensores e a falta de atrito (deslizamento) com a superfície, podendo gerar uma rotação sem um movimento efetivo. Por esses motivos apresentados, esse processo de estimativa de localização pode se tornar insuficiente e muito impreciso ([THRUN, Feb. 2002](#)).
- Câmera: muito utilizada em situações de ambiente fechado, é constituída por lentes que direcionam a luz para um receptor que registra a imagem, a definição da imagem depende da quantidade de pixels que esse receptor possui, entretanto a qualidade da imagem também pode ser afetada pela qualidade das lentes e o foco. Para registro de vídeos as câmeras podem possuir diferentes frequências de captura, como por exemplo 30 HZ ou 60 Hz.

É possível aplicar técnicas de processamento de imagem para identificar marcadores fixados no corpo do veículo. É necessário um poder de processamento do sistema computacional adequado, de acordo com a aplicação desejada, pois caso a velocidade de resposta seja muito lenta, sua utilização pode se tornar inviável.

- Sonar: é composto por um emissor e um receptor de ondas ultrassônicas que são usados para determinar a distância até um obstáculo a partir do intervalo de tempo necessário para o som se propagar no ambiente, atingir o obstáculo e então retornar e ser detectado pelo receptor. Considerando a velocidade do som no ambiente (343 m/s no ar e 1480 m/s na água), a distância d pode ser calculada pela multiplicação da velocidade v (m/s) pelo tempo t (s), como visto na [Equação 2.6](#).

$$d = v \cdot t \quad (2.6)$$

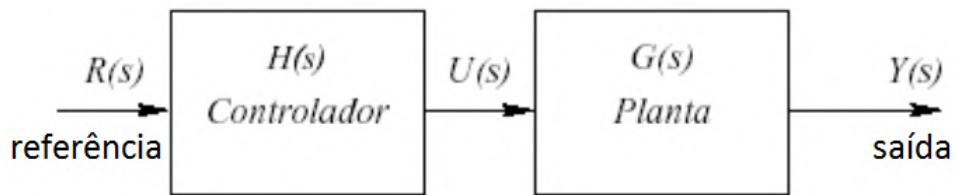
Esse sensor não necessita de um poder de processamento tão elevado quanto o processamento de imagem e, por se tratar de um sensor de baixo custo e de fácil utilização, é bastante empregado, principalmente, em robôs de ambientes fechados. Também existem diversas aplicações em ambientes abertos, entretanto é preciso considerar as limitações de alcance do sensor, devido a dissipação do som no ambiente ser exponencial em relação a distância.

2.2.4 Sistema de Controle

Um sistema de controle é definido por um conjunto de dispositivos que coordenam o comportamento de outros sistemas ou máquinas, de maneira geral um controlador é desenvolvido a partir da modelagem matemática da planta, permitindo um controle preciso e refinado (NISE, 2012).

Os sistemas de controle podem ser classificados como sistemas de malha aberta e sistemas de malha fechada. Os sistemas de malha aberta não realizam a leitura do sinal de saída do próprio sistema para comparar com a entrada, ou seja, não é feito o processo de realimentação do sistema, dessa forma o controle do sistema não é influenciado pelo sinal de saída (OGATA, 2015). Na Figura 16 é visto um diagrama de blocos de um sistema de malha aberta, onde a entrada $R(s)$ alimenta diretamente o bloco com a função de controle $H(s)$ e então, é obtido o sinal de controle $U(s)$ que irá controlar o sistema $G(s)$, entretanto a saída $Y(s)$ do sistema não é utilizada para realimentar o sistema.

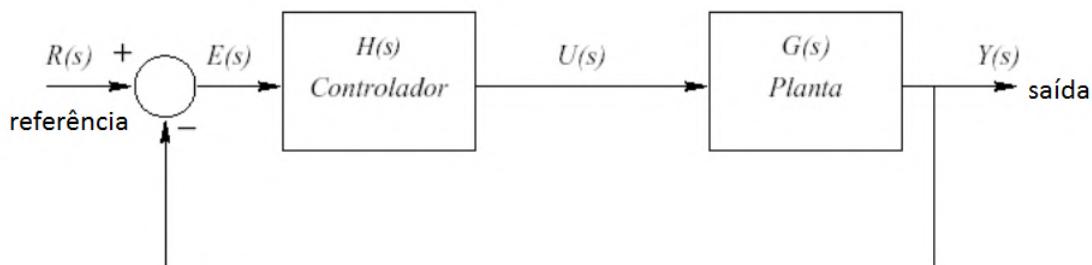
Figura 16 – Diagrama de blocos de um sistema de malha aberta



Fonte: COELHO (COELHO, 2012)

Os sistemas de malha fechada, como pode ser visto na Figura 17, a saída $Y(s)$ realiza a realimentação do sistema pelo somador, sendo comparado com a entrada $R(s)$. O resultado do somador $E(s)$ alimenta o bloco com a função de controle $H(s)$, que a partir do sinal $U(s)$ irá controlar o sistema $G(s)$. Nesse ponto o ciclo se repete, onde a resposta $Y(s)$ do sistema é novamente lida para realimentar o sistema.

Figura 17 – Diagrama de blocos de um sistema de malha fechada



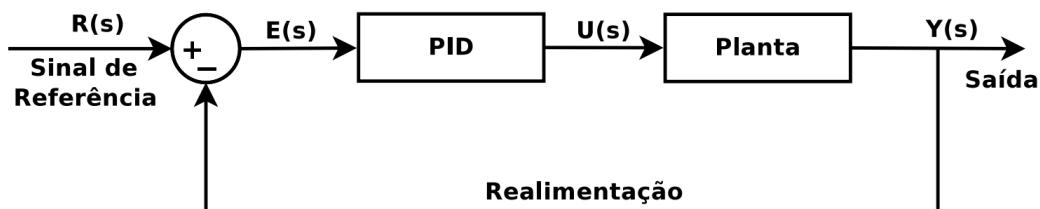
Fonte: COELHO (COELHO, 2012)

2.2.4.1 Controladores PID

Controladores PID (Proporcional, Integral e Derivativo) representavam mais da metade dos controladores em uso na indústria no ano de 2015 ([OGATA, 2015](#)), tornando esse algoritmo o mais utilizado no mundo para controle industrial. Fatores como facilidade de implementação, baixo custo e versatilidade, junto com um desempenho robusto em diversas aplicações, resultam em uma operação de maneira simples e direta, o que justifica essa grande popularidade desse tipo de controladores ([ÅSTRÖM; HÄGGLUND, 2006](#)).

Os controladores PID utilizam o sinal de erro como entrada que é constituído pela comparação entre o sinal de saída do sistema e o sinal de referência aplicado, estabelecendo assim o sistema em malha fechada, como visto na [Figura 18](#). Utilizando o sinal de erro gerado pelo comparador, o controlador PID gera um sinal de controle adequado para a planta (sistema a ser controlado), como por exemplo a velocidade de um automóvel, a altitude de um avião, a trajetória de um robô terrestre, dentre outras aplicações.

Figura 18 – Diagrama de blocos de um sistema PID



Fonte: MIRANDA ([MIRANDA, 2017](#))

A partir do erro $E(s)$ o controlador gera o sinal $U(s)$. Esse sinal é definido a partir do sinal de controle no domínio do tempo $u(t)$, que é obtido a partir da soma dos três termos do controlador, visto na [Equação 2.7](#).

$$u(t) = u_p(t) + u_i(t) + u_d(t) \quad (2.7)$$

Esses termos são definidos como:

1. Ganho proporcional: obtido a partir da multiplicação entre o erro $e(t)$ e a constante K_p , que é apresentado na [Equação 2.8](#). Um aumento no erro pode ser observado em regime estacionário e no sobressinal máximo. Em valores de ganhos maiores, o sistema tende a se tornar mais oscilatório, o que pode gerar instabilidade ([ÅSTRÖM; HÄGGLUND, 2006](#)).

$$u_p(t) = K_p e(t) \quad (2.8)$$

2. Ganho integral: obtido pela multiplicação da integral do erro pela constante K_i , definido pela [Equação 2.9](#). Garante que o erro seja igual a zero em um regime estacionário.

$$u_i(t) = K_i \int_0^t e(\tau) d\tau \quad (2.9)$$

3. Ganho derivativo: onde $u_d(t)$ é definido pela multiplicação da constante K_d pela derivada do erro, definido pela [Equação 2.10](#). Possui a função de melhorar a estabilidade, reduzir o sobressinal máximo e diminuir o tempo de resposta do sistema.

$$u_d(t) = K_d \frac{de(t)}{dt} \quad (2.10)$$

Portanto o controlador PID pode ser definido pela seguinte [Equação 2.11](#).

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt} \quad (2.11)$$

As constantes K_p , K_i e K_d são valores que geralmente são ajustados em campo, o processo de selecionar esses parâmetros é chamado de sintonia do controlador e existem diferentes métodos na literatura para realizar esse procedimento, como por exemplo as regras de Ziegler-Nichols as quais sugerem que as constantes devem gerar uma resposta estável do sistema, de maneira experimental. O método descrito fornece valores estimados para os parâmetros, sendo necessário realizar uma sintonia fina posteriormente ([OGATA, 2015](#)).

2.3 Robot Operating System

O *Robot Operating System* é um *framework* que facilita a integração entre dispositivos, como sensores e atuadores, composto por ferramentas e bibliotecas que propiciam o desenvolvimento de sistemas robóticos de diferentes complexidades ([CERIANI; MIGLIAVACCA, 2015](#)). Os processos que compõem a rede *ROS* são representados por um grafo de computação, possibilitando a comunicação entre os nós conectados.

O *ROS* também permite o uso de diferentes linguagens de programação, como *Python*, *C* e *C++*. Suas bibliotecas facilitam a integração com diferentes aplicações, como *OpenCV*¹, e realiza a modularização dos componentes do sistema, resultando em uma abstração dos dispositivos, principalmente do hardware. Todos esses fatores contribuem para uma maior agilidade no desenvolvimento dos projetos ([JUNIOR, 2016](#)).

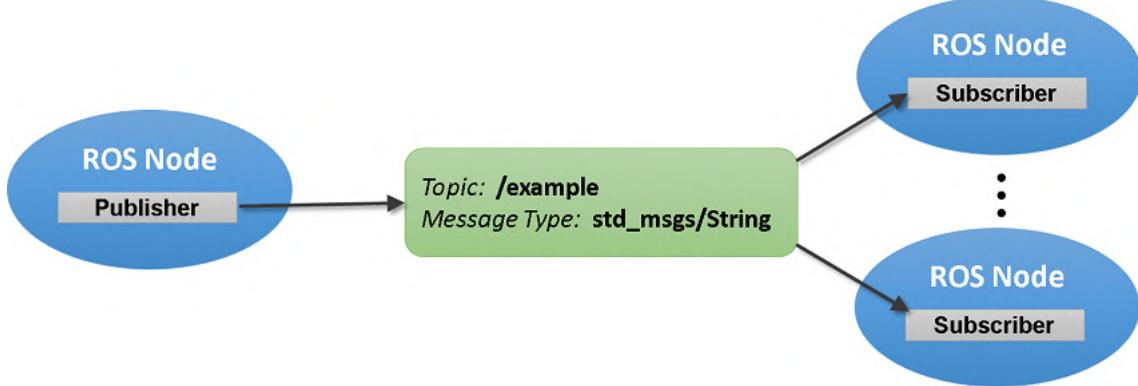
A seguir iremos abordar as principais estruturas que compõem o funcionamento do *ROS*:

- Nós²: são processos responsáveis por um determinado evento. Um sistema robótico pode conter diversos nós, sendo um responsável pelo sensor de proximidade, outro pela atuação dos motores e um outro pela localização, por exemplo.

¹ <https://opencv.org/about/>

² <http://wiki.ros.org/Nodes>

- Tópicos³: os nós se comunicam por meio de mensagens, para enviar uma mensagem o nó deve publicá-la em um tópico e para receber é preciso subscrever, portanto é possível publicar ou subscrever em um tópico. Na [Figura 19](#) temos um exemplo do envio de uma mensagem do tipo *String*.

Figura 19 – Tópicos do *ROS*Fonte: MATHWORKS ([MATHWORKS, 2020](#))

- Serviços⁴: semelhante aos Tópicos, mas os serviços garantem a comunicação síncrona, que é comumente requisitada em sistemas distribuídos, onde primeiramente é necessário realizar uma solicitação para, posteriormente, receber uma resposta.
- Nô mestre⁵: todos os outros nós (escravos) se identificam para o nô mestre, com isso o mestre possui as informações necessárias sobre o grafo computacional, para permitir a identificação e a comunicação. Seu modelo de funcionamento é comparável com um servidor de nomes (DNS).
- Mensagens⁶: é uma estrutura de dados por onde os nós enviam e recebem as informações. Pode ser constituído por variáveis de tipos primitivas, como por exemplo *integer*, *floating point*, *boolean*, *string* ou por estruturas aninhadas.

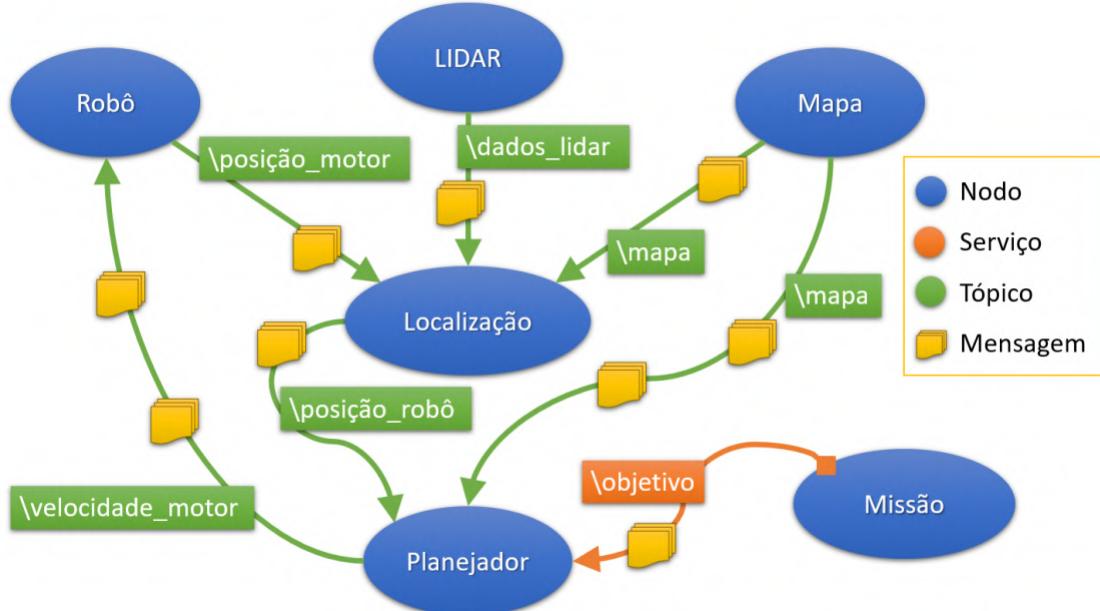
É visto na [Figura 20](#) uma representação do funcionamento do *ROS*. Nessa ilustração podemos notar a flexibilidade do sistema, permitindo que diversas comunicações sejam realizadas de maneira simultânea, um mesmo nô possui a capacidade de receber e enviar mais de uma mensagem por diferentes tópicos e serviços.

³ <http://wiki.ros.org/Topics>

⁴ <http://wiki.ros.org/Services>

⁵ <http://wiki.ros.org/Master>

⁶ <http://wiki.ros.org/Messages>

Figura 20 – Exemplo de didático de funcionamento do *ROS*Fonte: BUZZI ([BUZZI, 2019](#))

Nessa seção foram abordadas uma breve apresentação e descrição das principais funcionalidades utilizadas nesse trabalho, entretanto o *ROS* possui uma estrutura com muito mais recursos do que aqui foi apresentado. Esse *framework* além de ser utilizado em trabalhos acadêmicos, também oferece suporte a diversas aplicações comerciais, seria possível desenvolver um estudo apenas descrevendo suas funcionalidades e seu funcionamento. Assim sendo, caso o leitor se interesse por mais informações, é recomendado que consulte a wiki oficial da ferramenta em português⁷ ou em inglês⁸ por possuir alguns conteúdos extras.

⁷ http://wiki.ros.org/pt_BR⁸ <http://wiki.ros.org/>

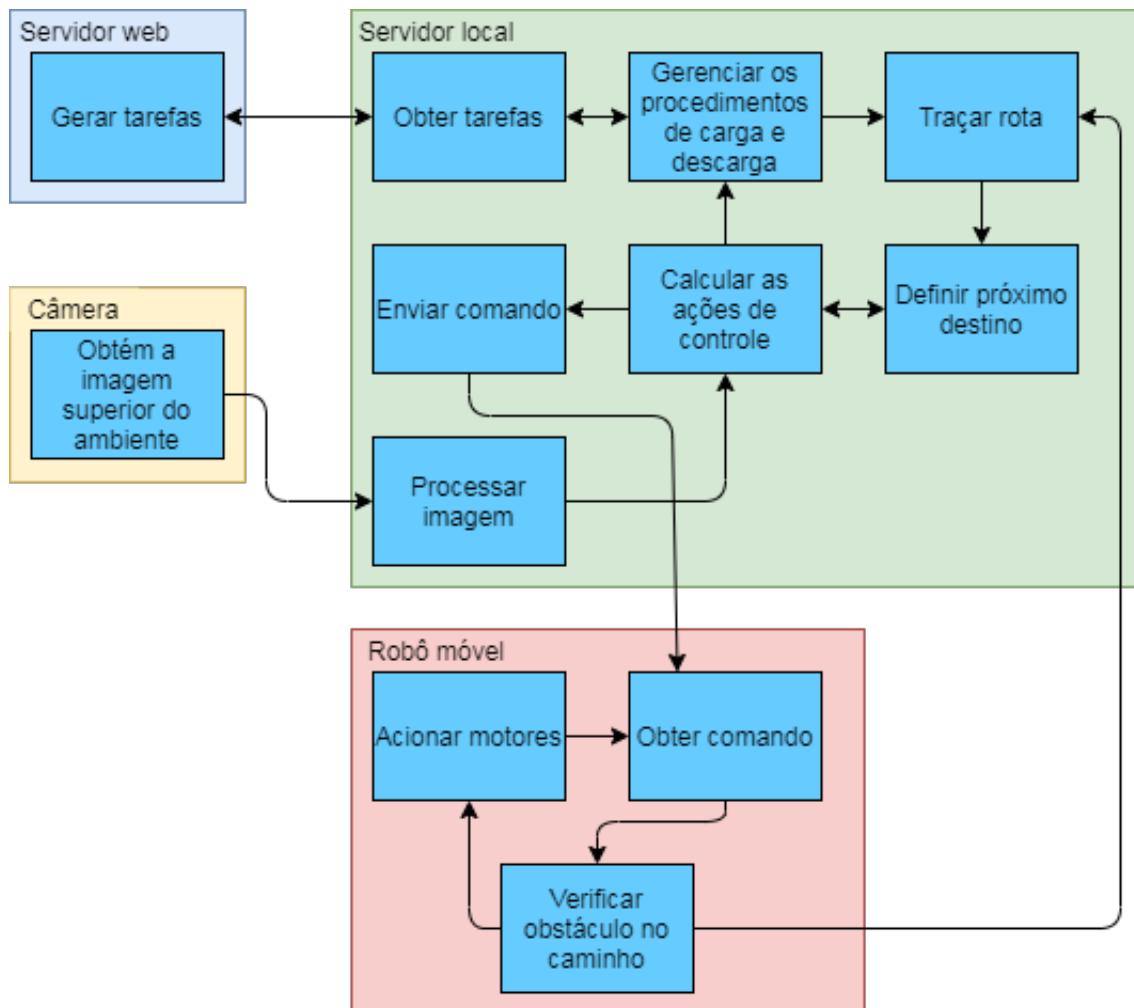
3 Desenvolvimento do Transportador Autônomo

Neste capítulo são apresentadas as ferramentas e métodos aplicados no desenvolvimento do projeto, sendo elas as características construtivas do robô, os algoritmos, técnicas utilizadas e também a interface *Web* implementada.

3.1 Arquitetura Geral do Sistema

A arquitetura geral do sistema é ilustrada na [Figura 21](#) por um diagrama de bloco que ilustra as conexões entre os subsistemas e também o que é executado em cada um deles.

Figura 21 – Diagrama de bloco geral do sistema



A interface com o usuário é feita por meio de uma página *Web* (hospedada no Servidor *Web*), nessa página o usuário cria as tarefas que o robô irá executar, cada tarefa é constituída por

um ponto de carga e outro de descarga, essas tarefas são lidas pelo Servidor Local, que deve processá-las e gerar os procedimentos de carga e descarga.

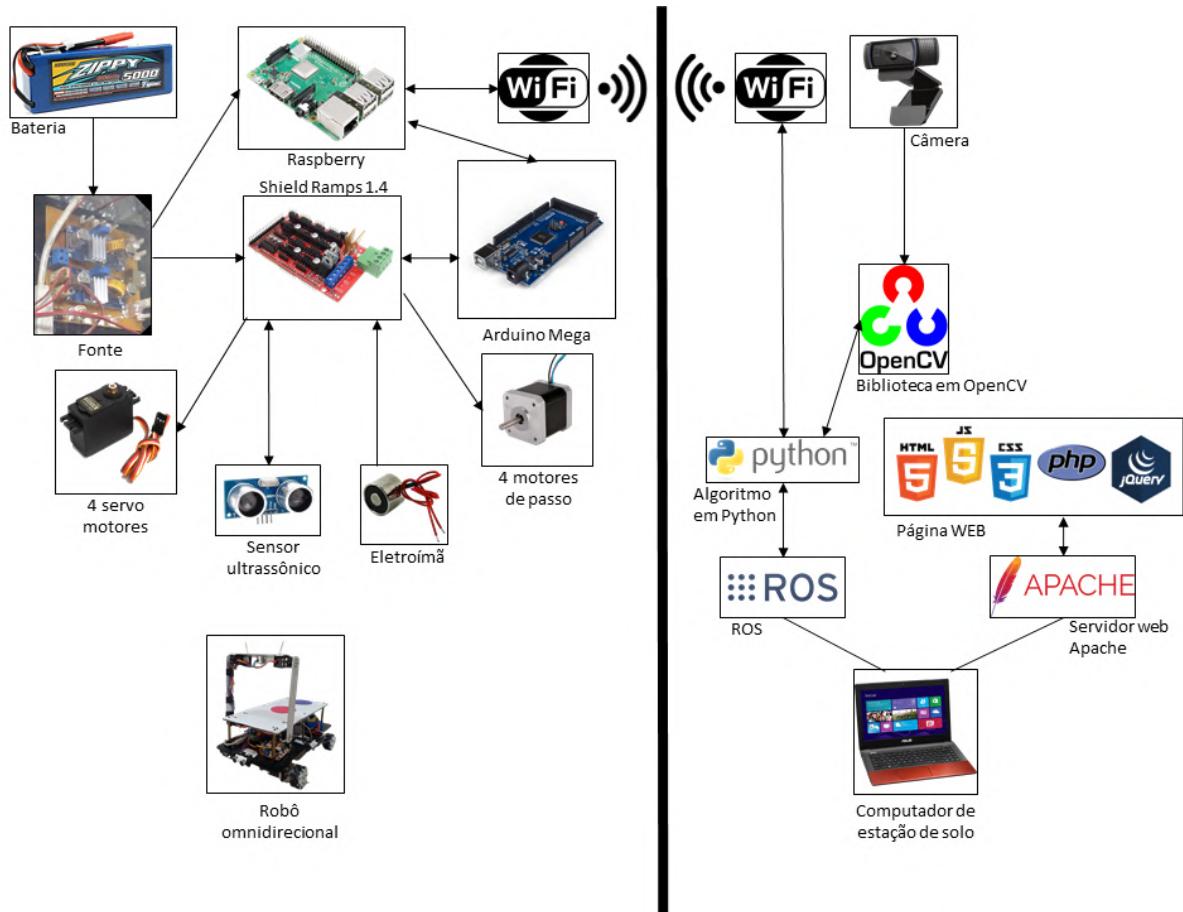
Após a leitura é feito o planejamento da rota para o primeiro objetivo, o trajeto da rota é composto por um ou mais nós do grafo, portanto para que o robô execute a trajetória entre os pontos de carga e descarga, ele deve passar por vários nós intermediários e, quando o robô atinge um nó intermediário, ele toma como destino o próximo nó do trajeto, até atingir o nó final e realizar a ação de carga ou descarga. Esse ciclo se repete até que todas as tarefas sejam executadas.

Na [Figura 21](#) no bloco denominado Calcular as ações de controle, é determinada qual ação será executada pelo robô, o cálculo dessas ações de controle são estimados pelo controlador P. Para realizar esse cálculo é preciso conhecer a localização atual do robô, para isso é feito o processamento de imagem, que possui como saída a posição x,y e orientação do theta (θ).

Após definir qual ação o robô deve executar, esse comando é enviado para a plataforma robótica móvel, que irá ler a mensagem, verificar se é possível executar essa ação (caso não exista um obstáculo no caminho) e então irá realizar o acionamento dos motores. Caso seja encontrado algum obstáculo, o robô deve aguardar 5 segundos, se esse obstáculo continuar obstruindo o caminho, ele será considerado um obstáculo fixo e uma mensagem é enviada para o Servidor Local calcular uma rota alternativa.

As ferramentas e dispositivos utilizados durante o funcionamento do sistema estão dispostos na [Figura 22](#). As setas indicam a conexão física entre os dispositivos e também apontam o sentido do fluxo de dados entre os subsistemas, as setas bidirecionais revelam um fluxo nos dois sentidos e as setas simples revelam um sentido único.

Figura 22 – Diagrama dos componentes do sistema



Na parte esquerda da divisão são representados todos os instrumentos que compõem fisicamente a plataforma robótica móvel, suas características são descritas a seguir:

- Bateria: armazena e disponibiliza a energia necessária para a operação de toda plataforma robótica móvel.
- Fonte: converte a tensão de entrada da bateria de 14,8V para tensões de saída em 12V (motores de passo) e 5V (servomotores e eletroímã).
- *Raspberry*: computador de baixo custo e compacto, responsável por realizar a interface de comunicação entre o *Arduino* e a rede *ROS*.
- *Arduino Mega*: constituído por uma placa de prototipagem ATmega 2560 para controle de todo os atuadores.
- *Shield Ramps 1.4*: nódulo eletrônico compatível com o *Arduino Mega* 2560 que facilita o controle de diversos atuadores.
- 4 motores de passo: realizam a movimentação da plataforma através de rodas fixadas nos eixos de cada um dos motores.

- Eletroímã: compõe o braço robótico, utilizado para agarrar e manipular as cargas por meio de superfícies metálica fixadas nos objetos.
- 4 sensores ultrassônicos: dispositivos utilizados para detectar obstáculos no caminho do robô, por se tratar de um robô omnidirecional, existem 4 sensores distribuídos em suas laterais.
- 4 servomotores: constituem o braço robótico, dois motores realizam o movimento do braços e outros executam a movimentação do efetuador (*end-effector*).
- Robô omnidirecional: plataforma robótica móvel que comporta todos os componentes citados.

Os dispositivos (câmera e *laptop*) e as ferramentas (*software*) apresentados do outro lado da divisória, compõem o Servidor Local e o Servidor *Web*. Como descrito a seguir:

- Câmera: realiza a captura de imagens que são utilizadas para estimar a localização do robô no ambiente.
- Biblioteca *OpenCV*: utilizada para realizar o processamento de imagem.
- Algoritmo em *Python*: processa todas as atividades do Servidor Local descrita na [Figura 21](#).
- *ROS*: permite a comunicação e a execução dos algoritmos de processamentos de imagem, de planejamento de trajetória e de controle do robô.
- Página *Web*: interface que permite ao usuário interagir com a plataforma.
- Servidor *Web Apache*: hospeda a página *Web* e a torna disponível para acesso.
- Computador do Servidor Local: computador onde a câmera está conectada, suporta todas as aplicações aqui citadas e realiza a comunicação local com a plataforma robótica.

Nas próximas seções serão apresentados maiores detalhes dos principais dispositivos citados nessa seção.

3.1.1 Concepção do Robô Transportador

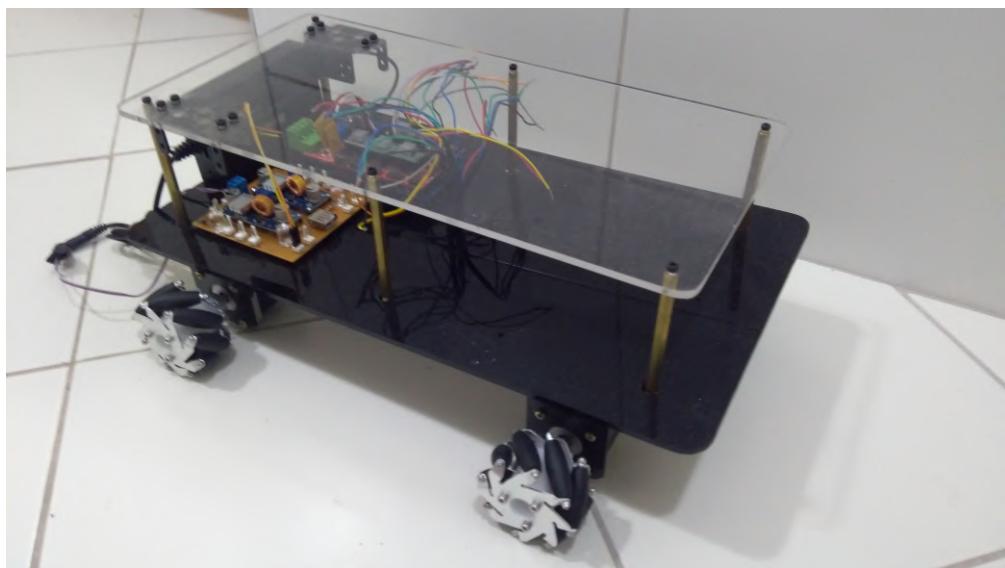
Com a necessidade de uma estrutura compatível com a aplicabilidade e objetivos do projeto, foi escolhido uma estrutura que atende as demandas de espaço e resistência para suportar todos os dispositivos já citados. Também foi concebido o braço robótico com 4 servomotores, como visto na [Figura 23](#). No efetuador do braço robótico foi feito um furo circular para acoplar o eletroímã.

Figura 23 – Componentes do braço robótico



Nos eixos dos 4 motores de passos, foram fixadas as rodas mecanum e então os motores foram fixados na estrutura. Com os motores já parafusados em suas posições, os fios foram organizados, de maneira que possibilite o acesso e a conexão com a *shield* que está conectada ao *Arduino*. A estrutura utilizada pode ser vista na [Figura 24](#).

Figura 24 – Estrutura principal



Antes de iniciar a montagem do braço robótico, vários testes de ajuste e calibração dos servomotores foram executados, para isso foram definidas 14 posições para o braço, os ângulos em graus executados pelo braço pode ser visto na [Tabela 1](#). O efetuador possui uma amplitude maior e foram definidas 19 posições, que são apresentadas na [Tabela 2](#). Para cada posição precisa ser determinado um valor em graus para ambos os motores, um em cada lado. Dessa forma o movimento dos motores opostos são síncronos e equivalentes.

Tabela 1 – Servomotores - Braço

Posição	Servo Direito	Servo Esquerdo
0	7°	151°
1	17°	141°
2	25°	134°
3	33°	126°
4	41°	118°
5	50°	110°
6	58°	102°
7	68°	94°
8	77°	86°
9	87°	77°
10	96°	69°
11	106°	60°
12	117°	51°
13	125°	43°

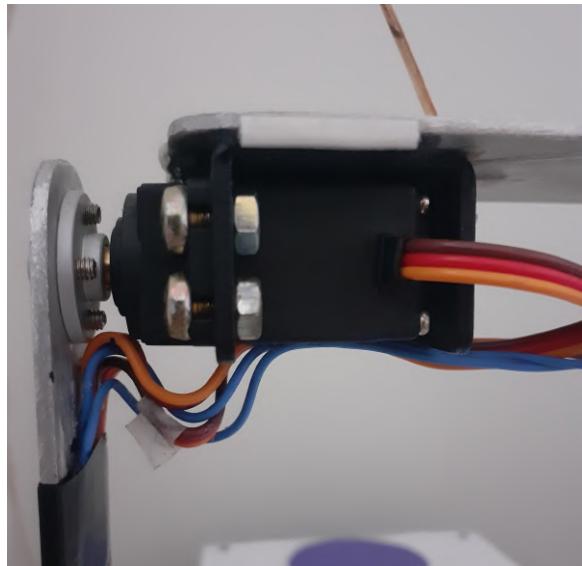
Tabela 2 – Servomotores - Efetuadores

Posição	Servo Direito	Servo Esquerdo
0	8°	167°
1	10°	164°
2	16°	156°
3	24°	147°
4	32°	139°
5	40°	130°
6	48°	121°
7	57°	112°
8	65°	104°
9	72°	93°
10	83°	84°
11	92°	76°
12	101°	67°
13	111°	58°
14	120°	49°
15	128°	41°
16	137°	33°
17	145°	26°
18	154°	16°

Posteriormente, todos os servomotores foram parafusados na estrutura principal e também no braço, já os motores do efetuador, além de serem parafusados no braço, foi preciso colar o suporte junto com a placa de metal com o eletroímã, que compõem a mão robótica. Dessa forma a superfície da mão continua plana, não atrapalhando a funcionalidade do eletroímã. Caso

fossem utilizados parafusos, seria preciso desbastar suas cabeças até que fiquem nivelados com a atual superfície. O servomotor fixado ao efetuador é visto na [Figura 25](#).

Figura 25 – Servomotor conectado no efetuador



Com a parte estrutural do robô finalizada, iniciou-se a fixação de todos os outros componentes. A fonte e o *Arduino* junto com a *shield* foram colocados em sua posição fixa, como visto na [Figura 26](#), também na parte interna do robô encontram-se o *Raspberry* e a bateria, os 4 sensores ultrassônicos estão fixados nas laterais, que são vistos na [Figura 27](#).

Figura 26 – Fonte e *Arduino* com a *shield*.



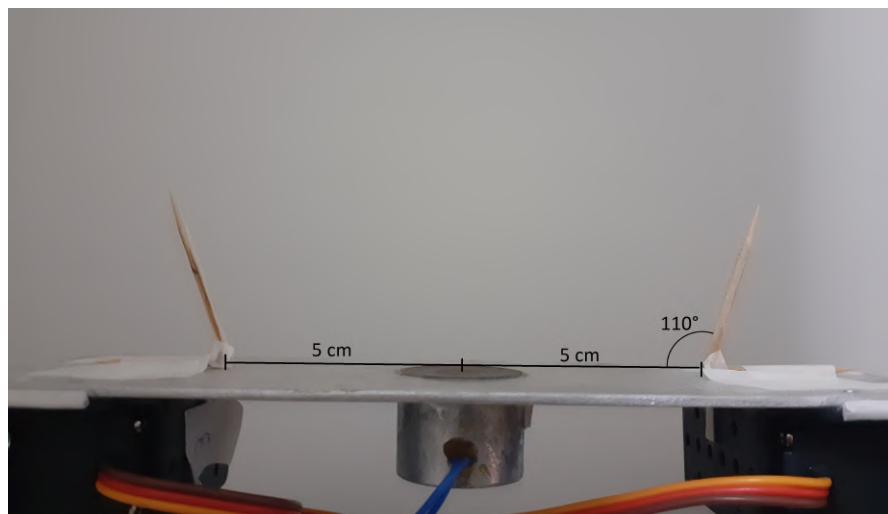
Figura 27 – Bateria, Raspberry e sensor ultrassônico



Na base superior foram fixados um círculo vermelho e outro azul, que são identificados pelo algoritmo de processamento de imagem para estimar a localização do robô, o círculo vermelho está posicionado no centro e o azul na traseira do robô. Os círculos estão sobre um fundo branco, o que gera um maior contraste e facilita a identificação desses marcadores.

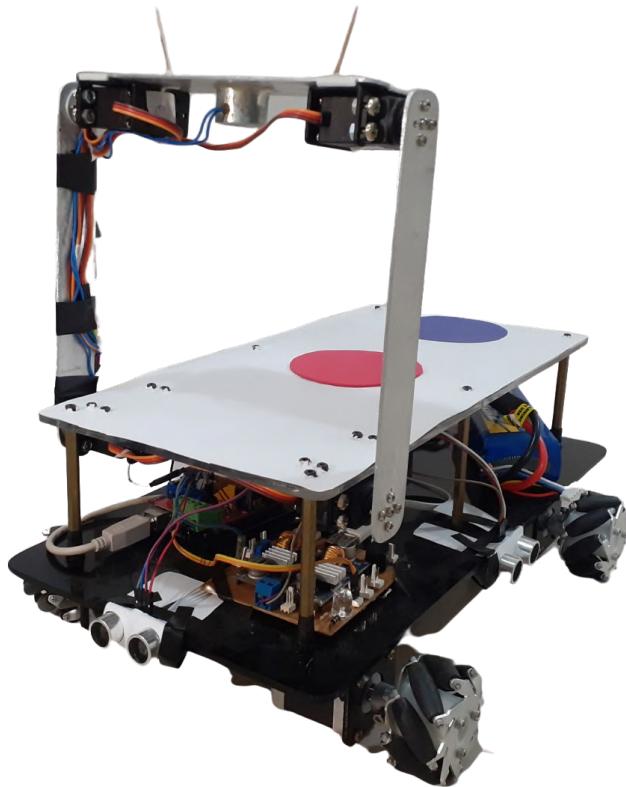
No efetuador foram fixados dois guias, que possuem o intuito de fazer o ajuste fino para posicionar a carga no centro do eletroímã, visto que várias cargas e descargas consecutivas poderiam gerar um acúmulo de erro, esse ajuste fino elimina essa possibilidade. Esses guias possuem um ângulo de 110° em relação à superfície e equidistam 5 centímetros do centro do eletroímã, como pode ser visto na [Figura 28](#).

Figura 28 – Guias do efetuador



Após todo o procedimento de montagem descrito, podemos observar o robô móvel desenvolvido na [Figura 22](#).

Figura 29 – Robô desenvolvido



3.1.2 Configuração *ROS*

A comunicação entre o Servidor Local e o sistema robótico móvel é feita pelo uso do *Wi-Fi* utilizando-se o *framework ROS*. A rede a ser configurada pelo *ROS* irá integrar três dispositivos, o Servidor Local se conecta ao *Raspberry* por *Wifi*, sendo que o *Raspberry* é conectado ao *Arduino* por cabo.

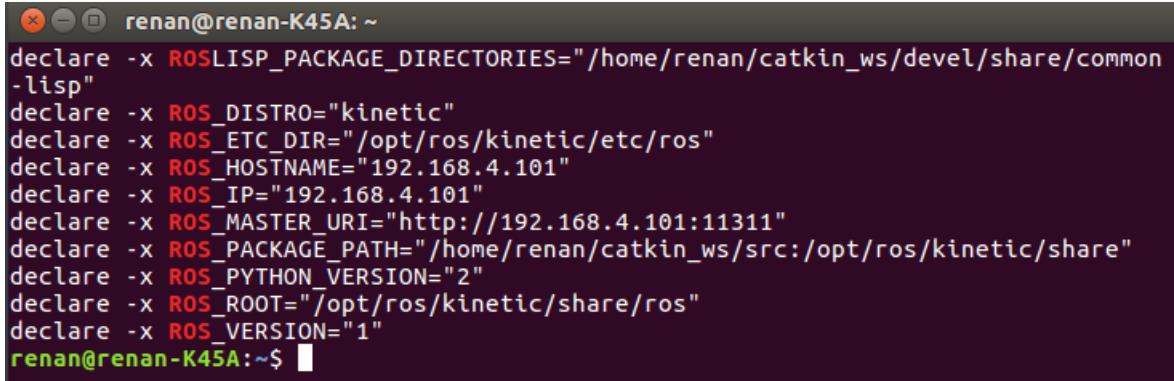
A comunicação com o *Arduino* é feita por comunicação serial *USART* assíncrona, diferentemente da comunicação entre o Servidor Local e o *Raspberry*, que é necessário configurá-los na rede *ROS* para que identifiquem o seu próprio IP e o IP do nó mestre.

A configuração dos dispositivos da rede *ROS* (Servidor Local e *Raspberry*) é salva em um arquivo oculto do sistema chamado *bashrc*. Os comandos desse arquivo são executados no momento em que um novo terminal é aberto. Para editar esse arquivo devemos executar o comando a seguir no terminal:

```
1 nano ~/.bashrc
```

Os comandos vistos na [Figura 30](#), referente ao Servidor Local (faz uso do sistema operacional Ubuntu 16.04.6 LTS), permitem que o nó se identifique na rede *ROS*. Para que isso ocorra de forma automática, esses comandos são acrescentados no arquivo `bashrc`, dessa forma ao abrir o terminal o nó já estará incluso na rede *ROS* e pronto para uso.

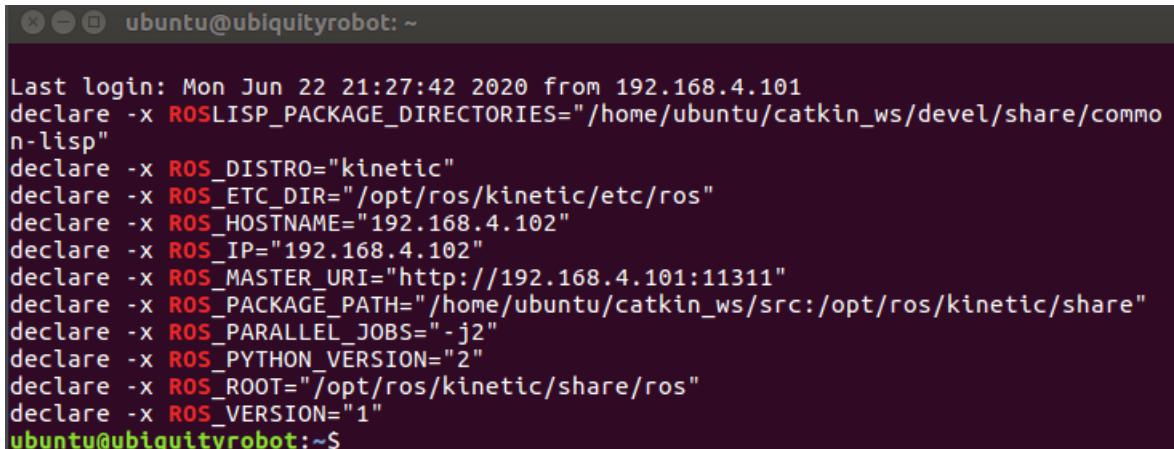
Figura 30 – Terminal do Servidor Local



```
renan@renan-K45A: ~
declare -x ROSLISP_PACKAGE_DIRECTORIES="/home/renan/catkin_ws/devel/share/common-lisp"
declare -x ROS_DISTRO="kinetic"
declare -x ROS_ETC_DIR="/opt/ros/kinetic/etc/ros"
declare -x ROS_HOSTNAME="192.168.4.101"
declare -x ROS_IP="192.168.4.101"
declare -x ROS_MASTER_URI="http://192.168.4.101:11311"
declare -x ROS_PACKAGE_PATH="/home/renan/catkin_ws/src:/opt/ros/kinetic/share"
declare -x ROS_PYTHON_VERSION="2"
declare -x ROS_ROOT="/opt/ros/kinetic/share/ros"
declare -x ROS_VERSION="1"
renan@renan-K45A:~$
```

O mesmo procedimento deve ser feito no *Raspberry* (utiliza o sistema operacional Ubuntu Mate 16.04¹), que deve ficar com a configuração apresentada na [Figura 31](#).

Figura 31 – Terminal do *Raspberry*



```
ubuntu@ubiquityrobot: ~
Last login: Mon Jun 22 21:27:42 2020 from 192.168.4.101
declare -x ROSLISP_PACKAGE_DIRECTORIES="/home/ubuntu/catkin_ws/devel/share/common-lisp"
declare -x ROS_DISTRO="kinetic"
declare -x ROS_ETC_DIR="/opt/ros/kinetic/etc/ros"
declare -x ROS_HOSTNAME="192.168.4.102"
declare -x ROS_IP="192.168.4.102"
declare -x ROS_MASTER_URI="http://192.168.4.101:11311"
declare -x ROS_PACKAGE_PATH="/home/ubuntu/catkin_ws/src:/opt/ros/kinetic/share"
declare -x ROS_PARALLEL_JOBS="-j2"
declare -x ROS_PYTHON_VERSION="2"
declare -x ROS_ROOT="/opt/ros/kinetic/share/ros"
declare -x ROS_VERSION="1"
ubuntu@ubiquityrobot:~$
```

Sempre que o *ROS* é executado, devemos verificar se os IPs dos dispositivos estão corretos, caso contrário é preciso atualizar os campos descritos a seguir.

- `ROS_HOSTNAME` é o nome que este dispositivo será visto pelos outros dispositivos da rede *ROS*.
- `ROS_IP` deve-se inserir o IP atual do dispositivo, permitindo que ele se identifique na rede.
- `ROS_MASTER_URI` representa o IP do dispositivo master, é preciso inserir o IP e a porta, que por padrão é 11311.

¹ <http://wiki.ros.org/ROSberryPi/Installing%20ROS%20Kinetic%20on%20the%20Raspberry%20Pi>

Após essas configurações, toda comunicação é feita dentro do *ROS* utilizando-se os nós e tópicos. No projeto existem 3 nós, sendo eles o Servidor Local, o *Raspberry* e o *Arduino*. Como o *Raspberry* se comunica com o *Arduino* pela porta serial, ele se torna uma interface de comunicação entre o *Arduino* e o Servidor Local.

Toda a comunicação é definida por dois tópicos, o principal é o *talker* que é utilizado para enviar os comandos de controle entre o Servidor Local e o robô, com uma frequência de 20 Hz, essa frequência foi definida de forma empírica, visto que frequências maiores não resultariam na melhoria do sistema, devido a limitação do tempo de resposta do hardware, e frequências menores podem resultar na diminuição da precisão. A comunicação é constituída por uma mensagem de texto do tipo *String*, seu formato de instrução é apresentado na [Tabela 3](#), onde o campo Comando representa a ação que o robô irá executar (transladar, rotacionar, mover braço), o campo Velocidade controla a intensidade da navegação, o campo Curva determina se o robô irá transladar e rotacionar ao mesmo tempo e o campo Intensidade determina a acentuação do movimento de rotação durante o movimento de translação.

Tabela 3 – Formato da mensagem *talker*

Tamanho (caractere)	2	2	2	2
Campo	Comando	Velocidade	Curva	Intensidade

O outro tópico desenvolvido é definido por "answer", que é utilizado no momento em que o robô identifica um obstáculo fixo e envia uma mensagem para informar o Servidor Local de que a trajetória atual está obstruída. Essa mensagem também é do tipo *String*, constituída por um único caractere que indica a detecção de um obstáculo.

3.2 Interface Web

A interface *Web* desenvolvida deve permitir ao usuário controlar o sistema, inserindo as tarefas que o robô deve executar e também deve promover o acompanhamento e monitoramento de todo o processo, de maneira remota. Com isso o sistema robótico pode ser controlado a partir de qualquer lugar que possua acesso a internet e a interface *Web*, independentemente de onde o operador está fisicamente alocado. Essa característica é bastante relacionada com a indústria 4.0.

A implementação dos elementos visuais da página foi feita com o uso de *HTML* e *CSS*, o *Javascript* também foi utilizado para a manipulação de alguns elementos visuais e processamento dos dados. Para realizar a comunicação com o algoritmo em *Python*, foi utilizado *PHP*, que permite a manipulação de arquivos no Servidor *Web*, onde o site está hospedado. O site foi hospedado em um Servidor *Web Apache*, permitindo que ele seja acessível na rede.

O controle do sistema robótico pelo usuário é feito pela criação de tarefas, nessas tarefas deve-se incluir o ponto onde deve-se manipular a carga e o ponto onde a mesma deve ser

descarregada, sendo possível criar uma lista de tarefas, que o robô irá executar de maneira sequencial. A interface para criar as tarefas é vista na [Figura 32](#).

Figura 32 – Criar uma tarefa

Criar tarefas

Selecione

3	▼
Ponto de Descarga	
Ponto de Descarga	▼
0	
5	
8	
Excluir tarefas	

Lista de tarefas

EXECUTAR

Após inserir na página *Web* o ponto de carga e o ponto de descarga, é preciso clicar em "Adicionar tarefa", dessa forma ela será incluída na Lista de tarefas, como visto na [Figura 33](#), esse processo deve ser repetido para inserir mais de uma tarefa (ou seja, as cargas a serem transportadas). Caso seja preciso excluir ou corrigir alguma tarefa, é possível clicar em "Excluir tarefas", para que a Lista de tarefas seja limpa, possibilitando que as tarefas sejam inseridas novamente.

Figura 33 – Tarefa adicionada à lista

Criar tarefas

Selecione

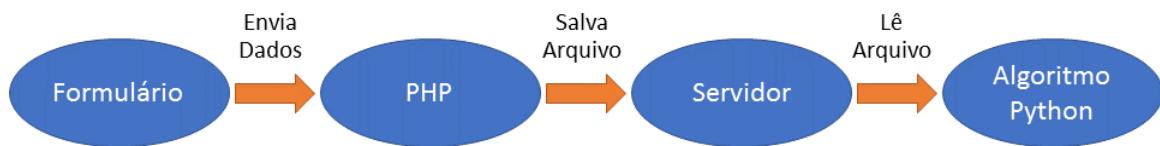
3	▼
5	▼
Adicionar tarefa	
Excluir tarefas	

Lista de tarefas

Carga:3;Descarga:5;
EXECUTAR

Após inserir todas as tarefas desejadas e clicar em "EXECUTAR", os dados são encapsulados e enviados para o *PHP*, que por sua vez irá salvar um arquivo no Servidor *Web*. O arquivo gerado será identificado e lido pelo algoritmo *Python* no Servidor Local, onde as tarefas serão lidas e executadas. O fluxo de dados é representado na [Figura 34](#).

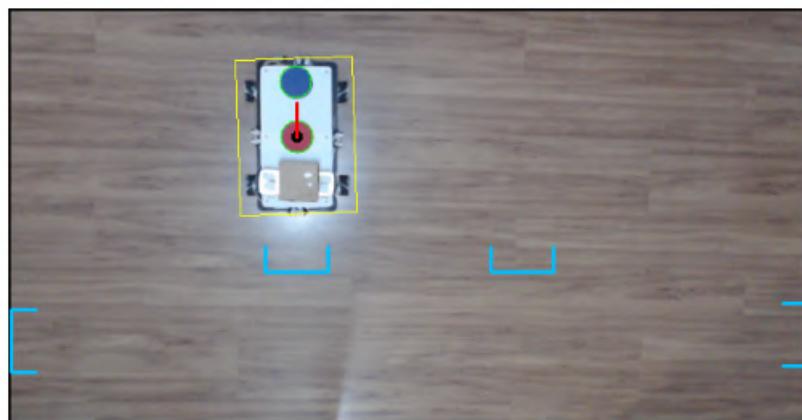
Figura 34 – Fluxo de dados ao criar uma tarefa



Os procedimentos de carga e descarga podem ser acompanhados pelo usuário através da página *Web* em tempo real, como é apresentado na [Figura 35](#). Na figura podemos visualizar a imagem em tempo real da vista superior do ambiente que a plataforma robótica opera, essa visão é a mesma utilizada pelo algoritmo de processamento de imagem e é possível acompanhar todas as etapas de transporte. Abaixo do vídeo, existe um histórico de todas as tarefas já executadas, as novas tarefas são incluídas em tempo real, por exemplo, quando o robô acaba de realizar a Carga no ponto 3, já é acrescentado no histórico "Carga:3", representando que essa etapa foi concluída.

Figura 35 – Visão superior do ambiente

Acompanhe em tempo real



Histórico de tarefas

- > Tarefa 3 - Carga:3
- > Tarefa 2 - Carga:3 - Descarga:0
- > Tarefa 1 - Carga:2 - Descarga:4 - Carga:2 - Descarga:4

As imagens de vídeo em tempo real são salvas no Servidor *Web* após o processamento de imagem no Servidor Local. Existe um temporizador que permite o sistema salvar 10 imagens por

segundo, essas imagens são salvas de maneira sequencial, ordenadas de 0 até 9, formando uma fila cílica. A página *Web* irá ler a última imagem da lista, ou seja, a imagem mais antiga. Esse processo é feito para evitar que o vídeo fique intermitente, um evento conhecido como *flickering*.

Para exibir as novas tarefas no histórico, em tempo real, ao final de cada tarefa o algoritmo em *Python* mantém um arquivo de texto atualizado, adicionando as novas tarefas concluídas. Por sua vez a página *Web* irá ler esse arquivo, possibilitando que as novas tarefas sejam exibidas para o usuário, o site possui uma taxa de atualização do histórico de 1 Hz.

Toda a interface e funcionalidades da página *Web* desenvolvida é apresentada na [Figura 36](#).

Figura 36 – Interface *Web* desenvolvida



3.3 Planejador de Trajetória em Mapas Topográficos

O planejamento da trajetória é necessário para realizar a navegação no ambiente (evitando obstáculos) de modo que os procedimentos de carga e descarga possam ser devidamente executados pelo robô. O mapa topográfico é representado por um grafo que representa o ambiente em que o sistema robótico está inserido, após isso é possível inserir o nó origem e o nó destino para obter a trajetória ótima, caso exista. O planejador também deve aceitar a exclusão e inclusão de nós e arestas durante sua execução. O algoritmo Dijkstra foi implementado para realizar o planejamento de trajetória e está sendo executado no Servidor Local.

3.3.1 Dijkstra

Para gerar o grafo (que representa o ambiente utilizado), é necessário adicionar todas as arestas unidirecionais do grafo, como é apresentado na [Figura 37](#), cada aresta deve conter o nó origem, nó destino, a distância entre os nós (representada pela função d_p , onde é calculada

a distância euclidiana que já foi apresentada na [Equação 2.5](#)) e também um valor que indica a direção permitida no trajeto, pois em alguns casos existem obstáculos que impedem o robô de se deslocar para os lados. Na [Tabela 4](#) podemos verificar o que esses valores representam.

Figura 37 – Inserindo um novo grafo no Dijkstra

```
Dijkstra.graph = Dijkstra.Graph([
    ("0", "1", d_p(targets[0], targets[1]),2),
    ("1", "2", d_p(targets[1], targets[2]),2),
    ("2", "3", d_p(targets[2], targets[3]),2),
    ("3", "4", d_p(targets[3], targets[4]),2),
    ("4", "5", d_p(targets[4], targets[5]),2),
    ("5", "6", d_p(targets[5], targets[6]),2),
    ("6", "7", d_p(targets[6], targets[7]),2),
    ("7", "8", d_p(targets[7], targets[8]),4),
    ("8", "9", d_p(targets[8], targets[9]),4),
    ("4", "9", d_p(targets[9], targets[0]),0),
    ("9", "8", d_p(targets[0], targets[1]),3),
    ("8", "7", d_p(targets[1], targets[2]),3),
    ("7", "6", d_p(targets[2], targets[3]),2),
    ("6", "5", d_p(targets[3], targets[4]),2),
    ("5", "4", d_p(targets[4], targets[5]),2),
    ("4", "3", d_p(targets[5], targets[6]),2),
    ("3", "2", d_p(targets[6], targets[7]),2),
    ("2", "1", d_p(targets[7], targets[8]),2),
    ("1", "0", d_p(targets[8], targets[9]),2),
    ("9", "4", d_p(targets[9], targets[0]),0),
])
])
```

Tabela 4 – Direções que podem ser realizadas pelo robô

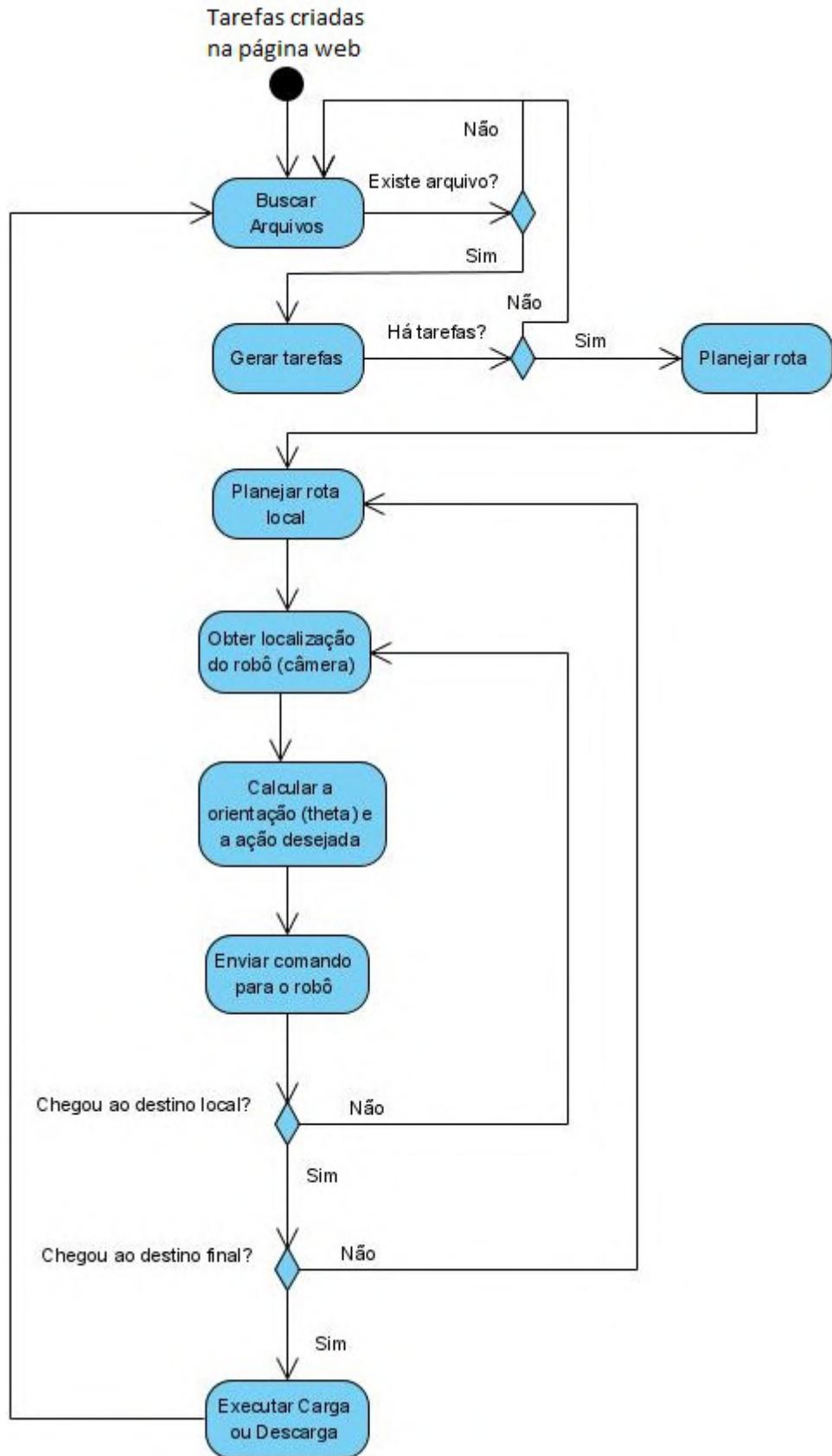
Valor	Direção
0	Frente e trás
1	Para os lados
2	Omnidirecional
3	Frente
4	Trás

A partir do mapa definido, o Servidor Local precisa receber um arquivo com tarefas para serem executadas; com as tarefas definidas é preciso separar cada tarefa em subtarefas, por exemplo "Carga:0;Descarga:3", que representam o ponto de carga e descarga, primeiramente o robô deve ir até o ponto 0, pegar a carga, e então ir até o ponto 3, para descarregar. Cada subtarefa irá gerar um destino para o robô, ao solicitar o destino 3 por exemplo, com o robô posicionado no ponto 0, o algoritmo de Dijkstra deve retornar uma rota que inicie em 0 e termine em 3, como por exemplo ["0", "1", "2", "3"].

Com a rota definida, é preciso iniciar o planejador local, o planejador local irá deslocar o robô entre dois nós conectados, por exemplo entre o ponto "0" e "1". O algoritmo deve identificar quando o robô chegou ao seu destino, nó "1", e então o planejador deve fornecer o próximo destino, nó "2". Esse procedimento é repetido até que se chegue no destino final, ponto "3". Quando o destino final é atingido é preciso executar a ação desejada de carga ou descarga.

Toda a lógica descrita é representada no diagrama disponível na [Figura 38](#).

Figura 38 – Diagrama planejamento de rota



3.3.2 Desvio dos obstáculos

No caso de existir algum obstáculo que impeça o robô de seguir a trajetória planejada, os sensores ultrassônicos são usados para identificá-los, apenas o sensor que está posicionado na mesma direção de movimentação do robô é ativo, por exemplo, caso o robô esteja se locomovendo para frente, o sensor da frente será constantemente monitorado para identificar o obstáculo e evitar colisões. Caso seja identificado algum obstáculo a menos de 10 cm, uma função embarcada no *hardware*, independente do Servidor Local, impede a locomoção.

3.3.2.1 Obstáculo móvel

Quando um obstáculo é identificado, o robô deve ficar imóvel, e um contador se inicia. Caso o obstáculo se desloque, como no caso de uma pessoa atravessando a pista, e o percurso volte a ficar livre em menos de 5 segundos, esse obstáculo é considerável móvel, portanto o robô irá continuar com sua trajetória normalmente.

3.3.2.2 Obstáculo fixo

Caso o obstáculo continue obstruindo a passagem por mais de 5 segundos, ele é considerado um obstáculo fixo. Com isso o robô irá enviar uma mensagem para o Servidor Local informando que um obstáculo fixo foi identificado, o servidor deve atualizar o mapa, removendo a aresta obstruída do grafo, e então deve-se planejar uma nova rota.

Se não existir uma nova rota, o mapa volta a sua configuração original e o robô irá ficar parado, aguardando que o caminho seja liberado. Entretanto, se houver uma ou mais rotas alternativas, o algoritmo de planejamento irá escolher a próxima rota mais curta. No caso de ser identificado algum obstáculo nessa nova rota, o processo se repete.

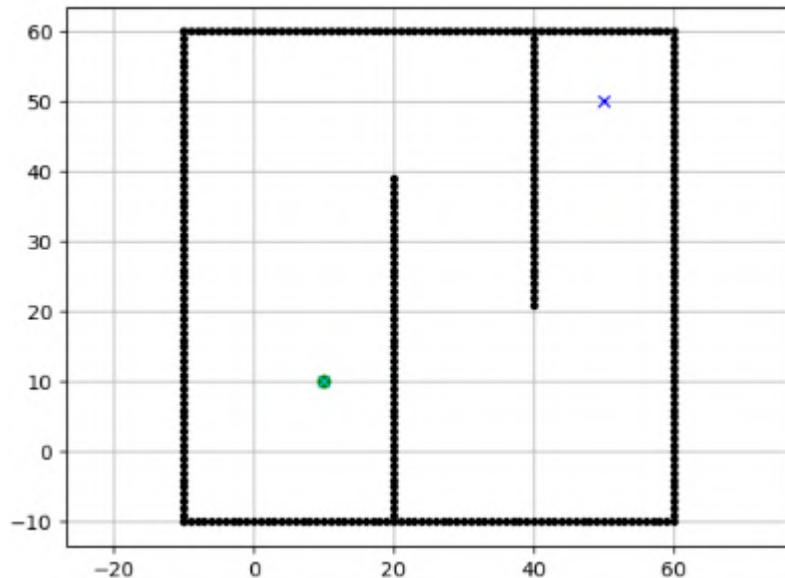
Caso existam mais de uma rota e todas estejam obstruídas, o robô irá tentar todas ciclicamente, na ordem da mais curta até a mais longa, até que consiga chegar em seu objetivo.

3.4 Planejador de Trajetória em Mapas Métricos

Com o intuito de estudar e compreender uma abordagem diferente ao mapa topográfico, foi desenvolvido em simulação o planejador de trajetória em mapas métricos. As simulações também foram feitas utilizando diferentes algoritmos, o Dijkstra e o A-estrela, onde é possível compreender as características, especificidades e diferenças entre os algoritmos.

Em ambos os algoritmos, primeiramente é preciso definir o mapa que será utilizado, especificando sua forma, tamanho e obstáculos. As áreas em preto representam obstáculos, portanto apenas é possível se locomover nas áreas em branco. Um exemplo do mapa descrito é apresentado na [Figura 39](#).

Figura 39 – Mapa métrico

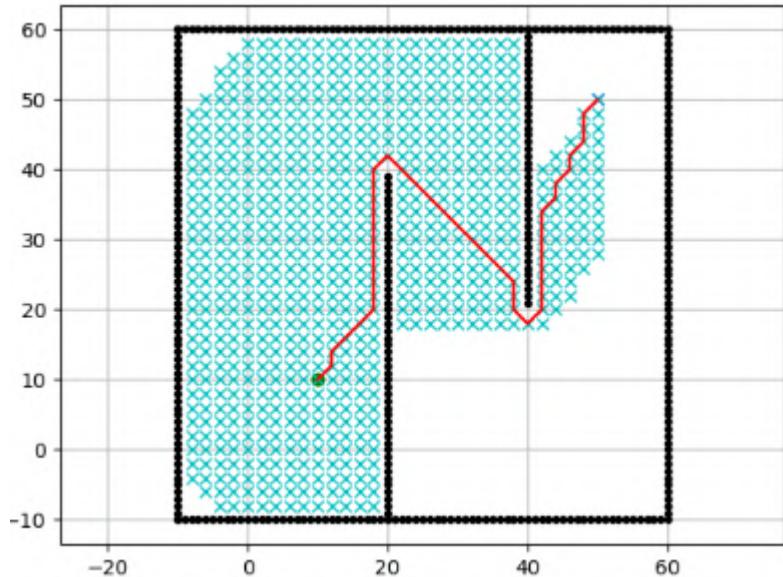


Além das características do mapa, também é preciso definir outras variáveis, que são apresentadas a seguir.

- Ponto inicial: é o ponto de partida do robô, o local de onde se inicia sua trajetória. No mapa apresentado podemos ver o robô em sua posição inicial, representado pelo ponto verde no canto inferior esquerdo;
- Objetivo final: é o ponto onde se deseja chegar, o objetivo final da trajetória, representado pelo "x" no canto superior direito do mapa;
- Raio do robô: o ponto representa o centro do robô, por isso devemos definir um tamanho para o robô, determinando o seu raio. Caso exista uma passagem menor do que o diâmetro considerado, a simulação não irá considerar como um caminho possível;
- Resolução do grid: toda representação de um mapa deve possuir uma resolução, devemos inserir uma resolução que consiga representar os detalhes do mapa, mas também que possibilite a execução em um tempo hábil, visto que mapas com muitos pontos podem necessitar de bastante processamento.

Durante sua execução podemos visualizar na animação quais são os grids que já foram testados, marcados pelo "x" em azul. Ao final, temos a trajetória em vermelho, definida e destacada no mapa, como vemos na [Figura 40](#).

Figura 40 – Ocupação do grid



3.5 Localização por Visão Externa

De maneira geral o sistema de localização é composto por um dispositivo de captura de imagem e o algoritmo de processamento de imagem. O *hardware* utilizado é uma câmera da Logitech modelo C920 HD Pro Webcam, esse modelo possui uma resolução máxima de 1080p (1920 x 1080) com uma taxa de atualização de 30 quadros por segundo com foco automático. A câmera utilizada é apresentada na [Figura 41](#). A câmera foi fixada a uma certa altura no ambiente para permitir a visualização superior da área que o robô irá se deslocar, essa altura deve ser ajustada de maneira que abranja toda a área de alcance do robô.

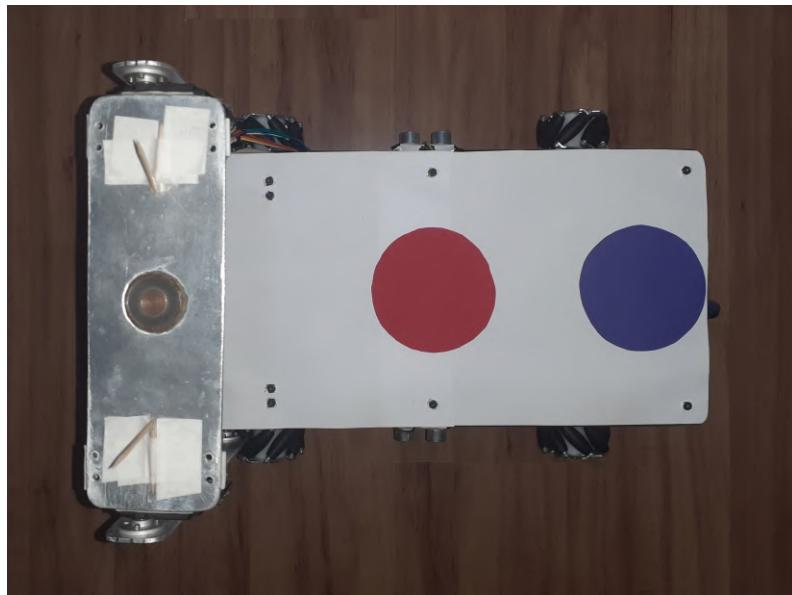
Figura 41 – Câmera C920



A câmera é conectada ao notebook por um cabo USB, entretanto, devido a sua posição fixa ao teto, foi preciso utilizar um cabo extensor USB. Também foi utilizado um Hub USB entre o cabo da câmera e o extensor, esse Hub possui uma fonte própria que, além de alimentar a câmera, também amplifica seu sinal, evitando perdas devido ao comprimento da extensão.

Com a câmera já se comunicando com o notebook, é preciso definir os marcadores que serão identificados pelo algoritmo de processamento de imagem. Neste projeto foram utilizados dois círculos, o círculo vermelho está no centro do robô, isso permite que a posição desse círculo defina a posição do robô. Também precisamos obter outra informação bastante importante, a sua orientação, para isso foi fixado um círculo azul na traseira do robô. Ambos os círculos possuem 8,5 cm de diâmetro, esse valor foi obtido empiricamente, pois o algoritmo identifica mais facilmente círculos maiores, portanto buscou-se otimizar o aproveitamento do espaço. Outro fator a ser considerado é o braço robótico, que pode obstruir a linha de visão entre a câmera e o círculo vermelho, sendo preciso verificar o ângulo de visão da câmera e limitar a posição máxima que o braço pode alcançar de modo a não prejudicar a detecção dos marcadores. A visão superior do sistema robótico é disponibilizada na [Figura 42](#).

Figura 42 – Visão superior do robô

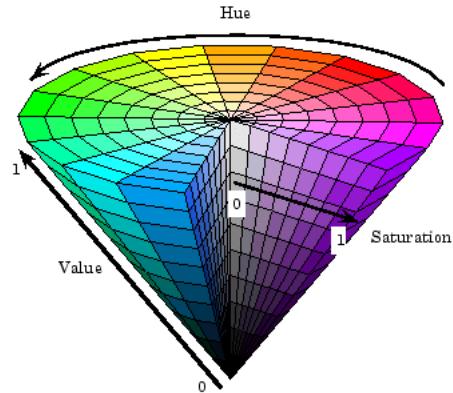


O processamento de imagem é executado em um algoritmo *Python*, integrado com o *ROS* usando a biblioteca *OpenCV*. O algoritmo utilizado utiliza o padrão de cores HSV e destaca os marcadores de interesse, é baseado no algoritmo apresentado pelos autores em ([RUZZON, 2019](#)).

Primeiramente é preciso identificar o código das cores dos marcadores. Uma característica importante é que iluminações diferentes afetam a identificação, por isso foram registradas imagens em diferentes iluminações com a câmera instalada em sua posição final, possibilitando a identificação de um espectro de valores aceitáveis. Esses valores são identificados utilizando o

parâmetro *HSV* (*hue*, *saturation*, *value*), sua representação é vista na Figura 43.

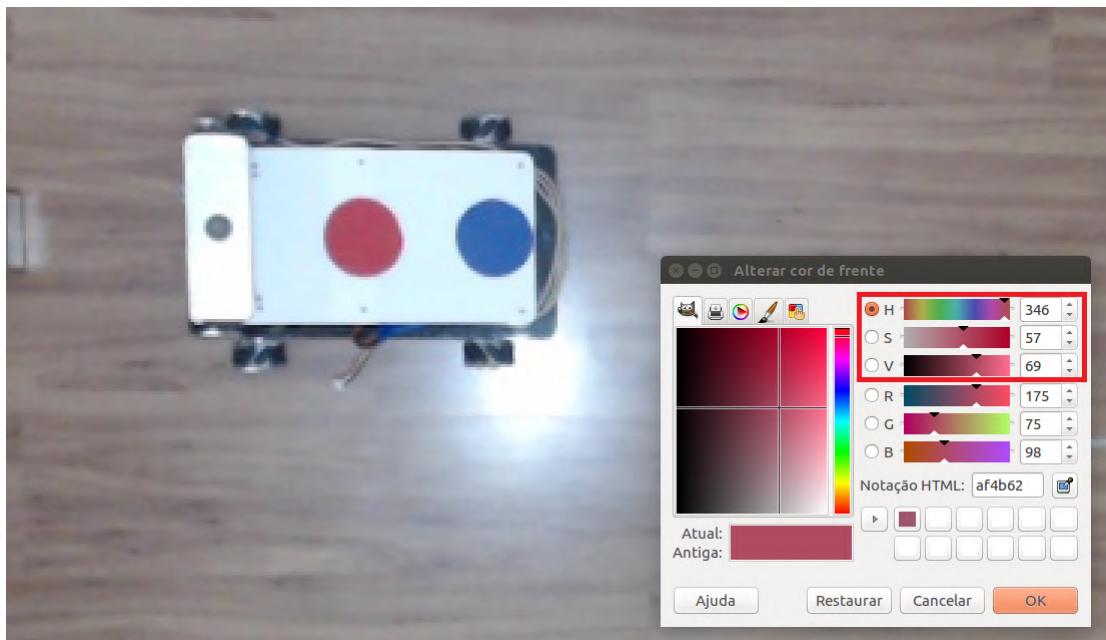
Figura 43 – Representação do parâmetro *HSV*



Fonte: Ravindran (G., 2017)

Para identificar os parâmetros da imagem obtida foi utilizada a ferramenta de seleção de cores do *software GIMP 2.8.22*, para isso é preciso clicar na área em que se deseja identificar a cor (ou seja, nos marcadores vermelho e azul), dessa forma o programa irá capturar a média dos pixels na área selecionada. Foi utilizado um raio de 5 e um exemplo de valores pode ser visto na Figura 43.

Figura 44 – Parâmetros *HSV* no *GIMP*



Contudo, os valores obtidos no *GIMP* não podem ser diretamente utilizados no *OpenCV*, visto que possuem um espectro de valores (escala) diferente, é preciso realizar a conversão de cada parâmetro para o padrão *HSV* do *OpenCV*. Essas conversões foram definidas de maneira

proporcional e são vistas na [Equação 3.1](#), onde H , S e V representam os valores que devem ser utilizados no algoritmo *OpenCV*.

$$\begin{cases} H = 0,5 * H_{gimp} \\ S = 2,55 * S_{gimp} \\ V = 2,55 * V_{gimp} \end{cases} \quad (3.1)$$

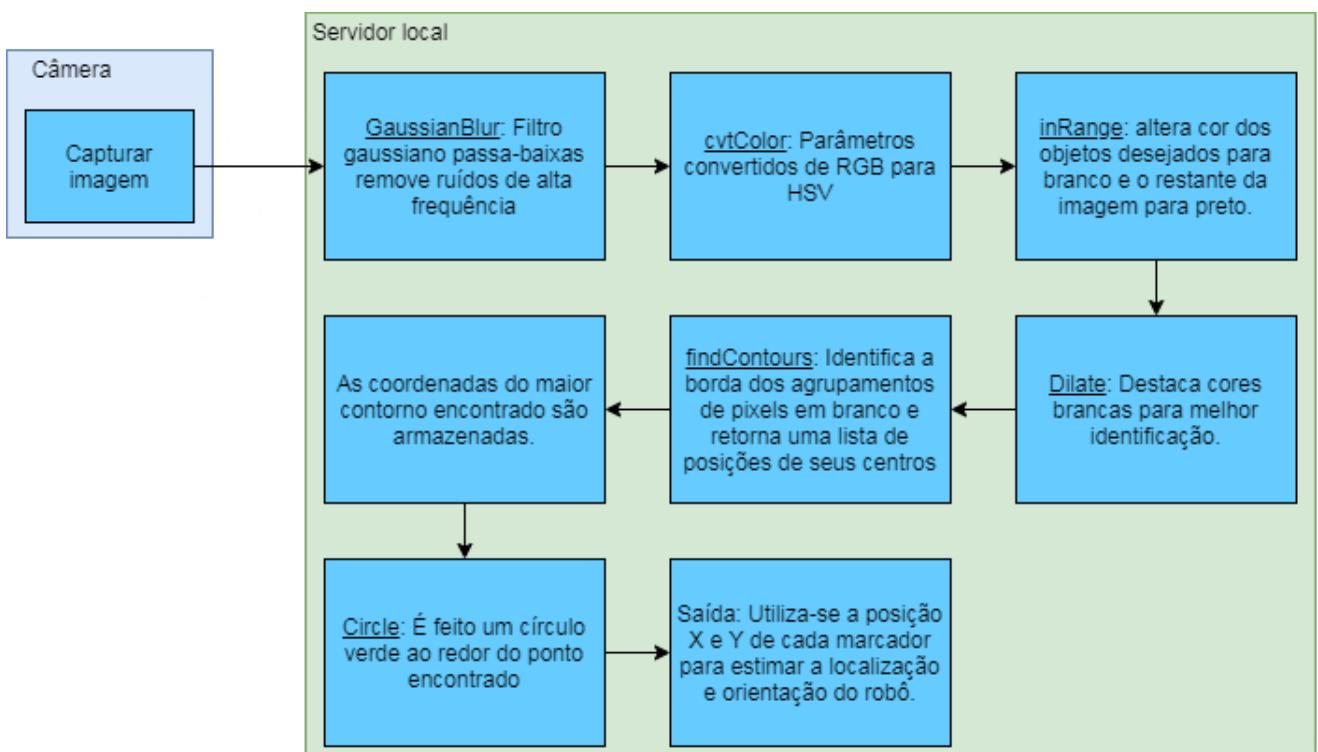
Após a conversão, os dados devem ser analisados, de forma que sejam abrangentes o suficiente para identificar os marcadores em diferentes condições de iluminação e posições, mas que também sejam restritos de modo que evite erros de identificação com outros objetos no ambiente, os valores utilizados são vistos na [Tabela 5](#).

Tabela 5 – Valores aceitáveis de cores no padrão HSV para identificação do robô

Cor	H (matriz)	S (saturação)	V (valor)
Azul	85 a 125	155 a 255	135 a 192
Vermelho	160 a 190	90 a 180	130 a 190

O processamento da imagem deve ser executado em seguida, o algoritmo implementado é representado na forma de um diagrama de bloco na [Figura 45](#), esse algoritmo é executado para ambos os marcadores e, possui como saída, suas respectivas localização X e Y, e orientação theta em relação ao centro do robô. No diagrama, estão em sublinhados os nomes dos métodos da biblioteca *OpenCV* utilizados no algoritmo.

Figura 45 – Diagrama de blocos - Processamento de imagem



A saída do algoritmo de processamento de imagem é usada para se estimar a localização do robô, utilizando a coordenada do marcador vermelho. Entretanto, para estimar a orientação, é preciso utilizar as coordenadas de ambos os marcadores, assim a orientação, representada pelo θ , é definida na [Equação 3.2](#), onde é utilizado o *arctan 2* para se obter o inverso da tangente nos quatro quadrantes do plano cartesiano.

$$\theta = \text{arctan} 2 \frac{y}{\sqrt{x^2 + y^2}} \quad (3.2)$$

3.6 Controlador de Seguimento de Trajetória

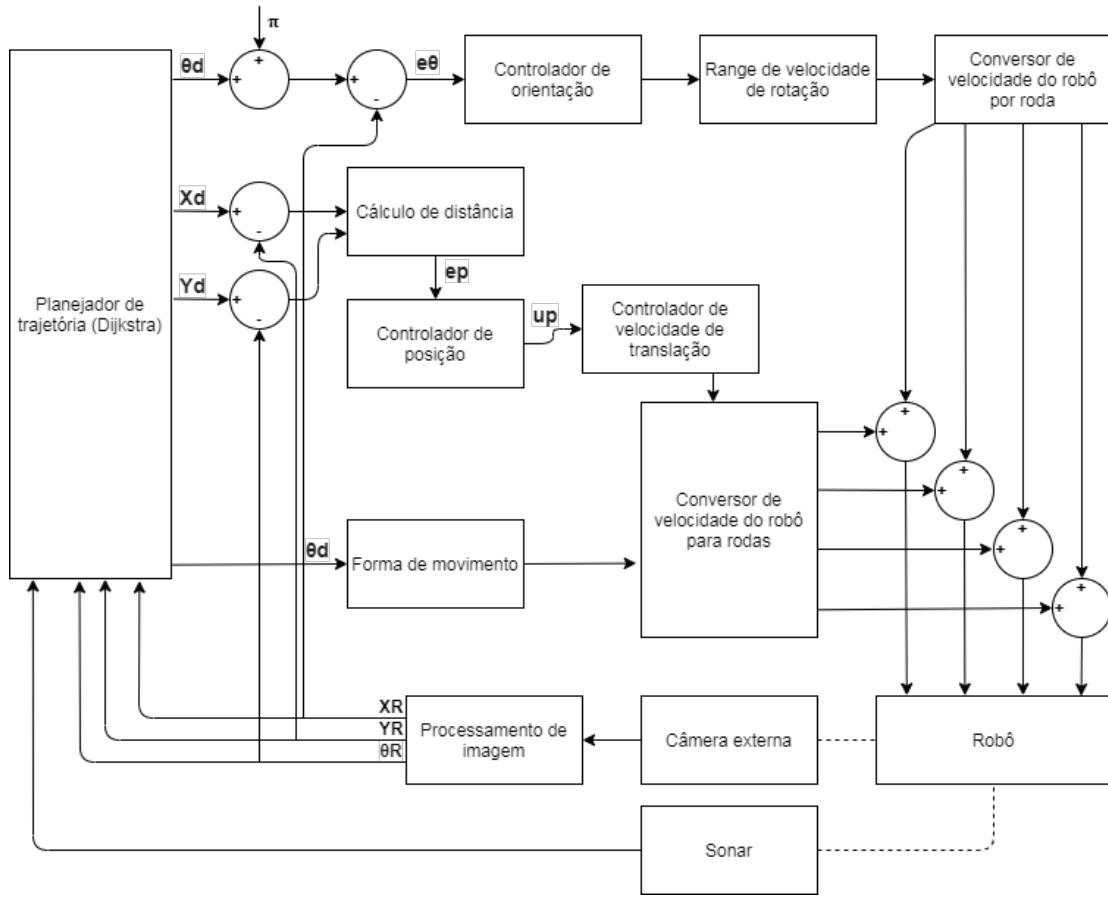
O controle do seguimento de trajetória do robô é realizado a partir de dois controladores lineares do tipo P, onde apenas o termo proporcional K_p do controlador PID é utilizado e essa constante possui valor 1. Os controladores PID são usados para controlar a distância entre a plataforma e a posição (nó) desejado no plano X e Y, e a orientação do robô em relação à posição alvo. Note que a estimativa de posição e orientação do robô é realizada em relação a um sistema de referência global estabelecido no ambiente.

Os controladores atuam em malha fechada, onde eles recebem como referência a posição X, Y e a orientação desejada que o robô deve alcançar e geram como saída os sinais de controle usados para acionar os motores.

Na [Figura 46](#) é apresentado o diagrama de bloco do sistema de controle, onde vemos os valores de referência X, Y e a orientação oriundas do planejador de trajetória. Faz-se a soma do valor da orientação (θ) com π , dessa forma os valores de θ variam entre 0 e 2π , ao invés de ser entre $-\pi$ e π , o que facilita a manipulação.

Os controladores representam o controle proporcional de ganho unitário e os saturadores são representados pelos limites de velocidade. A forma de movimento é referente a orientação do robô, como por exemplo, se está andando para frente ou para trás. O $e\theta$ é o erro de orientação e o up representa o erro de posição.

Figura 46 – Diagrama de blocos - controlador



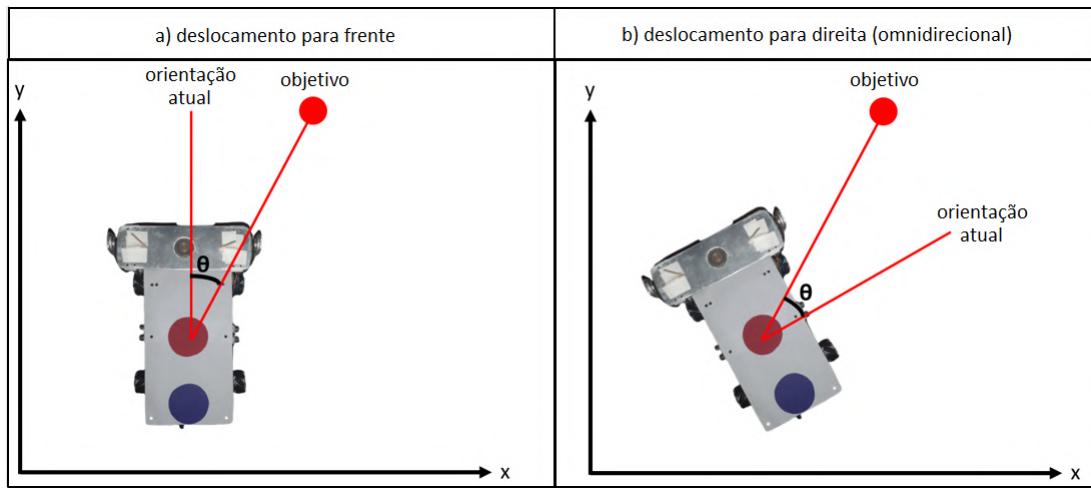
Para realizar a navegação no ambiente, podemos dividir o controle de trajetória, de maneira geral, em três etapas:

- O robô deve rotacionar sobre seu próprio eixo e se alinhar com a trajetória atual até que o theta seja menor do que o erro aceitável de 0,02 rad;
- Inicia-se o movimento de deslocamento linear em direção ao destino, caso o robô saia da rota desejada, é feita uma curva, com o intuito de corrigir a rota;
- Ao chegar próximo do destino, apenas rotações em seu próprio eixo são permitidas para correção de rota. Esse processo irá ocorrer até que se atinja a coordenada desejada com um erro menor do que 5,65 centímetros. Caso seja o local de manipulação, o erro deve ser menor do que 1,41 centímetros e deve-se ajustar a orientação do robô de acordo com a orientação da peça a ser manipulada, de forma que fique alinhado com a carga.

Na primeira etapa o robô pode se alinhar em quatro diferentes configurações, frente, lado direito, lado esquerdo ou para trás, o robô irá escolher a configuração mais próxima (que necessita de uma menor rotação para ser alcançada), a menos que exista alguma restrição no ambiente que limite a ir apenas para frente, por exemplo. Na [Figura 47](#) o *theta* representa a

diferença entre a orientação atual e a orientação desejada (objetivo), como visto no exemplo a) em que o robô irá se posicionar para andar para frente, e no caso b) o robô irá se deslocar de lado (movimento omnidirecional).

Figura 47 – Direção de deslocamento



O algoritmo de controle de trajetória é detalhado no diagrama apresentado na [Figura 48](#), onde o "Step" determina a execução de cada uma das três etapas e pode assumir valor 0, 1 ou 2.

A partir do valor estimado do erro de trajetória e distância do objetivo, os controladores irão aplicar nos motores de passo diferentes valores de velocidade de rotação, velocidade linear ou de curva. Em complemento ao diagrama, são apresentadas as propriedades na [Tabela 6](#), essas propriedades apresentam a lógica existente nos controladores.

Figura 48 – Fluxograma de execução da movimentação do robô

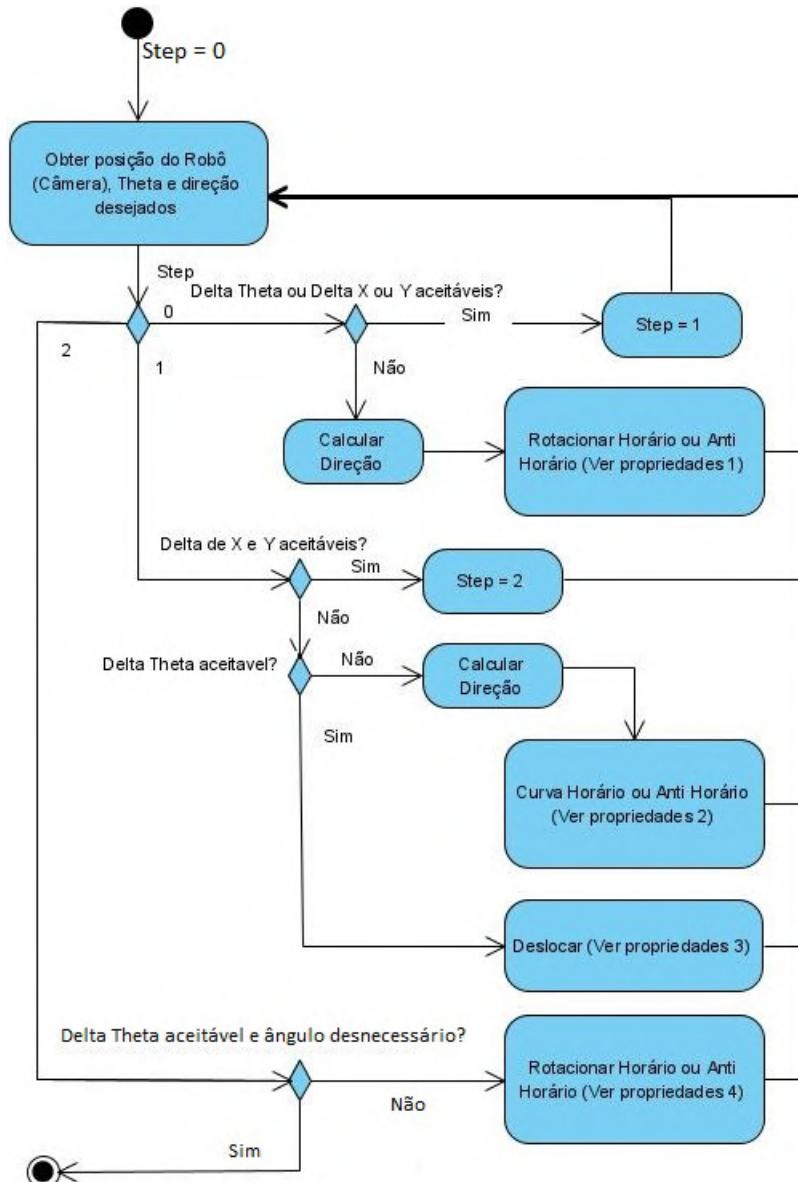


Tabela 6 – Condições de controle do robô

Propriedades 1	Propriedades 2	
Se DeltaTheta >0,4 rad Velocidade = 84 Senão Se DeltaTheta >0,15 rad Velocidade = 65 Senão Velocidade = 55	Se Distancia>25 cm Velocidade = 84 Senão Se Distância >15 cm Velocidade = 75 Senão Velocidade = 65	
Propriedades 3	Propriedades 4	
Se Distancia >25 cm Velocidade = 84 Senão Se Distância >25 cm Velocidade = 65 Senão Velocidade = 55	Se DeltaTheta >0,2 rad Velocidade = 84 Senão Velocidade = 65	

Onde:

- DeltaTheta representa a diferença, em radianos, entre a orientação atual do robô e a orientação desejada.
- Velocidade é o valor enviado ao *Arduino* proporcional a uma determinada velocidade de rotação das rodas.
- ValorCurva é um valor que determina a intensidade da curva a ser realizada.

O valor da velocidade definido no algoritmo pode assumir valores entre 0 e 99 e, após ser enviado ao *Arduino*, esse valor é manipulado para se obter o *delay* que define a velocidade do motor. Como o motor de passo necessita de pulsos para rotacionar, esse pulso é definido a partir de um pequeno *delay* que deve gerar uma onda quadrada, o *delay* é definido a partir das igualdades definidas na [Equação 3.3](#).

$$\begin{aligned} Velocidade &= 100 - \text{Velocidade} \\ \text{delayPasso} &= \text{Velocidade} * \text{Velocidade} \end{aligned} \tag{3.3}$$

Essa equação permite que, ao aumentar o valor da Velocidade no algoritmo, o delayPasso diminui, fazendo com que o robô ande mais rápido. Além disso, quando elevamos a Velocidade ao quadrado, é possível obter uma faixa de valores do delayPasso entre 1 e 10000, já que se trata de uma equação exponencial, conseguimos a partir de números de controle entre 0 e 99 obter um intervalo de valores finais bem maior.

Um método semelhante é utilizado para realizar curvas, visto que a variação de velocidade entre os pares de rodas paralelos faz o robô rotacionar em torno de seu eixo, o ValorCurva tem a

função de aumentar o valor do *delayPasso* nos pares de rodas na direção desejada. Vejamos um exemplo a seguir.

Caso o robô possua um deslocamento para frente, mas, nota-se que uma correção para a direita é necessária, o *delayPasso* das duas rodas da direita irá ser multiplicado pelo *ValorCurva*/100, como visto na [Equação 3.4](#). Portanto, como visto em Propriedades 2 especificado na [Tabela 6](#), o *delayPasso* pode ter um aumento de 60% ou 30%, dependendo da correção de rota exigida.

$$\text{delayPasso} = \text{delayPasso} * (\text{ValorCurva}/100) \quad (3.4)$$

3.7 Controlador do braço robótico

O controlador do braço robótico permite a manipulação de cargas, dessa forma, após o robô chegar em seu destino, o braço robótico deve atuar de forma a realizar a carga ou a descarga, conforme aplicável. O controlador utilizado é composto por diversos comandos sequenciais que devem posicionar o efetuador no local determinado e também controlar o eletroímã no momento correto.

A lógica utilizada no controlador é apresentada no pseudocódigo a seguir:

```

1 INICIO
2   LEIA comando
3   LEIA posicao
4   LEIA localizacao_destino
5   LEIA localizacao_carga
6   ComandarEfetuador("Tras")
7   ComandarBraco("Baixar")
8   ENQUANTO posicao != localizacao_carga FACA
9     LEIA posicao
10    LocomoverRobo("Frente")
11  FIM_ENQUANTO
12  SE comando == "Carga" ENTAO
13    ComandarEletroima("Ativar")
14  SENAO
15    ComandarEletroima("Desativar")
16  ENQUANTO posicao != localizacao_destino FACA
17    LEIA posicao
18    LocomoverRobo("Tras")
19  FIM_ENQUANTO
20  ComandarBraco("Levantar")
21  ComandarEfetuador("Frente")
22 FIM

```

4 Resultados

Nesse capítulo iremos realizar a verificação em simulação e validação experimental dos algoritmos desenvolvidos para executar a manipulação e o transporte de cargas usando a plataforma robótica real. A partir dessa plataforma diversos testes serão realizados para demonstrar todas as funcionalidades apresentadas no capítulo anterior, incluindo a integração entre os subsistemas. Em seguida, os resultados obtidos serão analisados para diferentes estudos de casos.

Na [Tabela 7](#) são apresentadas as características dos ambientes utilizados (ambiente simulado e ambiente físico), o grid só é aplicável aos ambientes métricos, portanto não é aplicável ao ambiente físico, que é topológico.

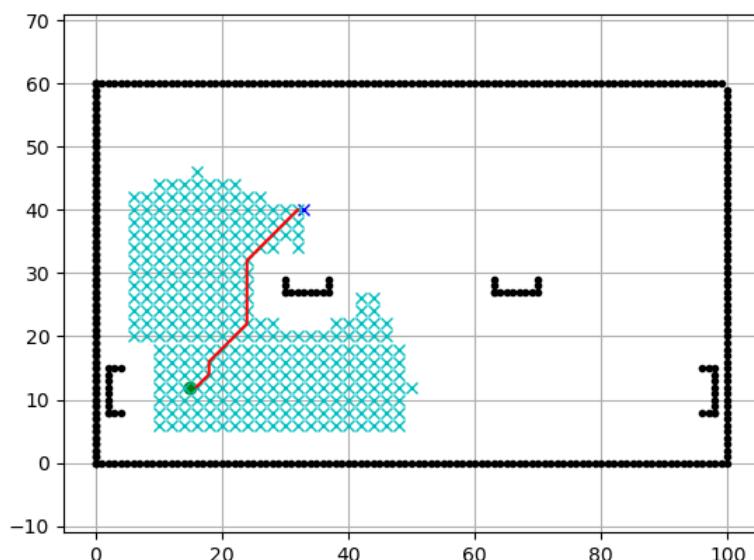
Tabela 7 – Características dos ambientes utilizados

	Ambiente Simulado	Ambiente Físico
Comprimento	100 cm	220 cm
Largura	60 cm	110 cm
Grid	2 cm x 2 cm	N/A

4.1 Avaliação em Simulação

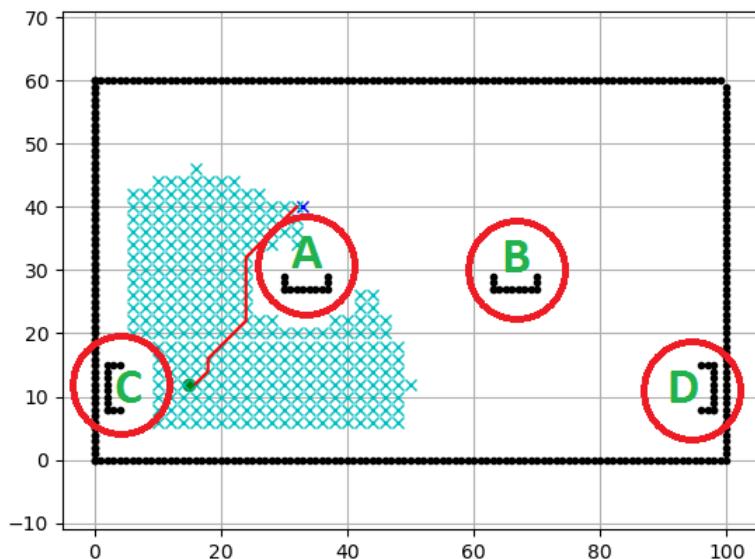
Todos os testes em simulação foram feitos em um mesmo ambiente, representado por um mapa métrico, similar ao ambiente físico (a proporção entre a largura e o comprimento é similar), apesar de não ser exatamente igual, o mapa pode ser visto na [Figura 49](#).

Figura 49 – Ambiente representando por um mapa métrico (grid)



Na [Figura 50](#) vemos a manipulação de uma carga do ponto A para o ponto C, para manipular várias cargas, a simulação deve ser executada diversas vezes consecutivas, por exemplo primeiro o robô vai do ponto B ao ponto A para buscar uma carga e depois deve ir do ponto A para o ponto C para deixar a carga em seu destino, nesse exemplo serão feitas duas simulações, uma entre o ponto B e o ponto A e outra entre o ponto A e o ponto C (essa última é vista no exemplo).

Figura 50 – Simulação - Pontos de carga e descarga



Foram definidas quatro categorias de testes simulados para comprovar e analisar o funcionamento dos algoritmos que iremos ver a seguir. A primeira categoria foi com apenas uma carga, onde o robô em simulação deve se locomover de um ponto ao outro. Foram registrados o tempo total de execução do algoritmo e também a distância total percorrida pelo robô em simulação. Esse teste foi repetido 5 vezes, dessa forma foram obtidos 5 resultados para o tempo total de execução e a distância percorrida. Outros testes simulados foram realizados para 4, 7 e 10 cargas, cada um repetido 5 vezes. A repetição de 5 vezes em cada categoria nos permite observar a precisão e a variação do algoritmo.

Na [Tabela 8](#) são apresentadas as tarefas executadas em cada categoria, as letras A, B, C e D referem-se aos locais de carga e descarga, onde "Carga" se refere ao vértice no qual a carga se encontra atualmente e "Desc" se refere ao vértice onde a carga deve ser entregue.

Tabela 8 – Categorias de testes em simulação - parte 1

Categoria	Carga 1	Desc 1	Carga 2	Desc 2	Carga 3	Desc 3	Carga 4	Desc 4
1 Carga	D	A						
4 Cargas	D	A	C	B	D	B	C	A
7 Cargas	D	A	C	B	D	B	C	A
10 Cargas	D	A	C	B	D	B	C	A

Tabela 8: Categorias de testes em simulação - parte 2

Categoria	Carga 5	Desc 5	Carga 6	Desc 6	Carga 7	Desc 7	Carga 8	Desc 8
1 Carga								
4 Cargas								
7 Cargas	D	A	C	B	D	B		
10 Cargas	D	A	C	B	D	B	D	A

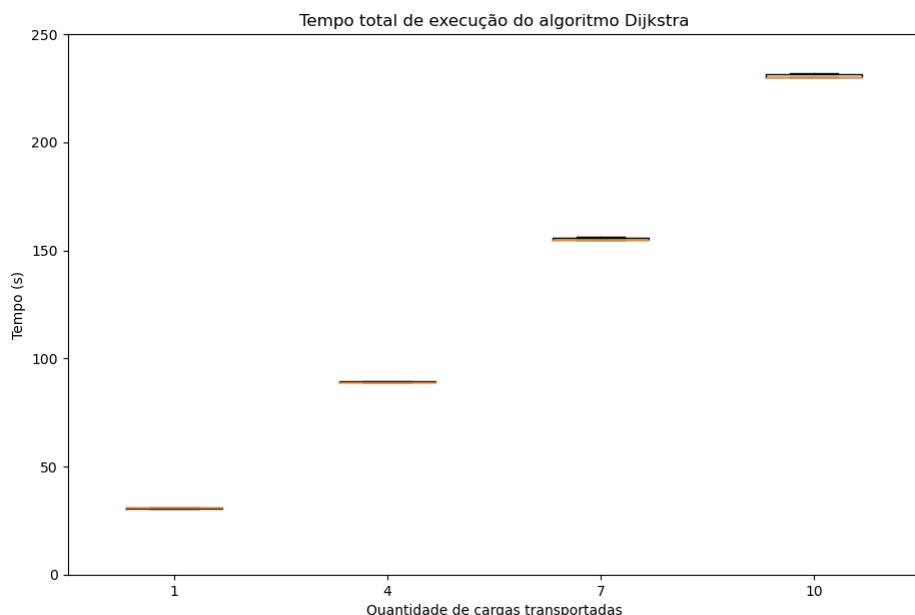
Tabela 8: Categorias de testes em simulação - parte 3

Categoria	Carga 9	Desc 9	Carga 10	Desc 10
1 Carga				
4 Cargas				
7 Cargas				
10 Cargas	C	B	D	B

4.1.1 Dijkstra

O primeiro resultado a ser apresentado é o tempo total que o algoritmo demorou para calcular a rota desejada. A [Figura 51](#) mostra a representação em boxplot dos valores do tempo total de execução do algoritmo utilizado. Como podemos notar o algoritmo não possui muita variação dentro de uma mesma categoria entretanto, quando aumentamos o número de cargas, é visto um aumento aproximadamente linear do tempo para calcular as rotas desejadas.

Figura 51 – Simulação Dijkstra - Tempo total

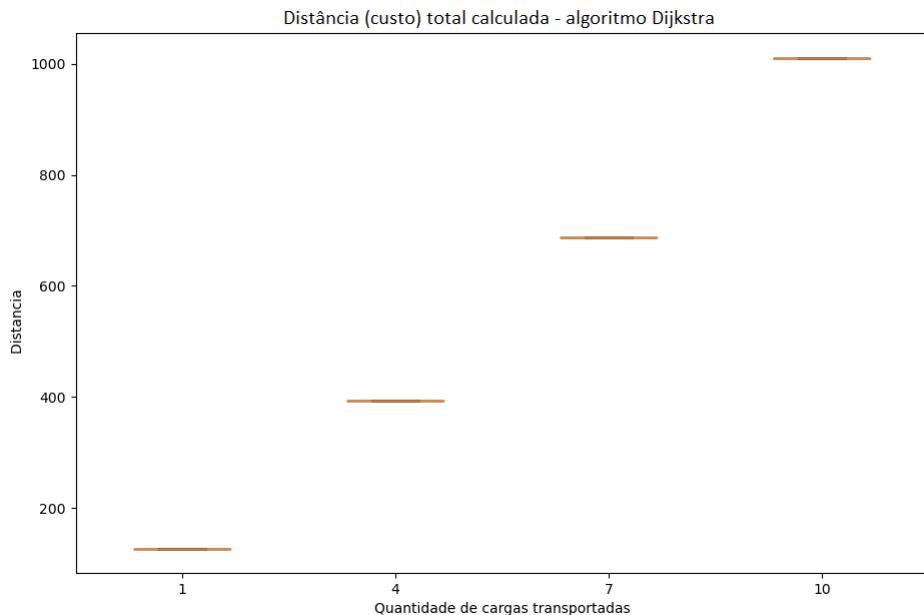


A [Figura 52](#) mostra os boxplot referente a distância total do caminho ótimo obtido. No caso de mais de uma carga, é feita a soma de todos os trajetos. Como podemos observar o

algoritmo se mantém constante em uma mesma categoria, o que mostra que ele sempre obtém a mesma distância ótima entre os mesmos dois pontos.

Entre diferentes categorias vemos um aumento linear da distância percorrida, conforme o esperado, visto que foi preciso um maior deslocamento para entregar mais cargas.

Figura 52 – Simulação Dijkstra - Distância ideal estimada pelo algoritmo Dijkstra

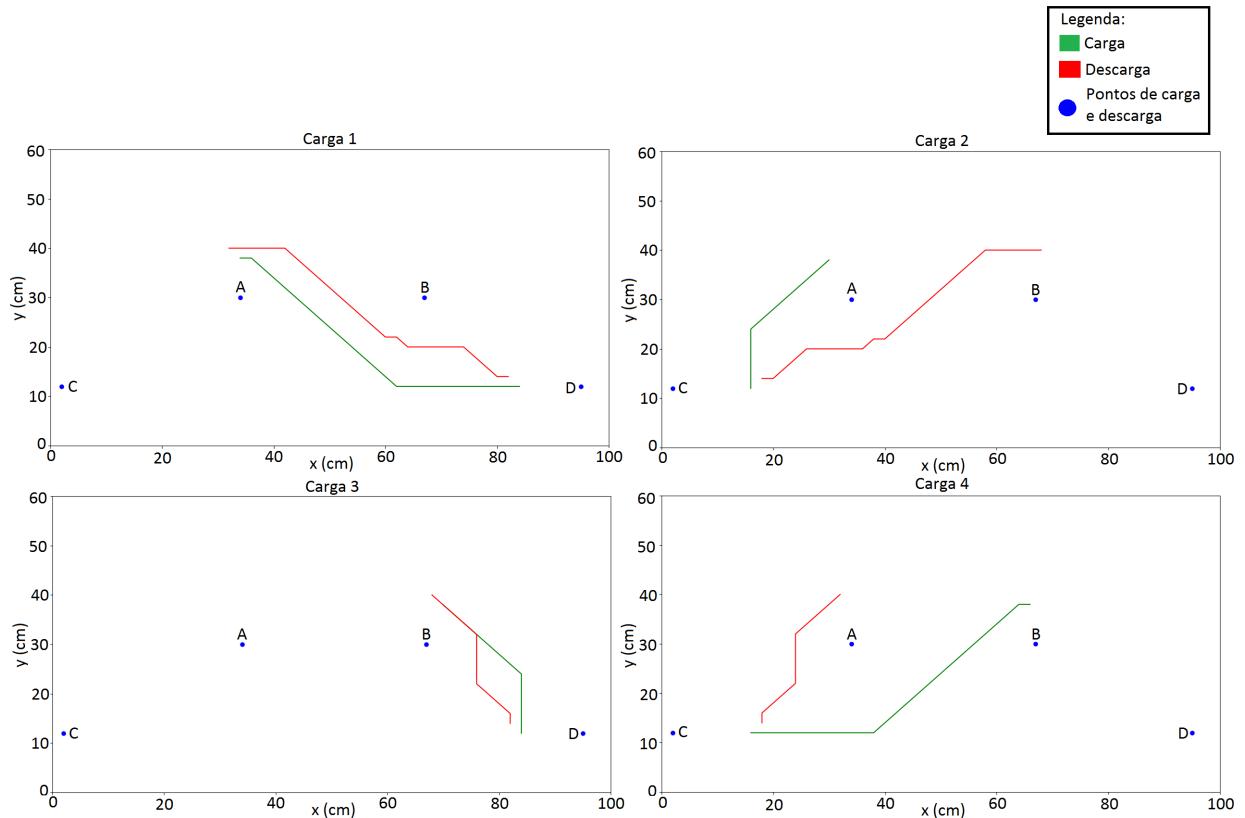


No próximo teste foram transportadas 4 cargas, conforme descrito abaixo, onde podemos visualizar as rotas obtidas na [Figura 53](#). O trajeto se inicia no ponto A, e então o robô deve executar as seguintes ações sequencialmente:

- Carga 1 - Buscar no ponto D, entregar no ponto A;
- Carga 2 - Buscar no ponto C, entregar no ponto B;
- Carga 3 - Buscar no ponto D, entregar no ponto B;
- Carga 4 - Buscar no ponto C, entregar no ponto A;

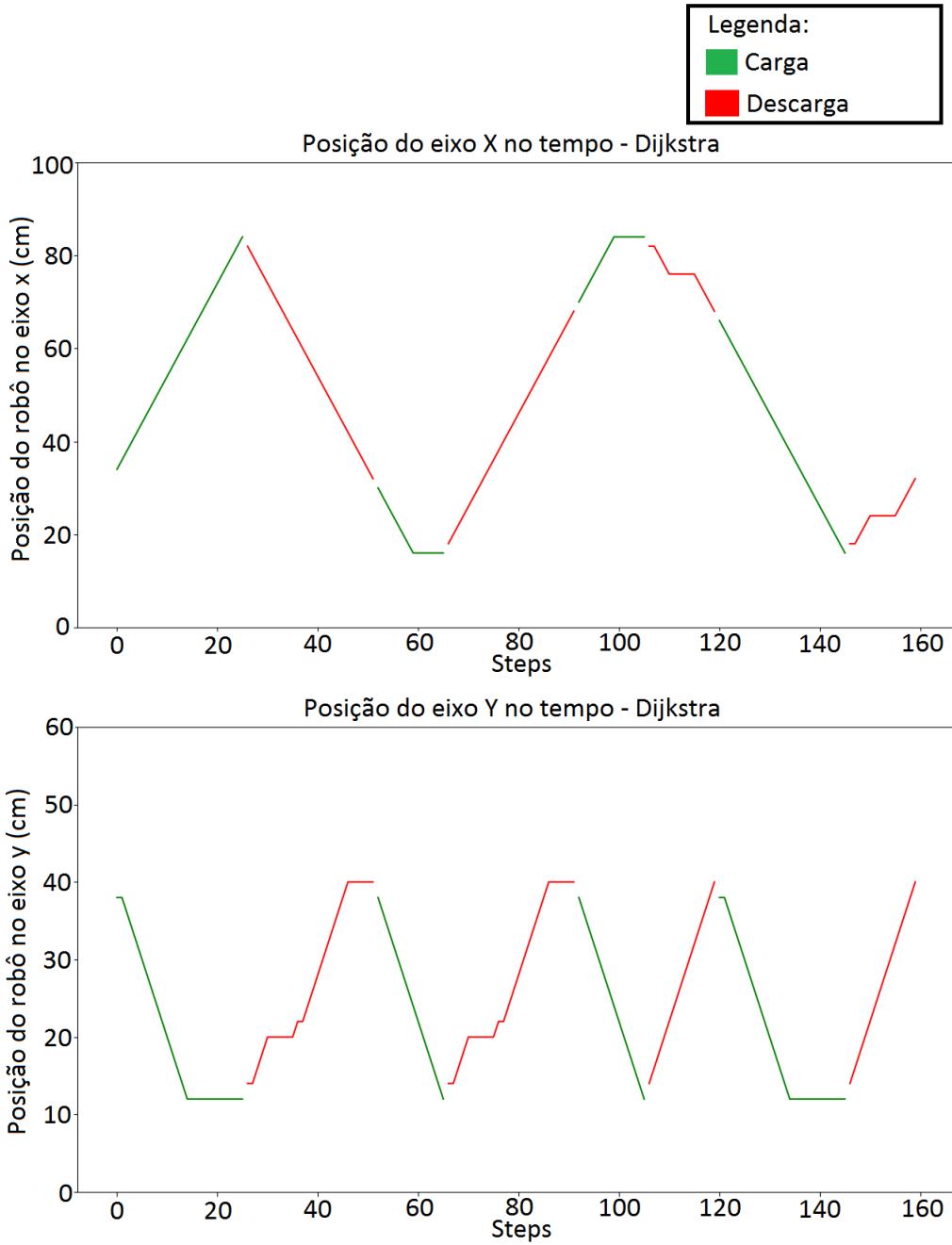
A cada grid o algoritmo possui três opções, percorrer o próximo grid na horizontal, vertical ou na diagonal, e, como vemos, em todos os casos foi encontrado o caminho ótimo entre os dois pontos. Em alguns casos como na Carga 1 e na Carga 3, os pontos de destino e de origem são os mesmos, porém foram obtidos diferentes rotas no trajeto de ida e de volta, ambos com distâncias iguais.

Figura 53 – Simulação Dijkstra - Trajetória para manipulação e transporte de 4 cargas



A [Figura 54](#) apresenta os valores do eixo X e do eixo Y conforme o trajeto executado na [Figura 53](#), a simulação possui como saída (resultado) a rota que deve ser executada, ou seja, uma lista de coordenadas (*Steps*) que o robô deve seguir. O transporte das 4 cargas estão compilados em um mesmo gráfico, onde as linhas em verde representam o trajeto para buscar a carga e em vermelho é o trajeto para transportar a carga até o seu destino. Portanto são apresentados 4 percursos para carga e 4 para descarga, de maneira alternada.

Figura 54 – Simulação Dijkstra - Trajetória executada nas direções x e y



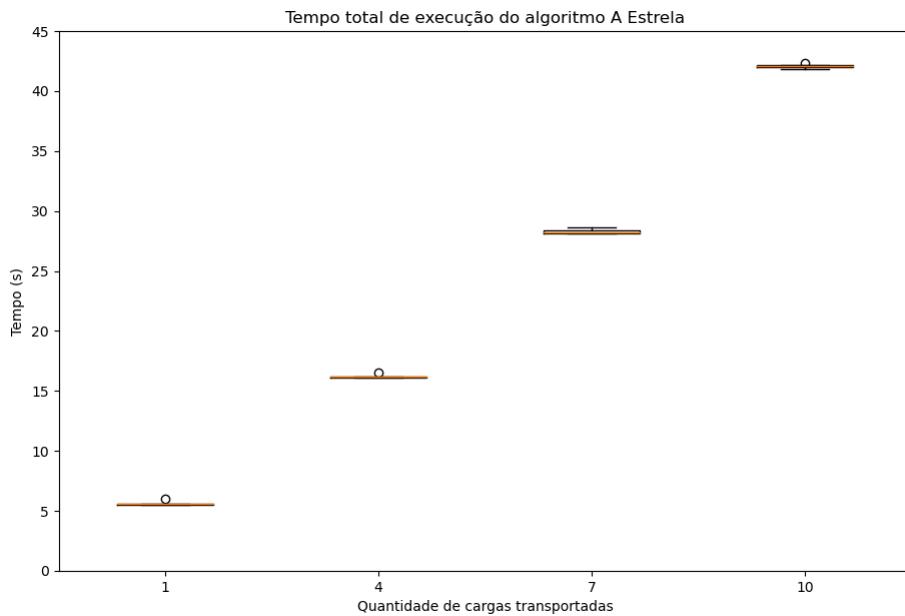
4.1.2 A-estrela

O mesmo conjunto de testes, apresentado na [Tabela 8](#), executado para o algoritmo Dijkstra, também foi executado para o algoritmo A-estrela. A [Figura 55](#) apresenta o tempo total que o algoritmo utilizou para calcular as rotas ótimas. Como podemos observar existe uma mínima variação dentro de uma mesma categoria, muito provavelmente isso se deve ao escalonador de processos do sistema operacional. Todavia, em categorias diferentes, podemos notar um aumento linear no tempo de execução.

Em comparação com o algoritmo Dijkstra (ver resultado na [Figura 51](#)), podemos observar

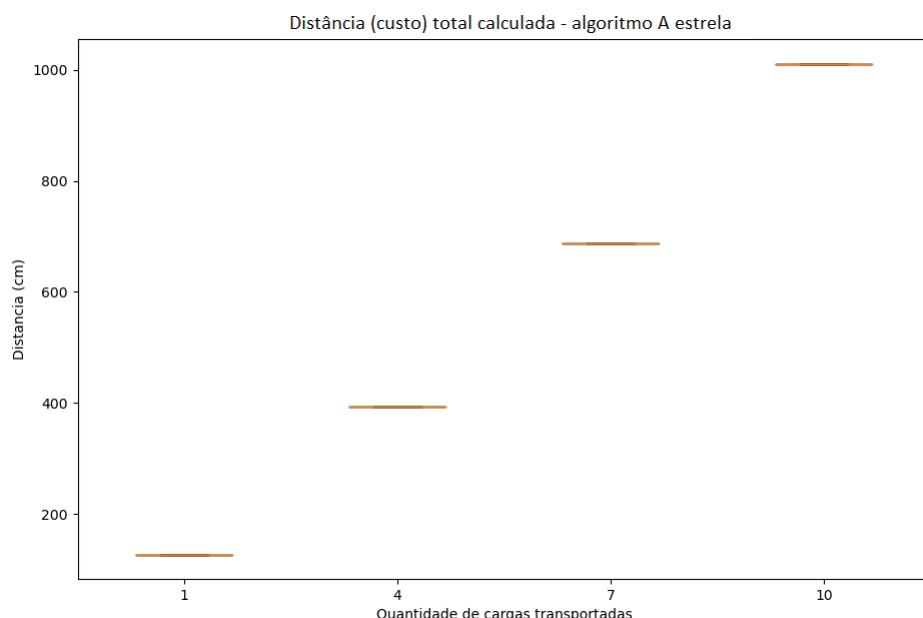
que o algoritmo A-estrela foi executado em um tempo significativamente menor, o que mostra que existe uma maior eficiência na utilização desse algoritmo no ambiente e nas condições simuladas.

Figura 55 – Simulação A-estrela - Tempo total



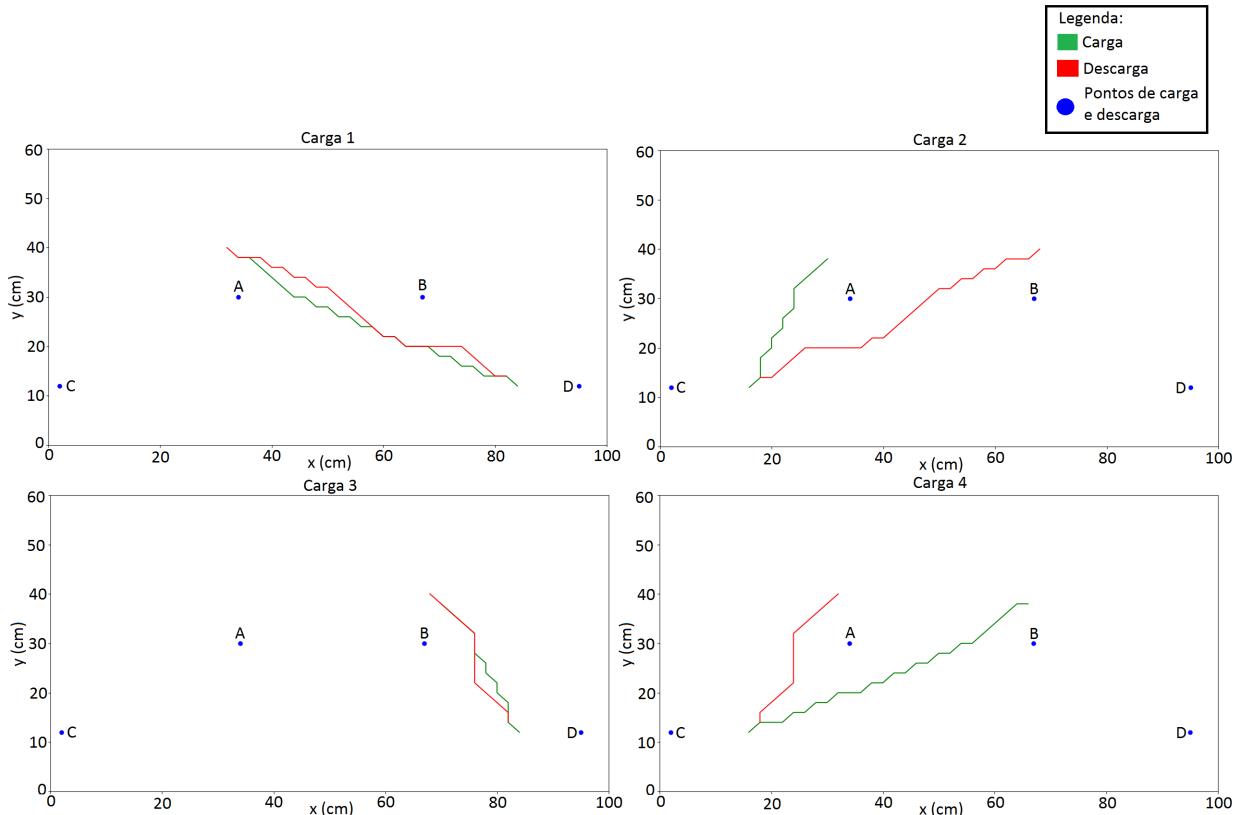
A [Figura 56](#) apresenta as distâncias ótimas obtidas, o resultado alcançado é idêntico ao resultado apresentado no algoritmo Dijkstra (ver resultado na [Figura 52](#)). Com isso podemos concluir que ambos os algoritmos encontraram as menores distâncias (os caminhos ótimos), isso também mostra que mesmo com um tempo de execução menor, o algoritmo A-estrela obteve um resultado final equivalente.

Figura 56 – Simulação A-estrela - Distância ideal estimada pelo algoritmo A-estrela



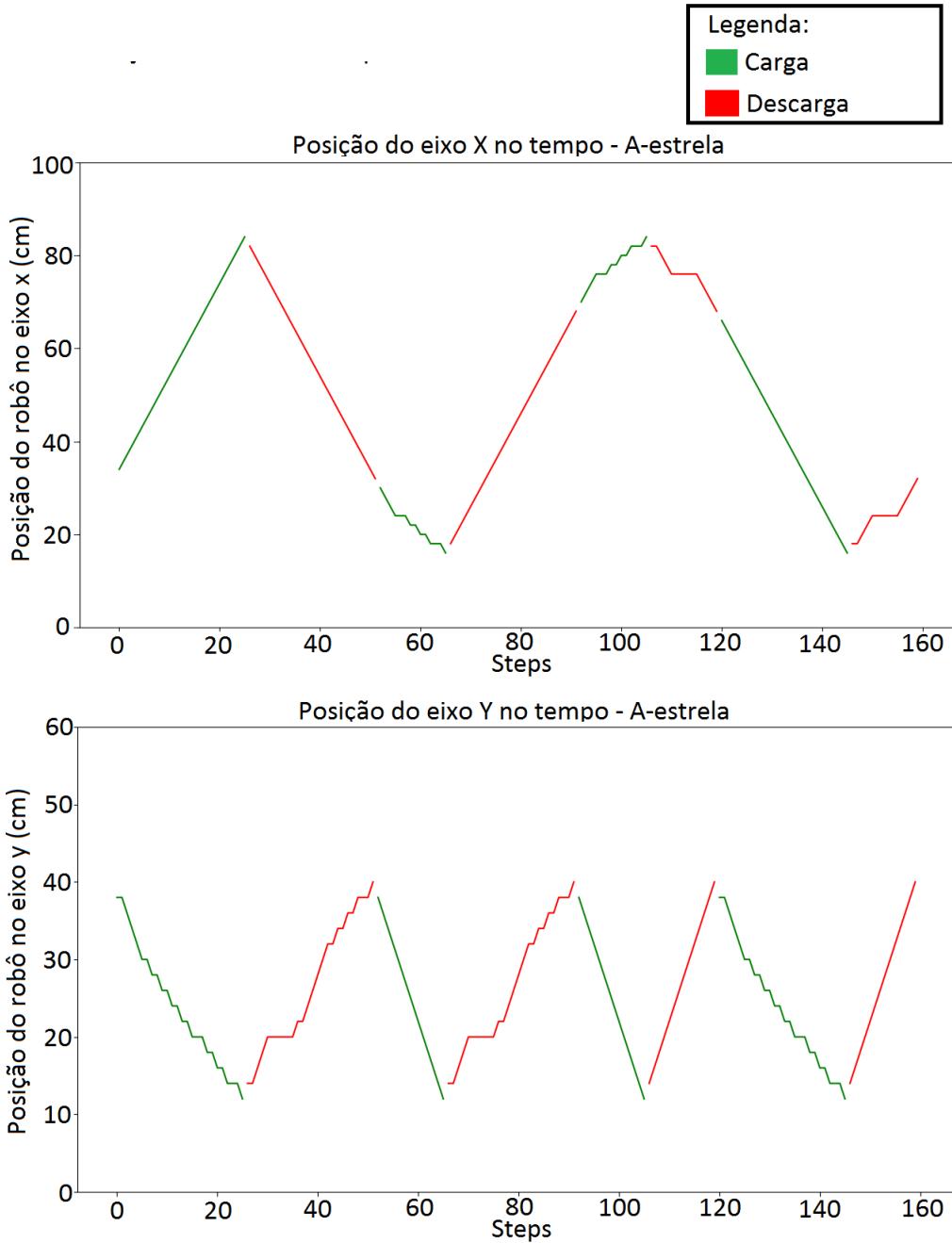
No teste realizado para transportar 4 cargas, a trajetória executada pelo robô em simulação é apresentada na [Figura 57](#). Podemos notar que mesmo com distâncias equivalentes, as trajetórias apresentadas são diferentes das trajetórias obtidas com o algoritmo Dijkstra. Os resultados atuais mostram trajetórias que tendem a ser mais próximas de uma trajetória linear.

[Figura 57 – Simulação A-estrela - Trajetória para manipulação e transporte de 4 cargas](#)



A [Figura 58](#) mostra os valores dos eixos X e Y. Como consequência do maior número de mudança de direção no trajeto, podemos observar diversos degraus no gráfico apresentado. Conforme o esperado, os valores obtidos são equivalente às trajetórias representadas na [Figura 57](#).

Figura 58 – Simulação A-estrela - Trajetória executada nas direções x e y



4.2 Avaliação Experimental

Nessa etapa também serão executados testes em quatro categorias, semelhante aos testes em simulação. No teste experimental temos uma interface desenvolvida em um site que nos permite inserir uma lista de tarefas que devem ser executadas de maneira sequencial, desse modo podemos programar o sistema para executar uma lista de tarefas desejadas.

Os experimentos irão contemplar todas as funcionalidades do sistema ciberfísico desenvolvido, incluindo a criação das tarefas na página *Web* e também o monitoramento do que

está sendo executado pelo robô. Então serão verificados o processamento de imagem a partir da câmera, o planejador de trajetórias, os controladores P de baixo nível, o sistema de desvio de obstáculos móveis e fixos e do acionamento dos servomotores e do braço robótico.

Todas as tarefas que serão expostas nos testes a seguir, foram criadas utilizando-se a interface *Web* vista na [Figura 59](#), nesse caso foi adicionado uma tarefa para retirar a carga no ponto 8 e entregar no ponto 3. Os pontos de carga e descarga e o grafo utilizado para representar o ambiente a partir de um mapa topológico são apresentados mais a frente.

Figura 59 – Página *Web* - Criando tarefa

Criar tarefas

Acompanhe em tempo real

Histórico de tarefas

- > Tarefa 9 - Carga:3 - Descarga:5
- > Tarefa 8 - Carga:0 - Descarga:3
- > Tarefa 7 - Carga:3 - Descarga:0
- > Tarefa 6 - Carga:3 - Descarga:0 - Carga:5 - Descarga:8
- > Tarefa 5 - Carga:0 - Descarga:3 - Carga:8 - Descarga:5
- > Tarefa 4 - Carga:3 - Descarga:8 - Carga:5 - Descarga:0
- > Tarefa 3 - Carga:0 - Descarga:5 - Carga:8 - Descarga:3
- > Tarefa 2 - Carga:3 - Descarga:0
- > Tarefa 1 - Carga:2 - Descarga:4 - Carga:2 - Descarga:4

Após clicar em "EXECUTAR" o robô deve realizar a tarefa solicitada, onde é possível acompanhar todo o processo pelo vídeo da vista superior do ambiente vista no campo "Acompanhe em tempo real" e, após a conclusão da tarefa, vemos no histórico a "Tarefa 10", como é ilustrado na [Figura 60](#).

Figura 60 – Página *Web* - Tarefa concluída

Criar tarefas

Acompanhe em tempo real

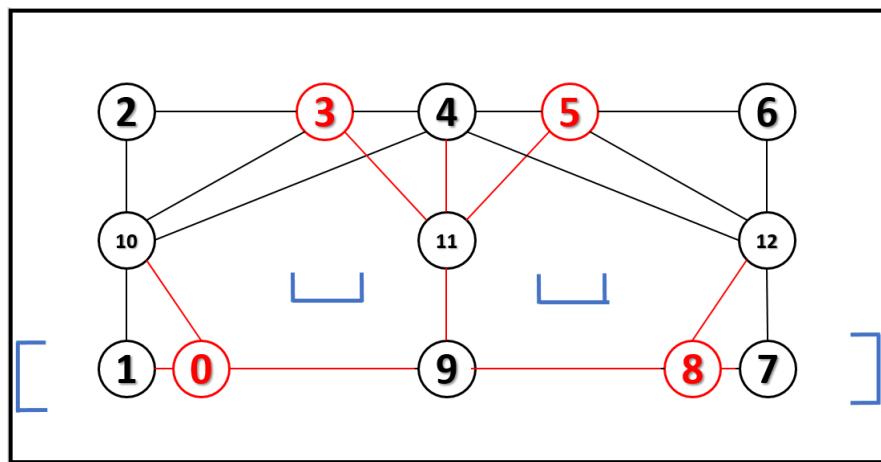
Histórico de tarefas

- > Tarefa 10 - Carga:3 - Descarga:3
- > Tarefa 9 - Carga:3 - Descarga:5
- > Tarefa 8 - Carga:0 - Descarga:3
- > Tarefa 7 - Carga:3 - Descarga:0
- > Tarefa 6 - Carga:3 - Descarga:0 - Carga:5 - Descarga:8
- > Tarefa 5 - Carga:0 - Descarga:3 - Carga:8 - Descarga:5
- > Tarefa 4 - Carga:3 - Descarga:8 - Carga:5 - Descarga:0
- > Tarefa 3 - Carga:0 - Descarga:5 - Carga:8 - Descarga:3
- > Tarefa 2 - Carga:3 - Descarga:0
- > Tarefa 1 - Carga:2 - Descarga:4 - Carga:2 - Descarga:4

O mapa topológico desenvolvido, definido por um grafo não orientado, é apresentado na Figura 61. O planejamento da rota será realizado pelo algoritmo Dijkstra a partir de uma busca no grafo que representa o ambiente. Algumas características adicionais sobre esse grafo são apontadas a seguir.

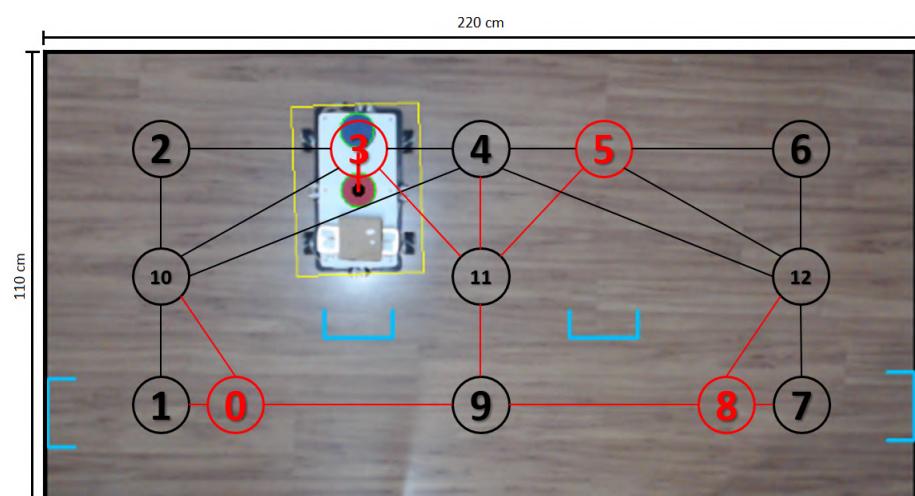
- Vértices vermelhos: utilizados para realizar carga ou descarga;
- Vértices pretos: utilizado apenas para passagem;
- Areias pretas: permite movimento omnidirecional (valor 2 na Tabela 4);
- Areias vermelhas: permite movimento apenas para frente ou para trás (valor 0 na Tabela 4).

Figura 61 – Mapa topológico - Grafo completo



A Figura 62 mostra o grafo sobreposto no ambiente utilizado, com isso podemos ter uma ideia do espaço, da localização da plataforma robótica, do ambiente e de possíveis rotas.

Figura 62 – Mapa topológico - Sobreposto no ambiente



4.2.1 Sem obstáculos

O primeiro conjunto de testes executado está apresentado na [Tabela 9](#), divididos em quatro categorias, sendo elas com 1, 4, 7 e 10 cargas transportadas. Nota-se que os testes são iguais aos testes executados em simulação, entretanto a nomenclatura dos pontos de carga e descarga foi modificada para se tornar equivalente aos nós de carga e descarga (em vermelho na [Figura 62](#)), diferentemente dos testes em simulação, em que o mapa utilizado é o métrico e não se faz o uso de um grafo.

Tabela 9 – Categorias de testes experimental - parte 1

Categorias	Carga 1	Desc 1	Carga 2	Desc 2	Carga 3	Desc 3	Carga 4	Desc 4
1 Carga	8	3						
4 Cargas	8	3	0	5	8	5	0	3
7 Cargas	8	3	0	5	8	5	0	3
10 Cargas	8	3	0	5	8	5	0	3

Tabela 9: Categorias de testes experimental - parte 2

Categorias	Carga 5	Desc 5	Carga 6	Desc 6	Carga 7	Desc 7	Carga 8	Desc 8
1 Carga								
4 Cargas								
7 Cargas	8	3	0	5	8	5		
10 Cargas	8	3	0	5	8	5	8	3

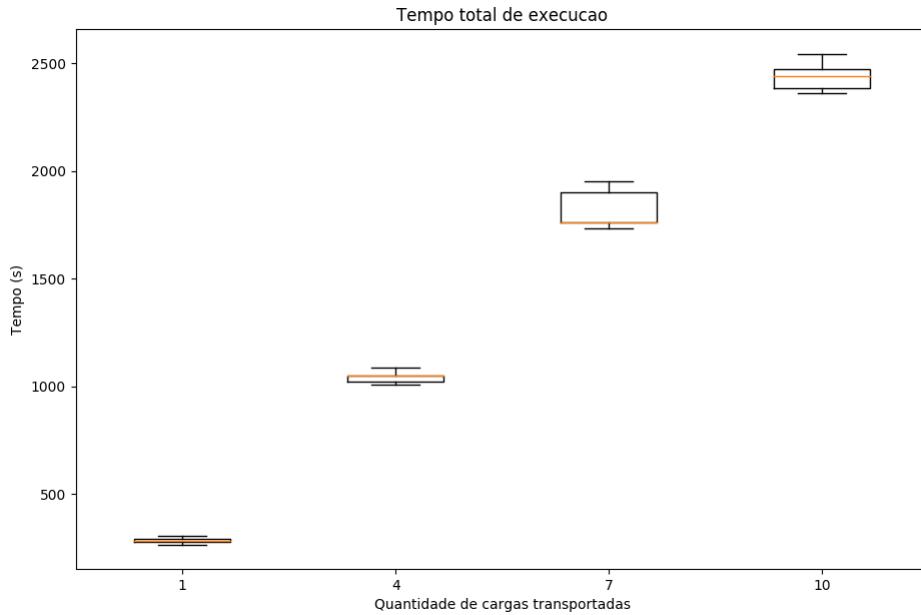
Tabela 9: Categorias de testes experimental - parte 3

Categorias	Carga 9	Desc 9	Carga 10	Desc 10
1 Carga				
4 Cargas				
7 Cargas				
10 Cargas	0	5	8	5

Durante a bateria de testes, foram registrados diversos dados, na [Figura 63](#) podemos visualizar o tempo total utilizado para concluir as tarefas de manipulação e transporte das cargas. Podemos notar quatro diferentes conjuntos de valores de tempo, uma para cada categoria, cada categoria foi executada 5 vezes. De maneira geral vemos uma variação no tempo em uma mesma categoria, proveniente do fato de existir algumas características inerentes ao sistema real, como por exemplo o tempo para realizar a comunicação *Wi-Fi* e o derrapamento do robô durante a execução da trajetória.

Entre as diferentes categorias vemos um aumento (variação) linear do tempo, o que mostra que o tempo de execução para cada carga individualmente se manteve aproximadamente constante.

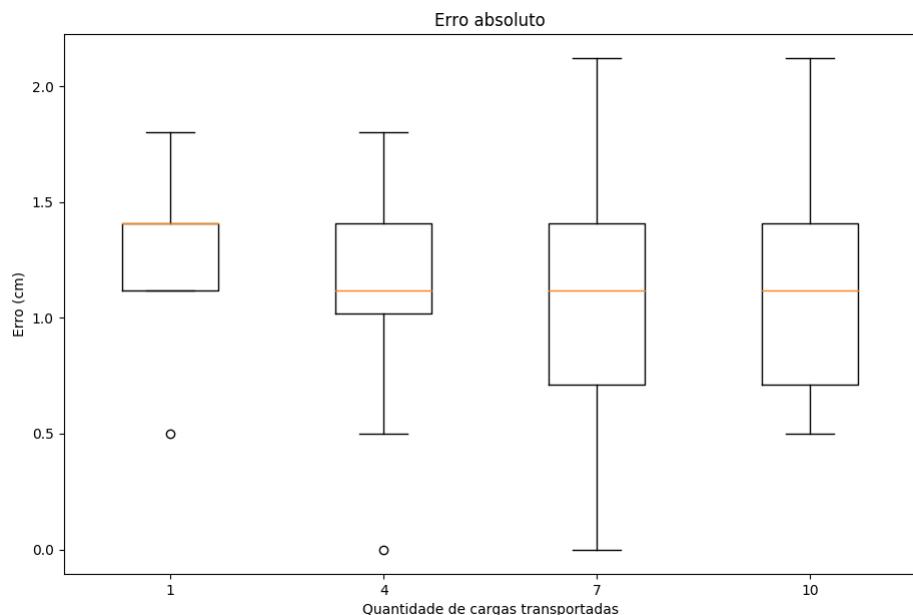
Figura 63 – Experimental - Tempo total



Além de planejar e percorrer o trajeto, o experimento também deve realizar cargas e descargas, é possível que exista um erro de posicionamento para realizar o procedimento de descarga, que é definido pela diferença entre a posição ideal de descarga (definida pela posição do nó de manipulação de carga) e a posição atingida pelo robô. O erro absoluto em centímetros pode ser visto na [Figura 64](#).

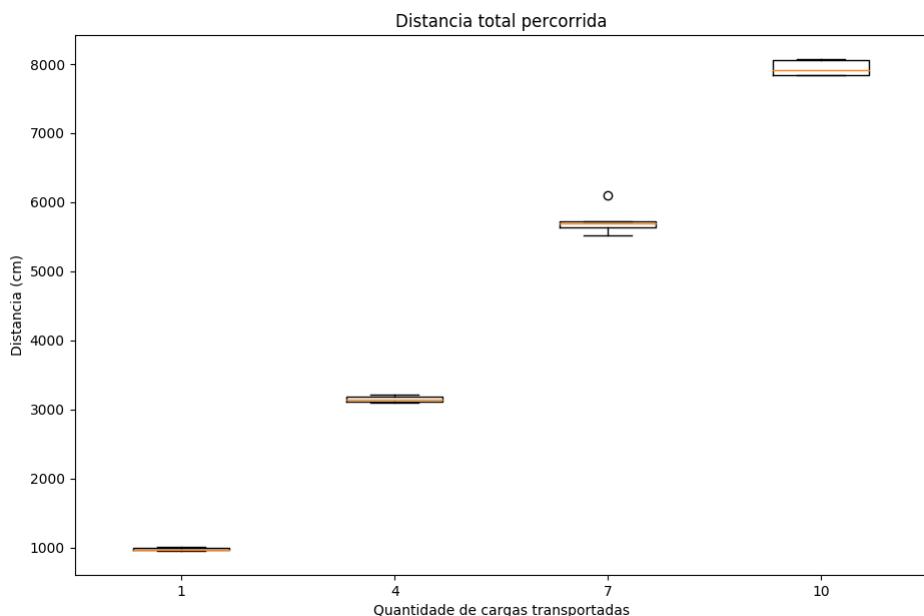
Como podemos observar, o erro varia entre 0,5 cm e 1,8 cm para a descarga de apenas 1 carga; quando são executados o transporte e manipulação de mais cargas, vemos um aumento desse intervalo, que varia entre 0 cm e 2,1 cm.

Figura 64 – Experimental - Erro absoluto para descarregar



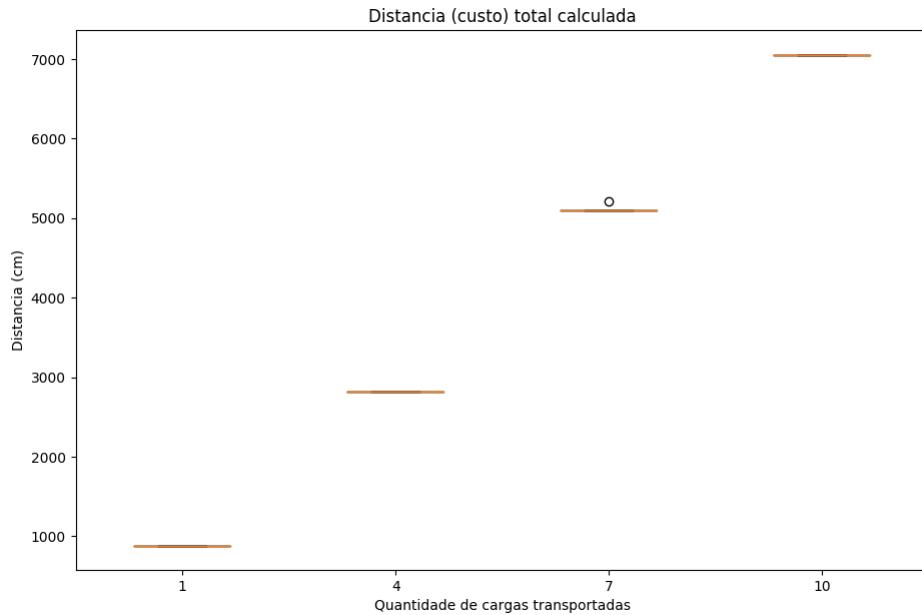
A distância total percorrida pelo robô em cada experimento pode ser vista na [Figura 65](#). Os valores obtidos referem-se a soma de todo o trajeto percorrido pelo robô desde o início, até a entrega da última carga. No gráfico apresentado vemos que a distância percorrida possui uma pequena variação dentro de cada categoria, essa variação ocorre devido a pequenas variações e erros nas trajetórias, que desviam da rota originalmente planejada. Essas variações, quando acima dos limites determinados, são corrigidas pelos controladores P e, no final, resulta em uma distância total percorrida maior do que a planejada.

Figura 65 – Experimental - Distância total executada pelo robô



Na [Figura 66](#) podemos observar a distância total do trajeto planejado, conforme o esperado, o caminho ótimo planejado se manteve aproximadamente igual em uma mesma categoria, apenas em um dos testes para o transporte de 7 cargas foi calculado uma rota maior, em decorrência de uma diferença entre a posição inicial do robô e o nó inicial do trajeto, com isso o algoritmo precisou calcular um pequeno deslocamento até o nó inicial. Esse deslocamento da posição inicial ocorreu, pois o robô foi colocado manualmente próximo da posição desejada, portanto, como não estava dentro dos limites do grafo, o algoritmo precisou realizar essa correção na posição.

Figura 66 – Experimental - Distância total ideal estimada pelo algoritmo Dijkstra

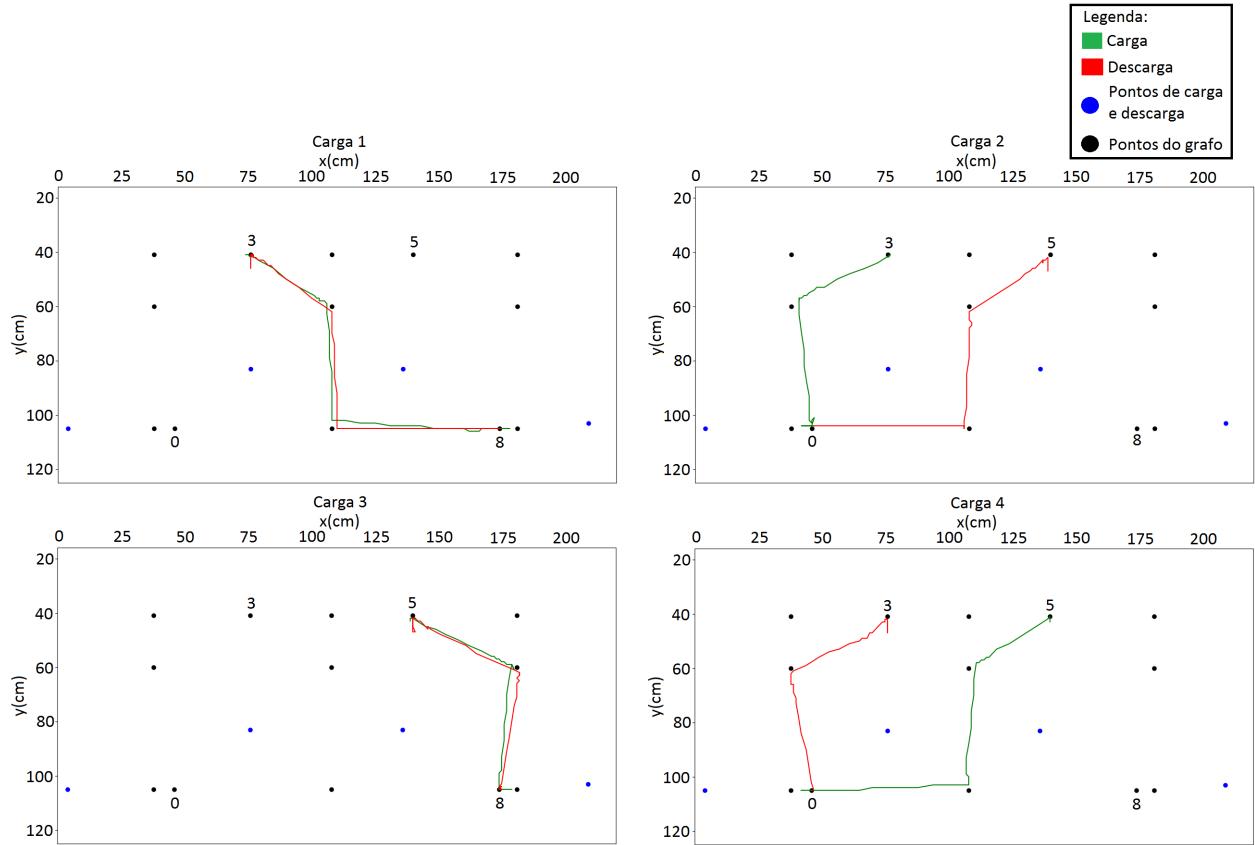


No teste prático também foi feito o registro da trajetória completa percorrida pelo robô para deslocar 4 cargas, o teste se inicia com o robô no vértice 3 que, no eixo cartesiano apresentado na [Figura 67](#), está localizado na posição (75, 40). As rotas percorridas pelo robô são apresentadas em verde e vermelho, as linhas em verde representam o trajeto percorrido para buscar a carga e o trajeto em vermelho determina o caminho percorrido para realizar a entrega (descarga).

Como pode ser observado, em alguns casos a trajetória executada pelo robô se aproxima do nó desejado, entretanto não alcança a posição ideal definida na trajetória (exatamente no centro do nó), isso ocorre com maior frequência nos nós de passagem que é permitido um erro absoluto de até 5,65 cm. Quando se trata dos nós de carga e descarga (onde é feita a manipulação da carga) é aceito um erro absoluto de até 1,41 cm.

O robô pode precisar de um tempo maior para garantir erros menores, por causa do tempo que será despendido com movimentos lentos e suaves para alcançar a posição desejada. Portanto é permitido um erro maior nos nós de passagem, reduzindo assim o tempo de execução de uma determinada tarefa sem afetar o seu correto desempenho, diferentemente dos pontos de carga e descarga que precisam de uma excelente precisão (menor erro possível).

Figura 67 – Experimental - Trajetória executada pelo robô para manipulação e transporte de 4 cargas

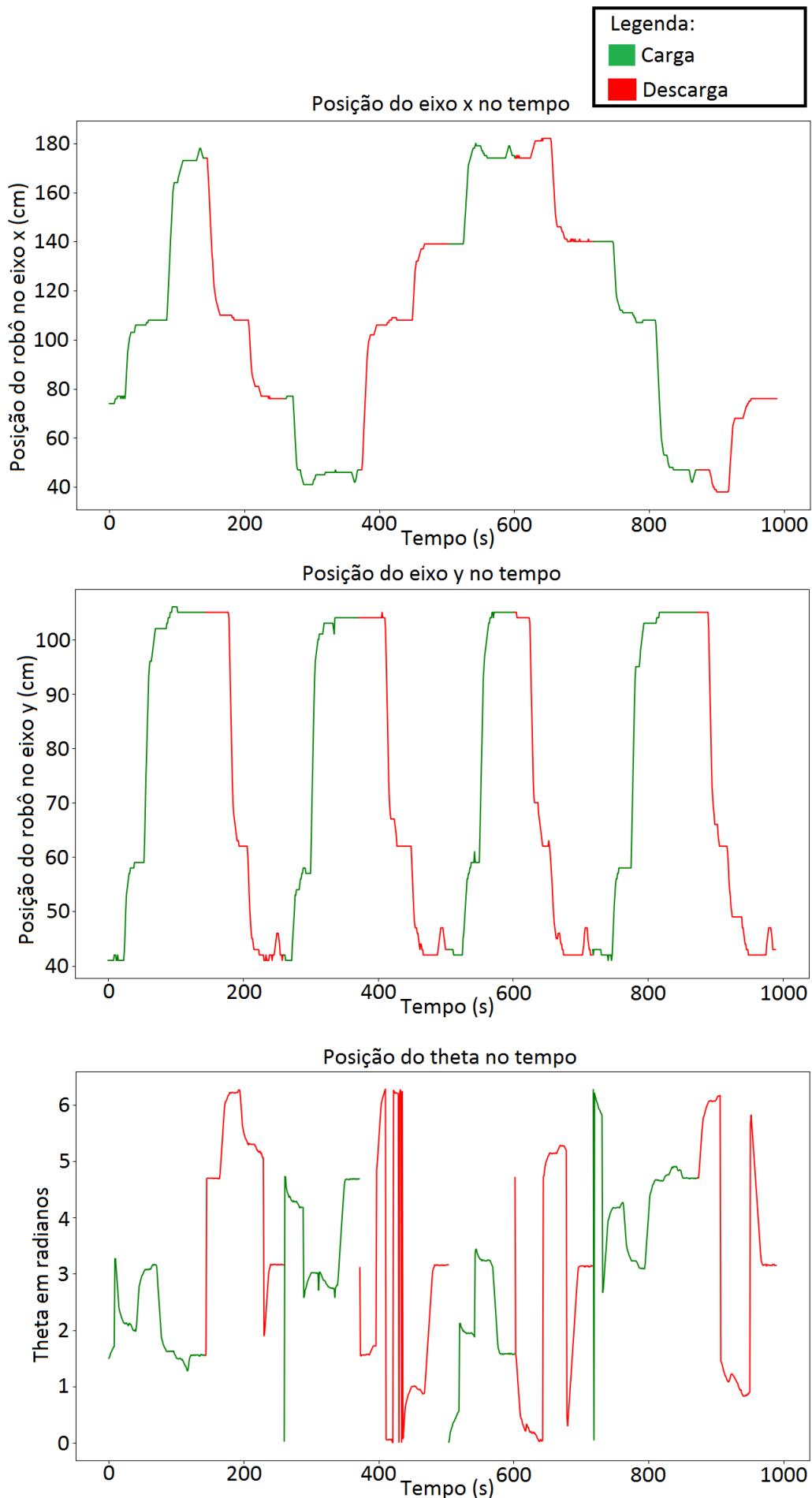


A [Figura 68](#) ilustra os valores de deslocamento no eixo X, eixo Y e também da orientação (theta), conforme visto nos capítulos anteriores theta obtido foi acrescido de π , fazendo com que seu intervalo de valores fique entre 0 e 2π .

Nos gráficos podemos observar que os valores dos eixos equivalem a trajetória exposta anteriormente. Já o valor do theta mostra uma informação nova que é a direção que o robô está em um determinado momento. Como referência, quando o robô está se deslocando na vertical para cima (sentido do vértice 1 para 2, veja na [Figura 61](#)) o Theta deve ser 0 ou 2π , no caso de se deslocar no sentido oposto, o valor obtido será π .

Outro ponto de interesse é quando ocorre a mudança direta de 0 para 2π , essa mudança não é instantânea, o que nos gera uma mudança gradativa mas bastante rápida. Essas variações ficam bastante evidentes próximo dos 400 segundos, onde alguns traços verticais podem ser vistos na [Figura 68](#).

Figura 68 – Experimental - Trajetória executada nas direções x e y, e orientação (theta)

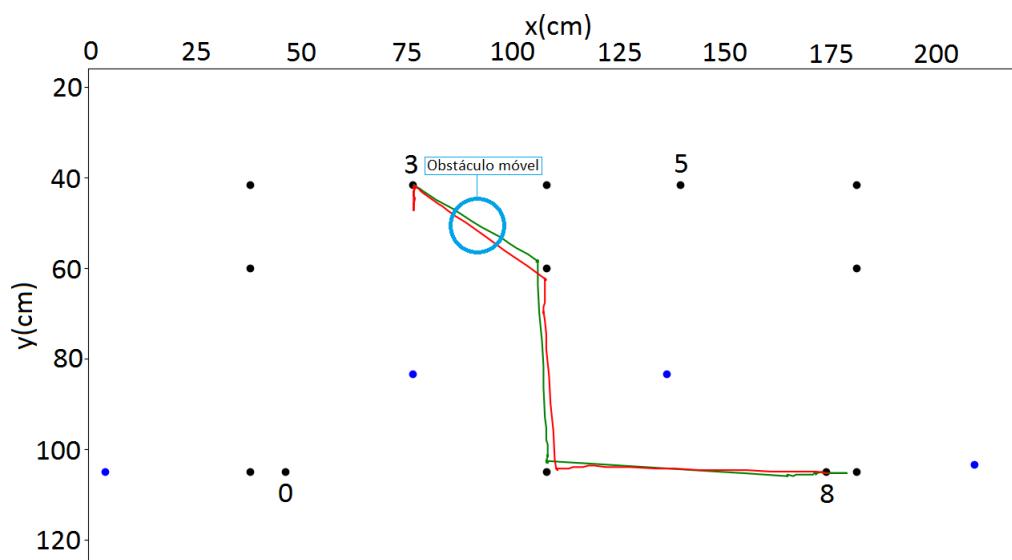


4.2.2 Obstáculo móvel

No caso de encontrar um obstáculo móvel a menos de 10 cm, o robô deve parar instantaneamente e iniciar o seu contador de tempo. Se o caminho for desobstruído em menos de 5 segundos o robô irá continuar em sua rota original. Caso contrário, o obstáculo será considerado fixo e um novo planejamento será realizado. No caso de encontrar novos obstáculos o contador de tempo é reiniciado.

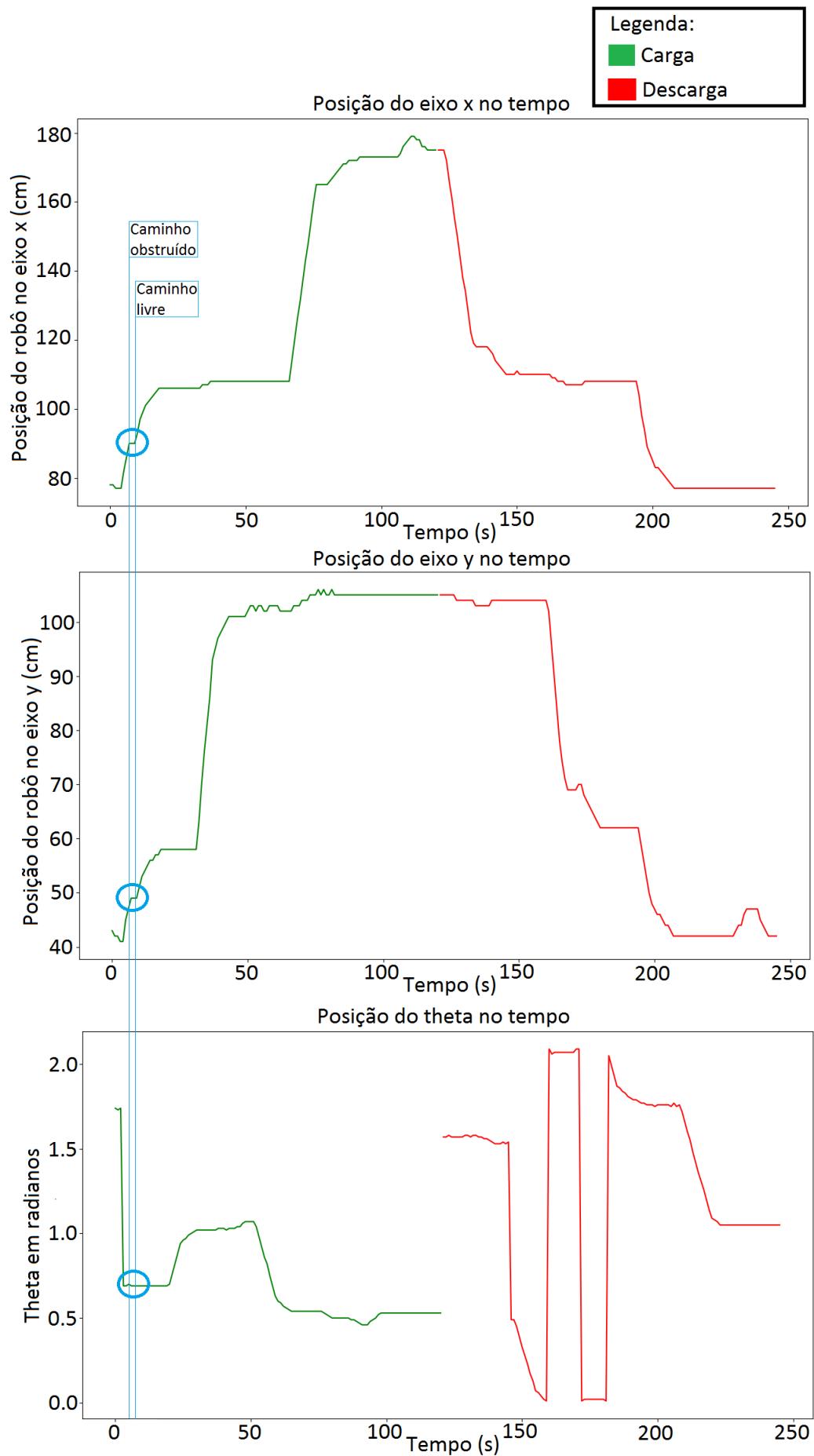
O trajeto executado para a manipulação e transporte de 1 carga e com um obstáculo móvel (destacado em azul) é apresentado na [Figura 69](#).

Figura 69 – Experimental - Trajetória executada pelo robô com um obstáculo móvel



Na [Figura 70](#) podemos visualizar os valores no plano x e y e do theta durante o trajeto, também é visto em destaque o intervalo em que o caminho estava obstruído e o robô ficou parado aguardando a sua desobstrução.

Figura 70 – Experimental - Trajetória executada nas direções x e y, e orientação (theta), com um obstáculo móvel

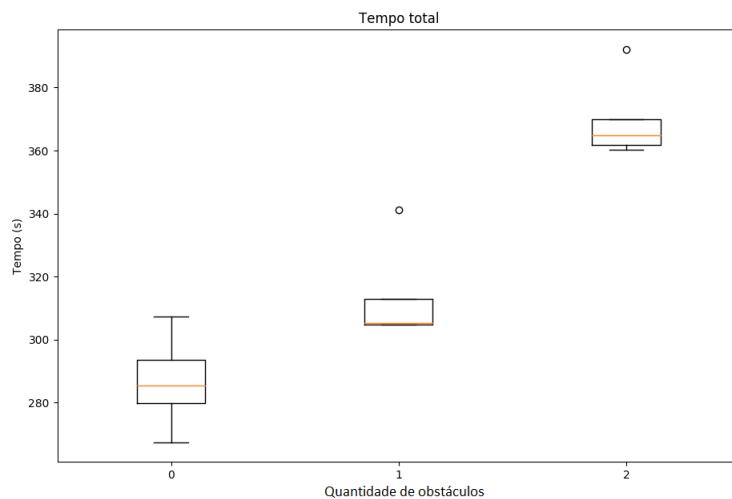


4.2.3 Obstáculo fixo

Após o robô detectar um obstáculo por mais de 5 segundos consecutivos, ele é considerado um obstáculo fixo e com isso o robô deve calcular uma rota alternativa, se existir.

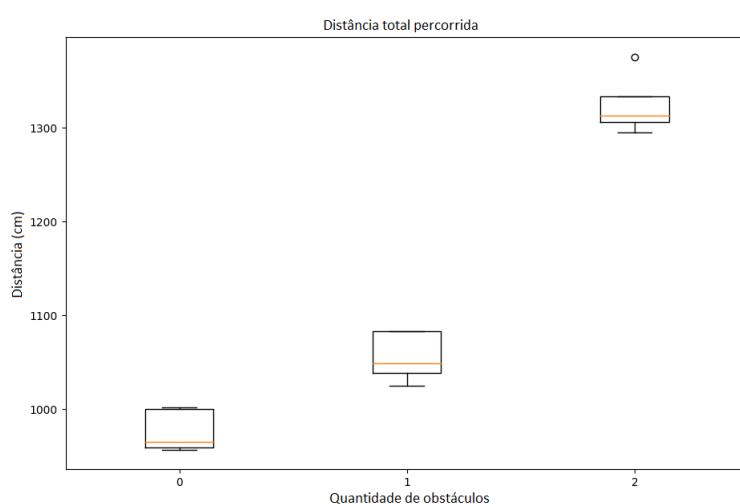
Foram realizados três diferentes testes e executados 5 vezes para cada um deles. O primeiro é sem obstáculo que deve ser utilizado como parâmetro, o segundo caso é com 1 obstáculo e o terceiro são utilizados 2 obstáculos. Na [Figura 71](#) podemos observar o tempo total para a manipulação e transporte de 1 carga nos três casos supramencionados.

Figura 71 – Experimental - Tempo total com obstáculos fixos



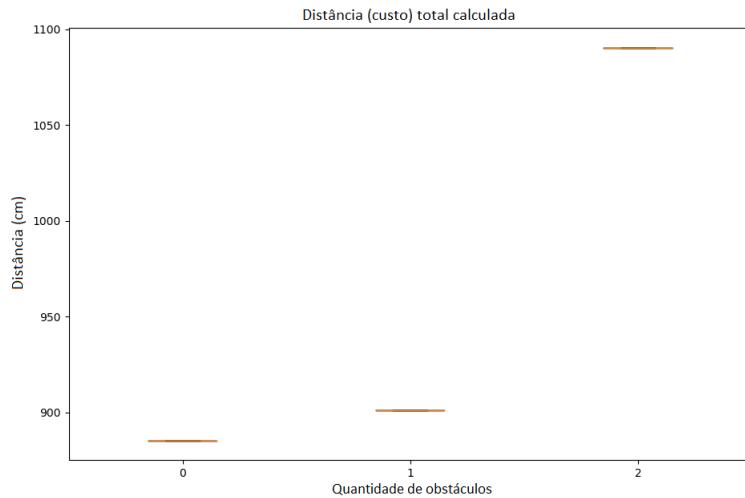
Na [Figura 72](#) podemos visualizar que a distância total percorrida pelo robô é diretamente proporcional ao gráfico com o tempo total visto na [Figura 71](#). Portanto, com apenas 1 obstáculo o robô conseguiu encontrar uma rota alternativa com um pequeno acréscimo da distância, entretanto com dois obstáculos foi preciso fazer um contorno maior, resultando em uma rota alternativa mais longa e demorada.

Figura 72 – Experimental - Distância total percorrida com obstáculos fixos



A distância obtida a partir do trajeto ótimo planejado pelo algoritmo é vista na [Figura 73](#), podemos observar que por se tratar de uma rota ideal estimada pelo algoritmo Dijkstra, os valores são menores do que o trajeto efetivamente feito pelo robô, entretanto podemos observar a mesma variação no tamanho das rotas com o aumento do número de obstáculos.

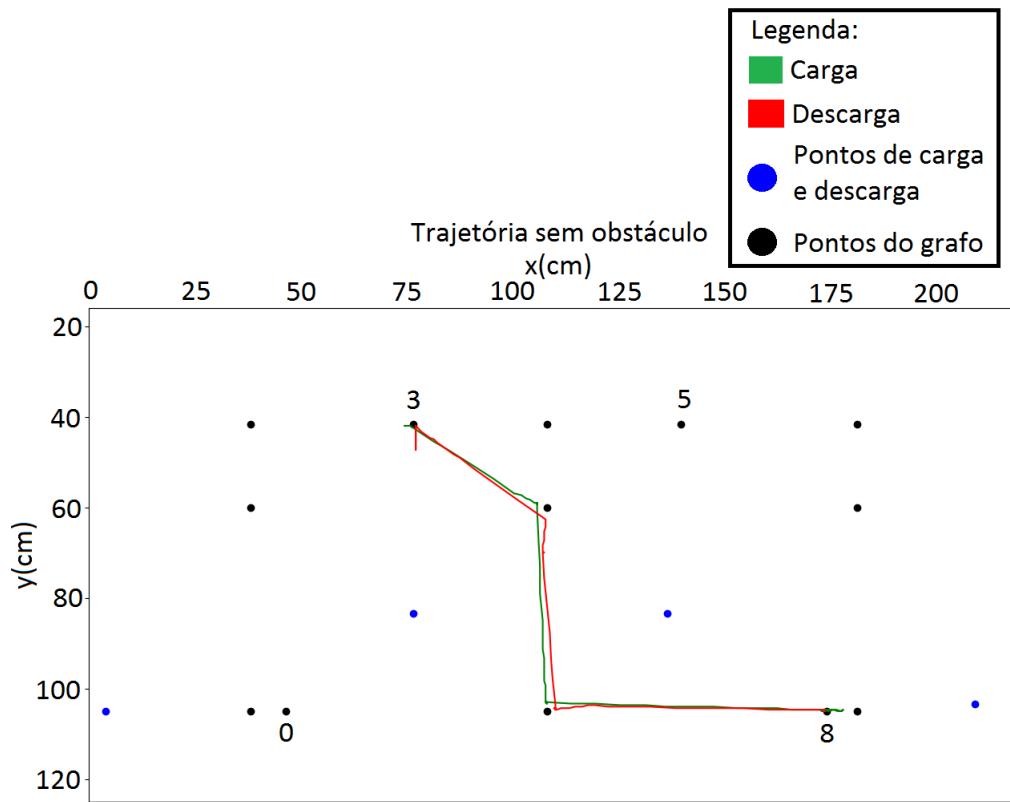
Figura 73 – Experimental - Distância total ideal estimada pelo Dijkstra com obstáculos fixos



4.2.3.1 Sem obstáculo fixo

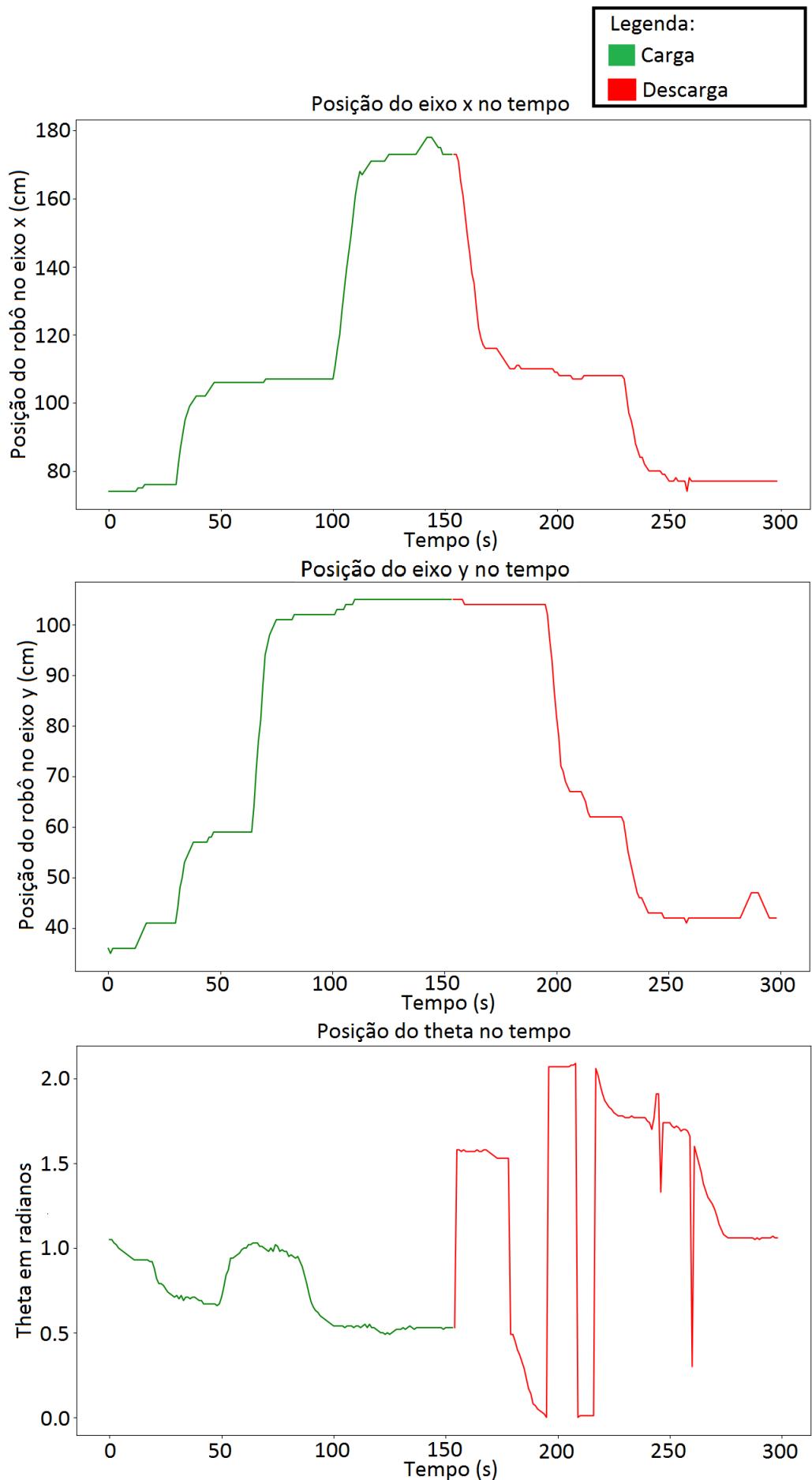
A rota percorrida pelo robô em um ambiente sem obstáculos é apresentada na [Figura 74](#), o deslocamento da carga é feito com sucesso do ponto 8 até o ponto 3, sem desvios de rotas.

Figura 74 – Experimental - Trajetória sem obstáculo fixo no plano x y



Os valores das trajetórias em x e y e a orientação (theta) no tempo são vistos na [Figura 75](#), que refletem a trajetória vista anteriormente, sem desvios de rotas e sem paradas.

Figura 75 – Experimental - Trajetória executada nas direções x e y, e orientação (theta), sem obstáculo fixo

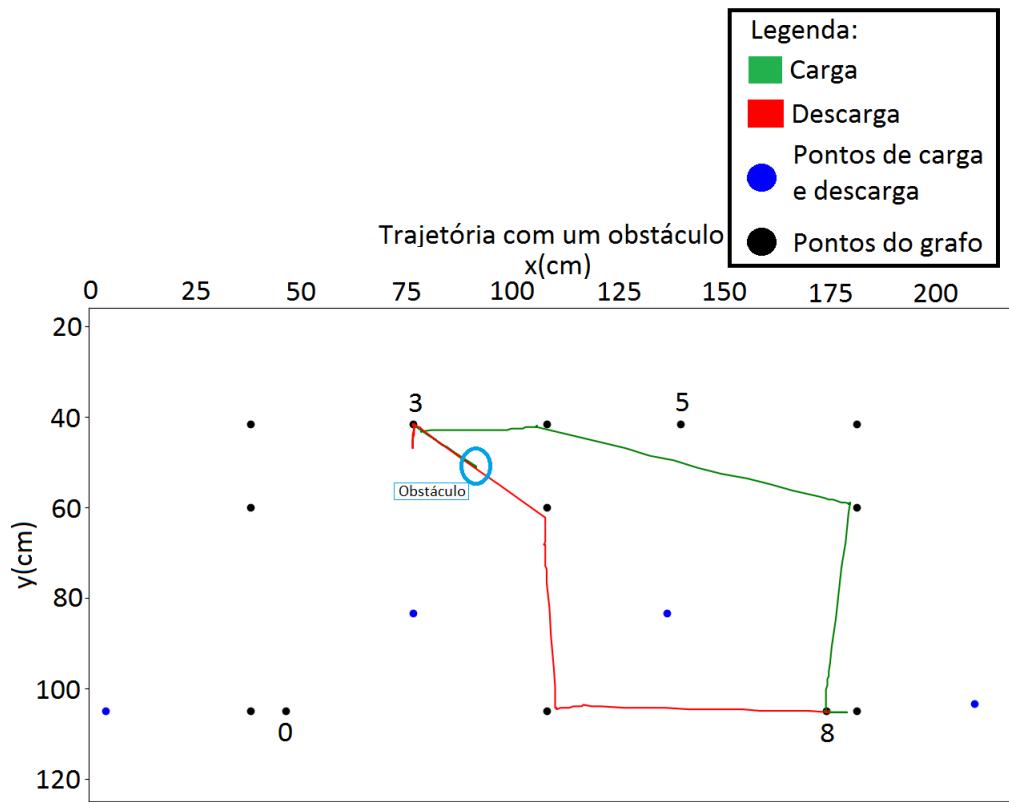


4.2.3.2 Um obstáculo fixo

Na manipulação e transporte de uma carga com um obstáculo fixo, destacado com um círculo azul na [Figura 76](#), podemos observar que, após encontrar uma obstrução, uma nova rota ótima foi calculada sem considerar a aresta obstruída. Com isso foi obtido um trajeto maior do que o trajeto anterior.

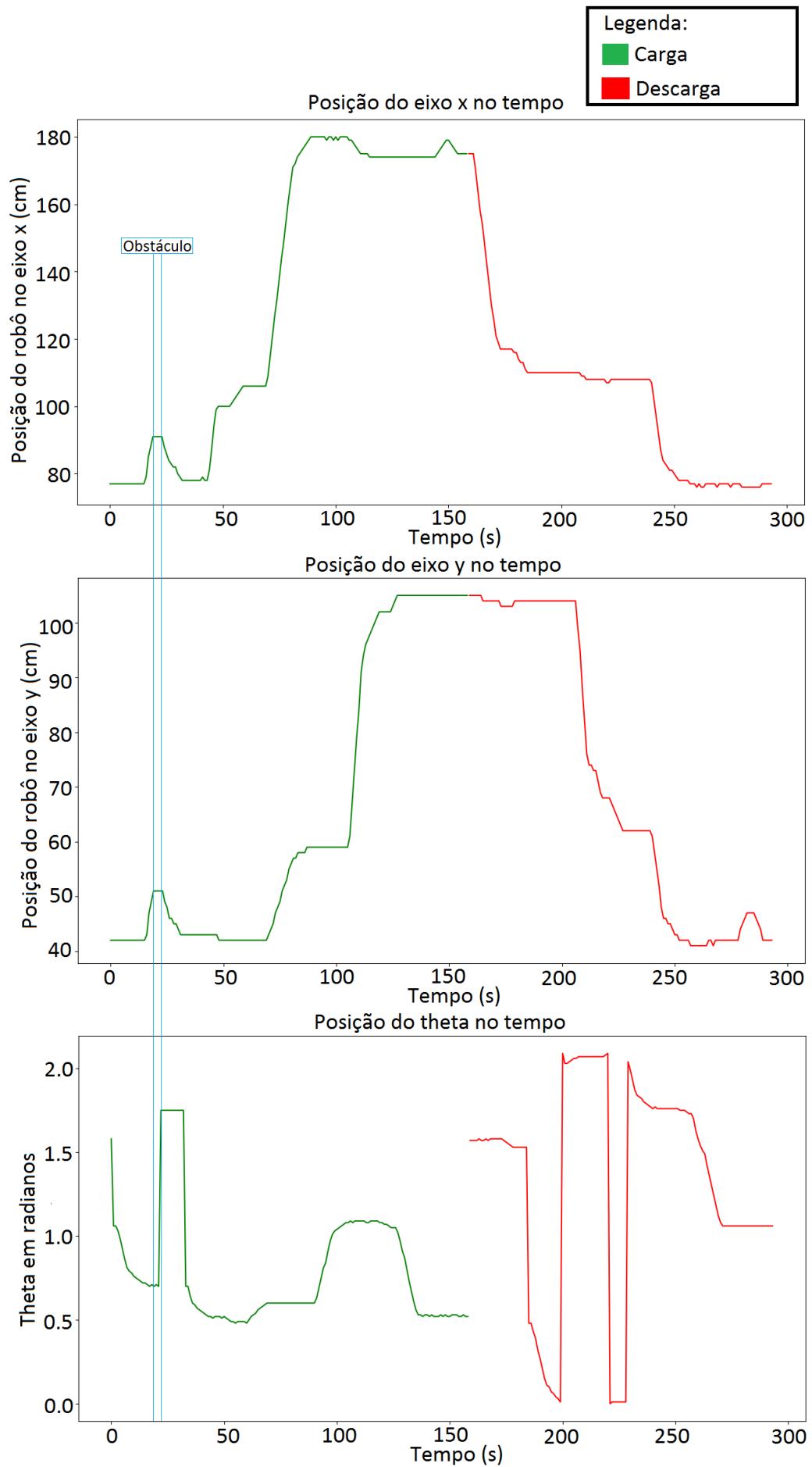
A obstrução foi feita apenas no trajeto de ida, onde o robô partiu do ponto 3 até o ponto 8. No trajeto de volta, onde o intuito era entregar a carga no ponto 3 o trajeto foi feito sem obstrução.

[Figura 76 – Experimental - Trajetória executada pelo robô com um obstáculo fixo](#)



Na [Figura 77](#) podemos observar no decorrer do tempo o intervalo em que o robô ficou parado aguardando 5 segundos e, como a aresta estava obstruída, foi preciso retornar ao nó 3 e executar a rota obtida no novo planejamento.

Figura 77 – Experimental - Trajetória executada nas direções x e y, e orientação (theta), com um obstáculo fixo

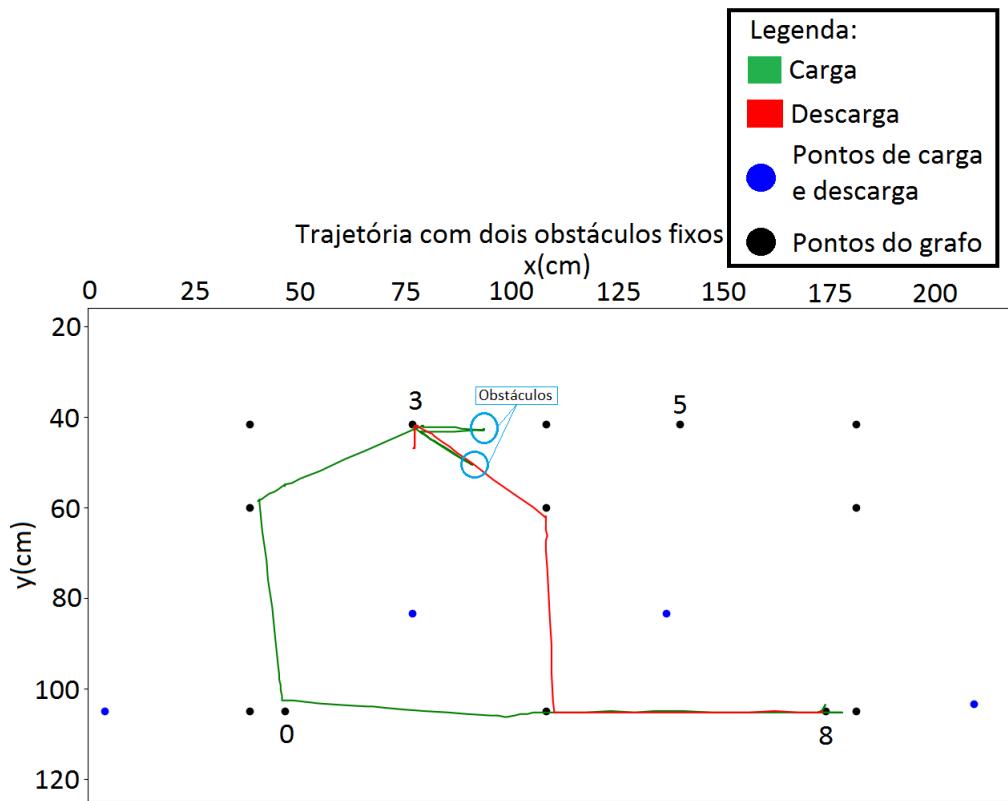


4.2.3.3 Dois obstáculos fixos

Na manipulação e transporte de uma carga, entre o nó 3 (carga) e o nó 8 (descarga), com dois obstáculos fixos destacados em azul na [Figura 78](#), podemos visualizar em verde toda a trajetória percorrida pelo robô. Quando um obstáculo é encontrado, é realizado um novo planejamento usando o algoritmo Dijkstra e depois o robô executa a rota obtida no novo planejamento, esse procedimento foi realizado duas vezes nesse experimento (uma vez para cada obstáculo).

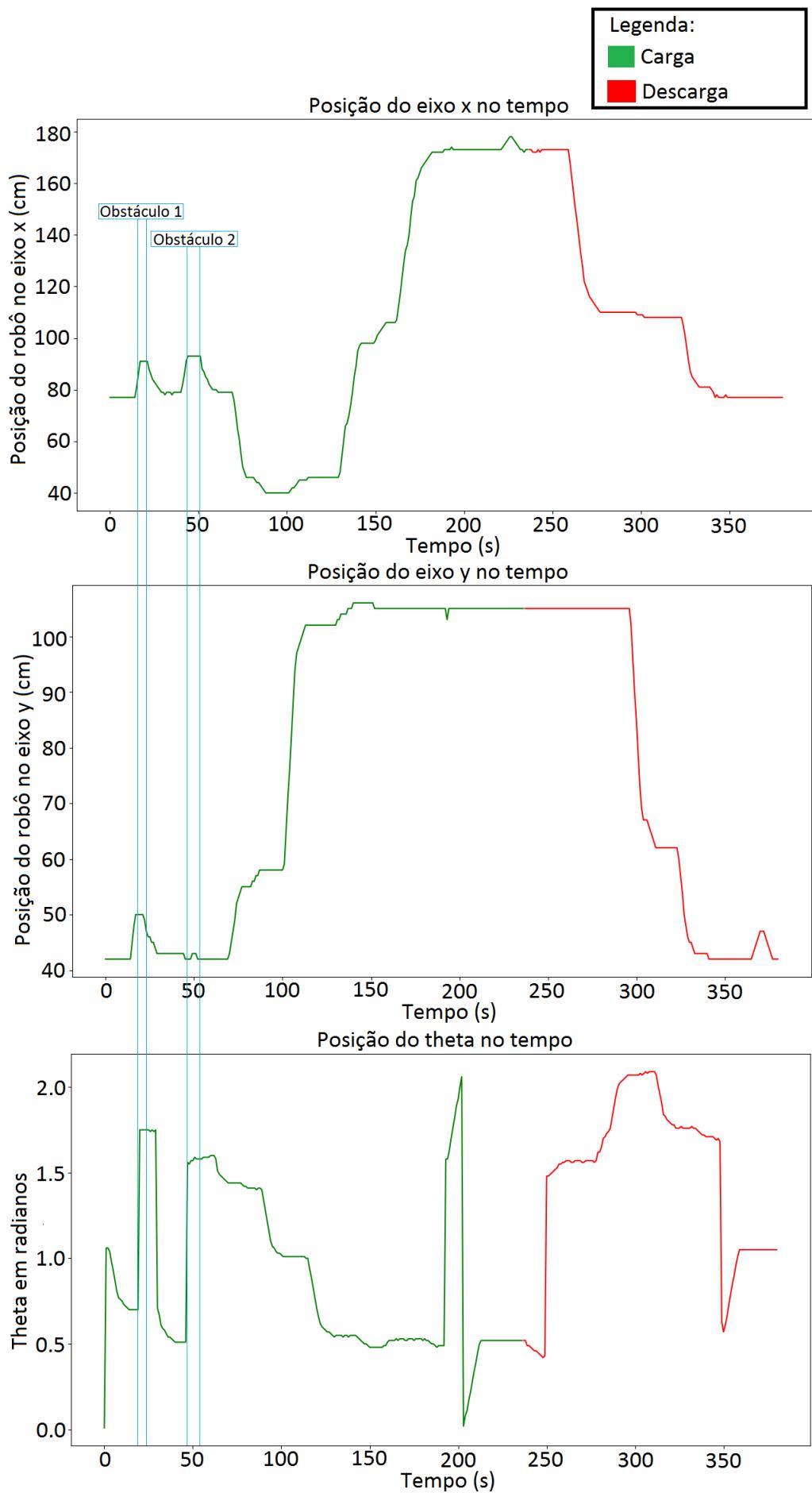
Conforme o esperado, foi preciso fazer um caminho bem mais longo do que os dois trajetos apresentados anteriormente.

[Figura 78 – Experimental - Trajetória executada pelo robô com dois obstáculos fixos](#)



Nos gráficos expostos na [Figura 79](#) vemos em destaque os dois momentos que o robô encontrou os obstáculos fixos, nesses intervalos de tempo o robô estava parado aguardando 5 segundos para confirmar que se tratava de um obstáculo fixo. Após as confirmações, em ambos os casos, o robô retornou ao nó 3 e executou a rota obtida no novo planejamento. No eixo Y do segundo obstáculo vemos uma pequena variação, que deve ser oriunda de uma mudança na imagem captada pela câmera, que pode ter afetado o processamento de imagem.

Figura 79 – Experimental - Trajetória executada nas direções x e y, e orientação (theta), com dois obstáculos fixos



5 Conclusão e Trabalhos Futuros

O projeto apresentado é bastante amplo, abrangendo diversos conceitos de diferentes áreas do conhecimento, diferentes tecnologias e diversos dispositivos integrados. No planejamento do trabalho necessitou-se visualizar todos os procedimentos e componentes do projeto, para isso foram criadas etapas e, a cada etapa, se tornou possível planejar a próxima com uma análise mais detalhada. Apesar de não ter sido utilizada nenhuma metodologia de planejamento e gestão de projeto, todo o desenvolvimento se assemelhou a um desenvolvimento ágil, pois a todo momento existiam pequenas tarefas a serem desenvolvidas e possíveis adaptações no projeto.

Durante todas as etapas foram encontradas diversas dificuldades, visto à complexidade e o tamanho do projeto, foi necessário um grande comprometimento, empenho e determinação. O tempo necessário para a implementação e integração dos componentes também foi um grande desafio, pois muitas etapas demoraram um tempo maior do que o esperado, principalmente no início, onde foi preciso instalar e aprender a utilizar o *framework ROS*, que possui uma curva de aprendizado considerada lenta.

Além do *ROS*, também foi preciso adaptar todo o projeto ao ambiente disponível que, devido ao espaço limitado, houve dificuldades para gerar um mapa compatível com as características do projeto e com diversidade de rotas. Com isso foi preciso garantir uma boa precisão de locomoção, pois não existe espaço para margens de erro no mapa, além disso também foi preciso considerar a limitação do sensor de proximidade utilizado, tornando necessária uma maior quantidade de testes para se obter um processamento apurado para uma correta obtenção e interpretação dos dados.

Outros dois pontos de grande desafio, foi a implementação da malha de controle fechada usando controladores P de baixo nível e exibir o vídeo em tempo real na página *Web*. Podemos considerar o controlador das partes mais complexas do trabalho, é preciso que sua lógica e parâmetros sejam ajustados em perfeita sincronia para que o robô execute a tarefa desejada. Para inserir o vídeo na página *Web* em tempo real foi preciso implementar uma solução bastante criativa (uma fila circular de imagens) para evitar que o vídeo ficasse piscando, um efeito conhecido como *flickering*.

Após todo o desenvolvimento e considerando os resultados apresentados, foi possível inferir que obtemos um sistema robótico de manipulação e transporte autônomo eficiente que é capaz de transportar cargas no contexto da indústria 4.0. Em todos os testes o robô executou com sucesso as tarefas determinadas, dentro dos limites estabelecidos e com um tempo de execução aceitável.

Visto que o projeto é bastante amplo, é possível sugerir aprimoramentos em diversas frentes, como apresentado a seguir:

- Página Web: melhorias na usabilidade ou incluir novas funcionalidades, como por exemplo permitir ao usuário interromper a execução de uma tarefa.
- Processamento de imagem: melhorias no algoritmo de detecção do robô para evitar um falso reconhecimento ou para detectar obstáculos e pontos de referência no ambiente.
- Controlador: implementação de um controlador que consiga um melhor desempenho, reduzindo o tempo de execução de uma determinada tarefa.
- Planejador de rota e representação do ambiente: com o uso de uma representação de ambiente mais precisa (mapa métrico) pode-se desenvolver um planejador de rota para acessar outros locais do ambiente.
- *SLAM*: implementar o *SLAM* (Localização e Mapeamento Simultâneos), possibilitando que o robô realize o mapeamento do ambiente enquanto navega.
- Planejador de tarefas: implementar um algoritmo para alocação de tarefas de transporte usando técnicas de Inteligência Artificial.
- Múltiplos robôs: expandir o projeto proposto para um sistema de transporte que utilize vários robôs autônomos operando simultaneamente.
- Caracterização: detalhar os limites e latências apresentadas pelo sistema robótico.
- Métrica: utilizar o tempo como métrica de melhor desempenho, pois no contexto da indústria o tempo é mais importante do que a distância.
- Código QR: utilizar o código QR para obter o destino.
- Estudo sobre eficiência: análise da eficiência energética entre diferentes algoritmos de planejamento de trajetória.
- Simplificar hardware: estudo sobre a simplificação do hardware, como por exemplo a centralização de todos os algoritmos no *Raspberry*.

Referências

- ALIEXPRESS. *Plataforma roda omnidirecional*. Acesso em: 05 setembro 2020. Disponível em: <<https://pt.aliexpress.com/item/32862938007.html>>. Citado na página 19.
- ÅSTRÖM, K.; HÄGGLUND, T. *Advanced PID Control*. [S.l.]: ISA - The Instrumentation, Systems and Automation Society, 2006. ISBN 978-1-55617-942-6. Citado na página 28.
- Betke, M.; Gurvits, L. Mobile robot localization using landmarks. *IEEE Transactions on Robotics and Automation*, v. 13, n. 2, p. 251–263, 1997. Citado na página 25.
- BRETTEL N. FRIEDERICHSEN, M. K.; ROSENBERG, N. How virtualization, decentralization and network building change the manufacturing landscape: An industry 4.0 perspective. *International Journal of Mechanical*, v. 8, n. 1, p. 37–44, 2014. Citado na página 10.
- BUZZI, L. H. *Projeto e concepção de uma plataforma robótica móvel integrada com o ROS*. 78 p. Dissertação (Engenheiro de Controle e Automação) — Universidade Federal de Santa Catarina Centro de Blumenau Departamento de Engenharia de Controle e Automação e Computação, 2019. Citado na página 31.
- CARDOSO, C. *Indústria 4.0 na teoria*. 2018. Acesso em: 25 maio 2019. Disponível em: <<http://www.kitemes.com.br/2018/02/26/industria-4-0-na-teoria/>>. Citado 2 vezes nas páginas 9 e 10.
- CERIANI, S.; MIGLIAVACCA, M. *Middleware in robotics*. [S.l.]. Citado na página 29.
- COELHO, F. A. A. *PROJETO E IMPLEMENTAÇÃO DE UM ROBÔ DO TIPO PÊNDULO INVERTIDO MÓVEL*. 49 p. Dissertação (Bacharel em Engenharia de Computação) — Universidade São Francisco, 2012. Citado na página 27.
- CORMEN, T. H. et al. *Introduction to Algorithms, Third Edition*. 3rd. ed. [S.l.]: The MIT Press, 2009. ISBN 0262033844. Citado na página 24.
- DJOJO, M.; KARYONO, K. Computational load analysis of dijkstra, a*, and floyd-warshall algorithms in mesh network. In: . [S.l.: s.n.], 2013. p. 104–108. ISBN 978-1-4799-1208-7. Citado na página 25.
- DRUCKER, P. Além da revolução da informação. *HSM Management, São Paulo*, v. 4, n. 18, 2000. Citado na página 10.
- FRANCIS, M. M. B. A. *Flocking and Rendezvous in Distributed Robotics*. [S.l.]: Springer International Publishing, 2016. ISBN 978-3-319-24729-8. Citado na página 17.
- G., R. Integrated feature extraction for image retrieval. 11 2017. Citado na página 53.
- JAZDI, N. Cyber physical systems in the context of industry 4.0. *2014 IEEE International Conference on Automation, Quality and Testing, Robotics*, v. 1, n. 1, p. 1–3, 2014. Disponível em: <<https://ieeexplore.ieee.org/document/6857843>>. Citado 2 vezes nas páginas 14 e 16.
- JUNIOR, F. E. F. *Uma introdução ao Robot Operating System (ROS)*. 2016. Acesso em: 16 junho 2020. Disponível em: <<https://www.embarcados.com.br/uma-introducao-ao-robot-operating-system-ros/>>. Citado na página 29.

KAGERMANN, H.; WAHLSTER, W.; HELBIG, J. *Recommendations for Implementing the Strategic Initiative industry 4.0 - final report of the working group industry 4.0, Office of the Industry-Science Research Alliance Secretariat of the Platform Industrie*. [S.l.: s.n.], 2013. Citado na página 9.

LATOMBE, J. *Robot Motion Planning: Edition en anglais*. Springer, 1991. (The Springer International Series in Engineering and Computer Science). ISBN 9780792391296. Disponível em: <<https://books.google.com.br/books?id=Mbo\p4-46-cC>>. Citado na página 23.

MACDOUGALL, W. *Industrie 4.0- Smart Manufacturing for the Future*. Alemanha, 2014. 40 p. ISSN 0042-8582. ISBN 9783981583328. Disponível em: <http://www.inovasyon.org/pdf/GTAI.industrie4.0_smart.manufact.for.future.July.2014.pdf>. Citado na página 16.

MATHWORKS. *Exchange Data with ROS Publishers and Subscribers*. 2020. Acesso em: 16 junho 2020. Disponível em: <<https://www.mathworks.com/help/ros/ug/exchange-data-with-ros-publishers-and-subscribers.html>>. Citado na página 30.

MIRANDA, R. C. de. *Sistema de Navegação para Robôs Móveis Baseado em PID-Fuzzy*. 84 p. Dissertação (Bacharel em Engenharia da Computação) — Universidade Federal do Amazonas, 2017. Citado na página 28.

NISE, N. S. *Engenharia de Sistemas de Controle*. 6th. ed. Rio de Janeiro: LTC Editora, 2012. 760 p. ISBN 8521621353. Citado na página 27.

OGATA, K. *Engenharia de Controle Moderno*. 5th. ed. [S.l.]: Pearson, 2015. Citado 3 vezes nas páginas 27, 28 e 29.

OLIVEIRA, J. R. *Um sistema integrado para navegação autônoma de robôs móveis*. 102 p. Dissertação (Mestrado em Ciências - Ciência de Computação e Matemática Computacional) — Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 2010. Citado 2 vezes nas páginas 22 e 24.

RUSSELL, S.; NORVIG, P. *Artificial Intelligence, A Modern Approach. Second Edition*. [S.l.: s.n.], 2003. Citado na página 25.

RUZZON, T. A. L. *Navegação de um robô hexápode usando sistema de visão externa*. 119 p. Dissertação (Engenharia de Computação) — Instituto de Ciência e Tecnologia da Universidade Federal de São Paulo, 2019. Citado na página 52.

SIEGWART, R.; NOURBAKHSH, I. R. *Introduction to Autonomous Mobile Robots*. USA: Bradford Company, 2004. ISBN 026219502X. Citado 6 vezes nas páginas 16, 17, 18, 20, 21 e 24.

Smith, C. M. et al. Feature-based concurrent mapping and localization for auvs. In: *Oceans '97. MTS/IEEE Conference Proceedings*. [S.l.: s.n.], 1997. v. 2, p. 896–901 vol.2. Citado na página 25.

THRUN, S. *Robot Mapping: A Survey*. Feb. 2002. CMU-CS-02-111. Disponível em: <http://www-2.cs.cmu.edu/afs/cs.cmu.edu/user/thrun/public_html>. Citado 4 vezes nas páginas 22, 23, 25 e 26.

WILSON, J. M. Henry ford vs. assembly line balancing. *International Journal of Production Research*, Taylor Francis, v. 52, n. 3, p. 757–765, 2014. Citado na página 9.

YOUTUBE. *Automated Guided Vehicle Weasel®, E-Commerce, Supply Chain, Hermes Fulfilment GmbH*. Acesso em: 06 agosto 2020. Disponível em: <<https://www.youtube.com/watch?v=WIIS3vNSuQ4>>. Citado na página 12.

YOUTUBE. *Autonomous Mobile Robot (AMR) for Industry 4.0 Smart Manufacturing*. Acesso em: 06 agosto 2020. Disponível em: <<https://www.youtube.com/watch?v=FDlkb1A5iE>>. Citado na página 11.

YOUTUBE. *Autonomous Mobile Robots (AMRs) in Action*. Acesso em: 06 agosto 2020. Disponível em: <<https://www.youtube.com/watch?v=M0fL5Q6rGws>>. Citado na página 12.

YOUTUBE. *Autonomous transportation with mobile robot KMR iiwa*. Acesso em: 06 agosto 2020. Disponível em: <<https://www.youtube.com/watch?v=9WNE3JAcO6U>>. Citado 2 vezes nas páginas 12 e 13.

YOUTUBE. *BMW Smart Transport Robots*. Acesso em: 06 agosto 2020. Disponível em: <https://www.youtube.com/watch?v=Wan_M1PwcOQ>. Citado na página 12.

YOUTUBE. *How to Setup MiR Autonomous Robots*. Acesso em: 06 agosto 2020. Disponível em: <<https://www.youtube.com/watch?v=TRJCkgUKSX8>>. Citado na página 12.

YOUTUBE. *An IIoT based Smart Robotic Warehouse Management System for Industry 4.0*. Acesso em: 06 agosto 2020. Disponível em: <<https://www.youtube.com/watch?v=1-o6tBhEHOU>>. Citado na página 11.

YOUTUBE. *Inside Alibaba's smart warehouse staffed by robots*. Acesso em: 06 agosto 2020. Disponível em: <<https://www.youtube.com/watch?v=FBI4Y55V2Z4>>. Citado na página 12.

YOUTUBE. *MiR500 Autonomous Pallet Transportation Demonstration*. Acesso em: 06 agosto 2020. Disponível em: <<https://www.youtube.com/watch?v=JuRYvOLLd5A>>. Citado na página 12.

YOUTUBE. *Nipper*. Acesso em: 06 agosto 2020. Disponível em: <<https://www.youtube.com/watch?v=uvsvrf-eB4g>>. Citado na página 12.

YOUTUBE. *Transport robot loves color*. Acesso em: 06 agosto 2020. Disponível em: <<https://www.youtube.com/watch?v=SvtF31waMnY>>. Citado na página 12.

Apêndices

APÊNDICE A – Codificação do projeto

Arquivos disponíveis em https://github.com/refulk/ROS_robot. Referente aos códigos desenvolvidos no Servidor Local, Servidor Web e *Arduino*.