



Tiago Alexander Leal Ruzzon

## Navegação de um robô hexápode usando sistema de visão externa

São José dos Campos - SP

2019

Tiago Alexander Leal Ruzzon

## **Navegação de um robô hexápode usando sistema de visão externa**

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

Universidade Federal de São Paulo – UNIFESP  
Instituto de Ciência de Tecnologia  
Engenharia da Computação

Orientador: Prof. Dr. Sérgio Ronaldo Barros dos Santos

São José dos Campos - SP  
Julho de 2019

Tiago Alexander Leal Ruzzon

## Navegação de um robô hexápode usando sistema de visão externa

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

Trabalho aprovado em 03 de Julho de 2019:

  
Prof. Dr. Sérgio Ronaldo Barros dos  
Santos  
Orientador

  
Professor  
Dr. Lauro Paulo da Silva Neto

  
Professor  
Dr. Alvaro Luiz Fazenda

São José dos Campos - SP  
Julho de 2019

*Este trabalho é dedicado a meus pais, Cristiane e José Osmilton, à minha avó, Maria Dolores, à minha namorada, Letícia, e a todas as pessoas que estiveram envolvidas não só no desenvolvimento deste trabalho, mas sim em toda minha evolução acadêmica e profissional.*

# Agradecimentos

Gostaria de agradecer a meus pais, Cristiane Leal Ruzzon e José Osmilton Ruzzon, por terem me motivado e me possibilitado a buscar um caminho, tão distante, para eu poder realizar o que sempre sonhei para enriquecimento pessoal e profissional.

À minha avó, Maria Dolores Leal, que me prestou uma ajuda inimaginável nessa jornada.

À minha namorada, Letícia Migoto Ribeiro, que sempre foi compreensiva com a minha necessidade de dispor de praticamente todo meu tempo livre para a realização do projeto, e além disso prestou todo apoio que poderia.

À minha família (especialmente meu tio Eduardo Leal) e amigos, que sempre estiveram próximos e fornecendo todo apoio possível.

Ao meu orientador, Prof. Dr. Sérgio Ronaldo Barros dos Santos, por ter apresentado essa proposta de trabalho e me acompanhado durante todo o trajeto de implementação, sem hesitar em prestar toda ajuda e também dar as devidas broncas necessárias.

# Resumo

Intitulado de Navegação de um robô hexápode usando sistema de visão externa, este projeto consiste de um robô com perna equipado com 18 servomotores, controlados por uma placa específica que recebe os sinais de movimentação de um microcontrolador de 8-bit *Atmega 2560* embarcado. A localização do robô é estimada através de uma câmera externa instalada no ambiente de experimentação, que detecta a movimentação do robô numa visão superior. O objetivo do robô é realizar o seguimento de caminhos especificados através de *waypoints* 2D definidos no ambiente. Para isso, é utilizado de *Python* e *OpenCV* para realizar o processamento de imagem necessário para se obter sua localização. A comunicação entre o PC (estaçao de solo) no qual a câmera está conectada e o sistema embarcado do robô se dá através da tecnologia *XBee*. Foram implementadas duas abordagens de comunicação, sendo uma de duas vias, na qual o robô solicita informações à estação de solo, e outra na qual apenas o PC envia informações ao robô. Foram definidos dois casos de testes, com três e cinco *waypoints*. Em todos os resultados, o robô conseguiu atingir as coordenadas e o caminho especificado com sucesso.

**Palavras-chaves:** robótica. processamento de imagem. hexápode. controle de servomotores.

# Abstract

*Entitled Navigation of a hexapode robot using external vision system, this project consists of a legged robot equipped with 18 servomotors, controlled by a specific board that receives the movement signals of an embedded 8-bit Atmega 2560 microcontroller. The identification of the location of the robot is estimated by an external camera installed in the experimentation room, that detects the movement of the robot in a superior vision. The objective of the robot is to follow the paths specified by 2D waypoints defined in the environment. For this, Python and OpenCV are used to perform the image processing necessary to obtain its location. The communication between the PC (ground station) in which the camera is connected and the system embedded in the robot is given by XBee technology. Two communication approaches were implemented: a two-way, in which the robot requests information from the ground station, and other one in which only the PC sends information to the robot. Two test cases were defined, with three and five waypoints. In all results, the robot was able to reach the specified coordinates and path successfully.*

**Key-words:** *robotics. image processing. hexapod. servomotors control.*

# Lista de ilustrações

Figura 1 – Classificação de robôs móveis de acordo com sua anatomia . . . . .	15
Figura 2 – <i>Walking Tractor Timberjack</i> . . . . .	17
Figura 3 – <i>SILO 6</i> . . . . .	17
Figura 4 – Robô projetado por Leonardo da Vinci . . . . .	18
Figura 5 – <i>Phony Pony</i> . . . . .	19
Figura 6 – OSU <i>Hexapod</i> . . . . .	20
Figura 7 – <i>TITAN VIII</i> . . . . .	20
Figura 8 – <i>Sutherland Hexapod</i> . . . . .	21
Figura 9 – <i>Sony Aibo 2018</i> . . . . .	21
Figura 10 – Controle de servomotor via PWM . . . . .	25
Figura 11 – Componentes de um Servomotor . . . . .	26
Figura 12 – Funcionamento de um sensor ultrassônico . . . . .	27
Figura 13 – Exemplo de robô hexápode do tipo <i>king spider</i> . . . . .	28
Figura 14 – Exemplo de robô hexápode <i>Lego Mindstorms NXT 2.0 Scorpion</i> . . . . .	28
Figura 15 – Estrutura de patas inspiradas em insetos . . . . .	29
Figura 16 – <i>Tripod Gait</i> . . . . .	31
Figura 17 – Resumo dos Hexápode <i>Gaits</i> . . . . .	32
Figura 18 – Destaque aos parâmetros dos <i>pixels</i> de uma imagem . . . . .	32
Figura 19 – Exemplo de aplicação de segmentação binária . . . . .	33
Figura 20 – Demonstração gráfica dos parâmetros do padrão HSV . . . . .	34
Figura 21 – Exemplo de uso de <i>waypoints</i> no <i>Google Maps</i> . . . . .	35
Figura 22 – Módulo <i>XBee</i> . . . . .	37
Figura 23 – Robô Desenvolvido . . . . .	38
Figura 24 – Robô com Acabamento em Madeira . . . . .	39
Figura 25 – Robô Hexápode . . . . .	40
Figura 26 – Diagrama de bloco para seguimento de caminhos . . . . .	41
Figura 27 – Diagrama dos componentes do sistema . . . . .	42
Figura 28 – <i>ToRobot 32-Channel Servo Controller</i> . . . . .	44
Figura 29 – Servomotor DS3218MG . . . . .	44
Figura 30 – Reconhecimento das peças . . . . .	45
Figura 31 – Segmentos da Perna . . . . .	45
Figura 32 – Suportes dos servomotores . . . . .	46
Figura 33 – Encaixe do servomotor na tibia . . . . .	47
Figura 34 – Finalização da montagem estrutural . . . . .	47

Figura 35 – <i>Arduino Mega</i> . . . . .	48
Figura 36 – Bateria <i>Li-PO Zippy 20C Series 5000 mAh 7.4 V</i> . . . . .	49
Figura 37 – <i>Shield XBee para Arduino e XBee Explorer</i> . . . . .	49
Figura 38 – Sensor de Distância Ultrassônico <i>HC-SR04</i> . . . . .	50
Figura 39 – Esquemático das conexões . . . . .	51
Figura 40 – Hexápode concluído . . . . .	52
Figura 41 – <i>Ultimate WYSIWG Motor Control V6.0.1.0</i> . . . . .	53
Figura 42 – Câmera instalada no teto da sala . . . . .	56
Figura 43 – Visão superior do robô . . . . .	57
Figura 44 – Ferramenta de seleção de cores do GIMP . . . . .	58
Figura 45 – <i>XCTU</i> . . . . .	60
Figura 46 – Representação gráfica da visão obtida pelo algoritmo de guiagem . . . . .	62
Figura 47 – Fluxograma do algoritmo de locomoção . . . . .	64
 Figura 48 – Trajetória de $A^g$ . . . . .	68
Figura 49 – Trajetória de $B^g$ . . . . .	69
Figura 50 – <i>Screenshots</i> do estudo de caso 1A (parte 1/2) . . . . .	70
Figura 51 – <i>Screenshots</i> do estudo de caso 1A (parte 2/2) . . . . .	71
Figura 52 – Trajetória executada pelo robô no estudo de caso 1A . . . . .	72
Figura 53 – Velocidade de translação para o estudo de caso 1A . . . . .	73
Figura 54 – Velocidade angular para o estudo de caso 1A . . . . .	73
Figura 55 – <i>Screenshots</i> do estudo de caso 2A (parte 1/2) . . . . .	75
Figura 56 – <i>Screenshots</i> do estudo de caso 2A (parte 2/2) . . . . .	76
Figura 57 – Trajetória executada pelo robô no estudo de caso 2A . . . . .	77
Figura 58 – Velocidade de translação para o estudo de caso 2A . . . . .	78
Figura 59 – Velocidade angular para o estudo de caso 2A . . . . .	79
Figura 60 – <i>Screenshots</i> do estudo de caso 1B (parte 1/2) . . . . .	80
Figura 61 – <i>Screenshots</i> do estudo de caso 1B (parte 2/2) . . . . .	81
Figura 62 – Trajetória executada pelo robô no estudo de caso 1B . . . . .	82
Figura 63 – Velocidade de translação para o estudo de caso 1B . . . . .	83
Figura 64 – Velocidade angular para o estudo de caso 1B . . . . .	83
Figura 65 – <i>Screenshots</i> do estudo de caso 2B (parte 1/2) . . . . .	85
Figura 66 – <i>Screenshots</i> do estudo de caso 2B (parte 2/2) . . . . .	86
Figura 67 – Trajetória executada pelo robô no estudo de caso 2B . . . . .	87
Figura 68 – Velocidade de translação para o estudo de caso 2B . . . . .	88
Figura 69 – Velocidade angular para o estudo de caso 2B . . . . .	88

# Lista de tabelas

Tabela 1 – Canais do circuito <i>ToRobot</i> conectados aos servomotores . . . . .	50
Tabela 2 – Conexões realizadas no <i>Arduino</i> embarcado . . . . .	51
Tabela 3 – Action groups guardados na memória do circuito <i>ToRobot</i> . . . . .	53
Tabela 4 – Formato de instruções para a placa <i>ToRobot</i> . . . . .	55
Tabela 5 – Referencial de cores no padrão HSV para identificação do robô . . . . .	58
Tabela 6 – Mensagens padrão de comunicação entre estação de solo e robô . . . . .	65
Tabela 7 – Formato da planilha relatório . . . . .	66
Tabela 8 – Tabela de erros nas chegadas para o estudo de caso 1A . . . . .	72
Tabela 9 – Tabela de erros nas chegadas para o estudo de caso 2A . . . . .	78
Tabela 10 – Tabela de erros nas chegadas para o estudo de caso 1B . . . . .	82
Tabela 11 – Tabela de erros nas chegadas para o estudo de caso 2B . . . . .	87
Tabela 12 – Referencial <i>action group</i> 5 (dar um passo a frente) . . . . .	98
Tabela 13 – Referencial <i>action group</i> 6 (virar a direita) . . . . .	98
Tabela 14 – Referencial <i>action group</i> 7 (virar a esquerda) . . . . .	98
Tabela 15 – Custo de montagem do sistema . . . . .	99

# Lista de abreviaturas e siglas

IDE	<i>Integrated Development Environment</i> , Ambiente de Desenvolvimento Integrado
ITA	Instituto Tecnológico de Aeronáutica
GPS	<i>Global Positioning System</i> , Sistema de Posicionamento Global
USB	<i>Universal Serial Bus</i>
A.G	<i>Action group</i>
HSV	<i>hue</i> (matiz), <i>saturation</i> (saturação) e <i>value</i> (valor)
RGB	<i>red</i> (vermelho), <i>green</i> (verde) e <i>blue</i> (azul)
PWM	<i>Pulse Width Modulation</i> , Modulação de Largura de Pulso
LTE	<i>Long Term Evolution</i> , Evolução de Longo Prazo
FPGA	<i>Field Programmable Gate Array</i> , Arranjo de Portas Programáveis em Campo
I2C	<i>Inter-Integrated Circuit</i>

# Lista de símbolos

$x^g$	Eixo $x$ do sistema de referência global
$y^g$	Eixo $y$ do sistema de referência global
$z^g$	Eixo $z$ do sistema de referência global
$A^g$	Matriz de <i>waypoints</i> para os testes A
$B^g$	Matriz de <i>waypoints</i> para os testes B
$E_{abs}$	Erro, em centímetros, calculado do módulo da diferença entre a coordenada do <i>waypoint</i> e a coordenada que o robô atingiu
$E_{rel}$	Erro percentual calculado do módulo da diferença entre a coordenada do <i>waypoint</i> e a coordenada que o robô atingiu

# Sumário

<b>1</b>	<b>Introdução</b>	<b>14</b>
1.1	Motivação	14
1.2	Revisão Bibliográfica	15
1.2.1	Breve Histórico de Robôs com Pernas	18
1.3	Objetivos	22
1.4	Estruturação do Trabalho	22
<b>2</b>	<b>Fundamentação Teórica</b>	<b>24</b>
2.1	Componentes Eletrônicos e Mecânicos	24
2.1.1	Servomotores	24
2.1.2	Sensores ultrassônicos	26
2.2	Mobilidade do Robô Hexápode	27
2.2.1	Estabilidade	29
2.2.2	Cinemática	29
2.2.3	Dinâmica	30
2.2.4	<i>Gait</i>	30
2.3	Processamento de Imagem	32
2.3.1	Segmentação Binária	33
2.3.2	Sistemas de Cores HSV	34
2.3.3	<i>OpenCV</i>	35
2.4	Navegação por <i>Waypoints</i>	35
2.5	Comunicação Sem Fios	36
2.5.1	<i>XBee</i>	36
<b>3</b>	<b>Trabalhos Relacionados</b>	<b>38</b>
<b>4</b>	<b>Desenvolvimento</b>	<b>41</b>
4.1	Visão Geral do Sistema Proposto	41
4.2	Montagem do Robô Hexápode	43
4.3	Controle da Locomoção	52
4.4	Estimativa de Localização	55
4.4.1	Sistema de Visão Externa	55
4.4.2	Algoritmo de Processamento de Imagem	57
4.5	Comunicações sem fio	59
4.6	Sistema de Guiagem	61
4.6.1	Sistema de Comunicação: Modos para troca de mensagens	64

4.6.2 Aquisição dos Resultados . . . . .	65
<b>5 Resultados . . . . .</b>	<b>67</b>
5.1 Definição dos Estudos de Caso . . . . .	67
5.2 Estudo de Caso 1A . . . . .	69
5.3 Estudo de Caso 2A . . . . .	74
5.4 Estudo de Caso 1B . . . . .	79
5.5 Estudo de Caso 2B . . . . .	84
<b>6 Conclusão . . . . .</b>	<b>90</b>
6.1 Trabalhos Futuros . . . . .	90
<b>Referências . . . . .</b>	<b>92</b>
<b>Apêndices . . . . .</b>	<b>95</b>
<b>APÊNDICE A Instruções para inicialização do sistema . . . . .</b>	<b>96</b>
A.1 Partida do robô . . . . .	96
A.2 Estação de solo . . . . .	96
<b>APÊNDICE B Referencial de movimentação dos servomotores . . . . .</b>	<b>98</b>
<b>APÊNDICE C Custos do sistema . . . . .</b>	<b>99</b>
<b>Anexos . . . . .</b>	<b>100</b>
<b>ANEXO A Codificação embarcada no robô . . . . .</b>	<b>101</b>
A.1 Abordagem 1: comunicação em duas vias . . . . .	101
A.2 Abordagem 2: comunicação em via única . . . . .	104
<b>ANEXO B Codificação da estação de solo . . . . .</b>	<b>107</b>
B.1 Abordagem 1: comunicação em duas vias . . . . .	107
B.2 Abordagem 2: comunicação em via única . . . . .	113

# 1 Introdução

Desde o início da humanidade, o homem vem criando ferramentas para o auxiliar e substituir em suas tarefas diárias. Isso pode ser evidenciado desde o mais antigo uso de arco e flecha para caça e obtenção de alimentos até os dias de hoje, nos quais se pode pedir comida sem sair de casa através de celulares, por exemplo.

Uma das vertentes com avanços mais representativos é a de robótica. Antigamente, essa tecnologia estava relacionada basicamente a máquinas de linhas de montagens industriais, realizando a automatização de repetitivos trabalhos humanos e até possibilitando trabalhos que não seriam possíveis, como movimentação de grande quantidade de massa e trabalho em altura. Entretanto, com o passar do tempo a robótica passou a participar de outros tipos de mercado, como o do entretenimento, segurança e logística, por exemplo.

O conceito do termo robô ainda não é um consenso na bibliografia. [Rabelo \(2015\)](#) define robôs como máquinas por meio de sua constituição através de sistemas de controle, de percepção (sensoriamento), de ação mecânica (manipulação ou locomoção) e de inteligência (processamento, decisões ou troca de informações). Já [Todd \(1985\)](#), define um robô como uma tecnologia (que deve ser fabricada, e não produzida biologicamente) que é capaz de mover objetos ou de se mover. Além disso, deve ser capaz de sustentar ações sem intervenções de agentes externos (o que exclui os automóveis regulares dessa categoria), mudando seu comportamento de acordo com respostas obtidas a sensores, por exemplo.

Na robótica móvel aplicada a este trabalho o objeto de estudo são os robôs que são capazes de se mover. O crescimento desse tipo de tecnologia só foi possível graças aos avanços na produção de microcontroladores, que passaram a processar dados com uma capacidade muito maior e ocupando menos espaço, o que reduziu também o consumo de energia e a dissipação de calor. Dessa maneira, é possível embarcar em pequenos robôs todo o *hardware* necessário para o processamento do mesmo, incluindo condições observadas em sensores e os comandos para os elementos mecânicos de movimentação. A redução no consumo energético também possibilitou a utilização de baterias, visto que esses dispositivos já não são mais grandes e pesadas máquinas e podem se locomover pelos mais diversos terrenos sem necessidade de alimentação cabeadas.

## 1.1 Motivação

Para este Trabalho de Conclusão do curso de Engenharia da Computação, foi escolhido realizar o desenvolvimento técnico de um robô móvel com morfologia hexápode (com seis pernas). Dessa maneira, é possível integralizar praticamente todo o conhecimento adquirido ao longo do curso.

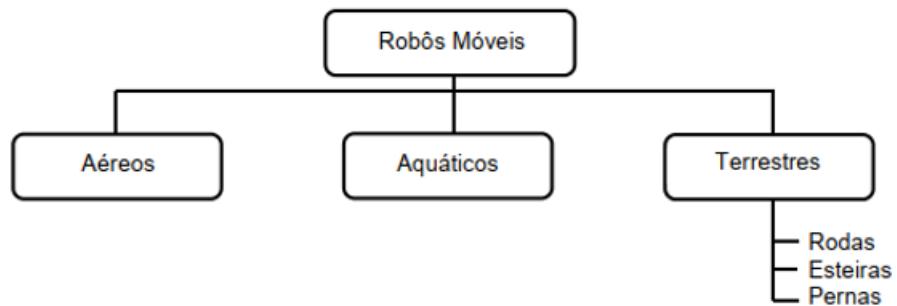
mento adquirido durante a realização do curso, visto que o projeto abrange conceitos de desenvolvimento de sistemas embarcados, programação, controle, montagem do *hardware* e processamento de imagem.

## 1.2 Revisão Bibliográfica

Com o recorrente avanço da eletrônica e mecânica, atualmente é possível se obter microcontroladores e motores com alto desempenho e baixo custo, o que torna completamente viável a aquisição dos componentes para a montagem do projeto.

Existem diversas opções de dispositivos de locomoção para robótica móvel. Segundo [Rabelo \(2015\)](#), os robôs móveis podem ser classificados por sua anatomia de acordo com a [Figura 1](#).

Figura 1 – Classificação de robôs móveis de acordo com sua anatomia



Fonte: [Rabelo \(2015\)](#)

Sendo o escopo desse projeto robôs terrestres, existem as opções de se realizar a locomoção através de rodas, esteiras ou pernas.

Entre as opções de rodas ou pernas, o robô escolhido é do tipo móvel com pernas. Essa escolha foi definida por causa da dinâmica disponível na mobilidade desta classe. Diferentemente de rodas, as pernas permitem maior variedade de acesso a terrenos mais restritos, simulando-se a extremamente avançada capacidade de locomoção dos animais. [Todd \(1985\)](#) cita várias vantagens do uso pernas, como:

- Capacidade de atravessar obstáculos, subir e descer escadas.
- Possibilidade de locomoção em terrenos acidentados e áreas arenosas.
- Pequena área de impacto sobre o chão, pois pernas possuem finitos contatos com o solo.

- Em determinados instantes e dependendo da forma de movimentação utilizada, é possível que o robô movimentado por pernas não mantenha nenhum contato com o solo e mesmo assim mantenha sua estabilidade, ao se mover em alta velocidade.
- Pernas causam menos prejuízos, pois não é necessária a construção de estradas como é para uso de rodas.

Segundo [Chávez-Clemente \(2011\)](#) e [Rabelo \(2015\)](#), esse tipo de ferramenta possui ampla aplicabilidade. Dentre suas possíveis funções, destacam-se:

- Uso industrial para transporte de cargas pesadas ou delicadas.
- Uso militar para carregamento bélico, acesso a área isoladas, remoção de minas terrestres em campos minados e reconhecimento em ambientes de guerra, poupando a presença humana de diversas situações de risco.
- Salvamento e resgate em ambientes hostis.
- Atividades de vigilância.
- Atividades florestais.
- Realização de manutenção e inspeção em locais de difícil acesso.
- Exploração planetária.
- Entretenimento pessoal.
- Assistência em ambientes domésticos e profissionais, como tarefas relacionadas a limpeza.

A [Figura 2](#) apresenta um exemplo de produto real, já lançado no mercado, o *Walking Tractor Timberjack*, fabricado pela *John Deere*. Segundo *The Old Robots*, essa máquina foi desenvolvida para se caminhar em florestas com a melhor estabilidade e mínimo impacto no terreno. Ela se adapta automaticamente ao solo, se move através de seis pernas articuladas e possui distância do solo ajustável. Além disso, também possui um braço articulado que pode ser equipado com pinças e serras, por exemplo.

Figura 2 – *Walking Tractor Timberjack*Fonte: [www.theoldrobots.com/](http://www.theoldrobots.com/)

A [Figura 3](#) apresenta o robô *SILO 6*, cuja atividade principal é a remoção de minas terrestres em campos minados.

Figura 3 – *SILO 6*Fonte: [http://www.car.upm-csic.es/fsr/provisional/silo6/SILO6\\_WalkingRobot.htm](http://www.car.upm-csic.es/fsr/provisional/silo6/SILO6_WalkingRobot.htm)

Robôs bípedes, inspirados na biologia humana, apresentam muitas dificuldades de implementação com relação ao equilíbrio que é necessário para se sustentar com apenas duas pernas. Esse problema seria solucionado com robôs quadrúpedes, porém para esse projeto preferiu-se adicionar mais duas pernas, obtendo-se, assim, um protótipo hexápode.

Apesar de se acrescentar gasto de energia e complexidade, esses robôs, em semelhança com insetos em seus formatos biológicos, conseguem ainda se locomover caso percam alguma de suas pernas ou qualquer outra avaria aconteça, permitindo a adapta-

ção, apesar da redução de velocidade e estabilidade. Isso não seria possível no caso do robô com quatro ou menos pernas.

Outro desafio que é encontrado na implementação de robôs desta classe (ainda mais num caso hexápode) é que, devido à sua quantidade de motores, eles necessitam de uma grande quantidade de energia para a locomoção. Outros problemas também podem ser encontrados com relação à estabilidade, velocidade e conforto (no caso de ser utilizado para transportar humanos). [Oliveira \(2016\)](#)

### 1.2.1 Breve Histórico de Robôs com Pernas

[Machado e Silva \(2006\)](#) explica que o primeiro registro de ideias com o objetivo de construir um veículo com patas são de Leonardo da Vinci, no século XV. Ele projetou (e provavelmente construiu) o primeiro veículo de patas articuladas. O material embarcado nesse projeto foi madeira, couro, latão e bronze. Ele tinha a forma de uma armadura. Um modelo desse robô está disponível na [Figura 4](#).

Figura 4 – Robô projetado por Leonardo da Vinci



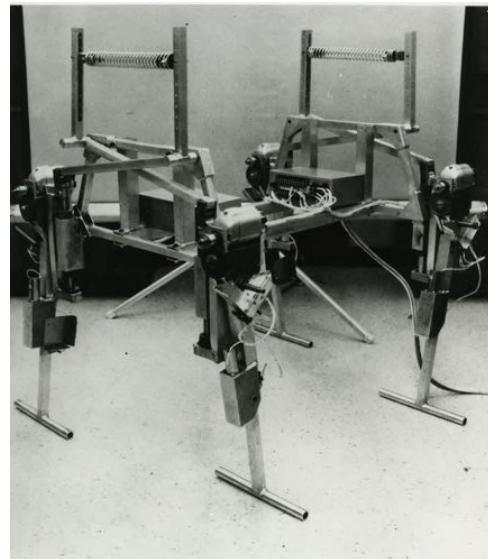
Fonte: [www.leonardorobotsociety.org](http://www.leonardorobotsociety.org)

Segundo [Todd \(1985\)](#), as pesquisas focadas em robôs com pernas ganharam força após a Segunda Guerra Mundial. Isso se deve ao fato de que esse conflito gerou um forte impulso para o desenvolvimento de tecnologias computacionais e eletrônicas. As criações dos primeiros robôs bem sucedidos ocorreram após o surgimento dos computadores modernos.

[Todd \(1985\)](#) apresenta que o primeiro robô com pernas e movimentação autônoma, controlado por computador e alimentado eletricamente, foi o *Phony Pony*, desenvolvido

em 1968 por *McGhee e Frank*. Ele era quadrúpede, com dois graus de liberdade por perna, e não possuía sua estrutura exatamente baseada num modelo biológico. Era movido por motores elétricos com redução realizada por engrenagens [Dantas (2014)]. O robô recebia alimentação de energia externamente via cabo. Além disso, ele apenas se movia em linha reta, não possuindo articulações para fazer curvas. A [Figura 5](#) apresenta uma fotografia do protótipo montado.

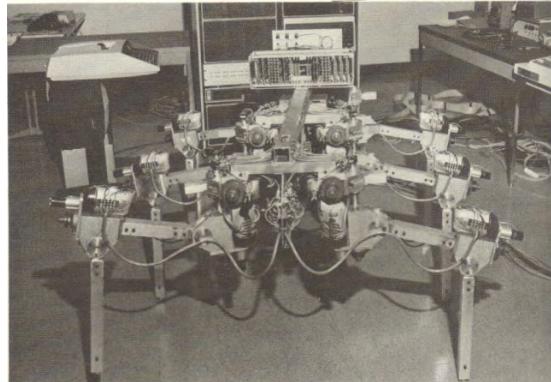
Figura 5 – *Phony Pony*



Fonte: [Todd \(1985\)](#)

Conforme [Dantas \(2014\)](#), em 1977 foi construído o hexápode OSU ([Figura 6](#)). Esse robô possui três graus de liberdade por perna, simulando a estrutura de um inseto. Foi utilizado em vários experimentos, como se locomover por diversos tipos de terrenos, subir ladeiras e ultrapassar obstáculos. Neste ano também foi construído o *Masha Hexapod*, que era muito semelhante ao OSU, porém sendo fruto de um trabalho russo, enquanto o primeiro é americano. [Demasi \(2012\)](#)

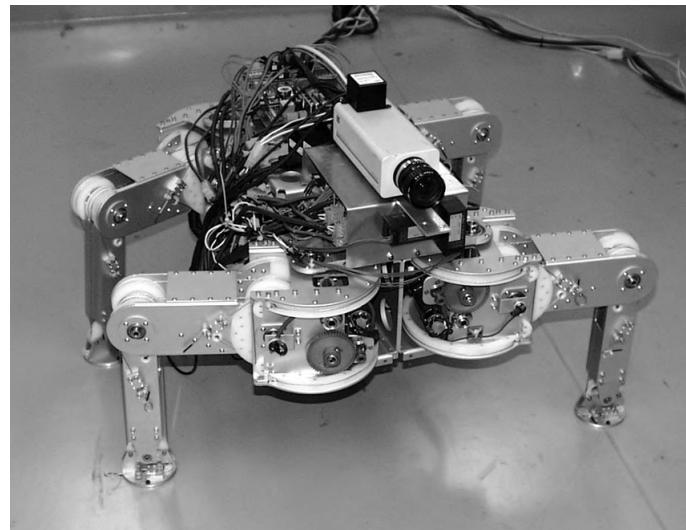
Figura 6 – OSU Hexapod



Fonte: [cyberneticzoo.com/walking-machines/](http://cyberneticzoo.com/walking-machines/)

Rabelo (2015) diz que, a partir dos anos 80, com o avanço das tecnologias de atuadores, sensores e computadores, o desenvolvimento de robôs com mais habilidades começou a ser realizado.

Na década de 1980, foi desenvolvido o robô quadrúpede *TITAN VIII* (Figura 7). Esse foi o primeiro robô desenhado para ser de fácil fabricação e custo acessível, de forma a ser comercializado principalmente para pesquisadores. O robô teve mais de 70 unidades vendidas no Japão. Dantas (2014)

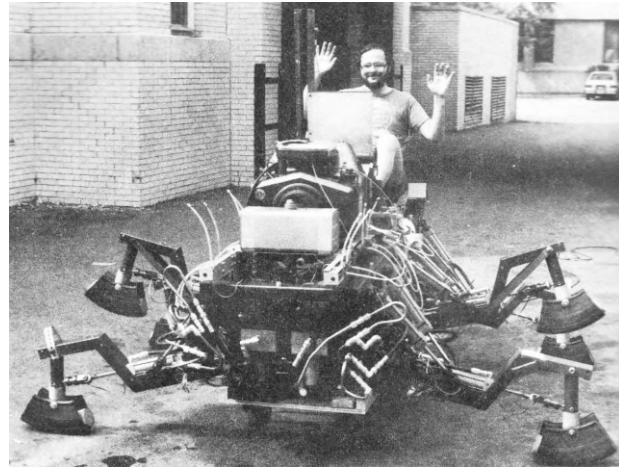
Figura 7 – *TITAN VIII*

Fonte: [www.researchgate.net/figure/4-legged-walking-robot-TITAN-VIII\\_fig5\\_224072213](http://www.researchgate.net/figure/4-legged-walking-robot-TITAN-VIII_fig5_224072213)

Segundo Demasi (2012), em 1983 foi construído por *Sutherland* o primeiro robô de morfologia hexápode com microcomputador embarcado. Esse robô, que está apresentado

na [Figura 8](#), tinha a capacidade de carregar uma pessoa.

Figura 8 – *Sutherland Hexapod*



Fonte: [cyberneticzoo.com/walking-machines/](http://cyberneticzoo.com/walking-machines/)

Um exemplo de projeto atual é o *Sony Aibo*, que se destaca no mercado de entretenimento. Trata-se de um robô quadrúpede que imita o comportamento de um cachorro doméstico. Ele foi lançado pela primeira vez em 1999, e, depois de mais de dez anos descontinuado, foi relançado numa nova versão em 2018 [[Sony \(2017\)](#)]. É equipado com um processador de quatro núcleos, 22 eixos de movimentação, duas câmeras e outros diversos sensores, como, por exemplo, capacitivos ao redor de sua estrutura para detecção de toque humano. Além disso, possui comunicação LTE e Wi-Fi. A [Figura 9](#) apresenta o robô em sua versão do ano de 2018.

Figura 9 – *Sony Aibo* 2018



Fonte: [Sony \(2017\)](#)

O [Capítulo 3](#) contém mais exemplos de projetos realizados na área, com relação direta à aplicabilidade desse trabalho.

### 1.3 Objetivos

Realizar a implementação de um robô hexápode com *software* embarcado controlado por *Arduino* cujos movimentos são definidos via reconhecimento de imagem por câmera fixada no teto do ambiente controlado. O robô deve ser capaz de atingir pontos coordenados previamente estabelecidos, denominados de *waypoints*.

O objetivo pode ser dividido em alguns pontos cruciais para o desenvolvimento:

- Montagem física dos componentes mecânicos do robô em si, incluindo o chassis, pernas, servomotores e as placas de controle.
- Programação dos motores para controle das pernas, possibilitando a execução de caminhada a frente e mudança de trajetória (curva) com o robô.
- Implementação dos algoritmos de processamento de imagem a serem executados pela estação de solo, que se baseiam em pontos predefinidos no robô para identificação da localização dele no espaço definido.
- Implementação da comunicação sem fios entre o robô e a estação de solo.
- A partir de todos os pontos anteriores, integralizar o sistema completo, de forma que o robô consiga percorrer os *waypoints* de forma autônoma em comunicação com a estação de solo.

### 1.4 Estruturação do Trabalho

O trabalho aqui apresentado está estruturado da seguinte forma:

- No capítulo 2, é apresentada a fundamentação teórica para o trabalho, na qual estão contidos os conceitos e informações necessárias para entendimento da implementação realizada.
- No capítulo 3, estão apresentados os trabalhos já realizados no meio acadêmico que estão relacionados a este projeto.
- No capítulo 4, está apresentado o desenvolvimento realizado para a solução de implementação proposta.
- No capítulo 5, estão contidos os resultados obtidos através dos testes realizados com o sistema desenvolvido.

- O capítulo 6 contém as conclusões obtidas e as propostas para aprimoramentos futuros ao projeto.

## 2 Fundamentação Teórica

Esse capítulo apresenta uma breve fundamentação teórica sobre as principais vertentes abordadas no projeto deste robô hexápode.

### 2.1 Componentes Eletrônicos e Mecânicos

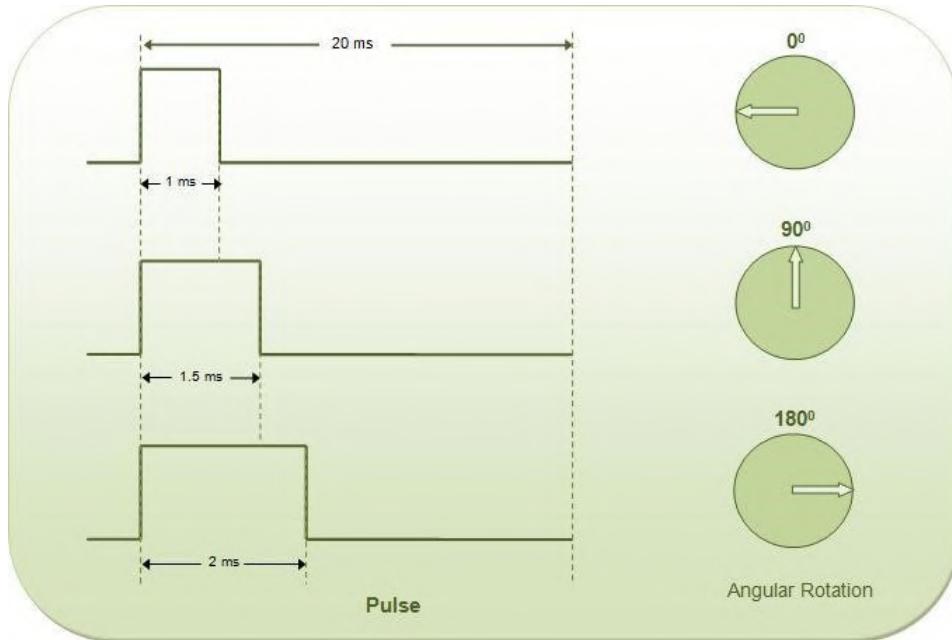
Sendo a eletrônica e mecânica as ciências básicas para a construção de um robô, é estritamente necessário incluir uma seção dedicada à definição dos componentes essenciais para o projeto.

#### 2.1.1 Servomotores

De acordo com [Santos \(2007\)](#), um servomotor é um dispositivo eletromecânico compacto que possui como característica posicionar seu eixo em determinada posição angular de acordo com o sinal elétrico recebido em sua entrada.

[Mota \(2017\)](#) também define um servomotor como um motor no qual é possível controlar sua posição angular através de um sinal PWM. A figura [Figura 10](#) demonstra como é feito o controle via PWM de um servomotor. Observa-se que o ângulo do eixo do servomotor é proporcional ao tempo em que o pulso de entrada está em nível alto dentro do tempo de ciclo (20 ms).

Figura 10 – Controle de servomotor via PWM



Fonte: [Mota \(2017\)](#)

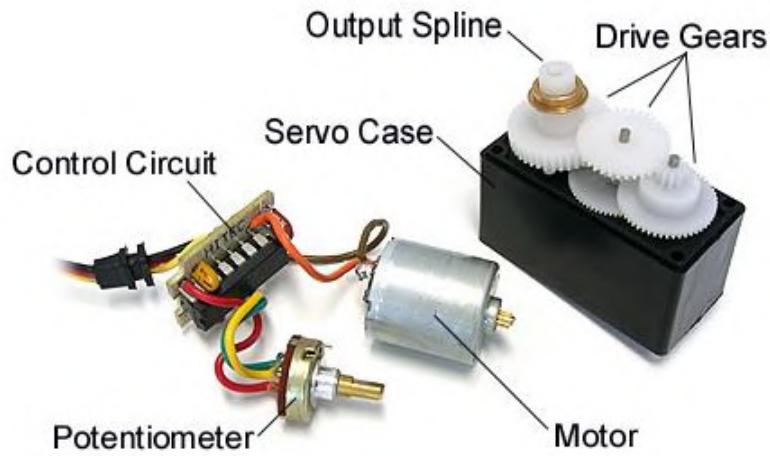
Segundo [Mota \(2017\)](#), um servomotor se diferencia de motores de corrente contínua ou motores de passo porque ele apenas gira seu eixo num ângulo 180°.

Um servomotor possui cinco partes principais [[Santos \(2007\)](#)]:

- **Circuito de controle:** de acordo com o sinal recebido pelo potenciômetro e com a entrada recebida pelo sistema, determina a movimentação do motor, seguindo o funcionamento explicado acima.
- **Potenciômetro:** monitora a posição do eixo de saída do servo.
- **Motor:** realiza o torque para movimentação das engrenagens e do eixo principal.
- **Engrenagens:** transfere o movimento do eixo do motor para as partes necessárias e movimentam o potenciômetro juntamente com o eixo. Realizam a redução necessária para aumento de torque.
- **Carcaça:** estrutura externa que serve como uma caixa para o servo.

Essas partes estão exemplificadas na [Figura 11](#), que demonstra uma foto de um servomotor desmontado.

Figura 11 – Componentes de um Servomotor

Fonte: [Santos \(2007\)](#)

### 2.1.2 Sensores ultrassônicos

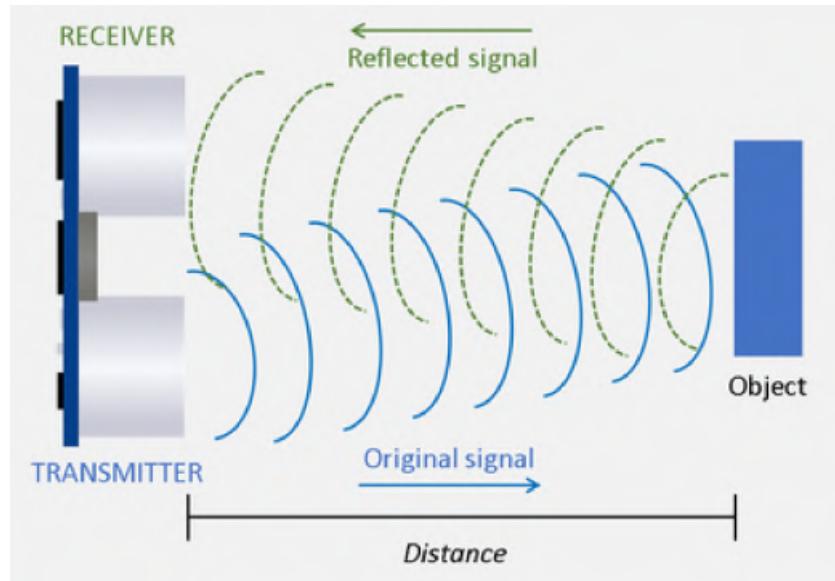
De acordo com [Vidal \(2018\)](#), o funcionamento de um sensor ultrassônico consiste de emitir sinais pelo sensor e ler o sinal de retorno (também conhecido como reflexo ou eco) do sinal emitido. Dessa maneira, é possível calcular a distância do objeto mais próximo à frente através da diferença entre tempo entre o envio e o retorno do sinal. Os sinais emitidos são geralmente com frequências mais altas do que as audíveis pelos seres humanos, de forma que seu funcionamento não seja perceptível.

Um sensor ultrassônico é composto por três partes principais:

- **Transmissor:** emite as ondas ultrassônicas.
- **Receptor:** identifica o sinal de retorno da onda enviada pelo transmissor.
- **Circuito de controle:** controla o transmissor e o receptor.

O funcionamento do sensor é demonstrado através do diagrama disponível na [Figura 12](#).

Figura 12 – Funcionamento de um sensor ultrassônico



Fonte: [osoyoo.com](http://osoyoo.com)

Além disso, esses sensores são geralmente conectados através de quatro pinos: um de alimentação, um de aterramento, *trigger* (no qual é comandado o envio dos sinais) e *echo* (retorna os pulsos com o tempo de duração entre o envio e recebimento) [Vidal (2018)].

## 2.2 Mobilidade do Robô Hexápode

Um robô de morfologia hexápode é um robô que tem sua locomoção baseada em seis pernas.

Atualmente, existem diversos tipos de robôs hexápodes no mercado, sendo possível encontrar kits prontos com valores bastante acessíveis. A Figura 13 apresenta um exemplo de um tipo de robô hexápode, chamado de *king spider*.

Figura 13 – Exemplo de robô hexápode do tipo *king spider*



Fonte: Medeiros (2014)

Na [Figura 14](#) está disponível uma fotografia do robô *Lego Mindstorms NXT 2.0 Scorpion*.

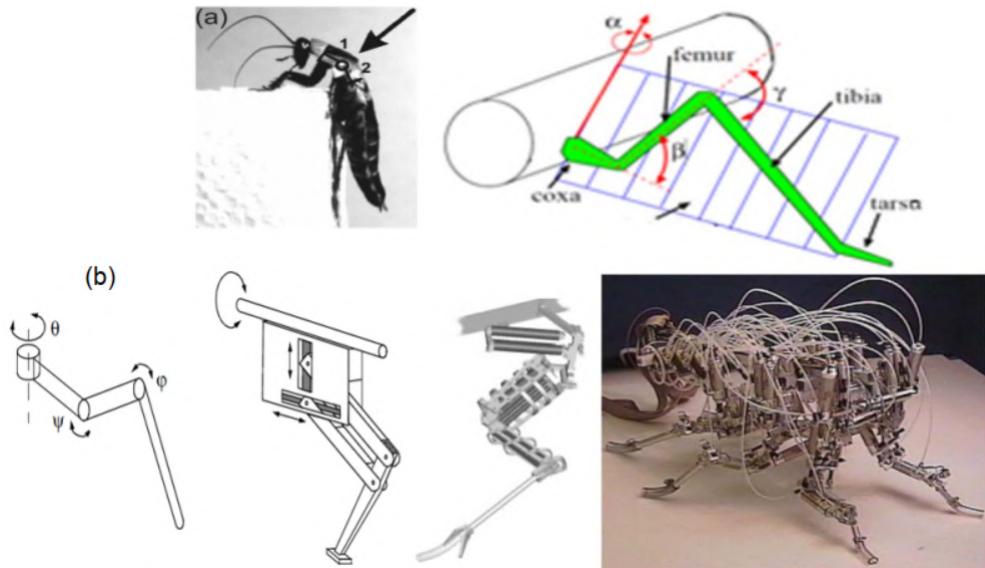
Figura 14 – Exemplo de robô hexápode *Lego Mindstorms NXT 2.0 Scorpion*



Fonte: [www.instructables.com](http://www.instructables.com)

A estrutura do robô hexápode deste trabalho possui três graus de liberdade por perna. Isso quer dizer que cada perna do robô possui três motores que rotacionam diferentes direções. Essa estrutura é baseada em sistemas bioinspirados, especialmente em insetos. Como pode ser visualizado na [Figura 15 \(a\)](#), a pata do inseto representado possui três ângulos de movimentação em diferentes eixos, representados por  $\alpha$ ,  $\beta$  e  $\gamma$ .

Figura 15 – Estrutura de patas inspiradas em insetos



Fonte: [Rabelo \(2015\)](#)

Na [Figura 15](#) (b), são apresentadas três abordagens de projeto de patas com três graus de movimentação para robótica (a fim de simular a movimentação apresentada em [Figura 15](#) (a)) utilizando de, respectivamente: motores elétricos, mecanismos pantográficos e atuadores pneumáticos. [Rabelo \(2015\)](#)

### 2.2.1 Estabilidade

Relaciona-se ao número e geometria de pontos em contato com o ambiente. Segundo [Souto \(2007\)](#), [Erden \(2006\)](#) e [Ingram \(2006\)](#), há dois tipos de estabilidade a serem considerados em robôs: estática e dinâmica. A estabilidade estática se dá pela baixa velocidade, quando o robô mantém seu centro de gravidade no chão. A estabilidade dinâmica aparece quando o robô se move em altas velocidades, nas quais a maior parte ou até mesmo todas suas pernas perdem contato com o solo ao mesmo tempo e, assim, ele continua equilibrado através da ação de forças, como a inercial. A opção do robô hexápode apresenta estabilidade estática durante sua locomoção (graças ao maior número de pernas do que um quadrúpede, por exemplo), reduzindo-se a complexidade do projeto, visto que não existe a necessidade de um sistema especificamente destinado ao equilíbrio do robô.

### 2.2.2 Cinemática

Segundo [Demasi \(2012\)](#), a cinemática tem como objetivo descrever o movimento dos corpos sem relacionar as forças que os causaram. Dessa maneira, é conhecida como o estudo da geometria de movimentos. Existem duas abordagens para se realizar a loco-

moção do protótipo. A cinemática direta determina a posição dos pés de acordo com as variáveis das juntas da perna. A cinemática inversa, pelo contrário, determina através do conhecimento das posições do pé o valor das variáveis das juntas.

### 2.2.3 Dinâmica

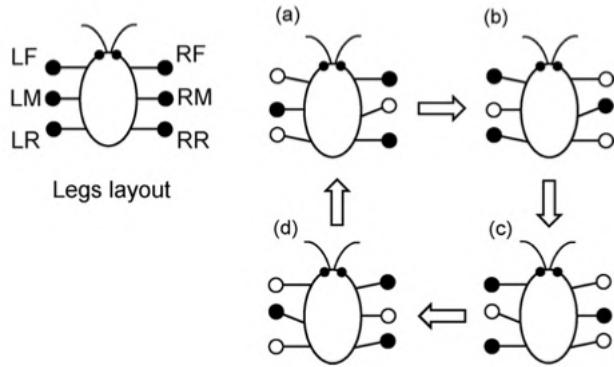
Segundo [Demasi \(2012\)](#), enquanto a cinemática, conforme visto acima, estuda os movimentos sem se preocupar com as causas, a dinâmica estuda o efeito que as forças aplicam no movimento dos corpos.

### 2.2.4 *Gait*

Segundo [Tedeschi e Carbone \(2014\)](#), *gait* é a sequência de movimentos entre as patas do robô que são realizados para possibilitar a sua locomoção. Nesses movimentos repetitivos, cada pata pode estar em contato com o solo ou fora do solo. As patas em contato com o solo têm a função de manter o robô em equilíbrio e realizar sua impulsão para frente, enquanto as patas que estão fora do solo se movimentam no sentido contrário para permitir a movimentação. Existem quatro tipos de *gaits* comumente conhecidos para robôs hexápodes [[\(Tedeschi e Carbone \(2014\) e Rabelo \(2015\)\)](#)].

- *Wave gait*: também conhecido por combinação 5+1 ou *one by one gait*. Nesse padrão, os robôs mantém cinco patas em contato com o solo durante a locomoção. Dessa maneira, há apenas uma pata fora do contato com o solo. Da traseira para a frente, todas as patas de um lado se movem sucessivamente, e depois isso é repetido do outro lado. É o tipo de *gait* com maior estabilidade, porém é também o mais lento.
- *Ripple gait*: nesse padrão, o robô alterna entre cinco e quatro patas no solo. Para movimentação, são levantadas e movidas para a frente duas pernas de lados opostos por vez. Geralmente, é utilizado para velocidades moderadas.
- *Tripod gait*: também conhecido por combinação 3+3. Para esse padrão, o robô separa suas pernas em dois grupos de três, formando um triângulo entre cada grupo. Enquanto um grupo serve de suporte, o outro é levantado e se move para a frente, realizando o passo. Logo após, o outro grupo realiza o mesmo movimento. É o tipo de *gait* mais comum para a implementação de robôs hexápode, assim como também é o que permite a movimentação com maior velocidade. A [Figura 16](#) apresenta um esquemático de como esse padrão é realizado, dividindo o movimento em quatro partes. As pernas representadas pelo círculo preenchido de preto estão erguidas, enquanto as pernas com o círculo com preenchimento em branco estão em contato com o solo.

Figura 16 – Tripod Gait

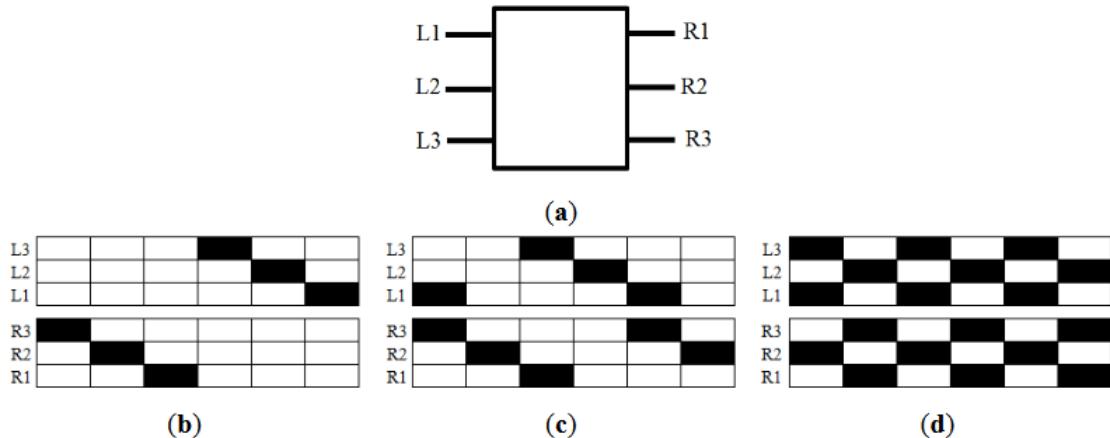


Fonte: [tripodgaitasamodelforrobots.weebly.com/tripod-walking-gait.html](http://tripodgaitasamodelforrobots.weebly.com/tripod-walking-gait.html)

Na primeira movimentação, representada por (a), o robô levanta o primeiro grupo de patas e as movimenta para frente. Em seguida, em (b), esse grupo de patas volta ter contato com o solo, enquanto o outro é levantado. Em (c), o grupo que está em contato com o solo traciona o robô para frente, enquanto o grupo que está levantado se move para frente. Em (d), as pernas que estavam no solo são levantadas e as que estavam fora do solo são abaixadas. Esses quatro movimentos representam uma passada do robô.

- *Free gait*: conforme o nome, é um padrão livre, no qual a inteligência do robô define (através de sensores) qual o melhor tipo de passo a ser utilizado dependendo dos fatores do ambiente, como condições do terreno, por exemplo.

A Figura 17 representa um resumo geral dos padrões apresentados acima. Em (a), está disponível a apresentação da estrutura das pernas de um robô hexápode. Em (b), está a locomoção por meio do *wave gait*, em (c), *ripple gait* e, em (d), *tripod gait*. O preenchimento em preto indica que aquela perna está fora do solo, enquanto os preenchimentos brancos indicam que a perna está em contato com o solo.

Figura 17 – Resumo dos Hexápode *Gaits*Fonte: [Tedeschi e Carbone \(2014\)](#)

## 2.3 Processamento de Imagem

Segundo [Faria \(2010\)](#), para um computador uma imagem pode ser definida como uma função de duas variáveis  $f(x, y)$ , sendo  $x$  e  $y$  suas coordenadas espaciais (para imagens com duas dimensões). O valor obtido em cada coordenada (elemento da imagem) equivale à intensidade daquele *pixel*, e pode se referir ao nível de cinza, cores RGB, entre outros parâmetros. Uma imagem digital possui um valor finito de *pixels*, e é representada nos computadores como uma matriz [[Lima \(2009\)](#)]. A Figura 18 apresenta um exemplo simples de processamento de imagem, com uma aproximação realizada numa faixa de *pixels* para exemplificação dos parâmetros de intensidade de cinza de cada um deles.

Figura 18 – Destaque aos parâmetros dos *pixels* de uma imagemFonte: [Faria \(2010\)](#)

[Albuquerque e Albuquerque \(2000\)](#) diferencia o processamento de imagem da computação gráfica. Enquanto computação gráfica parte de uma informação para se obter uma imagem ou filme, o processamento de imagem faz o caminho contrário, parte de uma imagem para se obter informações a respeito dela. Apesar disso, há técnicas semelhantes e equivalentes que envolvem os dois processos.

Ainda segundo [Albuquerque e Albuquerque \(2000\)](#), o termo análise está relacionado à extração de informações úteis da imagem a ser processada. Nessa fase, são realizadas parametrização de medidas quantitativas, descrevendo-se, assim, diferentes dados que podem ser obtidos dentro de uma imagem. Isso pode ser atribuído a diversas utilidades, como, por exemplo, realizar a contagem e identificação de células dentro de um tecido.

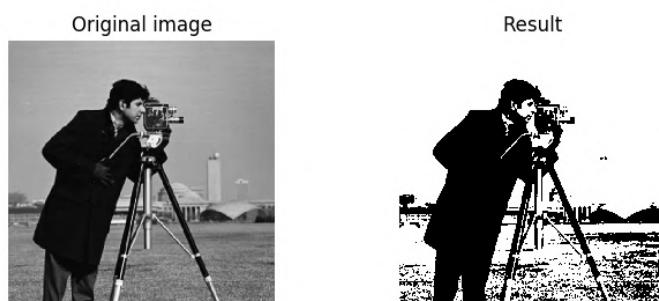
O processamento de imagem, segundo [Faria \(2010\)](#), pode ser dividido em três níveis:

- Baixo nível: somente atua ao nível dos *pixels*, em tarefas como redução de ruído, aumento de contraste e suavização de imagem. Geralmente, tanto a entrada quanto a saída desse processo são imagens.
- Médio nível: envolve tarefas mais avançadas, como identificação de regiões e objetos. O objeto de entrada é uma imagem, mas a saída pode ter informações obtidas da imagem, como contornos e bordas.
- Alto nível: está associado à análise e interpretação do conteúdo da imagem, simulando uma observação humana.

### 2.3.1 Segmentação Binária

Segundo [Jain e Petrol \(2011\)](#) e [Moreira \(2011\)](#), segmentação binária, também conhecida como *thresholding*, é o método mais simples para segmentação em processamento de imagem. A [Figura 19](#) apresenta um exemplo de aplicação desse método.

Figura 19 – Exemplo de aplicação de segmentação binária



Como pode ser observado, a imagem resultado contém apenas cores pretas e brancas. Isso se dá porque o objetivo do *thresholding* é justamente segmentar a imagem de acordo com os parâmetros especificados. Os *pixels* que estão dentro dos parâmetros desejados são alterados para a cor branca, enquanto que os que estão fora dos parâmetros são alterados para a cor preta (fazendo-se, assim, uma imagem binária). Dessa maneira, os próximos processamentos necessários com aquela imagem se tornam muito menos complexos, visto que há apenas um padrão de *pixels* para serem analisados.

### 2.3.2 Sistemas de Cores HSV

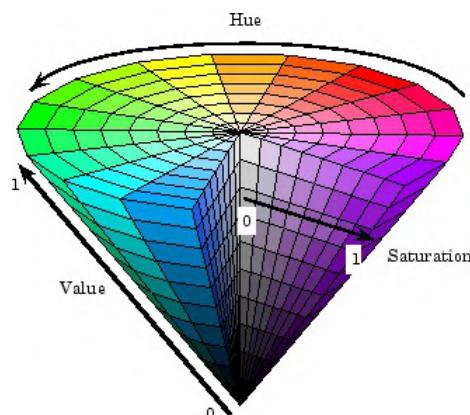
Sistema de cores é a forma em que se organiza os parâmetros de um *pixel* para que o processamento computacional consiga reproduzir a cor especificada de maneira correta.

Segundo Araujo, Mendonça e Freire (2011), o sistema de cores mais robusto e mais indicado para o processamento de imagens para identificação de robôs é o HSV. A sigla é formada pelas componentes *hue* (matiz), *saturation* (saturação) e *value* (valor). Esses componentes são definidos por:

- **Matiz:** está relacionada à tonalidade da cor.
- **Saturação:** está relacionada à pureza da cor. Quanto menos pura, mais próximo de cinza aquela cor se assemelhará.
- **Valor:** está relacionado ao brilho da cor.

A Figura 20 apresenta um esquemático visual do funcionamento dos parâmetros descritos acima.

Figura 20 – Demonstração gráfica dos parâmetros do padrão HSV



Fonte: [www.mathworks.com](http://www.mathworks.com)

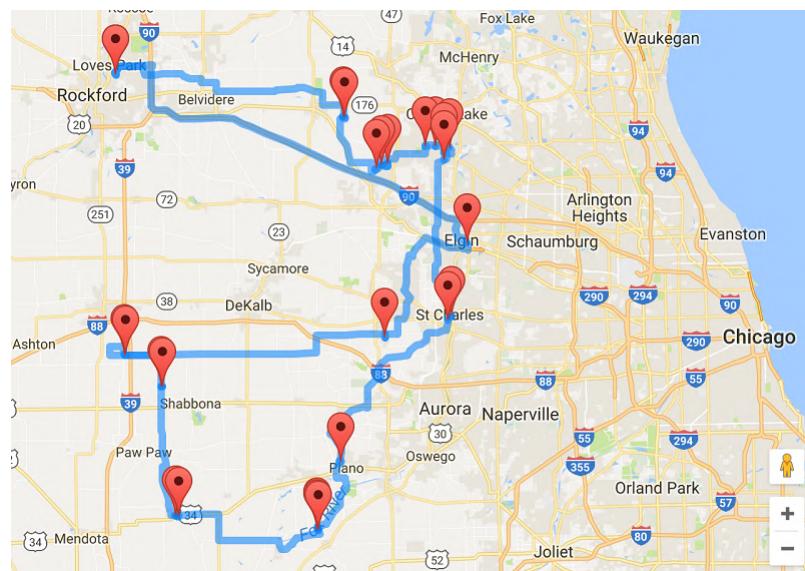
### 2.3.3 OpenCV

Segundo OpenCV (2019), *Open Source Computer Vision Library*, mais conhecido como *OpenCV*, é uma biblioteca de código aberto para computação visual e aprendizado de máquina. Foi desenvolvida para promover uma arquitetura comum para aplicações de processamento de imagem, além de agilizar o desenvolvimento de *softwares* e auxiliar para uma programação mais intuitiva. A biblioteca possui mais de 2500 funções, sendo amplamente utilizada para aplicações de reconhecimento facial, identificação de objetos, rastreamento de objetos móveis, extração de modelos 3D, entre outras. Além disso, a biblioteca também é suportada por praticamente todas as linguagens de programação utilizadas na atualidade.

## 2.4 Navegação por *Waypoints*

Segundo o dicionário *Lexico* (de *Oxford*), *waypoints* são pontos de parada num caminho e coordenadas computacionais de cada estágio de uma jornada de navio ou avião. Esses conceitos estão diretamente relacionados ao uso dessa ferramenta no projeto aqui apresentado. Na Figura 21 está disponível o uso do referencial de *waypoints* na ferramenta *Maps* do *Google*, na forma de pinos vermelhos ordenados e posicionados em cada localização espacial (coordenada) que se deseja atingir na trajetória.

Figura 21 – Exemplo de uso de *waypoints* no *Google Maps*



Fonte: [maps.google.com](https://maps.google.com)

Essa ideia é aplicada ao uso de aplicativos de GPS. Pode se ter apenas um *waypoint*, no caso de se apenas desejar chegar a um local, ou então múltiplos *waypoints*, caso

hajam paradas pelo caminho até a coordenada final. No caso de localizações terrestres, geralmente são compostos de informações de latitude e longitude.

## 2.5 Comunicação Sem Fios

De acordo com [Coulouris et al. \(2018\)](#), as redes sem fio utilizam de tecnologias como radiofrequência ou infravermelho para transmissão de dados, eliminando-se, assim, a necessidade de utilização de fios. O objetivo principal é fornecer conexão a dispositivos móveis, além de poder eliminar a necessidade de cabos para interligar computadores em casas ou prédios.

As redes Wi-Fi utilizadas residencialmente, por exemplo, trabalham com radiofrequências na faixa de 2,4 GHz e 5 GHz.

### 2.5.1 XBee

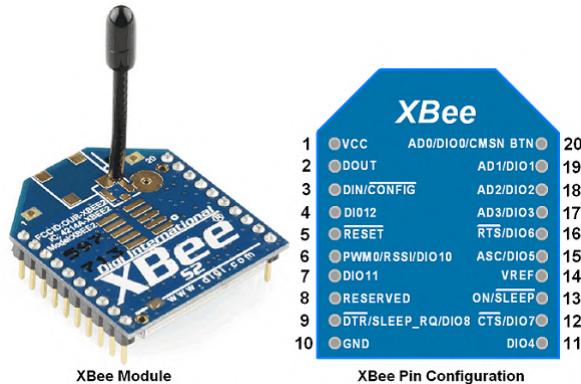
Segundo [MountainBaja \(2018\)](#), os módulos *XBee* trabalham com radiofrequência para realizar a comunicação serial entre dois (ou mais) pontos. Trabalham com taxas de transferência de dados entre 20 kbps e 250 kbps, através do protocolo *ZigBee*. De acordo com [IFSC \(2015\)](#), esse protocolo possui especificações para comunicação sem fios entre dispositivos eletrônicos, com foco em baixa potência de operação e baixo consumo de energia.

[IFSC \(2015\)](#) explica também que os módulos *XBee* são compostos por um microcontrolador e um receptor. O microcontrolador armazena o *firmware*, que contém os padrões do protocolo utilizado e a definição de comportamento daquele módulo, que é configurada pelo programador através do *software* da ferramenta. As opções de comportamento são coordenador, roteador ou dispositivo final.

Por conta da robustez e segurança da tecnologia, ela possui diversas aplicações, como acompanhamento de atividades de alto risco e automação residencial, comercial e industrial.

A [Figura 22](#) apresenta uma fotografia e o esquemático dos pinos de um módulo *XBee*.

Figura 22 – Módulo XBee



Fonte: [www.electronicwings.com](http://www.electronicwings.com)

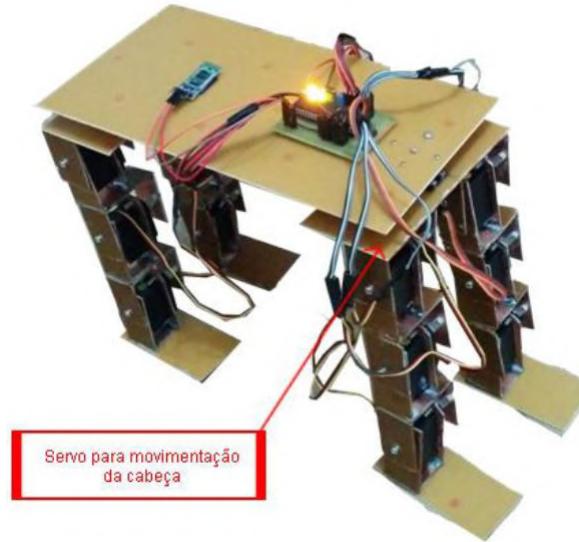
Por serem módulos bastante utilizados e difundidos no mercado, existem diversos componentes para integração com outras tecnologias, como *shields* para conexão direta com placas *Arduino* e a placa *XBee Explorer*, que possibilita a conexão do módulo diretamente com um computador através da porta USB.

### 3 Trabalhos Relacionados

A fim de sempre procurar manter uma melhoria contínua dos projetos científicos e de buscar uma base de informações para o desenvolvimento, foi realizada uma busca de monografias como esta apresentada que se relacionam no tema de robótica móvel com movimentação por pernas.

[Alves \(2014\)](#) desenvolveu como seu trabalho de conclusão do curso de Engenharia Eletrônica do ITA o protótipo de um robô quadrúpede de baixo custo para utilização nos laboratórios de sua própria instituição. Ele destaca o uso da topologia em estrela para ligação dos servomotores, o que torna seu robô menos suscetível a erros do que a versão em anel. A comunicação de sinais é realizada via *bluetooth* e uma aplicação em *Java* foi desenvolvida para controle dos movimentos via computador. O microcontrolador utilizado é o PIC. A [Figura 23](#) apresenta uma foto do robô desenvolvido nesse projeto.

Figura 23 – Robô Desenvolvido



Fonte: [Alves \(2014\)](#)

[Demasi \(2012\)](#) apresentou em sua dissertação de mestrado uma proposta de um mecanismo de três graus de liberdade para ser utilizado como uma perna de um robô hexápode. Para isso, ele utilizou de modelos teóricos, determinados algoritmos e equações para comprovar a estabilidade do sistema. Os recursos necessários para cálculos dos modelos matemáticos foram *MatLab* e *Simulink*. O projeto mecânico em si foi desenvolvido em ambiente *SolidWorks*. Uma implementação física do *hardware* em si não foi realizada.

[Kretzschmar \(2012\)](#) em seu trabalho de conclusão de curso realizou a implementa-

ção de um robô quadrúpede utilizando como microcontrolador a placa *Arduino*. O projeto também foi focado numa construção de baixo custo. O robô foi capaz de efetuar uma locomoção programada para qualquer direção em ambientes controlados, conforme o esperado. É interessante citar que para esse projeto o material utilizado foi madeira de pinus, o que trouxe desafios na confecção das peças do robô. A Figura 24 apresenta uma foto do robô desenvolvido nesse projeto.

Figura 24 – Robô com Acabamento em Madeira



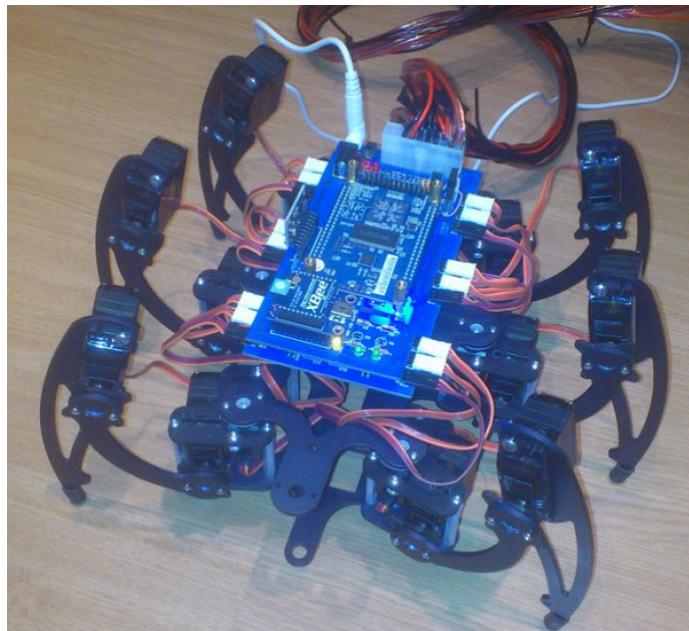
Fonte: [Kretzschmar \(2012\)](#)

O grupo [Fontoura, Nascimento e Gioppo \(2013\)](#) trouxe em sua monografia de graduação a implementação de um robô hexápode com três motores por perna, porém com a diferenciação de utilizar como controle uma placa FPGA com o processador embarcado NIOS II. Foi desenvolvido um *firmware* na linguagem C++ e uma aplicação gráfica para movimentação do robô em linguagem Java. Foi destacada a vantagem da placa FPGA, visto que, por ser um *hardware* sintetizável, a capacidade de expansão e adaptação do projeto é bem notável. A Figura 25 apresenta uma foto do robô desenvolvido nesse projeto.

[Oliveira \(2016\)](#) em sua tese de mestrado realizou um estudo sobre a modelagem e simulação de robôs hexápodes em sistemas mecânicos virtuais, a fim de analisar os dados e as trajetórias que seriam obtidas com o robô. Essa simulação permite viabilizar a pesquisa para modelos que se aproximam dos valores ideais. As ferramentas utilizadas foram *MatLab*, *Simulink*, *SimMechanics* e *SolidWorks*.

[Dantas \(2014\)](#) apresentou em sua monografia de conclusão de curso um protótipo funcional de um robô quadrúpede com três graus de liberdade por perna. O estudo foi dividido em três etapas: modo de locomoção (marcha "caranguejo" descontínua de duas

Figura 25 – Robô Hexápode



Fonte: [Fontoura, Nascimento e Gioppo \(2013\)](#)

fases), criação de modelo cinemático e construção do robô. A simulação da cinemática foi realizada via *MatLab* e o controle dos movimentos também através de uma placa *Arduino*.

Rabelo (2015) realizou a construção de um robô hexápode tipo *Phoenix* de baixo custo para operar em ambientes irregulares e hostis. Foi construída uma modelagem no programa *Solid Edge* e o modelo utilizou acrílico como material de estrutura. O micro-controlador escolhido foi o PIC18F4550, o que garantiu a programação do sistema em linguagem C.

Por sua vez, Medeiros (2014) realizou, em sua dissertação de mestrado, o estudo de um controlador de seguimento de caminhos para robôs móveis com pernas, de forma a que esse controlador faça com que o centro de gravidade do robô fique o mais próximo possível uma sequência pré estabelecida de *waypoints*. As simulações foram realizadas em *MATLAB/Simulink/SimMechanics*. O controlador foi testado com três tipos de robôs móveis: quadrúpede com esterçamento na articulação frontal; quadrúpede com esterçamento frontal e traseiro; hexápode. A localização do robô foi definida com base em reconhecimento de imagem por câmeras instaladas no teto da sala. Os resultados demonstraram que sua implementação foi mais eficiente para o robô quadrúpede com esterçamento frontal.

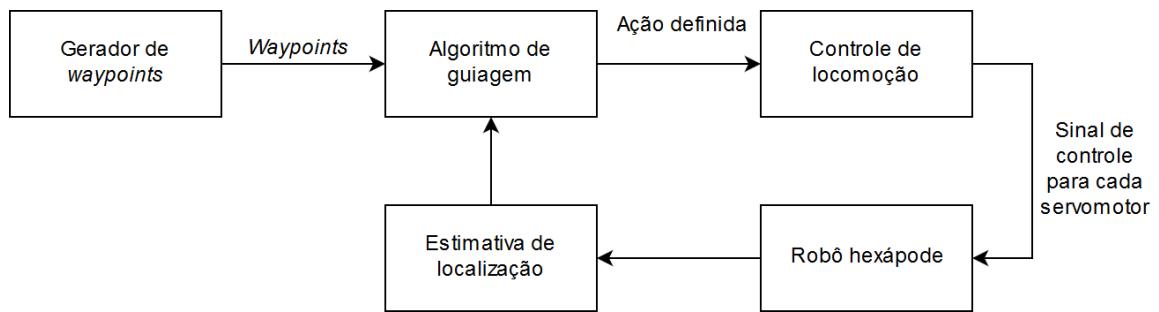
# 4 Desenvolvimento

Neste capítulo, serão discutidos os componentes, montagens e implementações essenciais para a montagem de todos mecanismos do robô, além do proposto sistema de navegação e de guiagem para o seguimento de caminhos, viável para ambientes internos.

## 4.1 Visão Geral do Sistema Proposto

Para se esquematizar o desenvolvimento do sistema, foi criado o diagrama de blocos gerais, que apresenta uma visão integrada dos sistemas do projeto. Esse diagrama está disponível na [Figura 26](#).

Figura 26 – Diagrama de bloco para seguimento de caminhos



Fonte: Autoria própria

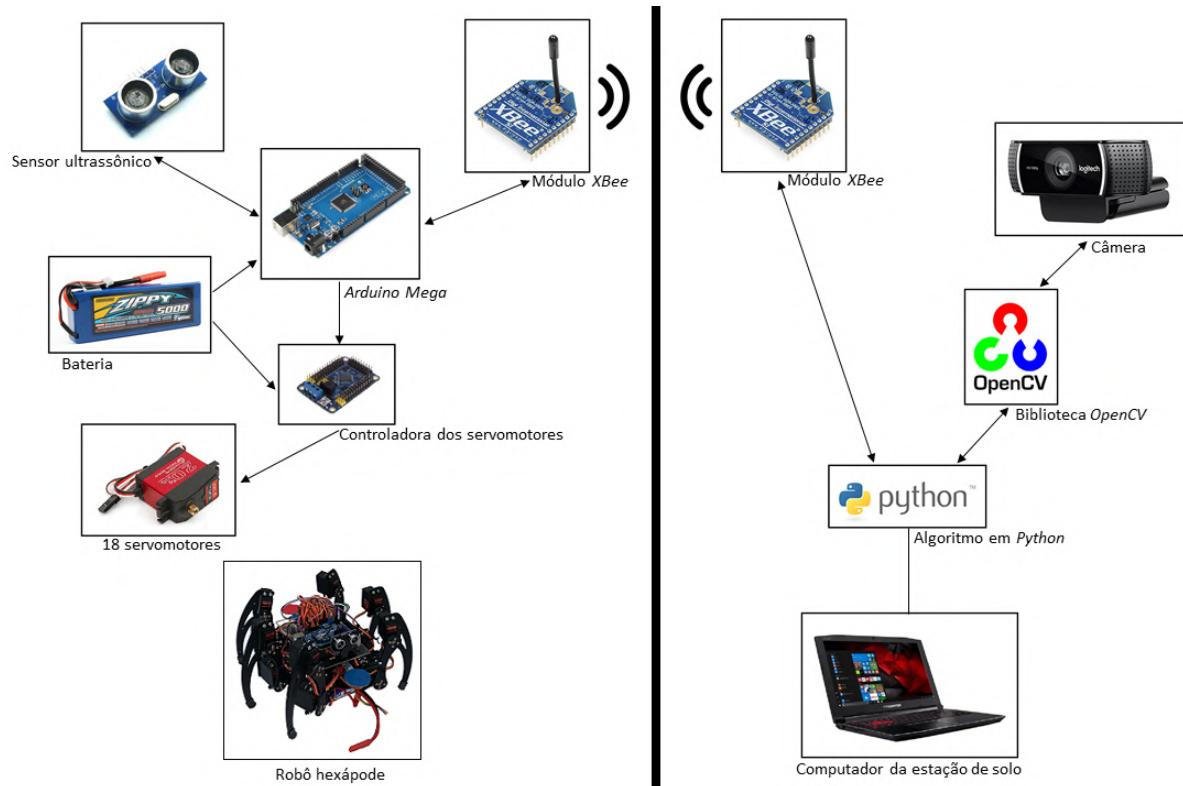
Cada elemento do diagrama é definido por:

- **Gerador de *waypoints***: mecanismo para se gerar e armazenar as coordenadas alvo que se deseja que o robô atinja em seu seguimento de caminho.
- **Algoritmo de guiagem**: utilizando-se da informação de coordenada alvo (*waypoint*) e da localização estimada do robô, esse algoritmo deve enviar ao controle de locomoção os dados necessários para que o robô defina qual movimentação realizar.
- **Controle de locomoção**: sistema que, a partir dos dados do algoritmo de guiagem, define a locomoção que o robô deve realizar, a partir do controle da posição dos eixos dos servomotores.
- **Robô hexápode**: o robô móvel em si, que deve desempenhar a atividade de movimentação e passagem pelos *waypoints* definidos.

- **Estimativa de localização:** sistema de processamento de imagem que, obtendo a imagem do robô, estima a coordenada em que ele se localiza no ambiente de testes.

Os dispositivos utilizados para a solução do problema estão dispostos na [Figura 27](#). As setas bidirecionais indicam que a comunicação de dados entre aqueles componentes ocorre nos dois sentidos, enquanto as setas simples indicam um fluxo de apenas um sentido.

Figura 27 – Diagrama dos componentes do sistema



Fonte: Autoria própria

A porção à esquerda da divisória central contém os responsáveis pelo funcionamento da unidade móvel, o robô hexápode. Esses componentes possuem como finalidade:

- **Sensor ultrassônico:** dispositivo utilizado para identificação de obstáculos à frente do robô.
- **Módulo XBee:** módulo instalado no robô para comunicação sem fio entre a unidade móvel e a estação de solo.
- **Arduino Mega:** placa com o microcontrolador *ATmega 2560* embarcada no robô para a execução do processamento de definição de movimentação dos servomotores.

- **Bateria:** fornece a alimentação necessária para todos os componentes instalados no robô.
- **Controladora dos servomotores:** circuito para conexão dos motores do robô. Define a movimentação através de sinais seriais de entrada.
- **18 servomotores:** motores responsáveis pela movimentação do robô. Sendo um robô de seis pernas, são utilizados três motores por perna.
- **Robô hexápode:** a unidade móvel em si, que possui todos os componentes citados acima instalados.

Por sua vez, a porção à direita da divisória apresenta as tecnologias utilizadas na estação de solo.

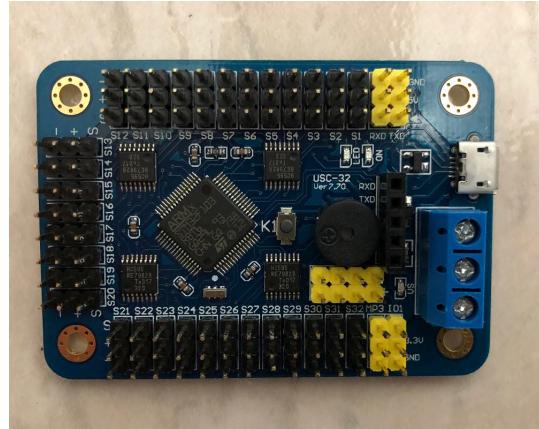
- **Módulo XBee:** módulo instalado no computador da estação de solo para comunicação sem fio entre a unidade móvel e a estação de solo.
- **Câmera:** dispositivo externo de captura de imagem para estimativa da localização do robô no ambiente de testes (ver [subseção 4.4.1](#)).
- **Biblioteca OpenCV:** biblioteca utilizada para implementação do algoritmo de processamento de imagem.
- **Algoritmo em Python:** algoritmo de processamento de imagem e de troca de mensagens com o robô.
- **Computador da estação de solo:** computador no qual os dispositivos da estação de solo estão conectados e o algoritmo de processamento de imagem é executado.

Os elementos apresentados serão discutidos com maior detalhamento nas seções a seguir.

## 4.2 Montagem do Robô Hexápode

Sob a premissa de se montar um robô hexápode, surgiu, então, a necessidade de se adquirir a estrutura do projeto. O robô utilizado no trabalho é constituído por 18 servomotores (modelo DS3218MG) e todas as peças estruturais necessárias para a realização da montagem estrutural, inclusive parafusos e também uma placa controladora de 32 canais (*32-Channel Servo Controller*, da *ToRobot*, disponível na [Figura 28](#)). Além disso, o kit possui uma ótima qualidade de construção, o que possibilitará diversos trabalhos futuros para aperfeiçoamento da implementação que já foi realizada.

Figura 28 – ToRobot 32-Channel Servo Controller



Fonte: Autoria própria

Um exemplo de servomotor do modelo DS3218MG está disponível na [Figura 29](#). Trata-se de um motor com torque elevado (20 kg/cm) e rolamentos de metal, o que é bastante robusto e ideal para a aplicação no projeto.

Figura 29 – Servomotor DS3218MG



Fonte: <https://www.bighobby.cz>

A [Figura 30](#) apresenta o reconhecimento inicial de cada peça. Nela pode-se observar os servomotores e seus suportes, as juntas das pernas, o esqueleto central e todos os parafusos que acompanharam o kit adquirido. O material empregado na construção das peças é metálico.

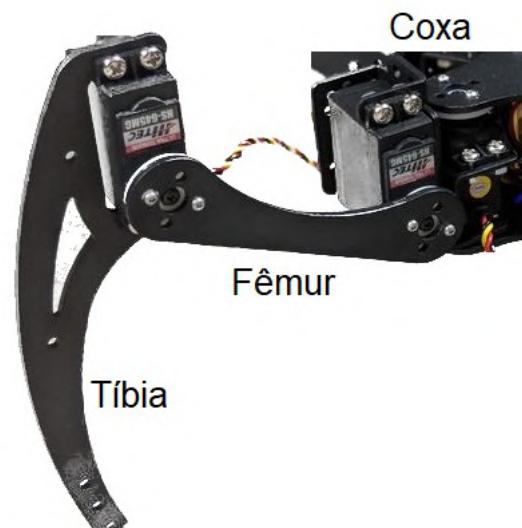
Figura 30 – Reconhecimento das peças



Fonte: Autoria própria

A [Figura 31](#) apresenta o modelo que exibe o posicionamento dos motores em uma das pernas do robô. Assim, pode-se observar os componentes (servomotores e conexões) que são responsáveis pela movimentação da coxa, pelo fêmur (osso da coxa) e pela tíbia (osso da canela), simulando o funcionamento de uma perna de um inseto.

Figura 31 – Segmentos da Perna



Fonte: [www.instructables.com/id/Capers-II-a-Hexapod-Robot/](http://www.instructables.com/id/Capers-II-a-Hexapod-Robot/) (adaptação)

O primeiro passo da montagem foi o encaixe dos suportes dos servomotores que correspondem ao movimento da coxa e do fêmur. Cada uma das seis pernas possui uma unidade desse conjunto, pois é ele que permite a fixação da perna à estrutura central do robô. A [Figura 32](#) apresenta uma foto desses suportes montados, já com os servomotores instalados.

Figura 32 – Suportes dos servomotores



Fonte: Autoria própria

Em seguida, realizou-se a montagem da parte inferior da perna, correspondente ao servomotor que interliga o fêmur à tíbia. Para isso, é necessário apenas parafusar o servomotor na peça estrutural correspondente à tíbia do robô. Essa montagem está apresentada na foto da [Figura 33](#).

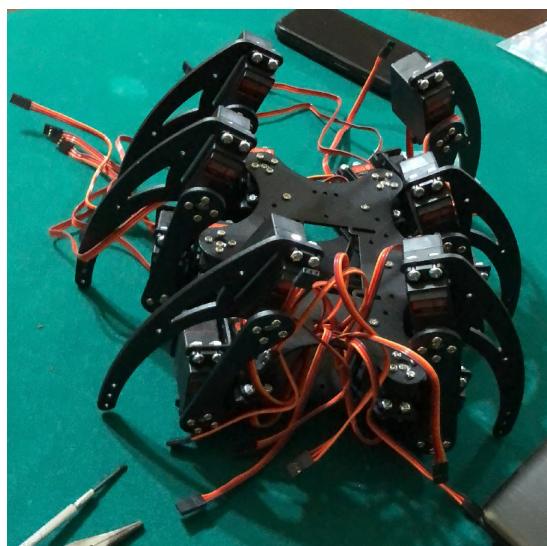
Figura 33 – Encaixe do servomotor na tíbia



Fonte: Autoria própria

A [Figura 34](#) apresenta a finalização da montagem inicial do chassi do robô, contendo apenas componentes estruturais. Nela, pode-se observar a conexão das pernas à estrutura central do robô e a localização de cada eixo responsável pelos três graus de movimentação por perna.

Figura 34 – Finalização da montagem estrutural

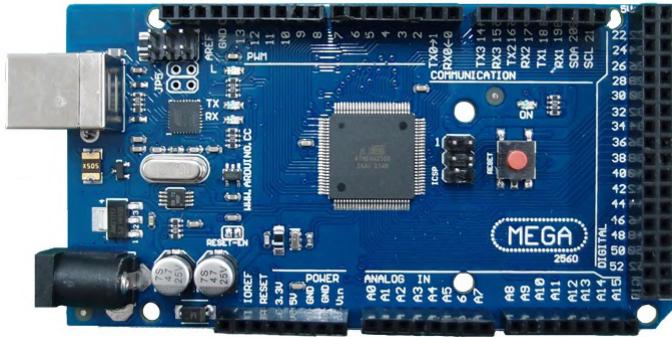


Fonte: Autoria própria

A plataforma escolhida para o processamento embarcado do hexápode é o *Arduino Mega 2560* ([Figura 35](#)), composta por um microcontrolador *AVR Atmega 2560* de 8-bit

que possui uma arquitetura Harvard com 256 kB de memória de programa e 8 kB de memória de dados. Sendo um modelo mais avançado, destinado a projetos mais complexos, como robótica [Arduino (2018)], essa placa possui 54 pinos de comunicação digital (que podem ser configurados como entrada ou saída). Desses 54 pinos, 15 podem ser configurados como PWM. Além disso, há também 16 pinos de entrada analógica, quatro interfaces seriais e uma I2C.

Figura 35 – *Arduino Mega*



Fonte: [www.makerlab-electronics.com/](http://www.makerlab-electronics.com/)

As placas *Arduino* possuem conexão USB para comunicação com o computador e são facilmente programadas pela IDE nativa.

Para a fixação do *Arduino* e da placa de controle dos servomotores, utilizou-se uma placa acrílica na medida de 16 cm x 10 cm.

Para a alimentação do robô, foi utilizada uma bateria de Lítio Polímero (LiPo) com duas células de 3.7 V do modelo *Zippy 20C Series*. Essa bateria é capaz de fornecer uma corrente de 5000 mAh e tensão de 7,4 V. Ela é suficiente para garantir o funcionamento de aproximadamente 15 minutos de carga dos servomotores. Além disso, a tensão possui um valor que é compatível com todos os dispositivos utilizados no hexápole, dispensando a necessidade de utilização de conversores de tensão. A [Figura 36](#) apresenta uma fotografia dessa bateria, já instalada na parte inferior do robô.

Figura 36 – Bateria Li-PO Zippy 20C Series 5000 mAh 7.4 V



Fonte: Autoria própria

A fim de se evitar a utilização de fios para a comunicação entre a estação de solo (computador na bancada) e o sistema embarcado no robô, decidiu-se implementar um sistema de transmissão de dados sem fios. A tecnologia escolhida foi a *XBee*, devido a fácil implementação de controle em *Python* e a integração com *Arduino*. Na Figura 37 tem-se à esquerda o módulo *XBee PRO* instalado na *shield* compatível com o *Arduino* (a ser utilizado de maneira embarcada no robô), e à direita encontra-se o outro módulo *XBee*, responsável pelo envio de dados da estação de solo. Como esse módulo necessita de comunicação com o computador, está instalado numa placa *XBee Explorer*, que possui uma conexão do tipo USB.

Figura 37 – Shield *XBee* para *Arduino* e *XBee Explorer*

Fonte: Autoria própria

O último componente adicionado ao hexápode foi um sensor de distância ultrasônico do modelo HC-SR04. Esse sensor consegue mensurar com precisão a distância de objetos à frente do robô. Esse componente é utilizado a fim de que o robô pare de se

movimentar quando houver um objeto à sua frente, evitando, assim, uma colisão. A movimentação para o próximo *waypoint* é retomada assim que o sensor deixa de detectar o obstáculo. A [Figura 38](#) apresenta uma fotografia desse componente.

Figura 38 – Sensor de Distância Ultrassônico HC-SR04



Fonte: [www.filipeflop.com/](http://www.filipeflop.com/)

A conexão dos servomotores com a placa *ToRobot* ([Figura 28](#)) está disposta como apresentado na [Tabela 1](#). As referências aqui apresentadas sobre a definição de cada servomotor de cada perna está de acordo com a nomenclatura apresentada na [Figura 31](#). Para esse trabalho, os 14 canais sobressalentes não foram utilizados. Porém, será possível utilizá-los em trabalhos futuros para um maior horizonte de possibilidades de implementações. A letra "S", apresentada antes do número de cada canal, indica que aquela porta é designada à conexão de servomotores.

Tabela 1 – Canais do circuito *ToRobot* conectados aos servomotores

Perna	Tíbia	Fêmur	Coxa
<b>Frontal esquerda</b>	S1	S3	S2
<b>Média esquerda</b>	S32	S31	S6
<b>Traseira esquerda</b>	S29	S27	S28
<b>Frontal direita</b>	S12	S11	S10
<b>Média direita</b>	S17	S19	S18
<b>Traseira direita</b>	S22	S23	S24

Fonte: Autoria própria

A placa *Arduino* embarcada no robô é conectada por meio das portas e componentes definidos na [Tabela 2](#) e no esquemático apresentado na [Figura 39](#).

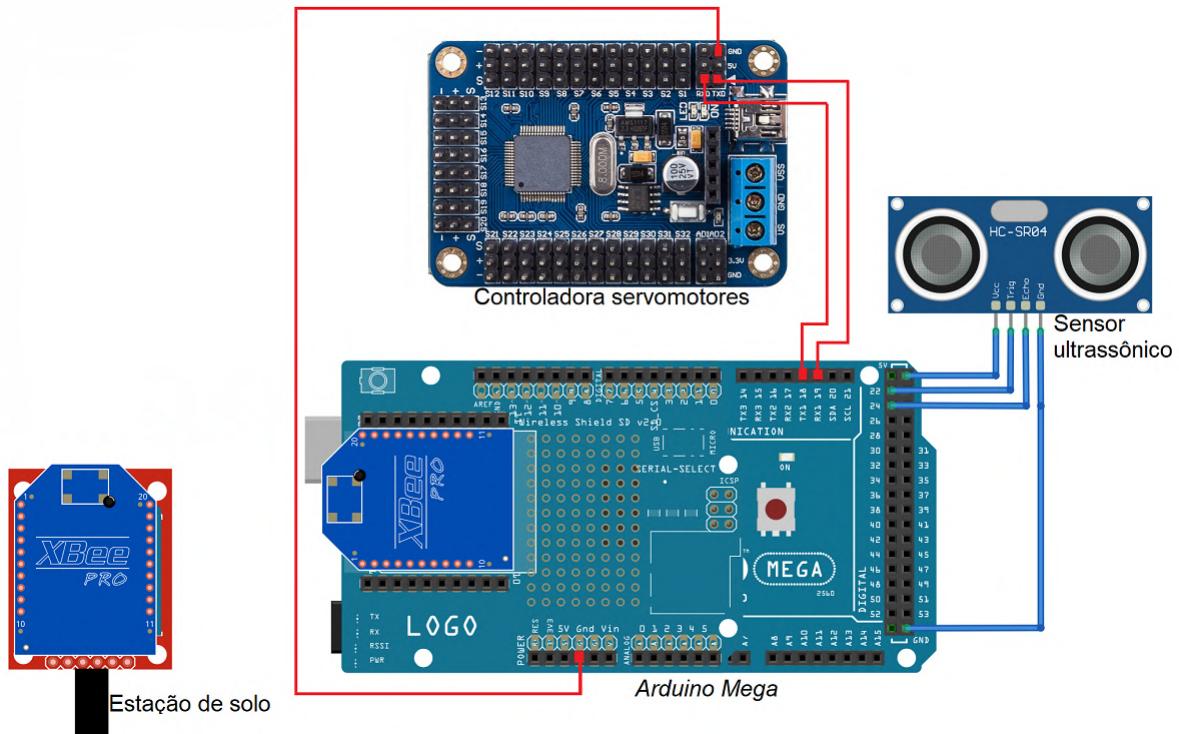
Tabela 2 – Conexões realizadas no *Arduino* embarcado

Conexões	Tipo de comunicação	Função
22	Digital	Porta <i>trigger</i> para o sensor ultrassônico
24	Digital	Porta <i>echo</i> para o sensor ultrassônico
RX1	Serial	Porta de recepção dos dados para comunicação com a controladora dos servomotores
TX1	Serial	Porta de transmissão dos dados para comunicação com a controladora dos servomotores
<i>Shield XBee</i>	Serial	Comunicação com o <i>XBee</i> para realizar a troca de dados com a estação de solo

Fonte: Autoria própria

No esquemático, estão omitidas as conexões de alimentação e as conexões dos servomotores (ver [Tabela 1](#)). Os fios vermelhos indicam a ligação do *Arduino Mega* com a placa controladora de servomotores, enquanto os fios azuis representam a ligação com o sensor ultrassônico. A *Shield XBee* está instalada acima do *Arduino*.

Figura 39 – Esquemático das conexões



Fonte: Autoria própria

A [Figura 40](#) apresenta o resultado final de toda montagem física do robô, que já

está equipado com todos os componentes descritos no decorrer dessa seção.

Figura 40 – Hexápode concluído



Fonte: Autoria própria

### 4.3 Controle da Locomoção

O padrão escolhido para a caminhada do robô foi o *tripod gait*, que apresenta movimentação conforme explicado previamente na [Figura 16](#).

O circuito *ToRobot*, apresentado na [seção 4.2](#), possui espaços de memória reservados para gravação de sequências de movimentação dos servomotores. Essas sequências de movimentação são chamadas de *action groups* (A.G.). A placa pode armazenar vários A.G. em sua memória, e cada A.G. pode conter múltiplos comandos sequenciais para alteração da posição do eixo dos servomotores.

A [Tabela 3](#) apresenta o referencial desenvolvido para a movimentação do robô, incluindo o tempo, em milissegundos, que foi determinado para que cada grupo de movimentação seja completamente realizado. As posições de memória 1, 2 e 3 não foram utilizadas.

Tabela 3 – Action groups guardados na memória do circuito *ToRobot*

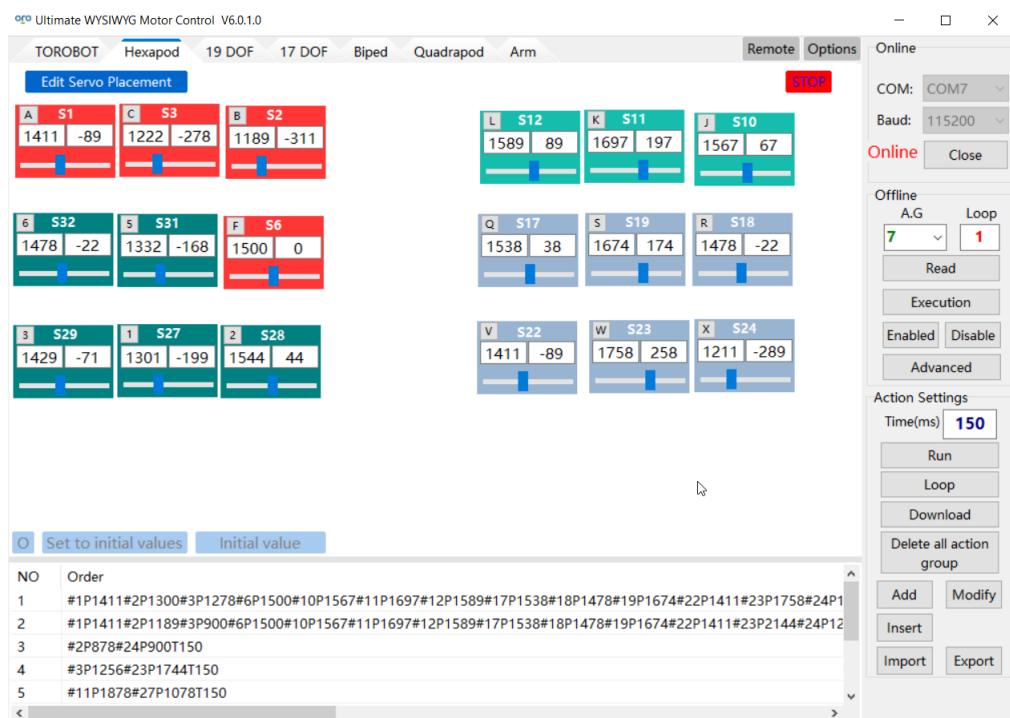
Action group	Função	Tempo total
4	Ficar de pé	1000 ms
5	Dar um passo a frente	800 ms
6	Virar a direita	1350 ms
7	Virar a esquerda	1350 ms

Fonte: Autoria própria

Tendo-se os A.G. guardados na memória do sistema, basta ao *Arduino* embarcado referenciá-los através de comunicação serial para que os servomotores do robô executem a devida locomoção desejada.

O referencial de movimentação por ações de grupo foi desenvolvido através do programa *Ultimate WYSIWYG Motor Control V6.0.1.0*, que é compatível com o circuito de controle dos servos *ToRobot*. Os ajustes definidos para cada servomotor foram realizados empiricamente.

Através da estrutura disponível no programa, foi criada uma aba específica para o controle do hexápode, com seis conjuntos de três motores (conforme referência da Tabela 1), de forma a se facilitar a representação de cada perna do robô para definição dos parâmetros.

Figura 41 – *Ultimate WYSIWYG Motor Control V6.0.1.0*

Fonte: Autoria própria

Como pode ser visto na [Figura 41](#), cada módulo de servomotor possui uma barra ajustável. Essa barra corresponde à posição relativa do eixo do motor, e seus valores podem variar de 500 a 2500 ( $0^\circ$  a  $180^\circ$ , respectivamente).

Na porção inferior da imagem, se armazenam as ordens de movimentação adicionadas. São esses grupos de ordens que podem ser armazenados como A.G. pelo sistema.

Analizando-se a porção lateral direita da janela do programa (faixa de opções), na parte superior tem-se as informações sobre a conexão do computador com a placa de controle dos servos. No caso, eles estão conectados através da porta USB, que está utilizando a comunicação serial COM7 do computador. Além disso, a taxa de transmissão de dados escolhida é de 115200 bits/s, que é o padrão do programa. A seção *Time* deve ser preenchida com o tempo no qual se deseja que os servomotores executem as instruções.

Através da realização desses ajustes manualmente para todos os motores do lado esquerdo do robô, foi possível a replicação da lógica utilizada para o lado direito, apenas espelhando a rotação, visto que esses motores trabalham em simetria.

O formato de instrução compatível com a placa *ToRobot* está especificado através da [Tabela 4](#). Esse referencial foi utilizado para se definir como o *Arduino* embarcado deve enviar seus sinais de controle para o circuito *ToRobot*.

Tabela 4 – Formato de instruções para a placa *ToRobot*

Objetivo	Instrução	Descrição
Controlar um servomotor	#1P1500T100	<ul style="list-style-type: none"> <li>- O dado 1 (após o #) se refere ao canal do servomotor</li> <li>- O dado 1500 (após o P) se refere à posição do eixo do servomotor</li> <li>- O dado 100 (após o T) se refere ao tempo a ser executado o movimento (em milissegundos)</li> </ul>
Controlar múltiplos servomotores ao mesmo tempo	#1P600#2P900#8P2500T100	<ul style="list-style-type: none"> <li>- Os dados 1, 2 e 8 se referem aos canais dos servomotores</li> <li>- Os dados 600, 900 e 2500 se referem às posições dos eixos dos servomotores</li> <li>- O dado 100 (após o T) se refere ao tempo a ser executado o movimento (em milissegundos)</li> </ul>
Executar um action group	#1GC2	<ul style="list-style-type: none"> <li>- O dado 1 (após o #) se refere ao número do A.G</li> <li>- O dado 2 (após o C) se refere à quantidade de vezes que será repetido o A.G</li> </ul>
Executar múltiplos <i>action groups</i> em sequência	#1G#3G#1GC2	<ul style="list-style-type: none"> <li>- Os A.G de número 1, 3 e 1 são executados em sequência</li> <li>- O dado 2 identifica que os A.G serão repetidos duas vezes</li> </ul>

Fonte: Documentação da placa *ToRobot 32-Channel Servo Controller* (adaptação)

A [Tabela 12](#), [Tabela 13](#) e [Tabela 14](#), que estão disponíveis no [Apêndice B](#), contém as instruções utilizadas pelos *action groups* definidos na [Tabela 3](#).

## 4.4 Estimativa de Localização

Nessa seção, serão estudadas as tecnologias utilizadas para o processamento de imagem do sistema.

### 4.4.1 Sistema de Visão Externa

O sistema para identificação da posição espacial do robô necessita, primariamente, de um dispositivo de captura de imagem. O modelo *C922 PRO* da marca *Logitech* foi escolhido para o sistema. Essa câmera possui especificações de captura com resolução 1920x1080 e uma taxa de atualização de 30 quadros por segundo ou 1280x720 com taxa de atualização de 60 quadros por segundo. A câmera também possui ajuste de foco automático, o que elimina a necessidade de se realizar ajustes de calibragem para o sistema.

A [Figura 42](#) apresenta o modo que a câmera foi instalada no ambiente de validação experimental do seguimento de caminho.

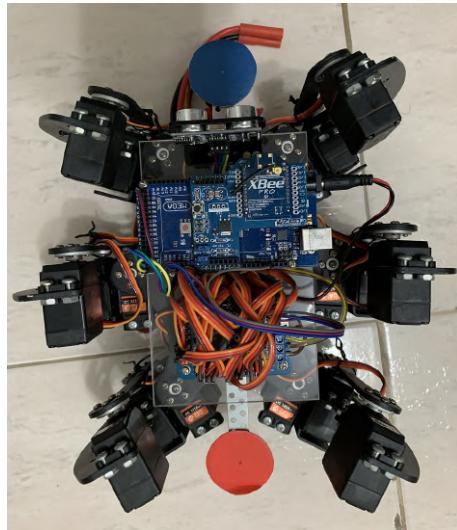
Figura 42 – Câmera instalada no teto da sala



Fonte: Autoria própria

Para o algoritmo de processamento de imagem identificar a localização do robô e definir qual a sua orientação, são necessários no mínimo dois pontos de conhecimento definido. Dessa maneira, foram fixados ao chassi do robô dois marcadores em formato circular, um da cor vermelha e outro da cor azul, com 4 centímetros de diâmetro. A peça azul foi posicionada na parte frontal do robô, enquanto a vermelha na traseira. A [Figura 43](#) apresenta uma fotografia da visão superior do robô, que é a mesma visão que a câmera encontrará da maneira em que está posicionada.

Figura 43 – Visão superior do robô



Fonte: Autoria própria

Pode-se verificar a possibilidade de se utilizar marcadores com diâmetro inferior a 4 cm, porém isso deve ser ponderado com a altura em que a câmera foi instalada. Para o presente projeto, o diâmetro escolhido apresenta o tamanho necessário para a correta identificação.

#### 4.4.2 Algoritmo de Processamento de Imagem

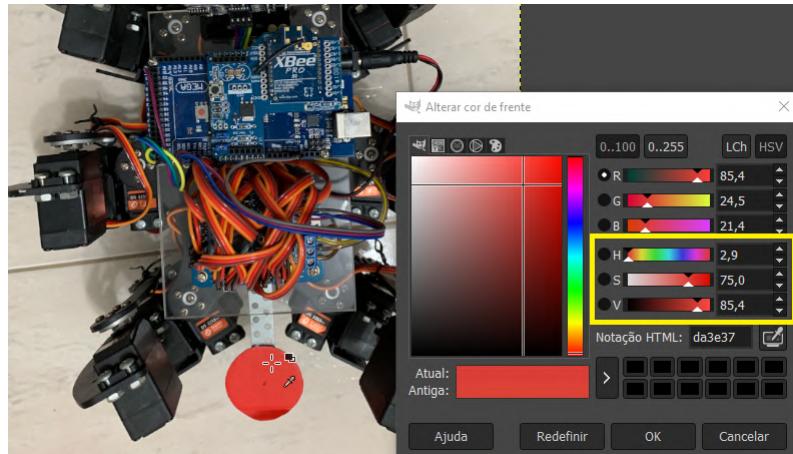
Neste projeto, o algoritmo de processamento de imagem executado na estação de solo tem a função de estimar a localização do hexápode no ambiente. Para essa implementação, foi utilizada a linguagem de programação *Python*, na versão 3.7 64-bit e *OpenCV* na versão 4.0.0. O ambiente de execução é baseado no sistema operacional *Windows 10 Home* 64-bit.

O algoritmo implementado é baseado no estudo do artigo de Araujo, Mendonça e Freire (2011), com algumas adaptações para o projeto.

O primeiro passo necessário é a identificação dos parâmetros dos *pixels* relacionados aos marcadores fixados no robô, discutidos na subseção anterior. Para isso, foram registradas fotografias com condições de iluminações diferentes (luzes acesas, apagadas, e o robô posicionado em locais diferentes da sala), a fim de se obter todos os casos possíveis de identificação. Para se obter os parâmetros dos *pixels*, essas imagens foram analisadas através do programa GIMP 2.10.8 com a ferramenta de seleção de cores. Dessa forma, é possível obter os parâmetros do tipo HSV (ver subseção 2.3.2). O modelo HSV foi utilizado pois, segundo Araujo, Mendonça e Freire (2011), é a abordagem mais robusta e mais adequada para segmentação de cores. A Figura 44 apresenta um *screenshot* de uma

região da tela na qual foi utilizada a ferramenta. A porção que indica a apresentação dos valores HSV está destacada em amarelo.

Figura 44 – Ferramenta de seleção de cores do GIMP



Fonte: Autoria própria

Após a mudança de escala (visto que o GIMP trabalha com uma escala diferente da que o *OpenCV* está baseado) e a ponderação necessária entre os valores mínimos e máximos para cada parâmetro (matiz, saturação e valor), a Tabela 5, que contém os valores referenciais para cada cor, foi construída. Esses valores precisam ser amplos o bastante para se reconhecer os círculos nas mais diversas condições de iluminação e reflexo, assim como também precisam ser restritos o suficiente para se evitar erros de identificação com outros possíveis objetos do ambiente.

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

Cor	H (matiz)	S (saturação)	V (valor)
Azul (parte frontal)	80 a 120	140 a 240	87 a 130
Vermelho (parte traseira)	0 a 15	130 a 240	120 a 180

Fonte: Autoria própria

A câmera foi setada na resolução 1024 x 768. O passo a passo do algoritmo está descrito a seguir. Ele é executado tanto para a identificação do círculo de cor vermelha quanto do azul.

1. A imagem é capturada através da câmera.
2. O método *GaussianBlur*, do *OpenCV*, é executado na imagem, a fim de se reduzir os ruídos. O filtro gaussiano é um filtro passa-baixa que remove os componentes de alta frequência da imagem.

3. Os parâmetros da imagem são convertidos de RGB para HSV, baseando-se em [Araujo, Mendonça e Freire \(2011\)](#). O *OpenCV* possui um método (*cvtColor*) que realiza essa conversão de forma rápida.
4. Todos os *pixels* que estão na faixa apresentada na [Tabela 5](#) são alterados para a cor branca. Os outros são alterados para a cor preta, de forma a se realizar a segmentação binária (ver [subseção 2.3.1](#)).
5. A imagem é dilatada, através do método *dilate* do *OpenCV*, para que as cores brancas fiquem melhores para identificação.
6. Utilizando o método *findContours*, do *OpenCV*, os centros dos agrupamentos de *pixels* brancos são identificados através do contorno de suas formas. Esse método retorna uma lista dos contornos encontrados e seus parâmetros.
7. O maior contorno da lista retornada pelo item anterior é utilizado pelo algoritmo, que desenha um círculo verde ao seu redor.
8. O centro do contorno é armazenado, sendo representado por um par ordenado (*x* e *y*) na matriz da imagem.

As coordenadas *x* e *y* obtidas na execução do passo a passo anterior são utilizadas pelo algoritmo do sistema de guiagem para se estimar a localização do robô.

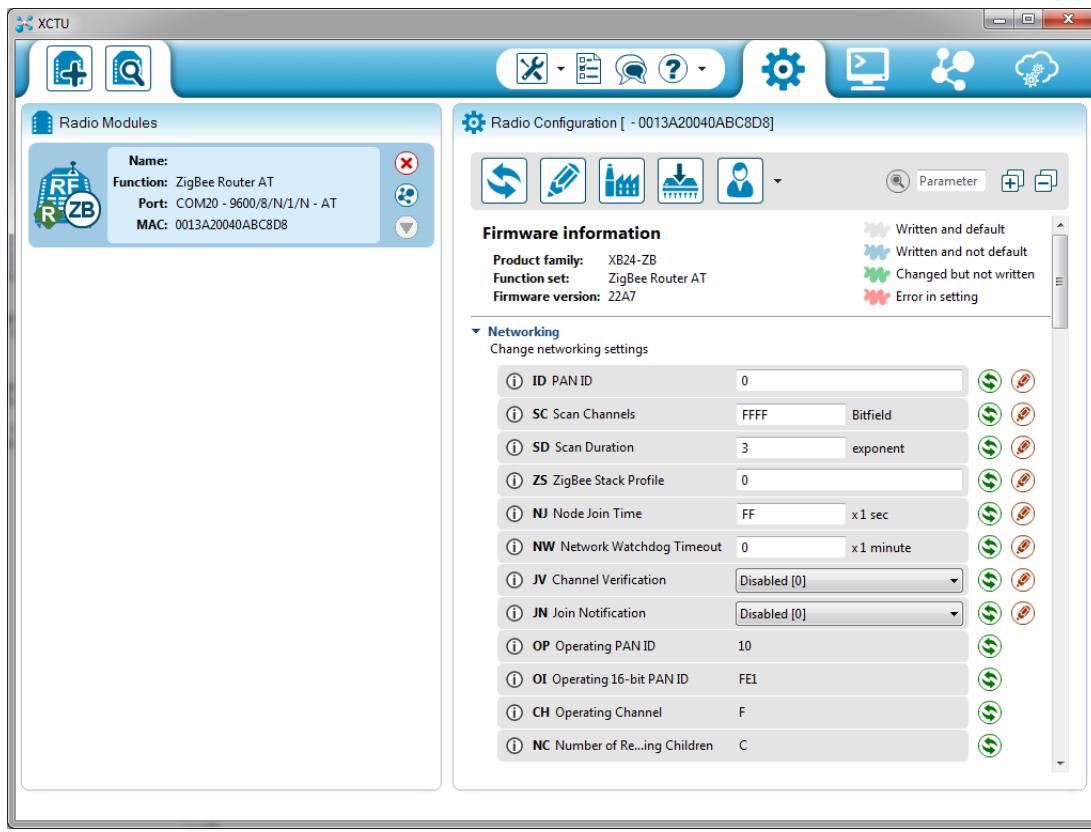
## 4.5 Comunicações sem fio

Para realizar a comunicação sem fio entre o robô e a estação de solo, foram utilizados dois módulos *XBee PRO S1*.

Dessa forma, um dos módulos deve ser configurado para trabalhar como *coordinator* (coordenador da rede, instalado na estação de solo), enquanto o outro deve ser configurado como *router* (embarcado no robô).

Segundo [Thomsen \(2014\)](#), a configuração dos módulos deve ser realizada individualmente através do programa *XCTU* ([Figura 45](#)). Para isso, deve-se baixar e instalar o programa, além de conectar o módulo *XBee* através do circuito *XBee Explorer* ao computador utilizando de uma porta USB.

Figura 45 – XCTU



Fonte: Thomsen (2014)

Através da [Figura 45](#), é possível verificar que o módulo exemplo está configurado como *router*. Para alterá-lo para *coordenador*, é necessário clicar no botão *update firmware* e alterar sua configuração.

Também é necessário realizar a configuração de outros três campos:

- *ID PAN ID*: identificação para a rede. O nome escolhido foi "FF";
- *DH Destination Address High*: primeira parte do endereço do outro módulo da rede (no caso, o módulo *router*). Essa informação se encontra fixada no módulo *XBee*;
- *DL Destination Address Low*: última parte do endereço do outro módulo da rede (no caso, o módulo *router*). Essa informação se encontra fixada no módulo *XBee*.

Para a configuração do módulo *router*, foram realizados os mesmos ajustes anteriores, sendo a única alteração que os parâmetros DH e DL são os fixados no módulo *coordenador*.

Para o algoritmo em *Python* executado no computador da estação de solo, foi utilizada a biblioteca *XBeeDevice*, visto que ela já apresenta as funções necessárias para

o funcionamento do sistema. No *Arduino* embarcado, a comunicação se dá através da conexão serial 0.

## 4.6 Sistema de Guiagem

O Sistema de Guiagem do projeto consiste da junção do algoritmo de processamento de imagem com a comunicação da estação de solo com o robô. Assim, toma-se decisões para a navegação de acordo com a posição espacial do robô, de maneira a guiá-lo para realizar o seguimento de trajetória.

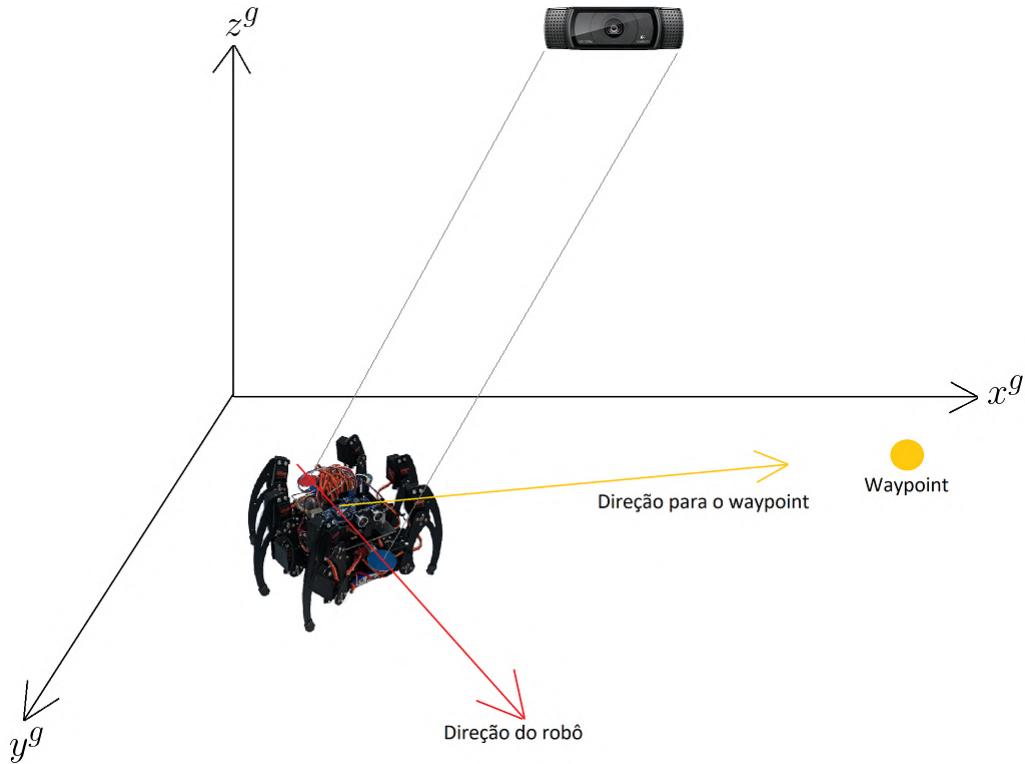
A fim de se obter uma visão geral do ambiente de deslocamento do hexápode, a câmera foi posicionada no teto da sala, na região central, e com a sua lente voltada para o solo. Dessa maneira, tem-se visão para identificação do posicionamento do robô na totalidade do ambiente destinado a sua movimentação.

O sistema para definição de *waypoints* e identificação da localização do robô é baseado em coordenadas globais  $x^g$ ,  $y^g$  e  $z^g$ .

Conforme especificado na seção anterior, as cores vermelha e azul foram escolhidas para o algoritmo realizar a identificação da traseira e da frente do robô, respectivamente. Tem-se no ambiente uma disposição conforme no diagrama apresentado na [Figura 46](#). Como a câmera está posicionada imediatamente acima do ambiente (numa altura de 282 cm, representada pelo eixo  $z^g$ ), ela gera uma visão do solo em duas dimensões: horizontal (representada pelo eixo  $x^g$ ) e vertical (representada pelo eixo  $y^g$ ). Por isso, para o sistema de definição de pontos, o eixo  $z^g$  não é considerado. A distância coberta pela visão da câmera é de 277 cm no eixo  $x^g$  e 209 cm no eixo  $y^g$ .

As coordenadas geradas pelo algoritmo apresentado na seção anterior estão no referencial da resolução da imagem gerada pela câmera. Ou seja, o maior ponto do eixo  $x$  é 1024 e do eixo  $y$  é 768. Entretanto, o algoritmo automaticamente converte esses valores para o referencial da distância real em centímetros, de forma a se trabalhar com valores compatíveis aos da realidade. Dessa maneira, a maior coordenada passa a ser 277 no eixo  $x^g$  e 209 no eixo  $y^g$ .

Figura 46 – Representação gráfica da visão obtida pelo algoritmo de guiagem



Fonte: Autoria própria

Sendo assim, é possível obter as coordenadas  $x^g$  e  $y^g$ , em centímetros, do círculo azul e do círculo vermelho. Através desses dados, é calculado o centro no qual o robô está posicionado espacialmente. Esses cálculos para a coordenada do eixo  $x^g$  estão representados pela [Equação 4.1](#).

$$x_{robô} = \frac{x_{azul} + x_{vermelho}}{2} \quad (4.1)$$

Na [Equação 4.2](#) é representado o cálculo análogo para a coordenada do eixo  $y^g$ .

$$y_{robô} = \frac{y_{azul} + y_{vermelho}}{2} \quad (4.2)$$

Os dados obtidos através das coordenadas das cores também possibilitam o cálculo da orientação que representa a direção em que a frente do robô está apontando, demonstrado na [Equação 4.3](#). Na [Figura 46](#), essa orientação está representada pela seta vermelha. Para esse tipo de problema, é utilizado arcotangente2 pois, segundo [Santos et al. \(2011\)](#), o cálculo da tangente nessa função é realizado considerando-se os quatro quadrantes (ou

seja, todas as possíveis orientações em que o robô pode se encontrar são consideradas).

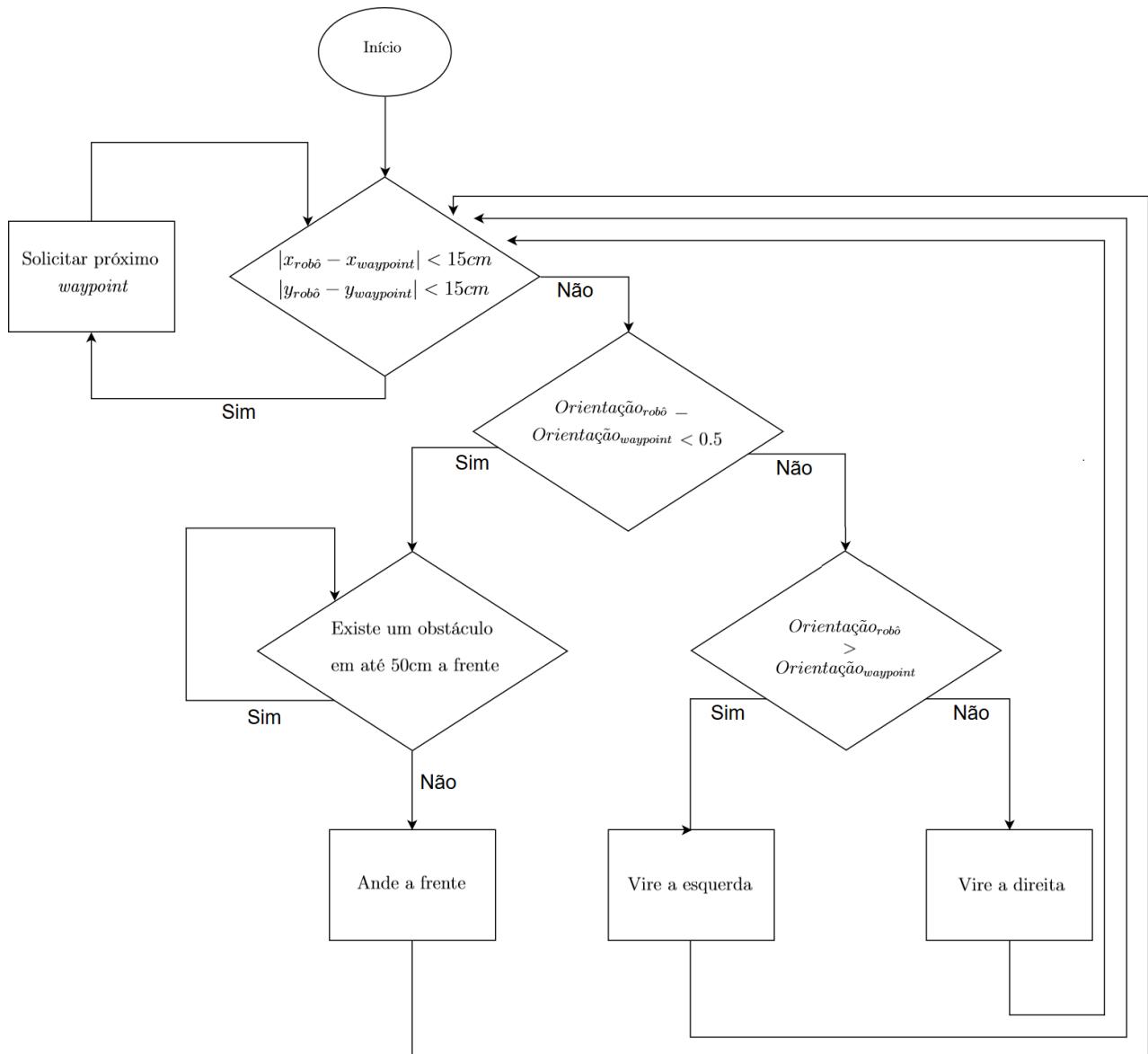
$$Orientação_{robô} = \text{atan}_2(y_{azul} - y_{vermelho}, x_{azul} - x_{vermelho}) \quad (4.3)$$

O próximo cálculo realizado é análogo ao anterior e objetiva-se encontrar a orientação entre a posição central do robô e o *waypoint* que se deseja atingir. Essa orientação está representada na [Figura 46](#) através da seta amarela e na [Equação 4.4](#).

$$Orientação_{waypoint} = \text{atan}_2(y_{waypoint} - y_{robô}, x_{waypoint} - x_{robô}) \quad (4.4)$$

Com base nos resultados das equações, o sistema pode tomar a decisão a ser realizada em cada passo guiado no caminho. Esses dados são analisados pelo algoritmo que define cada decisão a ser efetivada, de acordo com as condições obtidas. O fluxograma referente a esse algoritmo está disponível na [Figura 47](#). Ele é executado em repetição até que todos os *waypoints* definidos no início da execução do sistema tenham sido atingidos.

Figura 47 – Fluxograma do algoritmo de locomoção



Fonte: Autoria própria

Os valores das tolerâncias de coordenadas e de orientações foram definidos arbitrariamente, entretanto apresentaram bons resultados (que serão discutidos no próximo capítulo), não havendo, assim, a necessidade de se alterá-los no escopo desse projeto.

#### 4.6.1 Sistema de Comunicação: Modos para troca de mensagens

A partir das definições apresentadas anteriormente, foram desenvolvidas dois casos diferentes para a comunicação de dados entre a estação de solo e o robô, denominadas de "troca de mensagens em duas vias" e "troca de mensagens em via única".

No primeiro método desenvolvido, chamado de "troca de mensagens em duas vias", o computador processa a imagem obtida pela câmera e envia os dados de coordenadas e orientações para o robô apenas quando este realiza a solicitação. Dessa maneira, a frequência de atualização de dados depende da finalização da movimentação pelo robô, que só envia um novo pedido de mensagem de localização instantes antes de concluir a movimentação já atribuída aos servomotores (definido empiricamente), a fim de se tornar a locomoção o mais fluída possível. Com isso, obtem-se uma frequência de geração e envio de dados de aproximadamente 0,75 Hz. A [Tabela 6](#) apresenta as mensagens que foram definidas para a comunicação entre as unidades do sistema.

Tabela 6 – Mensagens padrão de comunicação entre estação de solo e robô

Dispositivo	Mensagem	Função
Robô	Posicao	Solicitar para estação de solo o processamento de imagem daquele instante e o recebimento de novas coordenadas
Robô	Cheguei	Solicitar para a estação de solo que se defina o próximo <i>waypoint</i> , pois o anterior já foi atingido
Estação de solo	$x_{robô}, y_{robô}; Orientação_{robô};$ $x_{waypoint}, y_{waypoint}; Orientação_{waypoint}$	Envia para o robô a <i>string</i> que contém suas coordenadas, sua orientação, as coordenadas do <i>waypoint</i> e a orientação entre eles

Fonte: Autoria própria

Na segunda implementação, denominada de "troca de mensagens em via única", a estação de solo possui uma frequência fixa de envio de dados para o robô. Nesse caso, o hexápode não envia mensagens a serem interpretadas pelo algoritmo do computador, ele apenas recebe. A frequência de geração de dados de localização é de aproximadamente 28 Hz, e esses são enviados ao robô com a frequência de 1 Hz. A mensagem de comunicação do sistema (de apenas uma direção) é a mesma apresentada na última linha da [Tabela 6](#).

#### 4.6.2 Aquisição dos Resultados

Nos algoritmos implementados para os dois casos de comunicação são gerados relatórios em planilhas que armazenam todas as variáveis obtidas no sistema a cada instante de execução do processamento de imagem (*log*). O formato dos dados disponíveis nesse relatório está apresentado na [Tabela 7](#).

Tabela 7 – Formato da planilha relatório

Coluna	Informação
$x_{robô}$	Coordenada $x^g$ , em centímetros, na qual o robô se encontra naquele instante
$y_{robô}$	Coordenada $y^g$ , em centímetros, na qual o robô se encontra naquele instante
$Orientação_{robô}$	Orientação na qual o robô está direcionado
$x_{waypoint}$	Coordenada $x^g$ , em centímetros, na qual o <i>waypoint</i> alvo se encontra
$y_{waypoint}$	Coordenada $y^g$ , em centímetros, na qual o <i>waypoint</i> alvo se encontra
$Orientação_{waypoint}$	Orientação que o robô deve se aproximar para andar em linha reta até o <i>waypoint</i>
Tempo	Tempo de execução naquele instante, em segundos
<i>Waypoint</i> atingido	Indica se, naquele registro, o robô atingiu o <i>waypoint</i>

Fonte: Autoria própria

Esses dados são utilizados para geração de gráficos e extração de informações como velocidade média, tempo de movimentação e em quais instantes o robô considerou que atingiu os *waypoints*.

Além disso, foi utilizada a biblioteca *Pyplot* para a plotagem do gráfico dos caminhos percorridos pelo robô no momento em que ele finalizar a trajetória estabelecida. Esses gráficos estão representados no próximo capítulo, destinado à exibição dos resultados.

Os códigos implementados para o sistema embarcado no robô estão disponíveis no [Apêndice A](#) e os códigos implementados para a estação de solo no [Apêndice B](#). Os arquivos também estão disponíveis no repositório [https://github.com/tiagoruzzon/hexapod\\_robot](https://github.com/tiagoruzzon/hexapod_robot).

# 5 Resultados

Após o desenvolvimento do sistema proposto, conforme discutido no capítulo anterior, tem-se agora o capítulo destinado à exposição dos resultados dos estudos de caso realizados no projeto.

O primeiro passo para realização de testes é a inicialização do sistema. Para a partida do robô, é necessário se certificar de que a bateria possui carga suficiente para o torque dos servomotores. Caso a tensão esteja com nível maior ou igual a 7 V, então a carga é suficiente para alguns minutos de movimentação do robô.

Em seguida, deve-se carregar no *Arduino* o algoritmo embarcado que será utilizado, visto que, conforme capítulo anterior, foram desenvolvidas duas abordagens de comunicação. Para isso, deve utilizar um computador com a IDE do *Arduino* instalada e um cabo USB tipo B (conector da placa *Arduino Mega*) para tipo A (que é a conexão padrão para computadores).

Com relação à estação de solo, é necessário primeiramente realizar a conexão da câmera externa e do *XBee Explorer*, ambos via USB. O *software* de processamento de imagem deve ser executado através da IDE do *Python*. Entretanto, antes de se iniciar o programa, é preciso definir no próprio código os *waypoints* desejados, definido-se suas coordenadas  $x^g$  e  $y^g$ .

As instruções completas para inicialização do sistema estão disponíveis no [Apêndice A](#).

## 5.1 Definição dos Estudos de Caso

Foram definidos quatro diferentes testes para comprovar o funcionamento do sistema desenvolvido. Os testes se diferenciam pela quantidade de *waypoints* que o robô deve atingir, além de também serem utilizadas duas abordagens de comunicação com a estação de solo.

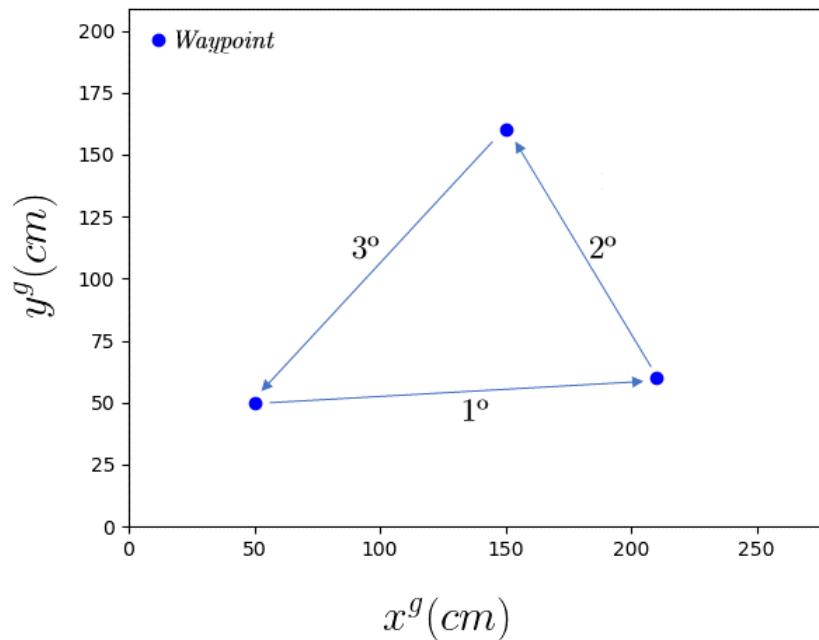
A lista disponível abaixo descreve os detalhes de cada estudo de caso, denominados de 1A, 2A, 1B e 2B.

- **Estudo de Caso 1A:** utilizando da abordagem de troca de mensagens em duas vias (definida na [subseção 4.6.1](#)), foi realizado um teste com três *waypoints*, representados na [Equação 5.1](#) e [Figura 48](#).
- **Estudo de Caso 2A:** utilizando da abordagem de troca de mensagens em via única

(definida na [subseção 4.6.1](#)), foi realizado um teste com três *waypoints*, representados na [Equação 5.1](#) e [Figura 48](#). Além disso, foi inserido um obstáculo à frente do robô, de forma a ele necessitar parar sua movimentação até que o objeto seja removido.

$$A^g = \begin{bmatrix} 210 & 60 \\ 150 & 160 \\ 50 & 50 \end{bmatrix} \text{ cm} \quad (5.1)$$

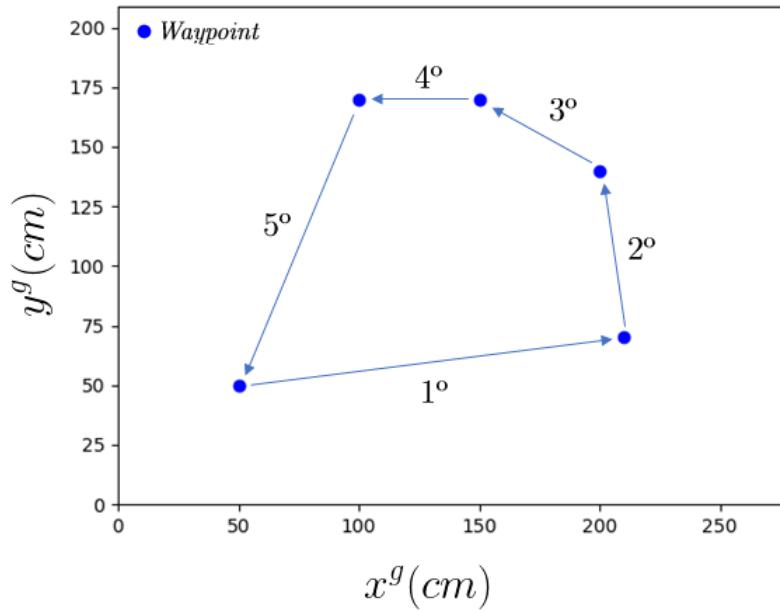
Figura 48 – Trajetória de  $A^g$



Fonte: Autoria própria

- **Estudo de Caso 1B:** utilizando da abordagem de troca de mensagens em duas vias (definida na [subseção 4.6.1](#)), foi realizado um teste com cinco *waypoints*, representados na [Equação 5.2](#) e [Figura 49](#).
- **Estudo de Caso 2B:** utilizando da abordagem de troca de mensagens em via única (definida na [subseção 4.6.1](#)), foi realizado um teste com cinco *waypoints*, representados na [Equação 5.2](#) e [Figura 49](#).

$$B^g = \begin{bmatrix} 210 & 70 \\ 200 & 140 \\ 150 & 170 \\ 100 & 170 \\ 50 & 50 \end{bmatrix} \text{ cm} \quad (5.2)$$

Figura 49 – Trajetória de  $B^g$ 

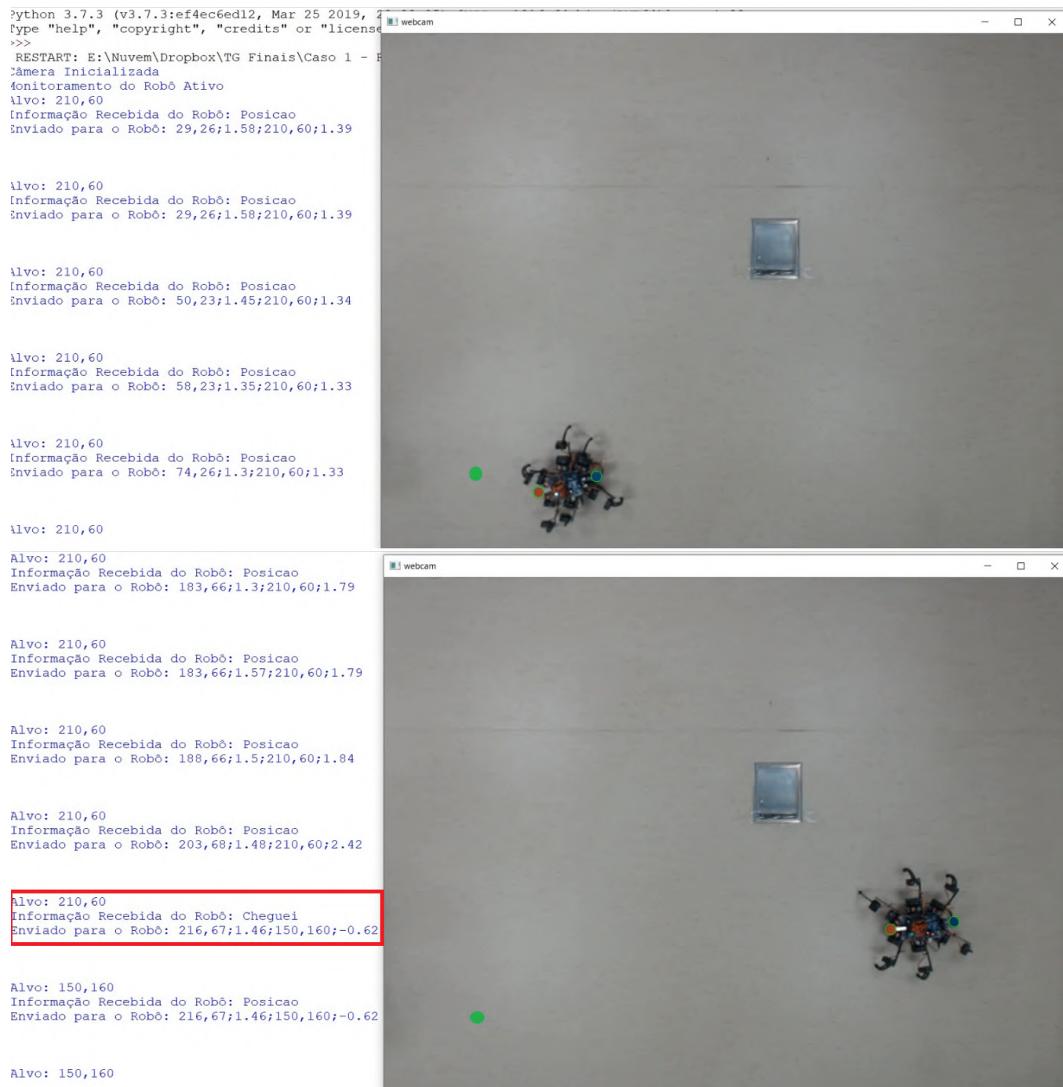
Fonte: Autoria própria

Todas as execuções foram realizadas com o robô partindo de coordenadas próximas a  $x^g = 50$  cm e  $y^g = 50$  cm, que são as mesmas coordenadas do último *waypoint* de  $A^g$  e  $B^g$ . Conforme especificado anteriormente, eixo  $z^g$  foi omitido.

## 5.2 Estudo de Caso 1A

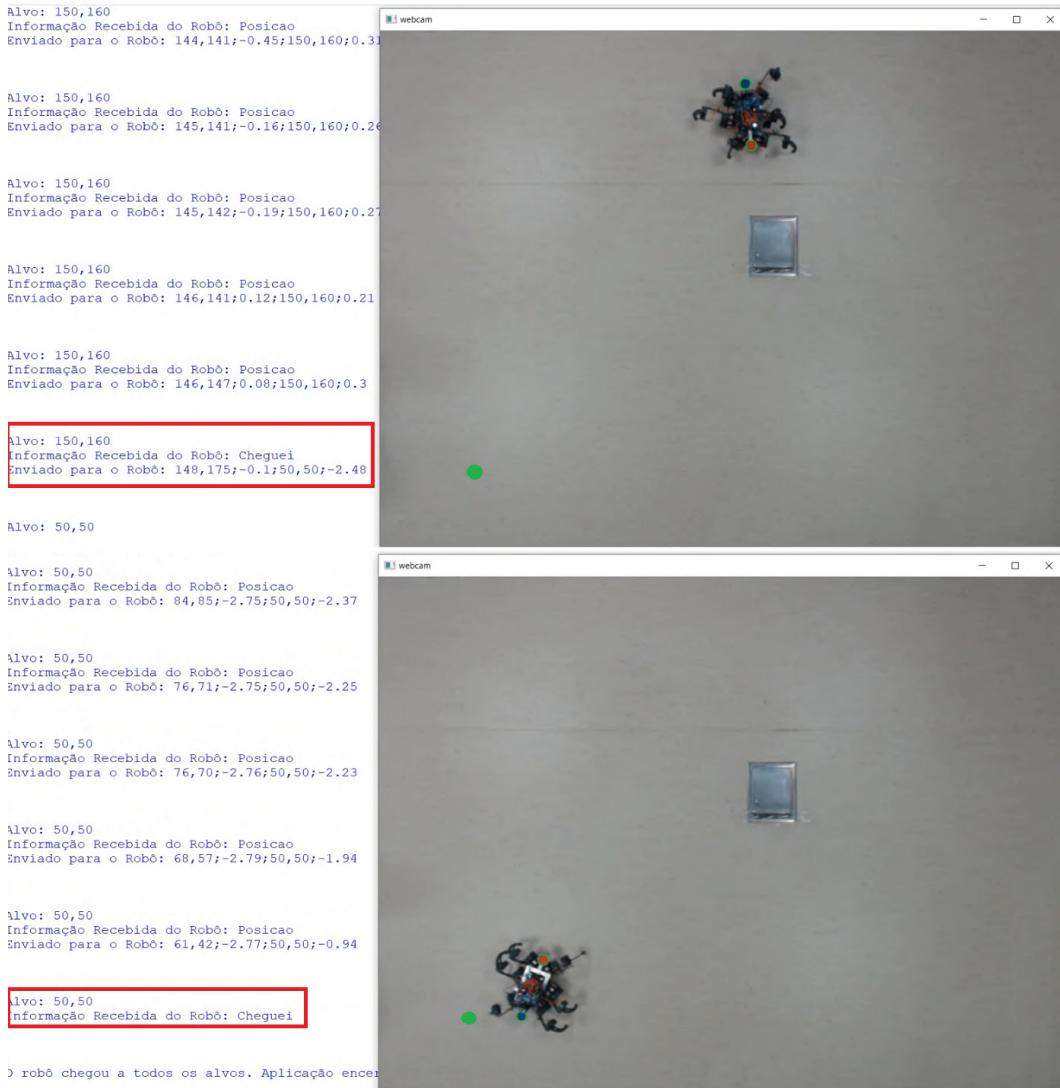
No estudo de caso 1A, o sistema executou o algoritmo de comunicação em duas vias e o robô necessitava passar por três *waypoints* de  $A^g$ . Foram armazenadas imagens dessa execução, que estão disponíveis na Figura 50 e Figura 51. Nas figuras, têm-se uma imagem do robô logo após sair de seu ponto inicial e as outras imagens correspondem aos momentos que o robô atinge os *waypoints* subsequentes. As mensagens de comunicação são exibidas na porção esquerda da tela, e os avisos de chegada estão destacados em vermelho. O ponto verde indica a coordenada do ponto inicial do robô.

Figura 50 – Screenshots do estudo de caso 1A (parte 1/2)



Fonte: Autoria própria

Figura 51 – Screenshots do estudo de caso 1A (parte 2/2)



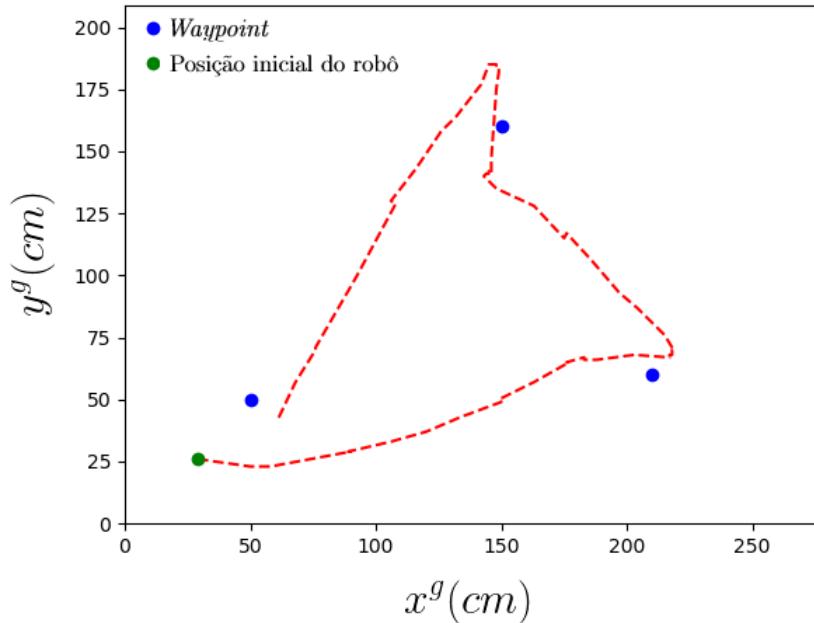
Fonte: Autoria própria

É possível observar que o hexápode percorreu o trajeto previamente definido conforme o esperado, atingindo todas as coordenadas desejadas.

As mensagens enviadas tanto pelo robô quanto pela estação de solo foram recebidas corretamente, assim como a identificação do hexápode de quando ele atinge uma coordenada alvo, enviando a mensagem "Cheguei" (destacadas em vermelho) e solicitando ao computador o próximo alvo.

Ao final da execução, o algoritmo da estação de solo gera automaticamente o gráfico da trajetória, em coordenadas  $x^g$  e  $y^g$ , que está disponível na [Figura 52](#).

Figura 52 – Trajetória executada pelo robô no estudo de caso 1A



Fonte: Autoria própria

O ponto verde indica a posição inicial do robô. O tracejado vermelho é o caminho que o robô percorreu durante o experimento, enquanto os pontos azuis são os *waypoints* estabelecidos. Através do gráfico, pode-se notar que o robô atingiu todas as áreas de tolerância das coordenadas conforme o especificado.

Os erros relativos e absolutos de chegada nos *waypoints* são apresentados na Tabela 8.

Tabela 8 – Tabela de erros nas chegadas para o estudo de caso 1A

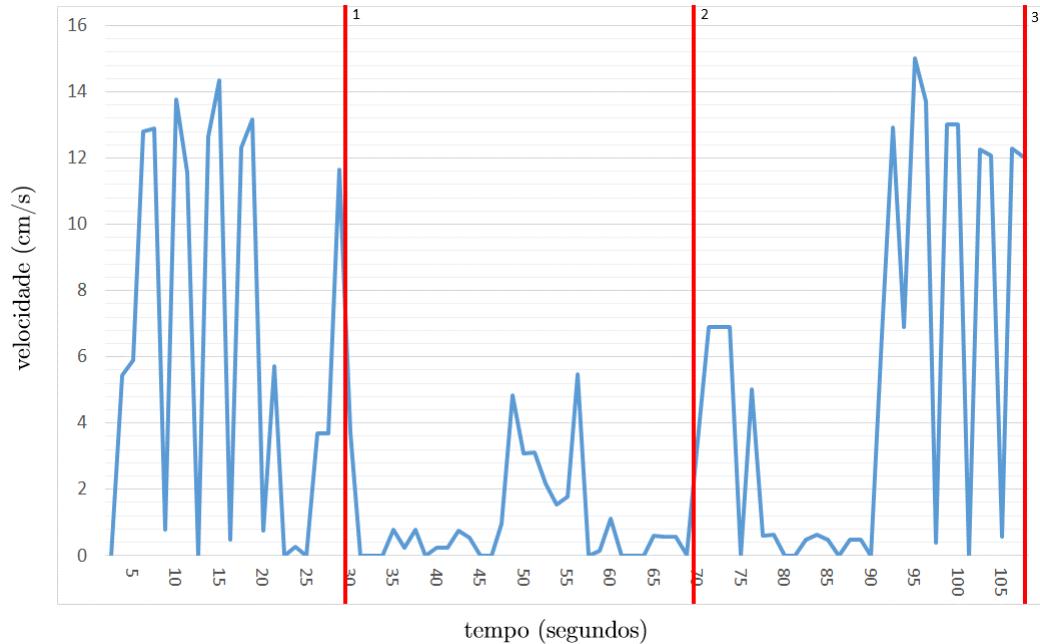
$x_{robô}$	$x_{waypoint}$	$E_{abs}(x^g)$ (cm)	$E_{rel}(x^g)$	$y_{robô}$	$y_{waypoint}$	$E_{abs}(y^g)$ (cm)	$E_{rel}(y^g)$
203	210	7	3,3%	68	60	8	13,3%
146	150	4	2,7%	147	160	13	8,1%
61	50	11	22%	42	50	8	16%

Fonte: Autoria própria

Todos os erros obtidos estão dentro da margem de tolerância predefinida anteriormente, de 15 cm.

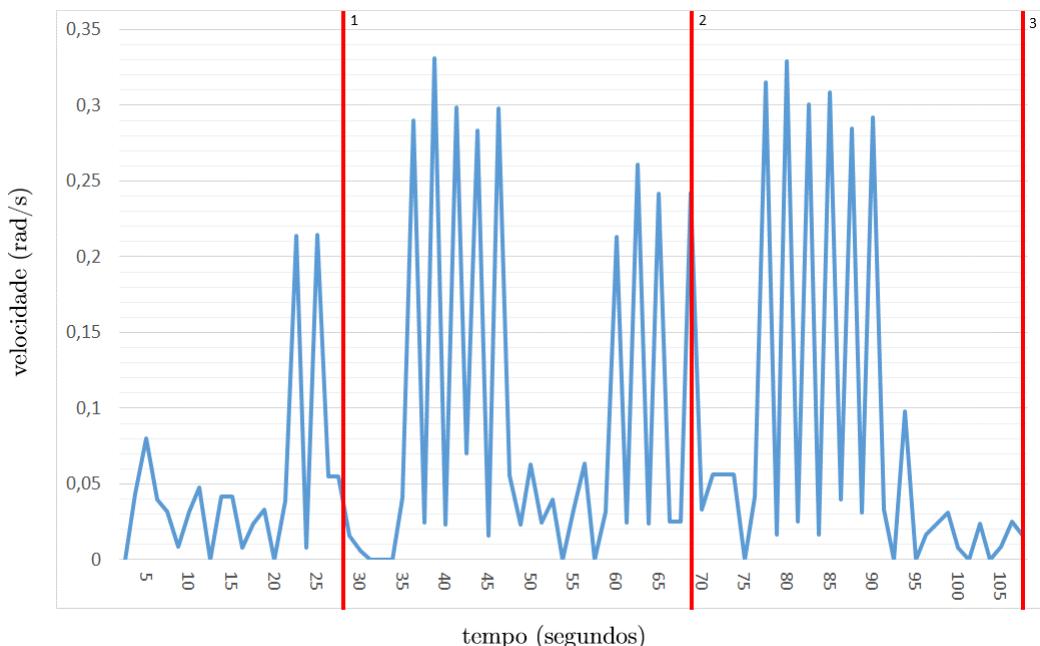
Com os dados disponíveis no relatório de execução, foram gerados gráficos da variação de velocidade de translação e da velocidade angular média, que estão disponíveis na Figura 53 e Figura 54, respectivamente. As linhas vermelhas indicam os instantes de chegadas ao *waypoints*.

Figura 53 – Velocidade de translação para o estudo de caso 1A



Fonte: Autoria própria

Figura 54 – Velocidade angular para o estudo de caso 1A



Fonte: Autoria própria

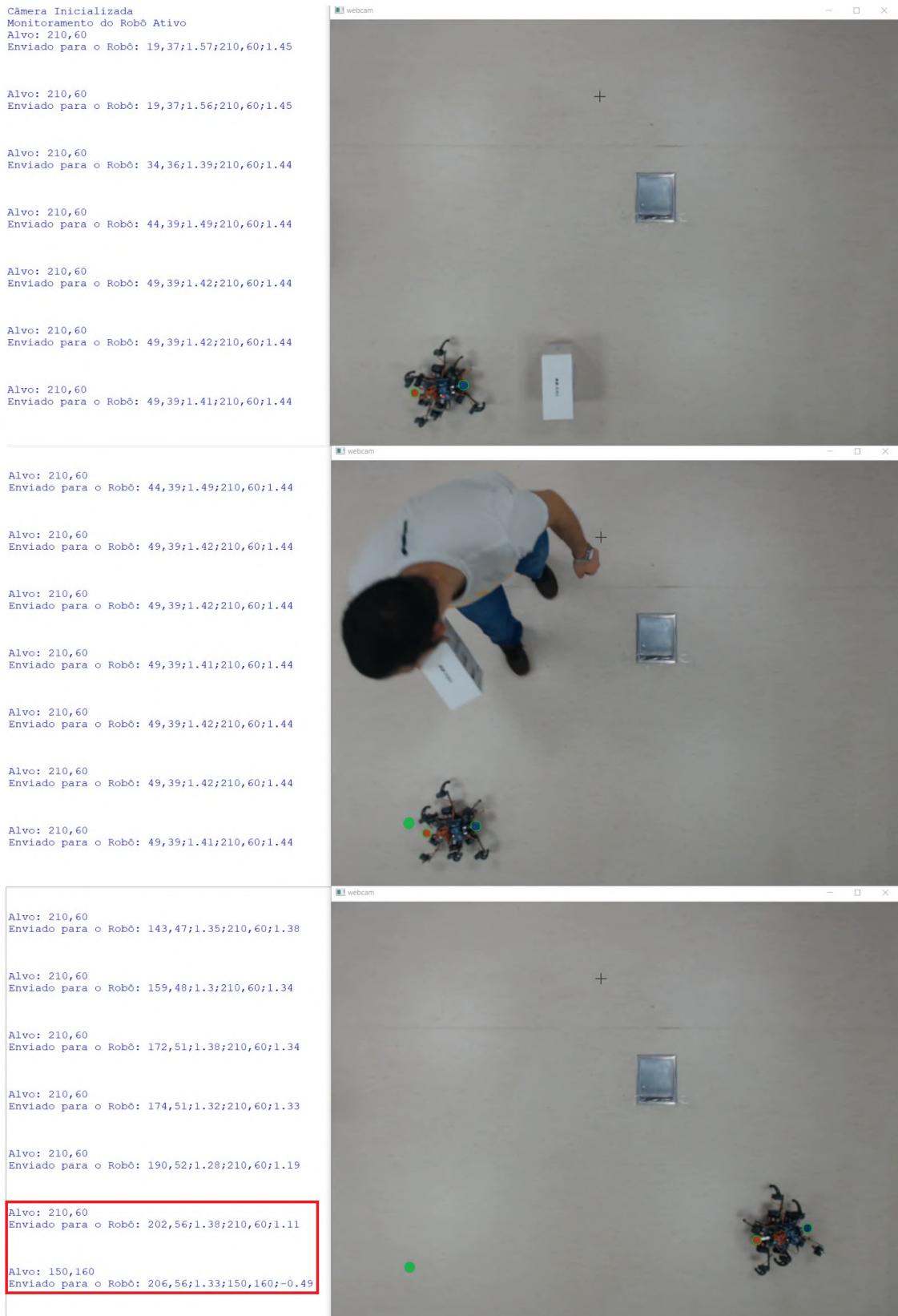
Pode-se observar que nos momentos em que a velocidade de translação é próxima a zero, a velocidade angular tem instantes de pico. Esse comportamento é o esperado, visto

que o robô apenas se movimenta em translação em linha reta e realiza seus movimentos de rotação apenas estando parado.

### 5.3 Estudo de Caso 2A

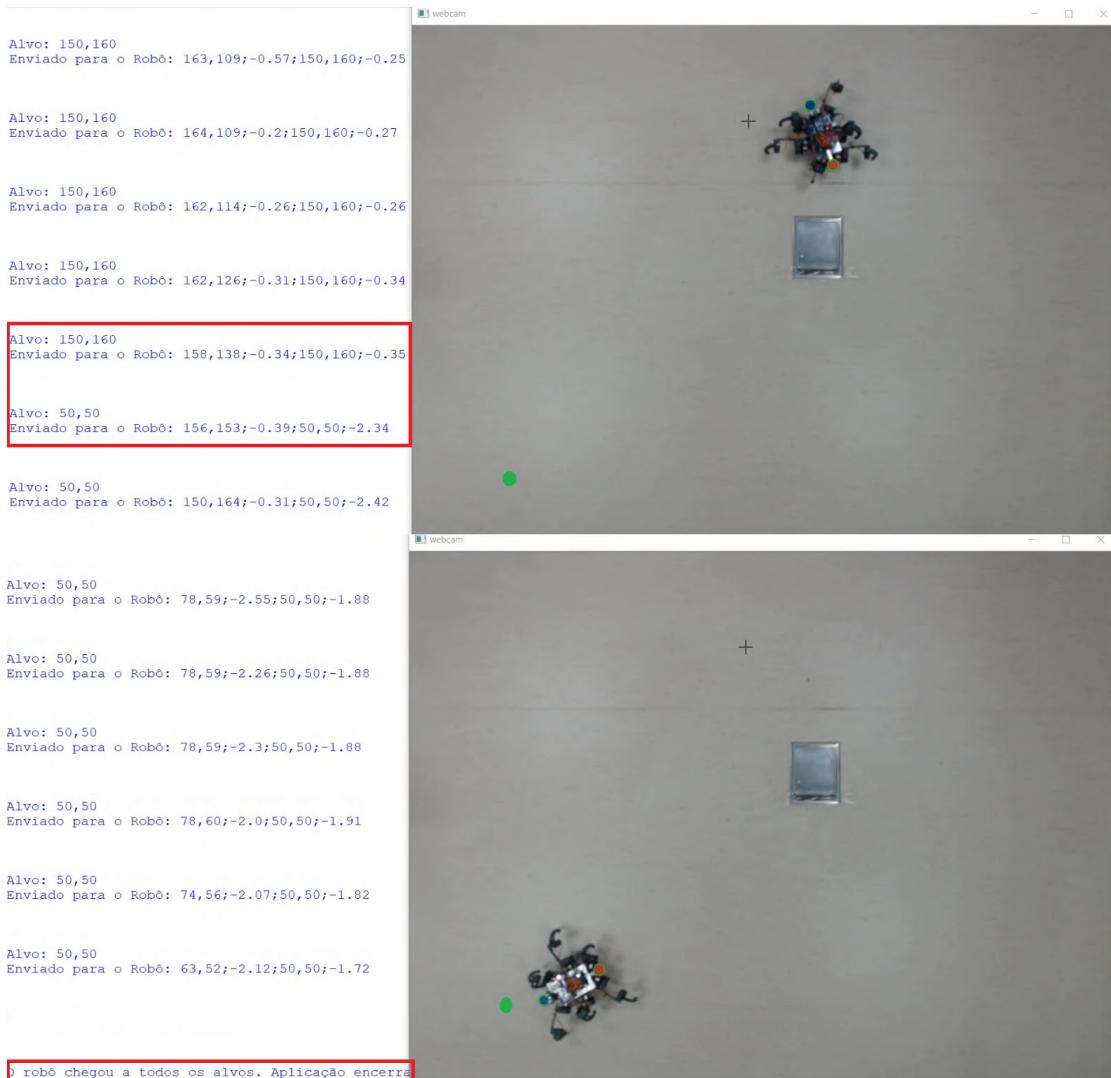
No estudo de caso 2A, o sistema estava executando a abordagem de troca de mensagens em via única para atingir três *waypoints* de  $A^g$ . Além disso, esse estudo de caso contou com o teste da utilização do sensor ultrassônico. Para isso, um obstáculo foi colocado na frente do robô, de forma a ser detectado alguns passos a após sua posição inicial. Os *screenshots* da execução estão disponíveis em [Figura 55](#) e [Figura 56](#). O ponto verde indica a coordenada do ponto inicial do robô. As mensagens destacadas em vermelho indicam o momento em que o sistema identificou que o robô atingiu o *waypoint* e, dessa maneira, a próxima mensagem enviada faz referência ao próximo alvo do caminho. Quando não há mais alvos, a execução é finalizada.

Figura 55 – Screenshots do estudo de caso 2A (parte 1/2)



Fonte: Autoria própria

Figura 56 – Screenshots do estudo de caso 2A (parte 2/2)

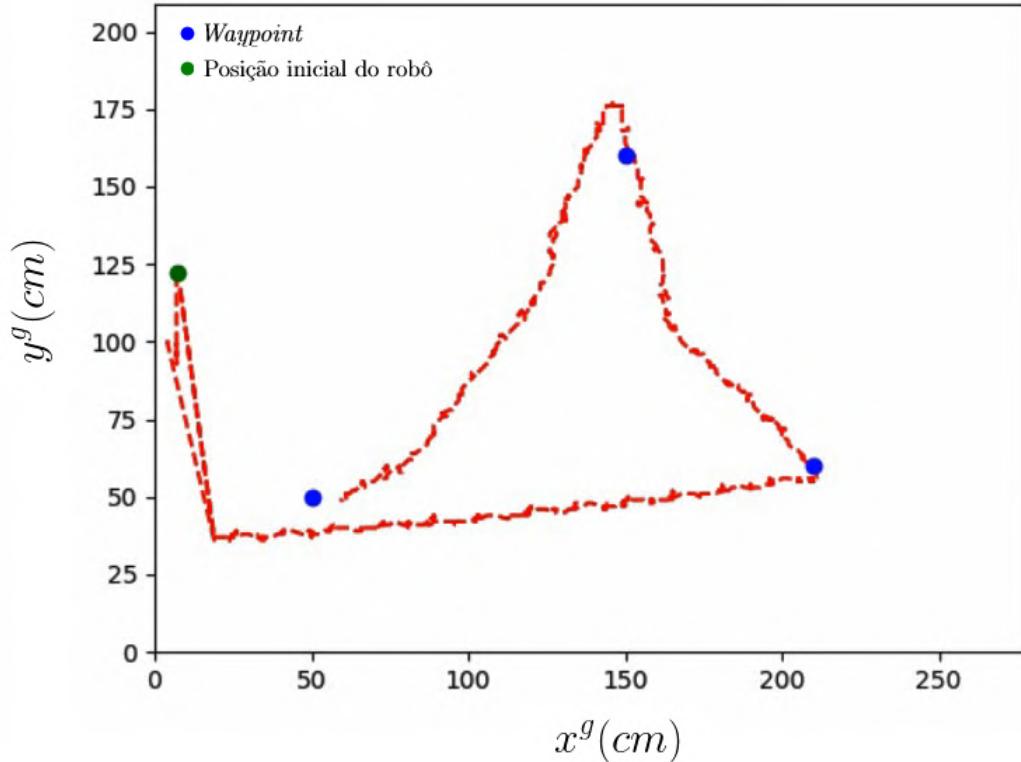


Fonte: Autoria própria

Conforme o esperado, assim que o objeto foi removido da frente do robô, o mesmo retomou imediatamente sua movimentação e atingiu a todos os *waypoints* estabelecidos. Como pode ser visto no painel de mensagens, nessa abordagem não há envio de mensagens do robô para a estação de solo.

A [Figura 57](#) apresenta o gráfico plotado pelo sistema.

Figura 57 – Trajetória executada pelo robô no estudo de caso 2A



Fonte: Autoria própria

O primeiro ponto a ser observado aqui é a inconsistência de informações no início da trajetória. Isso indica uma possibilidade de melhoria para o sistema em trabalhos futuros, visto que o processamento de imagem reconheceu que o robô iniciou sua movimentação e esteve em locais do ambiente equivocados. Esse problema pode ter ocorrido por causa da presença de outros seres na imagem, como o obstáculo e a pessoa que o removeu, assim como também divergências de iluminação. Outro ponto interessante é que a trajetória não apresentou um tracejado perfeito. Isso pode ser explicado pelo fato de que, nessa abordagem de execução, são geradas muito mais informações de localização do robô, visto que a frequência do processamento de imagem é maior. Dessa forma, são detectadas também, com mais precisão, as imperfeições na caminhada do robô.

Apesar das considerações, o gráfico apresentado demonstra uma maior precisão na movimentação, com um menor número de curvas necessárias para se atingir os alvos e uma melhor precisão sobre os *waypoints*, que pode ser evidenciada através da Tabela 9, que apresenta os erros relativos e absolutos de chegada nos *waypoints*.

Tabela 9 – Tabela de erros nas chegadas para o estudo de caso 2A

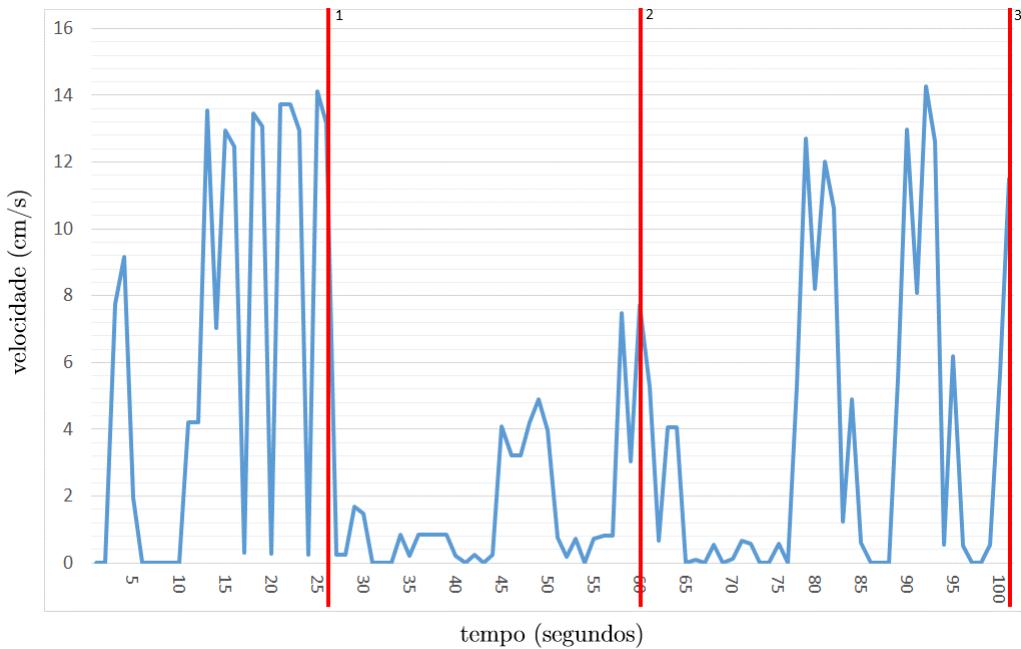
$x_{robô}$	$x_{waypoint}$	$E_{abs}(x^g)$ (cm)	$E_{rel}(x^g)$	$y_{robô}$	$y_{waypoint}$	$E_{abs}(y^g)$ (cm)	$E_{rel}(y^g)$
202	210	8	3,8%	56	60	4	6,7%
155	150	5	3,3%	151	160	9	5,6%
59	50	9	18%	49	50	1	2%

Fonte: Autoria própria

Através da tabela, é possível notar que todos os pontos foram atingidos conforme a tolerância definida (de 15 cm para cada eixo coordenado).

Os gráficos de velocidade de translação e velocidade angular estão disponíveis na Figura 58 e Figura 59, respectivamente. As linhas vermelhas indicam os instantes de chegadas ao *waypoints*.

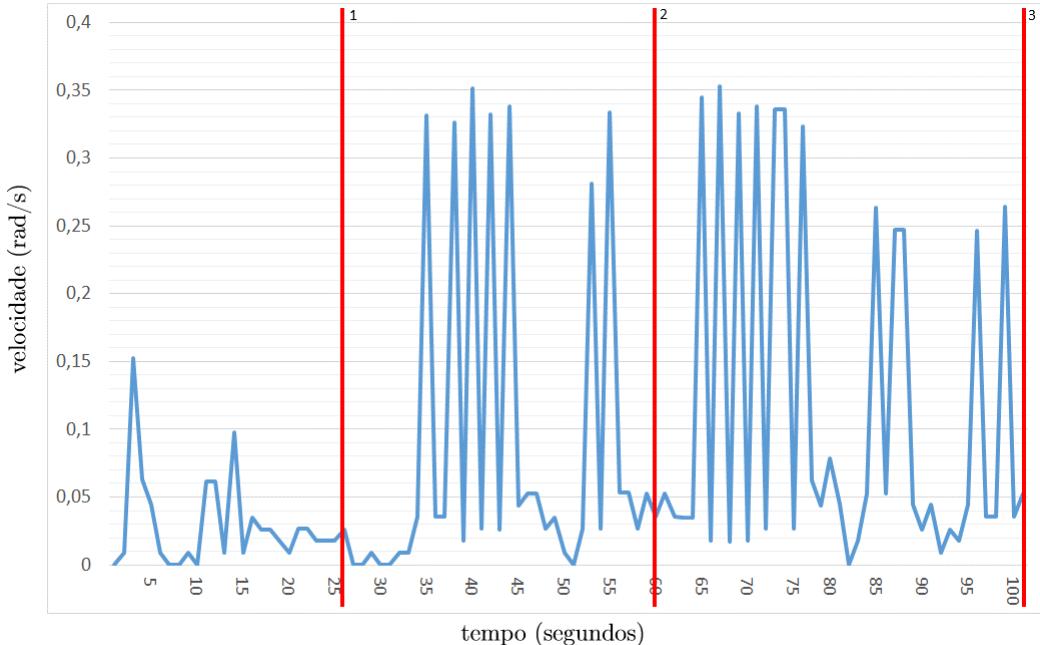
Figura 58 – Velocidade de translação para o estudo de caso 2A



Fonte: Autoria própria

Foi possível observar no gráfico de velocidade de translação, entre 5 e 10 segundos, os momentos em que o robô se manteve ocioso, aguardando até que o obstáculo a sua frente seja removido.

Figura 59 – Velocidade angular para o estudo de caso 2A



Fonte: Autoria própria

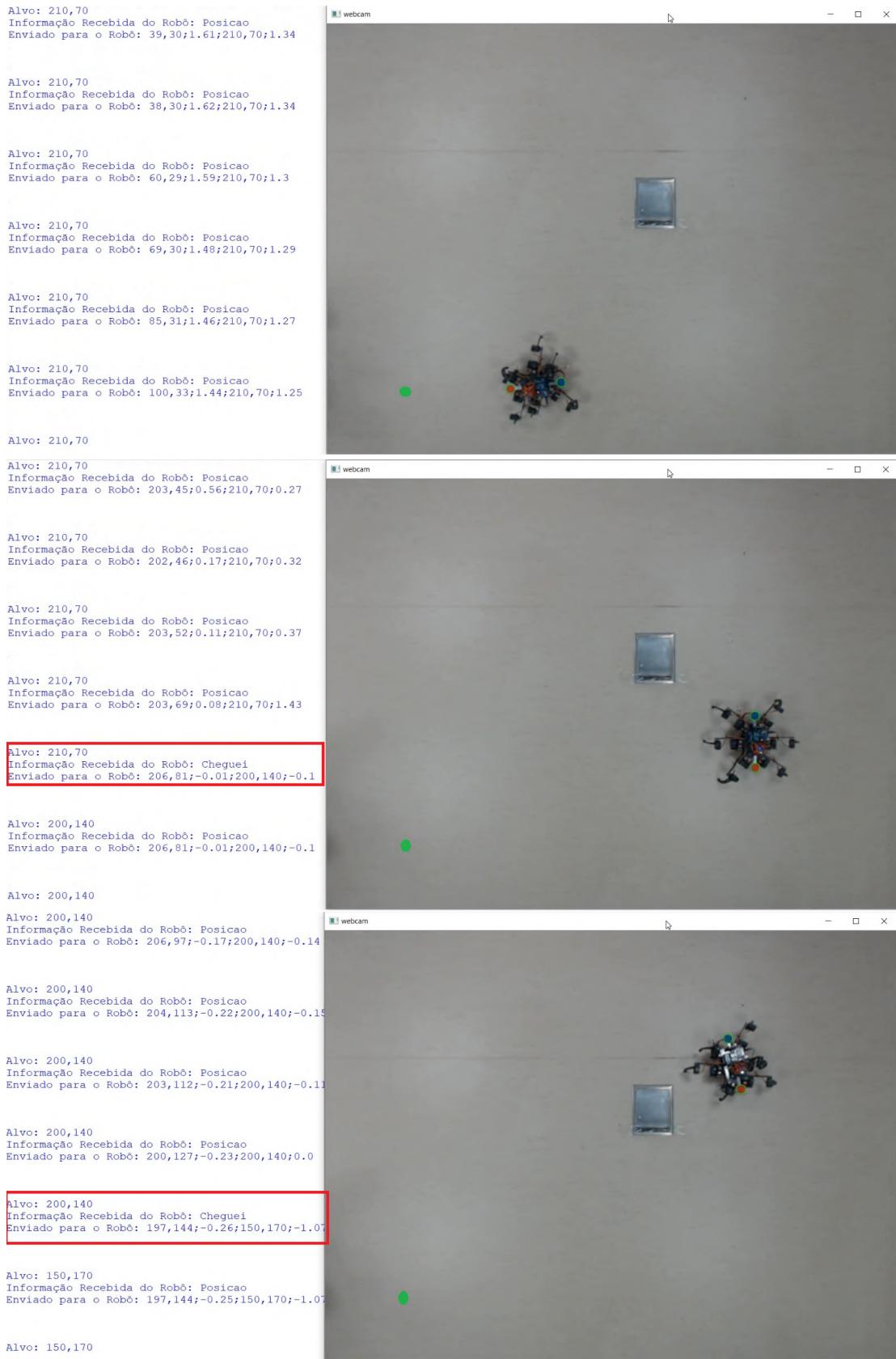
Assim como para o estudo de caso anterior, também foi possível visualizar picos de velocidade angular em momentos nos quais a velocidade de translação está mais baixa.

Realizando uma comparação direta entre os resultados obtidos no estudo de caso 1A e 2A (que foram realizados com o mesmo grupo de *waypoints*), observa-se que a abordagem 2 finalizou o caminho em um tempo ligeiramente inferior. Em geral, todos os *waypoints* foram atingidos em menos tempo na abordagem 2 (mesmo considerando o tempo em que o robô permaneceu parado para o teste do sensor ultrassônico). Além disso, a abordagem 1 apresentou maiores erros nas chegadas às coordenadas estipuladas. Por sua vez, a trajetória estabelecida na abordagem 2 foi mais consistente, visto que foram realizados menos movimentos de rotação. Dessa maneira, é possível considerar que a implementação de comunicação em via única apresentou melhor desempenho ao executar o grupo de *waypoints*  $A^g$ .

## 5.4 Estudo de Caso 1B

Para o estudo de caso 1B, foi executada a abordagem de que o robô necessita solicitar à estação de solo sua localização e atingir cinco *waypoints* de  $B^g$ . Os *screenshots* da execução foram registrados e estão disponíveis na [Figura 60](#) e [Figura 61](#). Novamente, a marcação verde indica o ponto inicial do robô. As marcações vermelhas demonstram as mensagens enviadas quando o robô atinge os alvos do caminho.

Figura 60 – Screenshots do estudo de caso 1B (parte 1/2)



Fonte: Autoria própria

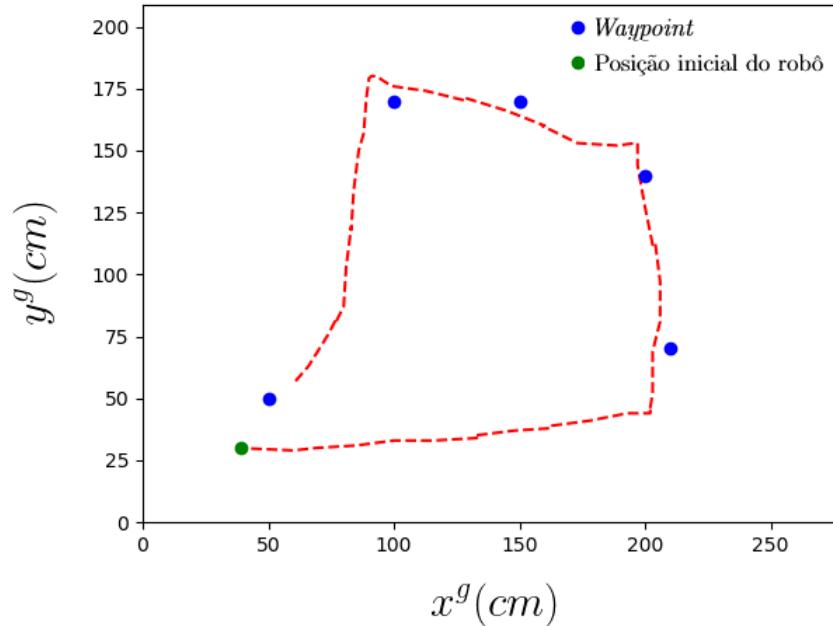
Figura 61 – Screenshots do estudo de caso 1B (parte 2/2)



Fonte: Autoria própria

A visão do caminho do robô está disponível no gráfico da Figura 62.

Figura 62 – Trajetória executada pelo robô no estudo de caso 1B



Fonte: Autoria própria

Esse gráfico demonstra que o robô apresentou uma movimentação consistente, atingindo a todos os *waypoints* com uma boa margem de erro.

Os erros relativos e absolutos de chegada nos *waypoints* são apresentados na Tabela 10.

Tabela 10 – Tabela de erros nas chegadas para o estudo de caso 1B

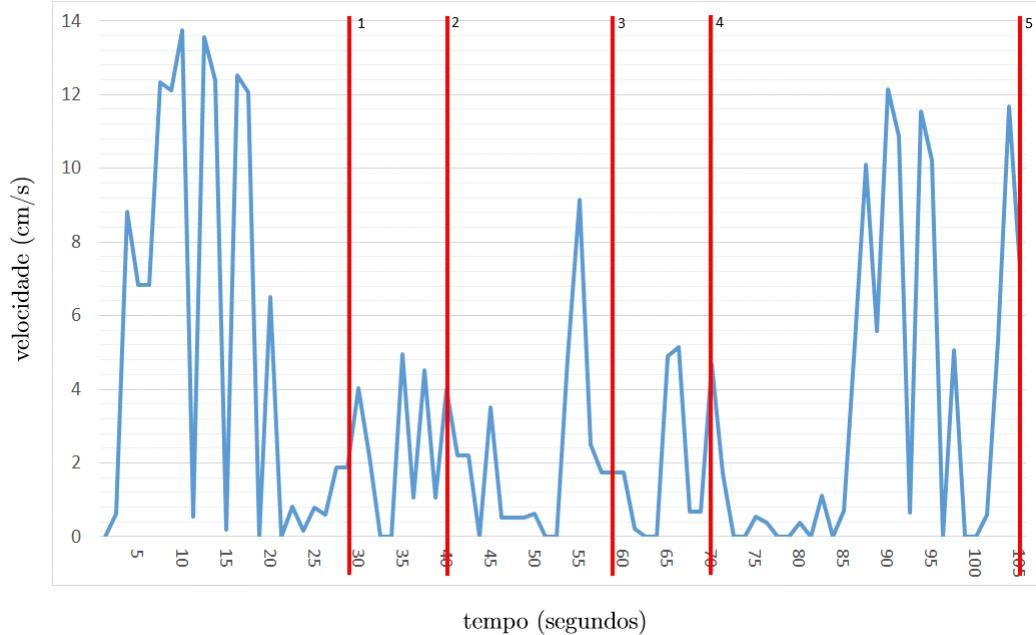
$x_{robô}$	$x_{waypoint}$	$E_{abs}(x^g)$ (cm)	$E_{rel}(x^g)$	$y_{robô}$	$y_{waypoint}$	$E_{abs}(y^g)$ (cm)	$E_{rel}(y^g)$
203	210	7	3,3%	69	70	1	1,4%
200	200	0	0%	127	140	13	9,3%
159	150	9	6%	160	170	10	5,9%
113	100	13	13%	174	170	4	2,4%
60	50	10	20%	56	50	6	12%

Fonte: Autoria própria

Através da tabela, é possível notar que todos os pontos foram atingidos conforme a tolerância definida (de 15 cm para cada eixo coordenado).

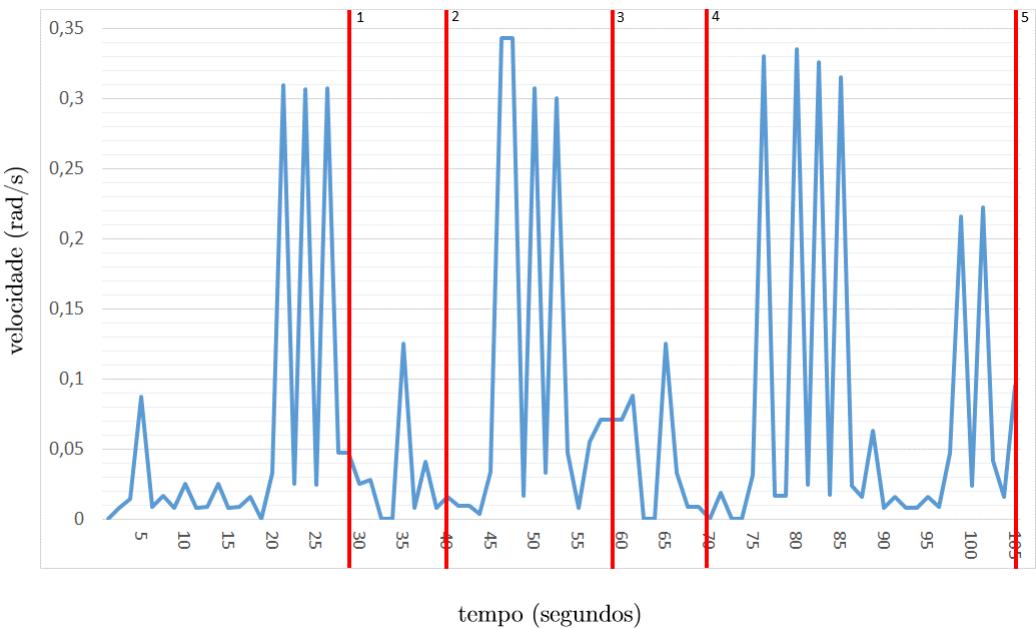
Os gráficos de velocidade de translação e velocidade angular estão disponíveis na Figura 63 e Figura 64, respectivamente. As linhas vermelhas indicam os instantes de chegadas ao *waypoints*.

Figura 63 – Velocidade de translação para o estudo de caso 1B



Fonte: Autoria própria

Figura 64 – Velocidade angular para o estudo de caso 1B



Fonte: Autoria própria

É possível se observar também o comportamento esperado nos casos anteriores,

visto que os picos de velocidade angular estão em momentos nos quais a velocidade de translação está baixa.

## 5.5 Estudo de Caso 2B

Para o estudo de caso 2B, a execução foi baseada na implementação de envio de mensagens em via única, com a trajetória composta pelos cinco *waypoints* de  $B^g$ . Os *screenshots* da execução foram registrados e estão disponíveis na [Figura 65](#) e [Figura 66](#). O ponto verde indica a coordenada inicial do robô. As mensagens destacadas em vermelho indicam os momentos nos quais o sistema detectou que o robô atingiu um *waypoint*.

Figura 65 – Screenshots do estudo de caso 2B (parte 1/2)



Fonte: Autoria própria

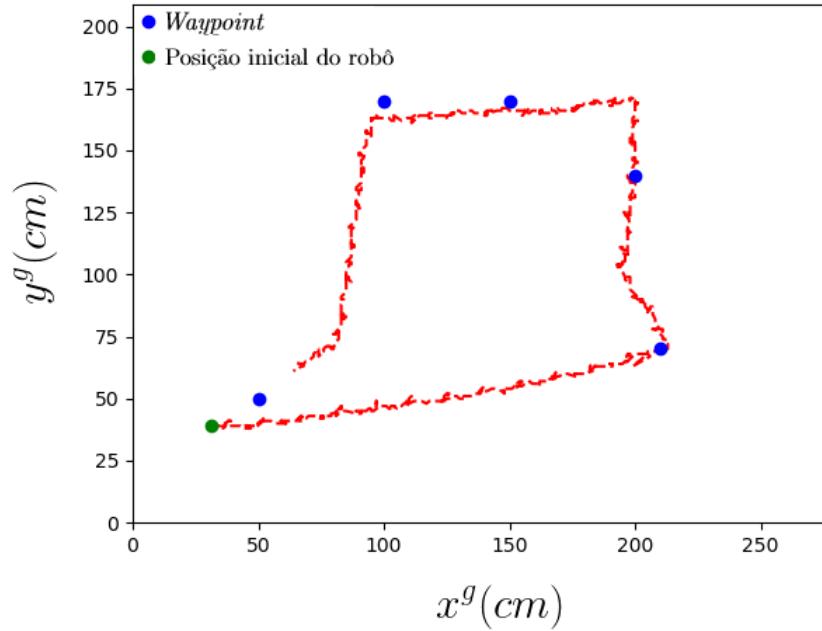
Figura 66 – Screenshots do estudo de caso 2B (parte 2/2)



Fonte: Autoria própria

A visão do caminho do robô está disponível no gráfico da Figura 67.

Figura 67 – Trajetória executada pelo robô no estudo de caso 2B



Fonte: Autoria própria

Os erros relativos e absolutos de chegada nos *waypoints* são apresentados na Tabela 11.

Tabela 11 – Tabela de erros nas chegadas para o estudo de caso 2B

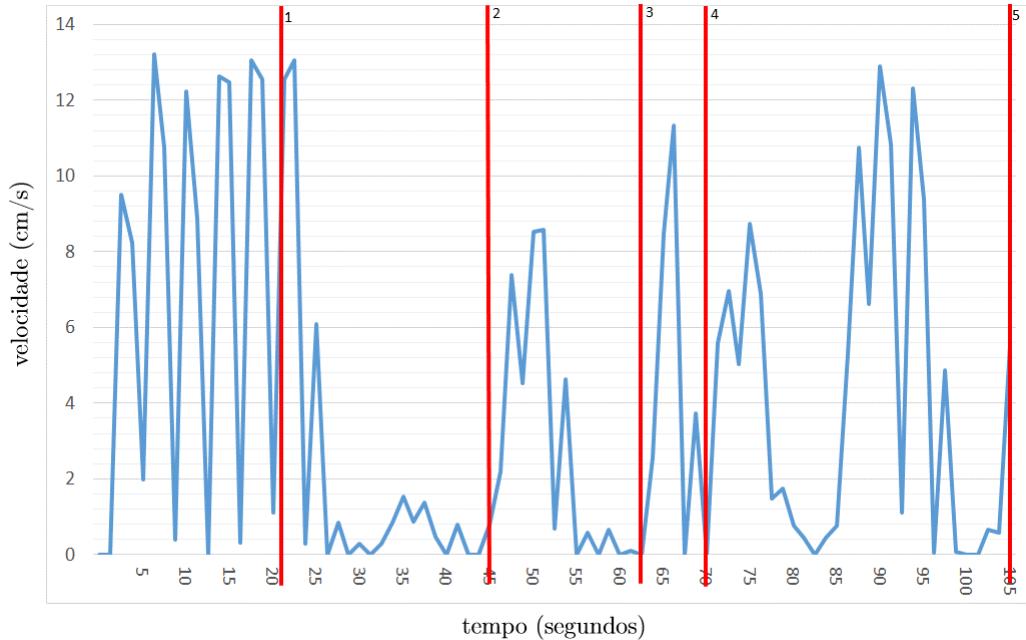
$x_{robô}$	$x_{waypoint}$	$E_{abs}(x^g)$ (cm)	$E_{rel}(x^g)$	$y_{robô}$	$y_{waypoint}$	$E_{abs}(y^g)$ (cm)	$E_{rel}(y^g)$
196	210	14	6,7%	65	70	5	7,1%
198	200	2	1%	126	140	14	10%
164	150	14	9,3%	165	170	5	2,9%
114	100	14	14%	164	170	6	3,5%
64	50	14	28%	61	50	11	22%

Fonte: Autoria própria

A tabela demonstra que o robô atingiu a todos os *waypoints* com a precisão especificada no projeto. O ponto que apresentou maior erro relativo foi o último. Isso também pode ser notado através da Figura 67.

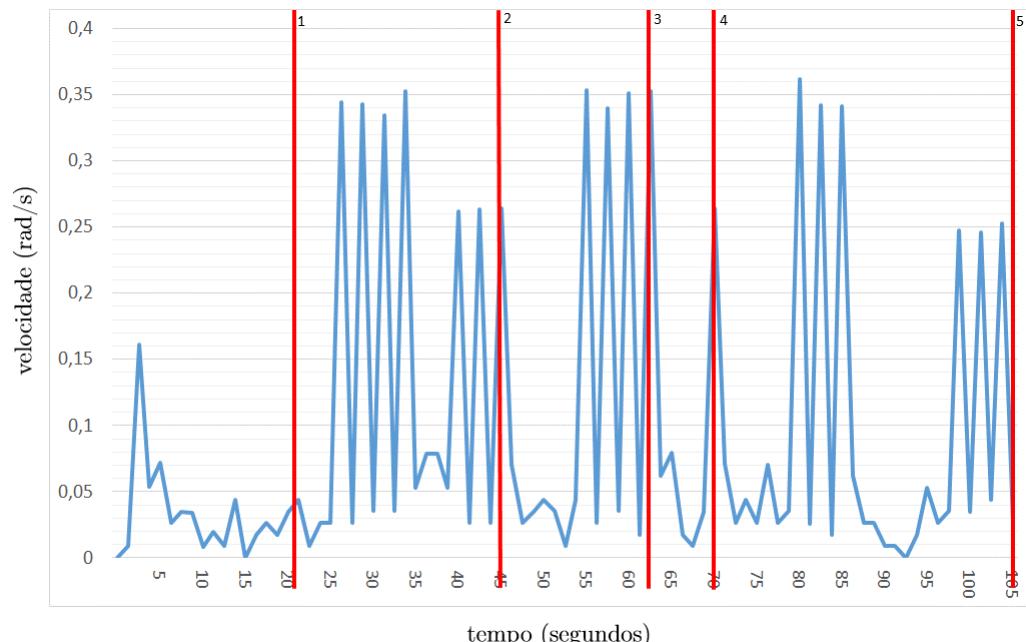
Os gráficos de velocidade de translação e velocidade angular estão apresentados na Figura 68 e Figura 69, respectivamente. As linhas vermelhas indicam os instantes de chegadas ao *waypoints*.

Figura 68 – Velocidade de translação para o estudo de caso 2B



Fonte: Autoria própria

Figura 69 – Velocidade angular para o estudo de caso 2B



Fonte: Autoria própria

Para esse estudo de caso, também foi possível observar que a velocidade angular apresentou picos em momentos nos quais a velocidade de translação era baixa.

Realizando uma comparação direta entre os resultados obtidos no estudo de caso 1B e 2B (que foram realizados com o mesmo grupo de *waypoints*), observa-se que a abordagem 1 e 2 finalizaram o caminho em um tempo aproximado. O menor tempo de chegada aos *waypoints* foi bastante dividido, sendo que, para os *waypoints* 1 e 3, a abordagem 2 foi mais rápida. Para o *waypoint* 2, a abordagem 1 foi mais rápida. Os *waypoints* 4 e 5 foram atingidos em tempos muito semelhantes. A abordagem 2 apresentou maiores erros nas chegadas às coordenadas estipuladas. Ambas trajetórias foram bastante consistentes, realizando poucos movimentos de rotação. Dessa maneira, é possível considerar que a implementação de comunicação em duas vias e a em via única apresentaram desempenho muito semelhante ao executar os *waypoints* do grupo  $B^g$ .

# 6 Conclusão

Os resultados apresentados nos estudos de caso do capítulo anterior demonstram que a proposta de projeto foi implementada com sucesso. Em todos os testes, o robô conseguiu atingir os *waypoints* predefinidos pelo usuário, respeitando a tolerância estabelecida e com um bom tempo de execução.

Os dados extraídos dos testes também demonstraram que as duas abordagens implementadas para o sistema de comunicação apresentaram o funcionamento esperado.

O desenvolvimento do projeto foi repleto de dificuldades. Por ser uma área bastante ampla, foi difícil definir, no início, quais componentes adquirir. Além disso, durante o início da implementação do algoritmo de movimentação, um dos servomotores queimou, mantendo seu eixo travado. Dessa forma, foi necessária a compra de mais um servomotor do modelo especificado, a fim de repor a unidade defeituosa. Também é válido comentar que, por ser um sistema que envolve várias implementações, componentes e tecnologias diferentes se comunicando de forma intermitente, foi despendida uma considerável quantidade de tempo com soluções de erros em mensagens de comunicação e em problemas de sincronismo.

## 6.1 Trabalhos Futuros

A lista a seguir apresenta sugestões de aprimoramentos e extensões do projeto em trabalhos futuros.

- Implementação de algoritmos de aprendizado de máquinas (redes neurais, algoritmos genéticos e Aprendizado por Reforço) para aprender e aprimorar o caminhar do robô hexápode, de forma a causar menos atrito com o solo.
- Melhorias para reconhecimento e segmentação dos marcadores fixados no robô utilizando outros métodos de processamento de imagem.
- Implementação de uma solução de desvio de obstáculos, caso o robô detecte um objeto a frente.
- Verificar a viabilidade do aumento na frequência de geração de dados e de transmissão para o robô.
- Estudo da modelagem cinemática e dinâmica do robô hexápode em um ambiente de simulação.

- Estudo de algoritmos de planejamento de trajetórias em ambientes dinâmicos, considerando as restrições de locomoção e do ambiente.
- Estudo e implementação de sistemas de controle para o rastreamento de trajetórias em ambientes dinâmicos considerando o tempo de locomoção.
- Implementação de um sistema de localização baseado em fusão sensorial IMU/GPS para operação em ambientes externos.

## Referências

- ALBUQUERQUE, M. P. de; ALBUQUERQUE, M. P. de. *Processamento de Imagens: Métodos e Análises*. 2000. Centro Brasileiro de Pesquisas Físicas – CBPF/MCT. Citado na página 33.
- ALVES, F. N. B. *Construção de robô quadrúpede experimental*. 2014. Trabalho de Graduação do Instituto Tecnológico de Aeronáutica - ITA. Citado na página 38.
- ARAUJO, G. M.; MENDONÇA, M. M.; FREIRE, E. O. *Reconhecimento Automático de Objetos Baseado em Cor e Forma Para Aplicações em Robótica*. 2011. Disponível em: <<http://www.gprufs.org/arquivos/reconhecimentoautomatico.pdf>>. Acesso em: 23.6.2019. Citado 3 vezes nas páginas 34, 57 e 59.
- ARDUINO. *Arduino - AboutUs*. 2018. Disponível em: <<https://www.arduino.cc/en/Main/AboutUs>>. Acesso em: 21.6.2018. Citado na página 48.
- CHÁVEZ-CLEMENTE, D. *Gait optimization for multi-legged walking robots, with application to a lunar hexapod*. 2011. Stanford Digital Repository. Disponível em: <<https://purl.stanford.edu/px063cb7934>>. Acesso em: 21.6.2018. Citado na página 16.
- COULOURIS, G. et al. *Sistemas Distribuídos - Conceitos e Projeto*. [S.l.]: Bookman, 2018. Citado na página 36.
- DANTAS, D. de O. *Construção e Acionamento de um Robô Móvel de Quatro Pernas*. 2014. Monografia apresentada ao Curso de Engenharia Elétrica da Universidade Federal do Maranhão. Citado 3 vezes nas páginas 19, 20 e 39.
- DEMASI, D. *Modelagem Dinâmica e de Controle de um Mecanismo de Três Graus de Liberdade para Aplicação em um Robô Hexápode*. 2012. Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Engenharia Mecânica e Tecnologia de Materiais do Centro Federal de Educação Tecnológica Celso Suckow da Fonseca - CEFET/RJ. Citado 5 vezes nas páginas 19, 20, 29, 30 e 38.
- ERDEN, M. S. *Six-Legged Walking Machine: The Robot-EA308*. 2006. Thesis submitted to the Graduate School of Natural and Applied Sciences of Middle East Technical University. Citado na página 29.
- FARIA, D. *Análise e Processamento de Imagem*. 2010. Trabalho realizado no âmbito do Mestrado Integrado em Engenharia Biomédica da Faculdade de Engenharia da Universidade do Porto. Citado 2 vezes nas páginas 32 e 33.
- FONTOURA, F. M.; NASCIMENTO, L. P. D.; GIOPPO, L. L. *Robô Hexápode Controlado por FPGA*. 2013. Trabalho de Conclusão do Curso de Engenharia da Computação da Universidade Tecnológica Federal do Paraná. Citado 2 vezes nas páginas 39 e 40.
- IFSC, W. *COMUNICAÇÃO XBEE*. 2015. Disponível em: <[https://wiki.sj.ifsc.edu.br-wiki/index.php/COMUNICAÇÃO\\_XBEE](https://wiki.sj.ifsc.edu.br-wiki/index.php/COMUNICAÇÃO_XBEE)>. Acesso em: 16.6.2019. Citado na página 36.

- INGRAM, A. J. *A new type of mechanical walking machine*. 2006. Dissertation submitted to the Faculty of Engineering in partial fulfilment of the requirements for the degree Doctor in Mechanical Engineering. Citado na página 29.
- JAIN; PETROL. *Thresholding*. 2011. Disponível em: <<https://www.cse.unr.edu/~bebis/CS791E/Notes/Thresholding.pdf>>. Acesso em: 23.6.2019. Citado na página 33.
- KRETZSCHMAR, C. *Robô Quadrúpede Utilizando Arduino*. 2012. Relatório do Trabalho Técnico-Científico de Conclusão do Curso de Ciência da Computação da Universidade do Vale do Itajaí. Citado 2 vezes nas páginas 38 e 39.
- LIMA, A. V. G. de. *Processamento de imagens para identificação de placas de automóveis*. 2009. Monografia para Trabalho de Conclusão de Curso de Ciência da Computação da Faculdade Farias Brito. Citado na página 32.
- MACHADO, J. A. T.; SILVA, M. F. *An Overview of Legged Robots*. 2006. ResearchGate. Disponível em: <[https://www.researchgate.net/publication/258972509\\_An\\_Overview\\_of\\_Legged\\_Robots](https://www.researchgate.net/publication/258972509_An_Overview_of_Legged_Robots)>. Acesso em: 21.6.2018. Citado na página 18.
- MEDEIROS, A. P. de. *Implementação de um controlador de seguimento de caminhos para robôs móveis com pernas*. 2014. Dissertação apresentada ao Instituto Tecnológico de Aeronáutica como parte dos requisitos para obtenção do título de Mestre. Citado 2 vezes nas páginas 28 e 40.
- MOREIRA, A. C. *Métodos para Segmentação Binária para Imagens em Tons de Cinza*. 2011. Disponível em: <<http://www.uel.br/grupos/gfna/segmentacaobinaria.pdf>>. Acesso em: 23.6.2019. Citado na página 33.
- MOTA, A. *O que é servomotor?* 2017. Disponível em: <<https://portal.vidadesilicio.com.br/o-que-e-servomotor/>>. Acesso em: 16.6.2019. Citado 2 vezes nas páginas 24 e 25.
- MOUNTAINBAJA, E. *Xbee – Comunicação entre dois Arduinos*. 2018. Disponível em: <<https://portal.vidadesilicio.com.br/xbee-comunicacao-entre-arduinoss/>>. Acesso em: 16.6.2019. Citado na página 36.
- OLIVEIRA, L. F. P. de. *Modelação, Simulação e Implementação de Padrões de Locomoção para Robôs Hexápodes*. 2016. Dissertação de Mestrado em Engenharia Electrotécnica e de Computadores - Instituto Superior de Engenharia do Porto. Citado 2 vezes nas páginas 18 e 39.
- OPENCV. *OpenCV - About*. 2019. Disponível em: <<https://opencv.org/about/>>. Acesso em: 16.6.2019. Citado na página 35.
- RABELO, H. P. *Desenvolvimento de um Robô Hexápode*. 2015. Trabalho de Final do Curso de Tecnologia em Mecatrônica Industrial do Instituto Federal de Educação, Ciência e Tecnologia do Ceará. Citado 7 vezes nas páginas 14, 15, 16, 20, 29, 30 e 40.
- SANTOS, A. *Servomotores*. 2007. Sumoderobos.org. Disponível em: <[www.pictronics.com.br/downloads/apostilas/servomotores.pdf](http://www.pictronics.com.br/downloads/apostilas/servomotores.pdf)>. Acesso em: 21.6.2018. Citado 3 vezes nas páginas 24, 25 e 26.

SANTOS, S. R. B. dos et al. *Experimental Framework for Evaluation of Guidance and Control Algorithms for UAVs*. 2011.

Disponível em: <[https://www.researchgate.net/publication/269700940\\_EXPERIMENTAL\\_FRAMEWORK\\_FOR\\_EVALUATION\\_OF\\_GUIDANCE\\_AND\\_CONTROL\\_ALGORITHMS](https://www.researchgate.net/publication/269700940_EXPERIMENTAL_FRAMEWORK_FOR_EVALUATION_OF_GUIDANCE_AND_CONTROL_ALGORITHMS)>. Acesso em: 20.6.2019. Citado na página [62](#).

SONY. *Entertainment Robot "aibo" Announced*. 2017. New Releases. Disponível em: <<https://www.sony.net/SonyInfo/News/Press/201711/17-105E/index.html>>. Acesso em: 21.6.2018. Citado na página [21](#).

SOUTO, R. F. *Modelagem Cinemática de um Robô Quadrúpede e Geração de Seus Movimentos Usando Filtragem Estocástica*. 2007. Trabalho de Graduação para a Universidade de Brasília. Citado na página [29](#).

TEDESCHI, F.; CARBONE, G. *Design Issues for Hexapod Walking Robots*. 2014. Article, Department DICEM, University of Cassino and Southern Lazio. Citado 2 vezes nas páginas [30](#) e [32](#).

THOMSEN, A. *Comunicação Passo-a-Passo entre XBEE e Arduino*. 2014. Filipe Flop. Disponível em: <<https://www.filipeflop.com/blog/tutorial-wireless-arduino-xbee-shield/>>. Acesso em: 21.6.2018. Citado 2 vezes nas páginas [59](#) e [60](#).

TODD, D. J. *Walking Machines - An Introduction to Legged Robots*. [S.l.]: Springer, 1985. Citado 4 vezes nas páginas [14](#), [15](#), [18](#) e [19](#).

VIDAL, V. *Sensor Ultrassônico HC-SR04 com Arduino*. 2018. Disponível em: <<http://blog.eletrogate.com/sensor-ultrassonico-hc-sr04-com-arduino/>>. Acesso em: 16.6.2019. Citado 2 vezes nas páginas [26](#) e [27](#).

## Apêndices

# APÊNDICE A – Instruções para inicialização do sistema

Esse apêndice é destinado à apresentação das instruções de como se inicializar o sistema para execução do robô.

## A.1 Partida do robô

1. Verificação do nível da bateria do robô. Pode-se utilizar um simples multímetro para isso. Se o nível estiver maior ou igual a 7 V, não é necessário recarregar a bateria.
2. *Upload* do algoritmo desejado para o *Arduino*. Definindo-se qual será a abordagem utilizada, deve-se carregar o código para o *Arduino* embarcado. Para isso, basta conectá-lo a um computador, equipado com arquivo do código desejado e a *IDE Arduino*, via conexão USB (utilizando um cabo USB tipo B para USB tipo A, no caso da placa *Mega*). Os *drivers* necessários são instalados automaticamente. É importante citar que a *shield XBee* deve ser removida fisicamente da placa para que essa comunicação seja realizada com sucesso, visto que ela utiliza a mesma porta serial internamente ao *Arduino* do que a comunicação USB.
3. Após esses passos, o robô está pronto para utilização. Basta conectar a bateria aos componentes a serem alimentados. Para isso, existe um conector na parte frontal do robô.
4. Ao término da execução, deve-se desconectar a bateria dos componentes, realizando um movimento contrário ao apresentado na instrução de número 3.

## A.2 Estação de solo

1. Conexão dos componentes a um computador. Deve-se conectar a câmera, que está fixada no teto da sala de testes, ao computador da estação de solo. Além disso, a placa *XBee Explorer*, equipada com um *chip XBee*, também deve ser conectada. Para isso, ambas utilizam de conexão USB.
2. Instalação do *Python* na versão 3. O computador da estação de solo deve estar equipado com o *Python* na versão 3. Também é necessária a instalação das seguintes bibliotecas: *cv2 (OpenCV)*, *XBeeDevice*, *Pyplot* e *openpyxl*.

3. Entre as duas opções de abordagem disponíveis, deve-se abrir o algoritmo desejado (sendo ele compatível ao carregado no *Arduino* embarcado anteriormente) para edição. Nesse momento, devem ser definidos os *waypoints* desejados, a distância visível pela câmera nos eixos e uma possível recalibração dos parâmetros estabelecidos para identificação das cores pelo processamento de imagem.
4. Execução da abordagem desejada. O robô deve estar numa posição visível pela câmera. É recomendável a utilização do botão *reset* no *Arduino* do robô, para re inicializar a execução do algoritmo embarcado.
5. Em caso de problemas de comunicação, deve-se verificar se os *XBee* estão configurados corretamente. Para isso, existe uma diversidade de tutoriais disponíveis na Internet, como por exemplo [www.filipeflop.com/blog/tutorial-wireless-arduino-xbee-shield/](http://www.filipeflop.com/blog/tutorial-wireless-arduino-xbee-shield/).

# APÊNDICE B – Referencial de movimentação dos servomotores

As tabelas apresentadas nesse apêndice contém as instruções utilizadas para movimentação do robô, através dos comandos individuais para cada servomotor.

Tabela 12 – Referencial *action group 5* (dar um passo a frente)

ID	Instrução para os servomotores
1	#1P1294#2P1544#3P1214#6P1322#10P1700#11P2122#12P2033#17P1538#18P1300#19P1616 #22P2109#23P2367#24P1567#27P1338#28P1656#29P1429#31P922#32P833T200
2	#1P767#2P1544#3P656#6P1322#10P1700#11P1668#12P1591#17P2211#18P1300#19P2100 #22P1383#23P1660#24P1567#27P878#28P1656#29P900#31P1339#32P1367T200
3	#1P767#2P1256#3P656#6P1811#10P1456#11P1668#12P1591#17P2211#18P1700#19P2100 #22P1383#23P1660#24P1211#27P878#28P1411#29P900#31P1339#32P1367T200
4	#1P1294#2P1256#3P1214#6P1811#10P1456#11P2122#12P2033#17P1538#18P1700#19P1616 #22P2109#23P2367#24P1211#27P1338#28P1411#29P1429#31P922#32P833T200

Tabela 13 – Referencial *action group 6* (virar a direita)

ID	Instrução para os servomotores
1	#1P1411#2P1300#3P1278#6P1500#10P1567#11P1697#12P1589#17P1538#18P1478#19P1674 #22P1411#23P1758#24P1367#27P1301#28P1567#29P1429#31P1332#32P1478T150
2	#1P1411#2P1189#3P900#6P1500#10P1567#11P1697#12P1589#17P1538#18P1478#19P1674 #22P1411#23P2144#24P1211#27P1301#28P1544#29P1429#31P1332#32P1478T150
3	#2P1500#24P1522T150
4	#3P1256#23P1744T150
5	#11P1878#27P1078T150
6	#10P1878#28P1855T150
7	#11P1674#27P1285T150
8	#19P1856#31P1189T150
9	#1P1411#2P1189#3P1222#6P1500#10P1567#11P1697#12P1589#17P1538#18P1478#19P1674 #22P1411#23P1758#24P1211#27P1301#28P1544#29P1429#31P1332#32P1478T150

Tabela 14 – Referencial *action group 7* (virar a esquerda)

ID	Instrução para os servomotores
1	#1P1411#2P1300#3P1278#6P1500#10P1567#11P1697#12P1589#17P1538#18P1478#19P1674 #22P1411#23P1758#24P1367#27P1301#28P1567#29P1429#31P1332#32P1478T150
2	#1P1411#2P1189#3P900#6P1500#10P1567#11P1697#12P1589#17P1538#18P1478#19P1674 #22P1411#23P2144#24P1211#27P1301#28P1544#29P1429#31P1332#32P1478T150
3	#2P878#24P900T150
4	#3P1256#23P1744T150
5	#11P1878#27P1078T150
6	#10P1256#28P1233T150
7	#11P1674#27P1285T150
8	#19P1856#31P1189T150
9	#1P1411#2P1189#3P1222#6P1500#10P1567#11P1697#12P1589#17P1538#18P1478#19P1674 #22P1411#23P1758#24P1211#27P1301#28P1544#29P1429#31P1332#32P1478T150

# APÊNDICE C – Custos do sistema

A [Tabela 15](#) apresenta o custo aproximado e o local de fornecimento dos componentes adquiridos para a implementação do projeto.

Tabela 15 – Custo de montagem do sistema

<b>Produto</b>	<b>Quantidade</b>	<b>Local de compra</b>	<b>Preço aproximado</b>
Kit do robô, com servomotores e placa controladora	1	<i>aliexpress.com</i>	U\$ 500,00
<i>Arduino Mega</i>	1	<i>americanas.com</i>	U\$ 15,00
<i>Logitech C922 PRO</i>	1	<i>americanas.com</i>	U\$ 87,50
<i>XBee PRO</i>	2	Fornecido pelo orientador	U\$ 90,00
Sensor ultrassônico HC-SR04	1	Fornecido pelo orientador	U\$ 2,50
Bateria <i>Zippy 5000mAh 7.4 V 2-cell</i>	1	Fornecido pelo orientador	U\$ 50,00

## Anexos

# ANEXO A – Codificação embarcada no robô

Arquivos disponíveis em [https://github.com/tiagoruzzon/hexapod\\_robot](https://github.com/tiagoruzzon/hexapod_robot).

## A.1 Abordagem 1: comunicação em duas vias

```
#include <Ultrasonic.h>
#define pino_trigger 22
#define pino_echo 24
String valores;
String coordenadas_robo;
String coordenadas_alvo;
int x_robo;
int y_robo;
float inclinacao_robo;
int x_alvo;
int y_alvo;
float inclinacao_alvo;
Ultrasonic ultrasonic(pino_trigger, pino_echo);
bool Verifica_Sensor()
{
    long microsec = ultrasonic.timing();
    float centimetros = ultrasonic.convert(microsec, Ultrasonic::CM);
    if (centimetros > 50)
    {
        return true;
    }
    else
    {
        return false;
    }
}
void De_Pe()
{
    Serial1.print("#4GC1\r\n");
}
```

```
    delay(2000);
}

void Andar_Frente()
{
    while(Verifica_Sensor()==false){}
    Serial1.print("#5GC2\r\n");
}

void Virar_Esquerda()
{
    Serial1.print("#7GC1\r\n");
}

void Virar_Direita()
{
    Serial1.print("#6GC1\r\n");
}

String getValue(String data, char separator, int index)
{
    int found = 0;
    int strIndex[] = {0, -1};
    int maxIndex = data.length()-1;

    for(int i=0; i<=maxIndex && found<=index; i++)
    {
        if(data.charAt(i)==separator || i==maxIndex)
        {
            found++;
            strIndex[0] = strIndex[1]+1;
            strIndex[1] = (i == maxIndex) ? i+1 : i;
        }
    }
    return found>index ? data.substring(strIndex[0], strIndex[1]) : "";
}

void Distribui_Valores()
{
    coordenadas_robo = getValue(valores, ';', 0);
    inclinacao_robo = getValue(valores, ';', 1).toFloat();
    coordenadas_alvo = getValue(valores, ';', 2);
    inclinacao_alvo = getValue(valores, ';', 3).toFloat();
    x_robo = getValue(coordenadas_robo, ',', 0).toInt();
```

```
y_robo = getValue(coordenadas_robo, ', ', 1).toInt();
x_alvo = getValue(coordenadas_alvo, ', ', 0).toInt();
y_alvo = getValue(coordenadas_alvo, ', ', 1).toInt();
}

void setup()
{
    Serial.begin(9600);
    Serial1.begin(9600);
}

void loop()
{
    De_Pe();
    Serial.print("Posicao");
    while (true)
    {
        while (!Serial.available()){}
        valores = Serial.readString();
        Distribui_Valores();
        if (abs(x_robo - x_alvo) < 15 && abs(y_robo - y_alvo) < 15)
        {
            De_Pe();
            Serial.print("Cheguei");
            delay(2000);
        }
        else
        {
            if (abs(inclinacao_alvo - inclinacao_robo) > 0.5)
            {
                if (inclinacao_alvo < inclinacao_robo)
                {
                    Virar_Esquerda();
                }
                else
                {
                    Virar_Direita();
                }
            }
            else
            {

```

```

        Andar_Frente();
    }
    Serial.print("Posicao");
}
}
}
```

## A.2 Abordagem 2: comunicação em via única

```

#include <Ultrasonic.h>
#define pino_trigger 22
#define pino_echo 24
String valores;
String coordenadas_robo;
String coordenadas_alvo;
int x_robo;
int y_robo;
float inclinacao_robo;
int x_alvo;
int y_alvo;
float inclinacao_alvo;
Ultrasonic ultrasonic(pino_trigger, pino_echo);
bool Verifica_Sensor()
{
    long microsec = ultrasonic.timing();
    float centimetros = ultrasonic.convert(microsec, Ultrasonic::CM);
    if (centimetros > 50)
    {
        return true;
    }
    else
    {
        return false;
    }
}
void De_Pe()
{
    Serial1.print("#4GC1\r\n");
    delay(2000);
}
```

```

void Andar_Frente()
{
    while( Verifica_Sensor()==false){}
    Serial1 . print( "#5GC2\r\n" );
}
void Virar_Esquerda()
{
    Serial1 . print( "#7GC1\r\n" );
}
void Virar_Direita()
{
    Serial1 . print( "#6GC1\r\n" );
}
String getValue( String data , char separator , int index )
{
    int found = 0;
    int strIndex [] = {0, -1};
    int maxIndex = data . length ()-1;
    for(int i=0; i<=maxIndex && found<=index ; i++)
    {
        if(data . charAt( i)==separator || i==maxIndex)
        {
            found++;
            strIndex [0] = strIndex [1]+1;
            strIndex [1] = ( i == maxIndex ) ? i+1 : i ;
        }
    }
    return found>index ? data . substring( strIndex [0] , strIndex [1] ) : "";
}
void Distribui_Valores()
{
    coordenadas_robo = getValue( valores , ';' ,0);
    inclinacao_robo = getValue( valores , ';' ,1).toFloat ();
    coordenadas_alvo = getValue( valores , ';' ,2);
    inclinacao_alvo = getValue( valores , ';' ,3).toFloat ();
    x_robo = getValue( coordenadas_robo , ';' ,0).toInt ();
    y_robo = getValue( coordenadas_robo , ';' ,1).toInt ();
    x_alvo = getValue( coordenadas_alvo , ';' ,0).toInt ();
    y_alvo = getValue( coordenadas_alvo , ';' ,1).toInt ();
}

```

```
}

void setup()
{
    Serial.begin(9600);
    Serial1.begin(9600);
}

void loop()
{
    De_Pe();
    while (true)
    {
        while (!Serial.available()){}
        valores = Serial.readString();
        Distribui_Valores();
        if (abs(x_robo - x_alvo) < 15 && abs(y_robo - y_alvo) < 15)
        {
            De_Pe();
            delay(2000);
        }
        else
        {
            if (abs(inclinacao_alvo - inclinacao_robo) > 0.5)
            {
                if (inclinacao_alvo < inclinacao_robo)
                {
                    Virar_Esquerda();
                }
                else
                {
                    Virar_Direita();
                }
            }
            else
            {
                Andar_Frente();
            }
        }
    }
}
```

# ANEXO B – Codificação da estação de solo

Arquivos disponíveis em [https://github.com/tiagoruzzon/hexapod\\_robot](https://github.com/tiagoruzzon/hexapod_robot).

## B.1 Abordagem 1: comunicação em duas vias

```

import cv2
import math
from digi.xbee.devices import XBeeDevice
import matplotlib.pyplot as plt
import time
from openpyxl import Workbook

def AdicionarPlanilha():
    global ws
    global wb
    global robot_center_x
    global robot_center_y
    global robot_inclination
    global target_x
    global target_y
    global target_inclination
    global split_time
    global cont_linhas_planilha
    linha = str(cont_linhas_planilha)
    ws[ 'A' + linha] = robot_center_x
    ws[ 'B' + linha] = robot_center_y
    ws[ 'C' + linha] = robot_inclination
    ws[ 'D' + linha] = target_x
    ws[ 'E' + linha] = target_y
    ws[ 'F' + linha] = target_inclination
    ws[ 'G' + linha] = split_time
    cont_linhas_planilha += 1

def CriarPlanilha():

```

```
global ws
global wb
wb = Workbook()
ws = wb.active
ws.title = 'Execucao_Robo'
ws = wb['Execucao_Robo']
ws['A1'] = 'Robo_X'
ws['B1'] = 'Robo_Y'
ws['C1'] = 'Inclinacao_Robo'
ws['D1'] = 'Waypoint_X'
ws['E1'] = 'Waypoint_Y'
ws['F1'] = 'Inclinacao_Waypoint'
ws['G1'] = 'Tempo'

def VerificaPosicao():
    global x_red
    global y_red
    global x_blue
    global y_blue
    global robot_center_x
    global robot_center_y
    global robot_inclination
    global target_x
    global target_y
    global target_inclination
    global List_X
    global List_Y
    global start_time
    global split_time
    ret_val, img = cam.read()
    img_filter = cv2.GaussianBlur(img.copy(), (3, 3), 0)
    img_filter = cv2.cvtColor(img_filter, cv2.COLOR_BGR2HSV)
    img_binary_red = cv2.inRange(img_filter.copy(), THRESHOLD_LOW_RED,
                                 THRESHOLD_HIGH_RED)
    img_binary_blue = cv2.inRange(img_filter.copy(), THRESHOLD_LOW_BLUE,
                                  THRESHOLD_HIGH_BLUE)
    img_binary_red = cv2.dilate(img_binary_red, None, iterations = 1)
    img_binary_blue = cv2.dilate(img_binary_blue, None, iterations = 1)
```

```

#Encontrar o círculo vermelho
img_contours = img_binary_red.copy()
contours = cv2.findContours(img_contours, cv2.RETR_EXTERNAL, \
                           cv2.CHAIN_APPROX_SIMPLE)[-2]
center = None
radius = 0
if len(contours) > 0:
    c = max(contours, key=cv2.contourArea)
    ((x_red, y_red), radius) = cv2.minEnclosingCircle(c)
    M = cv2.moments(c)
    if M["m00"] > 0:
        center = (int(M["m10"] / M["m00"]),
                   int(M["m01"] / M["m00"]))
    if radius < MIN_RADIUS:
        center = None
if center != None:
    cv2.circle(img, center, int(round(radius)), (0, 255, 0))

#Encontrar o círculo azul
img_contours = img_binary_blue.copy()
contours = cv2.findContours(img_contours, cv2.RETR_EXTERNAL, \
                           cv2.CHAIN_APPROX_SIMPLE)[-2]
center = None
radius = 0
if len(contours) > 0:
    c = max(contours, key=cv2.contourArea)
    ((x_blue, y_blue), radius) = cv2.minEnclosingCircle(c)
    M = cv2.moments(c)
    if M["m00"] > 0:
        center = (int(M["m10"] / M["m00"]),
                   int(M["m01"] / M["m00"]))
    if radius < MIN_RADIUS:
        center = None
if center != None:
    cv2.circle(img, center, int(round(radius)), (0, 255, 0))

#Exibindo resultados
x_red = x_red * dist_x / CAMERA_WIDTH
y_red = dist_y - (y_red * dist_y / CAMERA_HEIGHT)

```

```

x_blue = x_blue * dist_x / CAMERA_WIDTH
y_blue = dist_y - (y_blue * dist_y / CAMERA_HEIGHT)
robot_center_x = int( (x_blue + x_red) / 2 )
robot_center_y = int( (y_blue + y_red) / 2 )
List_X.append(robot_center_x)
List_Y.append(robot_center_y)
robot_inclination =
    round ( math.atan2(x_blue - x_red, y_blue - y_red), 2 )
target_inclination =
    round ( math.atan2(target_x - robot_center_x,
        target_y - robot_center_y) , 2)

#Atualizando imagem da camera
cv2.imshow('webcam', img)
cv2.waitKey(3)
message = str(robot_center_x) + "," + str(robot_center_y)
    + ";" + str(robot_inclination) + ";" + str(target_x)
    + "," + str(target_y) + ";" + str(target_inclination)
split_time = time.time() - start_time
AdicionarPlanilha()
print ("Enviado para o Robo:" + message)
return message

def main():
    global target_x
    global target_y
    global List_Target_X
    global List_Target_Y
    global List_X
    global List_Y
    global start_time
    global split_time
    global wb
    global cont_linhas_planilha
    global ws
    device = XBeeDevice(PORT, BUS)
    device.open()
    i = 0
    target = targets[i]

```

```

target_x = int(target.split(',') [0])
target_y = int(target.split(',') [1])
List_Target_X.append(target_x)
List_Target_Y.append(target_y)
start_time = time.time()
while True:
    print('Alvo:' + str(target))
    if target is not None:
        robot_info = device.read_data()
        while robot_info is None:
            robot_info = device.read_data()
        robot_info = robot_info.data.decode()
        print('Informacao_ Recebida do Robo:' + str(robot_info))
        if 'Posicao' in robot_info:
            device.send_data_broadcast(VerificaPosicao())
        elif 'Cheguei' in robot_info:
            ws[ 'H' + str(cont_linhas_planilha - 1)] = 'Cheguei'
            if i < len(targets) - 1:
                i += 1
                target = targets[i]
                target_x = int(target.split(',') [0])
                target_y = int(target.split(',') [1])
                List_Target_X.append(target_x)
                List_Target_Y.append(target_y)
                device.send_data_broadcast(VerificaPosicao())
            else:
                break
        wb.save('relatorio.xlsx')
        print("\n\n")
device.close()
print("\n\n")
print("O robô chegou a todos os alvos. A aplicação encerrada.")
plt.plot(List_X, List_Y, 'r--',
         List_Target_X, List_Target_Y, 'bo',
         List_X[0], List_Y[0], 'go')
plt.axis([0, dist_x, 0, dist_y])
plt.show()

```

#Parametros

```
# Azul
THRESHOLD_LOW_BLUE = (80, 140, 87);
THRESHOLD_HIGH_BLUE = (120, 240, 130);
# Vermelho
THRESHOLD_LOW_RED = (0, 130, 120);
THRESHOLD_HIGH_RED = (15, 240, 180);
# Distancia visveis nos eixos
dist_x = 277
dist_y = 209
# Resolucao desejada
CAMERA_WIDTH = 1024
CAMERA_HEIGHT = 768
# Raio minimo para o calculo de contorno
MIN_RADIUS = 2
PORT = "COM25"
BUS = 9600
# Inicializacao de variaveis globais
x_red = 0
y_red = 0
x_blue = 0
y_blue = 0
robot_center_x = 0
robot_center_y = 0
robot_inclination = 0
target_x = 0
target_y = 0
target_inclination = 0
start_time = 0
split_time = 0
cont_linhas_planilha = 2
ws = None
wb = None
List_X = []
List_Y = []
List_Target_X = []
List_Target_Y = []
#Lista de Waypoints
targets = [ '210,70', '200,140', '150,170', '100,170', '50,50' ]
# Inicializacao da camera
```

```

cam = cv2.VideoCapture(1)
# Define a resolução escolhida para a imagem
cam.set(cv2.CAP_PROP_FRAME_WIDTH, CAMERA_WIDTH)
cam.set(cv2.CAP_PROP_FRAME_HEIGHT, CAMERA_HEIGHT)
print ("Camera Inicializada")
print ("Monitoramento do Robô Ativo")
CriarPlanilha()
main()

```

## B.2 Abordagem 2: comunicação em via única

```

import cv2
import math
from digi.xbee.devices import XBeeDevice
import matplotlib.pyplot as plt
import time
from openpyxl import Workbook

def AdicionarPlanilha():
    global ws
    global wb
    global robot_center_x
    global robot_center_y
    global robot_inclination
    global target_x
    global target_y
    global target_inclination
    global split_time
    global cont_linhas_planilha

    linha = str(cont_linhas_planilha)
    ws[ 'A' + linha] = robot_center_x
    ws[ 'B' + linha] = robot_center_y
    ws[ 'C' + linha] = robot_inclination
    ws[ 'D' + linha] = target_x
    ws[ 'E' + linha] = target_y
    ws[ 'F' + linha] = target_inclination
    ws[ 'G' + linha] = split_time
    cont_linhas_planilha += 1

```

```
def CriarPlanilha():
    global ws
    global wb
    wb = Workbook()
    ws = wb.active
    ws.title = 'Execucao_Robo'
    ws = wb['Execucao_Robo']
    ws['A1'] = 'Robo_X'
    ws['B1'] = 'Robo_Y'
    ws['C1'] = 'Inclinacao_Robo'
    ws['D1'] = 'Waypoint_X'
    ws['E1'] = 'Waypoint_Y'
    ws['F1'] = 'Inclinacao_Waypoint'
    ws['G1'] = 'Tempo'

def VerificaPosicao():
    global x_red
    global y_red
    global x_blue
    global y_blue
    global robot_center_x
    global robot_center_y
    global robot_inclination
    global target_x
    global target_y
    global target_inclination
    global List_X
    global List_Y
    global start_time
    global split_time
    ret_val, img = cam.read()
    img_filter = cv2.GaussianBlur(img.copy(), (3, 3), 0)
    img_filter = cv2.cvtColor(img_filter, cv2.COLOR_BGR2HSV)
    img_binary_red = cv2.inRange(img_filter.copy(), THRESHOLD_LOW_RED,
                                 THRESHOLD_HIGH_RED)
    img_binary_blue = cv2.inRange(img_filter.copy(), THRESHOLD_LOW_BLUE,
                                 THRESHOLD_HIGH_BLUE)
    img_binary_red = cv2.dilate(img_binary_red, None, iterations = 1)
    img_binary_blue = cv2.dilate(img_binary_blue, None, iterations = 1)
```

```

#Encontrar círculo vermelho
img_contours = img_binary_red.copy()
contours = cv2.findContours(img_contours, cv2.RETR_EXTERNAL, \
    cv2.CHAIN_APPROX_SIMPLE)[-2]
center = None
radius = 0
if len(contours) > 0:
    c = max(contours, key=cv2.contourArea)
    ((x_red, y_red), radius) = cv2.minEnclosingCircle(c)
    M = cv2.moments(c)
    if M["m00"] > 0:
        center = (int(M["m10"] / M["m00"]),
                  int(M["m01"] / M["m00"]))
        if radius < MIN_RADIUS:
            center = None
if center != None:
    cv2.circle(img, center, int(round(radius)), (0, 255, 0))

#Encontrar círculo azul
img_contours = img_binary_blue.copy()
contours = cv2.findContours(img_contours, cv2.RETR_EXTERNAL, \
    cv2.CHAIN_APPROX_SIMPLE)[-2]
center = None
radius = 0
if len(contours) > 0:
    c = max(contours, key=cv2.contourArea)
    ((x_blue, y_blue), radius) = cv2.minEnclosingCircle(c)
    M = cv2.moments(c)
    if M["m00"] > 0:
        center = (int(M["m10"] / M["m00"]),
                  int(M["m01"] / M["m00"]))
        if radius < MIN_RADIUS:
            center = None
if center != None:
    cv2.circle(img, center, int(round(radius)), (0, 255, 0))

#Exibindo resultados
x_red = x_red * dist_x / CAMERA_WIDTH

```

```

y_red = dist_y - (y_red * dist_y / CAMERA_HEIGHT)
x_blue = x_blue * dist_x / CAMERA_WIDTH
y_blue = dist_y - (y_blue * dist_y / CAMERA_HEIGHT)
robot_center_x = int( (x_blue + x_red) / 2 )
robot_center_y = int( (y_blue + y_red) / 2 )
List_X.append(robot_center_x)
List_Y.append(robot_center_y)
robot_inclination =
    round ( math.atan2(x_blue - x_red, y_blue - y_red), 2 )
target_inclination =
    round ( math.atan2(target_x - robot_center_x,
    target_y - robot_center_y) , 2)

#Atualizando imagem da camera
cv2.imshow('webcam', img)
cv2.waitKey(3)
message = str(robot_center_x) + "," + str(robot_center_y) +
    ";" + str(robot_inclination) + ";" + str(target_x) +
    "," + str(target_y) + ";" + str(target_inclination)
split_time = time.time() - start_time
AdicionarPlanilha()
return message

def main():
    global target_x
    global target_y
    global List_Target_X
    global List_Target_Y
    global List_X
    global List_Y
    global start_time
    global split_time
    global wb
    global cont_linhas_planilha
    global ws
    global robot_center_x
    global robot_center_y
    global contador_envio
    device = XBeeDevice(PORT, BUS)

```

```

device.open()
i = 0
target = targets[i]
target_x = int(target.split(',')[0])
target_y = int(target.split(',')[1])
List_Target_X.append(target_x)
List_Target_Y.append(target_y)
start_time = time.time()
while True:
    if target is not None:
        if abs(robot_center_x - target_x) < 15
            and abs(robot_center_y - target_y) < 15:
            ws['H' + str(cont_linhas_planilha - 1)] = 'Cheguei'
            if i < len(targets) - 1:
                i += 1
                target = targets[i]
                target_x = int(target.split(',')[0])
                target_y = int(target.split(',')[1])
                List_Target_X.append(target_x)
                List_Target_Y.append(target_y)
                mensagem = VerificaPosicao()
                if contador_envio > 30 and mensagem != '0':
                    device.send_data_broadcast(mensagem)
                    print('Alvo:' + str(target))
                    print("Enviado para o Robo:" + mensagem)
                    ws['I' + str(cont_linhas_planilha)] =
                        'Enviado para Robo'
                    print("\n\n")
                    contador_envio = 0
                else:
                    contador_envio = contador_envio + 1
            else:
                break;
    else:
        mensagem = VerificaPosicao()
        if contador_envio > 30 and mensagem != '0':
            device.send_data_broadcast(mensagem)
            print('Alvo:' + str(target))
            print("Enviado para o Robo:" + mensagem)

```

```

ws[ 'I' + str(cont_linhas_planilha)] =
    'Enviado para Robo'
print ("\n\n")
contador_envio = 0
else:
    contador_envio = contador_envio + 1
device.close()
wb.save('relatorio.xlsx')
print ("\n\n")
print ("O robô chegou a todos os alvos. Aplicação encerrada.")
plt.plot(List_X, List_Y, 'r--',
          List_Target_X, List_Target_Y, 'bo',
          List_X[0], List_Y[0], 'go')
plt.axis([0, dist_x, 0, dist_y])
plt.show()

#Parametros
# Azul
THRESHOLD_LOW_BLUE = (80,140,87);
THRESHOLD_HIGH_BLUE = (120, 240, 130);
# Vermelho
THRESHOLD_LOW_RED = (0,130,120);
THRESHOLD_HIGH_RED = (15,240,180);
# Distancia visveis nos eixos
dist_x = 277
dist_y = 209
# Resolucao desejada
CAMERA_WIDTH = 1024
CAMERA_HEIGHT = 768
# Raio minimo para o círculo de contorno
MIN_RADIUS = 2
PORT = "COM25"
BUS = 9600
# Inicialização de variáveis globais
x_red = 0
y_red = 0
x_blue = 0
y_blue = 0
robot_center_x = 0

```

```
robot_center_y = 0
robot_inclination = 0
target_x = 0
target_y = 0
target_inclination = 0
start_time = 0
split_time = 0
cont_linhas_planilha = 2
ws = None
wb = None
List_X = []
List_Y = []
List_Target_X = []
List_Target_Y = []
contador_envio = 0
#Lista de Waypoints
targets = [ '210,70' , '200,140' , '150,170' , '100,170' , '50,50' ]
# Inicialização da camera
cam = cv2.VideoCapture(1)
# Define a resolução escolhida para a imagem
cam.set(cv2.CAP_PROP_FRAME_WIDTH, CAMERA_WIDTH)
cam.set(cv2.CAP_PROP_FRAME_HEIGHT, CAMERA_HEIGHT)
print ("Camera_Inicializada")
print ("Monitoramento_dos_Robo_Ativo")
CriarPlanilha()
main()
```