UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

ALEXSANDRO REICHERT WELTER

APRENDIZADO POR REFORÇO PROFUNDO PARA NAVEGAÇÃO DE UM VEÍCULO GUIADO AUTOMATICAMENTE

PATO BRANCO

2022

ALEXSANDRO REICHERT WELTER

APRENDIZADO POR REFORÇO PROFUNDO PARA NAVEGAÇÃO DE UM VEÍCULO GUIADO AUTOMATICAMENTE

Deep Reinforcement Learning Applied to a Automated Guided Vehicle

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia de Computação do Curso de Bacharelado em Engenharia de Computação da Universidade Tecnológica Federal do Paraná.

Orientador: Profa. Dra. Luciene de Oliveira Marin

Coorientador: Prof. Dr. Jeferson José de Lima

PATO BRANCO

2022

ALEXSANDRO REICHERT WELTER

APRENDIZADO POR REFORÇO PROFUNDO PARA NAVEGAÇÃO DE UM VEÍCULO GUIADO AUTOMATICAMENTE

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia de Computação do Curso de Bacharelado em Engenharia de Computação da Universidade Tecnológica Federal do Paraná.

Data de aprovação: 09/dezembro/2022

Luciene de Oliveira Marin
Doutorado em Engenharia Elétrica
Universidade Tecnológica Federal do Paraná (UTFPR) - Pato Branco

---

Jeferson José de Lima
Doutorado em Engenharia Elétrica
Universidade Tecnológica Federal do Paraná (UTFPR) - Ponta Grossa

---

Kathya Silvia Collazos Linares
Doutorado em Engenharia Elétrica
Universidade Tecnológica Federal do Paraná (UTFPR) - Pato Branco

---

Jorge Luis Roel Ortiz
Doutorado em Engenharia Elétrica
Universidade Tecnológica Federal do Paraná (UTFPR) - Pato Branco

PATO BRANCO

2022

O importante é a lembrança dos erros, que nos permite não cometer sempre os mesmos. O verdadeiro tesouro do homem é o tesouro dos seus erros, a larga experiência vital decantada por milênios, gota a gota. (Gasset, José Ortega).

RESUMO

Os veículos guiados automaticamente (AGVs) estão relacionados ao problema da navegação autônoma na logística interna, sendo essa uma área muito pertinente na indústria atual. Com a utilização de robôs móveis, que não precisam da atuação humana direta para o seu funcionamento, é possível reduzir custos, aumentar a eficiência e facilitar a manutenção de ambientes de logística interna. Dentre as várias soluções, existe o uso do aprendizado por reforço aplicado ao controle do robô. Essa técnica de aprendizado tem a característica de ter uma troca de informações entre o ambiente e o agente, com o agente buscando objetivos e o ambiente retornando recompensas para cada uma de suas ações. Tais características permitem que o aprendizado ocorra sem o conhecimento prévio do ambiente, sendo esse um grande diferencial para a área da robótica, pela sua adaptabilidade a ambientes dinâmicos e diversos. O presente trabalho tem o objetivo de avaliar o algoritmo *Deep Q-learning* (DQN) para o uso na navegação autônoma de um robô móvel em ambiente de simulação e realizar a análise dos resultados obtidos.

Palavras-chave: aprendizado por reforço. aprendizado por reforço profundo. veiculo guiado automaticamente. robô movel. robotica.

ABSTRACT

Automated Guided Vehicles (AGVs) are related to the problem of autonomous navigation in internal logistics, which is a very relevant area in the current industry. With the use of mobile robots, which are controlled without direct human interaction, it is possible to reduce costs, increase efficiency and facilitate the maintenance of internal logistics environments. Among the various solutions, there is the use of reinforcement learning applied to robot control. This learning technique has the characteristic of having an exchange of information between environment and agent, with the agent seeking goals and the environment returning rewards for each of its actions. Such characteristics allow learning to be conducted without prior knowledge of the environment, which is useful in the robotics field, due to its adaptability to dynamic and diverse environments. The present work aims to evaluate the *Deep Q-learning* (DQN) algorithm for use in autonomous navigation of a mobile robot in a simulation environment and to analyze the results obtained.

Keywords: reinforcement learning. deep q-learning. automated guided vehicle. mobile robot. robótica.

LISTA DE ALGORITMOS

LISTA DE FIGURAS

LISTA DE FOTOGRAFIAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

SIGLAS

AGV	*Automated Guided Vehicle*
AR	Aprendizado por Reforço
CIR	Centro Instantâneo de Rotação
CNN	*Convolutional Neural Network*
DQN	*Deep Q-learning* ou *Deep Q-network*
DT	Diferença-temporal
MC	Monte Carlo
MDP	*Markov Decision Process*
PD	Programação Dinâmica
ROS	*Robot Operating System*
SLAM	*Simultaneous Localization and Mapping*

SUMÁRIO

1 INTRODUÇÃO

O uso de robôs na indústria não é nada novo, tendo início na década de 50 com o braço robótico *unimate* (WALLÉN, 2008). Com o passar dos anos, existiram diversas evoluções nas funcionalidades e aplicações, sendo que atualmente quase toda atividade industrial usa sistemas robóticos ou autônomos no processo de manufatura, buscando aprimorar a qualidade e quantidade dos seus produtos.

Na atualidade tem-se a crescente demanda por robôs móveis, como em soluções para logística, resgaste, limpeza, vigilância e até mesmo exploração espacial. Em grandes armazéns já são utilizados sistemas de veículos guiados automaticamente (do inglês, *Automated Guided Vehicle* (AGV)) fazendo o transporte de toda categoria de carga na parte de logística interna (SCHULZE; WULLNER, 2006), facilitando o gerenciamento do estoque e transporte de cargas pesadas. O principais meios de navegação autônoma utilizados são seguidores de linha ou aplicações com percursos bem definidos.

O método tradicional de funcionamento de armazéns consiste em pessoas buscando itens e fazendo todo o processo de forma manual, já com a utilização de AGVs, como o sistema de robôs KIVA da amazon (GUIZZO, 2008), todo o trabalho manual é feito por robôs, reduzindo os custos e aumentando a eficiência do processo. Segundo estudos, cerca de 55% dos gastos em armazéns é proveniente do processo de busca de itens (KOSTER; LE-DUC; ROODBERGEN, 2007) e com a processo mais eficiente e sem erros humanos esses gastos podem ser reduzidos drasticamente ao longo prazo.

Fotografia 1 – Robôs do sistema KIVA trabalhando em um armazém da Staples em Denver/CO
Fonte: Fotografia de Joel Eden (GUIZZO, 2008).

Quando se trata do assunto de navegação autônoma em ambientes externos e dinâmicos, têm-se alguns exemplos, como os robôs autônomos da *Starship Technologies*, que realizam entregas percorrendo curtas distâncias por calçadas e ruas (CHEN *et al.*, 2021). Ao utilizarem sistemas de GPS, câmeras, imagens por satélite, entre outros sensores, os veículos autônomos

conseguem lidar com quase todos os obstáculos que aparecem no seu caminho. Entretanto, ainda há muito para ser aprimorado nos métodos de navegação mais utilizados.

A abordagem mais comum na navegação autônoma é o método de Localização e Mapeamento Simultâneos (do inglês, *Simultaneous Localization and Mapping* (SLAM)), que lida com esse problema utilizando mapas obtidos por um escaneamento a laser do ambiente de navegação (DURRANT-WHYTE; BAILEY, 2006) e um algoritmo de planejamento de rota. Soluções que utilizam o SLAM dependem de um mapa de pontos detalhado do ambiente, esses geralmente obtidos pelo controle humano do robô pelo ambiente, tornando o processo demorado e de pouca adaptabilidade para novos ambientes.

Outro meio para solucionar o problema da navegação autônoma é a utilização de aprendizagem por reforço, como Zhang *et al.* (2017) que proveram a solução utilizando o algoritmo *Deep Q-learning* (*Deep Q-learning* ou *Deep Q-network* (DQN)), método que faz o uso do algoritmo *Q-learning* dentro de uma rede neural convolucional. Diversos estudos fazem o uso desta técnica para a navegação de robôs móveis, Yang, Juntao e Lingling (2020) a utiliza para o planejamento de rotas de múltiplos robôs, um uso simples de DQN que pode ser visto na implementação de Zhou *et al.* (2018) e no trabalho de Xue *et al.* (2019) que utilizam uma variação chamada *double* DQN. A utilização do DQN obteve resultados promissores em ambientes diversos, utilizando a simulação como base para o treinamento dos robôs e após isso teve uma transição tranquila para a navegação no mundo real.

Tendo em vista as informações citadas acima, a proposta do presente trabalho é a implementação de um sistema de navegação autônoma para robôs móveis com movimentação baseada em rodas. A técnica utilizada para o controle das decisões é a do aprendizado por reforço, mais especificamente o algoritmo DQN. A solução proposta tem foco na utilização do autônomo para tarefas logísticas internas, visando a otimização para um sistema de fácil adaptação em ambientes diversos.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

Implementar e avaliar uma técnica de aprendizado de máquina por reforço para o problema de navegação autônoma de um veículo guiado automaticamente, em um ambiente interno simulado, aplicado a tarefas de logística interna.

1.1.2 Objetivos Específicos

- Realizar a integração dos sensores, robô e ambiente utilizando ROS (do inglês, *Robot Operating System*);

- Construir e utilizar um cenário representando o ambiente interno de um armazém construído com a ferramenta Gazebo;
- Aplicar os algoritmos de Deep Q-learning ao problema de navegação autônoma de um robô móvel do tipo veículo guiado automaticamente;
- Avaliar o desempenho da técnica de navegação do robô em ambiente simulado;
- Quantificar os resultados obtidos com relação a sua eficiência;

1.2 JUSTIFICATIVA

Considerando o crescimento da área de logística inteligente, junto a indústria 4.0, há a necessidade de meios de transporte de materiais e itens com maior eficiência, esse trabalho está, de forma gradual, migrando para sistemas que usam AGVs. Um dos mais básicos recursos de um AGV é a sua navegação autônoma, esse é um dos problemas principais na área da robótica móvel, ela consiste em movimentar um robô da sua posição atual até uma posição desejada usando apenas os seus sensores (ZHANG *et al.*, 2017).

A forma mais tradicional de navegação para um AGV é a utilização de trajetos bem definidos a serem seguidos, através de fitas coloridas ou magnéticas, fazendo o uso de sensores ou formas físicas para evitar colisões (FEDORKO; HONUS; SALAI, 2017). Essa técnica de navegação reduz de forma considerável os gastos do transporte manual de materiais, porém ainda há pouca flexibilidade e adaptabilidade para novos ambientes ou situações que podem ocorrer.

Uma forma de maior adaptabilidade para a navegação e planejamento de rotas é a utilização de técnicas baseadas em Localização e Mapeamento Simultâneos (SLAM) (DURRANT-WHYTE; BAILEY, 2006), técnica amplamente utilizada na robótica e com capacidade para solucionar problemas de navegação para sistemas de AGVs. Com esses métodos é necessário o conhecimento prévio de um ambiente para que a localização e a navegação funcionem corretamente, para isso é feito um mapeamento a *laser* das instalações. Quando pensamos em ambientes externos ou com objetos e rotas que alteram as suas posições, essas técnicas perdem boa parte da sua eficiência e funcionalidade.

Alguns métodos de planejamento de rotas para a navegação autônoma que tem relevância são, por exemplo, o algoritmo A* (SONG; LIU; BUCKNALL, 2019), o *artificial potential field* (KHATIB, 1985) e o algoritmo de colônia de formigas (WANG *et al.*, 2018). Esses algoritmos não conseguem processar informações ambientais de altas dimensões (como imagens) ou facilmente caem em um ótimo local em ambientes complexos (GUO *et al.*, 2020). Nesses pontos é que algoritmos baseados em aprendizado profundo por reforço (do inglês, DRL) apresentam resultados excelentes, unindo a capacidade de decisões do aprendizado por reforço e grande capacidade de percepção do aprendizado profundo.

No aprendizado por reforço, um algoritmo clássico é o chamado *Q-learning* (WATKINS, 1989), que foca em obter a maior recompensa acumulada possível para o agente de aprendizado. Em estudos recentes, foi utilizado por Mnih *et al.* (2015) técnicas de aprendizagem profunda junto ao algoritmo *Q-learning*, essa técnica foi denominada *Deep Q-learning*. A DQN faz o uso de uma Rede neural Convolucional (do inglês, *Convolutional Neural Network* (CNN)) profunda para aproximar o valor ótimo da função Q, se provando dessa forma uma arquitetura que pode aprender politicas de controle em diferentes ambientes com apenas um mínimo conhecimento prévio das recompensas e das ações possíveis nesse ambiente.

Com o crescimento do mercado de robótica móvel, a necessidade de soluções flexíveis e adaptáveis, e os trabalhos desenvolvidos na área de aprendizado profundo por reforço, este trabalho propõe-se a investigar, se a implementação de técnicas de aprendizado por reforço podem ser aplicadas na navegação autônoma de AGVs industriais.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, é apresentada a modelagem do ambiente e do veículo, os conceitos da aprendizagem por reforço e por fim os métodos e técnicas a serem utilizadas para o treinamento do robô.

2.1 MODELAGEM DO AMBIENTE E DO VEÍCULO

Os primeiros conceitos a serem apresentados, são relacionados a descrição das coordenadas do sistema robótico.

2.1.1 Cinemática Robô Diferencial

Este segmento trata o modelo de Robô Diferencial. Esse modelo serve como base para o robô a ser utilizado como veículo guiado automaticamente (AGV), sendo aplicável a diversos outros robôs que seguem um esquema de rodas similar.

O robô diferencial usa um mecanismo de direção simples, muito usado na prática por robôs de pequeno porte. Suas rodas principais, são duas fixas sobre um mesmo eixo e ficam nas laterais esquerda e direita do chassi do robô. A velocidade de cada roda é controlada por um motor separado. Geralmente tem uma ou mais pequenas rodas no formato de castor para auxiliar na sustentação do chassi (KLANCAR *et al.*, 2017), porém não apresentam restrições consideráveis ao sistema.

Para a representação do robô diferencial, são utilizados dois sistemas de referência. O sistema $\{R\}$ diz respeito ao robô e $\{M\}$ é a referência global, mapa em que o robô $\{R\}$ se locomove. Os sistemas estão representados na Figura 1.

As variáveis de entrada são a velocidade da roda direita v_D e a velocidade da roda esqueda v_E. A velocidade v_R é a velocidade do robô na totalidade. A sigla CIR vem de centro instantâneo de rotação (Centro Instantâneo de Rotação (CIR)) ou centro instantâneo de curvatura, que define um ponto onde todas as rodas seguem um movimento circular com a mesma velocidade angular ω (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011).

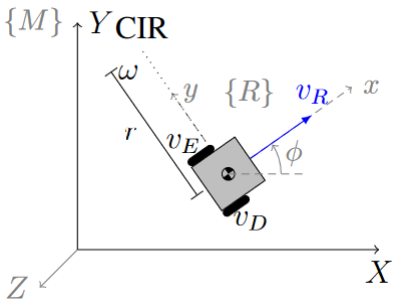


Figura 1 – Robô Diferencial.
Fonte: Adaptado de (KLANCAR *et al.*, 2017)

Tendo $r(t)$ como o raio em relação ao centro de massa do chassi e L a distância entre as rodas do robô, é possível chegar as equações da cinemática do robô diferencial. A velocidade angular para a roda esquerda é dado pela Equação 1 e para a roda direita pela Equação 2:

$$\omega(t) = \frac{v_E(t)}{r(t) - \frac{L}{2}} \tag{1}$$

$$\omega(t) = \frac{v_D(t)}{r(t) + \frac{L}{2}} \tag{2}$$

Dessa forma, têm-se as seguintes expressões:

$$\omega(t) = \frac{v_D(t) - v_E(t)}{L}, \qquad r(t) = \frac{Lv_D(t) + v_E(t)}{2v_D(t) - v_E(t)} \tag{3}$$

A partir disso é definida a velocidade resultante v_R, por

$$v_R(t) = \omega(t)r(t) = \frac{v_D(t) + v_E(t)}{2}. \tag{4}$$

Considerando as relações acima, é possível apresentar o modelo cinemático do robô diferencial $\{R\}$ relacionado ao sistema de global de $\{M\}$, demonstrado pela Equação 5.

$$\begin{bmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{\phi}(t) \end{bmatrix} = \begin{bmatrix} \cos(\phi(t)) & 0 \\ \sin(\phi(t)) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_R(t) \\ \omega(t) \end{bmatrix} \tag{5}$$

Também, de forma similar, são definidas as equações da cinemática interna do robô diferencial, dadas pelo sistema referência $\{R\}$, em que r_r representa o raio das rodas, representadas pela Equação 6.

$$\begin{bmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{\phi}(t) \end{bmatrix} = \begin{bmatrix} \frac{r_r}{2} & \frac{r_r}{2} \\ 0 & 0 \\ -\frac{r_r}{L} & \frac{r_r}{L} \end{bmatrix} \begin{bmatrix} \omega_E(t) \\ \omega_D(t) \end{bmatrix}. \tag{6}$$

A cinemática e dinâmica do robô tem algumas restrições de movimentação intrínsecas do seu formato. No caso apresentado pela Figura 1, o robô tem uma restrição com relação a $\dot{y}_R$, em que, em condições normais, sempre terá velocidade igual a zero, como visível na Equação 6.

2.1.2 Dinâmica e Restrições de Movimento

Um robô com rodas tem uma série de restrições dinâmicas, em que a resposta do sistema está limitada pela inércia e restrições dos atuadores, ou também restrições cinemáticas, com origem na própria construção do robô (KLANCAR *et al.*, 2017).

2.1.2.1 Restrições Holonômicos

Em um sistema holonômico, as suas restrições são dependentes das coordenadas generalizadas, para um sistema com n coordenadas $\mathbf{q} = [q_1, ..., q_n]$, sendo

$$f(\mathbf{q}) = f(q_1, \cdots, q_n) = 0, \tag{7}$$

a equação de restrições de vínculos. A função $f(\mathbf{q})$ e suas derivadas são funções contínuas.

De forma geral é possível ter m restrições holonômicas $(m < n)$. Outro fator a ser observado na Equação 7, é que ela não depende da velocidade ou qualquer derivada com relação a t.

Com isso podem ser definidas as equações de energia que regem o sistema do robô. Na Equação 8, é apresentada a formulação de Lagrange, em que $\mathcal{T}$ são as energias cinéticas e $\mathcal{V}$ são as energias potenciais.

$$\mathcal{L} = \mathcal{T} - \mathcal{V}. \tag{8}$$

Para a equação das energias cinéticas tem-se

$$\mathcal{T} = \frac{1}{2}\sum_{k=1}^{N} \dot{\mathbf{q}}_k{}^T \mathbf{M}_k \dot{\mathbf{q}}_k + \frac{1}{2}\sum_{k=1}^{N} \omega_k^T \mathbf{J}_k \omega_k, \tag{9}$$

em que $\mathbf{M}_k$ representa as massas relacionadas aos elementos k do sistema de equação de movimento, $\mathbf{J}_k$ as forças e inércia dos sistemas rotativos e ω_k as velocidades angulares.

A equação $\mathcal{V}$ de energia potencial difere para cada categoria de energia potencial ao qual o sistema robótico está situado. Vamos definir um robô que o sistema esteja sob o efeito da gravidade, então a energia potencial gravitacional é aplicável, sendo definida pela Equação 10

$$\mathcal{V}_g = \sum_{k=1}^{N} \mathbf{M} g \underbrace{\Delta y_k}_{\text{altura}}. \tag{10}$$

Desta forma, a Formulação de Lagrange pode ser descrita por

$$\frac{d}{\mathrm{d}t}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}_k}\right) - \frac{\partial \mathcal{L}}{\partial q_k} = f_k, \quad k = 1,2,...,n, \tag{11}$$

em que k é o índice das coordenadas generalizadas de g_k e f_k as forças externas que agem sob o sistema.

O modelo dinâmico de um robô móvel sem restrições de movimento pode ser definido por

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{F}(\dot{\mathbf{q}}) + \mathbf{G}(\mathbf{q}) = \mathbf{E}(\mathbf{q})\mathbf{u}, \tag{12}$$

em que,

$\mathbf{M}(\mathbf{q})$	Matriz de massa e inércia
$\mathbf{q}$	Vetor das coordenadas generalizadas
$\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$	Vetor de força centrifuga e Coriolis
$\mathbf{F}(\dot{\mathbf{q}})$	Vetor de atrito
$\mathbf{G}(\mathbf{q})$	Vetor da força gravitacional
$\mathbf{E}(\mathbf{q})$	Matriz de transformação dos atuadores
$\mathbf{u}$	Vetor de entradas do sistema

Para obter a solução numérica, basta calcular a integral da aceleração $\ddot{\mathbf{q}}$, conforme apresentado pela Equação 13.

$$\ddot{\mathbf{q}} = \mathbf{M}(\mathbf{q})^{-1} \left\{ -\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) - \mathbf{F}(\dot{\mathbf{q}}) - \mathbf{G}(\mathbf{q}) + \mathbf{E}(\mathbf{q})\mathbf{u} \right\}. \tag{13}$$

Quando as restrições de movimento são superiores ao grau de liberdade do sistema ($m > n$), como no exemplo do robô diferencial, esse sistema é classificado como não-holonômico. A classificação não-holonômica será vista no próximo segmento.

2.1.2.2 Sistemas Não-Holonômicos

De forma análoga ao sistema holonômico, os sistemas não-holonômicos apresentam limitação possíveis de velocidade ou direção de movimento (KLANCAR *et al.*, 2017). As equações de restrições podem ser formuladas como apresentado pela Equação 14.

$$f(\mathbf{q}, \dot{\mathbf{q}}) = f(q_1, \cdots, q_n, \dot{q}_1, \cdots, \dot{q}_n) = 0 \tag{14}$$

Uma restrição cinemática, Equação 14, é holonômica se ela pode ser integrável, ou seja, as velocidades $\dot{q}_1, ..., \dot{q}_n$ podem ser eliminadas da equação e a restrição pode ser expressar pela Equação 7, caso contrario ela é não-holonômica (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011).

As restrições estão relacionadas a categoria de rodas utilizadas no projeto do robô móvel. A Figura 2 apresenta os modelos possíveis, sendo que cada modelo impõe uma restrição que pode ser descrita geometricamente. Com as restrições bem definidas pode-se propor o modelo dinâmico para o robô diferencial, da Figura 1.

Tais restrições, com já comentado, estão diretamente associadas aos modelos de rodas utilizadas no projeto de robô móvel. A Figura 2 demonstra a variedade de opções. Cada modelo, impõe uma restrição que poderá ser descrita geometricamente.

2.1.2.3 Dinâmica do Robô Diferencial

As rodas convencionais, utilizadas pelo robô da Figura 1, apresentam restrições de locomoção limitada ao eixo x, por serem fixas as laterais do chassi. Com essas informações, é

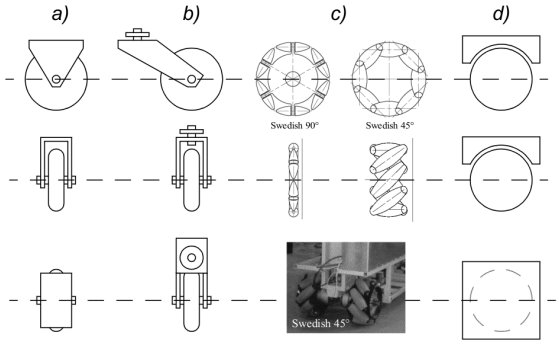


Figura 2 – Rodas (a) Convencionais (b) Castor (c) Sueca (d) Esférica. Fonte: Adaptado de (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011).

apresentado pela Equação 15, o modelo cinemático do robô diferencial.

$$\begin{bmatrix} \dot{X}(t) \\ \dot{Y}(t) \\ \dot{\phi}(t) \end{bmatrix} = \begin{bmatrix} \cos(\phi(t)) & 0 \\ \sin(\phi(t)) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_R(t) \\ \omega(t) \end{bmatrix}, \tag{15}$$

suas restrições são dadas por

$$-\dot{x}\sin(\phi) + \dot{y}\cos(\phi) = 0, \tag{16}$$

sendo descritas na forma matricial, por **A**, conforme:

$$\mathbf{A} = \begin{bmatrix} -\sin(\phi) & \cos(\phi) & 0 \end{bmatrix}. \tag{17}$$

A Figura 3, mostra de forma gráfica as possibilidades de movimentação do robô diferencial. O vetor de movimento v' não é valido dadas as restrições do modelo de rodas utilizado pelo robô, que impede que ele se mova lateralmente.

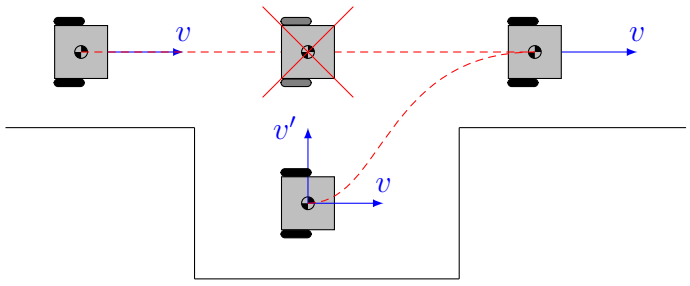


Figura 3 – Restrições de Movimento das Rodas do Robô Diferencial Fonte: (LIMA, 2021)

Com as restrições bem definidas, pode-se iniciar a definição do sistema dinâmico do robô diferencial pela Formulação de Lagrange, dada por

$$\mathcal{L} = \mathcal{T} - \mathcal{V},$$

em que a equação de energia cinética $\mathcal{T}$, é definida por

$$\mathcal{T} = \sum_{i=0}^{N-1} \frac{1}{2} {}^{i+1}_{i}\dot{\mathbf{P}}^{T} \cdot m_i \cdot {}^{i+1}_{i}\dot{\mathbf{P}} + \omega_i^T \cdot \mathbf{J}_i \cdot \omega_i. \tag{18}$$

Como o sistema é não-holonômico, tem-se que

$$\frac{d}{\mathrm{d}t}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}_k}\right) - \frac{\partial \mathcal{L}}{\partial q_k} + \frac{\partial P}{\partial \dot{q}_k} + \tau_{d_k} = f_k - \sum_{j=1}^{m} \lambda_j a_{jk}, \tag{19}$$

sendo m o número de restrições de movimento linearmente independentes, k o índice que descreve as coordenadas gerais para q_k, as energias dissipativas são dadas por P, os distúrbios no sistema são representados por τ_d, f_k são as forças externas relacionadas às coordenadas gerais q_k, λ_j é o multiplicador de Lagrange associado à j-ésima relação de restrição e a_{jk} são os coeficientes das restrições de movimento, dadas pela Equação 17.

O modelo dinâmico de um robô móvel com restrições pode ser expresso em forma de matriz da seguinte forma:

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{F}(\dot{\mathbf{q}}) + \mathbf{G}(\mathbf{q}) = \mathbf{E}(\mathbf{q})\mathbf{u} - \mathbf{A}(\mathbf{q})^T\boldsymbol{\lambda}, \tag{20}$$

em que:

$\mathbf{M}(\mathbf{q})$	Matriz de massas e inercia
$\mathbf{q}$	Vetor de coordenadas generalizadas
$\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$	Vetor da força Coriolis e centrifuga
$\mathbf{F}(\dot{\mathbf{q}})$	Vetor de atrito
$\mathbf{G}(\mathbf{q})$	Vector da força gravitacional
$\mathbf{E}(\mathbf{q})$	Matriz de transformação dos atuadores
$\mathbf{u}$	Vetor de entrada
$\mathbf{A}^T(\mathbf{q})$	Matriz de restrições de movimento
$\boldsymbol{\lambda}$	Multiplicador de Lagrange

Para resolver λ é utilizado as pseudo-velocidades. O objetivo, é solucionar as restrições de λ no sistema da Equação 21.

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{F}(\dot{\mathbf{q}}) + \mathbf{G}(\mathbf{q}) = \mathbf{E}(\mathbf{q})\mathbf{u} - \cancel{\mathbf{A}(\mathbf{q})^T\boldsymbol{\lambda}} \tag{21}$$

O modelo cinemático da Equação 15, pode ser reescrito no formato matricial, conforme

$$\dot{\mathbf{q}} = \mathbf{S}(q)\mathbf{v}, \tag{22}$$

em que, $\mathbf{S}(q)$ representa a matriz jacobiana do modelo cinemático e o termo $\mathbf{v}$ as velocidades atuantes no modelo. Sendo assim, é possível derivar a Equação 22, para obter as equações de acelerações do sistema, dadas por

$$\ddot{\mathbf{q}} = \dot{\mathbf{S}}(q)\mathbf{v} + \mathbf{S}(q)\dot{\mathbf{v}}. \tag{23}$$

Agora substituindo a Equação 23 na Equação 21 e aplicando $\mathbf{A}(q)\mathbf{S}(q) = 0$, obtendo a Equação 24, que representa a aceleração do sistema.

$$\dot{\mathbf{v}} = \tilde{\mathbf{M}}^{-1}\left(\tilde{\mathbf{E}}\mathbf{u} - \tilde{\mathbf{V}}\right) \tag{24}$$

Ainda, $\mathbf{S}^T\mathbf{E} \neq 0$ for verdade, a equação Equação 25 também é válida.

$$\mathbf{u} = \tilde{\mathbf{E}}^{-1}\left(\tilde{\mathbf{M}}\dot{\mathbf{v}} + \tilde{\mathbf{V}}\right) \tag{25}$$

Sendo assim, o sistema pode ser reescrito para $\dot{\mathbf{x}} = f(\mathbf{x}) + g(\mathbf{x})\mathbf{u}$, em que $\mathbf{x} = \begin{bmatrix}\mathbf{q}^T\ \mathbf{v}^T\end{bmatrix}^T$, desta forma, o sistema em espaço de estados é representado pela Equação 2.1.2.3.

$$\dot{\mathbf{x}} = \begin{bmatrix}\mathbf{S}(q)\mathbf{v} \\ -\tilde{\mathbf{M}}^{-1}\tilde{\mathbf{V}}\end{bmatrix} + \begin{bmatrix}\mathbf{0} \\ \tilde{\mathbf{M}}^{-1}\tilde{\mathbf{E}}\end{bmatrix}\mathbf{u}$$

em que

$$\begin{aligned}\tilde{\mathbf{V}} &= \mathbf{S}(q)^T\mathbf{M}\dot{\mathbf{S}}(q)\mathbf{v} + \mathbf{S}(q)^T(\mathbf{V} + \mathbf{F} + \mathbf{G}),\\ \tilde{\mathbf{M}} &= \mathbf{S}(q)^T\mathbf{M}\mathbf{S}(q),\\ \tilde{\mathbf{E}} &= \mathbf{S}(q)^T\mathbf{E}\mathbf{S},\end{aligned}$$

sendo $\mathbf{x}$ o vetor de estados e $\mathbf{S}$ a matriz Jacobiana.

O próximo passo é a definição das equações dinâmicas para o robô da Figura 1. Novamente começamos pela definição das energias cinética $\mathcal{T}$ e potêncial $\mathcal{V}$, para a Equação 8. $\mathcal{T}$ é dado pela Equação 26.

$$\mathcal{T} = \frac{m}{2}\left(\dot{x}^2 + \dot{y}^2\right) + \frac{J}{2}\phi^2. \tag{26}$$

Para a energia potencial, $\mathcal{V} = 0$, assim o funcional pode ser definido por

$$\mathcal{L} = \mathcal{T} - \mathcal{V} = \frac{m}{2}\left(\dot{x}^2 + \dot{y}^2\right) + \frac{J}{2}\phi^2. \tag{27}$$

Na formulação do Lagrange não são consideradas forças não conservativas, desta forma, elas podem ser adicionadas após a formulação. Então tem-se

$$\begin{aligned}\frac{d}{\mathrm{d}t}\left(\frac{\partial\mathcal{L}}{\partial\dot{x}}\right) &= m\ddot{x},\\ \frac{d}{\mathrm{d}t}\left(\frac{\partial\mathcal{L}}{\partial\dot{y}}\right) &= m\ddot{y},\\ \frac{d}{\mathrm{d}t}\left(\frac{\partial\mathcal{L}}{\partial\dot{\phi}}\right) &= J\ddot{\phi}\end{aligned} \tag{28}$$

e também,

$$\begin{aligned}\frac{d}{\mathrm{d}t}\left(\frac{\partial \mathcal{L}}{\partial x}\right) &= 0,\\ \frac{d}{\mathrm{d}t}\left(\frac{\partial \mathcal{L}}{\partial y}\right) &= 0,\\ \frac{d}{\mathrm{d}t}\left(\frac{\partial \mathcal{L}}{\partial \phi}\right) &= 0.\end{aligned} \tag{29}$$

Com as forças que atuam sob robô diferencial, é tido que

$$\begin{aligned}m\ddot{x} - \lambda_1 \sin(\phi) &= F_x,\\ m\ddot{y} - \lambda_2 \cos(\phi) &= F_y,\\ J\ddot{\phi} &= \mathbb{T}.\end{aligned} \tag{30}$$

Definindo $F_x = 1/r_r\,(\tau_d + \tau_e)\cos(\phi)$, em que τ_d é o torque da roda direita e τ_e da esquerda. E também, $F_y = 1/r\,(\tau_d + \tau_e)\sin(\phi)$. O torque aplicado ao robô é $\mathbb{T} = L/(2r)\,(\tau_d - \tau_e)$. Aplicando na Equação 30, são obtidas as seguintes equações:

$$\begin{aligned}m\ddot{x} - \lambda_1 \sin(\phi) - \frac{1}{r_r}(\tau_d + \tau_e)\cos(\phi) &= 0\\ m\ddot{y} - \lambda_2 \cos(\phi) - \frac{1}{r_r}(\tau_d + \tau_e)\sin(\phi) &= 0\\ J\ddot{\phi} - \frac{L}{2r_r}(\tau_d - \tau_e) &= 0\end{aligned} \tag{31}$$

O sistema da Equação 21, reescrito no formato matricial, para o robô diferencial é dado por

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{F}(\dot{\mathbf{q}}) + \mathbf{G}(\mathbf{q}) = \mathbf{E}(\mathbf{q})\mathbf{u} - \mathbf{A}(\mathbf{q})^{\mathbf{T}}\boldsymbol{\lambda}, \tag{32}$$

em que as matrizes são:

$$\mathbf{M} = \begin{bmatrix} m & 0 & 0\\ 0 & m & 0\\ 0 & 0 & J\end{bmatrix}, \quad \mathbf{E} = \frac{1}{r_r}\begin{bmatrix} \cos(\phi) & \cos(\phi)\\ \sin(\phi) & \sin(\phi)\\ \frac{L}{1} & -\frac{L}{2}\end{bmatrix}, \quad \mathbf{u} = \begin{bmatrix}\tau_1\\ \tau_2\end{bmatrix}$$

e a matriz de restrição:

$$\mathbf{A} = \begin{bmatrix} -\sin(\phi) & \cos(\phi) & 0\end{bmatrix}$$

Utilizando a Equação 24, as restrições do sistema podem ser resolvidas, como apresentado por:

$$\tilde{\mathbf{M}} = \begin{bmatrix} m & 0 \\ 0 & J \end{bmatrix}, \quad \tilde{\mathbf{V}} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad \text{e} \quad \tilde{\mathbf{E}} = \frac{1}{r_r} \begin{bmatrix} 1 & 1 \\ \frac{L}{2} & -\frac{L}{2} \end{bmatrix}.$$

O sistema em espaço de estados, na forma $\mathbf{x} = f(\dot{\mathbf{x}}) + g(\mathbf{x})\mathbf{u}$, é apresentado pela Equação 33.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\phi} \\ \dot{v} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} v\cos(\phi) \\ v\sin(\phi) \\ \omega \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ \frac{1}{mr_r} & \frac{1}{mr_r} \\ \frac{L}{2Jr_r} & -\frac{L}{2Jr_r} \end{bmatrix} \begin{bmatrix} \tau_d \\ \tau_e \end{bmatrix} \tag{33}$$

Assim fica definido o modelo dinâmico do robô diferencial.

Em uma aplicação prática do robô diferencial, o sistema pode divergir de forma considerável do modelo apresentado pelaEquação 33. Isso se da pelas incertezas relacionadas aos sensores que fazem a leitura dos estados do robô.

2.2 APRENDIZADO POR REFORÇO

O aprendizado por meio da interação é uma das formas mais básicas de aprendizado, um aprendizado orgânico obtido a partir de um retorno para uma interação feita, o simples movimento de um braço traz conhecimento do ambiente ao seu redor e das suas capacidades motoras. A todo momento o cérebro capta incontáveis respostas as suas ações e as utiliza para definir futuras ações que possam ter um retorno semelhante. Aprendizado por interações é uma ideia fundamental para quase todas as teorias de aprendizado e inteligência (SUTTON; BARTO, 2018), isso incluí a abordagem computacional do aprendizado por reforço (Aprendizado por Reforço (AR)).

O aprendizado por reforço é uma classe de algoritmos na área de aprendizado de máquina que foca em permitir que o agente aprenda como se comportar em um ambiente, em que o único retorno é um sinal de recompensa numérico (WIERING; OTTERLO, 2012). O objetivo do agente é obter a máxima recompensa numérica a longo prazo, sem receber nenhuma instrução de quais ações tomar, ele deve tentar realizar as ações possíveis e descobrir as recompensas máximas por si só. E em certos casos ações podem influenciar recompensas futuras, trazendo a necessidade de pensar em maximizações fora do estado imediato (SUTTON; BARTO, 2018).

Todo método adequado para resolver um problema, que envolva um agente que pode sentir o estado do seu ambiente de alguma forma e dentro desse ambiente possa tomar ações que influenciam esse estado e tenha um objetivo relacionado ao ambiente, pode ser considerado um método de aprendizado por reforço. Ele difere do aprendizado supervisionado por não usar um

conjunto de treinos fornecido por um agente externo com conhecimento do problema. Também não pode ser classificado como um aprendizado não supervisionado, pois este busca estruturas escondidas em meio a dados sem rotulação e o aprendizado por reforço foca na maximização das recompensas.

Além de um agente e um ambiente, é possível identificar quatro subelementos de um sistema de aprendizado por reforço: uma política, um sinal de recompensa, uma função de valor e um modelo do ambiente (SUTTON; BARTO, 2018), o último podendo ser opcional.

Uma política é um mapeamento de todos os estados para uma ação, baseado em medidas teóricas de decisão com relação a um objetivo a ser otimizado (WIERING; OTTERLO, 2012). Dessa forma ela define como o agente se comporta no ambiente, basicamente sendo um conjunto de regras que regem as ações possíveis dado um determinado estado do ambiente. Elas podem ser simples tabelas, sequências de direções ou até mesmo complexos algoritmos de busca.

O principal elemento básico de um sistema de aprendizado por reforço é o sinal de recompensa, é ele que define os objetivos para o problema, a cada troca de estado o ambiente traz um retorno numérico positivo ou negativo para o agente. De forma análoga seres vivos recebem sinais de prazer ou dor ao realizar uma ação. O sinal de recompensa é o fator decisivo para a alteração de política do agente, quando uma ação tomada por uma política traz um retorno negativo em ações futuras é possível que outra política seja usada para uma situação similar.

Um sinal de recompensa indica o que é bom imediatamente, uma função de valor especifica o que é bom a longo prazo (SUTTON; BARTO, 2018). Em termos gerais, funções de valor são uma maneira de ligar critérios de otimização com políticas, elas representam uma estimativa numérica de quão bom é para um agente estar em um estado, ou realizar uma determinada ação em um estado (WIERING; OTTERLO, 2012). Como exemplo temos duas recompensas, uma recompensa positiva com valor baixo e uma negativa que leva possivelmente a uma recompensa positiva de valor maior, a função de valor irá calcular valores para cada um dos caminhos, o que traz uma perspectiva geral para as possíveis ações. Em uma analogia com o pensamento humano, pode-se classificar a função de valor como um julgamento a longo prazo do que trará mais contentamento ou descontentamento. Mesmo o valor sendo um subproduto das recompensas é o valor a principal fonte de informações, para que um método de aprendizado por reforço possa tomar suas decisões, pois ela sempre busca a máxima recompensa total.

Por fim temos o modelo do ambiente, algo que não necessariamente está presente em todos os sistemas de aprendizado por reforço. O modelo ira imitar os comportamentos do ambiente, formando uma espécie de ambiente de testes, em que o agente pode prever os possíveis próximos estados e recompensas, sem a necessidade da tentativa e erro. Esse modelo pode ser criado com base nas experiências prévias do agente, facilitando assim o julgamento para situações similares as já experienciadas.

Definindo de forma mais específica as interações entre o agente e o ambiente, temos uma sequência definida por um tempo discreto, $t = 0, 1, 2, 3, 4...$ e a cada tempo t o agente

obtém uma representação do estado do ambiente, $S_t \in \mathcal{S}$, em que $\mathcal{S}$ é o conjunto de estados possíveis e com base nisso escolhe uma ação, $A_t \in \mathcal{A}(S_t)$, em que $\mathcal{A}(S_t)$ é o conjunto de ações possíveis no estado S_t. No próximo passo, como consequência da ação o agente recebe uma recompensa numérica, $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$ e com isso se encontra em um novo estado S_{t+1}. O ciclo é representado pela Figura 4. Já uma política do agente é denominada π_t, em que $\pi_t(a|s)$ é a probabilidade de que $A_t = a$ e $S_t = s$.

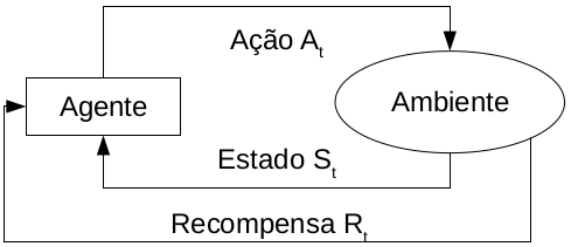


Figura 4 – Ciclo de interação entre agente e ambiente
Fonte: Autoria própria.

2.2.1 Processo de Decisão de Markov

O processo de decisão de Markov (do inglês, *Markov Decision Process* (MDP)) é uma forma de modelar processos em que as transições de estados são probabilísticas, o estado do processo é observável e o agente pode interferir no processo através de ações (PELLEGRINI; WAINER, 2007).

Antes de aprofundar no conhecimento de um MDP é preciso explicar a propriedade de Markov. Um estado que sumariza as sensações antecedentes a ele, de forma que toda informação relevante é mantida, sendo que todas as possibilidades de transição e recompensas dependem somente desse estado (PUTERMAN, 1990), pode ser chamado Markov, ou que possui a propriedade de Markov. Um exemplo é um estado de uma partida de xadrez, em que é possível observar todas as posições das peças do tabuleiro, mesmo que não tenhamos as informações de todos os estados é possível tomar uma decisão informada, pois toda a informação necessária está sumarizada nas posições atuais.

Toda tarefa de AR que satisfaz a propriedade de Markov pode ser chamada MDP, caso o espaço de estados e ações sejam finitos, então chamamos MDP finito. MDPs finitos são particularmente importantes na teoria de AR, eles são tudo o que é necessário para entender 90% do AR moderno (SUTTON; BARTO, 2018).

Um MDP finito é definido pelo seu conjunto de estados e ações, de forma que dado qualquer estados s e ação a, a probabilidade de cada estado s' e recompensa r possível é demonstrada por

$$p(s',r|s,a) = Pr\{S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a\}. \tag{34}$$

A partir de Equação 34, é possível obter todas as informações sobre o ambiente, como a recompensa esperada por um estado e ação,

$$r(s,a) = \mathbb{E}[R_{t+1}|S_t = s,A_t = a] = \sum_{r\in\mathbb{R}} r \sum_{s'\in\mathcal{S}} p(s',r|s,a), \tag{35}$$

a recompensa esperada para estado, ação e próximo estado,

$$r(s,a,s') = \mathbb{E}[R_{t+1}|S_t = s,A_t = a,S_{t+1} = s'] = \frac{\sum_{r\in\mathbb{R}} rp(s',r|s,a)}{p(s'|s,a)}, \tag{36}$$

e a probabilidade do estado e da transição,

$$p(s'|s,a) = Pr\{S_{t+1} = s'|S_t = s,A_t = a\} = \sum_{r\in\mathbb{R}} p(s',r|s,a). \tag{37}$$

2.2.1.1 Função de Valor

Para determinar o quão bom é um agente estar em um estado ou fazer uma ação nesse estado, é necessária uma função de valor para obter as possíveis recompensas (WIERING; OTTERLO, 2012).

Uma função valor é definida com base em uma política, π, que mapeia cada estado, $s \in \mathcal{S}$, e ação, $a \in \mathcal{A}(s)$, a uma probabilidade $\pi(a|s)$ de fazer uma ação a em um estado s. Com base nisso é definido uma função de estado-valor para a política π, $v_\pi(s)$, dada por

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}\middle| S_t = s\right], \tag{38}$$

em que $\mathbb{E}_\pi$ é o valor esperado para uma variável aleatória dado que o agente seguiu uma política π e γ é um valor de desconto tal que $0 < \gamma \leq 1$.

Para se determinar o valor de uma ação a em um estado s para uma política π, temos a função $q_\pi(s,a)$ chamada função ação-valor para política π, sendo representada por

$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}_\pi[G_t|S_t = s,A_t = a] \\ &= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}\middle| S_t = s,A_t = a\right]. \end{aligned} \tag{39}$$

Ambas funções de valores v_π e q_π podem ser obtidas por meio das experiências do agente. Ao seguir uma política π diversas vezes, armazenando os retornos para cada estado, o agente pode calcular uma média desses valores, que será o valor de v_π. Se, além disso, o agente armazenar as médias para cada ação tomada, ele irá obter o valor de q_π.

Um fator importante a ser notado é a relação entre o valor de um estado e o valor de seus estados sucessores, para isso temos a equação de Bellman para v_π, definida por

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')], \tag{40}$$

em que $a \in \mathcal{A}(s)$, o próximo estado $s' \in \mathcal{S}$ e $r \in \mathcal{R}$. A equação faz a média de todas as possibilidades, considerando a probabilidade de cada uma acontecer (SUTTON; BARTO, 2018).

De maneira geral resolver um problema MDP finito, requer encontrar a política que obtenha a maior recompensa a longo prazo. Para que uma política π seja melhor que uma política π', ela deve ter retornos melhores ou iguais para todos os estados (SILVER, 2015a), ou seja, $\pi > \pi'$ se e somente se $v_\pi(s) \geq v'_\pi(s)$ para todos $s \in \mathcal{S}$. Em todos os caso existe pelo menos uma política que é melhor ou igual a todas as outras políticas, essa ou essas políticas são denominadas de π_*. Elas compartilham a mesma função estado-valor, chamada função estado-valor ótima, definida por

$$v_*(s) = \max_\pi v_\pi(s), \forall s \in \mathcal{S}. \tag{41}$$

Da mesma forma compartilham uma função ação-valor, dada por

$$q_*(s,a) = \max_\pi q_\pi(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}, \tag{42}$$

dessa forma é possível escrever a função q_* em termos de v_*, como

$$q_*(s,a) = \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1})|S_t = s, A_t = a]. \tag{43}$$

Por fim é definido as equações ótimas de Bellman, que tem grande utilidade em soluções iterativas para problemas de AR. As equações $v_*(s)$ e $q_*(s,a)$ estão demonstradas abaixo

$$v_*(s) = \max_{a \in \mathcal{A}(s)} \sum_{s',r} p(s',r|s,a)[r + \gamma v_*(s')], \tag{44}$$

$$q_*(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma \max_{a'} q_*(s',a')]. \tag{45}$$

2.2.2 Métodos de Solução para o problema de AR

Para problemas que não podem ser solucionados de forma analítica, temos alguns métodos que conseguem solucionar problemas de forma iterativa. Os métodos discutidos nesse capítulo são: programação dinâmica, o método de Monte Carlo e o método das diferenças temporais. Cada um tem características que os tornam úteis em diferentes áreas.

2.2.2.1 Programação Dinâmica

As equações ótimas de Bellman são equações não lineares, que podem não ter uma solução analítica, que resolva de forma simples o problema (SILVER, 2015b). Para resolver problemas dessa natureza, é necessário a utilização de métodos iterativos, como o método tratado neste capítulo, chamado programação dinâmica.

A programação dinâmica (Programação Dinâmica (PD)), na área de AR, consiste em uma série de algoritmos para calcular as políticas ótimas de um problema MDP, desde que se tenha o conhecimento completo do ambiente. Em outras palavras é um método, na sua forma mais simples, que aproxima o resultado das equações ótimas de Bellman, através de iterações (SUTTON; BARTO, 2018). Mesmo que em sua forma básica não tenha tanta relevância atualmente, por ser um método custoso que depende de modelos perfeitos de MDP, ainda é um método base para o entendimento de outros métodos.

O primeiro fator na PD é a avaliação de políticas, em que é feito o cálculo da função estado-valor v_π para uma política aleatória π. Utilizando a equação de Bellman para v_π é feita uma iteração pelas políticas

$$\begin{aligned} v_{k+1}(s) &= \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_k(s')], \\ &\forall s \in \mathcal{S}. \end{aligned} \tag{46}$$

Essa equação é chamada avaliação iterativa de políticas. Dado que o valor de desconto $\gamma < 1$, a equação converge para o estado $v_k = v_\pi$, considerando $k \to \infty$.

Para produzir cada iteração da aproximação, a equação de avaliação de políticas iterativa aplica a mesma operação para todos os estados s, trocando os valores de s antigo por novos valores de s a cada iteração, essa é uma operação de *backup* completo (SUTTON; BARTO, 2018). Cada iteração guarda, faz *backup*, do valor da anterior, para produzir o valor da próxima iteração.

Indo adiante no método de PD, o próximo passo é a melhora das políticas. Com o valor de v_π calculado, pode-se tentar avaliar uma ação $a \neq \pi(s)$ para um estado s, em busca de aprimoramento no valor. Para esse calculo há a equação

$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a] \\ &= \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')], \end{aligned} \tag{47}$$

em que pode ser que sempre que escolher a resulte em um valor maior que $v_\pi(s)$, então essa ação traz melhores resultados.

Esse é um caso especial em que, dado um par de políticas π e π', temos que se para todo $s \in \mathcal{S}$,

$$q_\pi(s,\pi'(s)) \geq v_\pi(s), \tag{48}$$

então a política π' obtém um valor maior ou igual a π,

$$v_{\pi'}(s) \geq v_\pi(s). \tag{49}$$

Outra propriedade aplicável é a de que se $v_{\pi'}$ for igual a v_π ambas as funções de valor são funções ótimas v_*. Sendo assim o melhoramento de política sempre nos traz políticas melhores, a menos que a política original já seja ótima (SUTTON; BARTO, 2018).

Visto que exista uma nova política π' melhor que π, é possível começar uma sequência de avaliações de política e aprimoramentos de política. Temos então a sequência:

$$\pi_0 \xrightarrow{aval} v_{\pi 0} \xrightarrow{apri} \pi_1 \xrightarrow{aval} v_{\pi 1} \xrightarrow{apri} \pi_2 \xrightarrow{aval} \ldots \xrightarrow{apri} \pi_* \xrightarrow{aval} v_*, \tag{50}$$

em que a iteração pode ser feita até chegar na política ótima π_*, o limite do aperfeiçoamento.

Além das técnicas apresentadas, existem vários aperfeiçoamentos e outras técnicas de PD como: a iteração de valores, a versão assíncrona de PD, PD adaptativa e outras. Como não é o foco do trabalho, apenas os conceitos inicias bastam para a construção da fundamentação.

2.2.2.2 Método de Monte Carlo

O método de Monte Carlo (Monte Carlo (MC)) é uma técnica estatística de amostragem, que durante os anos foi aplicada com sucesso em vasto número de problemas cientificos (ECKHARDT; ULAM; NEUMANN, 1987). Esse, diferente do método de programação dinâmica, é um modo de aprendizado, que requer somente experiencia com o ambiente e não o conhecimento total dele. Mesmo utilizando apenas a experiência é possível chegar a um comportamento ótimo, isso também se aplica as experiências simuladas, feitas em um modelo do ambiente. Neste segmento será apenas tratado problemas com sequências finitas, ou episódicas, permitindo a realização de médias fechadas.

De forma similar a PD, o primeiro conhecimento necessário é relacionado a função de estado-valor $v_\pi(s)$ para uma política π, dado um conjunto de obtidos por seguir π e passando por s. Cada ocorrência de um estado s é chamada visita ao s, e o método chamado MC primeira-visita, que estima o valor de $v_\pi(s)$ a partir de uma média dos retornos de cada primeira visita a s (SUTTON; BARTO, 2018). O método de MC toda-visita faz a média de todas as visitas a s, será tratado apenas o método primeira-visita por ser o mais estudado. O algoritmo para estimar o valor da função estado-valor apresentado pelo Algoritmo 1.

Algoritmo 1 – Método Monte Carlo primeira-visita

$\pi \leftarrow$ política a ser estimada
$V \leftarrow$ uma função estado-valor arbitraria
$Retorna(s) \leftarrow$ uma lista vazia, $\forall s \in \mathcal{S}$
sequência
    Gera um episódio usando π
    para cada estado s que aparece no episódio **faça**
        $G \leftarrow$ retorna após a primeira ocorrência de s
        Coloca G no fim de $Retorna(s)$
        $V(s) \leftarrow$ média(Retorna(s))
    finaliza para
finaliza sequência

Fonte: Adaptado de (SUTTON; BARTO, 2018)

A estimativa de estado-valor depende de um modelo do ambiente para estimar o valor, para quando um modelo não exista é feita a estimativa da política pela função ação-valor. Sem o modelo é necessário estimar cada ação para os estados, de modo a ter os valores para uma política desejada.

Na função ação-valor, a estimativa é feita da mesma forma que para a função estado-valor, com a diferença que é estimada $q_\pi(s,a)$ para o retorno esperado começando em um estado s, tomando uma ação a e seguindo a política π. O MC primeira-visita agora se aplica a primeira vez que uma ação a é tomada em um estado s, o que gera um problema da necessidade de estimas o valor para todas as ações para um estado, não apenas a que a política atualmente favorece (SUTTON; BARTO, 2018).

De forma si miliar a PD é possível seguir uma sequência de avaliação e aprimoramento, como:

$$\pi_0 \xrightarrow{aval} q_{\pi 0} \xrightarrow{apri} \pi_1 \xrightarrow{aval} q_{\pi 1} \xrightarrow{apri} \pi_2 \xrightarrow{aval} \ldots \xrightarrow{apri} \pi_* \xrightarrow{aval} q_*, \tag{51}$$

em que a cada passo o método de Monte Carlo calcula o valor de $q_{\pi k}$ para uma política arbitraria πk. A etapa de aprimoramento acontece fazendo a política ser gulosa com relação ao valor da função, ou seja

$$\pi(s) = arg\max_a q(s,a), \tag{52}$$

assim o teorema de aprimoramento se aplica fazendo com que cada $\pi_{k+1} \geq \pi_k$ (SUTTON; BARTO, 2018).

Além das técnicas de Monte Carlo apresentadas, existem outras que usam essa base para obter resultados melhores. Para o foco do trabalho as bases são o suficiente para avançar na fundamentação.

2.2.2.3 Diferenças Temporais

Aprendizado de diferença-temporal (Diferença-temporal (DT)), método cunhado por Sutton (SUTTON, 1988), é uma combinação das ideias de Monte Carlo e programação dinâmica. É usada amplamente em métodos de AR para aprender a fazer previsões momento a momento de recompensas futuras, sendo normalmente mais simples e eficiente (SUTTON; TANNER, 2004) do que métodos anteriormente citados.

De maneira semelhante aos métodos de Monte Carlo, a DT pode aprender diretamente de experiências cruas, sem um modelo das dinâmicas do ambiente. E como a programação dinâmica, ela atualiza as estimativas com base em outras estimativas aprendidas, sem a necessidade do resultado para a atualização dos seus fatores (SUTTON; BARTO, 2018).

Seguindo o primeiro passo dos métodos de MC e PD, é inicialmente feita a avaliação de política, em que é estimado a função valor v_π para uma política π. A atualização dos valores de v_π ocorre, de forma semelhante ao método MC, ao experienciar uma política para os estados S_t. No Monte Carlo a atualização ocorre após o retorno para a visita já ser conhecido, ou seja, quando o episódio analisado chegou ao fim, já nos métodos de DT a atualização acontece a cada iteração. No tempo $t+1$ elas imediatamente formam um alvo e fazem uma atualização usando a recompensa observada R_{t+1} e estimam o valor $V(S_{t+1})$ (SUTTON; BARTO, 2018). O método de DT chamado TD(0), avalia conforme

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)], \tag{53}$$

em que $R_{t+1} + \gamma V(S_{t+1})$ é o fator de atualização.

De forma geral, os métodos de PD utilizam estimativas semelhantes, como visto na Equação 46, porém o valor $v_{k+1}(S_{t+1})$ já é conhecido previamente, enquanto no TD(0) o $V(S_{t+1})$ é uma estimativa calculada de forma dinâmica, semelhante ao Monte Carlo. Isso traz a possibilidade de o método DT fazer um bom uso das vantagens de ambos os métodos de MT e PD.

A forma de aprimoramento de políticas ocorre de forma semelhante às apresentadas no método PD e de MC, usando a iteração entre avaliação e aprimoramento. Com a construção dos conhecimentos antecessores a ele e a evolução apresentada, o método de diferença-temporal serve como uma base forte para outros métodos que serão tratados neste trabalho.

2.2.3 Elementos de Aprendizado

Nesta seção são tratados os métodos relacionados ao *Deep Q-learning* (DQN), método utilizado para o aprendizado de navegação autônoma do robô. Inicialmente é introduzido o método chave, chamado *Q-learning*, para o aprendizado por reforço escolhido, após isso o

método SARSA, que aprimora algumas técnicas do anterior e em sequência é introduzido o DQN.

2.2.3.1 Q-Learning

Q-learning (WATKINS, 1989) é uma forma de aprendizado por reforço livre de modelos, que pode ser vista como um método de programação dinâmica assíncrona (WATKINS; DAYAN, 1992). Da mesma forma que os métodos de Monte Carlo e diferenças-temporais, ele consegue fazer um agente aprender através de experiências das suas ações no ambiente, dado um problema Markoviano.

O método segue um caminho muito parecido com o método de DT, em que um agente realiza uma ação em um determinado estado, avaliando a sua recompensa imediata e a estimativa do valor do estado que essa ação o levou. Ao tentar todas as ações em todos os estados repetidas vezes, ele aprende as melhores políticas em um contexto geral (WATKINS; DAYAN, 1992). Com a sua forma simples de aprendizado, ela serve como uma boa base para aplicações mais complexas.

A forma mais simples do método, *Q-learning* de um passo, é definida por

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t,A_t)], \tag{54}$$

em que ele busca estimar a função de ação-valor ótima q_*, obtendo assim a ou as políticas ótimas π_*, independente de qual política está sendo seguida. Ele segue os mesmos teoremas que os métodos apresentados anteriormente, então dado um número infinito de episódios repetidos ele irá convergir para q_*. O Algoritmo 2 mostra a forma procedural de solução.

Algoritmo 2 – Método *Q-learning* de um passo

Inicializa $Q(s,a)$, $\forall s \in \mathcal{S}$, $\forall a \in \mathcal{A}(\int)$, de forma arbitraria e $Q(terminal) = 0$
para cada episódio **faça**
  Inicializa S
  para cada passo do episódio até o S ser terminal **faça**
    Escolhe A de S usando a política de Q
    Toma a ação A, observa R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S,A)]$
    $S \leftarrow S']$
  finaliza para
finaliza para

Fonte: Adaptado de (SUTTON; BARTO, 2018)

Um fator importante a ser definido, é que *Q-learning* é um método *off-policy*. Isso significa que o método aprimora uma política diferente da política usada para gerar os dados. No método é utilizado uma sequência, em que para atualizar a função de ação-valor atual é utilizada a próxima função ação-valor e mesmo que o próximo estado s' seja conhecido, a ação a' não é

conhecida (ZHAO *et al.*, 2016). De forma gulosa, *Q-learning*, escolhe a próxima ação buscando maximizar o valor de $Q(s',a')$.

Estes são os fatores definidores do método, que será discutido e aprofundado pelos métodos dos próximos segmentos.

2.2.3.2 SARSA

Pegando a base do *Q-learning*, o método SARSA criado por Rummery e Niranjan (1994), traz uma solução similar e aprimorada do método anterior. Tendo o seu diferencial por ser um método *on-policy*, ou seja, um método que aprimora a política utilizada para gerar os dados.

O algoritmo para a atualização da função de ação-valor do método SARSA é

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t,A_t)], \tag{55}$$

sendo muito similar a Equação 54 e também buscando chegar a função ação-valor ótima q_*, consequentemente a política ótima π_*. A diferença chave é que na atualização de Q a ação A_{t+1} sera tomada, em contraste com o método *Q-learning* que escolhia uma ação de forma gulosa. Esse algoritmo utiliza todos os elementos da quíntupla de eventos, $(S_t,A_t,R_{t+1},S_{t+1},A_{t+1})$, dando nome ao método SARSA (SUTTON; BARTO, 2018).

Como o método SARSA escolhe suas ações com base na política, ele possui um melhor desempenho durante o seu treinamento. No caso do Q-learning uma ação gulosa pode levar a um estado prejudicial a ele, dito isso, ambos aprendem a política ótima com vantagens para cada um dos métodos em cenários diferentes.

2.2.3.3 Deep Q-Learning

Deep Q-learning ou *Deep Q-network* é um método recente para solucionar problemas de aprendizado por reforço, inicialmente introduzido por Mnih *et al.* (2013). O método faz o uso de uma rede neural convolucional (do inglês, CNN) para estimar o valor ótimo q_*, do *Q-learning* com algumas alterações.

A simples iteração de valores apresentada por Equação 54, é impraticável por estimar a função ação-valor para cada sequência, sem nenhuma generalização. O mais comum é a utilização de uma função aproximadora $Q(s,a;\theta) \approx Q_*(s,a)$ (MNIH *et al.*, 2013). O mais comum é a utilização de funções lineares para a aproximação, visto que o AR é conhecido por ser instável ou até divergir com o uso de funções não lineares como redes neurais (MNIH *et al.*, 2015). Para tratar esse problema foram utilizadas duas modificações principais no *Q-learning*. A primeira é a repetição de experiências, que aleatoriza a escolha de dados. E o segundo foi a utilização da atualização dos valores de Q de forma periódica, sem atualizar a toda iteração.

Para a parametrização e aproximação de $Q(s,a;\theta_i)$ é utilizada uma CNN, em que θ_i são os pesos da rede neural para a iteração i. A repetição de experiências se da pelo armazenamento das experiências do agente $e_t = (s_t,a_t,r_t,s_{t+1})$ em um conjunto de dados $D_t = \{e_1, e_2, ..., e_t\}$ para cada amostra de tempo t, a atualização ocorre em pequenos conjuntos (s,a,r,s') $U(D)$, retirados uniformemente de forma aleatória das amostras guardadas(MNIH *et al.*, 2015). Sendo assim a função de perda é dada por

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\ U(D)} \left[(r + \gamma \max_{a'} Q(s',a';\theta_i^-) - Q(s,a;\theta_i))^2 \right], \tag{56}$$

em que θ_i^- são os parâmetros da rede na iteração i. Diferenciando a função de perda em relação aos pesos, temos o seguinte gradiente:

$$\nabla_{\theta_i} L(\theta_i) = \mathbb{E}_{(s,a,r,s')} \left[(r + \gamma \max_{a'} Q(s',a';\theta_i^-) - Q(s,a;\theta_i)) \nabla_{\theta_i} Q(s,a;\theta_i) \right]. \tag{57}$$

O algoritmo, para o método, é apresentado no Algoritmo 3.

Algoritmo 3 – Método *deep Q-learning*

Inicializa D para o tamanho N
Inicializa a função Q com pesos aleatórios
para episódio = 1 até M **faça**
    Inicializa $s_1 = \{x_1\}$ e pré-processa a sequência $\phi_1 = \phi(s_1)$
    para t = 1 até T **faça**
        Com uma probabilidade ϵ escolhe uma ação aleatória a_t
        caso contrario escolhe $a_t = \max_a Q(\phi(s_t),a;\theta)$
        Executa a ação a_t e observa a recompensa r_t e o estado x_{t+1}
        Define $s_{t+1} = s_t,a_t,x_{t+1}$ e pré-processa $\phi_{t+1} = \phi(s_{t+1})$
        Guarda a transição $(\phi_t, a_t, r_t, \phi_{t+1})$ em D
        Pega uma amostra aleatória de transições $(\phi_j,a_j,r_j,\phi_{j+1})$ de D
        Define $y_j = \begin{cases} r_j & \text{para } \phi_{j+1} \text{ terminais} \\ r_j + \gamma \max_{a'} Q`(\phi_{j+1},a';\theta) & \text{para } \phi_{j+1} \text{ não-terminais} \end{cases}$
        Realiza o passo do gradiente descendente em $(y_j - Q(\phi_j,a_j;\theta))^2$ conforme a Equação 57
        A cada C passos restaura $Q` = Q$
    finaliza para
finaliza para

Fonte: Adaptado de (MNIH *et al.*, 2013)

Em aplicações recentes, o método DQN com algumas modificações, foi utilizado em soluções na área de robótica, exemplos são: o trabalho de Guo *et al.* (2020) que utilizou em uma solução para AGVs, o trabalho de Zhang *et al.* (2017) que apresentou uma solução para navegação autônoma e Tai, Paolo e Liu (2017) que usou uma variação para o mesmo propósito. Os resultados obtidos foram promissores e demonstram o poder do aprendizado profundo por reforço.

2.3 FERRAMENTAS DE DESENVOLVIMENTO

Neste segmento são tratadas as ferramentas utilizadas para o desenvolvimento da técnica e dos seus testes. É passada uma visão geral do ambiente de desenvolvimento utilizado para o robô e para a DQN.

2.3.1 ROS — *Robot Opearating System*

O *Robot Opearating System* (ROS) é um meta sistema operacional de código aberto para robôs, provendo serviços esperados por um sistema operacional convencional, incluindo abstração de *hardware*, controle de dispositivos de baixo nível, implementação de funcionalidades comumente usadas, troca de mensagem entre processos e gerenciamento de pacotes (ROS..., 2021). Além disso, é um *framework* com ferramentas úteis para o desenvolvimento para robôs, contando com uma comunidade rica em código aberto.

No controle de um robô, o ROS, precisa fazer a comunicação entre sensores e atuadores do mesmo. Para realizar a comunicação entram alguns dos conceitos-chave para o funcionamento do sistema, eles são os nós, tópicos, mensagens e serviços. Conforme (JOSEPH, 2018), os nós são processos ou módulos que podem publicar ou subscrever a um canal de comunicação, chamado tópico, em que são trocadas mensagens, ou ainda podem utilizar um serviço para chamada de funções, utilizando um sistema de requisição/resposta. Usando esses sistemas cada nó faz o seu trabalho individualmente e de forma paralela, possibilitando, por exemplo, o controle dos atuadores e um processamento de navegação autônoma.

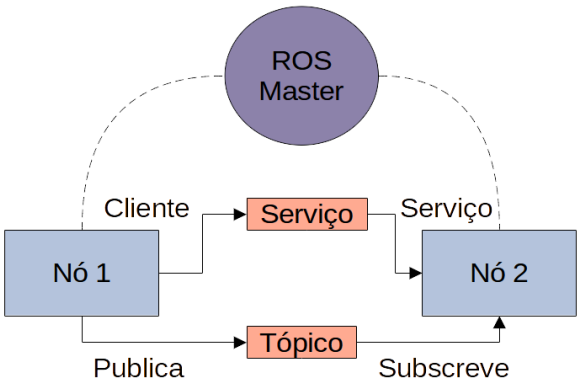


Figura 5 – Diagrama de blocos da comunicação do ROS
Fonte: Adaptado de (JOSEPH, 2018).

Na Figura 5, é apresentado em forma de diagrama de blocos como a comunicação do ROS funciona. O nó principal denominado ROS *master* armazena e gerencia todas as informações de todos os outros nós, tendo o papel de permitir que cada nó individual localize um ao outro, após isso a comunicação é par-a-par.

2.3.2 Gazebo

Para a simulação do robô com o sistema ROS, tem-se a ferramenta Gazebo. Ela oferece um ambiente de simulação que integra a criação de cenários em 3D a um mecanismo de física para a modelagem de sensores e atuadores que se comunicam com o ROS. Um dos pontos fortes do simulador é a sua interface gráfica que oferece praticidades ao criar os cenários, inspecionar os elementos de simulação e interagir com a mesma.

2.3.3 TurtleBot3

O TurtleBot é a plataforma padrão de robôs para o ROS. Sendo o TurtleBot3 a terceira interação dos modelos de robô. A plataforma consiste em modelos de robô diferencial modulares, que podem ser customizados com facilidade. O fator principal para a escolha da plataforma é a integração com o ROS e Gazebo, pois possui uma grande gama de conteúdo disponível na rede. Existem três modelos de robô TurtleBot3, apresentados na Figura 6, o modelo escolhido foi o Burger, pela praticidade e maior facilidade em simular em um ambiente pequeno.

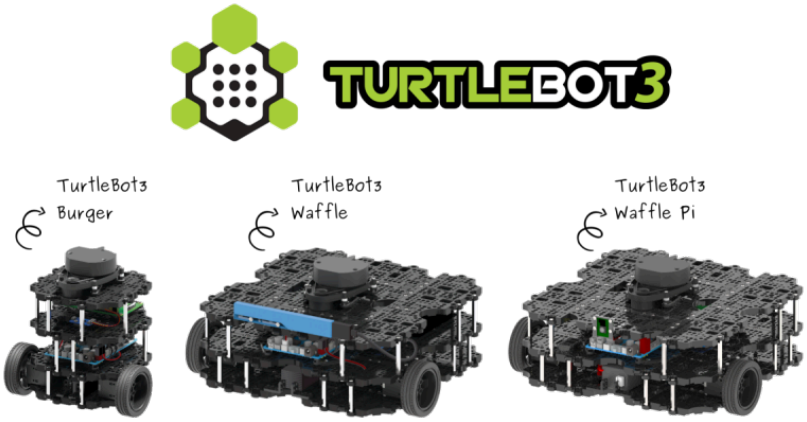


Figura 6 – Modelos TurtleBot3
Fonte: https://www.turtlebot.com/assets/images/turtlebot3_with_logo.png.

2.3.4 TensorFlow e Keras

O TensorFlow é uma plataforma com ferramentas úteis para o aprendizado de máquina. Ele possui uma gama de ferramentas voltadas para o aprendizado por reforço, provendo componentes modulares que facilitam o desenvolvimento e aperfeiçoamento das técnicas.

Para a parte de desenvolvimento do aprendizado profundo existe a API chamada Keras, sendo ela uma API que roda com base no TensorFlow. Ele é utilizado para criar e treinar modelos de aprendizado profundo, facilitando o desenvolvimento do método DQN.

3 MATERIAIS E MÉTODOS

No presente capítulo, são apresentados as tecnologias e ambientes de desenvolvimento utilizados no trabalho, também são definidos os métodos para o treinamento e definição dos hiperparâmetros.

As ferramentas utilizadas para a simulação e construção do algoritmo são as seguintes:

- ROS — conjunto de ferramentas para desenvolvimento de robôs, funcionando como um sistema operacional de código aberto, licença BSD, versão utilizada foi a *Melodic Morenia*.
- TurtleBot3 — robô simples, com modelos de código aberto, com a licença *Apache 2.0*, para a simulação no ambiente do ROS.
- Gazebo — simulador gráfico 3D e de física para robótica de código aberto, com a licença *Apache 2.0*.
- Numpy — biblioteca de código aberto, com a licença BSD 3-Clause, é utilizada para operações matemáticas e manipulação de vetores.
- Keras — API (do inglês, Interface de Programação de Aplicação) de código aberto, licença *Apache 2.0*, é utilizada para construir a rede neural.
- TensorFlow — biblioteca de código aberto, licença *Apache 2.0*, é utilizada como base para o Keras, escolhida a versão específica apenas para CPU, fazendo uso apenas do processador para os cálculos.
- pyqtgraph — utilizado para a construção dos gráficos de resultado, biblioteca de código aberto usando a licença MIT.

Para o processamento da simulação e do algoritmo foi utilizado um computador pessoal, as especificações relevantes do computador são:

- Processador — AMD Ryzen 5 5600G com 6 núcleos reais e 12 *threads*, com clock maximo de 4,4 Ghz.
- Memorias — 2 memorias de 8GB com clock de 3200Mhz.
- Sistema Operacional — Ubuntu 18.04 para a utilização do ROS Melodic.

3.1 TURTLEBOT3 E AMBIENTE DE SIMULAÇÃO

O robô TurtleBot3 Burger possui duas rodas que definem o seu movimento e podem ser controladas utilizando dois parâmetros, sendo eles a velocidade linear de no máximo $0{,}22m/s$ e

Tabela 1 – Ações definidas para o robô

N_a	Ação	Velocidade Angular (rad/s)
0	Esquerda	$-1{,}5$
1	Diagonal Esquerda	$-0{,}75$
2	Frente	0
3	Diagonal Direita	$0{,}75$
4	Direita	$1{,}5$

Fonte: Fonte: Adaptada de (ROBOTIS, 2022)

a velocidade angular de no máximo $2{,}84rad/s$. Para a visualização do estado do ambiente foram utilizadas a odometria do robô, essa que se baseia na posição inicial do robô e na movimentação das rodas e acelerômetro, para obter a informação, e o sensor Lidar (COLLIS, 1970) que faz o uso de laser para medir a distância até um obstaculo. O Lidar utilizado possui uma resolução de 360 medições ao redor do robô com a precisão da medição limitada a distância de $3{,}5m$.

Para o controle do robô no ambiente foi definida a velocidade linear como uma constante de $0{,}15m/s$, permitindo a movimentação do robô sem interferência significativa da perca de tração nas rodas, fornecendo assim uma odometria mais precisa. A velocidade angular é utilizada como as ações que o robô pode tomar, sendo elas apresentadas na Tabela 1.

Os estados do ambiente utilizados para o treinamento e tomada de decisões são 28, dentre eles 24 são leituras oriundas do Lidar. Foi escolhido trabalhar com apenas 24 amostras do Lidar pelo custo do processamento necessário para processar todas as 360 leituras. As 24 leituras são igualmente espaçadas ao entorno do robô, como representado pelas linhas azuis na Figura 7.

Os outros 4 estados são a distância até o objetivo, o angulo para o objetivo, a distância até o obstaculo mais próximo e o angulo para o obstaculo mais próximo.

A equação para calcular a distância até o objetivo D_o é

$$D_o = \sqrt{(o_x - r_x)^2 + (o_y - r_y)^2}, \tag{58}$$

em que o_x e o_y são as coordenadas cartesianas x e y do objetivo com relação ao ponto inicial da simulação, da mesma forma r_x e r_y são as coordenadas x e y do robô.

Para calcular o angulo θ_o com relação ao objetivo são utilizadas as equações

$$\theta_{op} = \arctan\left(\frac{o_y - r_y}{o_x - r_x}\right) - yaw, \tag{59}$$

$$\theta_o = \begin{cases} \theta_{op} - 2 * pi & \theta_{op} > \pi, \\ \theta_{op} + 2 * pi & \theta_{op} < -\pi, \\ \theta_{op} & \pi > \theta_{op} > -\pi, \end{cases} \tag{60}$$

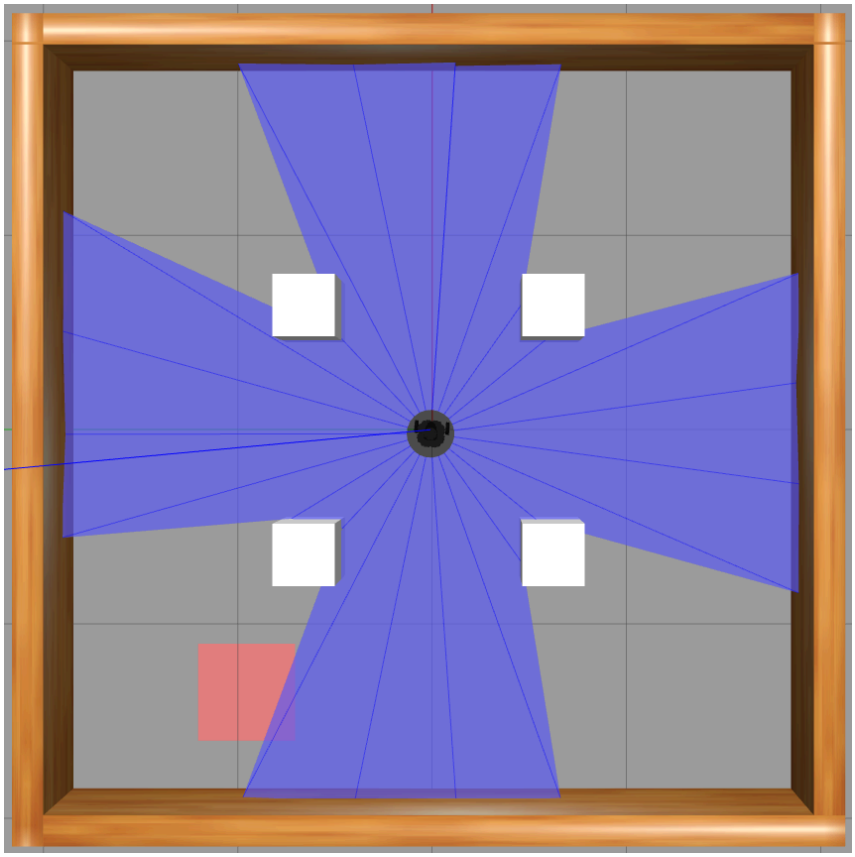


Figura 7 – Ambiente de treinamento com o robô TurtleBot3
Fonte: Autoria Própria.

em que yaw é obtido através do acelerômetro presente no sistema de odometria do robô e representa a guinada do robô com relação aos seus referenciais.

A distância até o obstaculo mais próximo é obtida através da leitura do Lidar, sendo ela o menor valor de leitura obtida naquele estado, de forma semelhante é obtido o ângulo para o obstaculo, sendo esse o índice do menor valor no vetor de leitura do Lidar.

Para a validação dos resultados do algoritmo é proposta a utilização de um ambiente simulado mais próximo à realidade. Para trazer o experimento para a área da logística foi definido o ambiente como sendo uma representação de um armazém, contendo um espaço maior para a navegação e obstáculos diversos. O ambiente utilizado é o *AWS RoboMaker Small Warehouse World* (AWS, 2021), dentre seus obstáculos estão prateleiras, caixas, um lixeiro, uma mesa e uma empilhadeira manual. O ambiente com o robô está na Figura 15.

3.2 UTILIZAÇÃO DA DQN E RECOMPENSAS

O projeto toma como base o algoritmo disponível em ROBOTIS (2018). No algoritmo da DQN é utilizada uma rede neural convolucional para estimar o valor Q, valor de recompensa estimada para cada uma das 5 ações em um determinado estado. A ação com o maior valor Q é tomada para chegar ao próximo estado. Essa rede neural é composta por 5 camadas, sendo

essas:

- Camada de entrada — camada densa com 64 filtros, ativação do tipo *ReLu*, *kernel* inicializado com a função *Lecun Uniform* e entrada com o mesmo tamanho do vetor de estados.
- Camada 2 — camada densa com 64 filtros, ativação *ReLu* e *kernel* inicializado com a *Lecun Uniform*.
- Camada 3 — camada de *dropout*, que passa uma fração de suas entradas como 0 para as saídas, camada utilizada para estabilizar a rede, a razão da fração é definida para 20%.
- Camada 4 — camada densa com 5 filtros, mesmo número que o de ações possíveis para o robô, sem função de ativação e *kernel* inicializado com a *Lecun Uniform*.
- Camada de saída — somente uma camada de ativação linear.

Uma camada densa é uma camada que está profundamente conectada com a camada anterior. Todos os neurônios da rede recebem todas as saídas da camada anterior como entrada (DUMANE, 2020). O número de filtros define o tamanho da saída da sua camada.

As funções de ativação definem qual filtro é aplicado para obter a saída da camada, a função *ReLu* ou Unidade Linear Retificada aplica um filtro $f(x) = max(0,x)$, sendo x a entrada, fazendo assim somente valores positivos passarem (AGARAP, 2018). A ativação linear simplesmente passa o mesmo valor da entrada para a saída.

A inicialização do *kernel Lecun Uniform* inicializa os pesos da rede pegando amostras de uma distribuição linear entre $[-limite,limite]$, dado que $limite = \sqrt{3/fan_{in}}$ em que fan_{in} é o número de filtros de entrada (KERAS, 2022).

O algoritmo DQN utilizado para o treinamento do robô pode ser representado em um diagrama de blocos como apresenta Figura 8.

O treinamento da rede acontece a cada passo do robô, começando a partir do hiperparâmetro de começo de treino, processo exemplificado na Figura 8, esse atraso no início do treinamento ocorre para obter uma quantidade de amostras na memória de repetição maior que a quantidade de amostras utilizadas no treinamento. Essa memória de repetição consiste em um vetor que armazena as tuplas de estado, ação, recompensa, próximo estado e ocorrência de colisão, o armazenamento ocorre a cada passo do robô. No treinamento é utilizada uma amostragem da memória, tendo o número de amostras definido por um hiperparâmetro.

A DQN funciona com a utilização de duas redes neurais, sendo uma atualizada a cada passo e uma *target network* que é atualizada a cada episódio após um número de passos inicial. Essa rede serve como base para estabilizar o aprendizado, como a rede principal está sempre sendo atualizada, ela tende a divergir após muitas iterações, sendo nesse momento que a rede *target* atua para voltar a um modelo mais confiável que já foi aprendido.

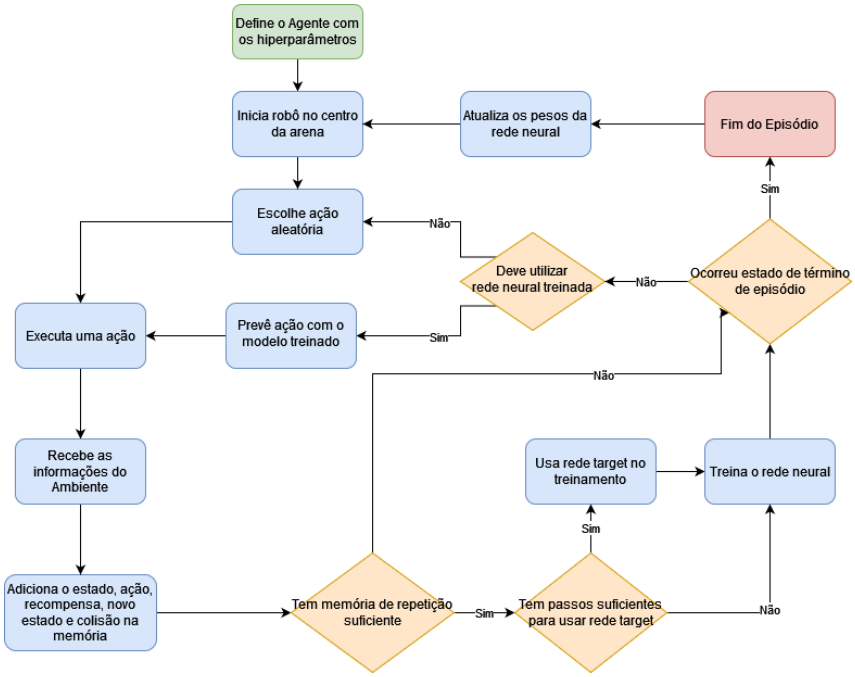


Figura 8 – Diagrama de blocos do fluxo de treinamento da DQN
Fonte: Autoria Própria.

Cada episódio de treinamento começa com o robô no centro da arena e com um objetivo definido de forma aleatória, o objetivo é o quadrado vermelho que pode ser visto na Figura 7. Cada episódio tem dois estados de término, um deles é a detecção de uma colisão com um obstaculo ou por um *timeout* definido em 500 passos. A colisão é detectada ao identificar uma leitura menor que $0{,}15m$ no Lidar. Ao alcançar o objetivo o robô continua no mesmo episodio, apenas tem um novo objetivo aleatório gerado na arena.

São definidos 3 tipos de recompensa para as ações do robô, uma recompensa de 200 pontos para alcançar o objetivo, uma recompensa negativa de -150 pontos por colidir com um obstaculo e recompensa por proximidade e direção com relação ao objetivo, sendo essa definida pela equação,

$$R = yaw_r * 5 * D_r, \tag{61}$$

em que yaw_r é a equação de recompensa pela direção que o robô está apontando em relação ao objetivo, variando também conforme a ação tomada e funcionando como definição para sinal que a recompensa tem, ela é definida por,

$$yaw_r = 1 - 4 * \left| 0{,}5 - \left(0{,}25 + \frac{0{,}5 * \left(\frac{-\pi}{4} + \theta_o + \frac{\pi}{8} * N_a + \frac{\pi}{2}\right) \%(2 * \pi)}{\pi} \right) \%1 \right|, \tag{62}$$

sendo N_a o número correspondente a ação tomada, número esse que está especificado na Tabela 1.

Toda ação recebe uma recompensa e é com base nas recompensas obtidas que a rede neural consegue estimar o valor Q para uma ação em um estado. Cada ação tomada pode ser prevista pela rede neural com base no maior valor Q dentre as ações ou então ser definida aleatoriamente, o que ira decidir como a ação será definida é uma comparação entre um valor aleatório definido a cada passo e o hiperparâmetro ϵ que inicia com um valor pré-determinado e decai com uma taxa ϵ *decay* a cada episódio. A escolha de ações aleatórias faz-se necessária para que o robô obtenha novas experiencias, por esse motivo a cada episódio a probabilidade de escolher uma ação aleatória diminui.

3.3 PARAMETRIZAÇÃO E TREINAMENTO

Os hiperparâmetros utilizados pela DQN e os seus valores base propostos são:

- **episode_step** — número máximo de passos por episódio, valor base 6000.
- **target_update** — passos entre cada atualização do modelo *target*, valor base 2000.
- **discount_factor** — fator com que eventos futuros perdem valor de acordo com a sua distância, valor base $0{,}99$.
- **learning_rate** — é a taxa de aprendizado da rede neural e define quanto mudar o modelo em resposta a taxa de erro estimada cada vez que os pesos são atualizados (BROWNLEE, 2019), valor base $0{,}00025$.
- **epsilon** — probabilidade ϵ de escolher uma ação aleatória, valor base $1{,}0$.
- **epsilon_decay** — taxa de decaimento do valor ϵ para cada episódio, valor base $0{,}99$.
- **epsilon_min** — valor mínimo permitido para ϵ após o decaimento, valor base de $0{,}1$.
- **batch_size** — tamanho das amostras de memória utilizadas para o treinamento, valor base 64.
- **train_start** — passos após o início que o treinamento começa a acontecer, valor base 64.
- **memory** — tamanho da memória de repetição que armazena as amostras, valor base 1000000.

4 RESULTADOS

Neste capitulo são apresentados os resultados obtidos com o uso do DQN para guiar o robô automaticamente pelo ambiente.

4.1 AMBIENTE DE ARENA COM OBSTÁCULOS

O ambiente utilizado para o treinamento do robô foi criado utilizando o simulador Gazebo, contendo uma arena simples com 4 paredes formando um quadrado com $4m$ de comprimento e 4 obstáculos cúbicos com laterais de $0{,}30m$ posicionados nas cordenadas $x, y = \{(0.6, 0.6), (0.6, -0.6), (-0.6, 0.6), (-0.6, -0.6)\}$. A Figura 7 descreve a visualização gráfica do ambiente com o robô posicionado no seu local de início, os traços e coloração azul são a representação visual da leitura do Lidar no estado atual.

4.1.1 Ensaio 1 - Hiperparâmetros Base

A partir dos parâmetros bases, seção 3.3, foram feitas modificações nos hiperparâmetros e realizadas simulações. Cada simulação tem 400 episódios e obteve-se os resultados de pontuação geral e a média do valor máximo de Q para cada episódio. A cada modificação de hiperparâmetro são restaurados os bases e apenas o hiperparâmetro em questão é alterado.

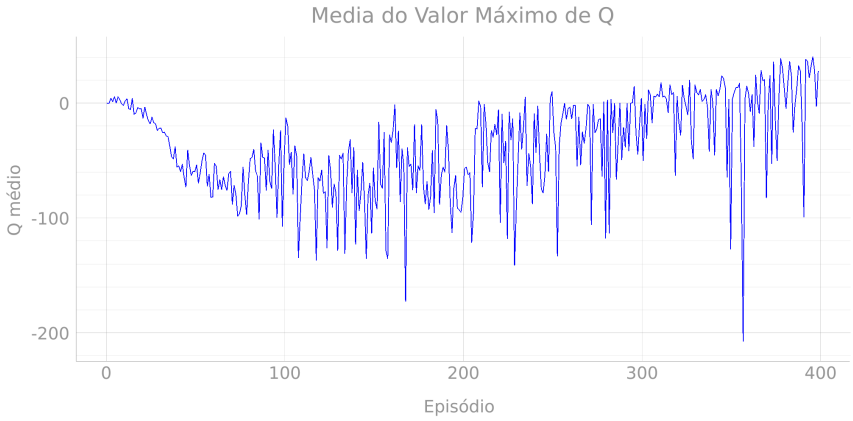
Os resultados com os hiperparâmetros base pode ser vistos na Figura 9. Analisando a Figura 9(a) é possível identificar que a partir do episódio ~ 250 a soma das recompensas por episódio se torna majoritariamente positiva, indicando que o robô está tomando decisões que o levam em direção ao objetivo final. Os picos de grande de pontuação, com recompensa maior que 1000, mostram episódios em que o robô chegou ao objetivo múltiplas vezes, sendo os 3 maiores picos episódios que terminaram em *timeout*. Boa parte do aumento na pontuação se dá pelo fato do robô conseguir desviar com maior consistência dos obstáculos entre ele e o objetivo.

Já na Figura 9(b) vemos o crescimento da média do valor máximo de Q para cada passo, ou seja, das estimativas de recompensa dentro de um episódio. Na fase inicial, até o episódio ~ 70, o valor Q decresce, indicando muita escolha aleatória pelo valor ϵ ainda estar grande e também pela rede neural não ter sido treinada o suficiente. Após isso temos uma zona de crescimento positivo, conseguindo chegar a valores positivos de média a partir do episódio 300, resultando em tomadas de decisão mais estáveis.

4.1.2 Ensaio 2 - modificação no $target\_update$ de 2000 passos para 1000 passos

Os resultados são apresentados na Figura 10

(a)



(b)

Figura 9 – Resultados hiperparâmetros base
Fonte: Autoria Própria.

Na Figura 10(a) é perceptível uma convergência para recompensas acumuladas majoritariamente maiores que 0 em torno do episódio ~ 180, se comparado aos resultados base, ~ 70 episódios mais cedo. Porém, apos essa convergência ficou estagnado sem ter crescimentos significativos na sua consistência em alcançar múltiplos objetivos consecutivamente.

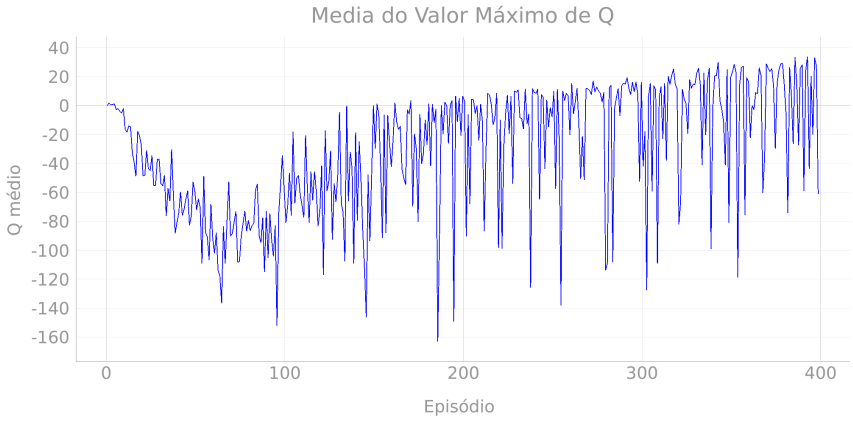
Ao observar a Figura 10(b) temos a mesma queda inicial que com os hiperparâmetros base, mas na sequência temos o crescimento constante a partir do episódio 100, resultando em médias de Q positivas a partir do episódio ~ 220. Ambos resultados mostram um aprendizado mais rápido, provenientes da utilização mais frequente da rede *target* estabilizando o aprendizado, entretanto ocorre uma redução nos novos aprendizados por limitar a rede neural principal de seguir novos caminhos diferentes.

4.1.3 Ensaio 3 - modificação no $discount\_factor$ de $0{,}99$ para $0{,}95$

Os resultados para o hiperparâmetro são vistos na Figura 11.

Com a análise da Figura 11(a) é possível identificar um início com picos negativos de menor magnitude, também temos a estabilização em números positivos acontecendo no episó-

(a)



(b)

Figura 10 – Resultados para alteração no $target\_update$
Fonte: Autoria Própria.

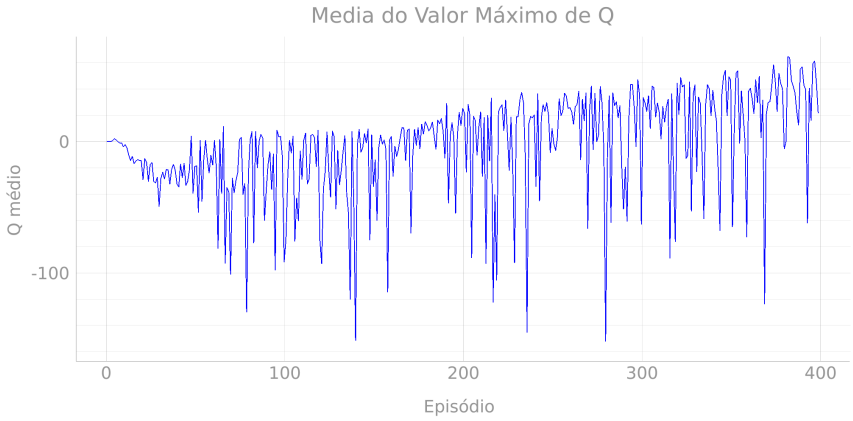
dio ~ 160, sendo ~ 90 episódios antes dos resultados base. Junto a isso temos uma consistência maior em alcançar objetivos consecutivos. Com a utilização de uma taxa de desconto menor é dada maior relevância as memórias mais recentes, fazendo com que memorias antigas com maior aleatoriedade tenham menos valor nas estimativas, obtendo assim resultados mais constantes.

No gráfico de média do valor máximo de Q, Figura 11(b), a análise é semelhante a dos resultados, em que o início até o episódio $\sim$ 70 temos uma decrescente menor que se comparado a base. Logo após isso começa o crescimento da média regularmente, obtendo picos mais elevados que com os hiperparâmetros base.

4.1.4 Ensaio 4 - modificação no $learning\_rate$ de $0{,}00025$ para $0{,}0004$

O hiperparâmetro $learning\_rate$ impacta na velocidade do aprendizado, com valores muito baixos pode resultar em um tempo de treinamento muito longo e ficar preso. Com valores muito grandes define pesos não ótimos muito rapidamente podendo fazer a rede ficar instável (BROWNLEE, 2019). Os resultados estão na Figura 12.

(a)



(b)

Figura 11 – Resultados para alteração no $discount\_factor$
Fonte: Autoria Própria.

Nessa simulação vemos uma queda no desempenho do algoritmo, visível na Figura 12(a), o treinamento apresenta pequenos focos de resultados positivos como o que começa ao entorno do episódio $\sim$ 300, porem apos um período volta a cair para níveis negativos. Esse padrão de instabilidade é proveniente de um valor muito grande para o hiperparâmetro.

A Figura 12(b) apresenta resultados que destoam dos resultados base apresentando muitos altos e baixos na média, causando assim os resultados vistos anteriormente.

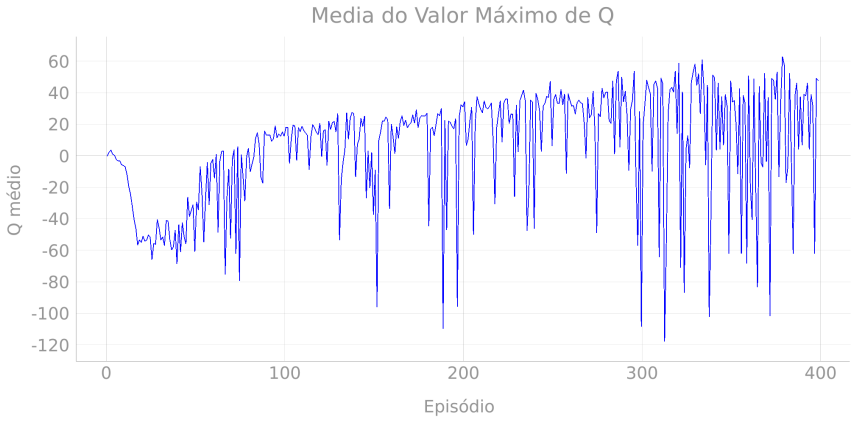
4.1.5 Ensaio 5 - modificação no $epsilon\_decay$ de 0,99 para 0,95

A Figura 13 contém os resultados obtidos para o hiperparâmetro.

Os resultados de recompensa total da Figura 13(a) mostram um aprendizado mais veloz que o resultado base, tendo recompensas majoritariamente positivas após o episódio ~ 150.

Os resultados mais interessantes dessa simulação estão no gráfico de valor médio de Q máximo, presentes na Figura 13(b), em que existe uma queda brusca nos valores até o episódio ~ 30 e em seguida uma crescente gradual até valores maiores que 0. O comportamento analisado vem do fato de $\epsilon = epsilon\_min$ acontecer no episódio 45, com os hiperparâmetros base

(a)



(b)

Figura 12 – Resultados para alteração no $learning\_rate$
Fonte: Autoria Própria.

o valor é atingido no episódio 90, causando assim uma confiança na rede que ainda não está suficientemente treinada.

4.1.6 Ensaio 6 - modificação no $batch\_size$ de 64 amostras para 32

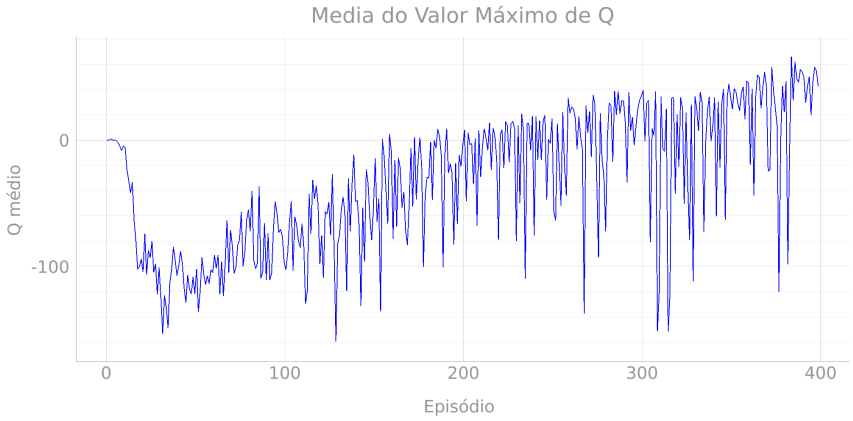
Na Figura 14 são apresentados os gráficos de resultado.

Com a alteração do hiperparâmetro foram obtidos resultados inconsistentes, como visto na Figura 14, mesmo após chegar em um estado de recompensas majoritariamente positivas ainda ocorrem focos de pouca recompensa a cada poucos episódios. Isso se da porque ao utilizar a metade das amostras para cada treinamento, não há informações o suficiente para a rede encontrar o melhor valor Q.

O mesmo caso reflete no gráfico da Figura 14, em que existem diversos pontos em que a rede não consegue estimar uma boa ação para cada passo.

Os hiperparâmetros restantes não causaram mudanças significativas ou relevantes para o treinamento, por esse motivo os seus resultados não são apresentados.

(a)



(b)

Figura 13 – Resultados para alteração no $epsilon\_decay$
Fonte: Autoria Própria.

4.2 AMBIENTE DE ARMAZÉM

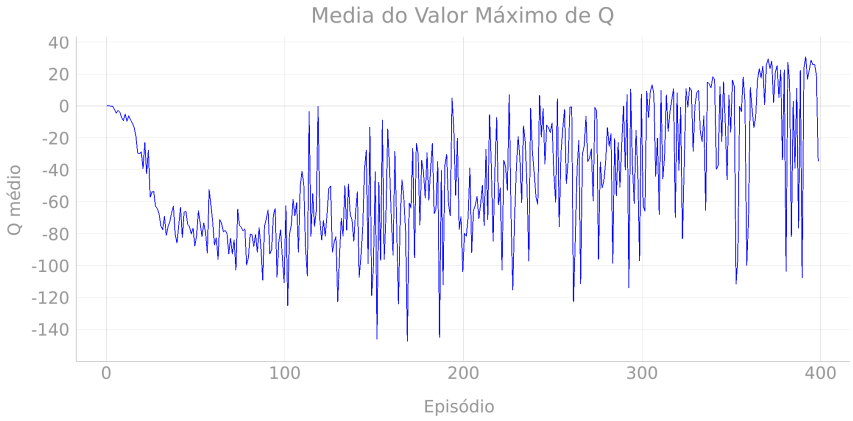
Após o treinamento no ambiente simples é trazido o algoritmo para o ambiente do armazém, sendo uma maneira de validar as características do algoritmo. O ambiente utilizado pode ser visto na Figura 15, com a posição vista do robô sendo nas coordenadas $[0,0]$.

A primeira simulação foi feita com um modelo de rede neural treinado no ambiente da Figura 7, o treinamento foi desabilitado deixando as decisões serem tomadas pela rede sem alterar seus pesos constantemente. Os hiperparâmetros utilizados no treinamento foram definidos com a análise dos resultados obtidos na seção 4.1, os seus valores estão na Tabela 2.

A simulação foi feita com o modelo de rede neural do episódio 400, e os resultados estão na Figura 16. São apenas 23 episódios de resultados, devido ao longo tempo que cada episódio pode levar em decorrência do tamanho do ambiente. O estado de término por *timeout* foi removido, para que não ocorressem interrupções no meio de um episódio longo.

O gráfico da Figura 16(a) mostra que mesmo em um ambiente novo o algoritmo treinado consegue obter resultados expressivos. Existe uma grande diferença na pontuação entre episódios, semelhante aos resultados obtidos com o ambiente pequeno, em que o robô consegue

(a)



(b)

Figura 14 – Resultados para alteração no $batch\_size$
Fonte: Autoria Própria.

Tabela 2 – Hiperparâmetros de treinamento

Hiperparâmetro	Valor
$episode\_step$	6000
$target\_update$	2000
$discount\_factor$	0,95
$learning\_rate$	0,00025
$epsilon$	1,0
$epsilon\_decay$	0,95
$epsilon\_min$	0,1
$batch\_size$	64
$train\_start$	64
$memory$	1000000
$episodes$	400

Fonte: Fonte: Autoria Própria.

chegar ao objetivo com certa consistência, porém não consegue evitar todos os obstáculos.

Na Figura 16(b) existe uma grande diferença se comparado aos resultados de recompensa, nesta é visível um número grande de estimativas Q negativos. O comportamento verificado se dá pelo tamanho do ambiente e quantidade de obstáculos, cada passo conta pouco para a recompensa final pela sua distância do objetivo.

A segunda simulação com o ambiente tem o foco na análise do treinamento do algoritmo

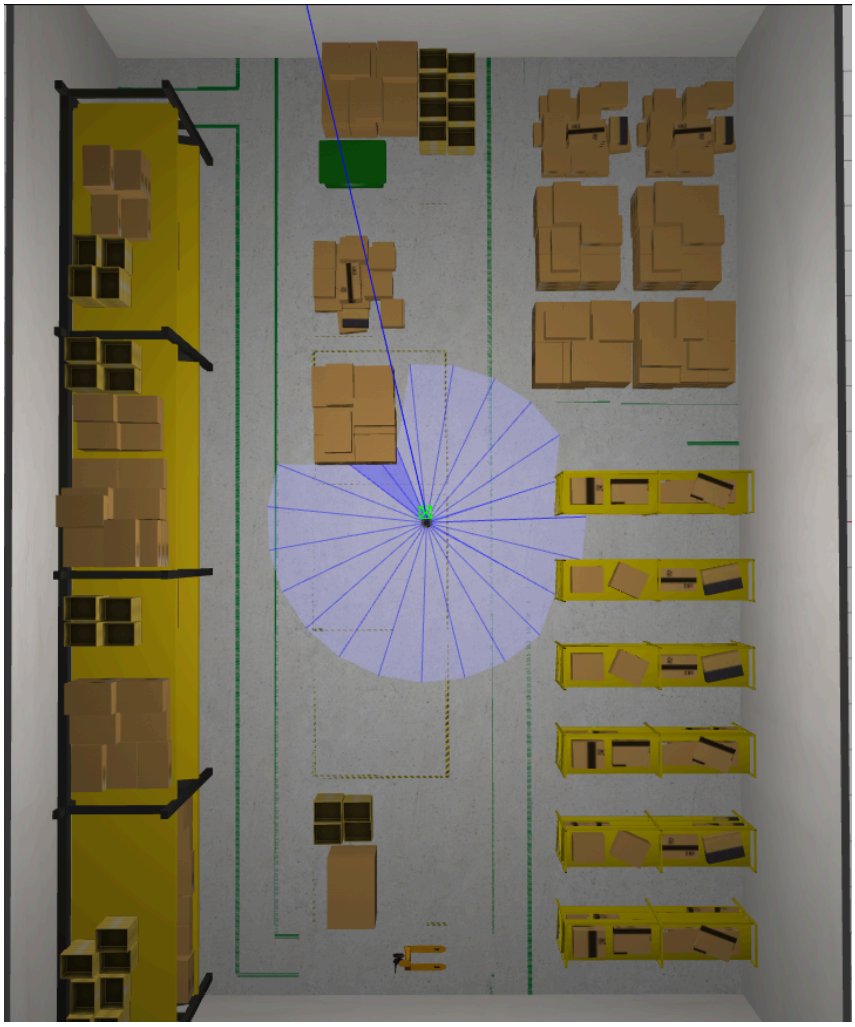


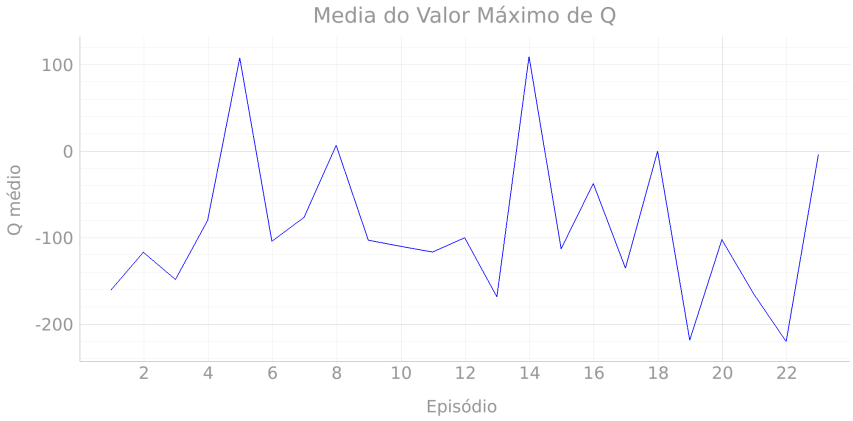
Figura 15 – Visão vertical do armazém com o robô
Fonte: Autoria Própria.

DQN em ambientes complexos. Para esse experimento foram definidos os mesmos hiperparâmetros da Tabela 2, porém desta vez foi utilizada uma rede nova e habilitado novamente o treinamento da mesma. Para que o robô não fique preso em um episódio foi novamente habilitado o estado de término de *timeout*, sendo esse definido para 2000 passos pelo tamanho do ambiente. Os resultados obtidos podem ser vistos na Figura 17.

Ao analisar a Figura 17(a), pode-se concluir que o robô não chegou a um estado de aprendizado satisfatório no período de 400 episódios. Ele manteve-se com recompensas positivas, porém raramente alcançou o objetivo. O comportamento do treinamento vem do fato de ser um ambiente grande demais para o robô, fazendo com que ele conseguisse recompensas positivas sem necessariamente alcançar o objetivo.

Nos resultados de Q, Figura 17(b), é perceptível o mesmo comportamento dos resultados, mesmo com valores bons nunca melhorou nas estimativas. Uma maneira para que o treinamento funcionasse corretamente seria ajustar os hiperparâmetros e recompensas para condizerem com o tamanho do ambiente. Com a alteração de ambas existiria ganhos conforme os episódios evoluem, porém, seria necessário um número maior de episódios para chegar a

(a)



(b)

Figura 16 – Resultados para o Ambiente complexo com a rede neural treinada
Fonte: Autoria Própria.

resultados semelhantes aos do ambiente simples.

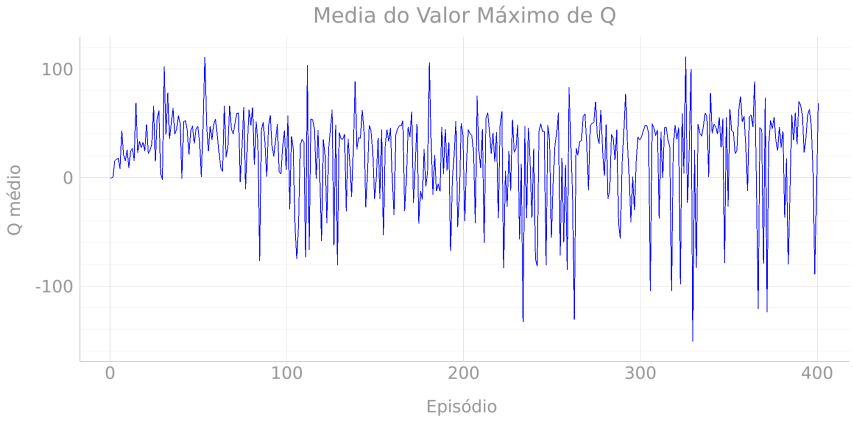
4.3 DISCUSSÕES

Uma característica do algoritmo treinado é a sua adaptabilidade, após o treinamento em um ambiente simples a rede consegue transpor o seu conhecimento para outros ambientes que tenha dificuldades similares, mesmo que completamente diferentes. Tendo em mente a necessidade de mais episódios de treinamento para o treinamento em um ambiente grande, é possível acelerar o processo com a utilização de um ambiente pequeno, mas que represente o todo.

Um dos problemas verificados durante o treinamento do algoritmo é o fato de ele nunca conseguir convergir para um estado em que não haja colisões durante um episódio. No *Q-learning* e DQN, o algoritmo usa o mesmo valor para selecionar e estimar uma ação, o que pode levar a uma superestimação dos valores de Q (WANG *et al.*, 2015).

Uma melhoria que poderia ser feita no algoritmo é no sistema de recompensas, no formato atual o robô continua ganhando recompensa quando está indo em direção a um obstaculo,

(a)



(b)

Figura 17 – Resultados para o Ambiente complexo com a rede neural treinada
Fonte: Autoria Própria.

se esse obstaculo estiver na mesma direção que o objetivo. Uma possível alternativa seria adicionar mais um tipo de recompensa, que retorne pontuação negativa quando o robô tomar a decisão de ir em direção ao obstaculo, desencorajando o robô de seguir diretamente para o objetivo.

Com relação ao uso em aplicações reais, existe o potencial da utilização do algoritmo para esse fim, porém seria necessário a otimização dos pontos citados acima e possivelmente um sistema mais confiável para detecção de colisões diretas. Dito isso existem outras implementações que ampliam sobre a DQN, como é o algoritmo utilizado por (WANG *et al.*, 2015) ou mais especificamente para AGVs o algoritmo proposto por (GUO *et al.*, 2020), ambos fazem o uso da *Double Deep Q-network* que melhora a convergência do algoritmo base.

5 CONCLUSÃO

O aprendizado por reforço é uma área com grande potencial na robótica móvel, tendo como diferencial a sua adaptabilidade para diversos ambientes. Isso vem do fato de não ser necessário o conhecimento prévio do ambiente para o aprendizado, funcionalidade muito útil na robótica móvel, devido à grande diversidade de aplicações na área.

Algo que impacta drasticamente no aprendizado para cada ambiente são os seus hiperparâmetros, conforme o tamanho e obstáculos encontrados é necessário o ajuste fino para que o algoritmo mantenha o desempenho. Como os fatores que impactam o resultado são muitos, é necessária a experimentação para chegar aos melhores hiperparâmetros. Com os hiperparâmetros errados é possível fazer com que o robô fique preso em um ciclo que não o deixa com pontuação negativa, porém também não fornece nenhum aprendizado relevante.

Os resultados obtidos com os experimentos apresentam diversos pontos de melhoria, pode-se citar o ajuste do sistema de recompensa, o refinamento dos valores para o treinamento e melhorias no algoritmo utilizado. Em diversos episódios o robô conseguiu alcançar objetivos em sequência, demonstrando a capacidade de aprendizado, em contrapartida, a rede tende a não estabilizar completamente, tendo queda de desempenho entre episódios distintos. Outro ponto a ser ressaltado, é que a quantidade de estados influencia no aprendizado todo, dificultando a convergência do algoritmo quando forem muitos estados para serem processados e também muito poucos para que o agente entenda o ambiente.

O algoritmo proposto para o controle do robô móvel teve o seu funcionamento testado, obtendo resultados promissores, que mostra a viabilidade do mesmo para essa função. Foram levantados diversos pontos de melhoria e verificado como cada hiperparâmetro age no algoritmo. O ambiente de simulação permitiu o treinamento e teste do robô de forma análoga à realidade, sem as dificuldades que surgem com a utilização de um agente em um ambiente real.

5.1 TRABALHOS FUTUROS

Para trabalhos futuros planeja-se melhorar o sistema de recompensas utilizado, otimizar os hiperparâmetros da rede e utilizar algoritmos que expandem sobre o conceito do DQN cobrindo os problemas existentes nele. Uma proposta interessante é levar o robô para o mundo real fornecendo novos desafios. Para a utilização em futuros trabalhos, foram disponibilizados em Welter (2022) o material e o algoritmo implementados no desenvolvimento do trabalho de conclusão de curso apresentado.

REFERÊNCIAS

AGARAP, Abien Fred. Deep learning using rectified linear units (relu). **CoRR**, abs/1803.08375, 2018. Disponível em: http://arxiv.org/abs/1803.08375. Citado na página 40.

AWS. **AWS, RoboMaker Small Warehouse World**. 2021. Accessed: 2022-11-30. Disponível em: https://github.com/aws-robotics/aws-robomaker-small-warehouse-world. Citado na página 39.

BROWNLEE, Jason. **Understand the Impact of Learning Rate on Neural Network Performance**. 2019. Accessed: 2022-11-30. Disponível em: https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/. Citado 2 vezes nas páginas 42 e 45.

CHEN, Cheng *et al.* The adoption of self-driving delivery robots in last mile logistics. **Transportation research part E: logistics and transportation review**, Elsevier, v. 146, p. 102214, 2021. Citado na página 11.

COLLIS, R. T. H. Lidar. **Appl. Opt.**, Optica Publishing Group, v. 9, n. 8, p. 1782–1788, Aug 1970. Disponível em: https://opg.optica.org/ao/abstract.cfm?URI=ao-9-8-1782. Citado na página 38.

DUMANE, Govinda. **Introduction to Convolutional Neural Network (CNN) using Tensorflow**. 2020. Accessed: 2022-11-30. Disponível em: https://towardsdatascience.com/introduction-to-convolutional-neural-network-cnn-de73f69c5b83. Citado na página 40.

DURRANT-WHYTE, Hugh; BAILEY, Tim. Simultaneous localization and mapping: part i. **IEEE robotics & automation magazine**, IEEE, v. 13, n. 2, p. 99–110, 2006. Citado 2 vezes nas páginas 12 e 13.

ECKHARDT, Roger; ULAM, Stan; NEUMANN, Jhon Von. the monte carlo method. **Los Alamos Science**, v. 15, p. 131, 1987. Citado na página 29.

FEDORKO, Gabriel; HONUS, Stanislav; SALAI, Roland. Comparison of the traditional and autonomous agv systems. In: EDP SCIENCES. **MATEC Web of Conferences**. [S.l.], 2017. v. 134, p. 00013. Citado na página 13.

GUIZZO, Eric. Three engineers, hundreds of robots, one warehouse. **IEEE spectrum**, IEEE, v. 45, n. 7, p. 26–34, 2008. Citado na página 11.

GUO, Xinde *et al.* A deep reinforcement learning based approach for agvs path planning. In: IEEE. **2020 Chinese Automation Congress (CAC)**. [S.l.], 2020. p. 6833–6838. Citado 3 vezes nas páginas 13, 34 e 52.

JOSEPH, Lentin. **Robot Operating System (ROS) for Absolute Beginners**. [S.l.]: Springer, 2018. Citado 2 vezes nas páginas 5 e 35.

KERAS. **tf.keras.initializers.LecunUniform | TensorFlow v2.11.0**. 2022. Accessed: 2022-11-30. Disponível em: https://www.tensorflow.org/api_docs/python/tf/keras/initializers/LecunUniform. Citado na página 40.

KHATIB, Oussama. Real-time obstacle avoidance for manipulators and mobile robots. In: IEEE. **Proceedings. 1985 IEEE International Conference on Robotics and Automation**. [S.l.], 1985. v. 2, p. 500–505. Citado na página 13.

KLANCAR, Gregor *et al.* **Wheeled mobile robotics: from fundamentals towards autonomous systems**. [S.l.]: Butterworth-Heinemann, 2017. Citado 4 vezes nas páginas 5, 15, 16 e 18.

KOSTER, René De; LE-DUC, Tho; ROODBERGEN, Kees Jan. Design and control of warehouse order picking: A literature review. **European journal of operational research**, Elsevier, v. 182, n. 2, p. 481–501, 2007. Citado na página 11.

LIMA, Jeferson José. **Mobile Robots - Robot Dynamics**. 2021. Accessed: 2021-12-08. Disponível em: https://cursoseaulas.gitlab.io/robotica-movel/dinamica-aula.pdf. Citado 2 vezes nas páginas 5 e 19.

MNIH, Volodymyr *et al.* Playing atari with deep reinforcement learning. **arXiv preprint arXiv:1312.5602**, 2013. Citado 2 vezes nas páginas 33 e 34.

____. Human-level control through deep reinforcement learning. **nature**, Nature Publishing Group, v. 518, n. 7540, p. 529–533, 2015. Citado 3 vezes nas páginas 14, 33 e 34.

PELLEGRINI, Jerônimo; WAINER, Jacques. Processos de decisão de markov: um tutorial. **Revista de Informática Teórica e Aplicada**, v. 14, n. 2, p. 133–179, 2007. Citado na página 25.

PUTERMAN, Martin L. Markov decision processes. **Handbooks in operations research and management science**, Elsevier, v. 2, p. 331–434, 1990. Citado na página 25.

ROBOTIS. **TurtleBot3 Machine Learning**. 2018. Accessed: 2022-12-01. Disponível em: https://github.com/ROBOTIS-GIT/turtlebot3_machine_learning. Citado na página 39.

____. **ROBOTIS e-Manual Machine Learning**. 2022. Accessed: 2022-12-01. Disponível em: https://emanual.robotis.com/docs/en/platform/turtlebot3/machine_learning/. Citado na página 38.

ROS / Introduction. 2021. Accessed: 2021-12-13. Disponível em: http://web.archive.org/web/20080207010024/http://www.808multimedia.com/winnt/kernel.htm. Citado na página 35.

RUMMERY, Gavin A; NIRANJAN, Mahesan. **On-line Q-learning using connectionist systems**. [S.l.]: Citeseer, 1994. v. 37. Citado na página 33.

SCHULZE, Lothar; WULLNER, Alexander. The approach of automated guided vehicle systems. In: IEEE. **2006 IEEE international conference on service operations and logistics, and informatics**. [S.l.], 2006. p. 522–527. Citado na página 11.

SIEGWART, Roland; NOURBAKHSH, Illah Reza; SCARAMUZZA, Davide. **Introduction to autonomous mobile robots**. [S.l.]: MIT press, 2011. Citado 4 vezes nas páginas 5, 15, 18 e 19.

SILVER, David. Lecture 2: Markov decision processes. **UCL. Retrieved from www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/MDP.pdf**, 2015. Citado na página 27.

____. Lecture 3: Planning by dynamic programming. **Google DeepMind**, 2015. Citado na página 28.

SONG, Rui; LIU, Yuanchang; BUCKNALL, Richard. Smoothed a* algorithm for practical unmanned surface vehicle path planning. **Applied Ocean Research**, Elsevier, v. 83, p. 9–20, 2019. Citado na página 13.

SUTTON, Richard S. Learning to predict by the methods of temporal differences. **Machine learning**, Springer, v. 3, n. 1, p. 9–44, 1988. Citado na página 31.

SUTTON, Richard S; BARTO, Andrew G. **Reinforcement learning: An introduction**. [S.l.]: MIT press, 2018. Citado 10 vezes nas páginas 23, 24, 25, 27, 28, 29, 30, 31, 32 e 33.

SUTTON, Richard S; TANNER, Brian. Temporal-difference networks. **Advances in neural information processing systems**, v. 17, 2004. Citado na página 31.

TAI, Lei; PAOLO, Giuseppe; LIU, Ming. Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. In: IEEE. **2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)**. [S.l.], 2017. p. 31–36. Citado na página 34.

WALLÉN, Johanna. **The history of the industrial robot**. [S.l.]: Linköping University Electronic Press, 2008. Citado na página 11.

WANG, Lanfei *et al.* Improved ant colony optimization for ground robot 3d path planning. In: IEEE. **2018 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)**. [S.l.], 2018. p. 106–1066. Citado na página 13.

WANG, Ziyu *et al.* **Dueling Network Architectures for Deep Reinforcement Learning**. arXiv, 2015. Disponível em: https://arxiv.org/abs/1511.06581. Citado 2 vezes nas páginas 51 e 52.

WATKINS, Christopher JCH; DAYAN, Peter. Q-learning. **Machine learning**, Springer, v. 8, n. 3-4, p. 279–292, 1992. Citado na página 32.

WATKINS, Christopher John Cornish Hellaby. Learning from delayed rewards. King's College, Cambridge United Kingdom, 1989. Citado 2 vezes nas páginas 14 e 32.

WELTER, Alexsandro Reichert. **TurtleBot3 DQN**. 2022. Accessed: 2022-12-01. Disponível em: https://github.com/Reicherd/turtlebot3_dqn. Citado na página 53.

WIERING, Marco A; OTTERLO, Martijn Van. Reinforcement learning. **Adaptation, learning, and optimization**, Springer, v. 12, n. 3, 2012. Citado 3 vezes nas páginas 23, 24 e 26.

XUE, Xidi *et al.* A deep reinforcement learning method for mobile robot collision avoidance based on double dqn. In: IEEE. **2019 IEEE 28th International Symposium on Industrial Electronics (ISIE)**. [S.l.], 2019. p. 2131–2136. Citado na página 12.

YANG, Yang; JUNTAO, Li; LINGLING, Peng. Multi-robot path planning based on a deep reinforcement learning dqn algorithm. **CAAI Transactions on Intelligence Technology**, IET, v. 5, n. 3, p. 177–183, 2020. Citado na página 12.

ZHANG, Jingwei *et al.* Deep reinforcement learning with successor features for navigation across similar environments. In: IEEE. **2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)**. [S.l.], 2017. p. 2371–2378. Citado 3 vezes nas páginas 12, 13 e 34.

ZHAO, Dongbin *et al.* Deep reinforcement learning with experience replay based on sarsa. In: IEEE. **2016 IEEE Symposium Series on Computational Intelligence (SSCI)**. [S.l.], 2016. p. 1–6. Citado na página 33.

ZHOU, Siyu *et al.* A deep q-network (dqn) based path planning method for mobile robots. In: IEEE. **2018 IEEE International Conference on Information and Automation (ICIA)**. [S.l.], 2018. p. 366–371. Citado na página 12.