



UNIVERSIDADE FEDERAL DE SÃO PAULO
Instituto de Ciência e Tecnologia - Campus São José dos Campos

**Aceleração do Aprendizado por Reforço
Aplicado ao Planejamento de Caminho para
Robôs Transportadores de Cargas**

São José dos Campos - SP

Agosto de 2021

João Victor de Mesquita Cândido dos Santos

**Aceleração do Aprendizado por Reforço Aplicado ao Planejamento de
Caminho para Robôs Transportadores de Cargas**

**Trabalho de Graduação apresentado a
Universidade Federal de São Paulo, como
requisito parcial para obtenção do título de
Engenheiro de Computação.**

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

Orientador: Prof. Dr. Sérgio Ronaldo Barros dos Santos

São José dos Campos - SP
Agosto de 2021

Este trabalho é dedicado à minha mãe Cristiane, minha sobrinha Eva Helena, minha querida tia Sandra, meu irmão Pedro, aos meus amigos e aos colegas que fiz durante minha vida e trajetória acadêmica, pois sem eles não chegaria onde estou hoje. Também dedico esse trabalho à todos aqueles que apesar das dificuldades ousam em sonhar e seguir em frente batalhando por uma vida melhor.

Agradecimentos

Gostaria de agradecer à Deus por ter me dado forças para não desistir durante esses anos de curso. Também agradeço à minha mãe Cristiane por sempre me motivar e me animar para prosseguir com os estudos e na busca de um sucesso pessoal e profissional.

Também agradeço aos meus amigos, tanto os de longe quanto os de perto, que me auxiliaram no suporte emocional e sempre me encorajavam durante o processo de desenvolvimento desse trabalho.

Agradeço também ao meu orientador Prof. Dr. Sérgio Ronaldo Barros pelo apoio e acompanhamento durante o desenvolvimento do trabalho dispondo do seu tempo para auxiliar no que fosse necessário.

Resumo

Robôs autônomos vem ganhando espaço dentro da sociedade devido a sua grande gama de aplicações. Uma utilização para robôs autônomos está no setor de transporte de produtos, entretanto pensando no processo de locomoção do robô dentro de um ambiente é necessário a realização do planejamento do melhor caminho. Inúmeros algoritmos buscam solucionar esse problema, dentre eles existe a abordagem de Aprendizado por Reforço através da utilização do algoritmo de *Q-learning*. O problema dessas abordagens é a lentidão no processo de convergência. Para isso, este projeto intitulado de Aceleração do Aprendizado por Reforço Aplicado ao Planejamento de Caminho para Robôs Transportadores de Cargas propõe uma variação do algoritmo de *Q-learning*, chamado de QLDR. O algoritmo utiliza a abordagem de uma dupla recompensa, uma estática recebida imediatamente após a tomada de ação e uma recompensa dinâmica utilizando um cálculo de variação da distância euclidiana entre o estado atual e o ponto de destino otimizando o processo na busca do melhor caminho. Através do desenvolvimento do algoritmo foi feita uma comparação com o *Q-learning* clássico e comprovada a otimização no processo de convergência fazendo com que o robô encontrasse o melhor caminho em menor tempo. Por fim, para validação de conceito foi utilizado um robô hexápode real para se locomover dentro de um ambiente treinado pelo algoritmo QLDR, para o processo de locomoção do robô foi utilizado um sistema de visão externa e o auxílio de algoritmos de processamento de imagem em *Python* tanto para o planejamento da rota, quanto para o processo de locomoção no ambiente em si.

Palavras-chaves: Robôs Hexápodes, Inteligência Artificial, *Reinforcement Learning*, *Q-learning*, Processamento de Imagem.

Abstract

Autonomous robots have been gaining space within society due to their wide range of applications. One use for autonomous robots is in the product transport sector, however, thinking about the robot's locomotion process within an environment, it is necessary to carry out the best path planning. Numerous algorithms seek to solve this problem, among them there is the Reinforcement Learning approach through the use of the Q-learning algorithm. The problem of these approaches is the slow convergence process. Therefore, this project entitled Accelerated Learning by Reinforcement Application to Path Planning for Product Transport Robots proposes a variation of the Q-learning algorithm, called QLDR. The algorithm uses the approach of a double reward, a static received immediately after taking action and a dynamic reward using a Euclidean distance variation calculation between the current state and the destination point, optimizing the process in search of the best path. Through the development of the algorithm, a comparison with the classic Q-learning was made and the optimization in the convergence process was proven, making the robot find the best path in less time. Finally, for concept validation, a real hexapod robot was used to move within an environment trained by the QLDR algorithm, for the robot locomotion process an external vision system and the aid of image processing algorithms in Python were used for planning of the route, as for the process of locomotion in the environment itself.

Keywords: Hexapod Robots, Artificial Intelligence, Reinforcement Learning, Q-learning, Image Processing.

Listas de ilustrações

Figura 1 – Robô com Rodas - <i>Rover</i>	20
Figura 2 – Exemplo de um Robô que utiliza locomoção por Trilhos	20
Figura 3 – <i>Spot</i>	21
Figura 4 – Centro de Massa Sobre um Polígono	22
Figura 5 – Fases de Postura e de Balanço	23
Figura 6 – Modelos Típicos de Robôs Hexápodos	24
Figura 7 – Estrutura de Pernas de Insetos	24
Figura 8 – Planejamento de caminho entre os pontos A(x,y) e B(w,z)	25
Figura 9 – Representação Gráfica para Localização de Robôs	28
Figura 10 – Representação Matricial de Uma Imagem	30
Figura 11 – Representação Matricial de Uma Imagem Colorida	30
Figura 12 – Representação digital de uma imagem: (a) Convenção de eixos e (b) Convenção matricial	31
Figura 13 – Qualidade da imagem no processo de amostragem	31
Figura 14 – Exemplo de processo de quantização de uma imagem	32
Figura 15 – Intensidade luminosa dos pixels na região em destaque na imagem: (a) colorida e (b) preto e branco	32
Figura 16 – Exemplos de imagem binária	34
Figura 17 – Sistema de cor RGB	34
Figura 18 – Sistema de cor RGB	35
Figura 19 – Modelo Padrão de Aprendizado por Reforço	36
Figura 20 – Funcionamento de um sistema modelado como MDP	37
Figura 21 – Processo de Decisão de Markov em Aprendizado por Reforço	38
Figura 22 – Hexápode concluído	42
Figura 23 – Protótipo SiWaRel	43
Figura 24 – Mapa da Cidade de São Paulo	45
Figura 25 – Planta Baixa da Cidade de São Paulo	46
Figura 26 – Fluxograma do Processo de Planejamento de Caminho e Locomoção do Robô no Ambiente	51
Figura 27 – Exemplo de configuração de um Ambiente 4×4	54
Figura 28 – Distribuição de Estados	55
Figura 29 – Distribuição das Ações	55
Figura 30 – Fluxograma da Função de Transição de Estados	57
Figura 31 – Melhor Caminho	61
Figura 32 – Representação de etapas em Q-learning	61
Figura 33 – Planta Baixa de Barcelona	65

Figura 34 – Conversão de pixels	67
Figura 35 – Mapa de Barcelona	69
Figura 36 – Identificação dos Obstáculos no Mapa de Barcelona	70
Figura 37 – Convergência de Recompensas no Treinamento	71
Figura 38 – Passos Dados no Treinamento do Mapa de Barcelona	72
Figura 39 – Relação entre Passos e Recompensas no Treinamento do Mapa de Barcelona	73
Figura 40 – Decaimento de epsilon	74
Figura 41 – Melhor caminho para o Mapa de Barcelona	75
Figura 42 – Estudo de Caso B - Chicago, Los Angeles, São Paulo e Vancouver	76
Figura 43 – Estudo de Caso B - Chicago, Los Angeles, São Paulo e Vancouver	77
Figura 44 – Configuração de Ambiente	79
Figura 45 – Disposição de Obstáculos	80
Figura 46 – Ambiente Segmentado	81
Figura 47 – Ambiente com Segmentação Binária	82
Figura 48 – Obstáculos Através da Segmentação Binária	82
Figura 49 – Melhor Caminho no Ambiente	83
Figura 50 – Espaço de Estados, ações e Tamanho de Caminho	83
Figura 51 – Identificação do Robô no Ambiente	84
Figura 52 – Coordenadas dos Estados no Ambiente	85
Figura 53 – Tradução dos Estados no Ambiente	86
Figura 54 – Robô Posicionado no Ambiente	87
Figura 55 – Diagrama dos Componentes do Sistema de Locomoção	97
Figura 56 – Servomotor DS3218MG	99
Figura 57 – Segmentação de Perna	99
Figura 58 – Placa Controladora de Servomotores	100
Figura 59 – Arduíno Mega 2560	100
Figura 60 – Bateria LiPo	101
Figura 61 – Instalação da Bateria no Robô	101
Figura 62 – Módulos <i>XBee</i>	102
Figura 63 – Esquemático sistema embarcado no robô	103
Figura 64 – Hexápode concluído	104
Figura 65 – Visão do <i>Software</i> de Definição de AG	105
Figura 66 – Formato de Instruções da Placa Controladora de Servos	106
Figura 67 – Câmera Instalada no Ambiente	107
Figura 68 – <i>Software</i> XCTU	109
Figura 69 – Representação Gráfica da Visão Obtida Pelo Algoritmo de Guiagem	111
Figura 70 – Fluxograma de Locomoção do Robô	112

Lista de tabelas

Tabela 1 – Representação da tabela Q	39
Tabela 2 – Avaliação das Recompensas e Passos dados no Treinamento dos Ambientes	78
Tabela 3 – Comparação de Aprendizado entre Algoritmo QLDR e Q- <i>learning</i> Clássico	78
Tabela 4 – Conexão entre canais de placa controladora e servomotores	102
Tabela 5 – Conexões do Arduino	103
Tabela 6 – Definição dos Grupos de Ação na Memória da Placa Controladora de Servos	105
Tabela 7 – Parametrização de Cores no Padrão HSV para Localização do Robô	108
Tabela 8 – Definição de Mensagens entre Robô e Estação de Solo	113

Lista de abreviaturas e siglas

IA - Inteligência Artificial

AR - Aprendizado por Reforço

AM - Aprendizado de Máquina

MDP - Markov Process Decision

PI - Processamento de Imagem

QLDR - Q-learning with Duple Reward

Sumário

1	INTRODUÇÃO	14
1.1	Contextualização e Motivação	14
1.2	Objetivos Gerais	16
1.3	Objetivos Específicos	16
1.4	Organização do Trabalho	16
2	REVISÃO DA LITERATURA	18
2.1	Fundamentação Teórica	18
2.1.1	Problemas Fundamentais da Robótica	18
2.2	Locomoção de Robôs	18
2.2.1	Locomoção de Robôs com Rodas	19
2.2.2	Locomoção de Robôs com Trilhos	20
2.2.3	Locomoção de Robôs com Pernas	21
2.2.4	Estabilidade de Robôs	22
2.2.4.1	Estabilidade Estática	22
2.2.4.2	Estabilidade Dinâmica	22
2.2.5	Velocidade de Locomoção dos Robôs com Pernas	23
2.2.6	Configurações de Robôs Hexápodos	23
2.3	Planejamento de Caminho	25
2.4	Localização	27
2.5	Percepção	29
2.5.1	Amostragem e Quantização	30
2.5.1.1	Amostragem	31
2.5.1.2	Quantização	31
2.5.2	Segmentação	33
2.5.2.1	Segmentação Binária (<i>Thresholding</i>)	33
2.5.3	Sistema HSV de Cores	34
2.6	Aprendizado de Máquina	35
2.6.1	Aprendizado por Reforço	35
2.6.1.1	<i>Q-Learning</i>	38
2.6.1.2	Estratégias de Exploração e Explotação	40
2.6.2	Pseudocódigo do treinamento utilizando <i>Q-learning</i>	41
2.7	Trabalhos Relacionados	41
3	DECLARAÇÃO DO PROBLEMA	45

4	DESENVOLVIMENTO	48
4.1	Metodologia	48
4.1.1	Simulação	48
4.1.2	Prova de Conceito	49
4.2	Integração Hardware e Software	50
4.3	Algoritmo Q-learning de Dupla Recompensa	53
4.3.1	Representação dos Estados	53
4.3.2	Espaço de Ações	55
4.3.3	Modelo de Transição de Estado	56
4.3.4	Esquema de Recompensa	58
4.3.4.1	Recompensa Estática	59
4.3.4.2	Recompensa Dinâmica	59
4.3.5	Implementação do Algoritmo	60
4.3.5.1	Política de ϵ	62
4.3.5.2	Parâmetros α e γ	63
4.3.5.3	Algoritmo	63
4.4	Algoritmo Gerador de Obstáculos	65
5	RESULTADOS OBTIDOS	68
5.1	Resultados em Simulação	68
5.1.1	Estudo de Caso A - Barcelona	69
5.1.1.1	Definição de Estados	69
5.1.1.2	Tempo de Treinamento	70
5.1.1.3	Convergência do Algoritmo	70
5.1.1.4	Número de Passos	71
5.1.1.5	Número de Passos <i>versus</i> Recompensa	72
5.1.1.6	Comportamento de <i>epsilon</i>	73
5.1.1.7	Avaliação do Melhor Caminho	74
5.2	Estudo de Caso B - Demais Ambientes	75
5.3	Comparação do Aprendizado	78
5.4	Resultados Experimentais	79
5.4.1	Mapeamento dos Estados no Ambiente	83
6	CONCLUSÃO	88
6.1	Trabalhos Futuros	89
	REFERÊNCIAS	90

APÊNDICES	95
APÊNDICE A – AMBIENTE EXPERIMENTAL PARA VALIDAÇÃO DE CONCEITO	96
A.1 Processo de Locomoção do Robô	96
A.1.1 Montagem e Adaptação do Robô	98
A.1.2 Controle de Locomoção do Robô	104
A.2 Localização do Robô	106
A.2.1 Sistema de Visão Externa	106
A.2.2 Algoritmo de Processamento de Imagem	107
A.2.3 OpenCV	109
A.3 Comunicação Sem Fio	109
A.4 Sistema de Guiagem do Robô	110
A.5 Modo de Troca de Mensagens	112
APÊNDICE B – CONSTRUÇÃO DO ALGORITMO QLDR	114
B.1 Classe de Geração de Ambiente de Treinamento	114
B.2 Algoritmo de Treinamento	115
B.3 Algoritmo Gerador de Resultados de Treinamento	117
APÊNDICE C – ALGORITMO DE ANÁLISE DA TABELA Q E GE- RAÇÃO DO CAMINHO	119
APÊNDICE D – ALGORITMO DE PROCESSAMENTO DE IMAGEM PARA LOCOMOÇÃO E MAPEAMENTO DO AM- BIENTE	122
APÊNDICE E – ALGORITMO RESPONSÁVEL PELA LOCOMOÇÃO DO ROBÔ	127

1 Introdução

1.1 Contextualização e Motivação

Nos últimos cem anos, a humanidade vem experimentando uma série de mudanças revolucionárias na sua relação com a tecnologia. O progresso tecnológico caminha dando passos largos o que muitas das vezes chega ser um desafio inenarrável acompanhar essa trajetória da evolução humana. Entretanto, por mais que os avanços tecnológicos tenham trazido benefícios, ainda estamos sujeitos a situações fora do nosso controle como pandemia, desastres naturais etc. Um dos exemplos foi o surgimento do coronavírus causando a grande pandemia da COVID-19 no ano de 2020 caracterizada pelo seu alto índice de contágio, que em pouco tempo se espalhou por todo mundo. A pandemia do coronavírus pegou a humanidade desprevenida obrigando governos, empresas e diversos setores da sociedade a adotarem medidas restritivas de distanciamento social como medida preventiva para evitar uma maior propagação do vírus e um colapso dos sistemas de saúde.

Com a pandemia de COVID-19, vários setores da sociedade precisaram se reinventar para continuar atendendo as demandas da população, um exemplo disso são restaurantes e supermercados que implementaram massivamente a utilização de *delivery*. Contudo, as entregas ainda são feitas por seres humanos e uma maior quantidade de motos e carros para realizar as entregas que acaba afetando o ambiente com um aumento das emissões de CO_2 na atmosfera. Pensando nisso, se faz importante trazer alternativas que evitem o contato humano, na entrega de pequenas cargas e produtos, utilizando algum meio que não venha prejudicar o ambiente. Neste contexto, o campo da robótica móvel pode trazer grandes benefícios para a sociedade.

Nas últimas duas décadas houve um crescimento exponencial em pesquisa no contexto de robôs móveis sejam eles com pernas ou com rodas([RODRIGUEZ; GARCIA, 2014](#)) devido à sua versatilidade com o advento da locomoção. Entretanto há uma diferença pontual entre esses modelos, segundo ([SANTOS; JÚNIOR et al., 2012](#)) os robôs com rodas correspondem ao modelo mais comum de robôs devido sua facilidade de operação e desempenho em terrenos regulares, mas o uso de rodas se torna inviável em terrenos irregulares como nos grandes centros urbanos, subir calçadas e escadas além de passar por ambientes acidentados, fazendo com que os robôs com pernas demonstrem um melhor desempenho. Essa diferença é explicada por ([CARBONE; CECCARELLI, 2005](#)) que diz que a diferença entre os robôs com pernas e robôs com rodas é que a primeira configuração não apresenta contato contínuo com o solo fazendo com que quando uma perna está levantada para se locomover o robô possa contar com as demais para sustentar o peso do seu corpo auxiliando na locomoção em terrenos desnivelados ou com degraus. Além disso os robôs com pernas podem ser utilizados para outras funcionalidades como no transporte de objetos e apresentam tolerância a falhas, caso uma das pernas

esteja danificada o robô pode continuar se locomovendo, diferente dos robôs com rodas que caso uma das rodas seja danificada a locomoção do robô se torna inviável (SPENNEBERG; MCCULLOUGH; KIRCHNER, 2004).

Pensando no problema de se realizar entregas em um ambiente urbano, os robôs com pernas se tornam mais promissores para a utilização de entregas de pequenos objetos. Entretanto, dentro da navegação autônoma existem alguns problemas clássicos em torno da locomoção do robô em um ambiente, entre elas tem-se o planejamento de caminho que é uma das mais importantes pois se faz necessária a definição de uma rota ótima para a locomoção do robô de um ponto inicial até um ponto final sem colidir com obstáculos.

Na literatura existem inúmeros algoritmos que buscam solucionar o problema de planejamento de caminho, como por exemplo os algoritmos clássicos de Dijkstra (CORMEN et al., 2009), PRM (*Probabilistic Roadmap* (KAVRAKI et al., 1996), RRT(*Rapidly-Exploring Random Tree*) (ZHANG et al., 2018) e A* (WARREN, 1993), entretanto, esses algoritmos apresentam uma baixa eficácia quando se trata da representação de ambientes reais onde se tem muitos obstáculos, além disso o processo de planejamento com esses algoritmos acaba sendo muito demorado, demandando um longo tempo de espera para a determinação da rota.

Uma alternativa aos modelos tradicionais e utilizados para o planejamento de rota é o *Q-learning*. Essa abordagem trata-se de uma técnica de Aprendizado por Reforço que busca encontrar a melhor solução através de um sistema de exploração e um esquema de recompensas que indicam as melhores ações a serem tomadas pelo robô no ambiente. Entretanto, o algoritmo de *Q-learning* acaba sendo uma abordagem tão lenta quanto as demais. Isso se dá pelo fato de que o robô precisa explorar o ambiente inúmeras vezes até encontrar uma solução ótima, ou seja, o tempo para a convergência do algoritmo se torna muito grande, tornando essa abordagem ineficiente para ambientes com uma grande densidade de obstáculos.

Para contornar o problema relacionado ao tempo de convergência do algoritmo em busca da solução ótima, o presente trabalho apresenta uma alternativa ao *Q-learning* clássico acelerando o processo de treinamento. A abordagem denominada de *Q-learning* com Dupla Recompensa, ou QLDR. Essa abordagem consiste na utilização de um esquema de dupla recompensa, a recompensa estática recebida imediatamente e uma recompensa dinâmica através da utilização do cálculo da distância euclidiana que avalia o quão longe ou quão perto o robô está em relação ao ponto de destino. Além disso, para validação de conceito foi utilizado um robô hexápode real para se locomover em um ambiente através de um sistema de visão externa e a utilização de algoritmos de processamento de imagem em *Python*, tanto para o planejamento da rota quanto para o processo de locomoção em si.

1.2 Objetivos Gerais

O objetivo principal desse trabalho consiste em propor uma nova abordagem de algoritmo de Aprendizado por Reforço denominado *Q-learning* de Dupla Recompensa, que visa acelerar o processo de treinamento e planejamento de rotas para robôs com pernas, a fim de realizar o transporte de cargas em centros urbanos ou grandes ambientes que possuem terrenos irregulares e desnivelado, e com diversos tipos de obstáculos.

1.3 Objetivos Específicos

- Delinear teoricamente o problema de otimização da locomoção de um agente em um circuito pré-definido;
- Investigar a aplicação dos algoritmos clássicos, principalmente abordagens de *Q-learning*, para a locomoção do robô em um circuito seguindo o melhor caminho possível;
- Construção do algoritmo QLDR responsável por encontrar o melhor caminho do circuito utilizando o esquema de dupla recompensa;
- Validação do algoritmo através de métricas de medição de aprendizagem e comparações com o modelo de *Q-learning* clássico;
- Validar experimentalmente o algoritmo proposto utilizando um robô hexápode real;
- Avaliar os resultados considerando o tempo de aprendizagem para encontrar o melhor caminho, qualidade da locomoção e sucesso ao atingir o objetivo.

1.4 Organização do Trabalho

Os capítulos deste trabalho estão organizados da seguinte forma:

- Capítulo 2: É apresentada a revisão da literatura onde são descritos os problemas recorrentes envolvendo a locomoção de robôs em um ambiente. Tal capítulo aborda também os métodos comumente utilizados para solucionar problemas envolvendo robôs, como por exemplo o Aprendizado por Reforço do qual dá a base para esse trabalho. Além disso, apresenta um levantamento de trabalhos relacionados à resolução do problema abordado nesse projeto.
- Capítulo 3: É apresentada a declaração do problema que se pretende solucionar no trabalho.
- Capítulo 4: Explana todo o processo de desenvolvimento do algoritmo QLDR, aprendizado por reforço *Q-learning* com recompensa dupla.

- Capítulo 5: Detalha os resultados obtidos do aprendizado em diversos ambientes e também a utilização do algoritmo para a resolução de um problema real, que foca encontrar o melhor caminho para um robô percorrer dentro de um ambiente urbano. O capítulo também apresenta um teste de conceito realizado em um ambiente interno de tamanho reduzido.
- Capítulo 6: No último capítulo é apresentada a conclusão do trabalho bem como os problemas encontrados e propostas para a evolução do projeto em trabalhos futuros.

2 Revisão da Literatura

2.1 Fundamentação Teórica

2.1.1 Problemas Fundamentais da Robótica

De acordo com (DUDEK; JENKIN, 2010), a mais simples abstração de um robô autônomo é considerá-lo como sendo um ponto operando em um ambiente cartesiano contínuo. O robô pode ser representado por um ponto em $(x,y) \in \mathbb{R}^2$, onde tal ponto descreve sua posição atual, mais conhecido como localização do robô.

A locomoção de robôs principalmente em um ambiente envolve uma série de processos através de uma mudança de estados, ou seja, transitar de um estado (x,y) para um (z,w) . O robô pode operar sobre todo o plano, porém, nem todas as posições do domínio podem ser acessadas. A validação das posições permitidas para a locomoção do robô no ambiente vem da definição dos estados. Se um estado permite o acesso do robô esse é chamado de zona livre representado por C_{free} , caso contrário, o estado é definido como sendo um obstáculo e é representado por $\mathbb{R} - C_{free}$. Através da definição do espaço existem algumas questões os autores em (DUDEK; JENKIN, 2010) explanam que existem algumas questões clássicas acerca da mobilidade de robôs, dentre elas foram escolhidas três para serem explanadas nas próximas seções, além disso foi incluída uma questão adicional que está relacionada à locomoção de robôs terrestres.

- **Locomoção:** Como é realizada a locomoção do robô em um ambiente?
- **Planejamento de Caminho:** É possível fazer o robô transitar de um estado (x,y) para um (w,z) se mantendo em uma zona C_{free} ?
- **Localização:** Como o robô irá determinar seu estado atual se o mesmo possui medições locais de C_{free} ?
- **Percepção do Ambiente:** Como o robô define quais partes do ambiente estão ocupadas? Isto é, como determinar as posições C_{free} ?

2.2 Locomoção de Robôs

No âmbito da robótica existem diversas vertentes que buscam descrever um comportamento de locomoção, dentre elas está a robótica móvel que almeja aumentar a versatilidade de diversos tipos de equipamentos(SANTOS; JÚNIOR et al., 2012).

Ao longo dos séculos, muitas máquinas e veículos foram criados utilizando tecnologia guiada por rodas. Porém, no caso de veículos existe o empecilho que é a locomoção por

ambientes inóspitos. Tal dificuldade levou ao desenvolvimento da locomoção por trilhos que é comumente utilizada em tanques de guerra. Entretanto, a locomoção por trilhos também tem seus problemas como por exemplo, a destruição do terreno por onde passa e no caso de mau funcionamento em qualquer parte do trilho inviabiliza o movimento do veículo inteiro.

A fim de contornar os problemas relacionados aos tipos de locomoção citados, surge locomoção por pernas ([SHAHRIARI, 2013](#))([MAHAJAN; FIGUEROA, 1997](#)), uma alternativa que tem como objetivo simular movimentos de animais.

O elemento principal que determinam o *design* e a locomoção de um robô é o levantamento das caracterizações do terreno ([SHAHRIARI, 2013](#)) onde o mesmo poderá ser inserido. De acordo com ([HARDARSON, 1998](#)), existem três propriedades que definem a caracterização de um terreno, são estas:

- Propriedades Geométricas: Determina o formato da superfície do terreno como a irregularidade e inclinação.
- Propriedades de Materiais: Incluem características de consistência, rigidez, fricção e densidade do terreno.
- Propriedades Temporais: Consiste na variação das mudanças do terreno.

Levando em conta as características citadas acima, as próximas sessões irão abordar os diferentes tipos de locomoção com robôs terrestres, bem como as vantagens e desvantagens na utilização de cada um.

2.2.1 Locomoção de Robôs com Rodas

A configuração mais comum para os robôs que se locomovem por terra é utilizando a abordagem de locomoção por rodas, isso se deve ao fato da sua facilidade de operação e o desempenho em terrenos regulares ([SANTOS; JÚNIOR et al., 2012](#)).

A abordagem por rodas é atrativa devido ao fato da possibilidade de carregar um maior peso de carga útil ([HOU et al., 2008](#)), uma boa eficiência energética e apresenta um sistema de controle e implementação simples, fazendo com que a simplicidade desse sistema seja vantajosa ao se comparar com outras abordagens. Um exemplo clássico de robôs com rodas são os *Rover* como pode ser visto na [Figura 1](#), tais modelos são muito utilizados pela NASA em viagens de exploração espacial.

Figura 1 – Robô com Rodas - *Rover*



Por mais vantajosa e atrativa que a abordagem de locomoção com rodas seja, a mesma apresenta certas desvantagens ao se deparar com terrenos desnivelados, superfícies macias, obstáculos e buracos maiores que o tamanho das rodas, fazendo com que a locomoção não seja performada de maneira correta.

2.2.2 Locomoção de Robôs com Trilhos

Em terrenos onde suas propriedades são desconhecidas, a locomoção por trilhos se torna uma opção vantajosa, as paletas mantém contato com o chão proporcionando empuxo suficiente para o veículo se mover em superfícies desregulares como por exemplo, na neve ou areia. A [Figura 2](#) mostra um modelo de robô que faz uso da locomoção por trilhos.



Figura 2 – Exemplo de um Robô que utiliza locomoção por Trilhos

Assim como na abordagem por rodas, essa configuração também pode carregar grande quantidade de peso de carga útil, porém, apresenta grande consumo de energia devido a fricção com o solo fazendo com que essa configuração tenha a desvantagem de ser energeticamente inefficiente. Uma outra desvantagem é a inflexibilidade das paletas causando destruição do terreno por onde o veículo percorre.

2.2.3 Locomoção de Robôs com Pernas

Robôs com pernas são apropriados para operarem em vários tipos de terrenos devido ao fato de não manterem contato contínuo com a superfície, pois utilizam pontos de apoio por coordenação para repor as pernas (CARBONE; CECCARELLI, 2005), isto acontece pois os robôs com pernas são capazes de manejar como distribuir o peso sobre todas as pernas em cada passo dado fazendo com que o robô tenha uma suspensão ativa (SHAHRIARI, 2013).

De forma simplificada, os robôs com pernas são compostos por um corpo principal e suas respectivas pernas. As pernas são um conjunto de elementos rígidos com uma ou mais articulações que poderão ou não ser acionadas por atuadores. Os modelos desse tipo de robô são classificados de acordo com o número de pernas que possui, podendo ser do tipo bípedes(duas pernas), como por exemplo robôs humanoides, quadrupedes (quatro pernas), hexápodes (seis pernas) e octópodes(oito pernas) (SANTOS; JÚNIOR et al., 2012).

Em relação à locomoção, as pernas apresentam como finalidade a sustentação e equilíbrio do corpo do robô gerando um impulso que é necessário para o deslocamento do corpo. É importante citar que o grau de estabilidade dos robôs vai aumentando de acordo com o aumento no número de pares de pernas, fazendo com que um robô quadrúpede seja mais estável que um bípede, consequentemente, um hexápode será mais estável que um quadrupede e assim por diante (SHAHRIARI, 2013). Um modelo de robôs com pernas bem conhecido é o *Spot* fabricado pela *Boston Dynamics* que pode ser visto na Figura 3 .

Figura 3 – *Spot*



As desvantagens que podem ser citadas dessa configuração de robôs são o alto consumo energético, o desenvolvimento de complexos modelos cinemáticos, a necessidade de mecanismos dinâmicos e a dificuldade na implementação dos algoritmos tanto de locomoção quanto de localização.

2.2.4 Estabilidade de Robôs

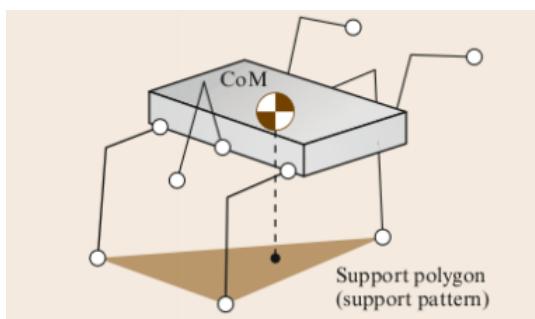
Estabilidade é uma característica fundamental de performance na análise de locomoção terrestre ([TING; BLICKHAN; FULL, 1994](#)). Em um panorama geral quando se fala de robôs existem dois tipos de estabilidade que precisam ser definidas, a estabilidade estática e a estabilidade dinâmica.

2.2.4.1 Estabilidade Estática

A estabilidade estática está relacionado ao fato do robô se manter em equilíbrio sem que haja alguma força ou movimento adicional, ou seja, o robô deve ser capaz de caminhar a uma velocidade constante em uma direção constante por si só ([GARCIA; ESTREMERA; SANTOS, 2002](#)).

Esse critério foi primeiramente definido por McGhee e Frank em 1968, onde o método adotado cita que um robô é estaticamente estável se a projeção horizontal do seu centro de massa está sobre um polígono de suporte ([MCGHEE; FRANK, 1968](#)). Mais tarde, esse critério foi estendido para terrenos desregulares redefinindo o polígono de suporte formado pela projeção horizontal das patas do modelo de suporte real ([MCGHEE; ISWANDHI, 1979](#)) ([RAIBERT, 1986](#)).

Figura 4 – Centro de Massa Sobre um Polígono



Fonte: *Design, Implementation and Control of a Hexapod Robot Using Reinforcement Learning Approach* ([SHAHRIARI, 2013](#))

De acordo com a [Figura 4](#) acima, os contatos da perna realizam o apoio das mesmas no solo enquanto o centro de massa do robô se encontra dentro do padrão do suporte, tornando-se então estaticamente estável.

2.2.4.2 Estabilidade Dinâmica

Estabilidade dinâmica em robôs com pernas, se refere a operação em regime permanente onde as velocidades e energias cinéticas das massas são determinantes para o comportamento ([RAIBERT, 1986](#)). A estabilidade dinâmica então se faz necessária todas as vezes que o centro de massa do modelo não está sobre o suporte padrão. Nesses casos, o robô poderá cair quando

estiver em regime permanente, ou seja, quando nenhum movimento ou forças são aplicadas para se manter estável (SHAHRIARI, 2013) (LEE; LIAO; CHEN, 1988) (CLARK et al., 2001).

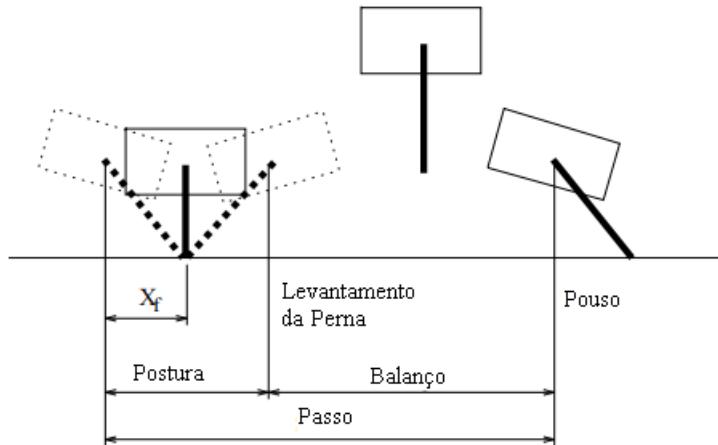
Para que uma locomoção seja feita de maneira estável, os robôs devem mover uma perna por vez enquanto as demais sustentam o corpo. Assim, um robô hexápode se torna mais estável do que um quadrúpede devido a uma maior quantidade de pernas disponíveis para o suporte à estabilidade estática (GARCIA; ESTREMERA; SANTOS, 2002).

2.2.5 Velocidade de Locomoção dos Robôs com Pernas

Uma das principais características em locomoção é a marcha. A marcha segundo (CARBONE; CECCARELLI, 2005) é um padrão característico de locomoção para pequenos intervalos de velocidade descrito pelas quantidades de transições descontínuas entre os movimentos.

O movimento de cada uma das pernas pode ser dividido em duas fases como mostra a Figura 5 . A primeira fase corresponde a fase de postura onde a perna fornece suporte e propaga por todo corpo, a segunda é a fase de balanço que corresponde a fase em que a perna levanta do chão e realiza um balanço para frente com o intuito de executar um próximo passo, iniciando assim uma outra fase de postura ao posar a perna no chão (BEER et al., 1997).

Figura 5 – Fases de Postura e de Balanço



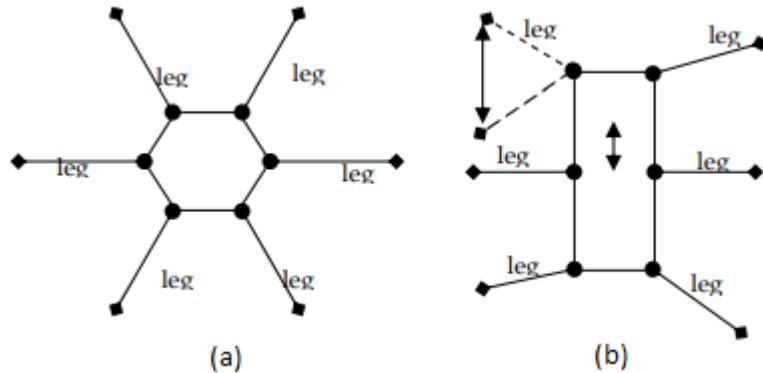
Fonte: O autor adaptado de *Design, Control, and Energetics of an Electrically Actuated Legged Robot* (GREGORIO; AHMADI; BUEHLER, 1997)

2.2.6 Configurações de Robôs Hexápodes

Como o presente trabalho utiliza um robô com seis pernas é importante citar as configurações de robôs hexápodes existentes. Há dois tipos de configurações de *design* para robôs hexápodes, retangular e hexagonal como mostra a Figura 6 . A abordagem retangular mostrada do lado direito da figura é baseada em insetos, apresentando suas seis pernas divididas em dois lados do seu corpo, três do lado direito e três do lado esquerdo. Já a abordagem hexagonal que

está representada do lado esquerdo da figura é composta por uma distribuição assimétrica das pernas em torno do corpo circular.

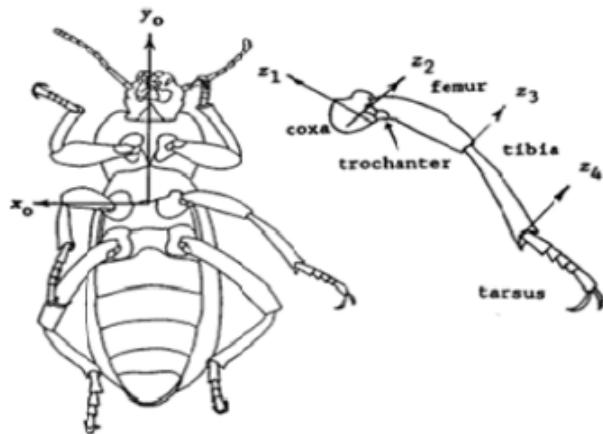
Figura 6 – Modelos Típicos de Robôs Hexápodes



Fonte: *Dynamics And Measuring Of Feet Force Distributions Of Six-Legged Robot* ([BADAWY; ZAGHLOUL; HUSSEIN, 2016](#))

Cada uma das pernas em ambos os modelos de robôs hexápodes pode ter de dois a seis graus de liberdade inspirado no comportamento biológico dos insetos. A [Figura 7](#) mostra a estrutura da perna de um inseto qualquer, tal estrutura serve como inspiração para o *design* de pernas robóticas.

Figura 7 – Estrutura de Pernas de Insetos



Fonte: ([SHAHRIARI, 2013](#))

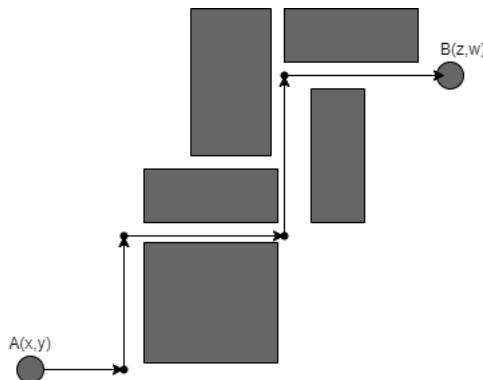
Robôs de formato retangular são mais rápidos e ágeis em se moverem para frente, entretanto não apresentam uma flexibilidade ao se moverem para outras direções em relação ao modelo hexagonal que apresentam uma distribuição assimétrica.

Em respeito a questão de estabilidade estática ambas as configurações compartilham do mesmo conceito, ou seja, as pernas que suportam o polígono apresentam a projeção horizontal do centro de massa do robô dentro dessa área([SHAHRIARI, 2013](#)).

2.3 Planejamento de Caminho

Um dos problemas mais importantes e conhecidos quando se trata da navegação de robôs autônomos (robôs artificialmente inteligentes) é o planejamento de caminho ou localização de caminho. A navegação é um processo de planejamento e direcionamento de rota ou caminho, tal tarefa deve ser exercida pelo robô a fim de fazer com que venha se mover com segurança de um ponto inicial $A(x,y)$ a um ponto final $B(w,z)$. Isto é, o objetivo de planejar a rota do robô é selecionar o caminho que seja adequado para se percorrer sobre o ambiente representado por uma curva contínua livre de obstáculos ([DUDEK; JENKIN, 2010](#)) conforme mostra a [Figura 8](#).

Figura 8 – Planejamento de caminho entre os pontos $A(x,y)$ e $B(w,z)$



Fonte: O Autor

Segundo ([SARIFF; BUNIYAMIN, 2006](#)), existem dois componentes importantes para o planejamento do caminho que é a representação do ambiente em que o robô está inserido por uma configuração C_{space} e a implementação de um algoritmo que seja capaz de encontrar um caminho ótimo para o robô percorrer. Os dois conceitos possuem forte correlação pois um afeta o outro ao definir o melhor caminho que movimente o robô no ambiente em um tempo ótimo.

Existem quatro tipos de configuração de ambiente como os diagramas de *Voronoi* ([DONALD, 1984](#)), representação por grades (matricial) ([NIE et al., 2021](#)), grafos ([YU; LAVALLE, 2016](#)) e árvores quádruplas ([KAMBHAMPATI; DAVIS, 1986](#)) onde o C_{space} está preenchido com estruturas de dados que indicam posições e orientações dos objetos no ambiente, incluindo as regiões C_{free} e as que possuem obstáculos.

A seleção do algoritmo adequado para o tipo de configuração de ambiente definirá o quanto complexo será o processo de planejamento do caminho. Além disso, a complexidade da

solução para esse problema é da classe NP-difícil, ou *NP-hard*, o que explica a crescente busca por novos algoritmos que confrontam as soluções clássicas para a resolução desse problema (PURCARU et al., 2013).

Existem diversos tipos de algoritmos que buscam resolver o problema do planejamento de caminhos e esse número aumenta com o avanço dos estudos e evolução da tecnologia. Contudo, existem alguns modelos clássicos que serão explanados a seguir:

- **Dijkstra** : Apresenta a representação de C_{space} por grafos onde cada um dos nós ficam dispostos por pesos em suas arestas. O algoritmo é capaz de encontrar o melhor caminho para se percorrer entre $A(x,y)$ e $B(y,z)$ procurando pelo caminho cujo o somatório dos valores das arestas seja mínimo. O algoritmo inicia em $A(x,y)$ procura por seus nós adjacentes e seleciona o de menor valor fazendo com que o algoritmo passe para o próximo nó onde este repetirá o processo anterior até que o agente chegue em $B(y,z)$ (CORMEN et al., 2009).
- **PRM (*Probabilistic Roadmap*)**: É definida como uma rede de grafos com possíveis caminhos em um ambiente através da geração livre de nós aleatórios que são conectados com o intuito de formar um mapa de rotas. Para que o mapa seja construído no ambiente ao invés de analisar todas as configurações possíveis é feita a seleção aleatória através da geração de um nó e a verificação se o mesmo está ou não em C_{free} , em caso positivo o mesmo é agregado à rede. Com a rede gerada, se pode estabelecer uma rota entre os pontos inicial e final gerando o melhor caminho (KAVRAKI et al., 1996).
- **RRT (*Rapidly-Exploring Random Tree*)**: É definido como sendo um algoritmo capaz de construir uma árvore onde os ramos indicam os caminhos possíveis a partir do nó representado pelo estado inicial $A(x,y)$. Em um espaço contínuo C iniciando em $A(x,y)$ a cada iteração é escolhido um estado aleatório $x_{rand} \in C_{free}$ caso o novo estado não seja um obstáculo, o algoritmo procura por um vértice existente na árvore que se aproxima desse novo estado, ou seja, escolhe o vértice que irá minimizar x_{near} e x_{rand} . O processo inteiro é realizado até que seja identificado um nó x_{rand} que seja igual a $B(x,y)$ e o mesmo possa ser inserido na árvore. Assim o algoritmo consegue encontrar um caminho ótimo entre os pontos inicial e final. (LAVALLE et al., 1998) Uma vantagem de se utilizar esse tipo de algoritmo é devido a sua alta escalabilidade em ambientes de grandes dimensões além da sua efetiva análise de detecção de obstáculos. Porém o RRT converge lentamente devido ao processo de busca uniforme para o espaço de configuração, além disso o algoritmo e as suas mais variadas versões não resolvem bem um problema de planejamento ao se deparar com um ambiente onde existem passagens estreitas (ZHANG et al., 2018).
- **A***: É um dos algoritmos mais eficientes em termos de distância entre o estado inicial e o estado final. O processo é inicializado com uma grade que indica se cada uma das células estão em C_{free} ou não. Após isso a busca pelo caminho começa pelo estado inicial $A(x,y)$

e procura em suas células adjacentes qual é a que tem o menor custo, ao identificar o estado de menor custo essa célula passa a ser o estado atual, além disso esse estado é adicionado a uma lista que indicam as células de menor custo no percurso, ou seja, são as células que representam os estados no melhor caminho. Esse processo é realizado a cada iteração até alcançar o estado final $B(w,z)$ (WARREN, 1993).

Além dos algoritmos clássicos citados acima, existem outras quatro abordagens que propõem a solução para o problema do planejamento de caminho. De acordo com (ZHAO et al., 2020) os algoritmos podem ser divididos em quatro tipos, Lógica *fuzzy*, Otimização Inteligente, SLAM e por último Aprendizado por Reforço que é o método utilizado nesse trabalho.

Os algoritmos de lógica *fuzzy* são implementados com o intuito de se construir uma base de dados que simula diferentes cenários para um determinado estado do robô. Esse tipo de algoritmo é utilizado quando se deseja percorrer um caminho seguro quando o robô está se movimentando em um ambiente desconhecido como é citado no estudo de (WANG et al., 2005) onde vários cenários são testados a fim de comprovar que o robô realiza o percurso sem colisões.

Ainda de acordo com (ZHAO et al., 2020), algoritmos de otimização incluem algoritmos genéticos, algoritmos imunes, algoritmos de colônia de formigas e algoritmos que simulam algum fenômeno natural ou o comportamento biológico de alguns grupos. Tais algoritmos trazem uma grande otimização quanto sua habilidade em simular a navegação de um robô.

O algoritmo de SLAM, em inglês, *Simultaneous Localization and Mapping* foi proposto inicialmente por (SMITH; CHEESEMAN, 1986) onde sensores presos a um robô forneciam dados externos a fim de fazer com que o algoritmo pudesse trazer uma estimativa do estado de movimentação atual do robô possibilitando identificar não só a posição do robô mas também possíveis obstáculos em tempo real. Entretanto, (ZHAO et al., 2020) ressalta que quando o robô planeja um caminho em um ambiente desconhecido com SLAM são utilizados vários cálculos de correspondência para realizar a localização e o mapeamento trazendo uma maior precisão do posicionamento.

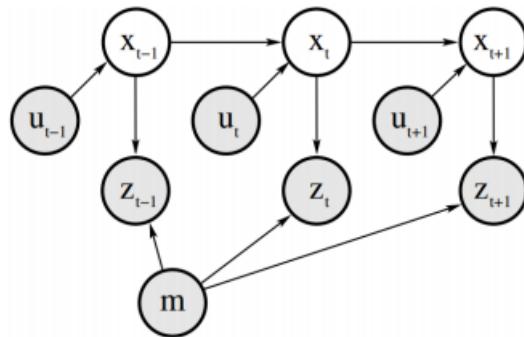
Outro tipo de algoritmo comumente utilizado é o de Aprendizado por Reforço, que se sustenta na base de várias tentativas até encontrar a solução sem a necessidade do fornecimento de qualquer dado prévio acerca do ambiente. O modelo clássico de *Q-Learning*, que é o utilizado no presente trabalho, vem sendo utilizado em diversos estudos no mundo inteiro devido à sua fácil implementação. Tal abordagem será explanada com mais detalhes na seção 2.6.

2.4 Localização

Uma dificuldade relacionada à locomoção de robôs autônomos se dá na localização do mesmo ao se locomover de um ponto (x,y) até um ponto $(x+\delta x, y+\delta y)$, ou seja, como saber

onde o robô está agora? (DUDEK; JENKIN, 2010). O problema da localização pode ter sua representação gráfica conforme a Figura 9, onde está representado um mapa m do ambiente e é preciso saber a cada momento qual a pose (posição e orientação) do robô em relação ao mapa de acordo com as suas percepções z e seus controles u (MONTEIRO et al., 2020). O problema principal que surge é que devido ao fato de muitos sistemas robóticos não apresentarem sensores de ruído que são capazes para determinar a pose, se faz necessária uma abordagem de localização externa a partir das percepções do ambiente.

Figura 9 – Representação Gráfica para Localização de Robôs



Fonte: (THRUN, 2002)

Uma das formas de obtenção de dados externos sobre a localização do robô no mapa, se faz através do apoio de um sistema de visão computacional que localiza o robô no ambiente. Caso o ambiente permita que o robô seja identificado e localizado é possível então se obter uma pose absoluta constantemente através de análises suscetivas de imagens do meio. Esse processo de medição de pose faz com que cada quadro de imagem possa fornecer coordenadas sem qualquer outra suposição, funcionando assim como uma espécie de GPS(*Global Positioning System*) em pequena escala (FIALA, 2004).

Para que a detecção do robô seja realizada com êxito a tarefa recai sobre as imagens da câmera, o sucesso na obtenção dos dados pode ser por meios passivos ou ativos. O meio ativo pode ser através de LEDs que enviam códigos seriais na medida que piscam (MATSUSHITA et al., 2003). Já os meios passivos podem ser utilizados através de um padrão único e distingível empregado no ambiente, como por exemplo, a utilização de marcadores com cores vibrantes colocados no robô como implementado no trabalho de (RUZZON, 2019).

Assim, a obtenção das imagens do ambiente facilita no mapeamento das coordenadas do robô das quais podem ser utilizadas como uma componente de medição em um sistema controlado orientando o robô ao longo do percurso escolhido.

2.5 Percepção

A percepção é uma informação primordial para a locomoção de robôs como por exemplo, a identificação de obstáculos, um risco de queda ou colisão além de poder estimar a pose do robô, medir distâncias e auxiliar na construção de mapas (MONTEIRO et al., 2020)(DUDEK; JENKIN, 2010).

A visão humana é baseada na transformação da informação recebida transformada de luz a sinais elétricos que são transmitidos pelos neurônios onde essa informação é processada e recebida pelo cérebro auxiliando na percepção do ambiente (DUDEK; JENKIN, 2010). Se pode então fazer uma analogia da visão como um sensor para os robôs visto que esse componente realiza a detecção de um estímulo físico ou químico e o converte em um sinal analógico enviando informações acerca de alterações do ambiente ao robô.

Visto que a visão humana está ligada à percepção, é natural estender a visão como um sensor de percepção para robôs, onde o sensor é representado por câmeras responsáveis por colher imagens do ambiente e gerar informações sobre o caminho que o robô deve seguir (DUDEK; JENKIN, 2010).

De acordo com (GONZALEZ; WOODS et al., 2002), uma imagem pode ser definida como uma função bi-dimensional, $f(x,y)$, onde x e y são coordenadas espaciais e a amplitude da função f em qualquer um dos pares de coordenadas é chamada de *intensidade* ou *nível de cinza* da imagem em determinado ponto. A partir do momento em que os pares x e y de uma imagem são finitos, ou seja, apresentam quantidades discretas, a imagem então é chamada de imagem digital.

Cada par x e y de uma imagem digital é chamado de elemento de imagem ou *pixel*, sendo assim uma imagem pode ser representada por uma matriz de *pixels* com M linhas e N colunas, e o valor da função no local de determinado pixel é proporcional ao nível de cinza naquele ponto, como pode ser observado na Figura 10

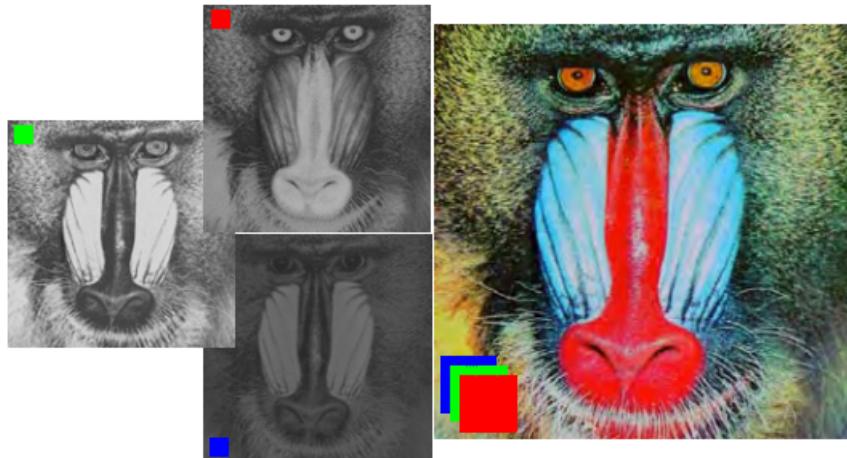
Figura 10 – Representação Matricial de Uma Imagem



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

No caso de imagens digitais coloridas, cada pixel é visto como um vetor onde as componentes representam não mais a intensidade de cinza, mas sim as intensidades do sistema RGB (vermelho, verde e azul). De maneira semelhante à imagens em preto e branco a [Figura 11](#) demonstra a representação de uma imagem colorida dividida em três matrizes de pixels que quando sobrepostos representam a imagem digital.

Figura 11 – Representação Matricial de Uma Imagem Colorida



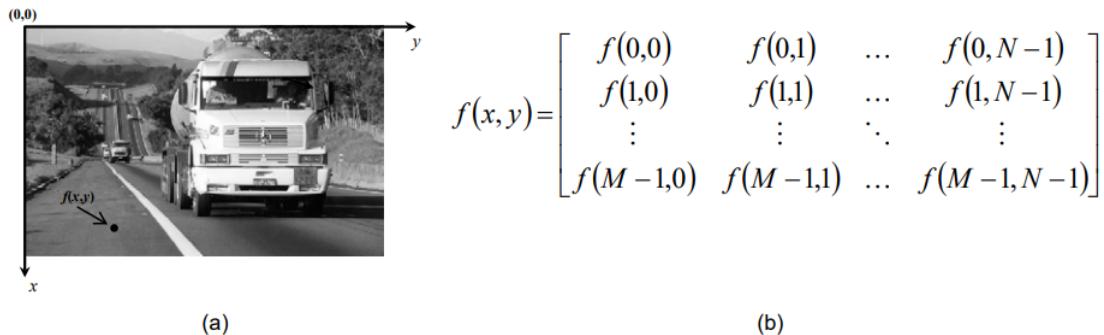
Fonte: ([QUEIROZ; GOMES, 2006](#))

2.5.1 Amostragem e Quantização

Para que uma imagem possa ser processada digitalmente se faz necessária a realização de dois processos de discretização em valores inteiros positivos. O primeiro é feito em nível de coordenadas espaciais (x e y) chamado de amostragem, já o segundo processo é feito em nível de escala de cinza ([GONZALEZ; WOODS et al., 2002](#)) ([QUEIROZ; GOMES, 2006](#)).

Ambos processos resultam na amostragem da imagem digital em pontos distribuídos em uma matriz $f(x,y)$ de dimensão $M \times N$, onde cada elemento representa uma aproximação do nível de cinza da imagem no ponto amostrado como mostra a [Figura 12](#).

Figura 12 – Representação digital de uma imagem: (a) Convenção de eixos e (b) Convenção matricial



Fonte: ([CUNHA, 2013](#))

2.5.1.1 Amostragem

No processo de amostragem, a qualidade da imagem digital irá depender do tamanho da matriz de representação, ou seja, quanto maior a matriz melhor será a qualidade dessa imagem como pode ser visto na [Figura 13](#)

Figura 13 – Qualidade da imagem no processo de amostragem



Fonte: ([CUNHA, 2013](#))

2.5.1.2 Quantização

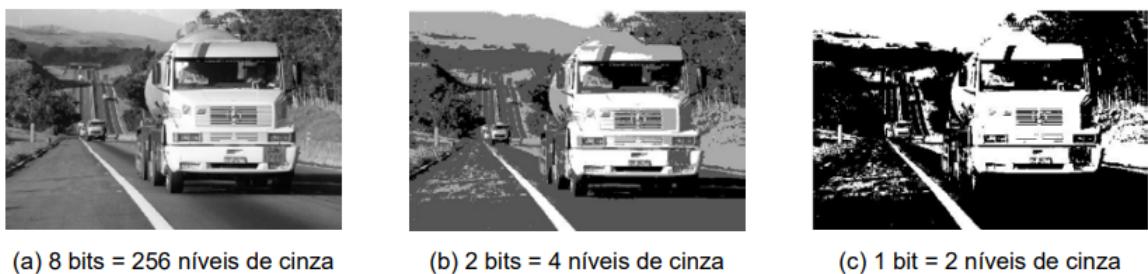
De acordo com ([GONZALEZ; WOODS et al., 2002](#)) a quantização é a representação digital da amplitude da imagem, também conhecido como a escala de variação da intensidade luminosa, definida por valores reais e positivos. Esse processo está diretamente relacionado ao espaço de memória necessário para o armazenamento de uma imagem. Para definir os níveis de

brilho (L) de uma imagem é comumente aplicada a quantização como potências de 2:

$$L = 2^x, x \in \mathbb{N} \quad (2.1)$$

De acordo com a equação acima, x representa a resolução da imagem em *bits*. Assim, uma imagem com resolução de 8 *bits* fornece 256 níveis de cinza, uma imagem com resolução de 2 *bits* fornece 4 níveis de cinza e uma imagem com 1 *bit* de resolução fornece apenas 2 níveis de cinza como pode ser visto na [Figura 14](#).

Figura 14 – Exemplo de processo de quantização de uma imagem



(a) 8 bits = 256 níveis de cinza

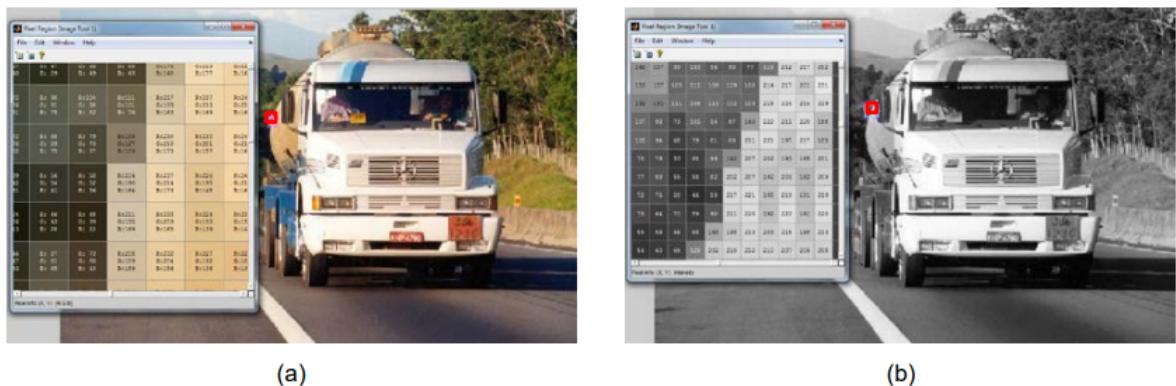
(b) 2 bits = 4 níveis de cinza

(c) 1 bit = 2 níveis de cinza

Fonte: ([CUNHA, 2013](#))

Como já mencionado, em uma imagem digital colorida representada por uma matriz de pixels, cada pixel armazena três valores que definem a intensidade do brilho naquele ponto amostrado. Enquanto uma imagem em preto e branco realiza a quantização para no máximo 256 níveis, uma imagem no padrão RGB consegue quantizar 16 milhões de cores diferentes, isso se dá ao fato de se ter três planos de 8 bits sobrepostos. Tal diferença entre a quantização de imagens em preto e branco e imagens coloridas pode ser vista na [Figura 15](#).

Figura 15 – Intensidade luminosa dos pixels na região em destaque na imagem: (a) colorida e (b) preto e branco



Fonte: ([CUNHA, 2013](#))

Vale citar que o número de *bits* necessários para representar uma imagem digital está diretamente relacionado entre o processo de amostragem e quantização. A Equação 2.2 representa o cálculo para o número de bits necessários para representar uma imagem, sendo M e N representando as dimensões da matriz de amostragem e l a resolução da imagem (QUEIROZ; GOMES, 2006).

$$b = M \times N \times l \quad (2.2)$$

2.5.2 Segmentação

Segundo (ESQUEF; ALBUQUERQUE; ALBUQUERQUE, 2003) segmentar uma imagem é o ato de separar uma imagem digital como um todo nas partes que a constituem que se diferenciam entre si, ou seja, é o processo de separação dos pixels da imagem em grupos de interesse ou que fornecem alguma informação para o processamento.

A segmentação é um processo empírico e adaptativo que busca se adequar de acordo com às características intrínsecas de cada imagem e ao objetivo que se queira alcançar.

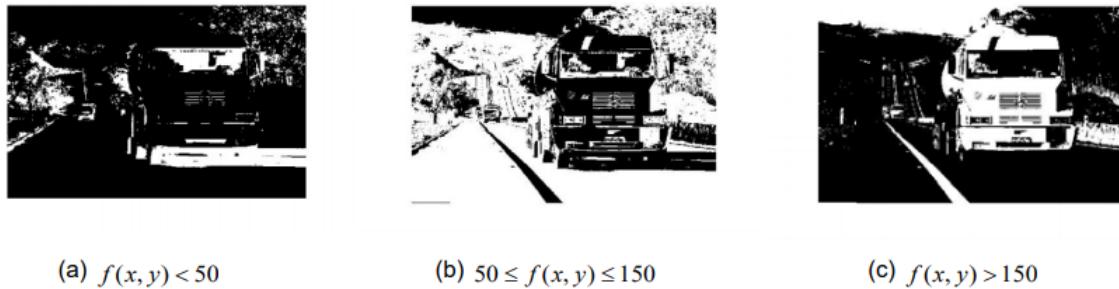
2.5.2.1 Segmentação Binária (*Thresholding*)

Um método bem conhecido de segmentação é a segmentação binária, ou *thresholding* que tem como objetivo transformar uma imagem com vários níveis de cinza em uma imagem binária (FARIA, 2010). O intuito dessa operação é rotular todos os pixels de uma imagem comparando-os a um valor limiar de *threshold* (T) de acordo com a Equação 2.3 .

$$T(x, y) = \begin{cases} 0, & \text{se } f(x, y) < \tau \\ 1, & \text{se } f(x, y) \geq \tau \end{cases} \quad (2.3)$$

Cada pixel da imagem é representado por $f(x,y)$ que por sua vez é comparado a um valor limiar τ , se o valor de cinza do pixel for menor que o limiar, a função binária $T(x,y)$ recebe um rótulo com o valor zero indicando que o pixel está mais próximo da cor preta e caso contrário, o pixel da imagem em $T(x,y)$ recebe o valor 1 indicando que o pixel está mais próximo da cor branca (CUNHA, 2013). A resultante da função *threshold* é uma imagem binária como pode ser visto na Figura 16 .

Figura 16 – Exemplos de imagem binária

(b) $50 \leq f(x,y) \leq 150$ (c) $f(x,y) > 150$ Fonte: ([CUNHA, 2013](#))

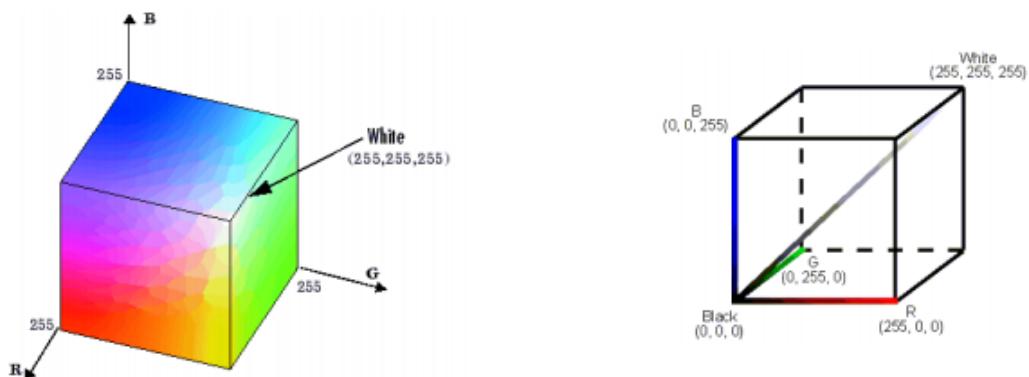
2.5.3 Sistema HSV de Cores

De acordo com ([RUZZON, 2019](#)) um sistema de cores é a forma de parametrização de um *pixel* para que o processamento computacional venha performar a cor desejada com êxito.

Existem vários sistemas de cores que visam especificar o espectro de uma imagem, os mais comuns são o RGB, HSV e o YCrCb.

De acordo com ([ARAUJO; MENDONÇA; FREIRE, 2008](#)) o modelo RGB, do inglês *Red, Green e Blue* é composto pelas cores primárias em que estão associadas a eixos ortogonais fazendo com que o espectro seja representado através de três componentes vetoriais como mostra a [Figura 17](#).

Figura 17 – Sistema de cor RGB

Fonte: ([MATHWORKS, 2012](#))

Entretanto, esse modelo não representa de melhor maneira a percepção humana das cores além disso as informações de cada cor está fortemente atrelada à informações de luminância, dificultando a segmentação quando a imagem não apresenta uma iluminação uniforme.

Para contornar o problema da segmentação de uma imagem independente da luminosidade, é preciso utilizar o sistema HSV para identificação das cores, que segundo ([PENHARBEL](#)

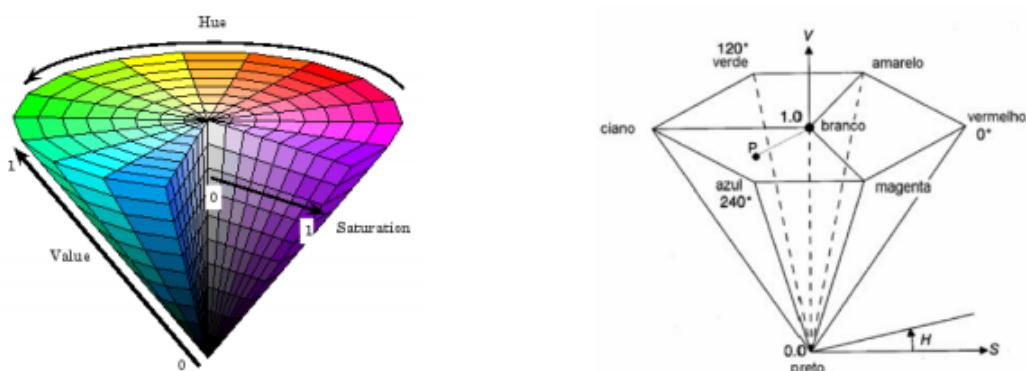
et al., 2004) é mais robusto que o RGB e menos suscetível a falhas.

O sistema HSV, *Hue*, *Saturation* e *Value* utiliza conceitos de crominância que podem ser definidos respectivamente como:

- **Matiz:** tonalidade da cor;
- **Saturação:** pureza da cor;
- **Valor:** brilho da cor;

Esse sistema apresenta representação tridimensional semelhante à uma pirâmide invertida de seis lados(ARAUJO; MENDONÇA; FREIRE, 2008). O eixo vertical da pirâmide representa a componente de valor, na base invertida se encontra a representação da componente matiz definida por um ângulo de 0 a 360, e o eixo horizontal representa a componente de saturação como pode ser visto na [Figura 18](#).

Figura 18 – Sistema de cor RGB



Fonte: ([MATHWORKS](#), 2012)

2.6 Aprendizado de Máquina

Segundo (MONARD; BARANAUSKAS, 2003) Aprendizado de Máquina, ou, *Machine Learning* é uma área de Inteligência Artificial que tem como objetivo o desenvolvimento de técnicas computacionais sobre o aprendizado, bem como a construção de sistemas capazes de adquirir conhecimento de maneira automática. Um dos métodos de AM é o Aprendizado por Reforço, do inglês *Reinforcement Learning* que será abordado nessa seção apresentando definições e conceitos introdutórios.

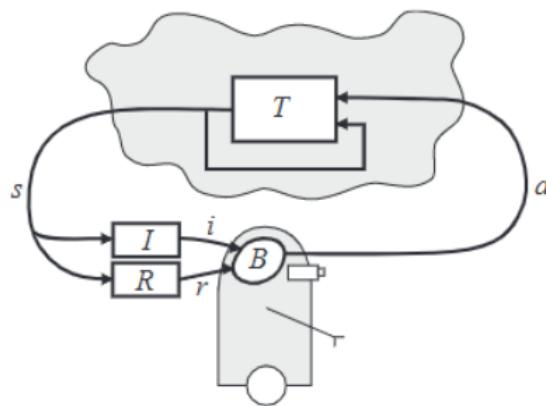
2.6.1 Aprendizado por Reforço

O conceito de Aprendizado por Reforço aparece desde o início do mundo cibernetico e trabalhos em estatística, psicologia, neurociência e ciência da computação. A proposta desse

método é sedutora - uma maneira de programar agentes através de recompensas e punições sem que haja a necessidade de especificar como o objetivo será alcançado. Segundo (KAELBLING; LITTMAN; MOORE, 1996) AR é o problema encontrado por um agente que precisa aprender um comportamento através de suscetivas interações de tentativa e erro em um ambiente dinâmico.

Em um modelo padrão, um agente é conectado com o ambiente através de percepção e ação como mostra a [Figura 19](#).

Figura 19 – Modelo Padrão de Aprendizado por Reforço



Fonte: (KAELBLING; LITTMAN; MOORE, 1996)

Em cada interação, o agente recebe uma entrada, i , e uma indicação do estado atual, s , proveniente do ambiente T . O agente então realiza uma ação, a , a fim de gerar uma saída. A ação muda o estado do ambiente, e o valor dessa transição é comunicado ao agente através de um escalar chamada sinal de reforço, r . O comportamento do agente, B , deve ser caracterizado por ações que apresentam uma maior tendência a longo prazo de maximizar a soma de todas as recompensas recebidas .

O modelo de AR se baseia na premissa de modelar situações onde é preciso executar várias ações em sequência em um ambiente incerto como por exemplo, a escolha da melhor trajetória em um percurso desviando de possíveis obstáculos. Uma situação pode ser modelada através do Processo de Decisão de *Markov* (MDP - *Markov Decision Process*) que envolve um sistema com vários estados e ações que modificam os estados do sistema e observam a tomada das ações com intuito de maximizar a recompensa esperada (PELLEGRINI; WAINER, 2007). Um modelo MDP modela etapas onde as transições entre os estados são probabilidades e podem sofrer alterações dependendo do estado que se encontra através da escolha das ações, no qual gera uma recompensa.

Modelos markovianos apresentam a propriedade de que a distribuição de probabilidade

do próximo $s+1$ depende somente do estado atual s dispensando a análise dos estados predecessores $s-1$ para a determinação de estados futuros ([CASSANDRAS; LAFORTUNE, 2009](#)) ([MONTEIRO et al., 2020](#)). A [Equação 2.4](#) demonstra essa premissa.

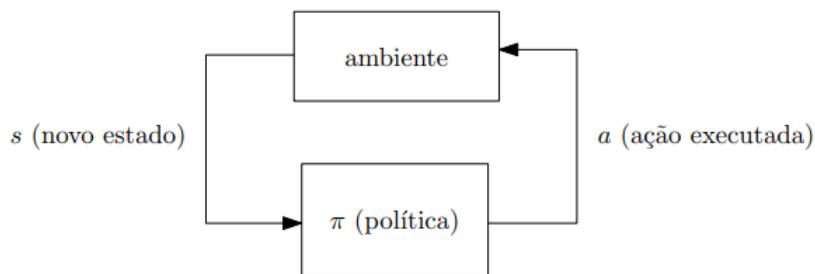
$$p(q + 1 = s + 1 | q = s, \dots, q_0 = s_0) = p(q + 1 = s + 1 | q = s) \quad (2.4)$$

O MDP pode ser definido como sendo uma tupla (S, A, R, T) , que pode ser definida como:

1. Um conjunto finito de estados S que descreve a situação atual;
2. Um conjunto de ações A que o agente realizará em cada estado.
3. Uma função de recompensa $R : S \times A \rightarrow \mathbb{R}$ dada através da execução de uma ação $a \in A$ que resulta em um estado $s' \in S$;
4. Uma função de estado de transição $T : S \times A \rightarrow \Pi(S)$ que define o próximo estado $s' \in S$. Onde um membro de $\Pi(S)$ é a distribuição da probabilidade sobre o conjunto S , ou seja, é a probabilidade do sistema passar de um estado $s \in S$ para um estado $s' \in S$ através da execução de uma ação $a \in A$. ([KAEELBLING; LITTMAN; MOORE, 1996](#)) ([PELLEGRINI; WAINER, 2007](#)).

Em cada época do processo de decisão markoviano pressupõe que o agente está em um estado (s), então o tomador de decisões irá verificar qual é o estado atual s , consulta uma política (π) e executa uma ação (a) conforme mostra a [Figura 20](#). Tal ação pode causar um efeito no ambiente e modificar o estado atual para um estado s' fazendo com que o tomador de decisões verifique esse novo estado e reinicie o processo ([PELLEGRINI; WAINER, 2007](#)).

Figura 20 – Funcionamento de um sistema modelado como MDP



Fonte: ([PELLEGRINI; WAINER, 2007](#))

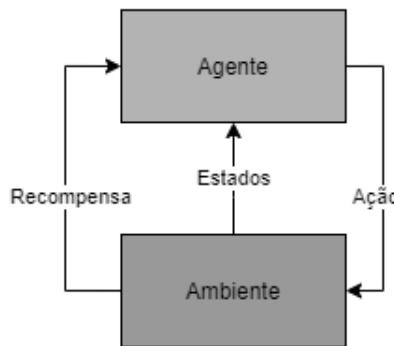
De acordo com ([PELLEGRINI; WAINER, 2007](#)) o MDP sempre tenta encontrar uma política que otimize certos critérios para o desempenho das ações. Uma política pode ser geral,

se as regras de decisão se aplicam a todos estados do MDP ou parcial se alguma das regras valem apenas para alguns estados.

Além disso, uma política pode ser definida como determinística quando cada estado é mapeado sobre apenas uma ação, por exemplo, um robô se move apenas para frente independente do estado. Caso contrário a política é não-determinística, ou seja, para um estado é mapeado um conjunto de ações onde para cada ação é definida uma probabilidade de escolha. Para esse caso, a decisão é definida como sendo $\mathcal{S} \times \mathcal{A} \rightarrow [0,1]$.

Ao executar uma política, o tomador de decisões recebe uma recompensa para cada decisão tomada, tal recompensa é o que norteia os algoritmos de AR, pois as mesmas podem ser positivas em caso da ação tomada resultar em um estado s' válido e negativas caso a ação leve a um estado inválido. Assim, o objetivo dos algoritmos por reforço é fazer com que o agente aprenda a política do ambiente mapeando os estados e ações que otimizam as recompensas (HUNG; GIVIGI, 2016). A aplicação do MDP para AR pode ser representada na [Figura 21](#).

Figura 21 – Processo de Decisão de Markov em Aprendizado por Reforço



Fonte: O Autor

2.6.1.1 Q-Learning

A partir dos elementos definidos por um MDP um algoritmo de AR deve aprender a política ótima que irá trazer o maior benefício a longo prazo. Nesse contexto, se assume que o agente não apresenta um modelo para executar suas ações, ou seja, a política para cada estado é não determinística dando liberdade ao agente explorar as ações disponíveis.

O algoritmo de Q-learning é um dos métodos clássicos de AR. Apresentando referências da psicologia comportamental, nenhuma informação prévia é necessária para que o agente aprenda a política do ambiente, tal informação é recebida gradualmente ao longo do processo de treinamento. A ideia desse algoritmo é de estimar incrementalmente um valor $Q^*(s,a)$ para cada par estado-ação (s,a) , atribuindo assim um valor de recompensa esperada de longo prazo que é obtida através da transição entre os estados (HUNG; GIVIGI, 2016).

Seguindo o modelo MDP, existem quatro elementos importantes para o processo de *Q-learning*, sendo eles: a tabela Q; o conjunto de estados \mathcal{S} ; o conjunto de ações \mathcal{A} e os valores de recompensa \mathcal{R} .

Quando esse método é utilizado para o planejamento do caminho de robôs, onde o robô é o agente, os estados são definidos como sendo coordenadas no ambiente, as ações podem ser definidas como as direções em que o robô pode se mover e a recompensa estará relacionada com as propriedades dos estados que o robô alcança. Por exemplo, caso o robô vá para uma zona de obstáculos receberá uma recompensa negativa, caso contrário, o robô receberá uma recompensa positiva (ZHAO et al., 2020).

A tabela Q é uma tabela de m linhas representando todos os estados possíveis do conjunto \mathcal{S} por n colunas representando as ações disponíveis pelo conjunto \mathcal{A} , como pode ser visto na Tabela 1.

Tabela 1 – Representação da tabela Q

estados	ações			
	cima	esquerda	baixo	direita
(0,1)	0	0	0	0
(0,2)	0	0	0	0
...	0	0	0	0
(m,n)	0	0	0	0

Fonte: O Autor

Esta por sua vez é utilizada para armazenar os pares $Q^*(s,a)$ e guiar as tomadas de decisão, onde em cada célula da tabela é armazenado um valor chamado de Q, esse valor é a representação da recompensa a longo prazo que o agente irá receber ao tomar aquela determinada ação naquele estado em particular. O agente aprende sobre o valor $Q^*(s,a)$ de melhor recompensa a longo prazo utilizando um procedimento simples que itera sobre todos os valores $Q^*(s,a)$, ou seja, o algoritmo precisa aprender tudo sobre os valores de Q através da observação (HU, 2017).

O procedimento para a realização de uma iteração *Q-learning* é composta de dois passos:

1. O agente inicializa todos os $Q^*(s,a)$ com um valor 0 para cada par estado seguido da ação correspondente. Ou seja, $Q(s,a)$, significando que não há nenhuma informação sobre a recompensa a longo prazo para cada um dos pares estado-ação.
2. O agente passa por uma mudança de estado saindo de s para um estado s' através da decisão de tomar uma ação a . O agente então atualiza a tabela Q no par (s,a) com o valor de $Q(s,a)$, tal valor é calculado através da equação de *Bellman* como mostra abaixo.

$$Q(s, a) = (1 - \alpha) Q(s, a) + \alpha \left(R(s, a) + \gamma \max_a Q(s', a) \right) \quad (2.5)$$

Da [Equação 2.5](#) acima, temos que α é conhecido como taxa de aprendizado, do inglês *learning rate*, esse parâmetro é definido por um número entre 0 e 1, indicando a relevância dada para a nova informação em comparação com a velha informação.

Se o agente realiza uma ação a em um estado s este recebe uma recompensa imediata $R(s,a)$ somada a uma recompensa adiantada representado por $\max_a Q(s', a)$ que é o valor máximo da soma de todas as recompensas futuras do próximo estado, s' , isto é, assumindo que esse agente sempre toma as melhores ações no futuro ([ZHAO et al., 2020](#)).

As recompensas futuras multiplicadas por uma taxa de desconto γ que também é um valor entre 0 e 1 que determina o grau de importância dado para as recompensas futuras. Caso a taxa de desconto for próxima de 1, o agente dará mais importância em uma eficiência para recompensas a longo prazo, um valor próximo de 0 indica que o agente considera apenas as recompensas imediatas ([KANSAL; MARTIN, 2021](#)).

Ainda de acordo com a [Equação 2.5](#) a cada transição é calculado um erro $(1 - \alpha) Q(s, a)$ que relaciona a ação tomada com a melhor ação que o agente pode realizar para poder transitar para um novo estado s' . Tal erro então é reduzido ou mantido de acordo ao peso dado por α , esse erro indica a relevância que é dada para o antigo valor de Q , assim se α se aproxima de 1 a velha informação não tem relevância para o novo valor de Q . Por fim o erro é somado com a estimativa atual de Q definindo $Q(s, a)$ para ação tomada no estado atual ([DATAHUBBS, 2021](#)).

Nesse método de atualização, Q leva consigo a memória do passado em relação aos eventos ocorridos para próximos passos no futuro. A medida que o agente visita todos os estados e tenta todas as diferentes possibilidades de ações, a equação de *Bellman* auxilia na convergência da tabela Q ajudando o agente na escolha do máximo valor para cada estado da tabela, e fazendo com que este por sua vez percorra o melhor caminho no ambiente ([ZHAO et al., 2020](#)).

2.6.1.2 Estratégias de Exploração e Exploração

Um grande dilema nos algoritmos por reforço é a relação entre exploração e exploração. A exploração toma a melhor decisão devido a situação atual, ou seja, analisa cada uma das duplas estado-ação baseadas em eventos anteriores e prossegue com a ação que permite receber a melhor recompensa. Muitas vezes, quando o ambiente muda com o tempo, o agente pode não conseguir tomar a melhor decisão analisando os eventos anteriores, então o mesmo opta pela estratégia de exploração onde o agente opta por executar diversas ações que não são ótimas para elencar dentre elas a melhor, ou seja, busca por mais informações para tomar a melhor decisão ([ISHII; YOSHIDA; YOSHIMOTO, 2002](#))[\(LAZARIC; RESTELLI; BONARINI, 2008\)](#).

Após diversas tentativas de exploração dos ambientes, os valores de Q convergem fornecendo ao agente uma função de $\mathcal{S} \times \mathcal{A}$, onde o mesmo pode então explotar e consequentemente selecionar qual é a ação ótima a ser tomada para acessar um novo estado. Mas, segundo ([RI-BEIRO, 1999](#)) para que o algoritmo consiga chegar a um estado de convergência o agente se

encontra em um dilema onde é preciso encontrar uma solução ótima que balanceie as alternativas de exploração e os mecanismos de escolhas da melhor solução com exploração. Isso se faz necessário pois, ao tomar ações aleatórias até o fim do treinamento se torna uma estratégia arriscada não levando a uma solução ótima e demandando muito tempo, por um outro lado escolher sempre a política de melhor ação nunca fará com que o agente aprenda a solução que trará a melhor recompensa a longo prazo.

Para que o agente venha realizar a convergência dos valores de Q , em *Q-learning* é utilizado um parâmetro ϵ que balanceia as escolhas entre exploração e exploração. Ao invés de sempre selecionar a melhor ação de acordo com os valores de Q para aquele estado, às vezes o algoritmo opta por realizar o processo de exploração selecionando uma ação aleatória. O parâmetro ϵ é representado por um valor de 0 a 1 sendo que quanto mais próximo de 0 maior é o peso dado para exploração e quanto mais próximo de 1 o agente opta por priorizar a exploração (KANSAL; MARTIN, 2021).

2.6.2 Pseudocódigo do treinamento utilizando *Q-learning*

Dados todos os conceitos que envolvem o treinamento do agente em um ambiente utilizando o método *Q-learning*, o algoritmo 1 mostra de maneira sucinta o passo a passo para montar a tabela Q que definirá o melhor caminho para seguir dentro do ambiente.

Algorithm 1 Aprendizado por Reforço utilizando *Q-Learning*

```

1: Inicialização de  $Q(s,a)$ 
2: for  $Episodio = 1, 2, \dots, N$  do
3:   Inicializa ambiente  $S$ 
4:   for Passo no episódio do
5:     Escolhe ação  $a$  a partir do estado  $s$  utilizando política derivada de  $Q$ 
6:     Toma ação  $a$ , observa a recompensa e avança para estado  $s+1$  caso  $s+1$  não seja
      terminal
7:     Atualiza  $Q(s,a)$  utilizando a equação de Bellmont
8:     
$$Q(s,a) \leftarrow (1 - \alpha) Q(s, a) + \alpha \left( (R(s, a) + \gamma \max_a Q(s', a)) \right)$$

9:      $s \leftarrow s+1$ 
10:    Até  $s$  terminal
11:  end for
12: end for

```

2.7 Trabalhos Relacionados

Com o intuito de contextualizar o projeto e agregar informações relevantes que pudessem servir de base para a construção do presente trabalho, foi realizado um estudo da literatura científica, acerca da resolução de problemas envolvendo a locomoção de robôs e planejamento de caminho em um ambiente.

O trabalho de conclusão de curso de Engenharia de Computação da UNIFESP desenvolvido por ([RUZZON, 2019](#)) é uma das principais referências para este projeto, seu trabalho serve não só como uma base, mas também é o precursor do projeto atual. O trabalho desenvolvido teve como objetivo a construção de um sistema de navegação de um robô hexápode através da utilização um sistema de visão externa. O destaque do projeto é a utilização de um sistema de processamento de imagem utilizando *Python* e *OpenCV*, que consegue estimar a localização atual do robô para controlar a direção que o mesmo deve seguir. Além disso, o autor utiliza da comunicação sem fio por meio do módulo *XBee* para a comunicação do sistema embarcado no robô (que controla seus 18 servomotores) com a estação remota que gerencia todo o processo de locomoção. O robô desenvolvido não obstante é o mesmo utilizado no trabalho em questão e pode ser visto na [Figura 22](#).

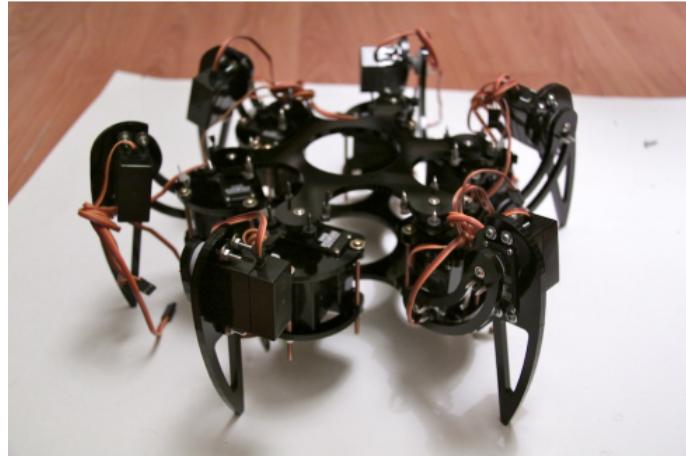
Figura 22 – Hexápode concluído



Fonte: ([RUZZON, 2019](#))

O autor em ([SHAHRIARI, 2013](#)) realizou uma análise em sua tese de doutorado acerca dos algoritmos de Aprendizado por Reforço aplicado em um robô hexápode. O trabalho foi feito através da criação de um modelo virtual de um robô com 18 servomotores simulando todas suas características de *design* e cinemática. Para isto foi utilizado o *software* MATLAB SimMechanics e implementado um algoritmo de *Q-learning* no qual foi utilizado para controlar os atuadores das pernas e identificar quais eram os melhores parâmetros para fazer o corpo se mover para frente. No final do processo de aprendizagem, foi demonstrado que o método de *Q-learning* encontrou a melhor solução para se atingir o objetivo final ao aplicar os resultados simulados com um robô hexápode que pode ser visto na [Figura 23](#).

Figura 23 – Protótipo SiWaRel



Fonte: ([SHAHRIARI, 2013](#))

Quando se fala no planejamento do caminho a ser percorrido por um robô existem inúmeros trabalhos na literatura com as mais diversas abordagens.

Os autores em ([KAVRAKI et al., 1996](#)) utilizaram em seu estudo o método de *Probabilistic Roadmap* para encontrar o melhor caminho que um carro deve percorrer em uma configuração de grande dimensão. No estudo é proposta a utilização de um robô planar a fim de se obter um aumento na eficiência do aprendizado do melhor caminho.

Outro estudo que aborda o planejamento de caminho para ambientes em grandes dimensões é o proposto por ([AMMAR et al., 2016](#)) onde aborda novas estratégias para os algoritmos Dijkstra e A* em ambientes de larga escala. A ideia principal consiste em explotar a estrutura do mapa para estabelecer uma aproximação acurada do melhor caminho sem visitar cada estado mais de uma vez, isso foi feito através do estabelecimento prévio de um custo para o melhor caminho, onde no processo de treinamento os caminhos de maior tamanho que o ideal eram descartados como solução. O estudo se baseou no fato de que ambas versões clássicas são ineficientes em termos computacionais ao serem aplicados em grandes ambientes visto que a abordagem por Dijkstra apresenta uma complexidade quadrática e o algoritmo A* gasta muito tempo processando ambiente com uma grande densidade de obstáculos.

O trabalho realizado por ([PRATIHAR; DEB; GHOSH, 2002](#)) apresenta uma solução para o planejamento do caminho utilizando uma mistura de um algoritmo *fuzzy* com um algoritmo genético, a fim de não apenas encontrar o melhor caminho em um percurso mas também determinar uma sequência de movimentos a serem realizados por um robô hexápode. Outro estudo que mescla algoritmo *fuzzy* com otimização foi proposto por ([GARCIA et al., 2009](#)) onde é utilizado um algoritmo de colônia de formigas que toma decisões a partir da distância do estado atual até o estado desejado, além disso armazena os estados já visitados anteriormente que trazem a solução ótima.

(SONG et al., 2012) apresentaram em seu estudo de planejamento de caminho uma comparação entre a versão clássica do algoritmo de *Q-learning* e uma versão otimizada aplicando uma rede neural sobre a tabela Q. O ponto principal desse estudo é comparar a versão clássica e a versão otimizada, analisando as recompensas recebidas pelo agente e encontro para a melhor solução. No trabalho foi provado que a utilização da rede neural traz os melhores resultados, isso se deve ao fato de que a tabela Q é inicializada com valores heurísticos e não apenas com valores em 0, fazendo com que o algoritmo de aprendizagem converja utilizando menos episódios e também com menos passos.

O grupo (ZHANG et al., 2012) traz em seu estudo uma análise de duas abordagens de *Q-learning* para o planejamento de caminho. A primeira utiliza a versão clássica do algoritmo, onde é considerado um ambiente determinístico, ou seja, são analisadas as diversas possibilidades de ação para um determinado estado até encontrar a solução ideal. A segunda versão que foi implementada é não determinística fazendo com que as ações não sejam apenas atualizadas em uma tabela Q, mas também tragam probabilidades de saída. O modelo foi implementado utilizando o *software* MATLAB e cita a importância de se planejar o caminho que o robô deve seguir desde o ponto de partida até o ponto de chegada fazendo com que o trajeto seja à prova de colisões.

Os estudos propostos por (LIM et al., 2003) e (JEON; KIM; KOPFER, 2011) trazem a abordagem da utilização de *Q-learning* no âmbito de entregas automatizadas a fim de se encontrar o melhor e o mais rápido caminho para entregar um objeto a um destinatário. O estudo de (LIM et al., 2003) aplica AR para estimar o tempo de viagem de veículos automatizados dentro de um segmento fazendo simulações virtuais e comparando os resultados com o estudo proposto por (KIM; TANCHOCO, 1993).

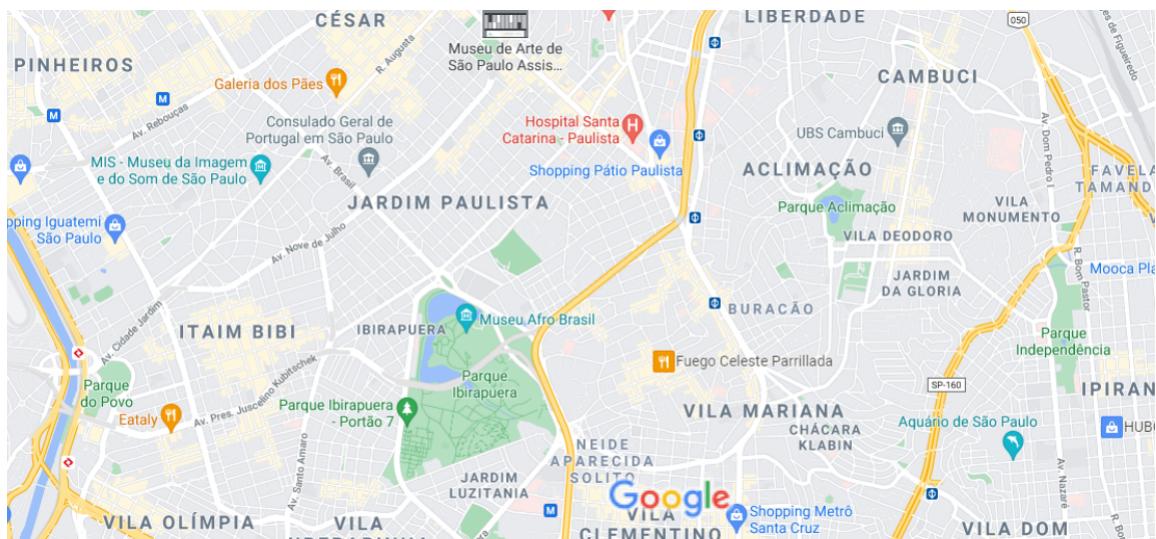
Já o estudo realizado por (JEON; KIM; KOPFER, 2011) utiliza *Q-learning* não apenas para encontrar o menor caminho mas também para calcular o menor tempo de espera dentro de um percurso; o estudo foi feito simulando carros dentro de um porto que tinham por objetivo carregar e descarregar contêineres de navios. Em ambas soluções com a aplicação de *Q-learning* os resultados simulados demonstraram sua eficiência ao encontrar um trajeto ideal.

Um estudo que inspirou e deu base ao algoritmo de *Q-learning* desenvolvido nesse trabalho e que será apresentado no próximo capítulo foi desenvolvido pelo grupo (ZHAO et al., 2020) que propõe um algoritmo chamado de *Experience-Memory Q-Learning*(EMQL), onde o algoritmo se baseia em uma contínua atualização da mínima distância entre o estado inicial e o estado atual. O algoritmo utiliza duas tabelas para o treinamento do agente ao planejar o caminho, a primeira é a tabela EM que armazena as informações de distância e a segunda é a tabela Q que realiza o papel de tabela auxiliar guiando a estratégia de transferência de experiência e o reuso das estratégias de desvio de obstáculos. A eficiência dessa abordagem é alcançada através da utilização de duas recompensas, uma estática e a outra dinâmica, fazendo com que o treinamento converja rapidamente mesmo em ambientes de larga escala.

3 Declaração do Problema

A entrega autônoma de objetos vem sendo explorada nos últimos anos como uma alternativa de baixa emissão de gás carbônico no ambiente em relação a utilização de veículos movidos a combustível fóssil. Contudo, essa abordagem trouxe uma maior visibilidade com o avanço da pandemia de COVID-19, onde o distanciamento social que intensificou a pesquisa e o desenvolvimento de sistemas autônomos de entrega de produtos principalmente nos Estados Unidos (PANI et al., 2020). Porém, além do desenvolvimento de modelos robóticos que sejam capazes de atender a uma demanda da população, em capítulos anteriores foi introduzida a importância de solucionar o problema de planejamento de caminho, ou seja, dado um determinado ambiente, como por exemplo o da [Figura 24](#), como fazer com que o robô possa percorrer o melhor caminho de um determinado ponto arbitrário A(x,y) até um destino em B(w,z).

Figura 24 – Mapa da Cidade de São Paulo



Fonte: Google Maps

Um exemplo clássico de análise de trajetória se faz pela utilização do *Google Maps*, uma grande ferramenta que auxilia na indicação do melhor caminho para se locomover, seja por intermédio de algum meio de transporte ou a pé através da navegação por GPS.

Entretanto, quando o assunto é trazido para o mundo da robótica existem diversas incógnitas que fazem com que uma análise mais apurada seja realizada para a locomoção do robô e a sua capacidade de realizar entregas. Os fatores que influenciam a análise leve em conta questões de desempenho e a dificuldade de localização e mapeamento de uma trajetória em tempo real.

Dentre todos os desafios existentes o mais importante é o planejamento da trajetória. Para isso, o robô precisa fazer as entregas o mais rápido possível, escolhendo o menor caminho

e evitando obstáculos. Com isso, pode-se então formular o problema utilizando o Processo de Decisão de *Markov* que foi introduzido na seção 2.6.1 semelhante ao proposto por (HUNG; GIVIGI, 2016).

Assim da notação de MDP se tem:

1. Um conjunto finito de estados \mathcal{S} .
2. Um conjunto finito de ações \mathcal{A} ;
3. Uma função de recompensa R através da execução de uma ação $a \in \mathcal{A}$ que resulta em um estado $s' \in \mathcal{S}$;
4. Uma função de estado de transição que define o próximo estado $s' \in \mathcal{S}$ após a execução da ação $a \in \mathcal{A}$.

O agente então tem por objetivo aprender a política do ambiente mapeando os estados e ações que maximizam a recompensa. Assim, no caso do projeto em questão o agente que realizará as entregas é representado por um robô hexápode e o ambiente pode ser representado por uma planta baixa de um mapa real como mostra a [Figura 25](#).

Figura 25 – Planta Baixa da Cidade de São Paulo



Fonte: [SCHWARZPLAN](#)

A imagem com a planta baixa do ambiente pode ser representada por uma configuração matricial, onde cada célula da matriz, ou seja, cada *pixel* da figura faz referência a uma coordenada $(x,y) \in \mathbb{R}^2$. Além disso, o conjunto de todas as coordenadas da matriz representa o conjunto \mathcal{S} e cada célula deve indicar um estado s para o agente. A transição entre os estados

são realizadas através da escolha de ações de movimentação para direita, esquerda, para cima e para baixo.

Em resumo, o problema principal que o trabalho busca solucionar é encontrar um algoritmo de rápida convergência que planeje o melhor caminho para um robô percorrer e que segundo ([NIEDERBERGER; RADOVIC; GROSS, 2004](#)) tenha as seguintes especificações:

1. O caminho resultante deve apresentar o menor caminho evitando qualquer desvio de direção;
2. O algoritmo deve ser rápido e eficaz ao simular o processo, por exemplo, deve ser robusto o suficiente para prevenir qualquer obstáculo durante a locomoção e evitar condições de *deadlock*;
3. O algoritmo deve respeitar os diferentes tipos de mapas, isto é, não deve ser construído sendo orientado apenas para um tipo de configuração específica do ambiente.

4 Desenvolvimento

No presente capítulo serão abordados os passos necessários para a realização do trabalho. Os processos que envolvem o desenvolvimento do projeto que busca encontrar o melhor caminho para um robô se locomover em um percurso incluem a construção do algoritmo QLDR que se baseia na abordagem apresentada por (ZHAO et al., 2020) como também a adaptação do trabalho desenvolvido por (RUZZON, 2019) utilizando o robô hexápode e o algoritmo de Processamento de Imagem para mapear e localizar o robô no ambiente.

4.1 Metodologia

Para que o objetivo final do trabalho pudesse ser atingido com sucesso duas etapas foram realizadas. A primeira etapa se refere ao processo de construção do algoritmo implementado e o processo de simulação em *software*. A segunda etapa corresponde à prova de conceito onde foi feito o planejamento do caminho para um robô hexápode percorrer em um ambiente real.

4.1.1 Simulação

A presente seção apresenta os passos realizados na etapa de simulação, contando com definições iniciais de projeto, construção de algoritmos e análises feitas em simulação. Tais passos são:

- Verificação de problemas relacionados ao campo da robótica que utilizem robôs autônomos e definição do foco do trabalho;
- Escolha pelo problema de planejamento de caminho devido ao fato de que muitos algoritmos clássicos que visam solucionar esse problema não serem eficientes para aplicação em grandes centros urbanos onde o ambiente apresenta uma grande densidade de obstáculos;
- Estudos sobre o funcionamento e aplicação dos algoritmos de AR por serem algoritmos de fácil implementação e vem ganhando notoriedade devido a sua aplicação em qualquer tipo de problema que envolva um processo de decisão;
- Levantamento de estudos utilizando *Q-learning* aplicada à resolução do problema de planejamento de caminho. Tal algoritmo foi escolhido pelo fato de apresentar uma melhor representação do conjunto de estados e ações. Entretanto, mesmo se encaixando na configuração do problema a ser solucionado, a abordagem de *Q-learning* leva muito tempo para encontrar a melhor caminho em grandes ambientes. Assim, a motivação para o trabalho se dá pela necessidade de modificar o algoritmo para acelerar o processo de aprendizado;

- Implementação do algoritmo de *Q-learning* clássico na linguagem de programação *Python* pela sua facilidade no desenvolvimento de códigos e acesso à uma gama vasta de bibliotecas que auxiliam na representação visual de resultados.
- Implementação do algoritmo *Q-learning* de dupla recompensa e ajuste dos parâmetros ideais de aprendizado;

4.1.2 Prova de Conceito

Nessa seção serão abordados os passos de desenvolvimento que auxiliaram a validar o conceito do algoritmo QLDR em um ambiente real no processo de locomoção de um robô. Tais passos são representados por:

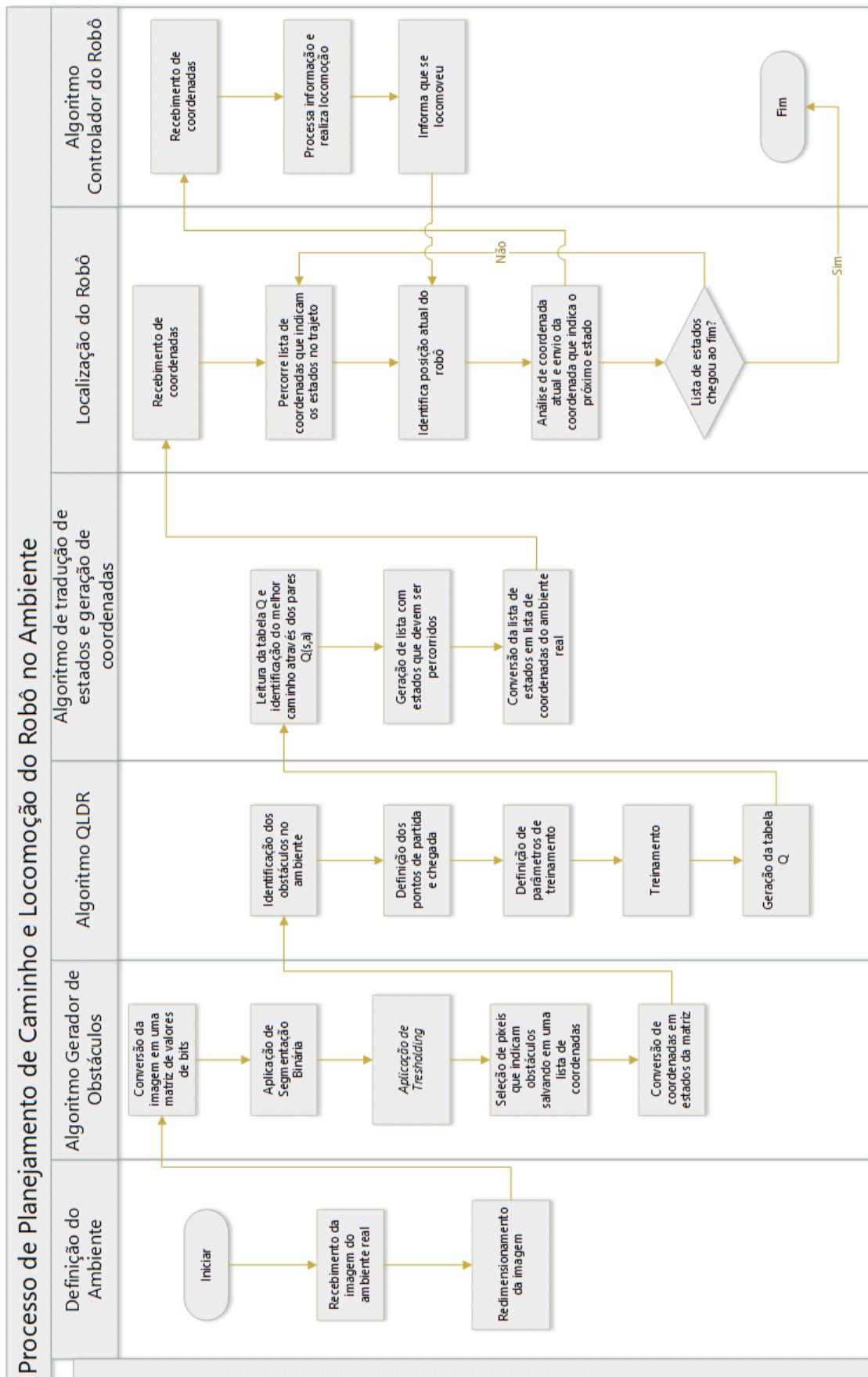
- Escolha da utilização de robôs com pernas para o desenvolvimento do trabalho, devido ao fato de sua morfologia ser mais eficiente no processo de locomoção em ambientes desnivelados, por exemplo, centros urbanos onde se tem calçadas, degraus etc. Fazendo com que as configurações por rodas e trilhos não sejam viáveis no processo de entrega de cargas;
- Estudo sobre análise e processamento de imagens para identificação de obstáculos e determinar a localização do robô. O processamento de imagens foi selecionado pensando em grandes centros urbanos, como o ambiente não é controlado, não há como determinar a melhor rota sem analisar o caminho que deve ser percorrido. Além disso, no caso de um ambiente controlado, como o selecionado para validação de conceito, a utilização de uma câmera de visão externa se torna uma melhor alternativa para além de determinar a localização do robô do que a utilização de sensores que estão mais suscetíveis a falhas;
- Análise da biblioteca *OpenCV* e implementação do algoritmo de Processamento de Imagem em *Python* para conversão de imagem em coordenadas cartesianas $\in \mathbb{R}^2$ e identificação de pixels que indicam obstáculos;
- Levantamento de uma base de imagens que contenham plantas baixas de centros urbanos que representem mapas reais. O intuito da utilização das imagens é converter as plantas através do algoritmo de PI em uma matriz que seja entendível para o algoritmo de *Q-learning* realizar o planejamento do caminho.
- Comparação dos resultados obtidos através da abordagem clássica e a QLDR em termos de tempo de aprendizado para validar a melhoria no processo de planejamento do caminho;
- Desenvolvimento do algoritmo responsável pela locomoção do robô. Tal etapa se fez necessária para que o robô possa performar ações de movimentação no ambiente através da manipulação de seus servomotores;

- Desenvolvimento do algoritmo de PI responsável por localizar o robô no ambiente para orientar o processo de locomoção no ambiente;
- Integração do algoritmo de localização e algoritmo de locomoção através da troca de mensagens utilizando comunicação sem fio entre o robô e o computador que executa o algoritmo de localização;
- Construção de cenário real em baixa escala simulando ambiente com obstáculos para prova de conceito. O objetivo dessa etapa é demonstrar que o caminho planejado em simulação pode ser transferido para um robô real que irá executar a trajetória planejada;
- Para a definição do conjunto de estados, foi utilizada uma câmera externa posicionada no teto do cenário. A câmera fornece por sua vez uma imagem do ambiente que é convertida em uma configuração matricial para ser utilizada pelo algoritmo QLDR;
- Através da obtenção do conjunto de estados, o algoritmo QLDR é utilizado para definir a melhor trajetória para a locomoção do robô;
- Os estados de melhor caminho gerados são convertidos em coordenadas reais do ambiente, esse processo é necessário para a interpretação do algoritmo de locomoção do robô que indicará os movimentos que devem ser realizados ao transitar pelos estados até o objetivo final.

4.2 Integração Hardware e Software

Com o objetivo de elucidar uma visão geral do trabalho, a [Figura 26](#) introduz um diagrama de raia, indicando a sequência de ações que os módulos deverão executar no ambiente mostrando uma visão geral de todo o processo de planejamento de caminho e locomoção do robô real no ambiente. Além disso, o desenvolvimento individual de cada um dos atores do diagrama será explicado nas seções a seguir.

Figura 26 – Fluxograma do Processo de Planejamento de Caminho e Locomoção do Robô no Ambiente



Fonte: O Autor

Do fluxograma se tem que cada raia representa um processo e os blocos determinam as ações que cada módulo deve executar. Os processos podem ser explicados da seguinte forma:

- **Definição do Ambiente:** O ambiente é representado por uma imagem de tamanho $n \times n$ representando a planta baixa de um centro urbano.
- **Algoritmo Gerador de Obstáculos:** O algoritmo de PI recebe a imagem representando o ambiente e a converte em um vetor de duas dimensões. Cada posição poderá ser acessada por uma coordenada $(x,y) \in \mathbb{R}^2$. Após essa etapa, é realizado o processo de segmentação binária onde são atrelados valores entre 0 a 255 aos *pixels* para definir os obstáculos. Ao final da análise, as coordenadas de obstáculos são traduzidas em estados para serem lidas no algoritmo de treinamento;
- **Algoritmo QLDR:** O algoritmo QLDR é o pilar principal do trabalho. Este módulo recebe o tamanho do mapa, que se refere à dimensão da imagem, e em quais coordenadas desse espaço existem obstáculos. O algoritmo por sua vez define os pontos de partida e chegada, que geralmente são fixos, iniciando no primeiro estado do ambiente até o último estado. O QLDR então atualiza a tabela Q com os valores de recompensa ao realizar uma ação em um determinado estado, o valor de recompensa é composto por uma recompensa estática imediata e uma recompensa dinâmica, definida pela taxa de variação da distância euclidiana entre o estado atual e o estado final;
- **Algoritmo de Tradução de Estados e Geração de Coordenadas:** Com a tabela Q finalizada, esse algoritmo faz a análise do melhor caminho através da avaliação dos pares $Q(s,a)$ da tabela, gerando uma lista de sequência de estados, ou seja, uma lista de coordenadas em que o robô deverá seguir. No caso para avaliação em ambiente real, os estados são traduzidos em coordenadas do ambiente real para que o algoritmo que controla a locomoção do robô possa indicar uma sequência correta de passos para o mesmo atingir seu objetivo final;
- **Algoritmo de Localização do Robô:** Tal processo é necessário para identificar qual é a posição atual do robô e para qual estado o mesmo deve se locomover. Para esse processo é necessário que o robô se comunique de alguma forma com um algoritmo de localização para poder informar qual trajeto está seguindo, se chegou ou se não chegou no seu estado objetivo. Para ambientes internos isso pode ser feito através do processamento de imagens utilizando sistemas de visão externa com auxílio de uma câmera. E em ambientes externos se pode utilizar o método de localização por GPS.
- **Algoritmo Controlador do Robô:** O algoritmo deve controlar a movimentação do robô a partir de informações recebidas sobre o trajeto a ser percorrido pelo ambiente além de informar se chegou ou não ao seu objetivo.

4.3 Algoritmo Q-learning de Dupla Recompensa

No algoritmo clássico de Q-learning, o agente, atualiza os pares de $Q^*(s,a)$ da tabela Q através da aplicação da equação de *Bellman*, conforme o valor de recompensa que o mesmo recebe ao realizar a transição de um estado $s \in \mathcal{S}$ para um estado $s' \in \mathcal{S}$. Ao longo do treinamento a tabela Q deverá convergir indicando ao robô o melhor caminho a seguir dentro do ambiente.

Entretanto, a complexidade do algoritmo depende das características do mapa que representa o ambiente e do número de ações que podem ser realizadas em cada estado s visto que o tamanho de uma tabela Q depende da quantidade de estados s e do número de ações. Se um ambiente $\in \mathbb{R}^2$ tem tamanho m em x e n em y e supondo um conjunto de ações $\in \mathbb{R}$ de tamanho a , então o tamanho da tabela Q é dado pela [Equação 4.1](#).

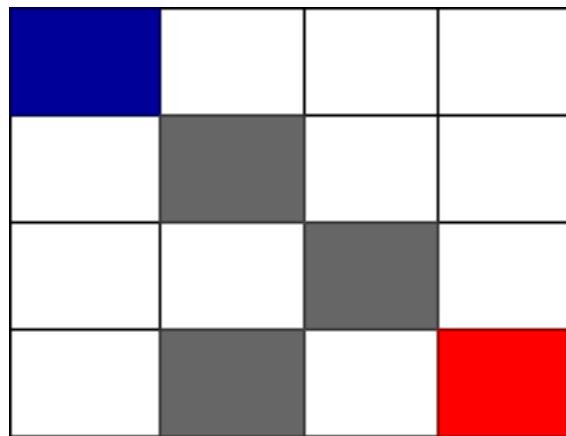
$$Q_{size} : m \times n \times a \rightarrow \mathbb{R} \quad (4.1)$$

Quando um agente interage sobre um ambiente que é complexo, o tempo para que a tabela Q venha convergir se torna muito grande, isso ocorre quando o ambiente apresenta uma grande quantidade de estados e obstáculos para serem analisados. Com o intuito de otimizar o tempo de aprendizado para o melhor caminho é proposto nesse trabalho a implementação do algoritmo QLDR baseado no trabalho proposto por ([ZHAO et al., 2020](#)).

A escolha de um processo de dupla recompensa faz com que o algoritmo aprenda uma política de ações de maneira mais rápida e eficiente quando comparado ao algoritmo clássico, viabilizando a utilização da abordagem de AR para o planejamento de caminho para robôs com pernas. Nas próximas seções será apresentado a proposta do algoritmo no ponto de vista de estados, ações, política do ambiente, recompensa e a convergência do algoritmo.

4.3.1 Representação dos Estados

Para a construção do algoritmo primeiramente foi definida qual a configuração do ambiente, ou seja, com que tipo de estrutura de dados o agente irá iterar para aprender a política do ambiente. Para tal, foi escolhida a notação matricial devido ao fato de que o planejamento do caminho será feito analisando imagens de planta baixa de centros urbanos, onde a imagem pode ser representada como uma matriz de *pixels*. Assim, os estados estão dispostos em uma grade que pode ser representado pela [Figura 27](#).

Figura 27 – Exemplo de configuração de um Ambiente 4×4 

Fonte: O Autor

Da imagem se tem um modelo básico de um mapa representando uma configuração matricial de tamanho 4×4 , onde cada célula é um estado para o agente. As cores da célula indicam a política do ambiente.

- **Azul:** Indica o estado inicial do agente;
- **Cinza:** Indica um estado que seja proibido o acesso, ou seja, um obstáculo;
- **Branca:** Indica um estado livre para movimentação do agente;
- **Vermelha:** Indica estado final, ou seja, o estado alvo que o agente deve alcançar.

Como mencionado anteriormente, cada célula pode ser acessada através da sua coordenada $(x,y) \in \mathbb{R}^2$ e cada célula representa um estado. Com o intuito de facilitar a identificação dos estados, principalmente para definição de zonas livres de movimento e de obstáculos, cada estado pode ser representado por um valor pertencente ao conjunto $\mathcal{S} : \{0,1,2,3, \dots, n*m\}$, sendo n representando o tamanho da matriz no eixo x e m representando o tamanho da matriz no eixo y . A distribuição dos estados de acordo a configuração apresentada anteriormente pode ser vista na [Figura 28](#).

Figura 28 – Distribuição de Estados

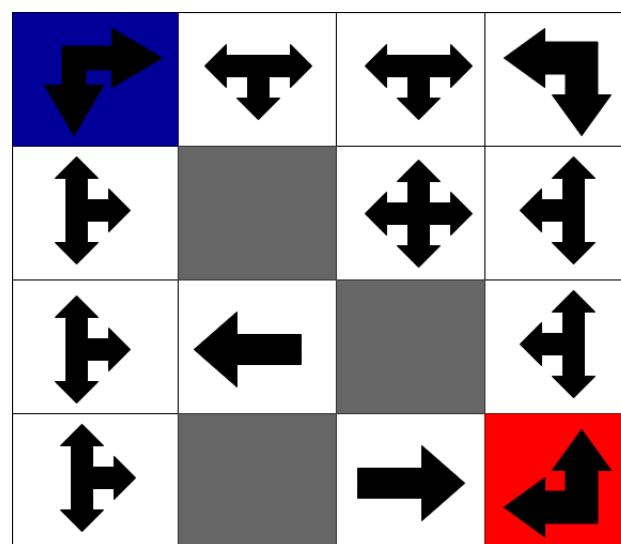
	0	1	2	3
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15

Fonte: O Autor

4.3.2 Espaço de Ações

Em relação as ações que o agente pode escolher no ambiente, o robô é livre para se movimentar em qualquer direção. Ou seja, uma ação é um elemento $a \in \mathcal{A} : \{\text{cima}, \text{direita}, \text{baixo}, \text{esquerda}\}$. Entretanto, alguns estados proíbem determinadas ações de serem realizadas como no caso de extremidades da matriz ou posições que contenham obstáculos. A Figura 29 mostra a distribuição do conjunto de ações na configuração apresentada na seção anterior. Para a facilidade de leitura, os estados foram representados pela tradução do conjunto \mathcal{A} em um conjunto de símbolos $\mathcal{A} : \{\uparrow, \rightarrow, \downarrow, \leftarrow\}$.

Figura 29 – Distribuição das Ações



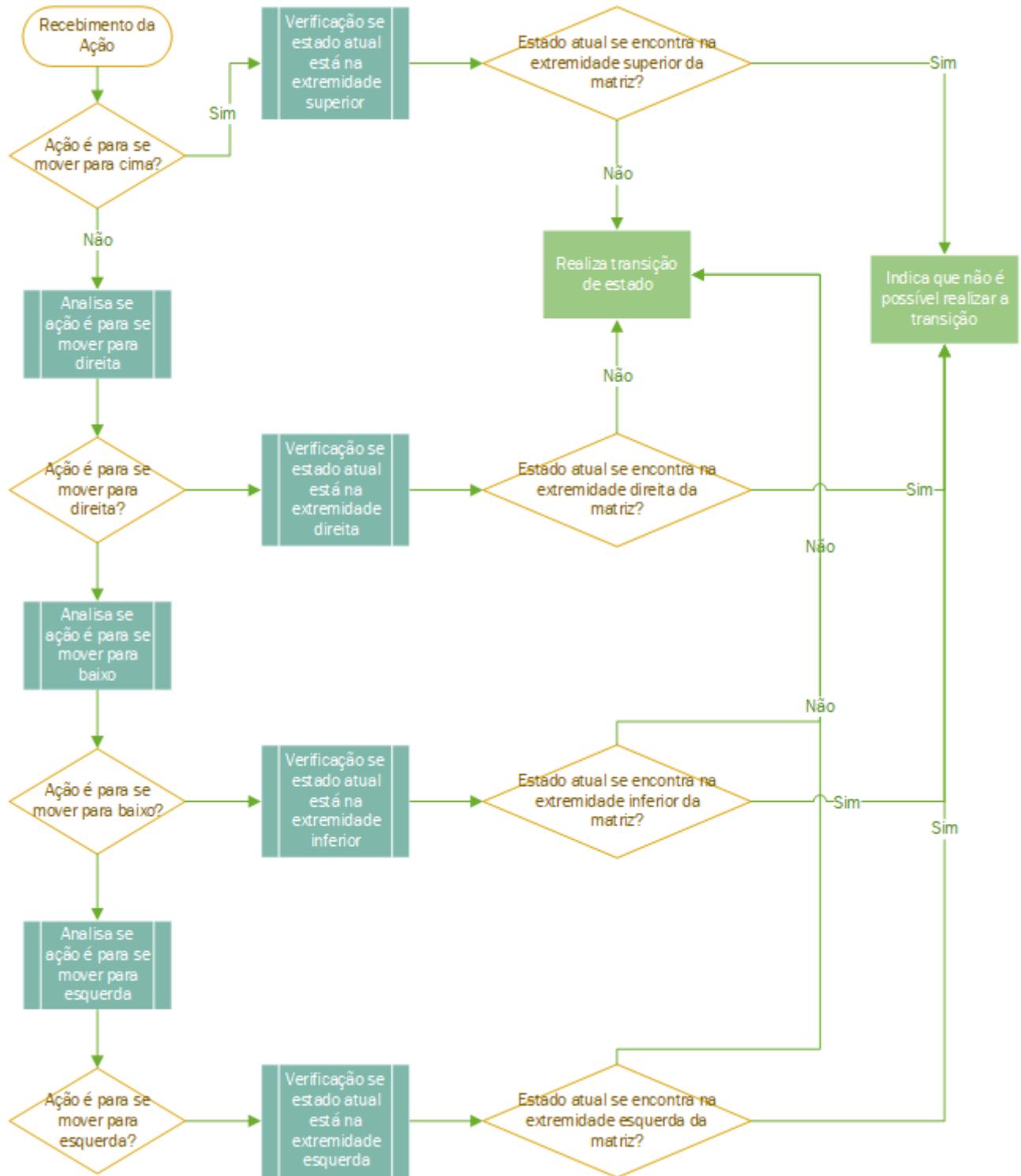
Fonte: O Autor

4.3.3 Modelo de Transição de Estado

A transição por entre os estados é dada através da seleção das ações que buscam mover o robô de um estado $s \in \mathcal{S}$ para um estado $s' \in \mathcal{S}$. Tal seleção irá depender do parâmetro ϵ que definirá se a ação será escolhida através do processo de exploração ou exploração.

Como dito na [subseção 4.3.1](#), cada célula da matriz (mapa do ambiente) pode ser acessada através das suas coordenadas x e y . A partir do momento que o algoritmo escolhe uma ação a ser realizada o mesmo deve analisar qual é o estado s atual do robô para o mesmo se locomover para um estado s' que seja acessível. A análise da transição de estados é definida pelo fluxograma apresentado na [Figura 30](#).

Figura 30 – Fluxograma da Função de Transição de Estados



Fonte: O Autor

A análise de transição de estados representada no fluxograma pode ser transcrita através de análise de coordenadas, conforme pseudocódigo apresentado no Algoritmo 2, onde é feita a análise da transição de estados pelas coordenadas. Caso a ação não se enquadre em ne-

nhum dos casos, o algoritmo deve indicar que o próximo estado não está acessível gerando uma recompensa negativa que será explicada na seção a seguir.

Algorithm 2 Função de Transição de Estados

```

1: Recebimento da ação
2: Identificação de estado s(x,y) atual
3: Identificação de dimensões m e n da matriz
4: if ação igual a a ir para cima e x maior do que 0 then x = x-1
5: else
6:   if ação igual a ir para esquerda e y for maior do que 0 then y = y-1
7:   else
8:     if ação igual a a ir para baixo e x for menor do que m-1 then x=x+1
9:     else
10:       if ação igual a ir para direita e y for menor do que n-1 then y = y+1
11:       end if
12:     end if
13:   end if
14: end if
15: return novo estado s'(x,y)
  
```

4.3.4 Esquema de Recompensa

A recompensa é uma análise temporal da ação desempenhada pelo robô servindo como principal forma de avaliar eficiência do aprendizado pois indica o acerto ou o erro ao performar determinada ação.

Estudos com AR demonstraram que se pode determinar uma pequena recompensa negativa quando o robô se move para um estado livre de obstáculos. Quando o robô selecionar uma ação que leva a uma zona proibida deve receber um alto valor negativo. Por fim, quando o robô selecionar uma ação que leve ao estado objetivo se deve receber um alto valor de recompensa positivo.

Tendo como premissas os valores definidos para as recompensas do agente, se pode então formular a abordagem com dupla recompensa. A proposta consiste em utilizar uma recompensa estática, que está baseada no estado atual de transição, e uma recompensa dinâmica. A recompensa dinâmica é um valor que muda de acordo com a distância Euclidiana entre o estado atual, o próximo estado e o estado final. A utilização de duas recompensas auxilia o agente a escolher ações que mais se aproximam do seu objetivo, evitando que o mesmo fique explorando o ambiente mais do que o necessário. Assim a recompensa final pode ser dada pela [Equação 4.2](#).

$$\text{Recompensa} = \text{Recompensa}_{estatica} + \text{Recompensa}_{dinamica} \quad (4.2)$$

4.3.4.1 Recompensa Estática

Segundo o que foi citado na [subseção 4.3.1](#) há quatro classificações para os estados no ambiente. O primeiro é o estado inicial, o segundo é o estado livre, o terceiro é o estado de obstáculo e por último o estado final. Quando o robô transita para qualquer estado, exceto o estado inicial, o mesmo recebe um valor de recompensa que pode ser categorizado de acordo com a [Equação 4.3](#) onde os valores foram definidos de maneira empírica.

$$Recompensa_{estatica} = \begin{cases} -500, & \text{se } s' \text{ é um obstáculo ou está fora da matriz} \\ -1, & \text{se } s' \text{ é um estado livre para se movimentar} \\ -100, & \text{se } s' \text{ é o estado inicial} \\ 5000, & \text{se } s' \text{ é o estado final} \end{cases} \quad (4.3)$$

O alto valor negativo dado ao robô ao escolher uma ação que o levará para um obstáculo, se dá pelo fato de que o agente fica sem outra direção para seguir, forçando-o a permanecer no mesmo estado s consumindo tempo de aprendizado, pois o agente ficará encurralado até encontrar uma outra ação que o faça se mover para outro estado s' livre.

Além disso, se o agente definir que a próxima ação fará o robô se mover para o estado s inicial, a recompensa terá um valor significativamente baixo pois o retorno ao estado inicial ocasiona num gasto de tempo de retreinamento.

4.3.4.2 Recompensa Dinâmica

A recompensa dinâmica se faz necessária para a redução no tempo de aprendizado para auxiliar o agente a selecionar o melhor caminho. Na etapa em que o robô explora o ambiente para colher informações e preencher a tabela Q, apenas a recompensa estática não se demonstra suficiente para o desvio completo de obstáculos e alcance do objetivo final.

Assim, em busca de uma otimização de *Q-learning* foi utilizado o cálculo da recompensa dinâmica através do cálculo de distância Euclidiana bidimensional. Ao utilizar esse método, o agente é melhor direcionado para atingir o estado objetivo além de se familiarizar com o ambiente com mais rapidez.

A recompensa dinâmica está relacionada com a mudança de distância entre o estado s que o robô se encontra e o estado s final em dois momentos. O primeiro momento, de acordo com [Equação 4.4](#) é o cálculo da distância atual, ou seja, antes do agente escolher uma ação que fará transitar para outro estado. O segundo momento, de acordo com a [Equação 4.5](#) é quando o agente transitou para o estado s' .

$$d_s = \sqrt{(x_{objetivo} - x_{atual})^2 + (y_{objetivo} - y_{atual})^2} \quad (4.4)$$

$$d_{s'} = \sqrt{(x_{objetivo} - x_{prox})^2 + (y_{objetivo} - y_{prox})^2} \quad (4.5)$$

Das duas equações se pode calcular então uma razão entre as distâncias do estado atual e o próximo estado em relação ao estado objetivo como mostra a [Equação 4.6](#).

$$Recompensa_{dinamica} = \lambda \times \frac{d_s - d_{s'}}{|d_s - d_{s'}|} \quad (4.6)$$

Com a definição da recompensa dinâmica, se tem que λ é o parâmetro que está relacionado ao valor dado pela recompensa estática, esse valor é utilizado para poder ajustar qual é o peso dado para a movimentação do robô em direção ao estado final, ou seja, λ define o quanto a recompensa estática influencia no valor final de recompensa.

Mesmo que o parâmetro λ influencie no peso da recompensa estática, de acordo com a [Equação 4.6](#) se o algoritmo escolher uma ação que fará com que o robô se move para mais distante do estado final, a recompensa nesse caso continuará sendo negativa, ou seja, o agente ainda receberá um valor negativo como punição, fazendo com que o valor de λ tenha pouca influência.

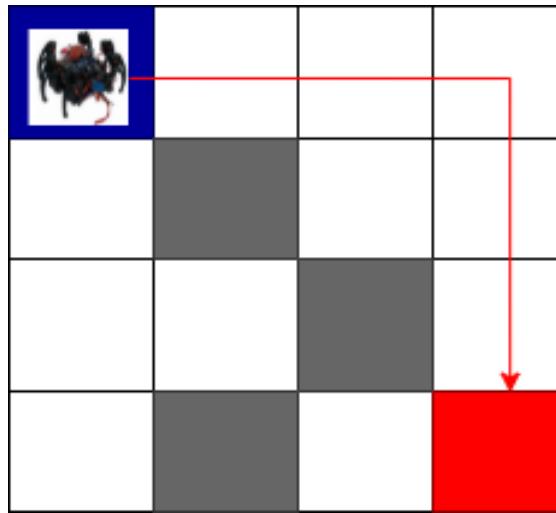
Entretanto, a influência de λ se demonstra de grande importância para o valor da recompensa final a medida que o agente se move para o seu objetivo. Caso o valor de λ seja alto, a recompensa estática trará pouco impacto para o agente. Caso λ tenha um valor baixo fará com que a recompensa dinâmica cause pouca influência na transição do robô para o estado final, nesse caso a recompensa final depende somente da recompensa estática. Tal valor, apresenta maior peso quando o agente opta por selecionar uma ação que o faça se afastar do seu estado final indicando que essa movimentação não foi uma boa opção dando um maior peso para a recompensa estática.

Assim sendo, para a implementação do algoritmo de *Q-learning* proposto foi definido que o valor de λ irá ser sempre o módulo do valor recebido pela recompensa estática.

4.3.5 Implementação do Algoritmo

Com os elementos que compõem o processo de treinamento definidos por um modelo MDP, fica em evidência que o objetivo do robô é percorrer o melhor caminho dentro do ambiente, ou seja, o algoritmo *Q-learning* deverá aprender sobre a política do ambiente e indicar a melhor rota para o robô percorrer, como mostra a [Figura 31](#).

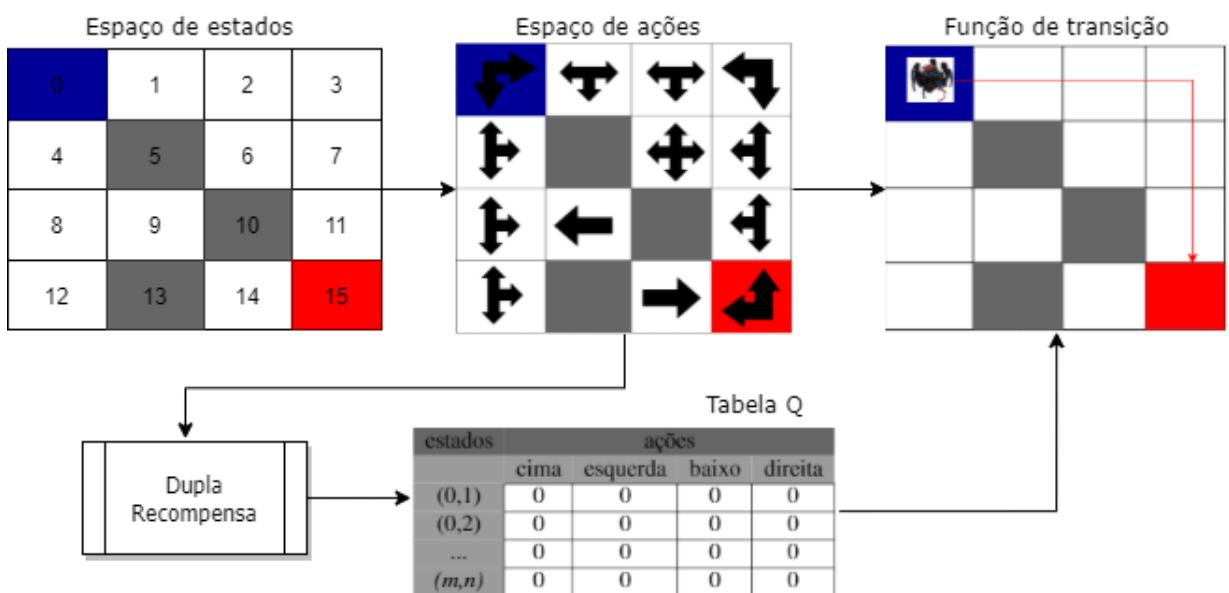
Figura 31 – Melhor Caminho



Fonte: O Autor

Sendo assim, a modelagem do problema faz com que os módulos de conjunto de estados, conjunto de ações e função de transição sejam dependentes um do outro de acordo com o MDP. Os estados definem como serão as ações, as ações definem como a função de transição irá se comportar e a transição define os possíveis valores de recompensa para atualização dos valores $Q(s,a)$ da tabela Q, que por sua vez auxilia na escolha da próxima ação do agente. Os elementos que compõem o processo de aprendizado podem ser interligados e representados conforme mostra a Figura 32.

Figura 32 – Representação de etapas em Q-learning



Fonte: O Autor

4.3.5.1 Política de ϵ

Do processo de aprendizado, o que define como a locomoção do robô será executada no ambiente é a escolha das melhores ações e a atualização da tabela Q. Entretanto para que a tabela Q venha operar de forma correta otimizando os estados a serem seguidos é preciso fazer a escolha da política de ϵ .

Conforme o que foi explicado na [subseção 2.6.1.2](#) a escolha de uma valor de ϵ que balanceie a escolha das ações entre exploração e exploração levará à solução ótima. Tal balançamento é chamado de política de ϵ .

A principal estratégia para ϵ em grande parte dos algoritmos é a estratégia gulosa na qual define um valor fixo para ϵ de forma que o agente selecione uma ação aleatória com probabilidade $1 - \epsilon$ o agente escolhe a melhor ação para aquele estado utilizando a tabela Q.

Entretanto, ao definir um valor fixo para ϵ faz com que o algoritmo não opere de maneira ótima, pois o mesmo não consegue extrair informações suficientes sobre o ambiente que está interagindo. Esse problema ocorre principalmente em grandes ambientes com grande densidade de obstáculos.

Pensando em ambientes complexos é preciso fazer como que o agente obtenha o máximo de informação possível sobre os estados e ações disponíveis. Assim, uma vez que se tenha colhido informações através da exploração nos episódios iniciais, os episódios futuros poderão fazer as melhores escolhas através da exploração. Para encontrar a melhor solução, foi definido que em um primeiro momento ϵ recebe um valor alto, próximo de 1, e ao longo dos episódios de treinamento o valor é diminuído a uma taxa fixa, tal decaimento é definido como decaimento exponencial.

A atualização do valor de ϵ é feita no fim de cada episódio, para que nas próximas iterações o agente tenda a escolher cada vez mais soluções ótimas. Para o presente trabalho foi definido que ϵ inicializa o processo de aprendizado com o valor de 0,9 e irá sofrer um decaimento de 0,05 ao final de cada episódio até atingir um valor mínimo.

A atualização do parâmetro pode ser visto no pseudocódigo mostrado no [Algoritmo 3](#).

Algorithm 3 Atualização de ϵ

```

1: epsilon-decay = 0,05
2: min-epsilon = 0,05
3: epsilon = 0,9
4: Episódios = N
5: ...
6: for Episódio em 1,2, ..., Episódios do
7:   ...
8:     if Episódio chegou ao fim then
9:       if epsilon>min-epsilon then epsilon = epsilon - (epsilon-decay)*epsilon
10:      end if
11:    end if
12: end for

```

4.3.5.2 Parâmetros α e γ

Assim como o parâmetro ϵ tem um papel chave na otimização do aprendizado, os valores α e γ também tem sua contribuição para levar o agente ao estado final de maneira ótima. Para tal, os valores foram definidos como sendo próximos de 1 onde α definido como 0,99 e γ igual a 0,95. A escolha desses valores foi feita baseando-se na premissa que nos primeiros episódios do treinamento o agente irá explorar até conseguir informações suficientes sobre o ambiente, o que acontece é que ao realizar esse processo, as próximas iterações correm o risco de trazerem consigo muita informação que não é relevante, por exemplo, supondo que nos primeiros episódios, em grande parte do tempo de processamento, o agente tenha o azar de escolher muitas ações que levam a obstáculos. Essas escolhas resultarão em uma grande média negativa de recompensas, então para as gerações futuras essa informação não poderá causar impacto, por isso o agente deve atribuir mais valor a recompensas futuras dos que as atuais.

4.3.5.3 Algoritmo

Com a definição do problema, a escolha da representação dos conjunto de estados, conjunto de ações, esquema de recompensa e forma de atualização da tabela Q foi desenvolvido um algoritmo que se assemelha muito ao modelo clássico de *Q-learning*, como pode ser visto na sequência de passos abaixo.

1. O ambiente é inicializado definindo as dimensões do ambiente, a posição inicial do agente e o conjunto de ações e os estados ;
2. Inicialização da tabela Q com todos valores em 0;
3. Definição de quantos episódios de treinamento serão necessários;
4. Itera sobre cada episódio;
5. Seleciona uma ação do conjunto de ações baseado na estratégia de ϵ guloso;

6. Executa a ação e observa a função de transição de estados;
7. Define o novo estado do agente;
8. Recebe a recompensa estática;
9. Calcula a recompensa dinâmica;
10. Atualiza a tabela Q através da equação de *Bellman* e da dupla recompensa;
11. Repete os passos anteriores a partir do item 5 até o final do episódio;
12. Calcula o novo valor de *epsilon* no final de cada episódio.

Da lista acima se pode então representar o processo através do pseudocódigo mostrado no Algoritmo 4.

Algorithm 4 Aprendizado por Reforço utilizando *Q-Learning* de Dupla Recompensa

```

1: Inicialização de  $Q(s,a) \forall s \in \mathcal{S}, a \in \mathcal{A}, \epsilon, \epsilon - decay, \epsilon - min, \alpha, \gamma$ 
2: for episodio = 1, 2, ..., N do
3:   Inicializa flag que indica se o episódio finalizou
4:   Inicializa ambiente S
5:   Inicializa tabela Q,  $\forall Q(s, a) \in Q, Q(s, a) \leftarrow 0$ 
6:   while done!=True do
7:     Escolhe ação a utilizando política de  $\epsilon$  ou  $\max(Q(s))$ 
8:     Executa ação a e avança para estado s' caso s' não seja obstáculo ou esteja fora dos
       limites do ambiente
9:     Recebe recompensa estática
10:     $Recompensa_{estatica} \leftarrow -1, -100$  ou  $-500$ 
11:    Calcula recompensa dinâmica a partir da recompensa estática
12:     $Recompensa_{dinamica} \leftarrow \lambda \times$  cálculo da razão da distância euclidiana de s e s' em
       relação ao estado final
13:    Atualiza  $Q(s,a)$  utilizando a equação de Bellman
14:     $Q(s,a) \leftarrow (1 - \alpha) Q(s, a) + \alpha \left( (R(s, a) + \gamma \max_a Q(s', a)) \right)$ 
15:     $s \leftarrow s'$ 
16:    if s'== estado final then done  $\leftarrow$  True
17:    end if
18:  end while
19:  Atualiza  $\epsilon$ 
20:  if  $\epsilon > \epsilon - min$  then  $\epsilon \leftarrow \epsilon - (\epsilon - decay) * \epsilon$ 
21:  end if
22: end for

```

O algoritmo completo desenvolvido em *Python* se encontra no [Apêndice B](#), onde na parte 1 se encontra a classe abstrata com todos os métodos necessários para o treinamento do ambiente e na parte 2 está o algoritmo de treinamento em si com o cálculo de recompensa, atualização da tabela Q e definição de parâmetros. Por fim, na parte 3 encontra-se o algoritmo responsável pela geração de métricas e resultados no treinamento.

4.4 Algoritmo Gerador de Obstáculos

No sistema cartográfico que representa o mundo real através de mapas, os valores de longitude e latitude possuem o papel de indicar algum lugar específico. Entretanto, ao trazer essa analogia para uma grade que representa o mundo real, os mapas de planta baixa trazem consigo uma grande vantagem para se ter uma referência do ambiente que o robô irá percorrer, consequentemente facilitando o planejamento do melhor caminho.

No projeto em questão, para auxiliar na definição de um ambiente de treinamento e facilitar a identificação de obstáculos, foram utilizados diversos mapas de planta baixa onde o robô é considerado como sendo uma partícula nesses ambientes. Entretanto, os mapas precisam de uma tradução das imagens como uma notação matricial que representa o conjunto de estados para o algoritmo de treinamento.

Para a realização dessa etapa foi utilizado como referência os mapas disponíveis no site [SCHWARZPLAN](#) onde consta inúmeros modelos de mapas de planta baixa de grandes cidades espalhadas pelo mundo inteiro. Cada planta apresenta uma escala de 1:100000 como por exemplo o mapa de Barcelona mostrado na [Figura 33](#). Contudo, mesmo que as plantas representem o mundo real nem sempre uma representação de uma via indica que aquela via de fato existe, além disso alguns mapas não estão atualizados. Assim, o presente trabalho faz a análise dos mapas considerando zonas livres ou zona de obstáculos sem comparações com mapas atualizados.

Figura 33 – Planta Baixa de Barcelona



Fonte: [SCHWARZPLAN](#)

Para que os estados de obstáculos pudessem ser gerados neste trabalho foi utilizado um algoritmo simples implementado em *Python*, utilizando a biblioteca de visão computacional OpenCV. Como dito nas seções que antecederam esta, os estados do ambiente de treinamento é nada mais do que um par de coordenadas (x,y), ou seja, um vetor de duas dimensões.

Sabendo que uma imagem é uma matriz de pixels, onde cada pixel possui a sua intensidade, fica maleável identificar as zonas livres e zonas proibidas de um ambiente. Para poder identificar tais zonas, a OpenCV através de funções nativas, oferece a solução simples para o problema conforme pode ser visto no algoritmo abaixo.

```

1 import cv2
2 import numpy as np
3 from PIL import Image
4 import matplotlib.pyplot as plt
5
6 list = []
7 imgg = []
8 img_path = r'C:\Users\mesqu\Downloads\Figure_11.png'
9 img = cv2.imread(img_path, 0)
10
11 ret, thresh = cv2.threshold(img, 180, 255, cv2.THRESH_BINARY)
12
13 h, w = thresh.shape
14 print('width: ', w)
15 print('height: ', h)
16 state_matrix = np.ascontiguousarray(np.arange(h*w).reshape(w,h), dtype=int)
17 new_img = np.argwhere(thresh==0)
18
19 for i in new_img:
20     list.append(state_matrix[i[0]][i[1]])
21
22 with open('states.txt','w') as output:
23     output.write(str(list))
24
25 plt.imshow(thresh)
26 plt.show()
```

Do algoritmo se tem que na linha 9 a biblioteca faz a leitura da imagem para que na linha 11 o algoritmo possa aplicar o método de segmentação binária (*thresholding*) apresentado na [subseção 2.5.2.1](#) com o intuito de transformar os vários níveis de cinza da representação do ambiente em uma figura binária, isso é feito aplicando o método *cv2.thresholding*.

Ainda sobre a linha 11, o primeiro parâmetro indica que de fato será convertida, o segundo parâmetro indica o valor de *thresholding*, no caso é representado pelo número 180 que foi selecionado empiricamente através de vários testes. O terceiro parâmetro indica o valor máximo, ou seja, caso o *pixel* exceda o valor de *thresholding* receberá o valor 255 que é o máximo na escala RGB. Por último, o parâmetro *TRESH_BINARY* indica ao *OpenCV* o tipo de segmentação binária que será realizada visto que o mesmo opera com diferentes tipos de segmentação.

A ideia da função de *thresholding* é simples, inicialmente ao receber a imagem verifica

o valor de cada *pixel*, caso o valor seja maior que 180, então o *pixel* é sobreescrito com valor 255 indicando a cor branca. Caso contrário, o *pixel* é rotulado com o valor 0 indicando a cor preta, ou seja, um obstáculo.

Após a rotulação dos *pixels* a imagem então passa a ser representada de acordo com a [Figura 34](#)

Figura 34 – Conversão de pixels



Fonte: O Autor

Com a geração da imagem é possível identificar os obstáculos através do valor agregado aos *pixels*, onde a cor amarela identifica os estados com valor 255 e os de cor roxa identificam os obstáculos.

Vale citar que o valor de *thresholding* foi escolhido pensando em algumas imagens que possuem baixa resolução. Uma baixa qualidade da imagem da planta a ser analisada indica uma maior dispersão nos valores dos *pixels* que podem indicar erroneamente obstáculos sem a devida análise.

Por fim, a posição dos pixels selecionados é salva em uma lista representando os estados que contém obstáculos para que então o algoritmo de *Q-learning* possa realizar o devido treinamento ao definir a melhor trajetória para a locomoção do robô.

5 Resultados Obtidos

Tendo executado todo o processo do desenvolvimento do projeto de acordo o que foi apresentado nas seções do capítulo anterior, o presente capítulo discutirá acerca dos resultados obtidos através do treinamento de várias configurações de ambientes onde através do algoritmo QLDR proposto pôde-se planejar o melhor caminho pelo ambiente. Além disso foi proposto um teste em um ambiente de tamanho reduzido para avaliação de conceito utilizando um robô hexápode real.

O presente capítulo então foi dividido também em duas partes, na primeira será descrito o processo de treinamento para diversas configurações de ambientes tanto em questão de obstáculos quanto no tamanho do mapa que representa o ambiente. Além disso, será discutido acerca da comparação entre o algoritmo QLDR proposto e o algoritmo de *Q-learning* clássico em questões de performance de aprendizado.

Na segunda parte será discutido acerca dos resultados experimentais com o robô hexápode real mostrando como o algoritmo de QLDR pode planejar e definir o melhor caminho para o robô se locomover em um ambiente de teste.

5.1 Resultados em Simulação

Para a avaliação do algoritmo QLDR e validação de sua eficácia foram selecionados cinco ambientes de testes diferentes. Os ambientes selecionados representam regiões dos mapas das cidades de Barcelona, Chicago, Los Angeles, São Paulo e Vancouver, as imagens por sua vez variam de tamanho e também de densidade de obstáculos fazendo com que se possa apurar o comportamento do treinamento nas mais variadas configurações.

Em todos os testes, a imagem é convertida para o formato matricial representando uma configuração espacial de grade com coordenadas x e y , onde o agente é visto como sendo uma partícula dentro do ambiente. Além disso, o objetivo do agente em todas as configurações é sair do primeiro estado representado pelas coordenadas $(x,y) = (0,0)$ e aprender uma trajetória para alcançar o último estado representado pelas coordenadas $(x,y) = (m,n)$, onde m e n representam o tamanho do ambiente em parâmetros de comprimento e largura.

Outro ponto relevante a citar é que todos os ambientes foram treinados ao longo de 300 episódios cada, mantendo os mesmos parâmetros de ϵ , α e γ .

5.1.1 Estudo de Caso A - Barcelona

A primeira configuração de ambiente para treinamento foi utilizando uma segmentação do mapa da cidade de Barcelona na Espanha.

5.1.1.1 Definição de Estados

O espaço selecionado é representado por uma figura de tamanho 139x139 que apresenta uma grande densidade de obstáculos porém com zonas livres específicas para que o agente possa se locomover conforme mostra a [Figura 35](#).

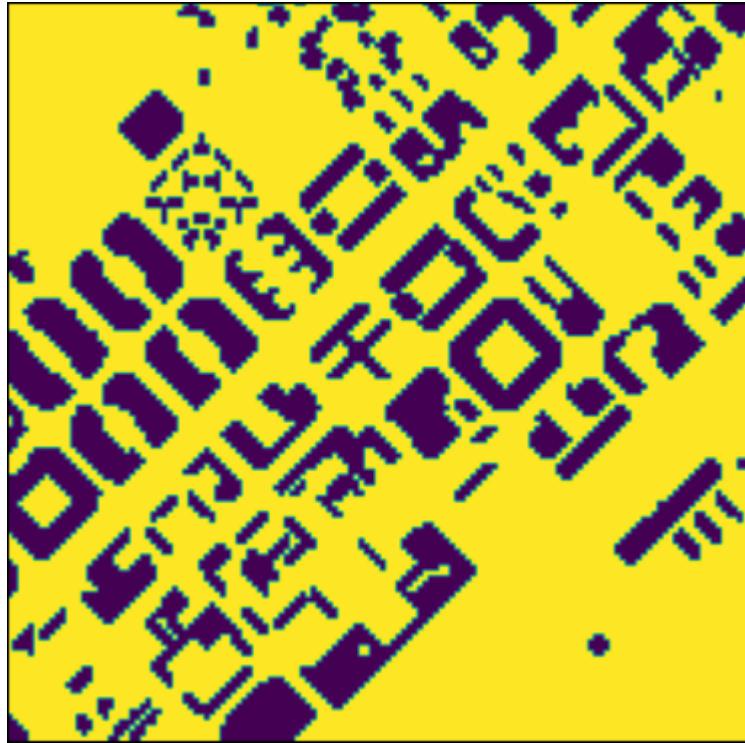
Figura 35 – Mapa de Barcelona



Fonte: [SCHWARZPLAN](#)

Com a definição do ambiente, a imagem então passa pelo algoritmo gerador de obstáculos que analisa quais são os estados que o robô não poderá acessar conforme mostra a [Figura 36](#). Após a aplicação do método de *thresholding* na imagem os obstáculos podem ser observados através da cor roxa.

Figura 36 – Identificação dos Obstáculos no Mapa de Barcelona



Fonte: O Autor

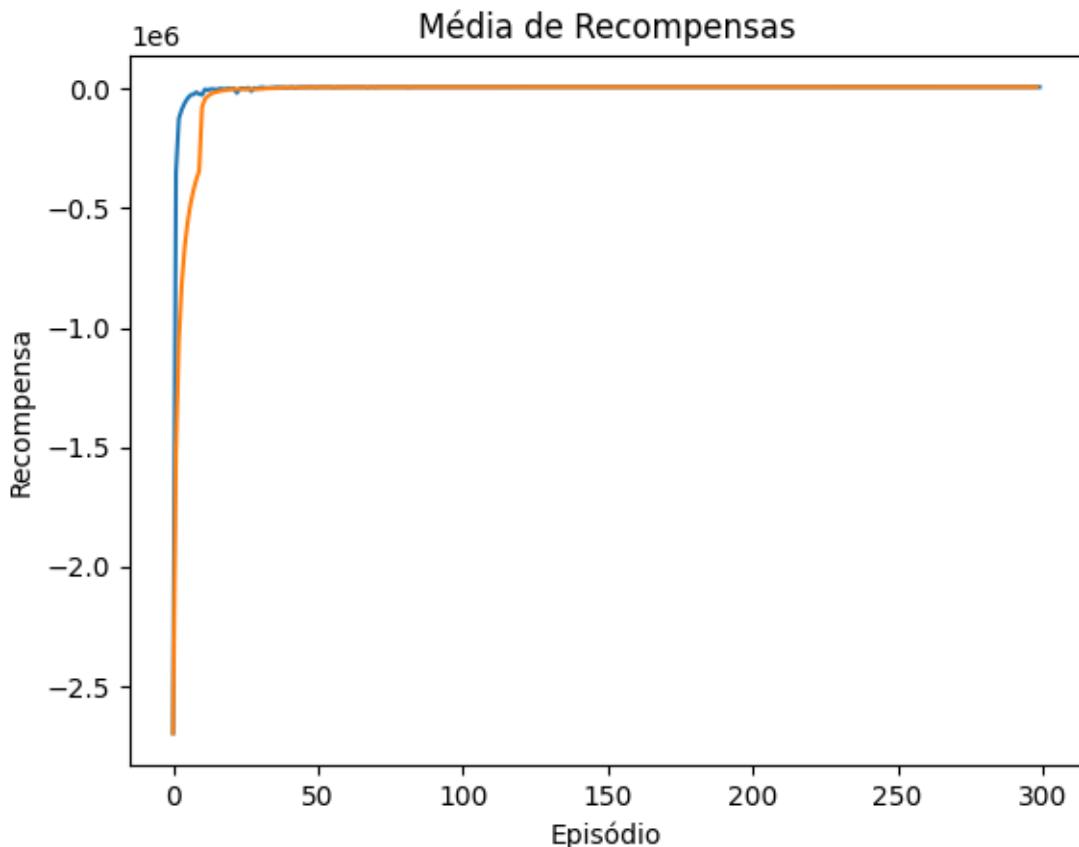
5.1.1.2 Tempo de Treinamento

Ao fim do treinamento para esse mapa se pôde observar que o algoritmo QLDR realizou o processo de treinamento do ambiente em um tempo total de aproximadamente 101,43 segundos com uma média de 627,53 passos em todos os 300 episódios . O tempo levado para o treinamento é dado pelo fato de que nas primeiras gerações de treinamento o algoritmo opta por explorar até colher informações suficientemente relevantes para tomar as melhores decisões no futuro.

5.1.1.3 Convergência do Algoritmo

Ao analisar as recompensas recebidas durante o treinamento ao realizar o planejamento de caminho através do ambiente, percebe-se que o consegue rapidamente atingir um estado de equilíbrio como mostra a Figura 37, onde a linha plotada em azul indica as recompensas e a linha em amarelo indica a média dos valores recebidos em cada episódio.

Figura 37 – Convergência de Recompensas no Treinamento



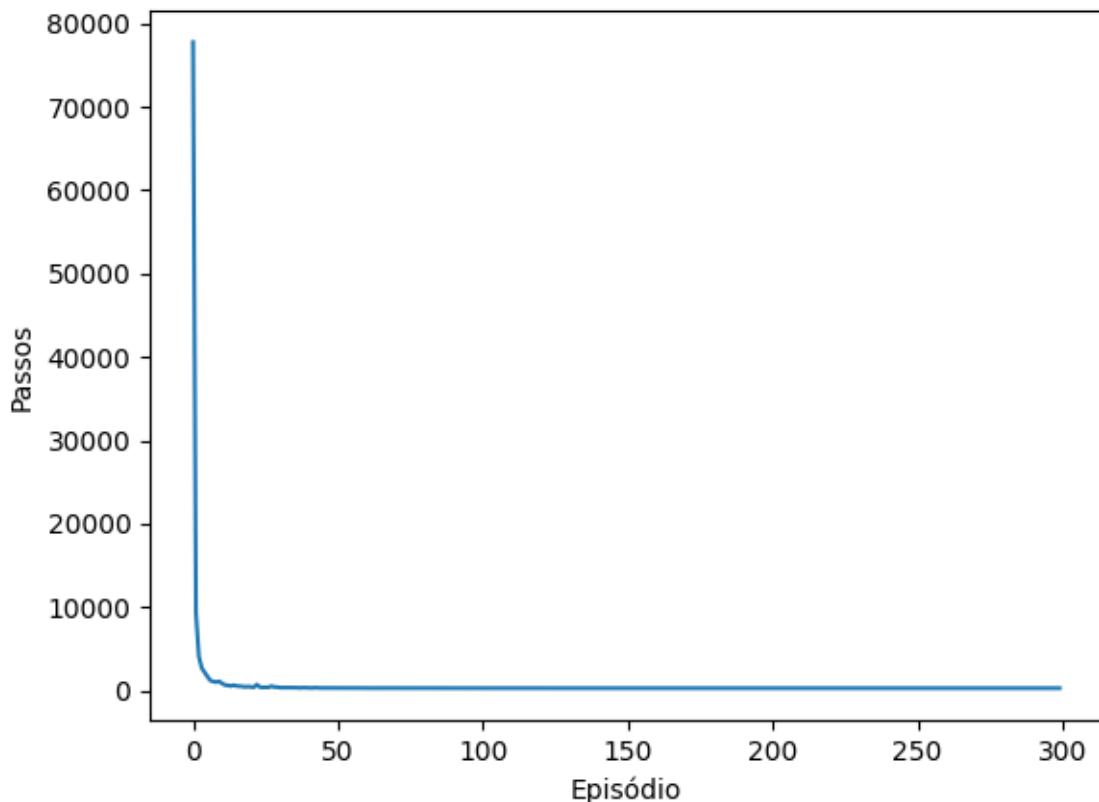
Fonte: O Autor

Ao analisar a imagem, se têm que em torno de 25 episódios o algoritmo QLDR entra em um estado de convergência, isso indica que o agente já colheu as informações necessárias para a locomoção e a partir disso o mesmo acaba tendendo a fazer escolhas ótimas resultando em valores semelhantes de recompensa. Vale citar que a convergência do algoritmo é o que garante o sucesso do planejamento do caminho, entretanto, essa métrica sozinha não indica que de fato o agente está se locomovendo para o melhor caminho, mas sim que suas escolhas levaram ao objetivo final.

5.1.1.4 Número de Passos

Outra métrica de análise que indica o sucesso do algoritmo no planejamento do caminho é a relação da quantidade de passos realizados pelo agente, como mostra a Figura 38. Da imagem se tem que nos primeiros episódios como o agente opta pelo processo de exploração do ambiente, o número de passos dados durante todo o percurso acaba sendo grande devido à escolha por ações aleatórias.

Figura 38 – Passos Dados no Treinamento do Mapa de Barcelona



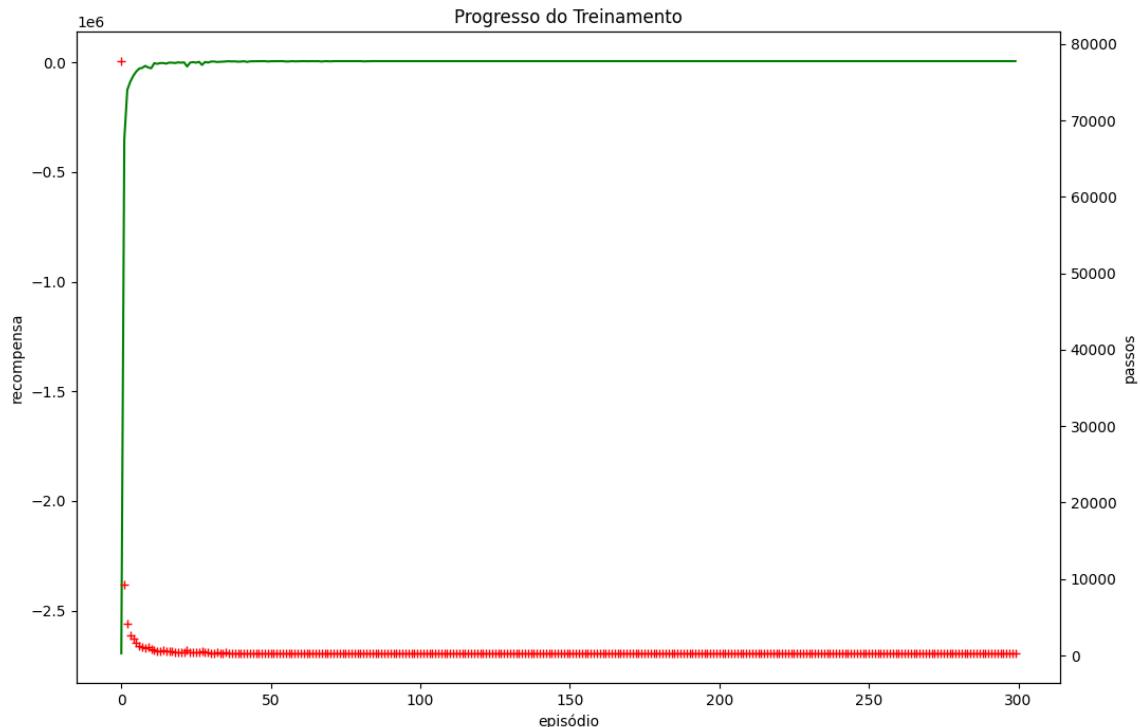
Fonte: O Autor

De acordo com os dados obtidos, nos dez primeiros episódios o agente percorreu uma média de 12166 passos, em contrapartida, ao alcançar o estado de convergência, os últimos dez episódios indicaram uma média de 278 passos dados no percurso.

5.1.1.5 Número de Passos *versus* Recompensa

Ao relacionar a quantidade de passos em relação ao valor de recompensas que o agente recebeu durante o treinamento, a Figura 39 apresenta um comportamento esperado. À medida que o robô inicia a exploração do ambiente, o número de passos dados é bem grande e devido a esse valor o robô fica muito tempo preso em estados de obstáculos trazendo o valor de recompensa para baixo. Entretanto, com a decaída de ϵ e a coleta de informações necessárias para preencher a tabela Q, o agente entra no estado de convergência, onde tanto os passos dados em cada episódio quanto as recompensas se estabilizam, indicando uma solução para o planejamento do caminho.

Figura 39 – Relação entre Passos e Recompensas no Treinamento do Mapa de Barcelona

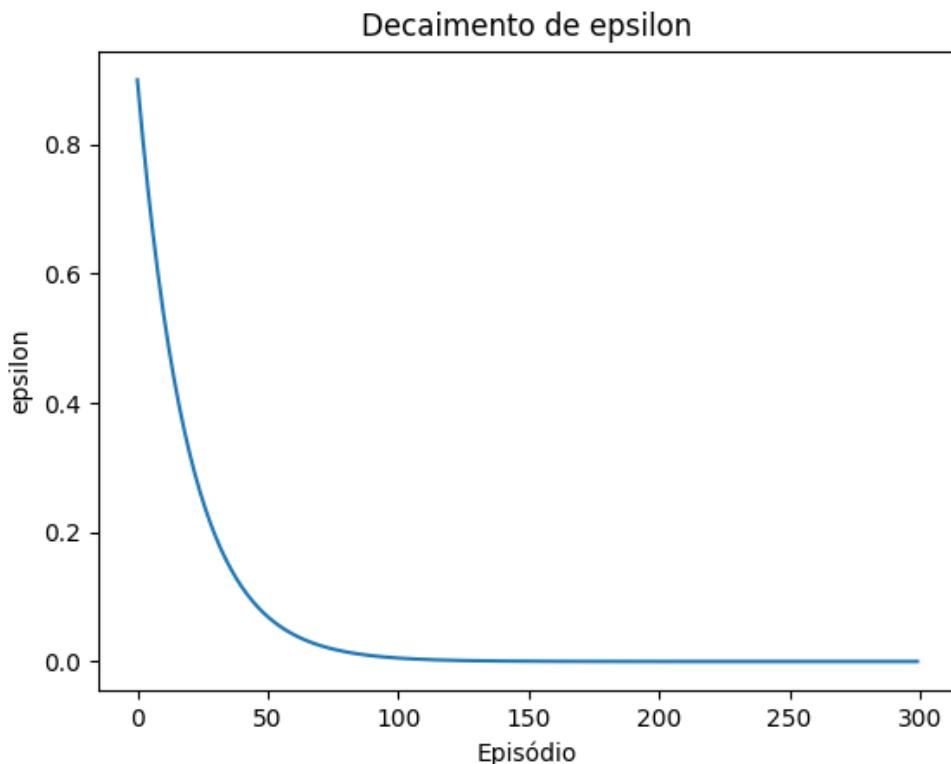


Fonte: O Autor

5.1.1.6 Comportamento de ϵ

De acordo com a Figura 40, o comportamento de ϵ durante o processo de treinamento é semelhante à tendência de passos no ambiente, ou seja, há um decaimento exponencial nos valores à medida que o agente vai colhendo as informações sobre o ambiente. Um ponto interessante a se analisar é que ao comparar com a curva de recompensas, ϵ apresenta um comportamento semelhante ao convergir seu valor, assim no exato momento que há a convergência das recompensas, ϵ também converge ficando claro a importância desse valor durante o processo de treinamento.

Figura 40 – Decaimento de epsilon

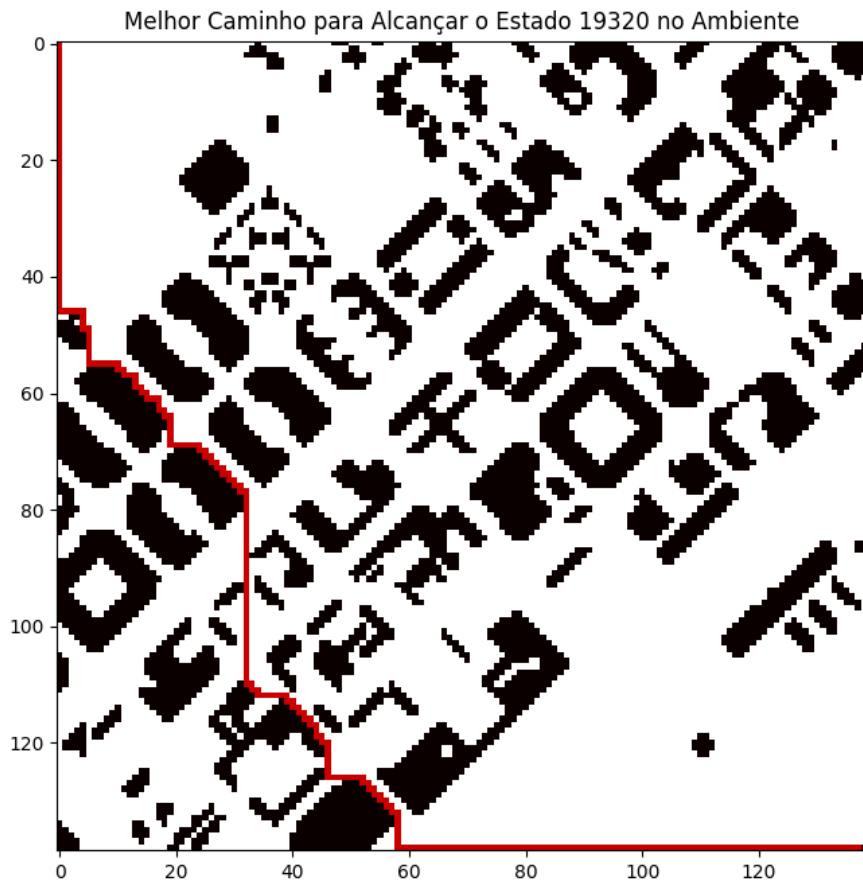


Fonte: O Autor

5.1.1.7 Avaliação do Melhor Caminho

Tendo finalizado o processo de treinamento e o preenchimento da tabela Q se faz necessária a análise do melhor caminho, ou seja, é preciso analisar se o algoritmo encontrou uma solução ótima que fará o robô a se locomover pelo menor caminho evitando obstáculos. O algoritmo de fato encontra uma solução ótima onde pela análise da tabela Q se tem que o robô irá se movimentar por 277 estados até atingir o seu objetivo. A [Figura 41](#) ilustra o trajeto, traçando uma linha vermelha nos estados selecionados pela análise da tabela Q.

Figura 41 – Melhor caminho para o Mapa de Barcelona



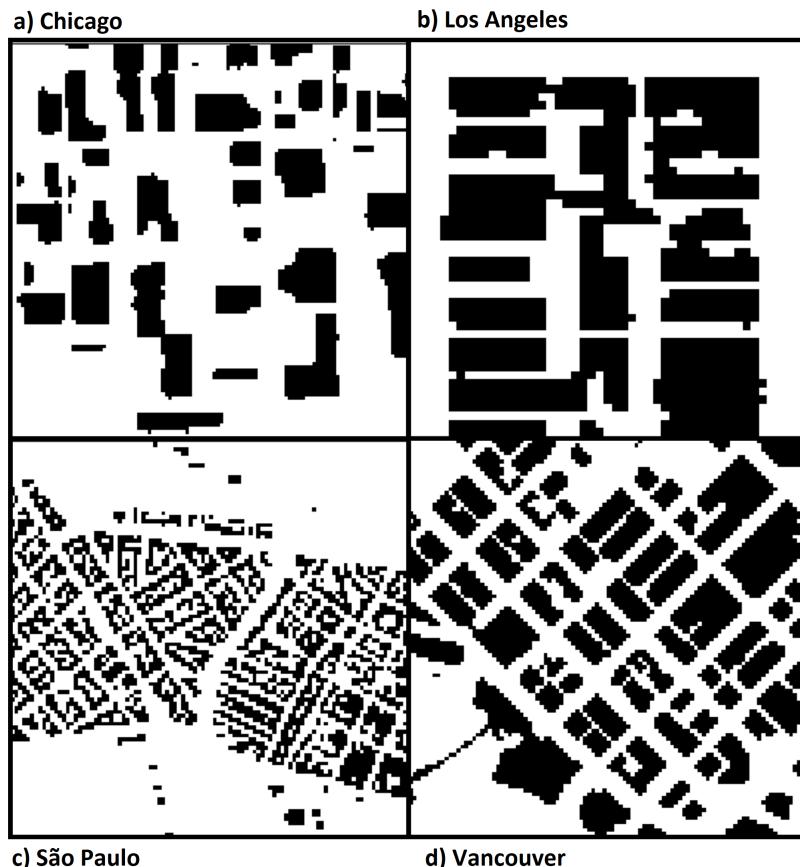
Fonte: O Autor

Vale citar que o algoritmo responsável pela geração da figura que ilustra o melhor caminho se encontra no [Apêndice C](#).

5.2 Estudo de Caso B - Demais Ambientes

De acordo com os resultados apresentados pelo estudo de caso A, foram testados outros ambientes com o intuito de validar o mesmo algoritmo em outros tipos de configurações. Para tal teste foram escolhidos segmentações dos mapas de Chicago, Los Angeles, São Paulo e Vancouver como mostra a [Figura 42](#).

Figura 42 – Estudo de Caso B - Chicago, Los Angeles, São Paulo e Vancouver



Fonte: O Autor

Em ambos os testes, os resultados foram semelhantes ao que foi abordado para a configuração de Barcelona, o algoritmo demonstrou resultados satisfatórios sem a necessidade da alteração de nenhum parâmetro. Com isso não se faz necessário demonstrar os mesmos comportamento de convergência, número de passo e decaimento de ϵ citados na seção anterior uma vez que as características do mapa não estão diretamente relacionados a complexidade de se encontrar uma ótima solução. Contudo, o objetivo da realização de testes com configurações diferentes foi de analisar os variados tamanhos de mapa e densidade de obstáculos. Vale citar que por mais que a figura anterior represente mapas de tamanhos iguais, as dimensões no treinamento são diferentes. O mapa (a) tem tamanho de 116x116, o (b) 49x49, o (c) o 100x100 e o (d) 101x101.

De acordo com a Figura 43 todos os ambientes chegaram numa melhor solução para o planejamento do caminho ao final dos 300 episódios.

Figura 43 – Estudo de Caso B - Chicago, Los Angeles, São Paulo e Vancouver



Fonte: O Autor

Da figura acima se tem que, o traçado vermelho desenhado nos mapas indica qual é o caminho que o robô precisará percorrer dentro do ambiente para atingir seu objetivo final.

O comportamento das recompensas e passos do treinamento dos ambientes pode ser visto na [Tabela 2](#). As segunda e terceira colunas da tabela indicam o menor e o maior valor de recompensa recebido, já as duas últimas indicam os valores limites de passos dados no ambiente. Analisando as colunas da tabela se tem que quanto maior o número de passos, menor será a recompensa. Um exemplo é o mapa de São Paulo onde o agente necessitou dar 34747 passos no pior cenário, o que acarretou em um valor mínimo de recompensa de -447752. Isso ocorre devido ao processo de exploração do ambiente que através da seleção aleatória de ações acarreta na transição para estados que contenham obstáculos, fazendo com que o algoritmo tenha que replanejar o caminho diversas vezes até chegar no seu objetivo.

Quando o algoritmo entra no estado de convergência, que ocorre em torno do episódio 25, o agente então tem informações suficientes através da tabela Q para poder se locomover pelo melhor caminho, fazendo com que a partir desse ponto não tenha grandes variações de valores de recompensa. Tal comportamento diminui o valor de passos necessários para se chegar ao

destino final, planejar uma rota livre de obstáculos e consequentemente eleva o número de recompensas.

Tabela 2 – Avaliação das Recompensas e Passos dados no Treinamento dos Ambientes

Ambiente	Menor Recompensa	Maior Recompensa	Maior Quantidade de Passos	Menor Quantidade de Passos
Chicago	-1953383	4968	50765	231
Los Angeles	-627764	5000	7848	97
São Paulo	-447752	5000	34747	199
Vancouver	-866965	5000	78262	201

Fonte: O Autor

5.3 Comparação do Aprendizado

Tendo finalizado todos os testes com os ambientes propostos foi feita uma comparação entre o algoritmo proposto QLDR e o algoritmo de *Q-learning* clássico para que assim ficasse provado a eficácia no processo de aceleração do treinamento.

A Tabela 3 deixa evidente quais são as diferenças em termos de tempo de treinamento, quantidade de episódios necessários para a convergência do algoritmo e o tamanho do caminho encontrado em ambas abordagens. Vale citar que a abordagem clássica utilizou os mesmos parâmetros de α e γ mantendo a decaída de ϵ .

Tabela 3 – Comparação de Aprendizado entre Algoritmo QLDR e *Q-learning* Clássico

Ambiente	Dimensões do Ambiente	Tempo de treinamento (s) QLDR	Episódios QLDR	Tamanho de Caminho QLDR	Tempo de treinamento (s) <i>Q-learning</i> Clássico	Episódios <i>Q-learning</i> Clássico	Tamanho de Caminho <i>Q-learning</i> Clássico	Variação Tempo QLDR versus Tempo <i>Q-learning</i>
Barcelona	139x139	101,43	300	277	3446,7	3000	277	3298,1%
Chicago	116x116	64,22	300	231	2404,2	5000	231	3643,7%
Los Angeles	49x49	11,22	300	97	70,74	750	101	530,48%
São Paulo	100x100	41,63	300	199	1125,18	3000	199	2602,81%
Vancouver	101x101	89,55	300	201	1784,44	3000	201	1892,67%

Fonte: O Autor

Ao analisar a tabela, se tem que por mais que o algoritmo de *Q-learning* clássico também traga resultados semelhantes, o tempo para realizar o treinamento nos ambientes se torna até 40 vezes maior, demandando cerca de 15 vezes mais episódios tendo um grande custo computacional e deixando evidente que o processo de dupla recompensa consegue definir a trajetória com um prazo menor de tempo e demandando menos episódios. Além disso fica evidente o quanto eficiente a abordagem proposta é ao analisar a variação entre o tempo de treinamento de ambos algoritmos onde a abordagem clássica pode chegar a levar 3298,1% a mais para encontrar uma solução do que o algoritmo QLDR.

Além disso, é de grande importância ponderar que o problema de planejamento de caminho utilizando o algoritmo *Q-learning* é um problema *NP-hard* com complexidade de $O(N^2)$, fazendo com que a solução apresentada possa não ser a mais rápida entre outras abordagens da literatura, entretanto se torna manejável em comparação a outras soluções existentes.

5.4 Resultados Experimentais

Tendo realizado os testes em simulação para averiguar a eficácia do algoritmo proposto, foi definido um teste simples com um robô hexápode em um ambiente de tamanho reduzido para validação de conceito. A construção do robô, o método de localização e demais especificações da configuração do ambiente se encontra no [Apêndice A](#).

Para a definição do espaço de estados de teste, foi analisado o ambiente disposto. Através da observação do espaço é possível definir uma configuração de 4x4 através dos pisos no chão conforme mostra a [Figura 44](#).

Figura 44 – Configuração de Ambiente



Fonte: O Autor

Do ambiente, foram selecionados três obstáculos através da utilização de objetos de formato retangular e da cor preta dispostos nos estados 5,8 e 10 como mostra a [Figura 45](#).

Figura 45 – Disposição de Obstáculos



Fonte: O Autor

A partir da definição dos estados, foi possível então realizar a segmentação binária na figura, entretanto para que isso fosse possível e para facilitar a identificação dos obstáculos a imagem foi recortada evidenciando apenas o ambiente que será utilizado para o processo de locomoção do robô conforme mostra a [Figura 46](#)

Figura 46 – Ambiente Segmentado

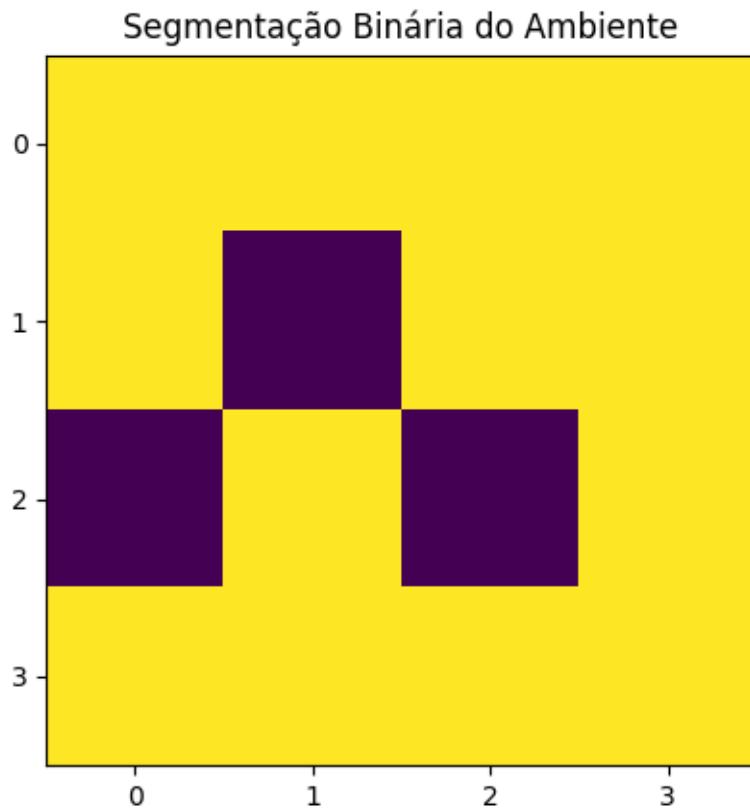


Fonte: O Autor

Com o recorte da imagem, a mesma foi redimensionada para um tamanho de 4x4; isso facilita no processo de identificação dos estados assim sendo, caso a imagem fosse maior a disposição dos *pixels* aumenta e acaba alterando a análise dos obstáculos.

Com a imagem redimensionada foi enfim aplicada a segmentação da imagem como mostra a [Figura 47](#) podendo assim identificar os obstáculos existentes no ambiente conforme mostra a [Figura 48](#).

Figura 47 – Ambiente com Segmentação Binária



Fonte: O Autor

Figura 48 – Obstáculos Através da Segmentação Binária

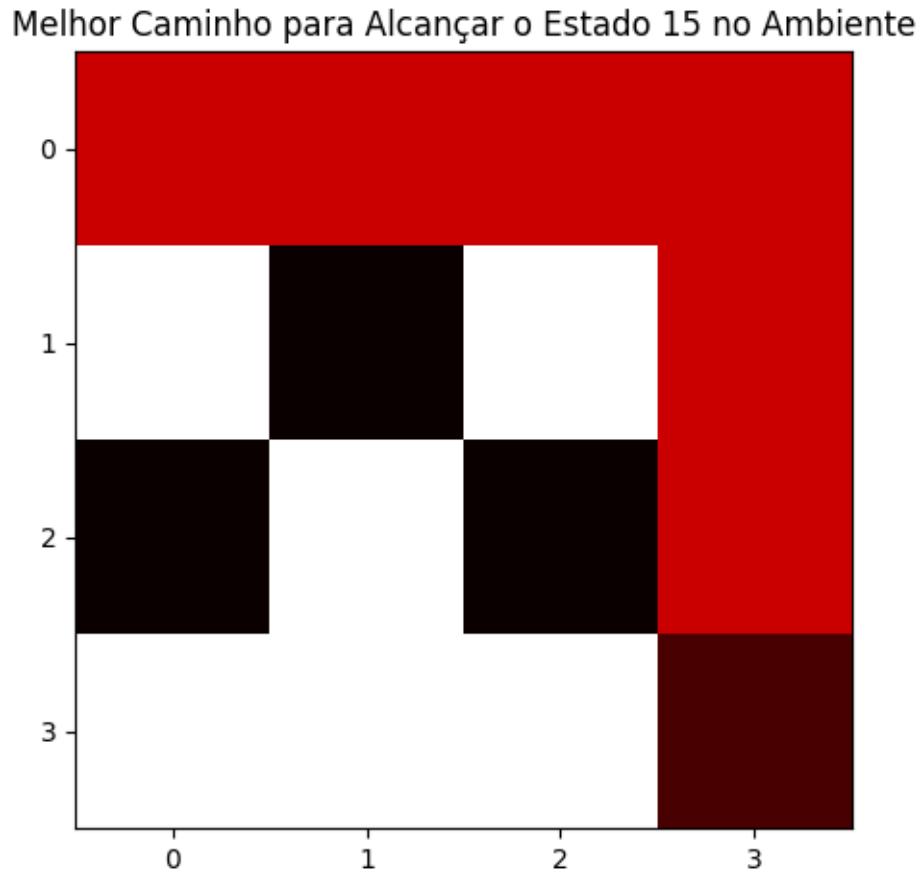
```
Height: 4
Obstáculos identificados nos estados: [5, 8, 10]
```

Fonte: O Autor

Tendo definido o espaço de estados e já sabendo o estado de ações e a função de transição de estados é necessário então treinar o ambiente. O objetivo do robô no ambiente é se locomover do estado 0 até o estado 15 ao longo de 300 episódios e seguindo os mesmos parâmetros de α , γ , ϵ .

Após o processo de treinamento se tem que o ambiente foi solucionado em um período de 0,12 segundos apresentando as mesmas características de convergência. Ao analisar o caminho indicado pelo ambiente se tem que o mesmo foi capaz de encontrar o melhor caminho conforme mostra a Figura 49 onde o caminho é representado pela cor vermelha e os obstáculos pela cor preta.

Figura 49 – Melhor Caminho no Ambiente



Fonte: O Autor

A partir do treinamento, se pode analisar que o robô deve percorrer um total de 6 estados até chegar o objetivo final, onde os estados podem ser definidos pela [Figura 50](#) onde a primeira linha indica de fato os estados, a segunda e a terceira indicam as ações que serão realizadas e a quarta linha indica o tamanho do caminho considerando o estado 0.

Figura 50 – Espaço de Estados, ações e Tamanho de Caminho

```
[0, 1, 2, 3, 7, 11, 15]
[3, 3, 3, 2, 2, 2]
['R', 'R', 'R', 'D', 'D', 'D']
path_length: 7
```

Fonte: O Autor

5.4.1 Mapeamento dos Estados no Ambiente

A partir dos estados que o robô deve percorrer é necessário o mapeamento dos estados no ambiente real para realizar a tradução dos estados em coordenadas. Para isso o robô foi

posicionado em todos os estados do ambiente e com o auxílio do algoritmo de processamento de imagem foi feita a identificação das coordenadas em cada estado para auxiliar na tradução dos estados e na localização do robô no ambiente no processo de locomoção. A [Figura 51](#) mostra o resultado do processamento da imagem para identificação do robô no ambiente através da busca pelos círculos em azul e vermelho posicionados nas extremidades do mesmo.

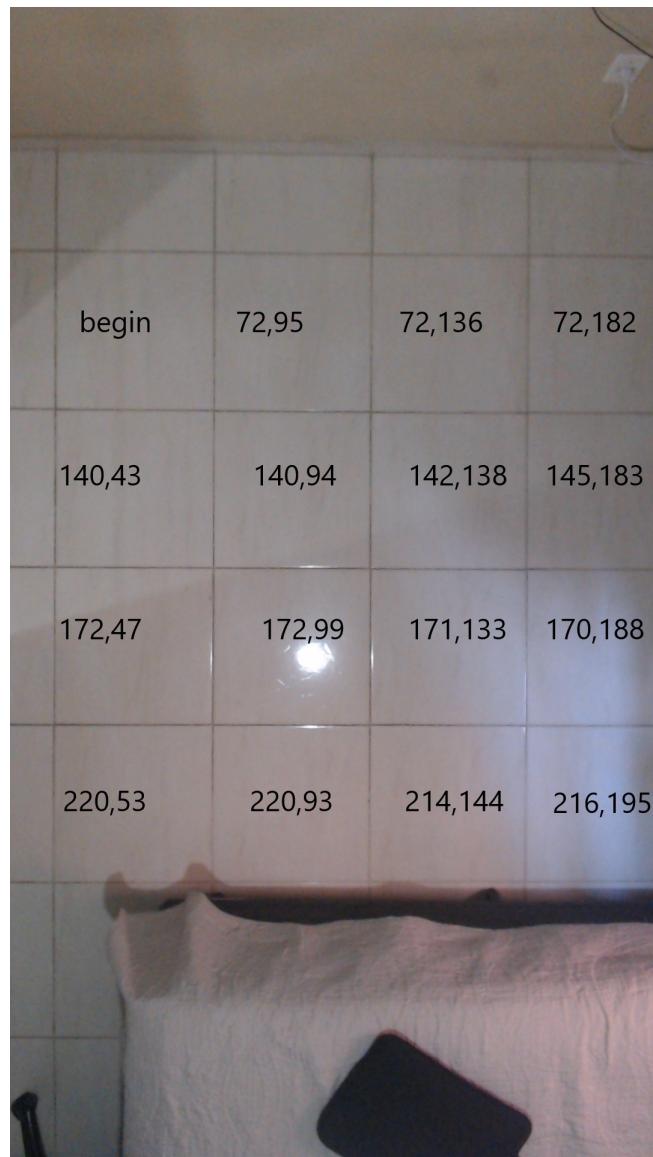
Figura 51 – Identificação do Robô no Ambiente



Fonte: O Autor

Colocando o robô em todos os estados, foi possível extrair informações acerca do ambiente para a tradução dos estados de 0,1,2...15 para coordenadas em x e y . A [Figura 52](#) mostra a tradução dos estados para coordenadas reais.

Figura 52 – Coordenadas dos Estados no Ambiente



Fonte: O Autor

Tendo os estados que o robô deve seguir e as coordenadas reais do ambiente, a lista gerada através do processo de treinamento que indica o melhor caminho para o robô seguir é então traduzida através do algoritmo abaixo. O código opera de maneira bem simples, recebe a lista de estados e traduz em coordenadas para serem utilizadas no algoritmo de locomoção do robô.

```

1 #Algoritmo Tradutor de Estados
2 dict_states = {
3     '1' : '72,95',
4     '2' : '72,136',
5     '3' : '72,182',
6     '4' : '140,43',
7     '5' : '140,94',
8     '6' : '142,138',
9     '7' : '145,183',

```

```
10     '8' : '172,47',
11     '9' : '172,99',
12     '10' : '171,33',
13     '11' : '168,88',
14     '12' : '220,53',
15     '13' : '220,93',
16     '14' : '214,144',
17     '15' : '216,195'
18 }
19
20 def translate_state(list_states):
21     new_states = []
22     for state in list_states:
23         new_states.append(dict_states.get(state))
24     return new_states
```

Ao final da tradução, para o estudo de caso em questão o resultado da tradução dos estados é apresentada através da [Figura 53](#).

Figura 53 – Tradução dos Estados no Ambiente

```
$ python translate.py
['72,95', '72,136', '72,182', '145,183', '168,88', '216,195']
(env)
```

Fonte: O Autor

Por fim, o robô foi posicionado no ambiente junto com seus respectivos obstáculos conforme a [Figura 54](#), entretanto o teste de locomoção não pôde ser realizado devido à problemas técnicos. Um dos servomotores queimou e após o gasto com três outros servos que vieram queimados ou com mau funcionamento do fornecedor decidiu-se então optar por não realizar o teste de locomoção e finalizar com o planejamento do caminho que é o objetivo central desse trabalho.

Figura 54 – Robô Posicionado no Ambiente



Fonte: O Autor

6 Conclusão

O planejamento de caminho sem dúvidas é um dos problemas mais difíceis de solucionar quando se trata da locomoção de robôs dentro de um trajeto. Dentre tantos algoritmos que buscam solucionar esse problema, o algoritmo de *Q-learning* se demonstra uma ótima opção devido à sua facilidade de implementação, entretanto, devido a sua complexidade de $O(N^2)$ torna o processo lento quando se trata de ambientes com uma alta densidade de obstáculos. A partir disso, o algoritmo proposto QLDR se demonstrou uma ótima opção para acelerar esse processo através da utilização de duas recompensas e uma decaída de *epsilon* atingindo seu estado de convergência mais rápido e trazendo a melhor solução utilizando menos episódios no treinamento.

Mesmo apresentando uma grande melhora em relação à sua versão clássica, a versão de QLDR ainda pode ser um pouco lenta dependendo do tipo de ambiente, isso se deve ao fato de que ao iniciar explorando o ambiente o robô apresenta uma grande probabilidade de escolher ações que não são ótimas e ficar durante muito tempo preso em um determinado estado.

No desenvolvimento do projeto foram encontradas muitas dificuldades, de início a proposta era de aplicar *Q-learning* apenas no processo de locomoção do robô, ou seja, fazer o robô aprender movimentos de locomoção através da exploração de ações. Entretanto, esse tipo de treinamento não seria possível ser realizado *online* devido à baixa autonomia da bateria para controlar os atuadores e a dificuldade de colher informações de cada um dos servomotores tornava inviável o desenvolvimento do projeto. Isso fez com que na metade do semestre o tema do trabalho fosse alterado para o planejamento do caminho. Durante a construção do algoritmo as dificuldades foram em encontrar valores de α , γ , ϵ , decaída de ϵ e número de episódios ideais que pudessem ser aplicados em um variado conjunto de ambientes, tal processo demandou semanas de testes e tempo gasto no processo de treinamento. Outra dificuldade encontrada foi em questão ao tamanho das figuras para representar o ambiente, devido à grande complexidade do algoritmo, matrizes com dimensões acima de 150x150 levam muito tempo para serem processadas tornando o algoritmo inviável para grandes imagens.

Outra dificuldade encontrada foi em relação a utilização do robô, pois foi necessário um processo de montagem, manutenção de componentes, adequação de ambiente e familiarização com o *software*. Além disso, com o algoritmo QLDR pronto para realizar testes no ambiente real um dos servomotores queimou, foram realizadas três consecutivas compras de um único fornecedor brasileiro em que todos os servos vieram com defeito fazendo com que, devido ao prazo de entrega do trabalho, o teste de locomoção não pudesse ser finalizado.

Contudo, como o objetivo do trabalho era a construção de um algoritmo que acelerasse o processo de planejamento do melhor caminho em *Q-learning*, os resultados obtidos se mos-

traram satisfatórios deixando uma gama de possibilidades de melhoria com trabalhos futuros.

6.1 Trabalhos Futuros

Como forma de indicar futuras implementações do algoritmo, a lista a seguir apresenta sugestões que podem ser exploradas.

- Mesclagem do algoritmo QLDR com outras abordagens de planejamento de caminho como por exemplo, a abordagem de Campos Potenciais ou alguma abordagem clássica como por exemplo *Dijkstra*;
- Melhoria no algoritmo de reconhecimento de obstáculos utilizando processamento de imagem;
- Implementação de QLDR com obstáculos móveis, ou seja, caso encontre um obstáculo não mapeado o robô possa desviar e recalcular a trajetória;
- Melhoria no processo de localização do robô no ambiente;
- Implementar algoritmo de QLDR melhorado com ambientes em larga escala;
- Alteração do algoritmo QLDR para levar em consideração o tamanho real do robô;
- Melhoria na estrutura do robô e autonomia de bateria.

Referências

- AMMAR, A. et al. Relaxed dijkstra and a* with linear complexity for robot path planning problems in large-scale grid environments. *Soft Computing*, Springer, v. 20, n. 10, p. 4149–4171, 2016. Citado na página 43.
- ARAUJO, G. M.; MENDONÇA, M.; FREIRE, E. Reconhecimento automático de objetos baseado em cor e forma para aplicações em robótica. In: *XVII Congresso Brasileiro de Automática*. [S.l.: s.n.], 2008. p. 18. Citado 4 vezes nas páginas 34, 35, 107 e 108.
- BADAWY, A.; ZAGHLOUL, A.; HUSSEIN, W. Dynamics and measuring of feet force distributions of six-legged robot. In: . [S.l.: s.n.], 2016. Citado na página 24.
- BEER, R. D. et al. Biologically inspired. *Communications of the ACM*, v. 40, n. 3, p. 31, 1997. Citado na página 23.
- BRADSKI, G. The opencv library. *Dr Dobb's J. Software Tools*, 2000. Citado na página 109.
- CARBONE, G.; CECCARELLI, M. Legged robotic systems. In: *Cutting Edge Robotics*. [S.l.]: IntechOpen, 2005. Citado 3 vezes nas páginas 14, 21 e 23.
- CASSANDRAS, C. G.; LAFORTUNE, S. *Introduction to discrete event systems*. [S.l.]: Springer Science & Business Media, 2009. Citado na página 37.
- CLARK, J. E. et al. Biomimetic design and fabrication of a hexapedal running robot. In: *IEEE Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164)*. [S.l.], 2001. v. 4, p. 3643–3649. Citado na página 23.
- CORMEN, T. H. et al. *Introduction to algorithms*. [S.l.]: MIT press, 2009. Citado 2 vezes nas páginas 15 e 26.
- CUNHA, A. L. B. N. d. *Sistema automático para obtenção de parâmetros do tráfego veicular a partir de imagens de vídeo usando OpenCV*. Tese (Doutorado) — Universidade de São Paulo, 2013. Citado 4 vezes nas páginas 31, 32, 33 e 34.
- DATAHUBBS. *Intro To Q-Learning*. 2021. Disponível em: <<https://www.datahubbs.com/intro-to-q-learning/>>. Citado na página 40.
- DONALD, B. R. *Motion Planning with Six Degrees of Freedom*. [S.l.], 1984. Citado na página 25.
- DUDEK, G.; JENKIN, M. *Computational principles of mobile robotics*. [S.l.]: Cambridge university press, 2010. Citado 4 vezes nas páginas 18, 25, 28 e 29.
- ELETRONICHUBS. *Arduíno Mega PinOut*. Disponível em: <<https://www.electronicshub.org/arduino-mega-pinout/>>. Citado na página 100.
- ESQUEF, I. A.; ALBUQUERQUE, M. P. d.; ALBUQUERQUE, M. P. d. Processamento digital de imagens. *Rio de Janeiro*, v. 12, 2003. Citado na página 33.

- FABMODELISMO. *Bateria de Lipe 2s 500mah*. Disponível em: <<https://www.fabmodelismo.com.br/produto/pecas-e-acessorios/baterias-em-geral/8278-bateria-de-lipo-2s-5000mah-20c-zippy-flightmax>>. Citado na página 101.
- FARIA, D. Analise e processamento de imagem. *Faculdade de Engenharia da Universidade do Porto*, 2010. Citado 2 vezes nas páginas 30 e 33.
- FIALA, M. Vision guided control of multiple robots. In: IEEE. *First Canadian Conference on Computer and Robot Vision, 2004. Proceedings*. [S.l.], 2004. p. 241–246. Citado na página 28.
- GARCIA, E.; ESTREMERA, J.; SANTOS, P. G. D. A classification of stability margins for walking robots. *Robotica*, v. 20, n. 6, p. 595–606, 2002. Citado 2 vezes nas páginas 22 e 23.
- GARCIA, M. P. et al. Path planning for autonomous mobile robot navigation with ant colony optimization and fuzzy cost function evaluation. *Applied Soft Computing*, Elsevier, v. 9, n. 3, p. 1102–1110, 2009. Citado na página 43.
- GONZALEZ, R. C.; WOODS, R. E. et al. *Digital image processing*. [S.l.]: Prentice hall Upper Saddle River, NJ, 2002. Citado 3 vezes nas páginas 29, 30 e 31.
- GREGORIO, P.; AHMADI, M.; BUEHLER, M. Design, control, and energetics of an electrically actuated legged robot. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, IEEE, v. 27, n. 4, p. 626–634, 1997. Citado na página 23.
- HARDARSON, F. *Locomotion for difficult terrain*. [S.l.]: Citeseer, 1998. Citado na página 19.
- HOU, F. et al. Wheeled locomotion for payload carrying with modular robot. In: IEEE. *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*. [S.l.], 2008. p. 1331–1337. Citado na página 19.
- HU, J. *Artificial Intelligence Now: Reinforcement learning explained*. [S.l.]: O'Reilly Media, Inc, 2017. Citado na página 39.
- HUNG, S.-M.; GIVIGI, S. N. A q-learning approach to flocking with uavs in a stochastic environment. *IEEE transactions on cybernetics*, IEEE, v. 47, n. 1, p. 186–197, 2016. Citado 2 vezes nas páginas 38 e 46.
- ISHII, S.; YOSHIDA, W.; YOSHIMOTO, J. Control of exploitation–exploration meta-parameter in reinforcement learning. *Neural networks*, Elsevier, v. 15, n. 4-6, p. 665–687, 2002. Citado na página 40.
- JEON, S. M.; KIM, K. H.; KOPFER, H. Routing automated guided vehicles in container terminals through the q-learning technique. *Logistics Research*, Springer, v. 3, n. 1, p. 19–27, 2011. Citado na página 44.
- KAELBLING, L. P.; LITTMAN, M. L.; MOORE, A. W. Reinforcement learning: A survey. *Journal of artificial intelligence research*, v. 4, p. 237–285, 1996. Citado 2 vezes nas páginas 36 e 37.
- KAMBHAMPATI, S.; DAVIS, L. Multiresolution path planning for mobile robots. *IEEE Journal on Robotics and Automation*, IEEE, v. 2, n. 3, p. 135–145, 1986. Citado na página 25.
- KANSAL, S.; MARTIN, B. *Reinforcement Q-Learning from Scratch in Python with OpenAI Gym*. 2021. Disponível em: <<https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/>>. Citado 2 vezes nas páginas 40 e 41.

- KAVRAKI, L. E. et al. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, IEEE, v. 12, n. 4, p. 566–580, 1996. Citado 3 vezes nas páginas [15](#), [26](#) e [43](#).
- KIM, K. H.; TANCHOCO, J. Economical design of material flow paths. *The International Journal of Production Research*, Taylor & Francis, v. 31, n. 6, p. 1387–1407, 1993. Citado na página [44](#).
- LAVALLE, S. M. et al. Rapidly-exploring random trees: A new tool for path planning. Ames, IA, USA, 1998. Citado na página [26](#).
- LAZARIC, A.; RESTELLI, M.; BONARINI, A. Reinforcement learning in continuous action spaces through sequential monte carlo methods. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2008. p. 833–840. Citado na página [40](#).
- LEE, T.-T.; LIAO, C.-M.; CHEN, T.-K. On the stability properties of hexapod tripod gait. *IEEE Journal on Robotics and Automation*, IEEE, v. 4, n. 4, p. 427–434, 1988. Citado na página [23](#).
- LIM, J. K. et al. Designing guide-path networks for automated guided vehicle system by using the q-learning technique. *Computers & industrial engineering*, Elsevier, v. 44, n. 1, p. 1–17, 2003. Citado na página [44](#).
- MAHAJAN, A.; FIGUEROA, F. Four-legged intelligent mobile autonomous robot. *Robotics and Computer-Integrated Manufacturing*, Elsevier, v. 13, n. 1, p. 51–61, 1997. Citado na página [19](#).
- MATHWORKS. *Matlab R2012a User's Guide*. [S.l.]: MathWorks, 2012. Citado 2 vezes nas páginas [34](#) e [35](#).
- MATSUSHITA, N. et al. Id cam: A smart camera for scene capturing and id recognition. In: *IEEE. The Second IEEE and ACM International Symposium on Mixed and Augmented Reality, 2003. Proceedings*. [S.l.], 2003. p. 227–236. Citado na página [28](#).
- MCGHEE, R. B.; FRANK, A. A. On the stability properties of quadruped creeping gaits. *Mathematical Biosciences*, Elsevier, v. 3, p. 331–351, 1968. Citado na página [22](#).
- MCGHEE, R. B.; ISWANDHI, G. I. Adaptive locomotion of a multilegged robot over rough terrain. *IEEE transactions on systems, man, and cybernetics*, IEEE, v. 9, n. 4, p. 176–182, 1979. Citado na página [22](#).
- MONARD, M. C.; BARANAUSKAS, J. A. Conceitos sobre aprendizado de máquina. *Sistemas inteligentes-Fundamentos e aplicações*, v. 1, n. 1, p. 32, 2003. Citado na página [35](#).
- MONTEIRO, N. S. et al. Localização e planejamento de movimento de robôs móveis em ambientes internos utilizando processos de decisão de markov. Universidade Federal de Minas Gerais, 2020. Citado 3 vezes nas páginas [28](#), [29](#) e [37](#).
- NIE, B. et al. Capability iteration network for robot path planning. *arXiv preprint arXiv:2104.14300*, 2021. Citado na página [25](#).
- NIEDERBERGER, C.; RADOVIC, D.; GROSS, M. Generic path planning for real-time applications. In: *IEEE. Proceedings Computer Graphics International, 2004*. [S.l.], 2004. p. 299–306. Citado na página [47](#).

- PANI, A. et al. Evaluating public acceptance of autonomous delivery robots during covid-19 pandemic. *Transportation research part D: transport and environment*, Elsevier, v. 89, p. 102600, 2020. Citado na página 45.
- PELLEGRINI, J.; WAINER, J. Processos de decisão de markov: um tutorial. *Revista de Informática Teórica e Aplicada*, v. 14, n. 2, p. 133–179, 2007. Citado 2 vezes nas páginas 36 e 37.
- PENHARBEL, E. A. et al. Filtro de imagem baseado em matriz rgb de cores-padrão para futebol de robôs. *Submetido ao I Encontro de Robótica Inteligente*, 2004. Citado na página 35.
- PRATIHAR, D. K.; DEB, K.; GHOSH, A. Optimal path and gait generations simultaneously of a six-legged robot using a ga-fuzzy approach. *Robotics and Autonomous Systems*, Elsevier, v. 41, n. 1, p. 1–20, 2002. Citado na página 43.
- PURCARU, C. et al. Optimal robot path planning using gravitational search algorithm. *Int. J. Artif. Intell.*, v. 10, n. 13, p. 1–20, 2013. Citado na página 26.
- QUEIROZ, J. E. R. de; GOMES, H. M. Introdução ao processamento digital de imagens. *Rita*, v. 13, n. 2, p. 11–42, 2006. Citado 2 vezes nas páginas 30 e 33.
- RAIBERT, M. H. Legged robots. *Commun. ACM*, v. 29, n. 6, p. 499–514, 1986. Citado na página 22.
- RIBEIRO, C. H. C. A tutorial on reinforcement learning techniques. In: *Supervised Learning Track Tutorials of the 1999 International Joint Conference on Neuronal Networks*. [S.l.: s.n.], 1999. Citado na página 40.
- RODRIGUEZ, A. G. G.; GARCIA, F. C. Improving the energy efficiency and speed of walking robots. *Mechatronics*, Elsevier, v. 24, n. 5, p. 476–488, 2014. Citado na página 14.
- RUZZON, T. A. L. *Navegação de um robô hexápode usando sistema de visão externa*. 2019. Citado 18 vezes nas páginas 28, 34, 42, 48, 96, 97, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 111 e 113.
- SANTOS, J. L. dos; JÚNIOR, C. L. N. et al. Coordenação dos atuadores das pernas de robôs móveis usando aprendizado por reforço: simulação e implementação. *Sba: Controle & Automação*, SciELO Brasil, v. 23, n. 1, p. 78, 2012. Citado 4 vezes nas páginas 14, 18, 19 e 21.
- SARIFF, N.; BUNIYAMIN, N. An overview of autonomous mobile robot path planning algorithms. In: IEEE. *2006 4th student conference on research and development*. [S.l.], 2006. p. 183–188. Citado na página 25.
- SCHWARZPLAN. *Site plans figure ground plans for architects, planners and creatives*. Disponível em: <<https://schwarzplan.eu/en/>>. Citado 3 vezes nas páginas 46, 65 e 69.
- SHAHRIARI, M. *Design, implementation and control of a hexapod robot using reinforcement learning approach*. Tese (Doutorado) — Master's thesis, Sharif University of Technology, Int. Campus, 2013. Citado 8 vezes nas páginas 19, 21, 22, 23, 24, 25, 42 e 43.
- SMITH, R. C.; CHEESEMAN, P. On the representation and estimation of spatial uncertainty. *The international journal of Robotics Research*, Sage Publications Sage CA: Thousand Oaks, CA, v. 5, n. 4, p. 56–68, 1986. Citado na página 27.

- SONG, Y. et al. An efficient initialization approach of q-learning for mobile robots. *International Journal of Control, Automation and Systems*, Springer, v. 10, n. 1, p. 166–172, 2012. Citado na página [44](#).
- SPENNEBERG, D.; MCCULLOUGH, K.; KIRCHNER, F. Stability of walking in a multilegged robot suffering leg loss. In: IEEE. *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04*. 2004. [S.l.], 2004. v. 3, p. 2159–2164. Citado na página [15](#).
- THRUN, S. Probabilistic robotics. *Communications of the ACM*, ACM New York, NY, USA, v. 45, n. 3, p. 52–57, 2002. Citado na página [28](#).
- TING, L.; BLICKHAN, R.; FULL, R. J. Dynamic and static stability in hexapedal runners. *Journal of Experimental Biology*, The Company of Biologists Ltd, v. 197, n. 1, p. 251–269, 1994. Citado na página [22](#).
- WANG, M. et al. Fuzzy logic based robot path planning in unknown environment. In: IEEE. *2005 International Conference on Machine Learning and Cybernetics*. [S.l.], 2005. v. 2, p. 813–818. Citado na página [27](#).
- WARREN, C. W. Fast path planning using modified a* method. In: IEEE. *[1993] Proceedings IEEE International Conference on Robotics and Automation*. [S.l.], 1993. p. 662–667. Citado 2 vezes nas páginas [15](#) e [27](#).
- YU, J.; LAVALLE, S. M. Optimal multirobot path planning on graphs: Complete algorithms and effective heuristics. *IEEE Transactions on Robotics*, IEEE, v. 32, n. 5, p. 1163–1177, 2016. Citado na página [25](#).
- ZHANG, H. et al. Path planning of industrial robot based on improved rrt algorithm in complex environments. *IEEE Access*, IEEE, v. 6, p. 53296–53306, 2018. Citado 2 vezes nas páginas [15](#) e [26](#).
- ZHANG, Q. et al. Reinforcement learning in robot path optimization. *JSW*, Citeseer, v. 7, n. 3, p. 657–662, 2012. Citado na página [44](#).
- ZHAO, M. et al. The experience-memory q-learning algorithm for robot path planning in unknown environment. *IEEE Access*, IEEE, v. 8, p. 47824–47844, 2020. Citado 6 vezes nas páginas [27](#), [39](#), [40](#), [44](#), [48](#) e [53](#).

Apêndices

APÊNDICE A – Ambiente Experimental para Validação de Conceito

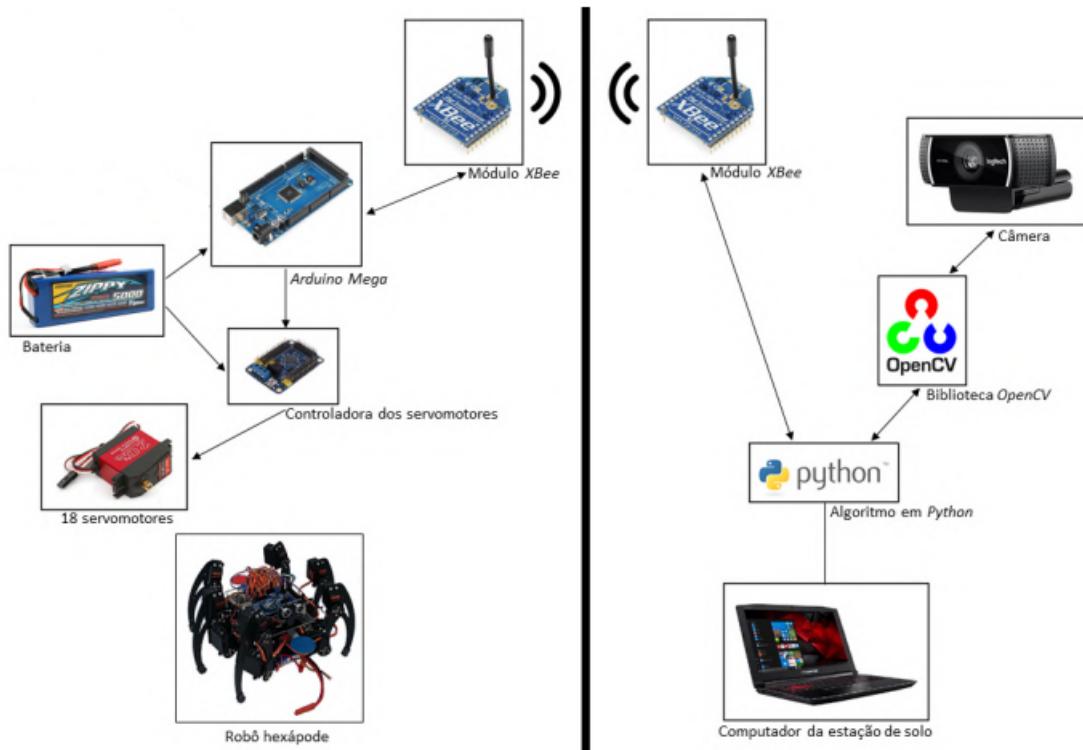
Neste capítulo será apresentado a composição do ambiente experimental construído para validação do planejamento de caminho realizado de forma *offline*.

A.1 Processo de Locomoção do Robô

O processo de locomoção do robô real consiste nas etapas de montagem do robô hexápode real e a construção do algoritmo de guiagem pelo ambiente após o treinamento realizado pela etapa de planejamento de caminho.

O processo é composto por um modelo baseado no trabalho feito por (RUZZON, 2019) utilizando de comunicação sem fio entre um algoritmo de guiagem e o robô através da utilização de uma câmera externa que monitora o projeto, o fluxo desse processo pode ser visto na [Figura 55](#) onde constam todos os elementos dessa etapa bem como o fluxo das informações entre os componentes.

Figura 55 – Diagrama dos Componentes do Sistema de Locomoção



Fonte: ([RUZZON, 2019](#))

Os componentes representados na figura acima exercem papéis importantes no sistema de locomoção como um todo onde cada parte depende da outra para o devido funcionamento.

A parte esquerda da imagem elucida os componentes que compõem a unidade móvel, ou seja, o robô hexápode em si, onde a finalidade de cada componente é definida como:

- **Módulo XBee:** Componente eletrônico responsável pelo processo de comunicação de via dupla entre a estação que monitora o ambiente e o robô.
- **Arduíno Mega 2560:** Módulo embarcado no robô responsável por definir a sequência de movimentos necessários para a movimentação de cada um dos servomotores.
- **Bateria:** Fonte de tensão responsável pelo fornecimento de energia para todos os componentes do robô.
- **Controladora dos servomotores:** Dispositivo eletrônico também conhecido como placa PWM responsável pelo controle da informação fornecida pelo Arduíno. Realiza a definição das possíveis de cada um dos servomotores através do envio de suscetivos sinais seriais de entrada.

- **Servomotores:** Componentes principais para estrutura do robô e possuem a responsabilidade de realizar os movimentos de locomoção do robô.
- **Robô hexápode:** Unidade móvel contendo os componentes citados anteriormente.

Na porção direita estão representados os componentes responsáveis pelo processo de localização e monitoramento do robô, assim como na representação da parte esquerda da imagem, os componentes exercem os seguintes papéis:

- **Módulo XBee:** Diferente do módulo da parte direita, o componente é instalado no computador que monitora o ambiente para a transferência de informações e comunicação entre o robô e a estação solo.
- **Câmera:** Dispositivo externo que exerce o papel da percepção do ambiente e capturar imagens que serão utilizadas para estimar a localização corrente do robô.
- **Biblioteca OpenCV:** Biblioteca utilizada no algoritmo em *python* para realização do processamento das imagens fornecidas pela câmera.
- **Algoritmo em Python:** Funcionando como "cérebro" do sistema o algoritmo é responsável pela troca de informações acerca do ambiente com o robô através do processamento das imagens.
- **Computador da estação solo:** Dispositivo que interliga os componentes da estação solo através do recebimento de informações que auxiliam no processamento das imagens e definição dos passos de locomoção do robô

A.1.1 Montagem e Adaptação do Robô

O robô utilizado nesse projeto é composto por 18 servomotores de 270° (modelo DS3218MG) com rolamentos de metal e um torque elevado de 20kg/cm. A [Figura 56](#) mostra um modelo do servo utilizado.

Figura 56 – Servomotor DS3218MG



Fonte: <https://www.rcmoment.com/pt/p-rm10488.html>

O robô é um modelo bioinspirado em insetos, para isso, os 18 servomotores estão distribuídos em suas 6 pernas, onde em cada perna se encontram 3 servomotores. Cada perna apresenta três componentes principais, a coxa, o fêmur e a tíbia conforme mostra a Figura 58 .

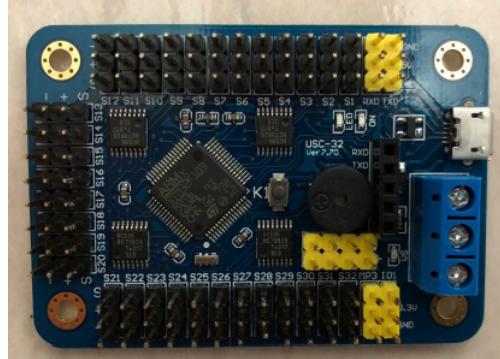
Figura 57 – Segmentação de Perna



Fonte: (RUZZON, 2019)

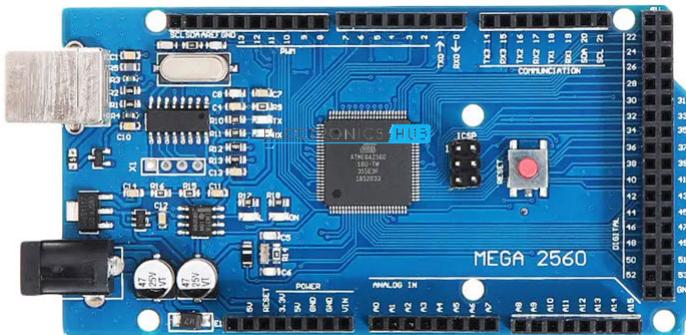
Para o devido controle das pernas, o robô conta uma placa controladora PWM de 32 canais que têm por intuito controlar cada um dos servos da maneira correta através do padrão de comunicação serial I2C com o Arduíno. O modelo da placa pode ser visto na Figura 58.

Figura 58 – Placa Controladora de Servomotores

Fonte: ([RUZZON, 2019](#))

Como já apresentado anteriormente, para o sistema embarcado do robô foi utilizado um Arduíno Mega 2560 semelhante ao da [Figura 59](#). O microcontrolador da placa é um AVR ATM2560 de 8 bits, possuindo uma arquitetura *Harvard*, ou seja, uma arquitetura com duas memórias. A primeira é a memória de programas possuindo 256kB de capacidade de armazenamento, e a segunda é a memória de dados possuindo esta 8kB de capacidade de armazenamento.

Figura 59 – Arduíno Mega 2560

Fonte: [ELETRONICHUBS](#)

Além das especificações já citadas, esse modelo de Arduíno é composto por 54 pinos de comunicação digital e 16 pinos de entrada analógica, além disso, possui quatro interfaces de comunicação serial e uma de comunicação I2C. O carregamento de programas no microcontrolador é feito através da conexão USB com a IDE nativa instalada no computador. Para facilidade de manuseio e instalação, o Arduíno e a placa controladora de servos foram fixadas em um placa acrílica de 16cm x 10cm servindo como base central do corpo do robô

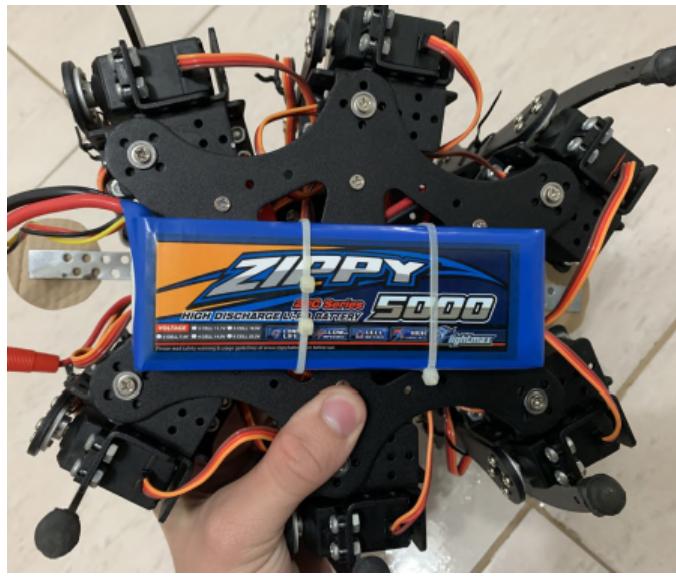
Buscando a autonomia do robô e uma melhor facilidade na sua locomoção sem precisar estar conectado em uma fonte de tensão, foi utilizada uma bateria do modelo *Zippy 20 C Series* semelhante ao mostrado na [Figura 60](#).

Figura 60 – Bateria LiPo

Fonte: [FABMODELISMO](#)

A bateria de Lítio-Polímero (LiPo) fornece uma tensão de até 7,4V com uma corrente contínua de 5000mAh, o suficiente para garantir uma autonomia de 15 minutos para o funcionamento dos servomotores, além de ser compatível com os demais componentes que auxiliam na locomoção do robô. A bateria então, foi instalada na parte inferior da base central do robô como mostra a Figura 61

Figura 61 – Instalação da Bateria no Robô

Fonte: ([RUZZON, 2019](#))

Com o intuito de se realizar uma comunicação sem a utilização de fios entre o robô e a estação solo, foi então mantida a utilização da tecnologia *XBee*, devido a sua fácil integração com o projeto e implementação do algoritmo de envio e recebimento de mensagens utilizando *Python*.

Na [Figura 62](#) apresenta os dois componentes responsáveis pelo processo de troca de informações no processo de locomoção. Na esquerda se encontra o módulo *XBee Pro* que é conectado no computador e responsável pelo envio e recebimento de dados na estação solo, já à direita se encontra o outro módulo que foi inserido em uma *shield* compatível com o Arduíno responsável pelo envio e recebimento de dados do robô

Figura 62 – Módulos *XBee*



Fonte: O Autor

Com os módulos devidamente apresentados, é necessário então realizar a conexão dos componentes. Primeiramente, os servomotores foram conectados na placa controladora de servo de acordo com a [Tabela 4](#), onde a referência da disposição dos servomotores está baseada na segmentação das pernas (coxa, tíbia e fêmur) apresentada anteriormente. Da tabela, se tem que cada número após a letra 'S' indica o número do canal da placa.

Tabela 4 – Conexão entre canais de placa controladora e servomotores

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

Fonte:([RUZZON, 2019](#)) (adaptado)

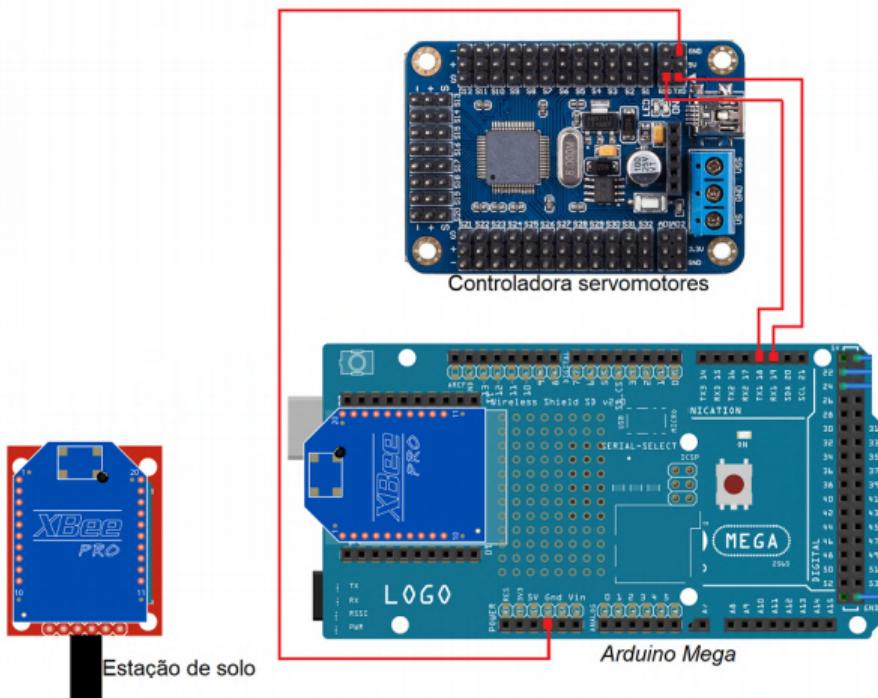
Por fim, para embarcar o Arduino no robô a conexão foi feita através das portas definidas na [Tabela 5](#) e no diagrama representando as conexões reais entre os componentes como mostra a [Figura 63](#) onde no esquemático foram abstraídas as conexões de alimentação e ligações com os servomotores.

Tabela 5 – Conexões do Arduino

Conexão	Tipo de comunicação	Função
RX1	Serial	Porta responsável pelo recebimento de dados provenientes da placa controladora de servomotores
TX1	Serial	Porta responsável pelo envio de dados do Arduíno para a placa controladora de servomotores
<i>Shield XBee</i>	Serial	Comunicação XBee para troca de dados entre robô e estação de solo

Fonte:O Autor

Figura 63 – Esquemático sistema embarcado no robô

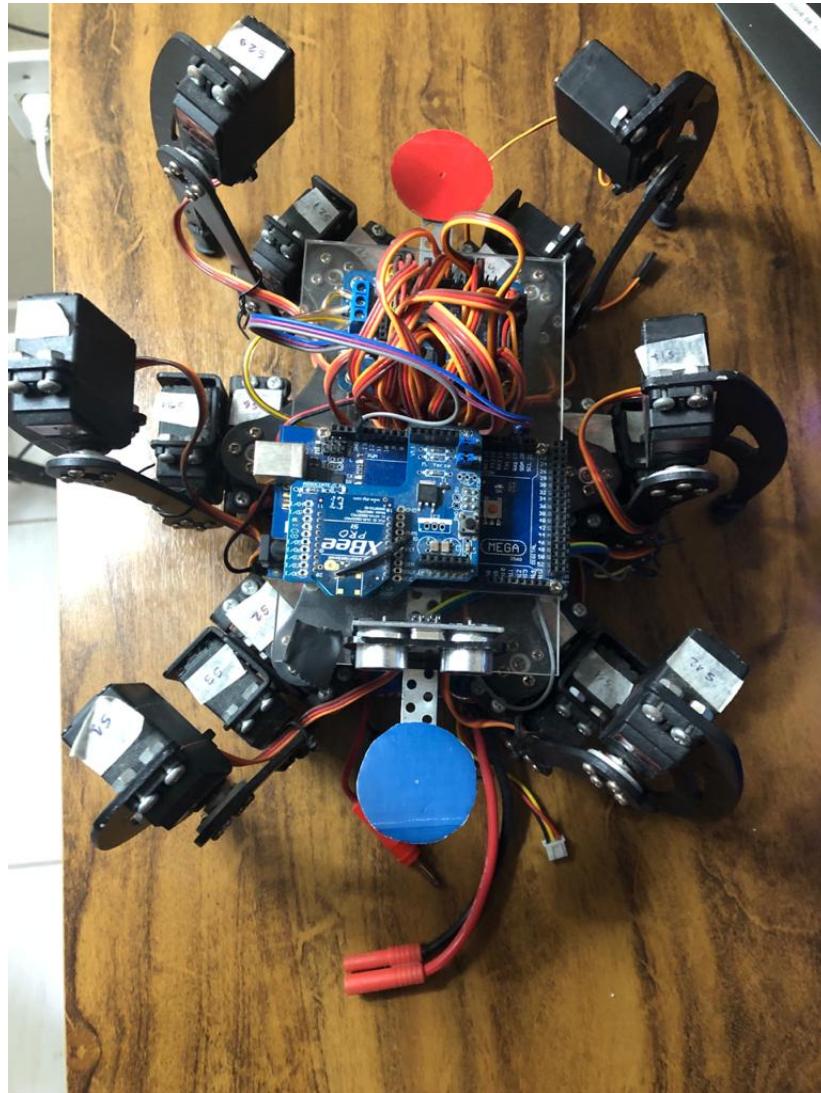


Fonte: ([RUZZON, 2019](#)) (adaptado)

Tendo realizada todas as conexões entre os módulos, a [Figura 64](#) apresenta o robô hexápode finalizado. Uma observação a ser feita é que no projeto original desenvolvido por ([RUZZON, 2019](#)) foi utilizado um sensor ultrassônico para identificar possíveis obstáculos, a utili-

zação de tal componente foi desprezada neste trabalho pois o algoritmo de planejamento de caminho já é capaz de identificar e desviar de possíveis obstáculos pré-determinados.

Figura 64 – Hexápode concluído



Fonte: O Autor

A.1.2 Controle de Locomoção do Robô

Para o processo de locomoção foi seguida a mesma abordagem implementada por (RUZ-ZON, 2019) em seu trabalho, onde o padrão da marcha dos robôs é fazer com que as pernas sejam divididas em grupos de ação ou do inglês, *Action Groups*. A placa controladora mencionada na seção anterior apresenta espaço interno de memória reservado para gravação dos AG que definem os movimentos de alteração da posição dos ângulos de cada servomotor.

Conforme testes feitos, foi seguida mesma lógica implementada no projeto original, a Tabela 6 apresenta os AG, a referência de movimentos que o robô como um todo deve realizar, bem como o período de duração de cada movimento.

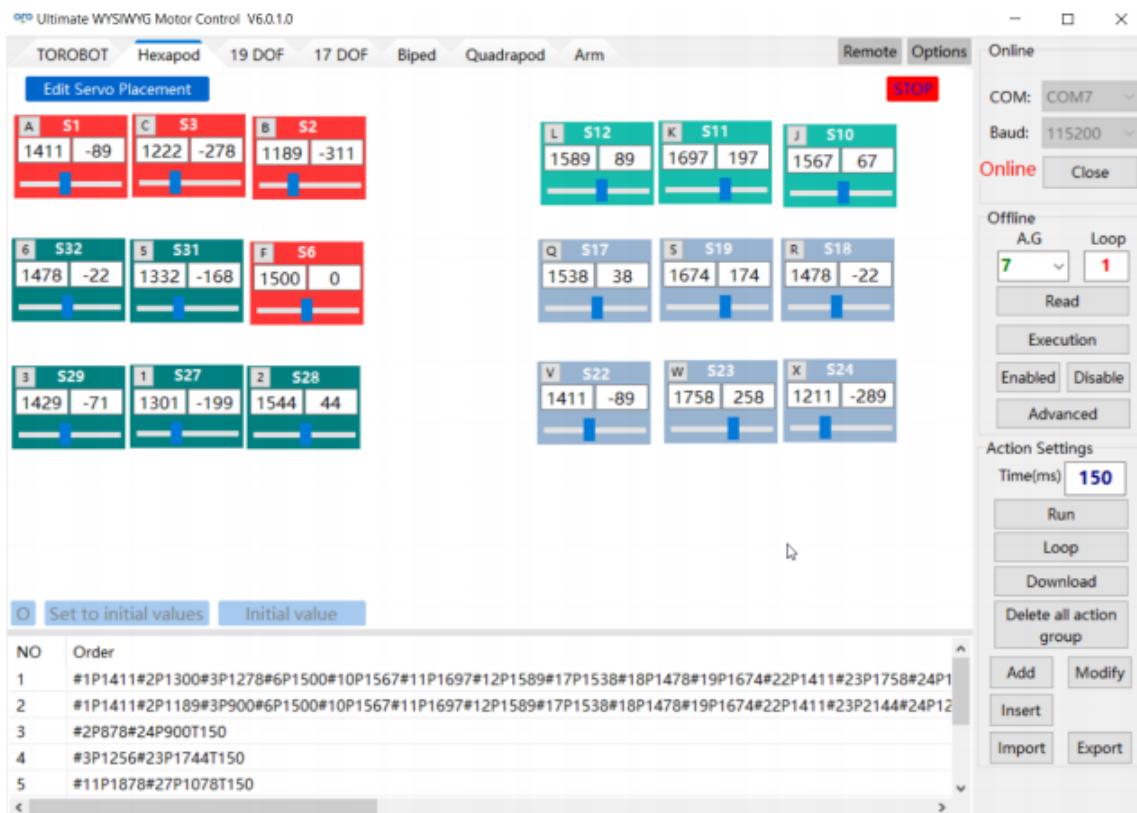
Tabela 6 – Definição dos Grupos de Ação na Memória da Placa Controladora de Servos

Posição de Memória (AG)	Movimento	Duração
4	Ficar em pé	1000 ms
5	Andar para frente	800 ms
6	Virar a direita	1350 ms
7	Virar a esquerda	1350 ms

Fonte: O Autor

Através da definição dos grupos responsáveis pelos movimentos do robô, é necessário então fazer com que o Arduíno faça a devida referência através do protocolo I2C de comunicação serial. O processo de referência dos movimentos é feito através da utilização do *software Ultimate WYSIWYG Motor Control V6.0.1.0* como mostra a Figura 65 .

Figura 65 – Visão do Software de Definição de AG



Fonte: (RUZZON, 2019)

Da figura apresentada, se tem que cada servomotor representado por sua nomenclatura de acordo com a Tabela 4 pode ser configurado através de uma barra ajustável que faz referência à posição relativa do eixo do motor. No caso, a barra pode ser ajustada com valores entre 500 (representando 0°) até 2500 (representando 180°).

Na região inferior da figura se encontram as instruções para a locomoção dos servos,

essas ordens podem ser definidas como sendo AG do sistema. Já a parte direita da figura se encontram as especificações da comunicação entre a placa e o computador, no caso a comunicação é feita via USB com uma taxa de transmissão de 115200 bits/s e além disso se pode definir o tempo em que cada instrução deve ser performada através do campo *Time*.

O formato de instruções da placa controladora de servos pode ser visto na [Figura 66](#).

Figura 66 – Formato de Instruções da Placa Controladora de Servos

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

Fonte: ([RUZZON, 2019](#))

A.2 Localização do Robô

Na segunda etapa, a câmera foi instalada no ambiente e foi implementado o algoritmo responsável por identificar e orientar o robô no ambiente. Além disso, foi definido o processo de comunicação entre o robô e o algoritmo orienta sua movimentação.

A.2.1 Sistema de Visão Externa

Para resolver o problema de localizar o robô em ambiente interno garantindo que o mesmo está seguindo a trajetória correta, primeiramente se faz necessário uma câmera para

captação de imagens. A câmera utilizada foi a C992 PRO fabricada pela *Logitech*, o dispositivo é capaz de capturar imagens com uma alta resolução de 1920x1080 a uma taxa de atualização de 30 quadros por segundo. Assim, devido a sua eficiência na captura de imagens a câmera foi instalada no ambiente conforme mostra a [Figura 67](#).

Figura 67 – Câmera Instalada no Ambiente



Fonte: O Autor

A.2.2 Algoritmo de Processamento de Imagem

O algoritmo de processamento de imagem é o principal responsável para fazer com que a locomoção seja performada com sucesso. Para isso foi desenvolvido um algoritmo utilizando a linguagem de programação *Python* na versão 3.9 e *OpenCV* na versão 4.5.1.48, além disso o código é executado no sistema operacional *Windows 10 64-bit*.

O algoritmo desenvolvido foi uma adaptação do que foi apresentado no projeto de ([RUZZON, 2019](#)) que se baseia numa versão apresentada por ([ARAUJO; MENDONÇA; FREIRE, 2008](#)).

O robô de acordo com a [Figura 64](#) apresenta dois marcadores em sua parte superior de

4 cm cada nas cores azul e vermelha respectivamente. A cor azul representa a parte frontal do robô, já a cor vermelha indica a região traseira do robô.

Sendo uma imagem a representação de pixels, é preciso então obter um parâmetro de *pixels* para que através das imagens fornecidas pela câmera se possa encontrar as cores exatas dos marcadores. Para a identificação dos marcados, de acordo com (ARAUJO; MENDONÇA; FREIRE, 2008), o padrão HSV é a abordagem mais robusta para a segmentação de cores.

Seguindo a parametrização indicada por (RUZZON, 2019) em seu trabalho, foram definidos valores máximos e mínimos para cada componente HSV (matiz, saturação e valor) como mostra a Tabela 7. Os valores foram escolhidos de forma a facilitar o reconhecimento das cores independente da luminosidade do ambiente porém restritos o suficiente para que não haja confusão com outras cores dentro do ambiente.

Tabela 7 – Parametrização de Cores no Padrão HSV para Localização do Robô

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

Fonte: (RUZZON, 2019) (adaptado)

Do algoritmo, alguns passos foram seguidos para o processo de localização do robô.

1. Captura da imagem pela câmera;
2. Utilização do método *GaussianBlur* do *OpenCV* para redução de ruídos e distúrbios na figura;
3. Conversão das cores RGB para HSV através do método *cvtColor* do *OpenCV*;
4. Aplicação de segmentação binária, onde as cores com os valores que estão na faixa da Tabela 7 são convertidos para a cor branca e o restante da imagem é convertida para cor preta;
5. Aplicação do método *dilate* do *OpenCV* para realçar cores brancas;
6. Utilização do método *findContours* do *OpenCV*, o método contorna as regiões de cor branca e retorna uma lista com os parâmetros da posição das formas;
7. O maior contorno da lista é selecionado e então identificado através de uma sinalização frente ao seu redor;
8. O centro do maior contorno é identificado e retorna a sua posição *x,y* da matriz de *pixels* da imagem indicando a posição atual do robô.

A.2.3 OpenCV

A biblioteca de código aberto OpenCV (*Open Source Computer Vision Library*), é uma biblioteca de visão computacional utilizada para extrair e processar dados significativos de imagens. Os dados podem fazer referência a parte de objetos, rastreio de movimentos e mapeamento de objetos 2D e 3D.

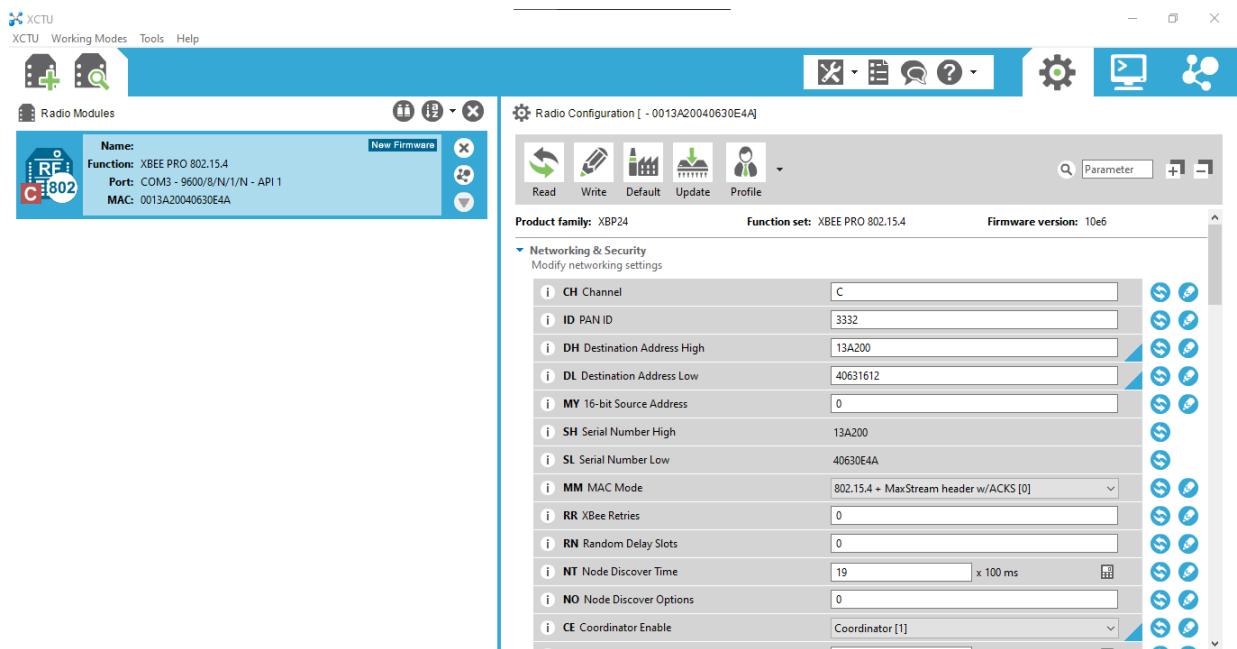
Segundo (BRADSKI, 2000), a biblioteca foi desenvolvida para tornar a visão computacional acessível para programadores e usuários na área de interação humano-computador e robótica, fazendo com que o desenvolvimento de *softwares* seja feito de maneira mais intuitiva. A biblioteca desenvolvida pela Intel é suportada em praticamente todas as linguagens de programação que possui mais de 2500 funções sendo divididas em cinco grupos: Processamento de imagens; Análise Estrutural; Análise de movimento e rastreamento de objetos; Reconhecimento de padrões, calibração de câmera e reconstrução 3D.

A.3 Comunicação Sem Fio

A comunicação sem fio entre a estação e o robô foi feita através da utilização de dois módulos *XBee PRO S1*. Um dos módulos foi definido como *coordinator*, o que coordenada a rede, e o outro como sendo *router* que foi embarcado no robô.

Para que ambos módulos pudessem se comunicar corretamente se fez necessário a configuração individual de cada um através do *software XCTU* como mostra a Figura 68

Figura 68 – *Software XCTU*



Fonte: O Autor

Da figura apresentando o *software* se tem que o módulo está configurado como *coordinator* como mostra a letra C no campo *Radio Modules* no canto esquerdo. O que define a maneira de se comunicar com o módulo *router* é o preenchimento do campo DH (*Destination Address High*) e DL (*Destination Address Low*) ambas informações está fixadas no módulo XBee indicando a primeira e segunda parte do endereço para comunicação. Vale citar que o endereço é o mesmo tanto para o módulo *coordinator* quanto para o módulo *router*.

Para que o algoritmo em *Python* pudesse enviar e receber informações através do computador da estação solo foi utilizada a biblioteca *XBeeDevice* que facilita a comunicação do módulo *coordinator* de maneira ágil. No robô, o módulo *router* é conectado na porta serial 0.

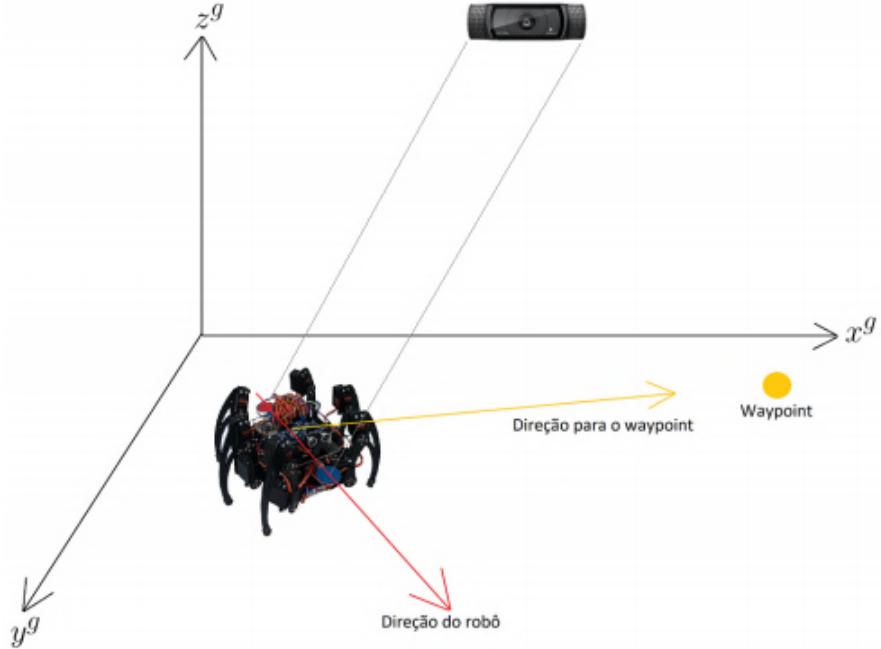
A.4 Sistema de Guiagem do Robô

A câmera responsável pela captura das imagens dando uma percepção do ambiente e auxiliando na localização do robô foi colocada próxima a região central do quarto com sua lente voltada para o chão com o intuito de ter um maior campo de visão na identificação do robô.

Para que o robô pudesse navegar de forma correta entre o ambiente, os estados gerados pelo algoritmo de planejamento de caminho são convertidos em *waypoints*, ou seja, cada estado é transformado em uma coordenada global real x^g, y^g, z^g no ambiente.

Uma vez todos os componentes inseridos no ambiente real, a Figura 69 elucida um diagrama indicando o funcionamento da locomoção do robô. Tendo a câmera instalada no teto do ambiente a mesma ao tirar fotos do local gera uma visão bidimensional do solo com coordenadas x^g, y^g fazendo com que na definição dos pontos o eixo z^g seja desconsiderado.

Figura 69 – Representação Gráfica da Visão Obtida Pelo Algoritmo de Guiagem



Fonte: (RUZZON, 2019)

Para a localização do robô no ambiente, ou seja, encontra as posições x^g e y^g do robô é preciso fazer um cálculo que relaciona as coordenadas dos círculos azul e vermelho como mostra a [Equação A.1](#) e [Equação A.2](#).

$$x_{robo} = \frac{x_{azul} + x_{vermelho}}{2} \quad (\text{A.1})$$

$$y_{robo} = \frac{y_{azul} + y_{vermelho}}{2} \quad (\text{A.2})$$

Além de indicar a localização corrente do robô, as coordenadas dos círculos azul e vermelho auxiliam no cálculo de orientação do robô indicando para qual direção o robô está se dirigindo como mostra a [Equação A.3](#). A orientação do robô está representada pela seta em vermelho na [Figura 69](#), onde segundo (RUZZON, 2019) é utilizado arcotangente_2 para o cálculo da tangente que leva em consideração os quatro quadrantes.

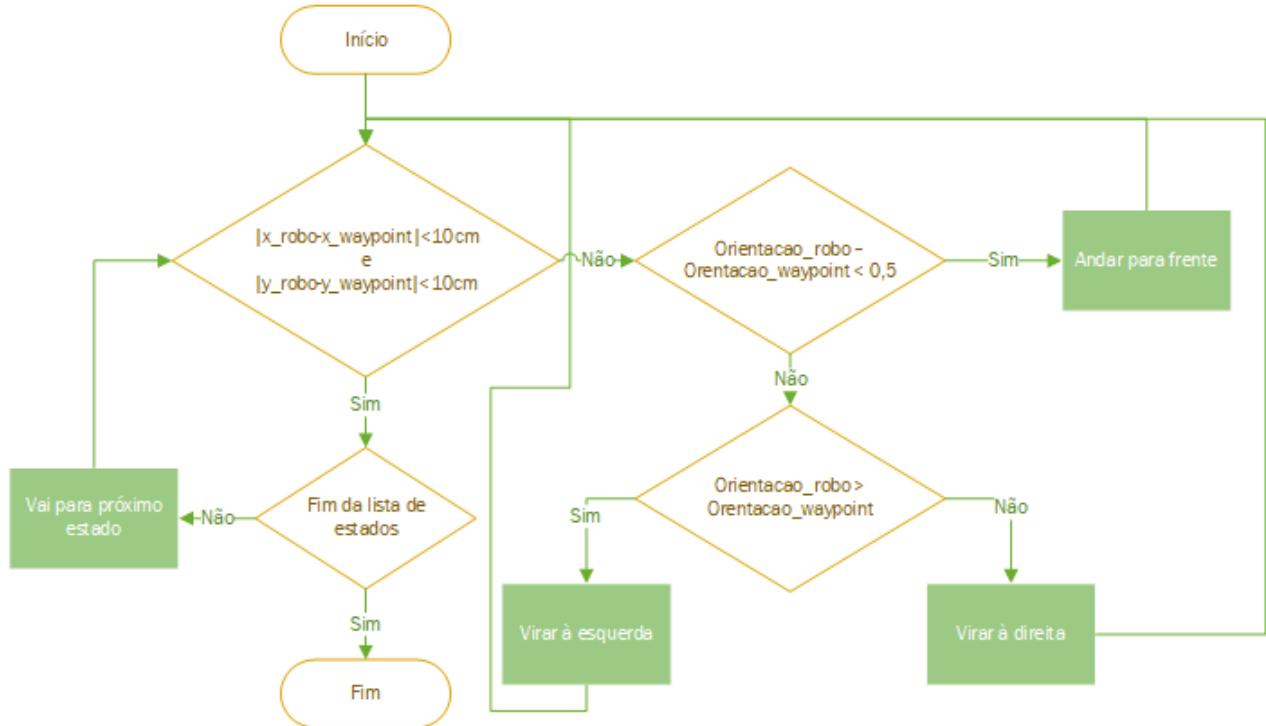
$$\text{Orientacao}_{robo} = \text{atan}_2(y_{azul} - y_{vermelho}, x_{azul} - x_{vermelho}) \quad (\text{A.3})$$

Semelhante ao cálculo anterior a [Equação A.4](#) é utilizada para saber a orientação entre o robô e o próximo estado como mostra a seta amarela na [Figura 69](#).

$$\text{Orientacao}_{waypoint} = \text{atan}_2(y_{waypoint} - y_{robo}, x_{waypoint} - x_{robo}) \quad (\text{A.4})$$

Através da obtenção da imagem, o processamento da mesma e a realização dos cálculos o sistema pode por fim tomar decisões que guiam o robô pelo ambiente. A parte do sistema responsável por tomar decisões é o algoritmo embarcado no *Arduino* que recebe as informações através da comunicação entre os módulos *XBee* processa as mesmas e define qual movimento será performado. Para a realização dos movimentos baseando-se na condição atual do ambiente, a tomada de decisão é feita através do processo mostrado na Figura 70.

Figura 70 – Fluxograma de Locomoção do Robô



Fonte: O Autor

Vale citar que cada vez que o robô realiza uma ação, ele envia uma mensagem para o algoritmo dizendo que terminou aquela ação, em seguida o algoritmo captura outra imagem do ambiente e envia os parâmetros da nova posição ao robô e assim suscetivamente até que o algoritmo indique ao robô que encontrou uma posição de tolerância finalizando o próximo de chegada no estado. Além disso, os valores de tolerância foram definidos arbitrariamente levando em conta o ambiente de avaliação de conceito.

A.5 Modo de Troca de Mensagens

Com as especificações definidas anteriormente foi escolhido um método de comunicação de duas vias entre o robô e a estação solo. Nesse método de comunicação a câmera tira uma foto do ambiente, o algoritmo processa a imagem obtida e envia para o robô os dados de coordenadas e orientação para a locomoção apenas quando é solicitado. A partir disso, a atualização

das posições depende da finalização do movimento do robô que por sua vez realiza uma requisição de localização instantes antes de concluir a movimentação anterior. Com uma frequência de troca de mensagens de aproximadamente 0,75Hz a Tabela 8 apresenta as mensagens definidas para guiar a comunicação entre robô e estação.

Tabela 8 – Definição de Mensagens entre Robô e Estação de Solo

Dispositivo	Mensagem	Função
Robô	Posicao	Solicita o processamento de imagem e recebimento de novas coordenadas.
Robô	Cheguei	Mensagem enviada para estação dizendo que terminou a transição de estado.
Estação de Solo	x_{robo},y_{robo},orientacao_{robo}; x_{estado},y_{estado},orientacao_{estado}	Envia informações sobre a posição e orientação do robô e do estado que se quer alcançar

Fonte: ([RUZZON, 2019](#)) (adaptado)

O código modificados que tem a responsabilidade pela troca de mensagens, o processamento de imagens e auxiliar no mapeamento do ambiente está apresentado no [Apêndice D](#). O algoritmo implementado para ser embarcado no Arduíno está no [Apêndice E](#).

APÊNDICE B – Construção do Algoritmo QLDR

Neste apêndice se encontram os algoritmos responsáveis pelo processo de treinamento de ambiente, onde na primeira parte está apresentada a classe abstrata que define o ambiente gera os métodos que serão utilizados no segundo algoritmo que de fato realiza o treinamento através da atualização da tabela Q.

B.1 Classe de Geração de Ambiente de Treinamento

```

1 import math
2 import numpy as np
3 import random
4
5 class enviroment:
6     def __init__(self):
7         self.x = ENV_SIZE
8         self.y = ENV_SIZE
9         self.cont = 0
10        self.x_position = 0
11        self.y_position = 0
12        self.x_old_state = 0
13        self.y_old_state = 0
14        self.goalx = self.x-1
15        self.goaly = self.y-1
16        self.action_space = [0,1,2,3]
17        self.obstacles_map = []
18        self.states = np.zeros(self.x*self.x).tolist()
19        self.actions_size = len(self.action_space)
20        self.states_size = self.x*self.y
21        self.state_matrix = np.ascontiguousarray(np.arange(self.states_size).reshape(self.y,
22                                         self.x), dtype=int)
22        self.matrix_list = self.state_matrix.tolist()
23
24    def reset_enviroment(self):
25        self.x_position = 0
26        self.y_position = 0
27        self.state = 0
28        self.reward = 0
29        return 0, 0, False
30
31    def update_instructions_list(self,state, state_list):
32        if state not in self.obstacles_map and state!=self.x_position:
33            if self.states[state]==0:
34                self.states[state] = len(state_list)
35            else:
36                if self.states[state]> len(state_list):
37                    self.states[state] = len(state_list)
38
39    def get_env_states(self):

```

```

40     return self.states
41
42     def next_step(self, action, state):
43         begin = False
44         old_x= self.x_position
45         old_y= self.y_position
46         if self.x_position==self.goalx and self.y_position==self.goaly:
47             done=True
48             reward = 5000
49             obstacle = False
50             return state, reward, done, obstacle, begin
51         else:
52             # up
53             if action==0 and self.x_position>0:
54                 self.x_position -= 1
55             # Move left
56             elif action== 1 and self.y_position>0:
57                 self.y_position-= 1
58             # Move down
59             elif action==2 and self.x_position<self.x-1:
60                 self.x_position+=1
61             # Move right
62             elif action==3 and self.y_position<self.y-1:
63                 self.y_position+=1
64             obstacle = False
65             reward=-1
66             done=False
67             next_state = self.state_matrix[self.x_position] [self.y_position]
68             if next_state in self.obstacles_map:
69                 reward = -500
70                 obstacle = True
71                 self.x_position = old_x
72                 self.y_position = old_y
73             if next_state == 0:
74                 reward = -100
75                 begin = True
76             return next_state, reward, done, obstacle, begin
77
78     def get_state_index(self):
79         return self.x_position, self.y_position
80
81     def insert_old_state_index(self, x, y):
82         self.x_old_state = x
83         self.y_old_state = y
84
85     def get_goal_index(self):
86         return self.goaly, self.goaly
87
88     def select_random_action(self):
89         return np.random.choice(self.action_space)

```

B.2 Algoritmo de Treinamento

```

1 from q_learning_env import enviroment
2 from evaluate_training import evaluate_training
3 import numpy as np
4 import matplotlib

```

```

5 import random
6 import time
7 import math
8 import pandas as pd
9
10 alpha = 0.99 # learning rate, learn more quickly if alpha is closer to 1
11 gamma = 0.95 # use a higher gamma for bigger spaces because we value later rewards rather
12     than former rewards
13 epsilon_decay = 0.05
14 min_eps = 0.0
15 NUM_EPISODES = 300
16
17 env = enviroment()
18 Q = np.zeros((env.states_size,env.actions_size)).tolist()
19 steps = []
20 rewards = []
21 penalties = []
22 states = []
23 epsilon_list = []
24 def calculate_dynamic_reward(xold, yold, xnext, ynext, xend, yend, obstacle, begin):
25     if begin == True:
26         lambda_val = 100
27     elif obstacle == True:
28         lambda_val = 500
29     else:
30         lambda_val = 1
31
32     xf = pow((xend-xold),2)
33     yf = pow((yend-yold),2)
34     dt = math.sqrt((xf+yf))
35     xfnext = pow((xend-xnext),2)
36     yfnext = pow((yend-ynext),2)
37     dtf = math.sqrt((xfnext+yfnext))
38     subdtf = dt-dtf
39     if subdtf!=0:
40         rewardend = lambda_val*((subdtf)/abs(subdtf))
41         return rewardend
42     else:
43         return 0
44
45 def select_optimal_action(state):
46     best = Q[state].index(max(Q[state]))
47     return best
48
49 def take_next_step(env, done, state, epsilon):
50     old_x, old_y = env.get_state_index()
51     env.insert_old_state_index(old_x, old_y)
52
53     if np.random.random() < epsilon:
54         action = env.select_random_action()
55     else:
56         action = select_optimal_action(state)
57
58     next_state,reward, done, obstacle, begin = env.next_step(action, state)
59
60     next_x, next_y = env.get_state_index()
61     goal_x, goal_y = env.get_goal_index()
62     if done == False:
63         reward_dynamic = calculate_dynamic_reward(old_x, old_y, next_x, next_y, goal_x, goal_y
64             , obstacle, begin)

```

```

63     reward += reward_dynamic
64
65     next_best_action = np.argmax(Q[next_state])
66     next_max_state_q_value = Q[next_state][next_best_action]
67     Q[state][action] += alpha * (reward + gamma*next_max_state_q_value - Q[state][action])
68
69     if obstacle==True:
70         next_state = state
71
72     return done, next_state, reward
73
74 def main():
75     epsilon = 0.9 # agent can do 100% exploitation (0) or 100% exploration (1)
76     steps, rewards, states = [], [], []
77     training_start = time.time()
78     for i in range(NUM_EPISODES):
79         print('episode', i)
80         state, reward, done = env.reset_enviroment()
81         step, total_reward = 0, 0
82         while not done:
83             done, state, reward = take_next_step(env, done, state, epsilon)
84             step+=1
85             total_reward+=reward
86             states.append(state)
87             epsilon_list.append(epsilon)
88             epsilon -= (epsilon_decay*epsilon) if epsilon>min_eps else 0
89             steps.append(step)
90             rewards.append(total_reward)
91         training_end = time.time()
92         evaluate_training(str(env.states_size),steps, rewards, training_end, training_start,
93                           NUM_EPISODES, Q, epsilon_list)
94
95 if __name__=='__main__':
96     main()

```

B.3 Algoritmo Gerador de Resultados de Treinamento

```

1
2 import matplotlib
3 import plotly.express as px
4 from matplotlib import pyplot as plt
5 import numpy as np
6 import seaborn as sns
7 import pandas as pd
8 import pickle
9 action_space = [0, 1, 2, 3]
10
11
12 def evaluate_training(state, steps, rewards, training_end, training_start, NUM_EPISODES,Q,
13                       epsilon_list):
14
15     Q = np.array(Q)
16     print(Q)
17     with open('pickle/' +str(state)+'.pickle', "wb") as write:
18         pickle.dump(Q, write)
19
20     all_rewards = []

```

```
20     all_rewards.append(rewards)
21     mean_rate = [np.mean(rewards[n-10:n]) if n > 10 else np.mean(rewards[:n])
22                 for n in range(1, len(rewards))]
23     elapsed_training_time = training_end-training_start
24     success_rate = np.mean(rewards)
25     epochs_step_rate = np.mean(steps)
26
27     print("\nEsse ambiente foi resolvido com um total de", str(NUM_EPISODES), "episódios com"
28           ", str(elapsed_training_time),
29           "segundos e uma média de passos por percurso de", str(epochs_step_rate))
30
31     plt.figure()
32     plt.plot(rewards)
33     plt.plot(mean_rate)
34     plt.title('Média de Recompensas')
35     plt.xlabel('Episódio')
36     plt.ylabel('Recompensa')
37     plt.savefig("imgs/mean/mean"+state+".png")
38
39     plt.figure()
40     plt.plot(epsilon_list)
41     plt.title('Decaimento de epsilon')
42     plt.xlabel('Episódio')
43     plt.ylabel('epsilon')
44     plt.savefig("imgs/mean/epsilon"+state+".png")
45
46     fig = plt.figure(figsize=(12,8))
47     ax1 = fig.add_subplot(111)
48     ax1.plot(rewards, '-g', label='reward')
49     ax2 = ax1.twinx()
50     ax2.plot(steps, '+r', label='step')
51     ax1.set_xlabel("episódio")
52     ax1.set_ylabel("recompensa")
53     ax2.set_ylabel("passos")
54     plt.title("Progresso do Treinamento")
55     plt.savefig(
56         'imgs/trainingprocess/trainingprocess'+state+'.png')
57
58     fig = plt.figure()
59     plt.plot(steps)
60     plt.xlabel('Episódio')
61     plt.ylabel('Passos')
62     plt.savefig('imgs/steps/steps'+state+'.png')
63
64     fig = plt.figure()
65     plt.plot(rewards)
66     plt.xlabel('Episódio')
67     plt.ylabel('Recompensa')
68     plt.savefig('imgs/rewards/rewards'+state+'.png')
```

APÊNDICE C – Algoritmo de Análise da Tabela Q e Geração do Caminho

Neste apêndice se encontra o algoritmo responsável por analisar a tabela Q e gerar a figura que desenha o caminho percorrido pelo agente no ambiente.

```

1 import numpy as np
2 import pickle
3 from qlearning_training import initialize_state_matrix, identifiesgoal_state, identifies_state
4 from matplotlib import pyplot as plt # pylint: disable=import-error
5 import plotly.express as px # pylint: disable=import-error
6 import pandas as pd
7 from PIL import Image as im
8 import cv2
9 action_space = np.array([0, 1, 2, 3])
10
11
12 def identifies_index(state):
13     for i in range(enviromentsize):
14         for j in range(enviromentsize):
15             if state_matrix[i][j] == state:
16                 return i, j
17
18
19 def identifies_state_matrix(i, j):
20     return state_matrix[i][j]
21 # verificar se tomndo uma nova decis o n o tomei a mesma anteriormente pra evitar caminhos
22 # longos
23
24 def next_step(action, state, goal_state):
25
26     if state == goal_state:
27         return 10, state
28     else:
29         i, j = identifies_index(state)
30         # up
31         if action == 0 and i > 0:
32             i -= 1
33         # Move left
34         elif action == 1 and j > 0:
35             j -= 1
36         # Move down
37         elif action == 2 and i < enviromentsize - 1:
38             i += 1
39         # Move right
40         elif action == 3 and j < enviromentsize - 1:
41             j += 1
42         steps.append(action)
43         next_state = identifies_state_matrix(i, j)
44         reward = -1
45         return reward, int(next_state)
46

```

```

47
48 def select_optimal_action(state):
49     optimal = np.argmax(Q[state], axis=0)
50     return optimal
51
52
53 def define_steps():
54     for step in steps:
55         if step == 0:
56             steps_desc.append('U')
57         elif step == 1:
58             steps_desc.append('L')
59         elif step == 2:
60             steps_desc.append('D')
61         elif step == 3:
62             steps_desc.append('R')
63
64
65 def select_optimal_path(q_table, enviroment, state_evaluate):
66     global steps, steps_desc
67     i, j = identifies_state(enviroment, enviromentsize)
68     k, l = identifiesgoal_state(enviroment, enviromentsize)
69     state = int(state_matrix[i][j])
70     goal_state = int(state_matrix[k][l])
71     states = []
72     steps = []
73     steps_desc = []
74     states.append(state)
75     reward, next_state = 0, 0
76     done = False
77     while(not done):
78         print(state)
79         action = select_optimal_action(state)
80         reward, next_state = next_step(action, state, goal_state)
81         state = next_state
82         states.append(state)
83         if reward == 10:
84             done = True
85     states = states[:-1]
86     define_steps()
87     print(q_table, '\n', '\n', states, '\n', steps, '\n', steps_desc, '\n', 'path_length:', len(states))
88     plot_q_with_steps(enviroment, states,enviromentsize, state_evaluate)
89     steps_matrix.append(steps_desc)
90
91 def plot_q_with_steps(enviroment, steps, enviromentsize, state):
92     for step in steps:
93         if step!=state:
94             i,j = identifies_index(step)
95             enviroment[i][j] = 5
96     plot_matrix(enviroment,enviromentsize,enviromentsize,state)
97
98
99 def reset_enviroment(enviroment, env_size, goal_position, obstacles_position):
100    enviroment = np.zeros((env_size, env_size))
101    i, j = identifies_state_train(goal_position, env_size)
102    enviroment[i][j] = 20
103    for obstacle in obstacles_position:
104        i,j = identifies_state_train(obstacle, env_size)
105        enviroment[i][j] = -1

```

```
106     enviroment[0][0] = 1
107     return enviroment
108
109
110 def identifies_state_train(goal_position, size):
111     for i in range(size):
112         for j in range(size):
113             if state_matrix[i][j] == goal_position:
114                 return i, j
115
116 def plot_matrix(matrix, x_size, y_size, state):
117     for i in range(x_size):
118         for j in range(y_size):
119             if matrix[i][j] == 0:
120                 matrix[i][j] = 20
121     matrix[x_size-1][y_size-1] = 1
122     cmap = plt.cm.hot
123     plt.imshow(matrix, cmap=cmap)
124     plt.title('Melhor Caminho para Alcançar o Estado ' + str(state) + ' no Ambiente')
125     plt.tight_layout()
126     plt.show()
127     plt.savefig("imgs/with_obstacles/Q/Q" + str(state) + "evaluate.png")
128
129
130 def main():
131     global state_matrix, enviromentsize, Q, steps_matrix
132     steps_matrix = []
133     enviromentsize = SIZE_ENV
134     print('begin')
135     state_matrix = initialize_state_matrix(
136         np.zeros((enviromentsize, enviromentsize)), enviromentsize)
137     obstacles = []
138
139     i = (enviromentsize*enviromentsize)-1
140     state_matrix = initialize_state_matrix(
141         np.zeros((enviromentsize, enviromentsize)), enviromentsize)
142     env = np.zeros((enviromentsize, enviromentsize))
143     env = reset_enviroment(env, enviromentsize, i, obstacles)
144     with open(r'PICKLE_Q_TABLE_PATH', "rb") as read:
145         Q = pickle.load(read)
146         print('evaluating optimal path to position:', i)
147         select_optimal_path(Q, env, i)
148
149         print('steps', steps_matrix)
150
151 if __name__ == '__main__':
152     main()
```

APÊNDICE D – Algoritmo de Processamento de Imagem para Locomoção e Mapeamento do Ambiente

Neste apêndice se encontra o algoritmo responsável pelo processamento da imagem que identifica o robô no ambiente e troca mensagens com o Arduíno embarcado.

```

1 import cv2 # pylint: disable=import-error
2 import math
3 import serial
4 from digi.xbee.devices import XBeeDevice
5 from digi.xbee.devices import *
6 from digi.xbee.util import *
7 from digi.xbee.packets import *
8 import matplotlib.pyplot as plt
9 import time
10 from openpyxl import Workbook
11
12 PORT = "COM3"
13 BUS = 9600
14
15 def AdicionarPlanilha():
16     global ws
17     global wb
18     global robot_center_x
19     global robot_center_y
20     global robot_inclination
21     global target_x
22     global target_y
23     global target_inclination
24     global split_time
25
26
27     linha = str(cont_linhas_planilha)
28     ws['A' + linha] = robot_center_x
29     ws['B' + linha] = robot_center_y
30     ws['C' + linha] = robot_inclination
31     ws['D' + linha] = target_x
32     ws['E' + linha] = target_y
33     ws['F' + linha] = target_inclination
34     ws['G' + linha] = split_time
35     cont_linhas_planilha += 1
36
37 def CriarPlanilha():
38     global ws
39     global wb
40     wb = Workbook()
41     ws = wb.active
42     ws.title = 'Execucao Robo'
43     ws = wb['Execucao Robo']
44     ws['A1'] = 'Robo X'
```

```
45     ws['B1'] = 'Robo Y'
46     ws['C1'] = 'Inclinacao Robo'
47     ws['D1'] = 'Waypoint X'
48     ws['E1'] = 'Waypoint Y'
49     ws['F1'] = 'Inclinacao Waypoint'
50     ws['G1'] = 'Tempo'
51
52 def VerificaPosicao():
53     global x_red
54     global y_red
55     global x_blue
56     global y_blue
57     global robot_center_x
58     global robot_center_y
59     global robot_inclination
60     global target_x
61     global target_y
62     global target_inclination
63     global List_X
64     global List_Y
65     global start_time
66     global split_time
67     ret_val, img = cam.read()
68     img_filter = cv2.GaussianBlur(img.copy(), (3, 3), 0)
69     img_filter = cv2.cvtColor(img_filter, cv2.COLOR_BGR2HSV)
70     img_binary_red = cv2.inRange(img_filter.copy(), THRESHOLD_LOW_RED,
71                                   THRESHOLD_HIGH_RED)
72     img_binary_blue = cv2.inRange(img_filter.copy(), THRESHOLD_LOW_BLUE,
73                                   THRESHOLD_HIGH_BLUE)
74     img_binary_red = cv2.dilate(img_binary_red, None, iterations = 1)
75     img_binary_blue = cv2.dilate(img_binary_blue, None, iterations = 1)
76
77     #Encontrar círculo vermelho
78     img_contours = img_binary_red.copy()
79     contours = cv2.findContours(img_contours, cv2.RETR_EXTERNAL, \
80                                cv2.CHAIN_APPROX_SIMPLE)[-2]
81     center = None
82     radius = 0
83     if len(contours) > 0:
84         c = max(contours, key=cv2.contourArea)
85         ((x_red, y_red), radius) = cv2.minEnclosingCircle(c)
86         M = cv2.moments(c)
87         if M["m00"] > 0:
88             center = (int(M["m10"] / M["m00"]),
89                        int(M["m01"] / M["m00"]))
90             if radius < MIN_RADIUS:
91                 center = None
92     if center != None:
93         cv2.circle(img, center, int(round(radius)), (0, 255, 0))
94
95     #Encontrar círculo azul
96     img_contours = img_binary_blue.copy()
97     contours = cv2.findContours(img_contours, cv2.RETR_EXTERNAL, \
98                                cv2.CHAIN_APPROX_SIMPLE)[-2]
99     center = None
100    radius = 0
101    if len(contours) > 0:
102        c = max(contours, key=cv2.contourArea)
103        ((x_blue, y_blue), radius) = cv2.minEnclosingCircle(c)
104        M = cv2.moments(c)
```

```
105     if M["m00"] > 0:
106         center = (int(M["m10"] / M["m00"]),
107                     int(M["m01"] / M["m00"]))
108         if radius < MIN_RADIUS:
109             center = None
110     if center != None:
111         cv2.circle(img, center, int(round(radius)), (0, 255, 0))
112
113     #Exibindo resultados
114     x_red = x_red * dist_x / CAMERA_WIDTH
115     y_red = dist_y - (y_red * dist_y / CAMERA_HEIGHT)
116     x_blue = x_blue * dist_x / CAMERA_WIDTH
117     y_blue = dist_y - (y_blue * dist_y / CAMERA_HEIGHT)
118     robot_center_x = int( (x_blue + x_red) / 2 )
119     robot_center_y = int( (y_blue + y_red) / 2 )
120     List_X.append(robot_center_x)
121     List_Y.append(robot_center_y)
122     robot_inclination = round(math.atan2(x_blue - x_red, y_blue - y_red), 2)
123     target_inclination = round(math.atan2(target_x - robot_center_x, target_y - robot_center_y
124                                         ), 2)
125
126     #Atualizando imagem da camera
127     cv2.imwrite('webcam.jpg', img)
128     message = str(robot_center_x) + "," + str(robot_center_y) + ";" + str(robot_inclination) +
129               ";" + str(target_x) + "," + str(target_y) + ";" + str(target_inclination)
130     split_time = time.time() - start_time
131     AdicionarPlanilha()
132     print ("Enviado para o Robô: " + message)
133     return message
134
135 def main():
136     global target_x
137     global target_y
138     global List_Target_X
139     global List_Target_Y
140     global List_X
141     global List_Y
142     global start_time
143     global split_time
144     target_x = 0
145     target_y = 0
146     i = 0
147     target = targets[i]
148     target_x = int(target.split(',') [0])
149     target_y = int(target.split(',') [1])
150     List_Target_X.append(target_x)
151     List_Target_Y.append(target_y)
152     start_time = time.time()
153
154     device = XBeeDevice("COM3", 9600)
155     device.open()
156     remote_device = RemoteXBeeDevice(device,XBee64BitAddress.from_hex_string("0013A20040631612
157                                         "))
158     device.send_data(remote_device, "0")
159     xbee_message = device.read_data()
160     while xbee_message is None:
161         xbee_message = device.read_data()
162     xbee_message = xbee_message.data.decode()
163     ok = 1
164     print(xbee_message)
```

```
162
163     while True:
164         print('Alvo: ' + str(target))
165         if target is not None:
166             if ok == 1:
167                 device.send_data(remote_device,VerificaPosicao())
168                 ok = 0
169             elif ok==0:
170                 robot_info = device.read_data()
171                 while robot_info is None:
172                     robot_info = device.read_data()
173                 robot_info = robot_info.data.decode()
174                 print ('Informacao Recebida do Robo: ' + str(robot_info))
175                 if 'Posicao' in robot_info:
176                     device.send_data_broadcast(VerificaPosicao())
177                 elif 'Cheguei' in robot_info:
178                     #ws['H' + str(cont_linhas_planilha - 1)] = 'Cheguei'
179                     if i < len(targets) - 1:
180                         i += 1
181                         target = targets[i]
182                         target_x = int(target.split(',') [0])
183                         target_y = int(target.split(',') [1])
184                         List_Target_X.append(target_x)
185                         List_Target_Y.append(target_y)
186                         device.send_data(remote_device,VerificaPosicao())
187                     else:
188                         break
189                     #wb.save('relat rio.xlsx')
190                     print ("\n\n")
191             device.close()
192             print ("\n\n")
193             print ("O robo chegou a todos os alvos. Aplicacao encerrada.")
194             plt.plot(List_X, List_Y, 'r--',
195                     List_Target_X, List_Target_Y, 'bo',
196                     List_X[0], List_Y[0], 'go')
197             plt.axis([0, dist_x, 0, dist_y])
198             plt.show()
199
200
201 if __name__ == '__main__':
202     global cont_linhas_planilha
203     #Parametros
204     # Azul
205     THRESHOLD_LOW_BLUE = (80, 140, 87)
206     THRESHOLD_HIGH_BLUE = (120, 240, 130)
207     # Vermelho
208     THRESHOLD_LOW_RED = (0, 130, 120)
209     THRESHOLD_HIGH_RED = (15, 240, 180)
210     # Distancia vis veis nos eixos
211     dist_x = 277
212     dist_y = 209
213     # Resolucao desejada
214     CAMERA_WIDTH = 1024
215     CAMERA_HEIGHT = 768
216     # Raio minimo para o circulo de contorno
217     MIN_RADIUS = 2
218
219     # Inicializacao de variaveis globais
220     x_red = 0
221     y_red = 0
```

```
222     x_blue = 0
223     y_blue = 0
224     robot_center_x = 0
225     robot_center_y = 0
226     robot_inclination = 0
227     target_inclination = 0
228     start_time = 0
229     split_time = 0
230     cont_linhas_planilha = 2
231     ws = None
232     wb = None
233     List_X = []
234     List_Y = []
235     List_Target_X = []
236     List_Target_Y = []
237     #Lista de Waypoints
238     targets = []
239     # Inicializacao da camera
240     cam = cv2.VideoCapture(1)
241     # Define a resolucao escolhida para a imagem
242     cam.set(cv2.CAP_PROP_FRAME_WIDTH, CAMERA_WIDTH)
243     cam.set(cv2.CAP_PROP_FRAME_HEIGHT, CAMERA_HEIGHT)
244     print ("Camera Inicializada")
245     print ("Monitoramento do Robo Ativo")
246     CriarPlanilha()
247     main()
```

APÊNDICE E – Algoritmo Responsável pela Locomoção do Robô

Neste apêndice se encontra o algoritmo que é responsável por analisar qual é a posição atual do robô e qual ele precisa alcançar e envia os comandos para os servomotores para realizar a locomoção no ambiente.

```

1 #define pino_trigger 22
2 #define pino_echo 24
3 #include "Servo.h"
4 Servo meuservo;
5 String valores;
6 String coordenadas_robo;
7 String coordenadas_alvo;
8 int x_robo;
9 int y_robo;
10 float inclinacao_robo;
11 int x_alvo;
12 int y_alvo;
13 float inclinacao_alvo;
14
15 void De_Pe()
16 {
17     Serial1.print("#4GC1\r\n");
18     delay(2000);
19 }
20 void Andar_Frente()
21 {
22     Serial1.print("#5GC2\r\n");
23 }
24 void Virar_Esquerda()
25 {
26     Serial1.print("#7GC1\r\n");
27 }
28 void Virar_Direita()
29 {
30     Serial1.print("#6GC1\r\n");
31 }
32
33 String getValue(String data, char separator, int index)
34 {
35     int found = 0;
36     int strIndex[] = {0, -1};
37     int maxIndex = data.length()-1;
38
39     for(int i=0; i<=maxIndex && found<=index; i++)
40     {
41         if(data.charAt(i)==separator || i==maxIndex)
42         {
43             found++;
44             strIndex[0] = strIndex[1]+1;
45             strIndex[1] = (i == maxIndex) ? i+1 : i;
46         }
47     }
48 }
```

```
47     }
48     return found>index ? data.substring(strIndex[0], strIndex[1]) : "";
49 }
50 void Distribui_Valores()
51 {
52     coordenadas_robo = getValue(valores,';',0);
53     inclinacao_robo = getValue(valores,';',1).toFloat();
54     coordenadas_alvo = getValue(valores,';',2);
55     inclinacao_alvo = getValue(valores,';',3).toFloat();
56     x_robo = getValue(coordenadas_robo,';',0).toInt();
57     y_robo = getValue(coordenadas_robo,';',1).toInt();
58     x_alvo = getValue(coordenadas_alvo,';',0).toInt();
59     y_alvo = getValue(coordenadas_alvo,';',1).toInt();
60 }
61 void setup()
62 {
63     Serial.begin(9600);
64     Serial1.begin(9600);
65     meuservo.attach(9); // Declara o pino do servo
66 }
67 void loop()
68 {
69     meuservo.write(90); // Comando para angulo especifico
70     De_Pe();
71     Serial.print("Posicao");
72     while (true)
73     {
74         meuservo.write(90); // Comando para angulo especifico
75         while(!Serial.available()){}
76         valores = Serial.readString();
77         Distribui_Valores();
78         if (abs(x_robo - x_alvo) < 10 && abs(y_robo - y_alvo) < 10)
79         {
80             De_Pe();
81             Serial.print("Cheguei");
82             delay(2000);
83         }
84         else
85         {
86             if (abs(inclinacao_alvo - inclinacao_robo) > 0.5)
87             {
88                 if (inclinacao_alvo < inclinacao_robo)
89                 {
90                     Virar_Esquerda();
91                     Serial.print("Virei Esquerda");
92                 }
93                 else
94                 {
95                     Virar_Direita();
96                     Serial.print("Virei Direita");
97                 }
98             }
99         else
100         {
101             Andar_Frente();
102             Serial.print("Frente");
103         }
104         Serial.print("Posicao");
105     }
106 }
```