UNIVERSIDADE FEDERAL DE PELOTAS
Centro de Desenvolvimento Tecnológico
Programa de Pós-Graduação em Computação

Tese

Navegação Autônoma em Ambientes Dinâmicos com Interação com Humanos baseada em Aprendizado por Reforço Profundo e Visão Computacional

Paulo de Almeida Afonso

Pelotas, 2023

Paulo de Almeida Afonso

Navegação Autônoma em Ambientes Dinâmicos com Interação com Humanos baseada em Aprendizado por Reforço Profundo e Visão Computacional

Tese apresentada ao Programa de Pós-Graduação em Computação do Centro de Desenvolvimento Tecnológico da Universidade Federal de Pelotas, como requisito parcial à obtenção do título de Doutor em Ciência da Computação.

Orientador: Prof. Dr. Paulo Roberto Ferreira Jr.

Pelotas, 2023

Universidade Federal de Pelotas / Sistema de Bibliotecas
Catalogação na Publicação

A111n Afonso, Paulo de Almeida

Navegação autônoma em ambientes dinâmicos com interação com humanos baseada em aprendizado por reforço profundo e visão computacional / Paulo de Almeida Afonso ; Paulo Roberto Ferreira Jr., orientador. — Pelotas, 2023.

103 f.

Tese (Doutorado) — Programa de Pós-Graduação em Computação, Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, 2023.

1. Robôs móveis. 2. Navegação autônoma. 3. Ambientes lotados. 4. Prevenção de colisões. I. Jr., Paulo Roberto Ferreira, orient. II. Título.

CDD : 004

Elaborada por Patricia de Borba Pereira CRB: 10/1487

Paulo de Almeida Afonso

Navegação Autônoma em Ambientes Dinâmicos com Interação com Humanos baseada em Aprendizado por Reforço Profundo e Visão Computacional

Tese aprovada, como requisito parcial, para obtenção do grau de Doutor em Ciência da Computação, Programa de Pós-Graduação em Computação, Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas.

Data da Defesa: 01 de Setembro de 2023

Banca Examinadora:

Prof. Dr. Paulo Roberto Ferreira Jr. (orientador)
Doutor em Computação pela Universidade Federal do Rio Grande do Sul.

Prof. Dr. Felipe de Souza Marques
Doutor em Computação pela Universidade Federal do Rio Grande do Sul.

Prof. Dr. Plinio Thomaz Aquino Junior.
Doutor em Engenharia Elétrica pela Escola Politécnica da Universidade de São Paulo.

Prof. Dr. João Alberto Fabro
Doutor em Engenharia Elétrica e Informática Industrial pela Universidade Tecnológica Federal do Paraná.

Dedico esta tese à minha esposa, Karem Mata Álvares, e ao meu filho, Gabriel Álvares Afonso.

AGRADECIMENTOS

Agradeço ao meu orientador, Prof. Dr. Paulo Roberto Ferreira Jr., que me orientou também no mestrado e com quem tenho o prazer de compartilhar conhecimento desde meu ingresso na Universidade Federal de Pelotas. Sou muito grato pela ajuda nesta jornada acadêmica e espero ter correspondido às expectativas em mim depositadas ao longo desses anos.

Agradeço à minha esposa Karem e ao meu filho Gabriel, pelo companheirismo e pela paciência e compreensão durante este período, especialmente ao meu filho, pelo tempo que não pude dedicar a ele durante o doutorado.

Por fim, sou especialmente grato à minha sogra Corintha, meu sogro Renato e minha cunhada Liege que, juntamente com minha esposa, sempre me apoiaram e incentivaram para que eu seguisse em frente e obtivesse sucesso nessa jornada.

RESUMO

AFONSO, Paulo de Almeida. **Navegação Autônoma em Ambientes Dinâmicos com Interação com Humanos baseada em Aprendizado por Reforço Profundo e Visão Computacional**. Orientador: Paulo Roberto Ferreira Jr.. 2023. 103 f. Tese (Doutorado em Ciência da Computação) – Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, Pelotas, 2023.

Pesquisas recentes demonstram que a navegação autônoma em ambientes dinâmicos, compartilhados com humanos, permanece como um problema em aberto. Nesse tipo de ambiente a movimentação das pessoas pode gerar obstruções, dificultando o sensoriamento e prejudicando a percepção do robô em relação à sua posição. Além disso, a incerteza do comportamento humano pode levar a situações inseguras para o robô e para as pessoas em seu entorno. Frente a esse cenário, tem se destacado o estudo de métodos baseados em aprendizado, assim como a implementação de estruturas híbridas, combinando diferentes arquiteturas e algoritmos, em busca de uma solução eficiente. Este trabalho propõe a combinação de técnicas de aprendizado por reforço profundo com técnicas de visão computacional para o desenvolvimento de uma solução capaz de permitir que o robô navegue de forma autônoma e segura em ambientes internos compartilhados com humanos, considerando as características particularmente associadas ao problema em questão. Assim, a navegação deve considerar questões de segurança, como o distanciamento entre o robô e as pessoas. Para isso, foi desenvolvida uma abordagem inédita, baseada em aprendizado por reforço profundo, que utiliza o algoritmo Deep Deterministic Policy Gradient (DDPG), combinado com técnicas de visão computacional. Foram conduzidos testes comparativos entre os algoritmos DDPG e Deep Q-Network (DQN), abordando quatro etapas, cada uma representando dois cenários diferentes do ambiente de treinamento e com níveis de complexidade superiores ao que o robô foi treinado. O DDPG demonstrou ser mais eficiente e estável que o DQN, com taxas médias de sucesso superiores em todas as etapas, demonstrando melhor capacidade de generalização e apresentando resultados consistentemente melhores. Por outro lado, o DQN teve dificuldades em evitar colisões e obteve taxas médias de sucesso significativamente mais baixas. Essas descobertas destacam a superioridade do DDPG e demonstram que a solução proposta é promissora, contribuindo para o avanço da pesquisa na área, possibilitando a análise de experimentos em ambiente simulado e realização de testes para posterior implantação de sistemas robóticos em cenários do mundo real.

Palavras-chave: Robôs Móveis. Navegação Autônoma. Ambientes Lotados. Prevenção de Colisões.

ABSTRACT

AFONSO, Paulo de Almeida. **Autonomous Navigation in Dynamic Environments with Human Interaction based on Deep Reinforcement Learning and Computer Vision**. Advisor: Paulo Roberto Ferreira Jr.. 2023. 103 f. Thesis (Doctorate in Computer Science) – Technology Development Center, Federal University of Pelotas, Pelotas, 2023.

Recent research demonstrates that autonomous navigation in dynamic environments shared with humans remains an ongoing challenge. In such environments, the movement of people can create obstacles, impeding sensing and hindering the robot's perception of its position. Furthermore, the uncertainty of human behavior can lead to unsafe situations for both the robot and the people around it. Given this scenario, the study of learning-based methods has gained prominence, along with the implementation of hybrid structures that combine different architectures and algorithms in pursuit of an efficient solution. This work proposes the integration of deep reinforcement learning techniques with computer vision methods to develop a solution capable of enabling the robot to navigate autonomously and safely in indoor environments shared with humans, considering the specific characteristics associated with the problem at hand. Thus, the navigation must take into account safety concerns, such as the distancing between the robot and the people in its vicinity. To achieve this, an innovative approach based on deep reinforcement learning has been developed, utilizing the Deep Deterministic Policy Gradient (DDPG) algorithm, combined with computer vision techniques. Comparative tests between the DDPG and Deep Q-Network (DQN) algorithms were conducted, addressing four distinct stages, each representing two different training environment scenarios and with complexity levels higher than what the robot was trained on. The DDPG algorithm demonstrated greater efficiency and stability than the DQN, with higher average success rates in all analyzed stages, showcasing excellent generalization capacity and consistently better results in different environments than the training setting. On the other hand, the DQN struggled to avoid collisions and achieved significantly lower average success rates. These findings underscore the superiority of DDPG and demonstrate the promise of the proposed solution, contributing to the advancement of research in the field. This allows for the analysis of experiments in simulated environments and testing for the subsequent deployment of robotic systems in real-world scenarios.

Keywords: Mobile Robots. Autonomous Navigation. Crowded Environments. Collision Avoidance.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AG	*Admissible Gap*
APF	*Artificial Potential Field*
AUV	*Autonomous Underwater Vehicle*
CADRL	*Collision Avoidance with Deep Reinforcement Learning*
CNN	*Convolutional Neural Network*
DBN	*Deep Belief Network*
DDPG	*Deep Deterministic Policy Gradient*
DDQN	*Double Deep Q-Network*
DL	*Deep Learning*
DQN	*Deep Q-Network*
DRL	*Deep Reinforcement Learning*
EP	*Experience Pool*
ER	*Experience Replay*
FPN	*Feature Pyramid Network*
FPS	*Frames per Second*
GA3C	*Asynchronous Advantage Actor-Critic*
GAN	*Generative Adversarial Networks*
HRVO	*Hybrid Reciprocal Velocity Obstacle*
HRI	*Human-Robot Interaction*
IRL	*Inverse Reinforcement Learning*
LIDAR	*Light Detection and Ranging*
LSTM	*Long Short-Term Memory*
MDP	*Markov Decision Process*
NLP	*Natural Language Processing*
ORCA	*Optimal Reciprocal Collision Avoidance*
OSRF	*Open Source Robotics Foundation*

PPO	*Proximal Policy Optimization*
ReLU	*Rectified Linear Unit*
RL	*Reinforcement Learning*
ROS	*Robot Operating System*
RNN	*Recurrent Neural Network*
RVO	*Reciprocal Velocity Obstacle*
SAN-DDPG	*Social Attention Navigation - DDPG*
SARSA	*State - Action - Reward - State - Action*
SLAM	*Simultaneous Localization and Mapping*
SLIC	*Simple Linear Iterative Clustering*
SSD	*Single Shot MultiBox Detector*
SVO	*Safety Velocity Obstacle*
TD	*Temporal Difference*
UAV	*Unmanned Aerial Vehicles*
VO	*Velocity Obstacle*
YOLO	*You Only Look Once*

SUMÁRIO

1 INTRODUÇÃO

A robótica móvel é uma área multidisciplinar que envolve o desenvolvimento e a implementação de robôs capazes de se movimentar de forma autônoma em diferentes ambientes. Essa área de pesquisa combina conhecimentos de robótica, inteligência artificial, visão computacional, aprendizado de máquina e planejamento de trajetória para criar sistemas robóticos capazes de perceber, interpretar e interagir com o ambiente ao seu redor.

Esses sistemas robóticos, ou simplesmente robôs, podem ser utilizados em diversas aplicações, como logística, serviços domésticos, inspeção industrial, exploração espacial, ambientes educacionais, entre outros, abrangendo desde pequenos robôs terrestres e aéreos até veículos maiores, como carros autônomos (ROBOTS AND THEIR APPLICATIONS, 2018).

Muitos robôs móveis são operados remotamente, executando tarefas que dependem de um operador para controlar o dispositivo. Esses robôs não possuem autonomia total, geralmente são utilizados para proporcionar ao operador acesso remoto a locais perigosos, distantes ou inacessíveis. Alguns deles podem ser semi-autônomos, realizando algumas tarefas de forma automática.

Um robô é autônomo quando possui a capacidade de determinar as ações necessárias para a execução das tarefas que devem ser realizadas, necessitando para tanto, de um sistema de percepção e controle (RUBIO; VALERO; LLOPIS-ALBERT, 2019). Isso significa que o robô deve ser capaz de planejar suas trajetórias, evitar obstáculos, tomar decisões em tempo real e interagir com o ambiente de forma inteligente.

A navegação autônoma, por sua vez, é uma área específica da robótica móvel que se dedica ao desenvolvimento de algoritmos e sistemas que permitem que os robôs se movimentem de forma autônoma em seu ambiente, ou seja, sem a necessidade de controle humano constante. Trata-se de um campo em constante evolução, impulsionado pela crescente demanda por sistemas robóticos, sendo uma funcionalidade indispensável para diversas aplicações que envolvem a utilização de robôs móveis, tais como: robótica de serviço, vigilância, logística, entre outros. Ao longo dos anos, várias abordagens e técnicas foram desenvolvidas para enfrentar os desafios relacio-

nados a essa área de pesquisa.

Uma das abordagens amplamente exploradas é a utilização de sistemas de localização e mapeamento simultâneos (SLAM). O SLAM permite que um robô construa um mapa do ambiente enquanto simultaneamente estima sua própria posição no mapa (CADENA et al., 2016). Isso é especialmente relevante em ambientes desconhecidos ou dinâmicos, nos quais o robô precisa se adaptar e atualizar seu conhecimento do ambiente em tempo real.

Apesar do sucesso do SLAM, a navegação autônoma em ambientes compartilhados com humanos apresenta desafios adicionais, como garantir a segurança das pessoas em torno do robô. Nesse contexto, a revisão da literatura evidencia um grande avanço na pesquisa de soluções baseadas em aprendizado por reforço (*Reinforcement Learning - RL*), principalmente a partir de 2017.

O RL permite que os robôs aprendam comportamentos adequados por meio da interação direta com o ambiente, recebendo *feedback* positivo ou negativo na forma de recompensas ou punições. Ao utilizar o aprendizado por reforço, os robôs podem adquirir habilidades de navegação segura e interação social (CIOU et al., 2018), aprender a evitar colisões (LONG et al., 2018), ou antecipar o comportamento humano (ALAHI et al., 2016) para responder de maneira adequada e desempenhar com sucesso a tarefa de navegação.

Além do aprendizado por reforço, a visão computacional desempenha um papel fundamental para a navegação autônoma, especialmente em ambientes compartilhados com humanos. Através da análise de dados visuais, como imagens ou vídeos capturados por câmeras, os robôs podem extrair informações valiosas sobre o ambiente e as pessoas ao seu redor. Essas informações são essenciais para a detecção de obstáculos, reconhecimento de objetos e pessoas, e interpretação do comportamento humano, proporcionando que o robô possa tomar decisões seguras e socialmente adequadas durante a navegação.

Avanços em técnicas de visão computacional, como redes neurais convolucionais, têm impulsionado o desenvolvimento de sistemas de percepção robustos. O trabalho de Alom et al. (2019) apresenta uma análise detalhada de várias arquiteturas de última geração, abordando diferentes modelos e suas aplicações.

A revisão da literatura demonstra que navegação autônoma tem evoluído com o desenvolvimento e a combinação de diferentes técnicas que abordam localização, mapeamento, planejamento de trajetória, navegação social, aprendizado por reforço e visão computacional. Apesar disso, a navegação em ambientes compartilhados com humanos permanece como um problema em aberto. Nesse sentido, novas soluções continuam sendo pesquisadas para lidar com o problema em questão.

1.1 Motivação

À medida que os robôs se tornam cada vez mais integrados à sociedade, é fundamental que sejam capazes de navegar em espaços compartilhados sem representar riscos para as pessoas. Os robôs devem ser capazes de compreender e responder às complexidades do ambiente, como obstáculos em constante mudança, restrições de espaço e comportamento humano imprevisível.

Trabalhos recentes têm demonstrado que a navegação autônoma nesse tipo de ambiente é uma tarefa particularmente desafiadora. Em ambientes compartilhados com pessoas, a incerteza do comportamento humano pode levar a situações inseguras para o robô. Além disso, a movimentação das pessoas pode gerar obstruções, dificultando o sensoriamento e prejudicando a percepção do robô em relação à sua posição no ambiente.

A aplicação do aprendizado por reforço tem despertado um grande interesse e motivado uma ampla gama de pesquisas (PANCHPOR; SHUE; CONRAD, 2018). Tal fato é decorrente da crescente demanda por sistemas robóticos, capazes de operar de forma autônoma e segura em ambientes sociais e dinâmicos, nos quais a interação com seres humanos é inevitável. Nesse tipo de cenário busca-se a aplicação de métodos eficientes, capazes de realizar a tarefa de navegação com segurança, sem o conhecimento prévio do ambiente ou das ações dos demais agentes e obstáculos.

A maioria das abordagens existentes dividem-se em métodos reativos, quando o processo de tomada de decisão do agente é iniciado ao identificar uma colisão iminente (ZHANG et al., 2015) e métodos preditivos, objetivando lidar com a incerteza comportamental, característica de obstáculos dinâmicos (PFEIFFER et al., 2016).

Destaca-se ainda uma sub-divisão de técnicas, considerando estratégias baseadas em aprendizado (FAN et al., 2018), ou o uso de algoritmos reativos (BAREISS; BERG, 2015), por vezes combinados com técnicas de aprendizado, ou por meio da modelagem do comportamento social dos seres humanos (CHEN et al., 2017).

Apesar da variedade de soluções existentes, a análise das pesquisas relacionadas ao tema demonstra que, do melhor do nosso conhecimento, esse problema ainda não possui uma solução definitiva. Alguns trabalhos destacam que técnicas de prevenção de obstáculos puramente reativas não são suficientes para solucionar problemas de navegação em ambientes dinâmicos (LORENTE; OWEN; MONTANO, 2018; FERRER; SANFELIU, 2018). Para os autores, nesse tipo de cenário, o robô deve coexistir ou cooperar com humanos ou outros veículos em movimento. Outros observam que o planejamento do movimento requer a capacidade de prever a evolução futura dos obstáculos, observando restrições de movimentos que envolvem a dinâmica da plataforma móvel (velocidades e acelerações) para obtenção de trajetórias viáveis, considerando segurança, manobrabilidade, além de restrições do ambiente (VEMULA; MUELLING;

OH, 2017). Além disso, alguns trabalhos sugerem que o agente deve ser capaz de aprender um modelo de interação a partir de dados de trajetória ou comportamento humano real, modelando velocidades de outros agentes na multidão ou por meio da identificação da personalidade cada pedestre (BERA et al., 2017).

A principal motivação para o desenvolvimento deste trabalho é impulsionada pelas dificuldades identificadas e pela necessidade de desenvolver uma solução segura, eficiente e adaptável, adequada para a navegação de robôs móveis autônomos em ambientes compartilhados com humanos.

Assim, espera-se colaborar com o desenvolvimento de uma solução capaz de proporcionar que os robôs possam navegar de forma autônoma, segura e confiável em ambientes sociais complexos, proporcionando benefícios significativos para sua integração em atividades cotidianas e contribuindo para o avanço do estado da arte.

1.2 Objetivos de Pesquisa

1.2.1 Objetivos Gerais

O objetivo deste trabalho é desenvolver um sistema para navegação autônoma de robôs móveis em ambientes internos compartilhados com humanos. Para isso, foi desenvolvida uma abordagem que envolve a percepção do ambiente por meio de sensores, aprendizado por reforço, visão computacional e técnicas para detecção e prevenção de obstáculos móveis e estáticos. Os experimentos e testes foram conduzidos apenas em ambiente simulado, por meio da implementação de diferentes cenários no simulador Gazebo.

1.2.2 Objetivos Específicos

Cadeiras de rodas motorizadas são dispositivos auxiliares destinados a melhorar a qualidade de vida de pessoas com deficiência. A tecnologia embarcada nesses dispositivos permite que indivíduos com mobilidade limitada naveguem para destinos específicos sem a necessidade de auxílio de outras pessoas. No entanto, indivíduos com deficiência motora, baixa acuidade visual ou falta de força muscular enfrentam desafios para manipular um *joystick* e evitar obstáculos ao se mover pelo ambiente.

Frente a esse problema, um dos principais objetivos específicos deste trabalho é o desenvolvimento de uma solução capaz de proporcionar que cadeiras de rodas robóticas possam navegar de forma autônoma em ambientes como aeroportos, museus e shoppings. Dessa forma, a plataforma robótica utilizada neste trabalho é um modelo simulado de uma cadeira de rodas motorizada. O objetivo é realizar um estudo de caso para avaliar a viabilidade de aplicação da solução proposta ao problema em questão.

A partir da construção e validação da abordagem aqui apresentada, espera-se:

- Desenvolver um sistema para navegação autônoma de cadeiras de rodas motorizadas em ambientes internos compartilhados com humanos, utilizando aprendizado por reforço e visão computacional;
- Desenvolver um sistema de prevenção de colisões para navegação autônoma eficiente, capaz de lidar com a incerteza comportamental dos seres humanos em ambientes dinâmicos, garantindo a segurança das pessoas em torno do robô;
- Proporcionar que o sistema possa ser estendido à diferentes plataformas robóticas, contribuindo para a integração de robôs em atividades humanas cotidianas e para o estado da arte;
- Analisar o comportamento do robô durante a navegação em meio a testes simulados, avaliando o aprendizado contínuo e sua adaptação à ambientes em mudança e propor melhorias para experimentos futuros;
- Comparar o sistema desenvolvido com abordagens recentes e analisar o desempenho e robustez da solução proposta.

1.3 Contribuições

Este trabalho contribui para o avanço do estado-da-arte no estudo e desenvolvimento de sistemas de navegação para robôs móveis autônomos em ambientes internos compartilhados com humanos, destacando-se:

- A abordagem proposta é nova e eficiente, contribuindo para o avanço da pesquisa na área, permitindo a análise de experimentos em ambiente simulado e realização de testes para a implantação segura e eficiente de sistemas robóticos em uma ampla gama de cenários;
- O algoritmo DDPG, combinado com a visão computacional, é capaz de resolver tarefas em ambientes complexos, apresentando grande capacidade de aprendizado e generalização. Essas características permitem que o robô adquira conhecimento a partir de dados e experiências, e aplique esse conhecimento para lidar com novas situações e ambientes de forma eficaz;
- A possibilidade de implantação do sistema em uma cadeira de rodas robótica representa um grande avanço tecnológico, com potencial de melhorar a qualidade de vida das pessoas, facilitando tarefas cotidianas, proporcionando maior independência e inclusão social para os usuários;

- O estudo realizado demonstra a importância de avaliar a capacidade de generalização do conhecimento adquirido através do aprendizado por reforço, por meio da aplicação de testes em diferentes cenários. Para que um robô autônomo navegue de forma segura em ambientes do mundo real, ele deve ser capaz de generalizar suas habilidades de navegação para além das situações específicas em que foi treinado.

1.4 Organização dos Capítulos

No restante desta tese, serão apresentados estudos aprofundados, experimentos e análises dos resultados obtidos, bem como discussões sobre os desafios e as perspectivas futuras.

Este trabalho está organizado da seguinte forma:

Capítulo 2: Neste capítulo são apresentados os conceitos relacionados ao tema da pesquisa, abordando o Aprendizado por Reforço, Aprendizado Profundo e o Aprendizado por Reforço Profundo. Por fim, são apresentados o algoritmo *Deep Q-Network (DQN)*, implementado para comparação com a solução proposta por este trabalho, e as considerações finais acerca deste capítulo.

Capítulo 3: Neste capítulo são apresentados os trabalhos relacionados ao tema desta pesquisa. São abordadas soluções referentes à prevenção de colisões, considerando métodos reativos e métodos preditivos, além de abordagens para detecção de pessoas. Por fim, são apresentadas as considerações finais acerca deste capítulo;

Capítulo 4: Neste capítulo é detalhada a abordagem proposta, apresentando a metodologia utilizada, a plataforma robótica simulada, a estrutura do ambiente de aprendizado e os ambientes implementados para aplicação de treinamento e de testes. Por fim, é descrito o algoritmo *Deep Deterministic Policy Gradient* (DDPG), sua arquitetura e aplicação neste trabalho;

Capítulo 5: Neste capítulo são apresentados os experimentos e resultados. São abordadas as etapas de treinamento, configurações, testes realizados, validação dos experimentos e resultados obtidos. Por fim, são apresentadas as considerações finais acerca deste capítulo;

Capítulo 6: Neste capítulo são apresentadas as conclusões, as contribuições decorrentes deste trabalho, publicações relacionadas à tese, dificuldades encontradas durante a realização dos experimentos e perspectivas para o desenvolvimento de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Aprendizado por Reforço

Os problemas de Aprendizado por Reforço (*Reinforcement Learning - RL*) envolvem aprender o que fazer, mapeando situações para ações, objetivando maximizar um valor de recompensa (SUTTON, 1992, 1998).

Em situações que abordam a complexidade do mundo real, para utilização do RL, os agentes devem derivar representações eficientes do ambiente e, a partir de entradas sensoriais recebidas, utilizá-las para generalizar a experiência passada, com vistas à aplicação em situações futuras (MNIH et al., 2015).

Existem quatro componentes básicos em RL: agente, ambiente, recompensa e ação. Um algoritmo de RL típico opera apenas com conhecimento limitado do ambiente e com feedback limitado sobre a qualidade das decisões (ZHANG; HAN; DENG, 2018). Os algoritmos mais populares de RL incluem o *Q-learning*, SARSA (*State - Action - Reward - State - Action*), DQN (*Deep Q-Network*) e DDPG (*Deep Deterministic Policy Gradient*).

Em uma abordagem RL, um agente autônomo, controlado por um algoritmo de aprendizado de máquina, observa um estado s_t de seu ambiente em uma determinada etapa t do tempo. Na ocorrência de uma interação entre o agente e o ambiente, executando uma ação no estado s_t, ambos fazem uma transição para um novo estado s_{t+1} (ARULKUMARAN et al., 2017a).

O estado é representado por uma estatística do ambiente, devendo incluir as informações necessárias para que o agente tome a melhor ação. A melhor sequência de ações é determinada pelas recompensas fornecidas pelo ambiente, que ao passar para um novo estado, fornece uma recompensa escalar r_{t+1} ao agente como *feedback*.

As ações do sistema de aprendizado influenciam suas entradas posteriores e podem afetar não apenas a recompensa imediata, mas também a próxima situação e, consequentemente, todas as recompensas subsequentes.

Geralmente, um problema sequencial de tomada de decisão pode ser formulado como um processo de decisão de Markov (MDP), descrito como $M = (S, A, R, P, \gamma)$,

onde S é o espaço de estados, A é o espaço de ação, R é a função de recompensa, P é o modelo de transição de estado e γ é um fator de desconto. Os principais elementos podem ser descritos conforme a seguir (XUE et al., 2019):

- Espaço de estados: o estado de entrada de todo o sistema é composto pelo estado do próprio robô e pelo estado do obstáculo (assim como outros robôs), que pode ser expresso como $s^c = [s,\ s^o] \in \Re^{10}$

- Espaço de ação: uma série de conjuntos de ações pré-projetados, denotados por $a(s_t) = a_t = v\ for\ v < v_{max}$

- Função de recompensa: Utilizada para recompensar o robô móvel para alcançar um determinado alvo ou aplicar uma penalidade por obstáculos de colisão. Consiste em quatro partes, e pode ser representada por $R(s^c,\ a) = R_1 + R_2 + R_3 + R_4$ onde:

  - $R_1 = k \cdot (dg_{t-1} - dg_t)$: utilizada para estimular o robô a se aproximar da posição de destino, onde:
    dg_{t-1} = distância linear entre o robô e a posição alvo no passo anterior;
    dg_t = distância linear entre o robô e a posição alvo no próximo passo; e
    k é um coeficiente constante.
  - R_2: indica a penalidade de colisão. As variáveis da e da_{min} representam, respectivamente, a distância atual entre o robô e o obstáculo e a distância mínima de segurança entre o robô e o obstáculo. O valor de R_2 é obtido da seguinte forma:
    $$R_2 = \begin{cases} -x\ if\ da < da_{min} \\ 0\ else \end{cases}$$
  - $R_3 = -c.timer$: um valor negativo para o tempo atual é utilizado. Quanto maior o custo de tempo, maior a punição imposta ao robô. É utilizado como um estímulo para encontrar o caminho ideal no processo de treinamento, objetivando que o tempo gasto para atingir a posição alvo seja o mais curto.
  - R_4: se o robô chegar à posição alvo, uma recompensa é dada, caso contrário, não é recompensado. O valor de R_4 é obtido da seguinte forma:
    $$R_4 = \begin{cases} x\ if\ reach\ goal \\ 0\ else \end{cases}$$

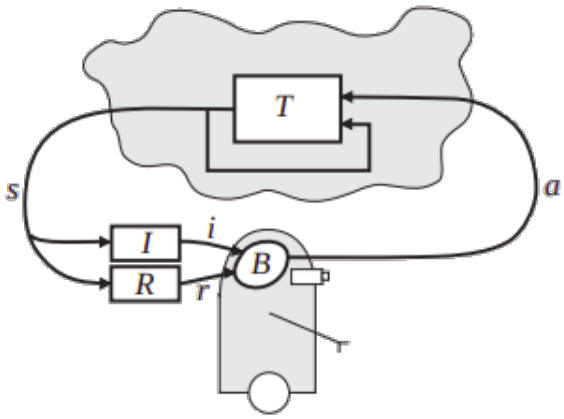


Figura 1 – Aprendizado por Reforço
Fonte: (KAELBLING; LITTMAN; MOORE, 1996)

A Figura 1 ilustra o modelo de aprendizado por reforço padrão, onde um agente é conectado ao ambiente por meio de percepções e ações.

Em cada etapa da interação com o ambiente T, o agente B recebe como entrada i alguma indicação do estado atual s e escolhe uma ação a para gerar como saída. A ação altera o estado do ambiente e o valor dessa transição de estado é comunicado ao agente por meio de um sinal de reforço escalar r.

O comportamento do agente a longo prazo deve ser capaz de escolher ações que tendem a aumentar a soma dos valores do sinal de reforço. Ele pode aprender a fazer isso ao longo do tempo por tentativa e erro sistemáticos, guiado por uma ampla variedade de algoritmos.

Métodos tradicionais de RL podem ser utilizados para modelar comportamentos reativos ou preditivos, a partir do processamento de dados de leitura dos sensores e posições relativas dos objetos que compõem o ambiente de navegação. Nesse tipo de solução, a prevenção de colisões é realizada com base em um modelo de velocidade e direção, onde a tarefa de navegação ocorre através do processamento dos dados recebidos, emitindo comandos de velocidade linear e angular para controlar o robô e evitar obstáculos (LIU; LIU; WANG, 2017; QIANG et al., 2018).

Geralmente, os dados de treinamento são pré-processados através de uma rede neural e, na etapa posterior, o mapa de recursos extraído do modelo de aprendizado supervisionado é utilizado como entrada de dados para a rede, resultando em um conjunto de comandos para determinar as ações do robô (TAI; LIU, 2016a).

Como forma de melhorar o desempenho dos algoritmos e reduzir o custo computacional, podem ser utilizados mapas gerados artificialmente e conjuntos de dados de treinamento contendo amostragens do mundo real (LIU; XU; CHEN, 2018), ou por meio da criação de modelos independentes de decisão, utilizando simulações offline repetidas (LONG; LIU; PAN, 2017).

Para minimizar o tempo de aprendizado, é possível armazenar transações de ex-

periência, através da utilização de um módulo de memória (WU et al., 2017). O tempo de computação também pode ser reduzido por meio da remoção de repetições desnecessárias durante a etapa de treinamento da rede.

2.2 Aprendizado Profundo

O Aprendizado Profundo (*Deep Learning - DL*) permite a descoberta de estruturas complexas em grandes conjuntos de dados. Para isso, utiliza um algoritmo de retropropagação para indicar como uma máquina deve alterar seus parâmetros internos, usados para calcular a representação em cada camada a partir da camada anterior (LECUN; BENGIO; HINTON, 2015).

Essa capacidade proporciona que modelos computacionais, compostos por várias camadas de processamento, aprendam representações de dados com vários níveis de abstração, resultando em melhorias significativas para o estado da arte em diversos domínios, como reconhecimento de fala, reconhecimento visual de objetos, detecção de objetos, entre outros.

Métodos baseados em DL são compostos por vários níveis de representação, obtidos pela composição de módulos simples, mas não lineares, que transformam dados brutos de entrada em um nível de representação mais alto e mais abstrato, necessários para detecção ou classificação (PRABHA; UMARANI SRIKANTH, 2019). O aprendizado se dá por meio da utilização de redes neurais profundas.

Conforme Prabha; Umarani srikanth (2019), o número de camadas ocultas entre as camadas de entrada e saída são determinantes para o aprendizado. As camadas iniciais são responsáveis pela extração dos recursos abstratos e, à medida que o aprendizado avança, as camadas profundas são responsáveis pelo fornecimento de informações importantes acerca dos recursos processados.

O DL pode ser classificado em duas arquiteturas principais: Redes Neurais Convolucionais (CNN) e Redes Neurais Recorrentes (RNN), descritas conforme a seguir, segundo a visão dos autores.

2.2.1 Redes Neurais Convolucionais

CNNs são algoritmos de aprendizado profundo amplamente utilizados e a categoria mais proeminente de redes neurais, principalmente para aplicações que envolvem dados de alta dimensionalidade, como imagens e vídeos (ALOM et al., 2019). Sua arquitetura é inspirada na neurobiologia do córtex visual e possui uma estrutura otimizada para processamento de imagens 2D e 3D, sendo muito eficazes no aprendizado e extração de abstrações de recursos 2D, com significativamente menos parâmetros do que uma rede totalmente conectada de tamanho semelhante.

A estrutura de rede foi proposta pela primeira vez por Fukushima em 1988 (FU-

KUSHIMA, 1988), sendo modificada na década de 1990, através da aplicação de um algoritmo de aprendizado baseado em gradiente para lidar com o problema de classificação de dígitos manuscritos (LECUN et al., 1998).

A CNN é uma rede *feedforward* utilizada principalmente para o processamento de imagens. A capacidade da CNN depende do número de camadas ocultas usadas entre as camadas de entrada e saída e cada camada é responsável pela extração de um conjunto de características. Os mapas de recursos são gerados pela aplicação de uma série de filtros sobre a entrada e cada filtro percorre toda a entrada, multiplicando seus pesos pelos valores obtidos.

O resultado é passado para uma função de ativação como ReLU (*Rectified Linear Units*), sigmoide ou tangente hiperbólica (tanh), enquanto uma função de perda é usada para avaliar o conjunto de pesos. Os mapas de recursos gerados pelos filtros destacam diferentes características dos dados de entrada. A CNN tem quatro tipos de camadas principais: camada de convolução; camada ReLU; camada de subamostragem (ou pooling) e; camada totalmente conectada. De acordo com Prabha; Umarani srikanth (2019), sua principal desvantagem é que a CNN não pode lidar com dados sequenciais.

Conforme Lecun; Bengio; Hinton (2015), diversas aplicações de aprendizado profundo usam arquiteturas de rede neural *feedforward*, que aprendem a mapear entradas de tamanho fixo, como imagens, para uma saída de tamanho fixo, como por exemplo, uma probabilidade para cada uma das várias categorias. Esse aprendizado se dá por meio de transição entre camadas, utilizando um conjunto de unidades para calcular uma soma ponderada de suas entradas da camada anterior e passar o resultado por uma função não linear.

A função não linear mais popular é a ReLU, um retificador de meia onda $f(z) = \max(z, 0)$. Segundo os autores, nas últimas décadas, as redes neurais usavam não linearidades mais suaves, como $\tanh(z)$ ou $\frac{1}{1+\exp(-z)}$, no entanto, a ReLU tende a aprende mais rápido em redes com muitas camadas, permitindo o treinamento de uma rede supervisionada profunda sem pré-treinamento não supervisionado. Unidades que não estão na camada de entrada ou saída são chamadas de camadas ocultas, responsáveis pelo processamento da camada de entrada de maneira não linear, de modo que as categorias se tornem linearmente separáveis pela última camada.

2.2.2 Redes Neurais Recorrentes

Ao contrário da CNN, que é uma rede de alimentação direta, Redes Neurais Recorrentes (RNNs) utilizam propagação reversa. Ou seja, as entradas atuais também consideram as entradas anteriores. RNNs foram projetadas com base no princípio de que os humanos não pensam do zero, podendo processar dados sequenciais com a ajuda de uma memória interna. As principais características das RNNs são:

- Propagação reversa: A RNN usa propagação reversa no tempo durante o treinamento. Isso significa que ela começa a processar a sequência de dados desde o final até o início, o que permite que a rede capture dependências de longo prazo, levando em consideração as entradas anteriores enquanto processa as atuais.

- Entradas sequenciais e memória interna: RNNs são projetadas para trabalhar com dados sequenciais, como sequências de texto, áudio ou séries temporais. Elas possuem uma memória interna que lhes permite "lembrar" informações das etapas anteriores ao processar cada elemento da sequência.

- Inspiradas no princípio do pensamento humano: A ideia de usar memória interna nas RNNs é inspirada na forma como os humanos processam as informações, ou seja, as informações não são processadas a partir do zero, mas sim com base no conhecimento adquirido anteriormente. Essa capacidade das RNNs de manter um estado interno e usar informações anteriores para influenciar decisões futuras as torna adequadas para tarefas que envolvem dependências sequenciais, como aplicações em NLP (*Natural Language Processing*), reconhecimento de voz e tradução, entre outros.

Apesar de suas vantagens, as RNNs também têm algumas limitações, como dificuldades em lidar com dependências de longo prazo devido ao problema de desvanecimento ou explosão do gradiente, o que dificulta a aprendizagem em sequências muito longas. Para resolver esses problemas, foram desenvolvidas outras arquiteturas de redes neurais, como as redes LSTM (*Long Short-Term Memory*), que são variantes das RNNs, projetadas para mitigar os problemas de dependências de longo prazo.

Atualmente, existem diferentes tipos de arquiteturas de aprendizado profundo, disponíveis para uma variedade de soluções, de acordo com o contexto de aplicação (ZHANG; HAN; DENG, 2018): Máquina de Boltzmann (BM), Redes DBN (*Deep Belief Network*, Redes FDN *Feedforward Neural Network*, Redes Neurais Convolucionais (CNN), Redes Neurais Recorrentes (RNN), Redes de Memória de Longo Prazo (LSTM), Redes GAN *Generative Adversarial Networks*, entre outras.

2.2.3 Detecção de pessoas

A detecção de pessoas desempenha um papel fundamental na navegação autônoma, é crucial para garantir a segurança e a interação eficiente entre os robôs e as pessoas que compartilham o mesmo espaço.

A capacidade de detectar e reconhecer pessoas com precisão permite que os robôs tomem decisões informadas e naveguem com mais segurança. Ao reconhecer a presença de pessoas, o robô pode adaptar seu comportamento e movimento para garantir uma navegação suave e não invasiva e evitar caminhos congestionados.

Isso é especialmente importante em ambientes dinâmicos, como hospitais, fábricas ou espaços públicos, onde há um alto fluxo de pessoas.

As CNNs têm se destacado na detecção de pessoas devido à sua capacidade de aprender características discriminativas diretamente dos dados de entrada. Esse tipo de arquitetura é amplamente utilizado em tarefas de visão computacional, como classificação de textos (WANG et al., 2019), segmentação de imagem, classificação de imagens e detecção de objetos (PATHAK; PANDEY; RAUTARAY, 2018; DRUZHKOV; KUSTIKOVA, 2016) e reconhecimento facial.

A detecção de pedestres com base em CNN tem se dividido em duas categorias distintas (XIAO et al., 2021). Uma delas é conhecida como estrutura de dois estágios, também chamada de método baseado em região. Nesse método, inicialmente são geradas propostas de regiões que podem conter pedestres. Em seguida, as características dessas regiões são extraídas utilizando CNN e, por fim, um classificador é utilizado para classificar e reconhecer os pedestres nessas regiões.

Conforme Xiao et al. (2021), o outro método de detecção é denominado estrutura de estágio único, também conhecida como método de detecção direta, que visa acelerar a velocidade de detecção ao eliminar a etapa de geração de propostas regionais e, em vez disso, realizar uma regressão direta na área predefinida. Nessa abordagem, a CNN é treinada diretamente para regredir e identificar a área predefinida onde os pedestres podem estar presentes. O objetivo é reduzir a complexidade computacional e melhorar a eficiência do sistema.

A seguir são listadas as principais estruturas de detecção de dois estágios e estruturas de estágio único:

2.2.3.1 Estruturas de detecção de dois estágios

- *Regions with CNN features (R-CNN)*: Combina redes neurais convolucionais (CNNs) de alta capacidade a propostas de regiões independentes de categoria para localizar e segmentar objetos. Utiliza um algoritmo de busca seletiva para extrair e combinar apenas 2.000 regiões da imagem a ser classificada (GIRSHICK et al., 2014).

- *Fast R-CNN*: Abordagem semelhante ao algoritmo R-CNN, porém mais rápida. Não é necessário alimentar 2.000 propostas de região para a rede neural convolucional a cada etapa de convolução. Em vez disso, a operação de convolução é feita apenas uma vez por imagem e um mapa de características é gerado a partir dela (WANG; SHRIVASTAVA; GUPTA, 2017).

- *Faster R-CNN*: Essa abordagem elimina o algoritmo de busca seletiva e permite que a rede aprenda as propostas de região. De forma similar ao Fast R-CNN, uma imagem é fornecida como entrada para uma rede convolucional que retorna

um mapa de características, então uma rede separada é usada para prever as propostas de região (REN et al., 2015).

- *Mask R-CNN*: Método para detecção de objetos em uma imagem, capaz de gerar simultaneamente uma máscara de segmentação de alta qualidade para cada instância, além de estimar poses humanas na mesma estrutura (HE et al., 2017). O Mask R-CNN estende o Faster R-CNN adicionando uma ramificação para prever uma máscara de objeto em paralelo com a ramificação existente para reconhecimento de caixas delimitadoras.

- *Focal loss for dense object detection (RetinaNet)*: Detector de um estágio convolucional que aplica um termo modulador à perda de entropia cruzada para lidar com o problema de desequilíbrio de classe atribuindo mais pesos a exemplos difíceis ou facilmente mal classificados (LIN et al., 2017).

2.2.3.2 Estruturas de detecção de estágio único

- *Single Shot MultiBox Detector (SSD)*: Método para detecção de objetos em imagens usando uma única rede neural profunda. A abordagem SSD é baseada em uma rede convolucional *feedforward* que produz uma coleção de tamanho fixo de caixas delimitadoras e pontuações para a presença de instâncias de classe de objeto nessas caixas, seguida por uma etapa de supressão não máxima para produzir as detecções finais (LIU et al., 2016).

- *You Only Look Once (YOLO)*: Método para detecção unificada de objetos em tempo real. Uma única rede neural prevê caixas delimitadoras e probabilidades de classe diretamente de imagens completas em uma avaliação (REDMON et al., 2016). O algoritmo utiliza uma única propagação direta através da CNN para detecção de objetos. Atualmente, várias versões foram atualizadas para melhorar seu desempenho.

  De acordo com o estudo realizado por Xiao et al. (2021), o SSD tem vantagens sobre o YOLO na resolução de problemas de pequena escala e localização, porém, apresenta limitações na redução de falsos positivos ao lidar com pedestres em cenários complexos.

2.3 Aprendizado por Reforço Profundo

Conforme descrito na seção anterior, o Aprendizado Profundo consiste em aproximar funções não lineares através do treinamento de redes neurais profundas. Essa capacidade de representação proporcionou um avanço para o RL ao integrar-se com redes neurais profundas, formando o Aprendizado por Reforço Profundo (*Deep Rein-*

forcement Learning - DRL), uma combinação do aprendizado por reforço com o aprendizado profundo.

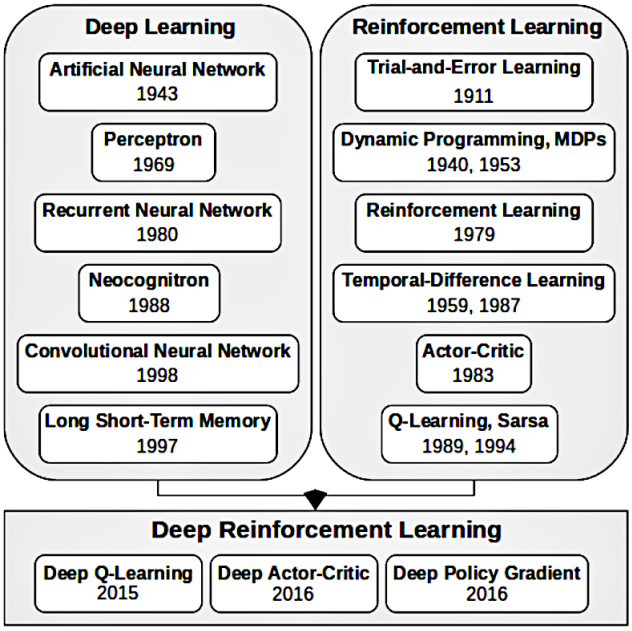


Figura 2 – DL, RL e DRL
Fonte: Adaptado de Fenjiro; Benbrahim (2018)

O DRL foi proposto pela primeira vez em 2013 para o aprendizado de políticas de controle aplicadas aos jogos de computador Atari 2600 (MNIH et al., 2013). A solução utilizou uma variante do algoritmo *Q-Learning*, conectado a uma rede neural profunda, para o processamento de dados de treinamento a partir de imagens RGB.

Desde então, sua utilização tem sido pesquisada para uma variedade de aplicações, tais como navegação autônoma em ambientes internos (TAI; LIU, 2016b), condução de veículos autônomos em cenários urbanos (SALLAB et al., 2017; WULFMEIER et al., 2017), reconhecimento de objetos estáticos e móveis (ZUO; DU; LU, 2017), prevenção de colisões durante a tarefa de navegação autônoma (RUAN et al., 2019), entre outras.

Diferente de métodos puramente reativos, em uma abordagem DRL busca-se codificar comportamentos cooperativos através do aprendizado de uma função de valor (CHEN et al., 2017), ou aprender com as experiências de vários agentes durante as etapas de treinamento (EVERETT; CHEN; HOW, 2018). Essas estratégias permitem que os algoritmos escolham as ações a serem realizadas, com base nas observações de um número arbitrário de agentes próximos, sem assumir que outros agentes sigam qualquer modelo comportamental específico.

O DRL pode ser classificado em três abordagens: métodos baseados em valor, métodos baseados em políticas e métodos ator-crítico, uma abordagem híbrida que combina as duas primeiras, descritas conforme a seguir (ARULKUMARAN et al., 2017b):

2.3.1 Métodos baseados em valor

Métodos baseados em valor obtêm indiretamente a política do agente ao atualizar iterativamente a função de valor. Quando a função de valor alcança seu valor ótimo, a política do agente é obtida a partir dessa função. São baseados na estimativa do valor (retorno esperado) de estar em um determinado estado. A função de valor do estado $V_\pi(s)$ representa o retorno esperado ao iniciar no estado s e seguir a política π a partir daquele ponto em diante:

$$V_\pi(s) = \mathbb{E}[R \,|\, s, \pi]$$

Onde:

$V_\pi(s)$ é a função de valor do estado s sob a política π; e

$\mathbb{E}[R \,|\, s, \pi]$ é o valor esperado do retorno (recompensa acumulada) ao seguir a política π a partir do estado s.

A política ótima π^* tem uma função de valor do estado correspondente $V^*(s)$ e vice-versa, a função de valor do estado ótima pode ser definida como:

$$V^*(s) = \max_\pi V^\pi(s) \quad \text{para todo } s \in S.$$

Onde:

$V^*(s)$ é a função de valor ótima do estado s;

$V^\pi(s)$ é a função de valor do estado s sob a política π;

S é o conjunto de todos os estados possíveis.

Com $V^*(s)$ disponível, a política ótima pode ser recuperada escolhendo entre todas as ações disponíveis no estado s_t e selecionando a ação a que maximiza $\mathbb{E}_{s_{t+1}\sim T}[V^*(s_{t+1})]$.

No cenário de RL, a dinâmica de transição T não está disponível. Portanto, outra função é necessária, a função valor estado-ação ou função qualidade $Q_\pi(s, a)$, que é similar a V_π, exceto que a ação inicial a é fornecida, e a política π é seguida somente a partir do próximo estado em diante:

$$Q_\pi(s, a) = \mathbb{E}[R \,|\, s, a, \pi].$$

A melhor política, dada a função $Q_\pi(s, a)$, pode ser encontrada escolhendo a de forma gananciosa em cada estado: $\text{argmax}_a Q_\pi(s, a)$. Sob essa política, também é possível definir $V_\pi(s)$ maximizando $Q_\pi(s, a)$:

$$V_\pi(s) = \max_a Q_\pi(s, a).$$

Ou seja, o valor do estado s sob a política π é igual ao máximo valor da função $Q_\pi(s, a)$ em relação a todas as ações possíveis a.

Algoritmos típicos de DRL, baseados em valor, incluem o Q-Learning, Deep Q-Network (DQN), Double Q-Learning, Dueling DQN, SARSA (*State-Action-Reward-State-Action*), entre outros.

2.3.2 Métodos baseados em política

Métodos baseados em políticas utilizam diretamente o método de aproximação de função para estabelecer uma rede de políticas. As ações são então selecionadas por meio dessa rede para obtenção do valor de recompensa, enquanto os parâmetros da rede de políticas são otimizados ao longo da direção do gradiente para obter uma política capaz de maximizar o valor de recompensa.

Conforme Arulkumaran et al. (2017b), nessa abordagem não é necessário manter um modelo de função de valor, mas buscar diretamente uma política ótima π^*. Tipicamente, uma política parametrizada π_θ é escolhida, cujos parâmetros θ são atualizados para maximizar o retorno esperado $\mathbb{E}[R \mid \theta]$, utilizando otimização baseada em gradientes ou sem gradientes.

Segundo os autores, a otimização sem gradiente pode abranger efetivamente espaços de parâmetros de baixa dimensão, mas apesar de alguns sucessos em aplicá-los a redes grandes, o treinamento baseado em gradiente continua sendo o método de escolha para a maioria dos algoritmos de DRL, sendo mais eficiente em termos de amostras, ao lidar com políticas que possuem um grande número de parâmetros.

2.3.3 Métodos Ator-Crítico

A combinação de funções de valor com uma representação explícita da política resulta em métodos ator-crítico. O "ator" (política) aprende usando o *feedback* do "crítico" (função de valor). Dessa forma, esses métodos proporcionam um equilíbrio entre a redução da variância dos gradientes de política e a introdução de viés a partir dos métodos de função de valor.

Em métodos ator-crítico a função de valor é utilizada como uma linha de base para os gradientes de política, de modo que a principal diferença com outros métodos de linha de base é que os métodos ator-crítico utilizam uma função de valor aprendida (ARULKUMARAN et al., 2017b).

Os autores observam que em vez de utilizar a média de vários resultados de Monte Carlo, como a linha de base para os métodos de gradiente de política, abordagens ator-crítico ganharam popularidade como um meio eficaz de combinar os benefícios dos métodos de busca de política com funções de valor aprendidas. Dessa forma, podem aprender a partir de retornos completos e/ou erros TD (*Temporal Difference*) e também se beneficiar de melhorias tanto dos métodos de gradiente de política, quanto dos métodos de função de valor.

2.4 Deep-Q Network

O Deep-Q Network (DQN) foi a primeira aplicação do *Q-learnin* ao aprendizado profundo, constituindo uma das abordagens mais conhecidas e bem-sucedidas para aprender políticas ótimas em ambientes de aprendizado por reforço.

O DQN é uma variação do algoritmo *Q-Learning*, composto por três principais melhorias em sua arquitetura: uma rede neural convolucional profunda para aproximação da função Q; utilização de mini-lotes de dados de treinamento aleatórios, em vez de atualização de um único passo na última experiência; e o uso de parâmetros de rede mais antigos para estimar os valores Q do próximo estado (RODERICK; MACGLASHAN; TELLEX, 2017).

O pseudocódigo a seguir descreve o algoritmo DQN, conforme proposto por (MNIH et al., 2015):

Algorithm 1 Algoritmo DQN

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
for $episode = 1, M$ **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  for $t = 1$ **do**
    With probability ε, select a random action a_t
    Otherwise, select $a_t = \arg\max_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in the emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi_{(S_{t+1})}$
    Store experience $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of experiences $(\phi_t, a_t, r_t, \phi_{t+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \, max_{a'} \, \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the weights θ
    Every C steps reset $\hat{Q} = Q$
  end for
end for

As principais características do DQN são descritas conforme a seguir (FENJIRO; BENBRAHIM, 2018):

- Rede-alvo: configurada para lidar separadamente com o erro de diferença temporal (TD) no algoritmo. O parâmetro θ_i da rede Q atual $Q(s, a; \theta_i)$ é copiado para θ'_i da rede-alvo $Q(s', a'; \theta'_i)$ a cada n passos de tempo, o que evita a instabilidade da rede-alvo devido às alterações feitas na rede Q atual durante o treinamento.

A ideia consiste em usar uma rede separada para estimar os valores $Q - alvo$ que serão usados para calcular a perda para cada ação. Essa rede-alvo possui a mesma arquitetura do aproximador de função, mas com pesos fixos. A cada T passos (por exemplo, a cada 1000 passos), os pesos da rede Q são copiados para a rede-alvo, o que proporciona maior estabilidade ao DQN.

- Pool de experiências (*Experience Pool - EP*) $\mathcal{U}(D)$: usado para armazenar e gerenciar amostras (s, a, r, s'), enquanto um mecanismo de repetição de experiências (*Experience Replay - ER*) é utilizado para selecionar as amostras. Essas amostras são armazenadas no EP, de onde lotes são selecionados aleatoriamente para treinar a rede Q. O mecanismo de ER ajuda a eliminar a correlação entre as amostras.

  A repetição de experiências quebra a similaridade entre as amostras de treinamento subsequentes, o que poderia levar a rede a um mínimo local, e resolve os desafios de correlação de dados e distribuições de dados não estacionárias. Essa técnica permite que a rede aprenda de maneira mais estável, reduzindo a dependência excessiva das amostras temporais e melhorando a eficácia do aprendizado com base em experiências passadas.

Os parâmetros da rede neural são atualizados por meio do gradiente descendente. A função de perda do DQN é representada como:

$$L(\theta_i) = \mathbb{E}\left[(r + \gamma \max_{a'} Q(s', a'; \theta_i') - Q(s, a; \theta_i))^2\right] \quad (1)$$

Onde:

- θ_i representa os parâmetros da rede Q atual $Q(s, a; \theta_i)$;
- $theta_i'$ representa os parâmetros da rede Q alvo $Q(s, a; \theta_i')$;
- s é o estado atual;
- a é a ação tomada no estado atual;
- r é a recompensa recebida após tomar a ação a no estado s;
- s é o próximo estado após tomar a ação a;
- a é a ação selecionada no próximo estado;
- γ é o fator de desconto, que determina a importância das recompensas futuras.

A função de perda representa o erro quadrático médio entre o valor Q previsto para o par estado-ação atual e o valor $Q - alvo$, que é a recompensa recebida mais

o valor Q máximo descontado de todas as possíveis ações no próximo estado. O objetivo do treinamento é minimizar essa função de perda para melhorar a precisão das estimativas dos valores Q e, consequentemente, o desempenho do agente.

Segundo Roderick; Macglashan; Tellex (2017), durante o treinamento usando o DQN, as curvas de aprendizado médio do Q-learning em configurações tabulares geralmente mostram melhorias relativamente estáveis, e problemas de aprendizado profundo supervisionado também costumam ter melhorias médias bastante estáveis à medida que mais dados se tornam disponíveis. No entanto, os autores observam que não é incomum no DQN ocorrer o chamado "esquecimento catastrófico", no qual o desempenho do agente pode cair drasticamente após um período de aprendizado.

Diante desse problema, os autores destacam a solução proposta por Mnih et al. (2015), que consiste em salvar os parâmetros da rede que resultaram no melhor desempenho nos testes. Assim, é possível restaurá-los posteriormente, permitindo que o agente mantenha a capacidade de atingir pontuações mais altas e continue aprendendo sem regredir significativamente em seu desempenho.

Essa abordagem é uma forma de mitigar o problema do esquecimento catastrófico e ajudar o agente a se manter em níveis de desempenho mais altos ao longo do tempo, mesmo durante o treinamento contínuo.

2.5 Considerações finais

Este capítulo apresentou os principais conceitos relacionados ao tema desta pesquisa. A solução proposta neste trabalho fundamenta-se nesses conceitos, constituindo um sistema baseado em Aprendizado por Reforço Profundo, uma combinação do aprendizado por reforço com o aprendizado profundo. Essa abordagem permite que um agente aprenda a tomar decisões, através da interação com o ambiente, utilizando redes neurais profundas para extrair padrões complexos e representações de alto nível a partir dos dados recebidos. Foram apresentados conceitos acerca do algoritmo DQN, uma das técnicas mais proeminentes nessa interseção, que introduziu a ideia de usar redes neurais para aproximar funções Q e tornar o aprendizado por reforço mais eficiente e estável. Em seguida, exploramos o algoritmo DDPG, utilizado nesta abordagem e que se destaca ao permitir o aprendizado de políticas determinísticas em ambientes de ação contínua. Por fim, foram apresentados conceitos relacionados às principais estruturas de detecção, abordando um problema fundamental em visão computacional que é a detecção de pessoas. Ao unir esses conceitos, percebemos que o aprendizado por reforço profundo e a detecção de pessoas podem se complementar em cenários de robótica avançada, onde um agente pode aprender a navegar em ambientes desconhecidos de forma segura e eficiente.

3 TRABALHOS RELACIONADOS

A seguir serão apresentados e discutidos os trabalhos relacionados a este que foram realizados na área de navegação autônoma em ambientes internos.

Foram utilizados como principal fonte de consultas para esta pesquisa a biblioteca digital IEEEXplore[1], a base de dados SCOPUS[2] e o Portal de Periódicos da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior CAPES[3].

Através da busca por palavras-chave relacionadas ao tema em questão: (*autonomous navigation*, *crowded scenarios*, *crowd navigation*, *collision avoidance in crowd*, *robot navigation in crowded*), foram selecionados trabalhos publicados em importantes periódicos científicos e conferências da área de robótica *International Journal of Robotics Research*, *Autonomous Robots*, *International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, *International Conference on Robotics and Automation (ICRA)* e *International Conference on Intelligent Robots and Systems (IROS)*.

Primeiramente serão apresentados os trabalhos que utilizam abordagens reativas para prevenção de colisões. Nesse contexto destacam-se soluções baseadas em Campo de Potência Artificial, *Velocity Obstacle* e métodos baseados em aprendizado que utilizam estruturas livres de modelos. A seção 3.1.2 apresenta os trabalhos baseados em aprendizado cujas soluções constituem métodos de prevenção de colisões preditivos, abordando soluções baseadas em modelos e soluções baseadas em trajetória. Por fim, são apresentados os trabalhos relacionados à detecção de pessoas.

[1] https://ieeexplore.ieee.org/
[2] https://www.scopus.com/home.uri
[3] https://www.periodicos.capes.gov.br/

O Quadro 1 apresenta uma listagem dos trabalhos que foram estudados para esta pesquisa e as características gerais de acordo com a abordagem proposta.

Tabela 1 – Abordagens Utilizadas e Características Gerais

AUTOR	Método			Base do Modelo			Cenário		ALGORITMOS
	R	P	BA	AR	CH	PT	D	E	
(CHEN et al., 2017)		x	x		x		x		SA-CADRL - Socially Aware CADRL
(GYENES; SZADECZKY-KARDOSS, 2018)		x				x	x		SVO - Safety Velocity Obstacle
(RUAN et al., 2019)	x		x	x				x	Dueling Double DQN
(XUE et al., 2019)	x		x	x				x	Deep Double Q-Learning
(CAO; TRAUTMAN; IBA, 2019)		x				x	x		Timed A*
(SASAKI et al., 2019)		x	x			x	x		A3C - Asynchronous Advantage Actor-Critic
(CHEN et al., 2017)	x		x	x			x		CADRL - Collision Avoidance with Deep Reinforcement Learning
(BRESSON et al., 2019)		x				x	x		DB-SCAN - Density- Based Spatial Clustering of Applications with Noise
(EVERETT; CHEN; HOW, 2018)	x		x	x			x		GA3C-CADRL - GPU/CPU Asynch. Advantage Actor-Critic CADRL
(FERRER; SANFELIU, 2018)		x				x	x		ESFM - Extended Social Force Model RRT - Rapidly-exploring Random Tree
(CHEN et al., 2021)		x				x	x		F-RVO - Frontal RVO DensePeds PPO - Proximal Policy Optimization
(FAN et al., 2018) (LONG et al., 2018)	x		x	x			x		PPO - Proximal Policy Optimization
(FIORINI; SHILLER, 1998) (LORENTE; OWEN; MONTANO, 2018)		x				x	x		VO - Velocity Obstacle
(BAREISS; BERG, 2015) (BERG; LIN; MANOCHA, 2008) (KIM et al., 2015)		x				x	x		RVO - Reciprocal Velocity Obstacle
(DOUTHWAITE; ZHAO; MIHAYLOVA, 2018) (SNAPE et al., 2011) (BERA et al., 2017)		x				x	x		HRVO - Hybrid RVO
(LIU; LIU; WANG, 2017) (QIANG et al., 2018) (YANG; LI, 2017)	x		x	x				x	Q-Learning
(ZHANG et al., 2015) (VAN DEN BERG et al., 2011) (LONG; LIU; PAN, 2017)		x	x	x		x	x		ORCA - Optimal Reciprocal Collision Avoidance
(ALAHI et al., 2016) (LISOTTO; COSCIA; BALLAN, 2019) (GUPTA et al., 2018) (PFEIFFER et al., 2018) (CHOI et al., 2019) (SUN; ZHAI; QIN, 2019)		x	x		x	x	x		LSTM - Long Short-Term Memory
(CIOU et al., 2018) (RIBEIRO et al., 2019) (OKUYAMA; GONSALVES; UPADHAY, 2018) (MOHANTY et al., 2017) (TAI; LI; LIU, 2016) (TAI; LIU, 2016a) (TAI; LIU, 2016b) (WU et al., 2017)	x		x	x	x		x	x	Deep Q - Network
(CHIANG et al., 2015) (MALONE et al., 2017) (WANG; BAN, 2018) (GU et al., 2019) (WU et al., 2015) (LEE et al., 2017)	x					x	x	x	APF - Artificial Potential Field

R: Reativo; P: Preditivo; BA: Baseado em Aprendizado; AR: Aprendizado por Reforço; CH: Comportamento Humano;
PT: Previsão de Trajetórias; D: Dinâmico; E: Estático.

3.1 Prevenção de Colisões

3.1.1 Métodos Reativos

3.1.1.1 Baseados em Campos Potenciais Artificiais

Várias soluções baseadas no algoritmo *Artificial Potential Field (APF)* tem sido propostas na literatura, combinando diferentes técnicas, ou propondo modificações para melhorar o planejamento de caminhos e evitar obstáculos.

A ideia principal por trás do algoritmo APF é criar um campo de força ao redor do robô, onde os obstáculos são representados como fontes de força repulsiva e a meta de navegação é representada como uma fonte de força atrativa. O objetivo é guiar o robô em direção à meta enquanto evita colisões com obstáculos.

O campo potencial artificial é definido como a soma dos campos repulsivos e atrativos. Os campos repulsivos são criados em torno dos obstáculos e aumentam em intensidade à medida que o robô se aproxima deles. Esses campos repulsivos incentivam o robô a se afastar dos obstáculos e evitá-los.

Por outro lado, o campo atrativo é criado em torno da meta de navegação e atrai o robô em direção a ela. O campo atrativo é mais forte perto da meta e diminui à medida que o robô se distancia.

Combinando os campos repulsivos e atrativos, o robô é direcionado para se mover em direção à meta enquanto evita colidir com obstáculos. Isso é feito calculando-se a força resultante do campo potencial e, em seguida, aplicando essa força ao robô para orientar seu movimento.

A figura 3 mostra como o algoritmo APF age ao detectar um obstáculo durante o processo de navegação:

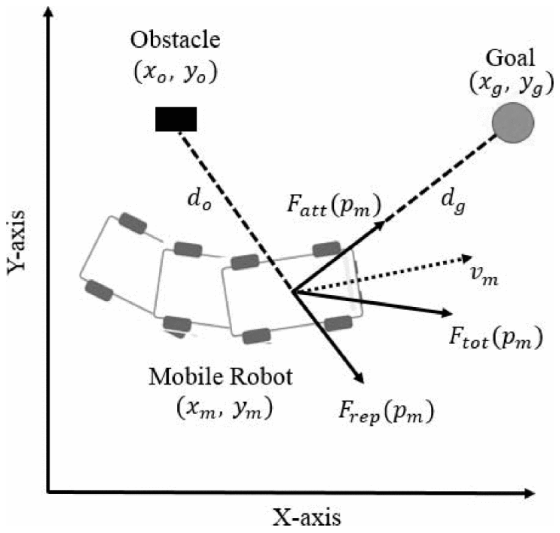


Figura 3 – Artificial Potential Field

Fonte: (LEE et al., 2017)

O Algoritmo APF, proposto por Khatib (1986), é descrito da seguinte forma: campo

potencial atrativo, relacionado ao objetivo $U_{att}(p_m)$ e campo potencial repulsivo, relacionado aos obstáculos $U_{rep}(p_m)$.

O campo potencial total $U_{tot}(p_m)$ é definido conforme a equação 2:

$$U_{tot}(p_m) = U_{att}(p_m) + U_{rep}(p_m) \tag{2}$$

onde:

$p_m = (x_m, y_m)^T$ é o vetor de posição do robô móvel em coordenadas cartesianas. A força é o gradiente negativo do campo potencial, obtida da seguinte forma:

$$F_{tot}(p_m) = -\nabla U_{tot}(p_m) = F_{att}(p_m) + F_{rep}(p_m) \tag{3}$$

onde:

$F_{att}(p_m)$ representa a força atrativa gerada pelo objetivo e $F_{rep}(p_m)$ é a força repulsiva gerada pelos obstáculos. $F_{tot}(p_m)$ é a soma de duas forças, denotando a força total que atua no robô móvel.

onde:

$d_o = (x_o, y_o)^T$ e $d_g = (x_g, y_g)^T$ representam, respectivamente, os vetores de posição em relação a um obstáculo e o objetivo do robô móvel.

A força atrativa e a força repulsiva são descritas conforme a seguir:

$$F_{att}(p_m) = k_{att} \cdot d_g \tag{4}$$

$$F_{rep}(p_m) = \begin{cases} 0, ||d_o|| > d_g \\ -k_{rep}(1/||d_o|| - 1/d_t)(1/(||d_o||)^2)(d_o/||d_o||), ||d_o|| \leq d_g \end{cases} \tag{5}$$

onde:

$d_o = \sqrt{(x_m - x_o)^2 + (y_m - y_o)^2}$: representa a distância do robô móvel a um obstáculo e;

$d_g = \sqrt{(x_m - x_g)^2 + (y_m - y_g)^2}$: representa a distância do robô móvel em relação ao seu objetivo.

Os parâmetros k_{att} e k_{rep} são fatores de escala da força atrativa e da força repulsiva, respectivamente.

O parâmetro d_g utilizado para o cálculo da força repulsiva, representa a distância limite entre o robô móvel e um obstáculo. O vetor p_m é o vetor de velocidade atual do robô móvel.

Modificações no algoritmo APF podem ser adotadas para criar novos pontos de força atrativa e ajudar o robô móvel a escapar dos mínimos locais (WU et al., 2015; LEE et al., 2017). Nesse tipo de solução o robô realiza o percurso utilizando o algoritmo APF tradicional e, ao atingir um mínimo local, uma função baseada no APF

modificado é aplicada para estimular o robô a alterar sua trajetória e, posteriormente, retomar o percurso em direção ao seu objetivo.

Técnicas baseadas em amostragem, para identificar caminhos sem colisões em ambientes dinâmicos, podem ser combinadas com métodos de planejamento de campos potenciais artificiais para navegação em ambientes dinâmicos (CHIANG et al., 2015; MALONE et al., 2017). A solução consiste em calcular um caminho sem colisões, com relação a obstáculos estáticos que é utilizado como um atributo intermediário para atingir o objetivo. Para melhorar a segurança do algoritmo é incorporado um campo potencial repulsivo para cada obstáculo em movimento, tomando-se por base conjuntos estocásticos pré-computados.

Outras soluções buscam predizer trajetórias livres de colisão com base no estado dos obstáculos em movimento. Um modelo de previsão baseado na rede neural de Elman foi projetado por Wang; Ban (2018) para estimar o estado de obstáculos em movimento para a navegação segura de *USVs (Unmanned Surface Vehicles)*. A distância relativa e o tempo de colisão entre o agente móvel e os obstáculos foram utilizados para melhorar o desempenho do algoritmo e evitar o problema de mínimos locais.

Uma modificação na função do campo de repulsão, introduzindo um valor de distância relativa entre o ponto de objetivo e o robô, foi apresentada por Gu et al. (2019). A solução adotou uma combinação do algoritmo APF com o algoritmo *Fuzzy Control*, criando uma função para ampliar a percepção do robô móvel para além da distância com os obstáculos. A medida que o robô se aproxima do objetivo, a força de repulsão é atualizada por um fator de regulação, tendendo a zero, até alcançar o objetivo.

3.1.1.2 Baseados no Método Velocity Obstacle

O método *Velocity Obstacle* (obstáculo de velocidade, em português) baseia-se na ideia de que cada agente tem uma região em torno dele, chamada de "espaço de velocidade segura", na qual ele pode se mover sem colidir com outros agentes. Essa região é definida pelas velocidades relativas permitidas em relação aos outros agentes.

Nessa abordagem, cada agente avalia as velocidades relativas dos outros agentes e determina os obstáculos de velocidade correspondentes. Esses obstáculos são representados geometricamente como regiões no espaço de velocidade. O agente então seleciona uma velocidade de movimento que esteja fora dessas regiões de obstáculo de velocidade, permitindo que ele se mova em segurança.

Uma vez que cada agente tenha escolhido uma velocidade segura, ele pode ser combinado com outras técnicas de planejamento de trajetória para determinar a trajetória final do agente, levando em consideração outros fatores, como metas e restrições do ambiente.

O conceito denominado *Velocity Obstacle (VO)*, baseado na estrutura geométrica

do Cone de Colisão (CC), foi apresentado pela primeira vez por Fiorini; Shiller (1998). Na estrutura proposta os obstáculos são observados no plano horizontal local (XY) do agente, com sua seção transversal plana centrada em $\vec{p}_j$ (Fig. 4):

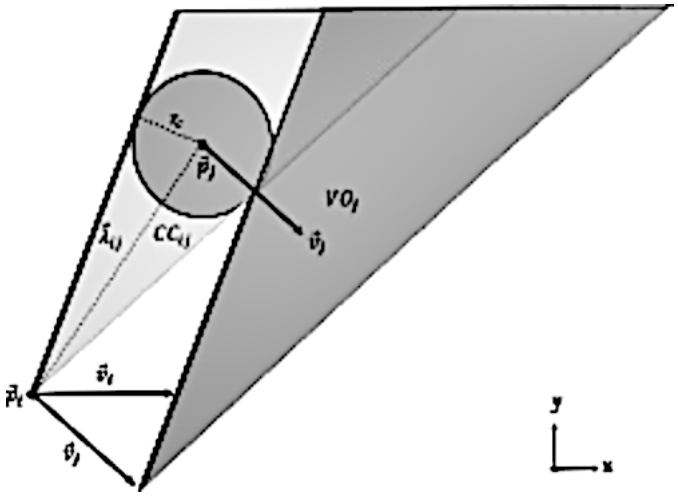


Figura 4 – Velocity Obstacles

Fonte: (DOUTHWAITE; ZHAO; MIHAYLOVA, 2018)

A estrutura geométrica do cone de colisão para o obstáculo j é definido como CC_ij a partir das propriedades da posição relativa dos obstáculos $\vec{\lambda}_ij$, raio de configuração r_c e velocidade $\vec{v}_j$. As velocidades que causam colisão com o obstáculo j são representadas no espaço de velocidade, substituindo CC_ij por $\vec{v}_j$ através da soma de Minkowski (LEE; KIM; ELBER, 1998): $VO_ij = CC_ij \oplus \vec{v}_j$.

Na consideração de múltiplos obstáculos, a união de múltiplos $VO_{1:n}$ é adotada.

As velocidades do agente são consideradas válidas se $\vec{v}_{i,k+1} \notin VO_k = \cup_{j=1}^{n} VO_{j,k}$. As velocidades que satisfazem essa restrição descrevem uma trajetória livre de colisão para o agente i na presença de obstáculos $VO_{j=1:n}$ para o tempo t_k.

O algoritmo *Optimal Reciprocal Collision Avoidance (ORCA)* é uma estratégia descentralizada de prevenção de colisões para vários agentes móveis, baseado no *Velocity Obstacle - VO*.

No ORCA, cada agente produz um obstáculo de velocidade para agentes vizinhos, com base em suas posições e velocidades. A união desses obstáculos de velocidade compõe o espaço de velocidades possíveis que levarão a uma colisão (o cone representado na Fig. 5).

A interpretação geométrica dos obstáculos de velocidade é mostrada na Fig. 5. Observe que $VO^{\gamma}_{A|B}$ e $VO^{\gamma}_{B|A}$ são simétricos na origem. Seja v_A e v_B atuais as velocidades dos robôs A e B, respectivamente. A definição do obstáculo à velocidade implica que, se $v_A - v_B \in VO^{\gamma}_{A|B}$, ou equivalente se $v_B - v_A \in VO^{\gamma}_{B|A}$, A e B colidirão em algum momento antes do tempo γ se continuarem se movendo na velocidade atual. Por outro lado, se $v_A - v_B \notin VO^{\gamma}_{A|B}$, os robôs A e B são livres de colisão por pelo menos γ tempo.

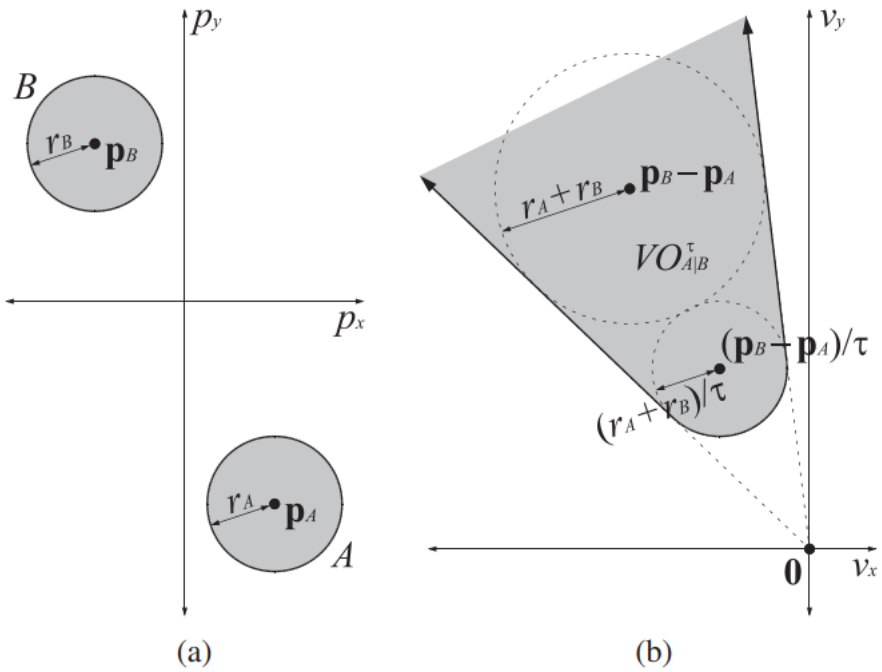


Figura 5 – ORCA - Representação geométrica para dois robôs móveis
Fonte: (VAN DEN BERG et al., 2011)

De maneira mais geral, $X \oplus Y$ denota a soma de Minkowski dos conjuntos X e Y;

$$X \oplus Y = \{x + y | x \in X, y \in Y\}, \tag{6}$$

então, para qualquer conjunto V_B, se $v_B \in V_B$ e $v_A \notin VO^{\gamma}_{A|B} \oplus V_B$, então A e B permanecem livres de colisões nas velocidades atuais, por pelo menos γ. Isso leva à definição do conjunto de velocidades de prevenção de colisões $CA^{\gamma}_{A|B}(V_B)$ para A, dado que B seleciona sua velocidade de V_B, conforme pode ser visto na figura 6:

$$CA^{\gamma}_{A|B}(V_B) = \{v | v \notin VO^{\gamma}_{A|B} \oplus V_B\} \tag{7}$$

Chamamos um par de conjuntos V_A e V_B de velocidades para A e B que evitam colisão reciprocamente se $V_A \subseteq CA^{\gamma}_{A|B}(V_B)$ e $V_B \subseteq CA^{\gamma}_{B|A}(V_A)$.

Se V_A = $CA^{\gamma}_{A|B}(V_B)$ e $VB = CA^{\gamma}_{B|A}(V_A)$, dizemos que V_A e V_B tem reciprocidade máxima.

Dada a configuração do robô da Fig. 5 (a), o conjunto de velocidades para evitar colisões (Fig. 6) $CA^{\gamma}_{A|B}(VB)$ para o robô A, uma vez que o robô B selecione sua velocidade de algum conjunto V_B (cinza escuro), é o complemento da soma de Minkowski (cinza claro) de $VO^{\gamma}_{A|B}$ (ver Fig. 5 (b)) e V_B.

O trabalho de Gyenes; Szadeczky-kardoss (2018) apresentou uma extensão do método VO, objetivando encontrar, não apenas o caminho mais rápido, mas também o caminho mais seguro entre a posição atual do robô e sua posição de destino. Na solução proposta os autores assumem que as velocidades do robô e os obstáculos

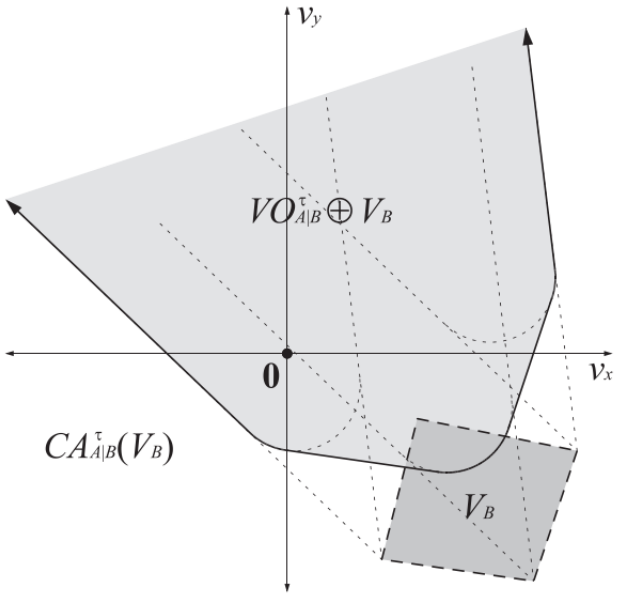


Figura 6 – ORCA - Conjunto de velocidades para evitar colisões
Fonte: (VAN DEN BERG et al., 2011)

são conhecidos ou mensuráveis. O método, denominado *Safety Velocity Obstacle (SVO)*, utiliza um componente do vetor de velocidade para calcular a velocidade mais segura a cada etapa de amostragem. Dependendo da necessidade de aplicação, se o tempo não for a propriedade mais importante, o método pode ser adotado como forma de garantir maior segurança durante a navegação.

Um estudo realizado por Douthwaite; Zhao; Mihaylova (2018) analisou e comparou o desempenho de várias abordagens bem estabelecidas para evitar colisões em sistemas multiagentes não cooperativos: os algoritmos *Velocity Obstacle - VO* (FIORINI; SHILLER, 1998), *Reciprocal Velocity Obstacle - RVO*, (BERG; LIN; MANOCHA, 2008), *Hybrid Reciprocal Velocity Obstacle - HRVO* (SNAPE et al., 2011) e *Optimal Reciprocal Collision Avoidance - ORCA* (VAN DEN BERG et al., 2011) foram estudados em vários cenários com diferentes níveis de dificuldade.

A análise dos resultados demonstrou que os métodos de prevenção de colisões reativos podem ser suficientes para evitar várias colisões em ambientes onde não há comunicação entre agentes. Os métodos HRVO e ORCA demonstraram ser mais eficientes em ambientes densos, ao lidar com incertezas na trajetória. O método ORCA também apresentou trajetórias mais suaves e melhor tempo de computação.

3.1.1.3 Baseados em Aprendizado

O algoritmo *Q-learning*, proposto por C. J. Watkins and P. Dayan (WATKINS; DAYAN, 1992), é um dos algoritmos de aprendizado por reforço mais populares. Sua aplicação tem sido adotada em diversos trabalhos para o planejamento de diferentes modelos de navegação e prevenção de colisões (RIBEIRO et al., 2019; OKUYAMA; GONSALVES; UPADHAY, 2018; MOHANTY et al., 2017).

O *Q-learning* foi aplicado com sucesso em estratégias de controle de navegação utilizando a definição do espaço de estado e o espaço de ação do robô móvel em ambientes de simulação (YANG; LI, 2017). A solução apresentada utilizou a distância do robô ao obstáculo e o ângulo do robô em relação ao seu objetivo, enquanto o processo de aprendizado para a seleção de ações se deu através do mecanismo de seleção de Boltzmann (CESA-BIANCHI et al., 2017), um método clássico para tomada de decisão sequencial sob incerteza.

Um sistema baseado na estrutura *Deep Q-Network (DQN)*, foi utilizado para explorar um ambiente com informações obtidas apenas de um sensor RGB-D. O modelo proposto separa o DQN em duas etapas, constituindo uma estrutura de aprendizado profundo supervisionado e uma rede *Q-learning* (TAI; LIU, 2016a). Inicialmente, os dados de treinamento são pré-processados através de uma rede neural convolucional de três camadas. Na etapa posterior, o mapa de recursos, extraído do modelo de aprendizado supervisionado, é utilizado como entrada de dados para a rede, gerando como saída um conjunto de comandos para determinar a ação do robô.

Uma solução baseada em DRL para navegação autônoma de robôs em ambientes desconhecidos utilizou apenas dados fundidos de um scanner a laser 2D e uma câmera RGB-D para treinar o agente (SURMANN et al., 2020). As ações de saída de uma rede neural, usando o algoritmo Asynchronous Advantage Actor-Critic (GA3C), foram usadas para determinar as velocidades lineares e angulares do robô. A rede do controlador foi pré-treinada em um ambiente de simulação e implantada no robô real. Nos testes realizados, os obstáculos foram representados por combinações de círculos e linhas, e a plataforma do robô foi modelada como uma forma circular simples.

Uma extensão do algoritmo *Double Q-learning* (HASSELT, 2010), denominada *Deep Double Q-Learning (DDQN)*, incluindo redes neurais profundas, foi capaz de retornar estimativas de valor bastante precisas durante a tomada de decisão na tarefa de navegação autônoma. No trabalho de Xue et al. (2019), um planejador de movimento reativo, baseado no DDQN, foi projetado para reduzir o atraso da reação ao detectar uma colisão iminente, apresentando velocidade de resposta superior em comparação com o algoritmo CADRL, reduzindo também o tempo de treinamento em relação ao método *Deep Deterministic Policy Gradient (DDPG)*.

Alguns autores sugerem a utilização de um módulo de memória para armazenar transações de experiência e minimizar o tempo de aprendizado (WU et al., 2017). Além disso, demonstram que o tempo de computação pode ser reduzido por meio da remoção de repetições desnecessárias durante a etapa de treinamento da rede.

No trabalho de Rodríguez-teiles et al. (2014) foi aplicada uma versão aprimorada do algoritmo *Simple Linear Iterative Clustering (SLIC)* para segmentação de imagens e prevenção de colisão com obstáculos em tempo real durante a navegação de veículos subaquáticos autônomos (AUVs), utilizando apenas informações visuais. Depois

disso, um classificador de vizinho mais próximo foi aplicado para separar e detectar objetos na água. A partir da classificação resultante e a direção e orientação atuais do robô, a próxima rota livre de colisão (denominada direção de fuga) pode ser estimada.

Em uma abordagem semelhante, Gaya et al. (2016) utilizaram uma rede neural convolucional, previamente treinada, para estimar mapas de profundidade relativa e obter direções livres de colisão ao detectar obstáculos na trajetória do veículo autônomo. Ao identificar uma colisão iminente a direção é calculada pela maior média de distância em uma área previamente determinada com base nas dimensões do robô e nas características da câmera. Os testes realizados foram bem sucedidos, demonstrando que o método pode ser aplicado para a prevenção reativa de colisões e navegação segura de veículos aéreos não tripulados (UAVs).

Uma solução abordando redes neurais convolucionais para evitar obstáculos durante a navegação de robôs móveis foi proposta por (TAI; LI; LIU, 2016). A partir de imagens de profundidade brutas, recebidas como a única entrada da rede, são gerados os mapas de recursos, responsáveis pelo fornecimento de informações referentes à capacidade de travessia e comandos de controle para determinar as ações do robô. O processo de treinamento foi realizado por um agente humano, responsável por guiar o robô móvel durante a exploração de um ambiente interno desconhecido, sem colidir com obstáculos. O robô aprende as experiências e as adapta a novos ambientes.

Outro método pode ser implementado por meio da criação de modelos independentes de decisão, utilizando simulações offline repetidas. Essa solução foi proposta por Long; Liu; Pan (2017), aplicando o aprendizado por reforço para calcular uma política de prevenção de colisões com base em um modelo de velocidade e direção.

Em Liu; Xu; Chen (2018) foi proposta a utilização de mapas de ocupação local pré-processados, para a construção de uma política de prevenção de obstáculos. Na solução apresentada, conjuntos de dados de treinamento, contendo amostragens do mundo real e mapas gerados artificialmente, foram utilizados para melhorar o desempenho dos algoritmos e reduzir o custo computacional.

3.1.2 Métodos Preditivos

Apesar da variedade de aplicações existentes, alguns trabalhos destacam que técnicas de prevenção de obstáculos puramente reativas não são suficientes para solucionar problemas de navegação em ambientes dinâmicos (LORENTE; OWEN; MONTANO, 2018; FERRER; SANFELIU, 2018). Para os autores, nesse tipo de cenário, o robô deve coexistir ou cooperar com humanos ou outros veículos em movimento.

Outros observam que o planejamento do movimento requer a capacidade de prever a evolução futura dos obstáculos, observando restrições kinodinâmicas para obtenção de trajetórias viáveis, considerando segurança, manobrabilidade, além de restrições do robô e do ambiente (VEMULA; MUELLING; OH, 2017). Além disso, o agente deve

ser capaz de aprender um modelo de interação a partir de dados de trajetória ou comportamento humano real, modelando velocidades de outros agentes na multidão ou por meio da identificação da personalidade variável no tempo de cada pedestre (BERA et al., 2017).

Alguns autores destacam a observância de métodos colaborativos para o cálculo de uma função de custo capaz de modelar o comportamento humano. A função pode ser calculada por meio da utilização do aprendizado por reforço inverso (IRL), através de dados obtidos a partir da interação humano-robô (HADFIELD-MENELL et al., 2016), ou com base na aprendizagem de um modelo de comportamento de navegação cooperativa entre humanos (KRETZSCHMAR et al., 2016).

3.1.2.1 Baseados em Modelos

Uma solução para criação de uma política para prevenção de colisões em um ambiente descentralizado, foi apresentada por Long et al. (2018). Os dados utilizados no planejamento foram coletados apenas de sensores laser e aplicados como entrada em uma rede neural profunda de quatro camadas. Uma extensão do algoritmo de aprendizado por reforço, *Proximal Policy Optimization (PPO)* (SCHULMAN et al., 2017), foi utilizado para a etapa de treinamento e atualização da política de prevenção de colisões com dados recebidos de cada agente móvel pertencente ao cenário. A política foi comparada com outros métodos em vários cenários simulados, demonstrando boa capacidade de generalização em diferentes situações.

No trabalho de Chen et al. (2017), foi implementada uma solução para codificar comportamentos cooperativos, através do aprendizado de uma função de valor, utilizando um algoritmo denominado CADRL (*Collision Avoidance with Deep Reinforcement Learning*). Nos testes realizados o algoritmo mostrou-se eficiente, com aplicação em tempo real, para um sistema descentralizado de dez agentes. Além disso, apesar de não ser um algoritmo voltado para planejamento de caminhos, nos resultados da simulação o CADRL apresentou performance superior em relação ao algoritmo ORCA (*Optimal Reciprocal Collision Avoidance*)(VAN DEN BERG et al., 2011), na qualidade dos caminhos percorridos.

Algumas aplicações utilizam a integração do algoritmo *Long Short-Term Memory (LSTM)* descrito por (HOCHREITER; SCHMIDHUBER, 1997), para refletir as características de memória dos seres humanos e acelerar o aprendizado de políticas para navegação de agentes autônomos (ALAHI et al., 2016; GUPTA et al., 2018).

A construção de uma política de navegação em ambientes dinâmicos desconhecidos, utilizando uma abordagem multiagente, foi apresentada por Sun; Zhai; Qin (2019). A solução aplicou o aprendizado por reforço profundo, combinando o LSTM com o algoritmo *Proximal Policy Optimization (PPO)* e o algoritmo *Reciprocal Velocity Obstacle (RVO)*. O algoritmo PPO, baseado no método ator/crítico, foi empregado

para treinar os agentes a aprender como atingir seus objetivos, enquanto o algoritmo RVO foi utilizado para evitar colisões durante a tarefa de navegação.

Os testes realizados demonstraram que a solução proposta foi capaz de treinar, simultaneamente, multiagentes com diferentes objetivos, obtendo bons resultados durante o planejamento de caminhos e apresentando boa capacidade de autoaprendizado. Além disso, os autores destacam que o LSTM é capaz de lidar com dois problemas comuns que podem ocorrer ao treinar redes neurais recorrentes tradicionais tradicionais: *Vanishing Gradient Problem* e *Exploding Gradient Problem*.

O LSTM é composto por uma célula de memória, um *gate* de entrada, um *gate* de saída e um *gate* de esquecimento. A célula lembra valores em intervalos de tempo arbitrários e os três *gate's* são responsáveis pela regulagem do fluxo de informações.

A arquitetura da unidade LSTM pode ser representada conforme a figura 8. Na qual C_t é o estado da célula, h_t é a saída, f_t é a ativação do *gate* de esquecimento, i_t é a ativação do *gate* de entrada e o_t é a ativação do *gate* de saída.

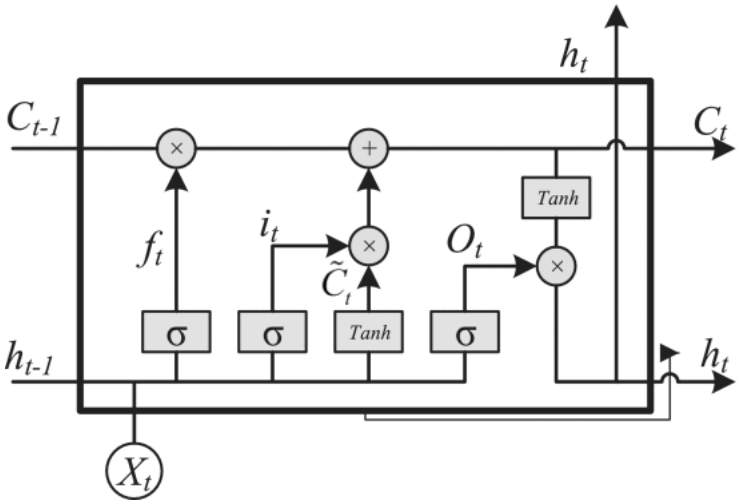


Figura 7 – Long Short-Term Memory

Fonte: (SUN; ZHAI; QIN, 2019)

O *gate* de esquecimento controla até que ponto um valor permanece na célula e sua saída é dada por f_t, que é representada conforme a equação 8:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{8}$$

onde σ é a função sigmoide.

O *gate* de entrada controla até que ponto um novo valor flui para a célula, o que é dado por:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{9}$$

$$C_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{10}$$

O antigo estado da célula (C_{t-1}) é atualizado pelo novo estado da célula C_t:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{11}$$

O *gate* de saída controla até que ponto o valor na célula é usado para calcular a ativação de saída da unidade LSTM, que é dado por:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{12}$$

$$h_t = o_t * \tanh(C_t) \tag{13}$$

Uma abordagem orientada a dados para modelar a interação e o movimento de pedestres, baseada em redes neurais LSTM foi apresentada por Pfeiffer et al. (2018). Na solução proposta o problema de previsão é tratado como uma tarefa de modelagem de sequência que funde três canais de informação por pedestre: velocidade atual do pedestre, informações sobre os obstáculos estáticos ao redor do pedestre e informações sobre os pedestres ao redor. A abordagem proposta foi capaz de prever interações entre pedestres e evitar obstáculos estáticos e dinâmicos ao mesmo tempo, superando outras soluções eficientes nos testes realizados.

Sabe-se também que o grande desafio para evitar colisões em ambientes dinâmicos é que o número de outros agentes é variável. Dessa forma, algumas abordagens tem buscado soluções para aprender uma política de prevenção de colisões sem prever um modelo comportamental de outros agentes.

A exemplo disso, em Everett; Chen; How (2018) foi apresentado um algoritmo de prevenção de colisões denominado GA3C-CADRL. A rede foi treinada em simulação com DRL sem assumir que outros agentes sigam qualquer modelo comportamental específico. A solução proposta utilizou o algoritmo *Long Short Term Memory (LSTM)* na entrada da rede, combinada com uma extensão do algoritmo GA3C, para aprender com as experiências de vários agentes a cada episódio de treinamento. A estratégia permite que o algoritmo selecione as ações a serem realizadas com base nas observações de um número arbitrário de agentes próximos.

3.1.2.2 Baseados em trajetórias

Diferente dos métodos reativos, métodos baseados em trajetória objetivam antecipar o movimento de outros agentes para prever a evolução futura dos estados conjuntos (caminhos, agente e vizinhos).

Estruturas modernas de aprendizado por reforço profundo e interação humano-robô (HRI) podem ser utilizadas para codificar o conhecimento prévio do ser humano (CIOU et al., 2018). O objetivo é introduzir características comportamentais cooperativas às ações do robô, reforçando a segurança da tarefa de navegação autônoma.

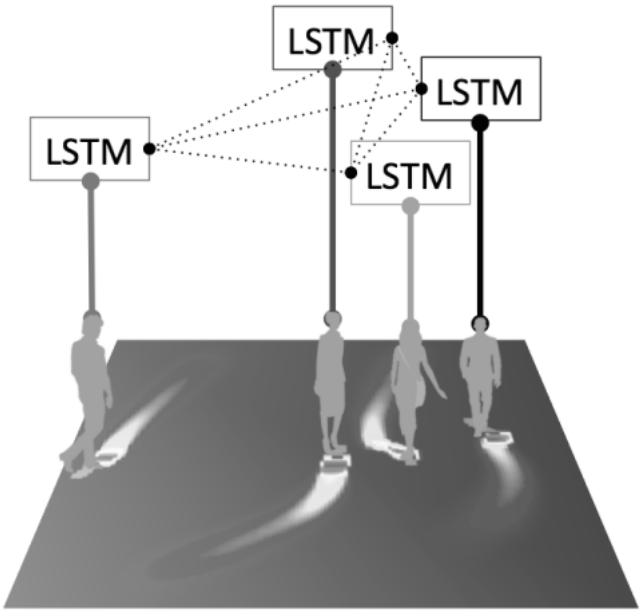


Figura 8 – Previsão de trajetórias
Fonte: (ALAHI et al., 2016)

Esses comportamentos podem ser usados para previsão de caminhos, assim como para calcular distâncias proxêmicas para navegação social e evitar colisões.

Alguns trabalhos buscam classificar automaticamente o comportamento dinâmico ou a personalidade de pedestres com base em seus movimentos na multidão. O trabalho de Bera; Randhavane; Manocha (2017) baseou-se na teoria dos traços de personalidade para aprender dinamicamente o comportamento de todos os pedestres em uma determinada cena e calcular um modelo de movimento para um pedestre. Esse modelo é combinado com as características globais da multidão e utilizados para calcular os padrões e a dinâmica do movimento, que também podem ser aplicados para prever o movimento e o comportamento da multidão.

Um método, baseado em agente, para previsão de trajetórias de pedestres utilizando o raciocínio velocidade-espaço, foi apresentado por Kim et al. (2015). A técnica proposta não depende de conhecimento prévio do ambiente e foi capaz de aprender modelos de movimento para prever trajetórias futuras. Segundo os autores, a solução pode ser integrada a outras técnicas de navegação local para melhorar as taxas de conclusão de tarefas e reduzir instâncias relacionadas ao problema de congelamento do robô.

Em Mujahed; Mertsching (2017) é apresentada uma solução para evitar obstáculos em tempo real, denominada *Admissible Gap (AG)*. Na solução proposta um intervalo é admissível se for possível encontrar um comando de movimento, que uma vez executado, o robô seja capaz de atravessar com segurança por esse intervalo, respeitando suas restrições de forma e movimento. A abordagem proposta considera diretamente a forma do robô e as restrições cinemáticas. A ideia básica consiste em descobrir um

conjunto de espaços visíveis a partir da localização atual do robô e selecionar aquele que estiver mais próximo ao seu objetivo. Os experimentos realizados demostraram que a abordagem AG é capaz de gerar trajetórias eficientes, no entanto não foram realizados teste em ambientes com obstáculos em movimento.

3.1.3 Detecção de Pessoas

Um sistema de aprendizado online de classificadores humanos por robôs de serviços móveis com sensores 3D LiDAR, utilizou um algoritmo de agrupamento em tempo real para segmentação de dados de nuvem de pontos 3D e obteve bons resultados durante a navegação com pessoas se movendo em um grande espaço público interno (YAN; DUCKETT; BELLOTTO, 2020). O sistema utilizou perfis humanos em nuvens de pontos com mudanças de distância, aumentando a sensibilidade do classificador para amostras distantes do robô. Segundo os autores o agrupamento de profundidade é um método rápido e com baixa demanda computacional. Além disso, destacam que a detecção humana pode ser melhorada combinando rastreamento e aprendizado online com um robô móvel, mesmo em ambientes altamente dinâmicos, e que tal abordagem fornece resultados comparáveis ou superiores em relação à métodos anteriores.

Um método para detecção de pernas, baseado em Filtros de Kalman, foi capaz de rastrear pessoas e seus movimentos, evitando obstáculos durante a navegação (ADIWAHONO et al., 2017). A abordagem proposta utilizou um scanner a laser 2D, alimentado por um algoritmo de rastreamento e duas camadas de filtro de Kalman para garantir a robustez do rastreamento, mesmo diante da proximidade de outras pessoas e da perda temporária da linha de visão para os grupos de pernas. Segundo os autores a solução não requer informações prévias de mapas e mostrou-se eficiente em qualquer condição de iluminação.

Um sistema para detecção e rastreamento de pessoas baseado em uma rede neural convolucional, treinada *offline*, foi capaz de rastrear pares de pernas em um ambiente desordenado, utilizando como entrada um mapa de ocupação construído a partir de medições do sensor LiDAR (GUERRERO-HIGUERAS et al., 2019). Segundo os autores, os resultados obtidos apresentaram melhor precisão do que o *Leg Detector* (LD), a solução padrão para robôs baseados no *Robot Operating System* (ROS)[4].

Um estudo comparativo sobre cinco algoritmos independentes baseados em aprendizado profundo (R-FCN, Mask R-CNN, SSD, RetinaNet, YOLOv4) para detecção de objetos rodoviários, utilizou o conjunto de dados BDD100K para treinar, validar e testar os modelos individuais de aprendizado profundo para detectar quatro objetos de estrada: veículos, pedestres, sinais de trânsito e semáforos (HARIS; GLOWACZ, 2021). Seus pontos fortes e limitações foram analisados com base em parâmetros como precisão (com/sem oclusão e truncamento), tempo de computação e curva de

[4] http://www.ros.org/

recuperação de precisão. Os resultados experimentais mostraram que o YOLOv4 no modelo de detecção de um estágio atinge a maior precisão de detecção para detecção de alvo em todos os níveis, enquanto no modelo de detecção de dois estágios, Mask R-CNN mostrou melhor precisão de detecção sobre RetinaNet, R-FCN, e SDD.

Um sistema baseado em aprendizagem profunda, com detector de distância para cadeira de rodas e andador, foi estendido para a tarefa de detecção de pessoas, constituindo, segundo os autores, o maior conjunto de dados publicamente disponível para detecção de pessoas em dados de alcance 2D (BEYER; HERMANS; LEIBE, 2017; BEYER et al., 2018). A abordagem consistiu em três etapas: pré-processamento, que corta uma janela reamostrada em torno de cada ponto do laser e calcula os locais de detecção em um sistema de coordenadas local; uma CNN que classifica essas janelas e prevê os locais de detecção relativos e; finalmente, um esquema de votação e supressão não máxima transformando previsões em detecções. A análise dos resultados demonstrou que o pré-processamento de profundidade e o esquema de votação adotados, permitem que as CNNs superem amplamente as linhas de base de detecção de CNN ingênuas e obtenham resultados de última geração em comparação à métodos anteriores.

3.2 Considerações finais

Este capítulo apresentou os trabalhos relacionados ao tema desta pesquisa. Desses, 55% abordam o uso de métodos reativos para prevenção de colisões durante a tarefa de navegação e 45% utilizam métodos preditivos. Destaca-se também um grande avanço do uso de técnicas baseadas em aprendizado, especialmente após o ano de 2017, correspondendo à 61% dos trabalhos estudados.

Percebe-se também um significativo avanço nas pesquisas voltadas para a navegação social, abordando o uso do aprendizado por reforço e métodos de análise comportamental de seres humanos. Esses trabalhos reforçam a necessidade da adoção de políticas para prevenção de colisões e combinação de técnicas que proporcionem uma navegação eficiente, sem colocar em risco as pessoas em torno do robô.

Com relação à detecção de pessoas, percebe-se que soluções baseadas na identificação de pernas podem enfrentar alguns desafios e problemas específicos, que podem afetar a precisão e o desempenho geral do sistema. Em cenários populosos, as pernas das pessoas podem ser parcialmente ocultas por objetos ou outras pessoas, resultando em falsos negativos ou detecções imprecisas. Além disso, pernas podem aparecer em diferentes escalas e distâncias da câmera, o que requer técnicas de detecção robustas que possam lidar com essas variações. Nesse contexto, soluções baseadas em arquiteturas de detecção de estágio único têm apresentado bons resultados, emergindo como soluções eficientes e que exigem menor custo computa-

cional para aplicação ao problema em questão.

Apesar da quantidade de trabalhos existentes, a revisão da literatura acerca do tema demonstra que a navegação de robôs autônomos em cenários compartilhados com humanos permanece como um problema em evidência. A maioria das abordagens desconsideram que o número de agentes é uma variável dinâmica, ou consideram que a trajetória ou velocidade dos demais agentes são conhecidas. Além disso, a maioria das soluções propostas consideram as pessoas como simples obstáculos móveis, o que pode afetar a segurança da navegação em aplicações do mundo real.

Alguns autores destacam que nesse tipo de cenário o robô deve ter a capacidade de navegar sem o conhecimento prévio do ambiente ou das ações dos demais agentes e obstáculos. Dessa forma, a análise dos trabalhos relacionados ao tema desta pesquisa evidencia que, para a navegação autônoma em ambientes dinâmicos, compartilhados com humanos, alguns pontos essenciais devem ser observados: a) A necessidade de uma política de prevenção de colisões, por meio de comportamentos reativos ou preditivos; b) O cálculo de uma trajetória viável, rápido e capaz de alterar a trajetória do robô, de acordo com informações atualizadas do ambiente e; c) A capacidade de identificar pessoas para melhorar a tomada de decisão e realizar a tarefa de navegação de forma mais segura.

4 ABORDAGEM PROPOSTA

Aplicar o aprendizado por reforço em robótica móvel requer um ambiente de simulação, definição adequada de estado, ações e recompensas, seleção de algoritmos de aprendizado, etapas de treinamento, além de avaliação contínua do desempenho do agente.

Neste trabalho foi utilizado o ambiente desenvolvido pela Robotis[1] para o treinamento e teste do agente em ambientes simulados. O ambiente utilizado tem como principal característica a fácil integração com o ROS e o simulador Gazebo, permitindo a implementação de estruturas de aprendizado por reforço e testes em diferentes cenários.

Na solução proposta, o algoritmo *Deep Deterministic Policy Gradient* (DDPG), originalmente proposto para resolver o problema do pêndulo invertido[2], foi combinado com técnicas de visão computacional para detecção de pessoas, resultando em uma abordagem para navegação autônoma em cenários complexos, doravante denominada Social Attention Navigation - DDPG (SAN-DDPG). Para validação da solução proposta foram conduzidos testes comparativos utilizando o *Deep Q-Network* (DQN), um algoritmo amplamente utilizado para navegação autônoma em ambientes internos.

Este capítulo apresenta de forma detalhada a abordagem proposta e está organizado da seguinte forma: na seção 4.1 são descritos os recursos de software e hardware utilizados; a seção 4.2 apresenta a plataforma robótica simulada para a realização dos experimentos e aplicação dos testes; a seção 4.3 apresenta o sistema utilizado para detecção de pessoas; a seção 4.4 descreve o algoritmo de aprendizado utilizado nesta abordagem; a seção 4.5 descreve a arquitetura do ambiente de aprendizado; por fim, a seção 4.6 apresenta os ambientes de treinamento, desenvolvidos no simulador Gazebo para aplicação dos experimentos e análise dos resultados.

[1] https://emanual.robotis.com/docs/en/platform/turtlebot3/machine_learning

[2] https://blog.paperspace.com/physics-control-tasks-with-deep-reinforcement-learning/

4.1 Ambiente de simulação

4.1.1 Software

As principais ferramentas de software utilizadas neste trabalho foram o ROS (Melodic Morenia[3]) e o simulador Gazebo.

O ROS é uma estrutura de software de código aberto para programação de robôs que fornece uma camada de abstração na qual os desenvolvedores podem criar aplicativos de robótica sem se preocupar com a camada de hardware (KOUBÂA et al., 2017). O software é organizado em pacotes oferecendo boa modularidade e reusabilidade, além de disponibilizar diferentes ferramentas de software para visualizar e depurar dados do robô. O núcleo do *framework* ROS é um *middleware* de passagem de mensagens no qual os processos podem se comunicar e trocar dados entre si, localmente ou em rede.

O projeto ROS foi iniciado em 2007 na Universidade de Stanford sob o nome Switchyard. Mais tarde, em 2008, o desenvolvimento foi realizado por uma *start-up* de pesquisa robótica chamada Willow Garage. O grande desenvolvimento no ROS aconteceu na Willow Garage. Em 2013, os pesquisadores da Willow Garage formaram a *Open Source Robotics Foundation* (OSRF)[4], responsável por manter o sistema até os dias atuais.

Gazebo[5] é um simulador de robôs 3D de código aberto, originalmente desenvolvido pela OSRF, tornando-se parte do projeto ROS, posteriormente.

O Gazebo permite simular ambientes complexos e interações entre robôs, sensores, atuadores e objetos em um ambiente virtual bastante realista. Entre as principais características do simulador, destacam-se:

- Modelagem de Robôs: é possível modelar robôs com detalhes precisos, incluindo geometria, cinemática, dinâmica e sensores. O Gazebo permite a criação de modelos baseados em robôs reais ou a criação de modelos personalizados para fins de simulação;
- Ambientes 3D: O simulador oferece uma representação 3D detalhada dos ambientes virtuais, permitindo a criação de cenários realistas para testar e avaliar o comportamento de robôs em diferentes cenários;
- Simulação Física: O simulador oferece um mecanismo de simulação física que inclui colisões, forças, atrito e outras propriedades físicas, permitindo que os robôs se movam e interajam com o ambiente de forma mais realista;

[3] http://wiki.ros.org/melodic
[4] https://www.openrobotics.org/
[5] http://gazebosim.org/

- Sensores e Atuadores: Gazebo oferece suporte a uma variedade de sensores e atuadores, como câmeras, sensor laser LiDAR (Light Detection and Ranging), sensores de proximidade, juntas articuladas, rodas, entre outros, permitindo simular facilmente o funcionamento de sistemas de percepção e controle dos robôs;
- Integração com ROS: Sua integração com o ROS permite combinar a simulação com o desenvolvimento e teste de algoritmos de controle e sistemas de percepção em um ambiente estável e de fácil configuração.

4.1.2 Hardware

Para realização dos experimentos foi utilizado um computador desktop com 16Gb de memória RAM, SSD de 500Gb, Processador 11th Gen Intel® Core™ i5-11400 2.60GHz com 06 núcleos, Sistema operacional Ubuntu 18.04.6 LTS (Bionic Beaver) 64 bits e GPU NVIDIA TITAN de 12GB.

4.2 Plataforma Robótica

A pesquisa relacionada a este trabalho teve início utilizando a plataforma robótica *The Home-Environment Technological-Agent* (Theta), um robô de serviço projetado para ser um sistema autônomo capaz de resolver tarefas domésticas.

A base do Theta é composta por uma cadeira de rodas, adaptada pela Freedom Veículos Elétricos LTDA. Sua estrutura foi modificada e equipada com câmera Kinect na parte superior para realização de visão computacional, microfone e alto-falante para interação humano-robô, sensor LiDAR (*Light Detection & Ranging*) para mapeamento e localização, hodômetros para realização de movimentos e um monitor de vídeo para representação de expressões faciais (Figura 9).

Figura 9 – *The Home-Environment Technological-Agent* (Theta)

Fonte: Autor

O Theta possui vários sensores que auxiliam nas tarefas domésticas e foi construído para atender alguns dos desafios clássicos do LARC/CBR @Home. O projeto é continuamente aprimorado, visando obter melhor desempenho nas tarefas, dentre as quais destaca-se a navegação autônoma em ambientes dinâmicos, compartilhados com humanos.

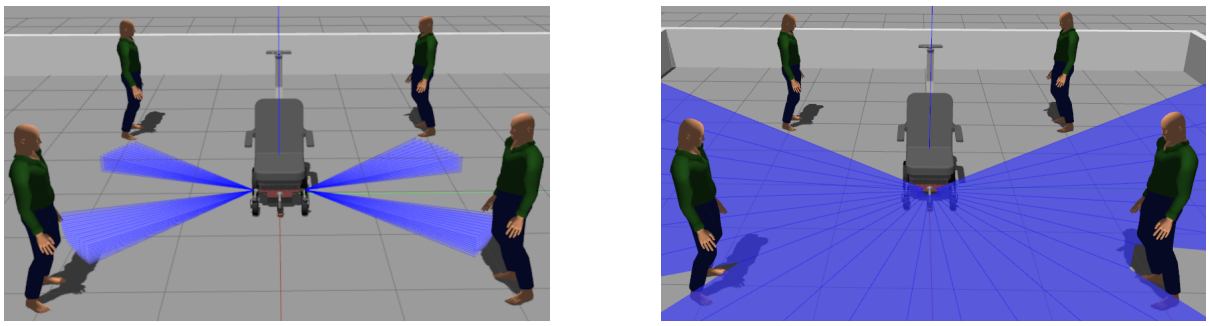
Devido à pandemia de Covid-19 as atividades de pesquisa em ambiente real foram suspensas. Dessa forma, os experimentos e testes passaram a ser realizados exclusivamente em ambiente simulado.

Assim, a plataforma robótica utilizada neste trabalho é um modelo simulado de uma cadeira de rodas motorizada. O objetivo é realizar um estudo de caso para avaliar a viabilidade de aplicação da solução proposta ao problema em questão.

Figura 10 – Simulação da cadeira de rodas

Fonte: Autor

A realização de experimentos e testes aplicados ao modelo específico permitirá uma análise mais detalhada do comportamento do robô, propiciando identificar problemas e ajustes necessários para uma navegação mais eficiente e segura.



(a) Sensores Infravermelhos (b) Sensor LiDAR

Figura 11 – Sensores infravermelhos e sensor LiDAR

Fonte: Autor

O modelo de cadeira de rodas 3D usado na simulação foi adaptado do projeto desenvolvido por Patil (2021). A versão original, com tração dianteira, foi adaptada para uma cadeira de rodas com tração traseira. Essa modificação foi necessária para implementação de um modelo mais próximo ao Theta.

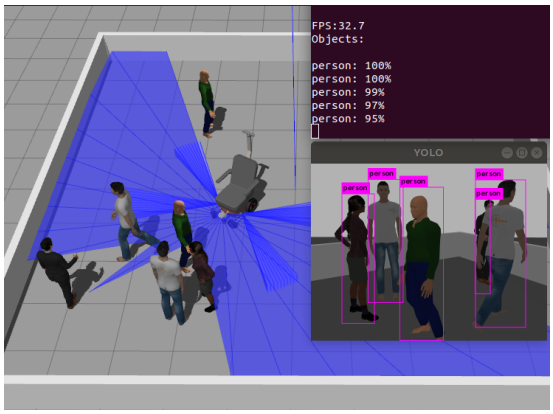
Além da modificação na tração, foi necessário realizar ajustes no sensor laser devido à movimentação do caster e das rodas dianteiras. Também foram adicionados quatro sensores infravermelhos, dois na lateral esquerda e dois na lateral direita da cadeira, para detecção de colisões durante as etapas de treinamento (Fig. 11a).

Os sensores infravermelhos foram utilizados apenas para detecção de colisões com obstáculos ou com as paredes em torno do robô. Os dados de distância do robô em relação aos objetos que compõem o ambiente são obtidos do sensor LiDAR (Fig. 11b) e compõem as informações de estado para o algoritmo de aprendizado.

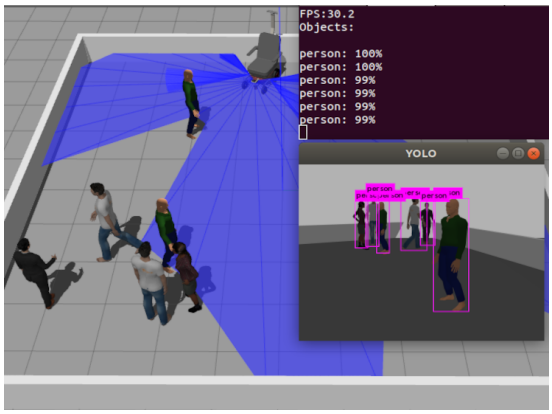
4.3 Detecção de pessoas

Para a detecção de pessoas, foi adotado o DarknetRos(BJELONIC, 2016–2018), um pacote disponível no ROS, projetado para detecção de objetos em imagens de câmera. O DarknetRos utiliza o You Only Look Once (YOLO), um sistema de visão computacional que pode identificar com precisão múltiplos objetos em uma única imagem, comparável ao RetinaNet, porém com velocidade de inferência mais rápida em comparação com outros sistemas avançados como SSD, R-FCN (*Region-based Fully Convolutional Networks*) e FPN (*Feature pyramid networks*) FRCN (*Faster Region based convolutional neural networks*) (REYES et al., 2019). Sua velocidade o torna altamente adequado para detecção de objetos em tempo real, o que é essencial em sistemas como robôs de serviço.

As imagens são capturadas usando um sensor Kinect acoplado a uma montagem no encosto da cadeira de rodas (Fig. 12a e 12b).



(a) Detecção de pessoas próximas



(b) Detecção de pessoas distantes

Figura 12 – Detecção de pessoas

Fonte: Autor

Nos testes realizados, o DarknetRos processou as imagens a mais de 30 FPS (*Frames per Second*) e alcançou uma precisão de 90% a 100% na detecção de pessoas no ambiente simulado.

4.4 Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient (DDPG) é um algoritmo proposto por Lillicrap et al. (2015) que utiliza uma arquitetura *actor-critic* baseada no algoritmo *Deterministic Policy Gradient* (DPG) (SILVER et al., 2014), livre de modelo e *off-policy*, com aproximadores de função profunda capazes de aprender políticas em espaços de ação contínuos e de alta dimensão.

Uma das principais características do DDPG é a sua abordagem baseada em políticas, que estima diretamente uma política determinística mapeando estados para ações. Essa política é representada por uma rede neural, conhecida como ator, que é treinada para fornecer a melhor ação possível para cada estado.

O pseudocódigo a seguir descreve o algoritmo DDPG, conforme proposto por Lillicrap et al. (2015):

Algorithm 2 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for $episode = 1, M$ **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for $t = 1, T$ **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_i + 1, \mu'(s_i + 1|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|s = s_i, a = \mu(s_i) \nabla_{\theta^\mu} \mu(s|\theta^\mu)|s_i$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1 - \tau)\theta^{Q'}$$

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1 - \tau)\theta^{\mu'}$$

  end for
end for

A função de ator parametrizada $\mu(s|\theta\mu)$, especifica a política atual mapeando de-

terministicamente estados para uma ação específica. Nesse caso, a saída da política de rede é um valor que corresponde à ação a ser executada no ambiente.

Além do ator, o DDPG também utiliza uma rede neural chamada crítico, que estima o valor Q para o estado atual e ação selecionada pelo ator. O crítico $Q(s, a)$ é aprendido utilizando a equação de Bellman como no *Q-learning*. O ator é atualizado seguindo a regra de cadeia aplicada ao retorno esperado da distribuição inicial J em relação aos parâmetros do ator.

Segundo os autores, um desafio ao usar redes neurais para aprendizado por reforço é que a maioria dos algoritmos de otimização assume que as amostras são distribuídas de forma independente e idêntica. No entanto, quando as amostras são geradas a partir da exploração sequencial em um ambiente, essa suposição não é mais válida. Além disso, para fazer uso eficiente das otimizações de hardware, os autores destacam que é essencial aprender em minilotes, em vez de online.

O DDPG utiliza um *buffer* de repetição para resolver esses problemas. O *buffer* de repetição é um cache R de tamanho finito. As transições são amostras obtidas do ambiente, de acordo com a política de exploração, resultando em uma tupla (s_t, a_t, r_t, s_{t+1}) armazenada em R. Quando o *buffer* está cheio, as amostras mais antigas são descartadas. A cada passo de tempo, o ator e o crítico são atualizados amostrando um minilote uniformemente de R. Como o DDPG é um algoritmo fora da política, o espaço de R pode ser grande, permitindo que o algoritmo se beneficie do aprendizado em um conjunto de transições não correlacionadas.

Outra vantagem do DDPG é que ele utiliza atualizações de destino “suaves” em vez de copiar diretamente os pesos da rede. Para calcular os valores de destino são criadas cópias das redes ator $Q(s, a|\theta^Q)$ e crítico $\mu(s|\theta^\mu)$, respectivamente. Os pesos dessas redes de destino são então atualizados fazendo com que rastreiem lentamente as redes aprendidas: $\theta' \leftarrow \tau\theta + (1 - \tau)\theta'$ onde $\tau \ll 1$. Essa característica proporciona que os valores de destino mudem lentamente, melhorando a estabilidade do aprendizado.

Para os autores um grande desafio da aprendizagem em espaços de ação contínua é a exploração. Uma vantagem dos algoritmos *off-policies* como o DDPG é a possibilidade de tratar o problema de exploração independentemente do algoritmo de aprendizado. No DDPG a política de exploração μ' é construída adicionando ruído à um processo N da política do ator: $\mu'(s_t) = \mu(s_t|\theta_t^\mu) + \mathcal{N}$

O DDPG também utiliza a técnica de gradiente ascendente para ajustar os parâmetros do ator e do crítico. O ator é atualizado procurando maximizar o valor Q estimado pelo crítico para a ação selecionada.

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|s = s_i, a = \mu(s_i) \nabla_{\theta^\mu} \mu(s|\theta^\mu)|s_i \tag{14}$$

O crítico, por sua vez, é atualizado procurando minimizar a diferença entre o valor Q estimado e o valor Q real calculado com base na recompensa obtida e nos valores Q do próximo estado.

$$L = \frac{1}{N}\sum_{i}(y_i - Q(s_i, a_i|\theta^Q))^2 \tag{15}$$

O objetivo é melhorar a medida de desempenho J para acompanhar a maximização da função de valor Q, minimizando a perda de diferença temporal como aconteceu com o Deep Q-Network para jogos de Atari.

Uma diferença fundamental entre o DDPG e o DQN é o uso de uma política determinística em vez de uma política estocástica. Enquanto o DQN lida com ações incertas e estocásticas, o DDPG assume que há uma ação determinística ótima para cada estado. Isso simplifica o processo de tomada de decisão, permitindo que o agente aprenda diretamente uma política determinística sem a necessidade de exploração estocástica.

4.5 Configuração do Ambiente

Da mesma forma que em abordagens tradicionais de RL, o agente autônomo proposto aqui, controlado pelo algoritmo DDPG, observa um estado s_t do ambiente em um determinado momento t do tempo. Durante a interação entre o agente e o ambiente, ao executar uma ação no estado s_t, ambos fazem uma transição para um novo estado s_{t+1}.

O estado é representado por uma estatística do ambiente e deve conter as informações necessárias para que o agente tome a melhor ação.

A sequência ótima de ações é determinada pelas recompensas fornecidas pelo ambiente. Ao fazer a transição para um novo estado, o ambiente fornece uma recompensa escalar r_{t+1} ao agente como *feedback*.

As ações do sistema de aprendizado têm influência nas próximas entradas e podem afetar não apenas a recompensa imediata, mas também a situação subsequente e, consequentemente, todas as recompensas subsequentes.

A abordagem consiste em um problema sequencial de tomada de decisão, que pode ser formulado como um Processo de Decisão de Markov (MDP), descrito por Xue et al. (2019) da seguinte forma: $M = (S, A, R, P, \gamma)$, onde S é o espaço de estados, A é o espaço de ação, R é a função de recompensa, P é o modelo de transição de estado e γ é um fator de desconto. Os principais elementos que compõem o ambiente aqui representado utilizam os parâmetros de configuração adotados pela Robotis, conforme descrito a seguir:

4.5.1 Espaço de estados

O estado representa as informações sobre o ambiente que são relevantes para o agente. Refere-se a uma representação do ambiente em um determinado instante de tempo, capturando as informações necessárias para a tomada de decisão e determinação das próximas ações.

A configuração do ambiente utilizado neste trabalho compreende as seguintes informações:

- Odometria: Utilizada para retornar os valores relacionados à posição e orientação (pose) do robô no ambiente, além de fornecer dados para representação de orientação em um conjunto de ângulos de Euler para o cálculo do ângulo de movimento angular do robô (*yaw*).

- Sensoriamento: neste trabalho são utilizadas 24 amostras do sensor LiDAR. Os dados coletados são utilizados para o cálculo da distância entre o robô e os obstáculos em seu entorno e também são fornecidos como dados de entrada para o algoritmo de aprendizado, juntamente com outras informações obtidas do ambiente.

- Ângulo do robô em relação ao objetivo: Corresponde à diferença entre dois ângulos, o ângulo de destino e o ângulo de guinada do robô.

$$\theta = \theta_{\text{goal}} - \theta_{\text{yaw}}$$

  onde:

  θ representa o ângulo de orientação, θ_{goal} representa o ângulo de destino desejado e θ_{yaw} representa o ângulo de guinada atual. O resultado corresponde a diferença angular necessária para que o agente alcance o ângulo desejado, retornando a direção a ser seguida para alinhar-se com o ângulo do objetivo. Por exemplo, se o valor de θ for positivo, significa que o agente precisa girar no sentido horário, enquanto um valor negativo indica a necessidade de girar no sentido anti-horário.

- Distância atual até o objetivo: Utilizado para calcular a distância euclidiana entre a posição atual do robô e a posição do objetivo. Esse valor é utilizado no cálculo da recompensa, servindo para estimular o agente a seguir em direção ao objetivo e também para identificar quando o agente atinge o objetivo.

$$d_{\text{c}} = \sqrt{(x_{\text{g}} - x_{\text{p}})^2 + (y_{\text{g}} - y_{\text{p}})^2}$$

  onde:

x_{g} e y_{g} são as coordenadas x e y do objetivo desejado, respectivamente. x_{p} e y_{p} são as coordenadas x e y da posição atual do robô.

- Menor distância até um obstáculo: Corresponde ao valor mínimo encontrado em um conjunto de leituras do sensor LiDAR. É utilizado para determinar a distância entre o robô e o obstáculo mais próximo, servindo também para estimular o agente a manter uma distância segura dos obstáculos durante a navegação. Na configuração adotada para este trabalho, o agente é penalizado ao manter uma distância menor do que um limiar pré-estabelecido.

$$r_{\text{min}} = \min(\mathbf{r})$$

onde:

r_{min} representa a menor distância encontrada no conjunto de leituras do sensor LiDAR, representado por $\mathbf{r}$.

- Ângulo do obstáculo mais próximo: Corresponde ao índice do menor valor em um conjunto de leituras do sensor LiDAR. É utilizado para determinar o ângulo do obstáculo mais próximo em relação ao robô.

$$\theta_{\text{obs}} = \arg\min(\mathbf{r})$$

onde:

θ_{obs} representa o ângulo no qual a menor distância é encontrada no conjunto de leituras do sensor LiDAR, representado por $\mathbf{r}$.

- Detecção de pessoas: Corresponde ao número de pessoas detectadas em um determinado instante de tempo durante a navegação. Essa informação é utilizada em conjunto com a posição dos obstáculos (ângulo e distância) para o cálculo da recompensa, objetivando estimular um comportamento seguro durante a navegação.

- Distância de colisão: Utilizada para identificar quando o robô efetua uma colisão. Essa informação é obtida a partir da leitura dos dados fornecidos pelo sensor LiDAR e dos sensores infravermelhos. Uma colisão é identificada quando os sensores detectam uma distância menor do que um limiar pré-estabelecido.

4.5.2 Espaço de ação

O espaço de ação define as ações que o robô pode executar em um determinado momento. As ações representam os movimentos que o robô irá executar com base

em seu estado atual no ambiente. Elas correspondem às velocidades angulares do robô e são determinadas pelo algoritmo de aprendizado por reforço (Tabela 2).

Tabela 2 – Ações

Ação	Velocidade angular (rad/s)
0	-1.5
1	-0.75
2	0
3	0.75
4	1.5

Se a ação retornada pelo algoritmo for 2, a velocidade angular será 0. Portanto, o robô se moverá em linha reta de acordo com a velocidade linear configurada para o ambiente. Caso contrário, assumirá velocidades angulares, que podem ser negativas (esquerda) ou positivas (direita).

Quando o agente realiza uma ação em um determinado estado, ele recebe uma recompensa que pode ser negativa ou positiva.

Na configuração adotada por este trabalho, quando o agente se aproxima de uma certa distância do objetivo desejado, a recompensa acumulada é positiva, e quando atinge o objetivo o agente recebe uma recompensa maior. Da mesma forma, se o agente se afasta do objetivo desejado, a recompensa acumulada é negativa, e ele recebe uma recompensa negativa significativa por colidir com um obstáculo.

Nas etapas de treinamento com pessoas paradas e em movimento, o agente recebe uma recompensa negativa sempre que sua distância para uma pessoa detectada for menor que a distância mínima especificada. O objetivo é manter uma distância segura entre o robô e as pessoas durante a tarefa de navegação.

As configurações foram ajustadas de acordo com os objetivos de cada etapa de treinamento, visando avaliar a capacidade de aprendizado do agente em diferentes cenários e situações.

4.5.3 Recompensa

A recompensa é uma medida usada para fornecer *feedback* ao agente sobre a qualidade de suas ações. É utilizada para recompensar o robô por atingir o objetivo ou aplicar penalidades por aproximação ou colisões com obstáculos.

Na abordagem aqui proposta, a recompensa consiste em cinco partes, e pode ser representada conforme a seguir:

$$R = (R_d \cdot R_\theta) + R_a + R_c + R_s$$

- $R_d = 2^{(\frac{d_c}{d_a})}$: utilizada para ajustar a taxa de distância com base na proximidade atual em relação ao objetivo. Quanto mais próxima a distância atual estiver da

distância do objetivo, maior será o valor da taxa de distância. Por outro lado, se a distância atual for grande em relação à distância do objetivo, a taxa de distância será menor, onde:

d_c = representa a distância atual até o objetivo;
d_a = representa a distância absoluta até o objetivo.

$$R_d = \begin{cases} if\ d_c < d_a \\ \quad r_d > 2 \\ else \\ \quad 1 < r_d \leq 2 \end{cases}$$

- $R_\theta = 5 * 1^{-\theta}$: recompensa angular, assim como R_d, é utilizada para estimular o robô a realizar ações que maximizem a recompensa, onde:

  θ representa o ângulo do robô até o objetivo;

$$R_\theta = \begin{cases} if\ -\frac{1}{2}\pi < \theta < \frac{1}{2}\pi \\ \quad R_\theta \geq 0 \\ else \\ \quad R_\theta < 0 \end{cases}$$

- R_a: indica a penalidade por aproximação de obstáculos ou pessoas. É obtido da seguinte forma:

$$R_a = \begin{cases} if\ hd > 0\ and\ da < da_{min} \\ \quad R_a = hd \cdot (-k) \\ elif\ hd == 0\ and\ da < da_{min} \\ \quad Ra = -k \\ 0\ otherwise \end{cases}$$

  da e da_{min} representam, respectivamente, a distância atual entre o robô e o obstáculo e a distância mínima de segurança entre o robô e o obstáculo;
  hd representa o número de pessoas detectadas em determinado instante;
  k é um valor atribuído para penalizar o robô;
  R_a representa o valor total da penalidade aplicada, tomando-se por base o número de detecções de pessoas.

- R_c: recompensa por colisão. Representa um valor negativo, dado quando o robô colide com algum obstáculo ou pessoa. Esse valor pode ser ajustado de

acordo com a complexidade do ambiente e serve para estimular o agente a evitar colisões. Na configuração utilizada neste trabalho ao colidir o agente recebe a recompensa negativa e o episódio é encerrado.

- R_s: recompensa de sucesso. Se o agente chegar ao objetivo ele recebe uma recompensa positiva, caso contrário, não é recompensado. O valor de R_s é pré-definido na configuração do ambiente e é obtido da seguinte forma:

$$R_s = \begin{cases} R_s \; if \; reach \; goal \\ 0 \; else \end{cases}$$

Para evitar que o agente permaneça por um longo tempo sem atingir o objetivo um tempo máximo de navegação é determinado (t). Quando o episódio atinge o tempo determinado por t, o episódio é encerrado e é iniciado um novo episódio de treinamento.

4.5.4 Aplicação do Algoritmo DDPG

A rede neural do algoritmo Deep Deterministic Policy Gradient (DDPG) é composta por duas partes principais: o ator (*actor*) e o crítico (*critic*). Essas duas partes trabalham em conjunto para aprender e otimizar uma política determinística que guia o agente a tomar ações no ambiente (Fig. 13).

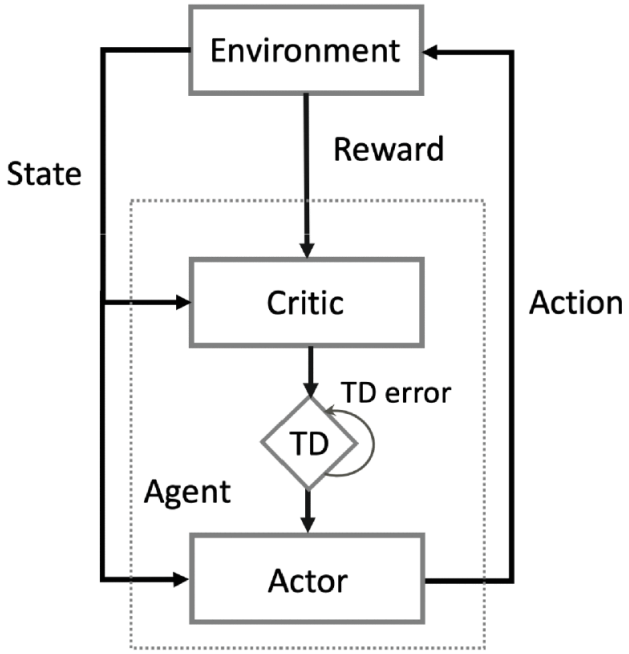


Figura 13 – Arquitetura *Actor-critic* (AC)

Fonte: (ZENG; WANG; GE, 2020)

O ator é responsável por mapear os estados do ambiente para ações. A rede

possui três camadas lineares, onde a primeira camada recebe o estado como entrada, a segunda camada recebe a saída da primeira camada, e a terceira camada gera as ações escolhidas pelo ator. A entrada da rede são os estados do ambiente, conforme detalhado na seção 5.1.1 e descritas a seguir:

Neste trabalho a entrada da rede é dada por: $[\ l_r, \theta, d_c, r_{min}, \theta_{obs}, h_d\ ]$ onde:

- l_r: representa o número de amostras do sensor LiDAR. Nos experimentos realizados foram utilizadas 24 amostras, porém podem ser utilizados valores diferentes, bastando ajustar os parâmetros do algoritmo;
- θ: é o ângulo do robô em relação ao objetivo;
- d_c: é a distância atual até o objetivo;
- r_{min}: corresponde à distância do obstáculo mais próximo;
- θ_{obs}: corresponde ao ângulo do obstáculo mais próximo em relação ao robô;
- h_d: corresponde ao número de pessoas detectadas;

O crítico, por sua vez, tem como objetivo estimar a função Q (valor de ação) para o par estado-ação. A rede crítico possui três camadas lineares, onde a primeira camada recebe o estado como entrada, a segunda camada combina as informações do estado e da ação, e a terceira camada gera uma estimativa do valor Q para o par estado-ação. A função de ativação ReLU é aplicada após cada camada linear. O objetivo do crítico é aprender uma função Q que forneça uma boa estimativa do retorno esperado, levando em consideração as ações escolhidas pelo ator.

Durante o treinamento do DDPG, o agente interage com o ambiente, observando estados, tomando ações com base nas políticas aprendidas e recebendo recompensas. Essas experiências são armazenadas em um *buffer* de repetição (SCHAUL et al., 2015), que consiste em uma memória para armazenar uma coleção de transições passadas. Periodicamente, amostras são retiradas desse *buffer* para o treinamento da rede neural do ator e do crítico.

A otimização dos pesos da rede neural é realizada utilizando a técnica de gradiente ascendente para ajustar os parâmetros do ator e do crítico. Durante o treinamento, a diferença entre o valor estimado pelo crítico e o retorno real obtido é usada para calcular a perda (*loss*) e atualizar os pesos da rede.

A saída da rede são as ações escolhidas pelo agente. Conforme descrito na seção 5.2.2, elas representam os movimentos que o robô irá executar com base em seu estado atual no ambiente e correspondem às velocidades angulares do robô (Fig. 14).

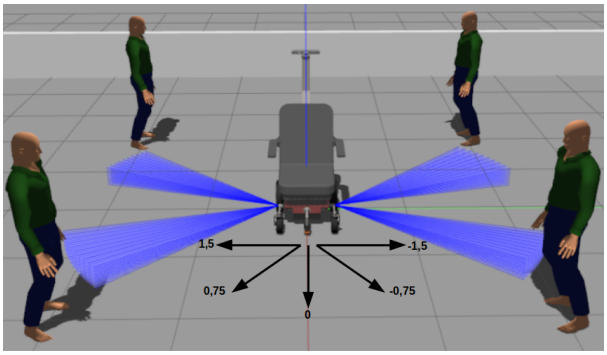


Figura 14 – Ações - Velocidade angular

Fonte: Autor

A função de ativação final do ator é escolhida de acordo com o intervalo de ação requerido pelo ambiente. O espaço de ação corresponde a um espaço tridimensional onde cada ação é um valor contínuo dentro de um limiar pré-estabelecido. O objetivo do ator é aprender uma política determinística que maximize a recompensa esperada.

4.6 Ambientes de Treinamento

A etapa de treinamento envolve diversos ciclos de interação entre o agente e o ambiente. Esta seção descreve os ambientes implementados para realização do treinamento e as configurações e métricas utilizadas para avaliar o desempenho do agente.

Para os experimentos e análises de resultados, foram configurados quatro ambientes de navegação. Esses ambientes possuem características distintas e níveis de complexidade variados.

Os quatro ambientes implementados possuem o mesmo tamanho, representando um cenário de 10x10 metros, implementado no simulador Gazebo. A posição alvo é gerada de forma randômica, com base nas coordenadas do ambiente. Nos ambientes 02 e 03 são consideradas também as coordenadas dos obstáculos estáticos, dessa forma é possível evitar gerar alvos na mesma posição dos obstáculos.

4.6.1 Etapa 01: Ambiente sem obstáculos

Nesta etapa o objetivo principal é avaliar a capacidade do robô de navegar até o ponto de destino com base na leitura dos sensores e em sua posição em relação ao ambiente (Fig. 15). Essa análise preliminar é fundamental para aferir a capacidade de aplicação da solução em ambientes mais complexos.

Apesar de não conter outros obstáculos, além das paredes em torno do robô, esse cenário permite avaliar se o sensoriamento está sendo executado de forma correta e também comparar o desempenho dos algoritmos em relação ao tempo de aprendizado, considerando o número de vezes que o robô atingiu o objetivo alvo e as recompensas obtidas.

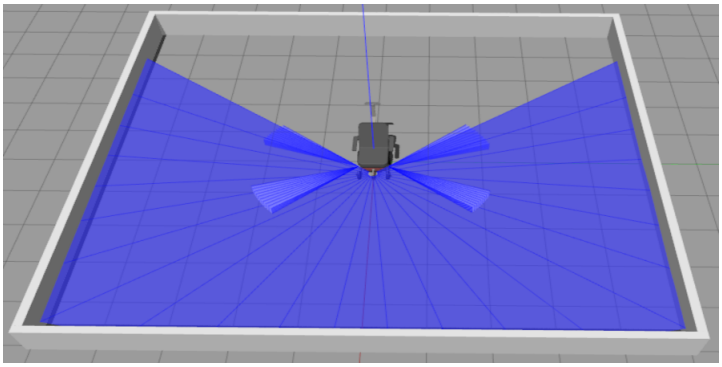


Figura 15 – Etapa de treinamento 01: Ambiente sem obstáculos

Fonte: Autor

4.6.2 Etapa 02: Ambiente com obstáculos estáticos

O ambiente desenvolvido nesta etapa objetiva treinar o robô para alcançar o ponto de destino evitando colisões com obstáculos estáticos ao seu redor. (Fig. 16). Foram adicionados quatro obstáculos estáticos, como se representassem pilares em torno do robô.

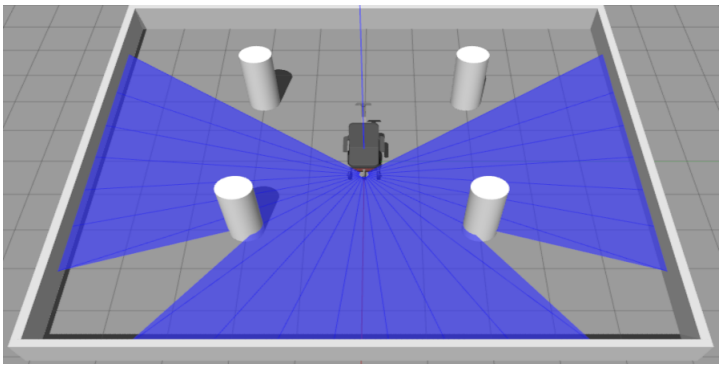


Figura 16 – Etapa de treinamento 02: Ambiente com obstáculos estáticos

Fonte: Autor

4.6.3 Etapa 03: Ambiente com pessoas paradas

Nesta etapa o robô deve ser capaz de detectar pessoas e é treinado para alcançar o ponto de destino, evitando colisões e mantendo uma distância segura em relação às pessoas paradas no ambiente de navegação. (Fig. 17).

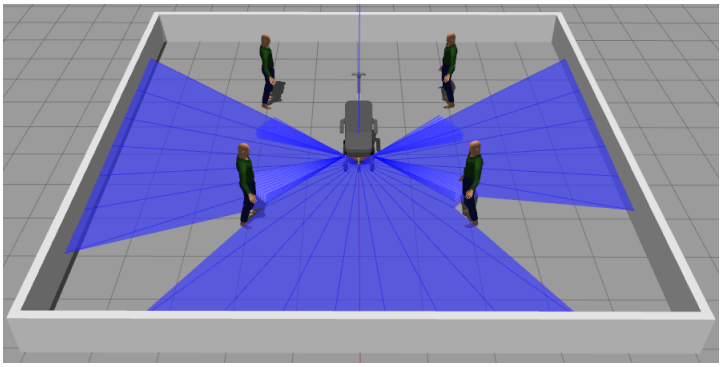


Figura 17 – Etapa de treinamento 03: Ambiente com pessoas paradas

Fonte: Autor

4.6.4 Etapa 04: Ambiente com pessoas dinâmicas

Nesta etapa o robô deve ser capaz de detectar pessoas e é treinado para alcançar o ponto de destino, evitando colisões e mantendo uma distância segura em relação às pessoas que estão caminhando no ambiente de navegação. Na configuração do ambiente 04 as pessoas simuladas movem-se numa determinada linha, seguindo um trajeto específico, não aleatório, de ida e volta em linha reta (Fig. 18);

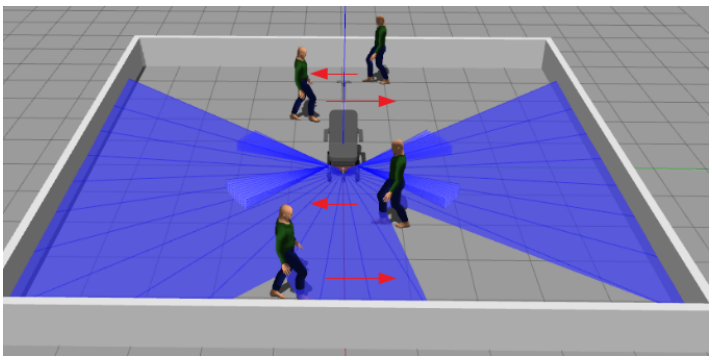


Figura 18 – Etapa de treinamento 04: Ambiente com pessoas em movimento

Fonte: Autor

4.6.5 Configurações

Um ambiente de aprendizado por reforço é composto por vários elementos que determinam a dinâmica do processo de aprendizado. O agente é o componente que está aprendendo a tomar decisões e realizar ações no ambiente para maximizar uma recompensa cumulativa ao longo do tempo, sendo representado pela cadeira de rodas robótica, conforme descrito na seção 4.2.

A tabela a seguir apresenta os parâmetros de configuração referentes ao espaço de estados e ao espaço de ação, assim como a duração e o número de episódios adotado para cada etapa de treinamento. Laser Distance Sensor (LDS) refere-se ao número de amostras do sensor LiDAR que foram utilizadas durante os experimentos.

As informações de estado, são obtidas a partir da leitura dos dados dos sensores conectados ao robô. Neste trabalho, em todas as etapas de treinamento foram utiliza-

Tabela 3 – Parâmetros de Treinamento

Ambiente	Espaço de estados	Espaço de ação	Amostras LDS	Nº máx. de episódios	Duração de um episódio
Etapa 01	26	5	24	500	300
Etapa 02	28	5	24	1500	500
Etapa 03	29	5	24	1500	500
Etapa 04	29	5	24	2000	500

das 24 amostras do sensor LiDAR, informações de odometria e dados de 04 sensores infravermelhos conectados nas laterais do robô. As imagens obtidas do sensor Kinect foram utilizadas apenas nas etapas de treinamento 03 e 04.

Conforme descrito na seção 4.5.1, o estado refere-se a uma representação do ambiente em um determinado instante de tempo, capturando as informações necessárias para a tomada de decisão e determinação das próximas ações. Portanto, as configurações referentes ao espaço de estados diferem de acordo com a etapa de treinamento. Essa diferença ocorre porque os componentes de entrada da rede variam de acordo com a etapa de treinamento e a complexidade do ambiente, conforme descrito a seguir:

- **Etapa 01**: Nesta etapa a entrada da rede não necessita de informações sobre ângulo e distância em relação à obstáculos, nem sobre pessoas detectadas. Portanto, é dada por: $[\ l_r, \theta, d_c\ ]$ onde:
  - l_r: representa o número de amostras do sensor LiDAR;
  - θ: é o ângulo do robô em relação ao objetivo;
  - d_c: é a distância atual até o objetivo;
- **Etapa 02**: Nesta etapa, além dos dados coletados na etapa 01, são necessárias informações referentes aos obstáculos estáticos em torno do robô. Portanto, a entrada da rede é dada por: $[\ l_r, \theta, d_c, r_{min}, \theta_{obs}\ ]$ onde:
  - l_r: representa o número de amostras do sensor LiDAR;
  - θ: é o ângulo do robô em relação ao objetivo;
  - d_c: é a distância atual até o objetivo;
  - r_{min}: corresponde à distância do obstáculo mais próximo;
  - θ_{obs}: corresponde ao ângulo do obstáculo mais próximo em relação ao robô;
- **Etapas 03 e 04**: Correspondem às etapas em que o robô deve ser capaz de aprender a navegar em meio à pessoas paradas e em movimento. Dessa forma, a entrada da rede recebe um novo parâmetro referente ao número de pessoas detectadas, e é dada por: $[\ l_r, \theta, d_c, r_{min}, \theta_{obs}, h_d\ ]$ onde:

- l_r: representa o número de amostras do sensor LiDAR;
- θ: é o ângulo do robô em relação ao objetivo;
- d_c: é a distância atual até o objetivo;
- r_{min}: corresponde à distância do obstáculo mais próximo;
- θ_{obs}: corresponde ao ângulo do obstáculo mais próximo em relação ao robô;
- h_d: corresponde ao número de pessoas detectadas.

As configurações referentes ao número de episódios e ao tempo de duração de um determinado episódio foram ajustadas de acordo com a etapa de treinamento, tomando-se por base a complexidade do ambiente (Tabela 3, colunas 04 e 05). Esses valores foram determinados após a realização de testes que permitiram avaliar se o tempo de treinamento especificado seria suficiente para o agente navegar atingindo os objetivos sem colidir com obstáculos.

Além das configurações anteriormente descritas, a configuração correta do algoritmo de aprendizado é fundamental para o treinamento do agente.

A seguir são listados os principais hiperparâmetros utilizados na configuração dos algoritmos utilizados neste trabalho:

- buffer_size: representa o número de experiências (amostras) armazenadas em memória. É usado para armazenar transições passadas (estado, ação, recompensa, próximo estado) e realizar amostragem aleatória durante o treinamento;
- batch_size: Número de amostras retiradas do *buffer* de repetição em cada atualização da rede neural. Um lote (*batch*) é usado para calcular gradientes e realizar atualizações mais frequentes e estáveis;
- gamma: Fator de desconto, corresponde a um valor no intervalo [0, 1] que determina o quão importante é a recompensa futura em relação à recompensa imediata. Um valor de gamma próximo de 1 indica que o agente valoriza recompensas futuras mais fortemente, enquanto um valor próximo de 0 indica que o agente se concentra apenas em recompensas imediatas;
- TAU: Representa a taxa utilizada para atualizar os parâmetros da rede-alvo (*target network*) com os parâmetros da rede principal. Ajuda a melhorar a estabilidade do treinamento, evitando atualizações bruscas;
- learning_rate_actor: É a taxa de aprendizado usada para atualizar os parâmetros da rede neural do ator (*policy network*);
- learning_rate_critic: É a taxa de aprendizado usada para atualizar os parâmetros da rede neural do crítico (valor da função Q);

- buffer_start: O tamanho inicial do *buffer* de repetição. É o número mínimo de experiências coletadas antes de começar o treinamento;
- epsilon: representa o valor de exploração inicial, utilizado para equilibrar a exploração e a prospecção (*exploitation*) durante o treinamento. É a probabilidade do agente escolher uma ação aleatória em vez de seguir a política aprendida;
- epsilon_decay: Fator de decaimento aplicado ao valor de epsilon ao longo do tempo. Permite que a estratégia de prospecção seja reduzida gradualmente à medida que o agente aprende.
- action_space_high: Representa o limite superior do espaço de ações contínuas. É um vetor que define o valor máximo para cada dimensão da ação, definido entre -1.5 e 1.5.
- action_space_low: Representa o limite inferior do espaço de ações contínuas. É um vetor que define o valor mínimo para cada dimensão da ação, definido entre -1.5 e 1.5.
- H1 e H2: Número de neurônios na 1ª e 2ª camadas. Estes são os hiperparâmetros que definem o número de neurônios (unidades) em cada camada da rede neural do ator e do crítico, que podem variar, dependendo do problema a ser resolvido.

As tabelas 4 e 5 apresentam os valores adotados para cada hiperparâmetro, de acordo com o algoritmo utilizado:

Tabela 4 – Configurações do algoritmo DDPG

Hiperparâmetro	Valor	Descrição
buffer_size	1000000	Tamanho do buffer de repetição
batch_size	64	Tamanho de um grupo de amostras de treinamento
gamma	0.99	Taxa de redução de recompensas futuras em relação às recompensas imediatas
TAU	0.001	Taxa de atualização de hiperparâmetros da rede de destino
LRA	0.00025	Taxa de aprendizado da rede ator
LRC	0.0025	Taxa de aprendizado da rede crítico
H1	400	Número de neurônios da 1ª camada
H2	300	Número de neurônios da 2ª camada
buffer_start	100	Número de transições que antecedem a amostragem aleatória
epsilon	1.0	Taxa de exploração
epsilon_decay	0.1	Redução da taxa de exploração (influencia o ruído aplicado à ação)
action_space_high	1.5	Valor máximo que cada dimensão da ação pode assumir
action_space_low	-1.5	Valor mínimo que cada dimensão da ação pode assumir

4.6.6 Métricas

A seguir são apresentadas as métricas utilizadas para medir o desempenho do agente em cada etapa de treinamento específica:

Tabela 5 – Configurações do algoritmo DQN

Hiperparâmetro	Valor	Descrição
TAU	2000	Taxa de atualização da rede de destino.
gamma	0.99	Taxa de redução de recompensas futuras em relação às recompensas imediatas.
learning_rate	0.00025	Taxa de aprendizado.
epsilon	1.0	A probabilidade de escolher uma ação aleatória.
epsilon_decay	0.99	Taxa de redução de epsilon ao término de um episódio.
epsilon_min	0.05	Valor mínimo de epsilon
batch_size	64	Tamanho de um grupo de amostras de treinamento.
train_start	64	Início do treinamento de acordo com o tamanho da memória de repetição.
buffer_size	1000000	O tamanho do *buffer* de repetição.

- Taxa de Sucesso (*Success Rate* - SR): A porcentagem de vezes em que o robô concluiu com sucesso a etapa de treinamento, dada por $SR = \frac{g}{st_{max}} \times 100$, onde g é o número de vezes que o robô alcançou o objetivo sem colidir, e st_{max} representa o tempo máximo de execução do experimento.

- Taxa de Colisão (*Collision Rate* - CR): A porcentagem de vezes em que o robô colidiu, dada por $CR = \frac{c}{ne_{max}} \times 100$, onde c é o número de vezes que o robô colidiu, e ne_{max} é o número total de episódios durante um experimento.

- Taxa de Tempo excedido (*Time Out rate* - TR): A porcentagem de vezes em que o limite de tempo foi excedido sem que o robô alcançasse o objetivo, dado por $TR = \frac{t}{ne_{max}} \times 100$, onde t é o número de vezes que o limite de tempo foi excedido, e ne_{max} é o número total de episódios durante um experimento.

5 EXPERIMENTOS E RESULTADOS

5.1 Treinamento

Esta seção apresenta os resultados obtidos durante cada etapa de treinamento. Os experimentos foram conduzidos em ambiente simulado e demonstram a capacidade de aprendizado e aplicabilidade desses algoritmos em diferentes cenários.

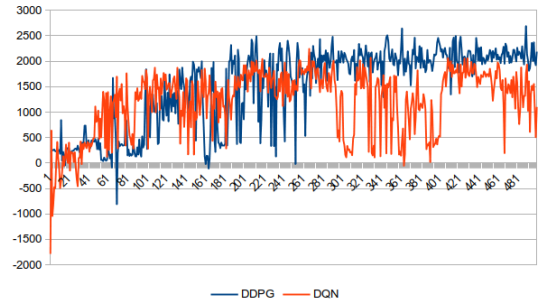
5.1.1 Resultados do Treinamento

Os valores na Tabela 6 correspondem aos resultados obtidos ao longo de todo o tempo de treinamento em cada ambiente específico. Esses valores são derivados com base em $st_{max} = ne_{max} * ns_{max}$, onde st_{max} é o tempo máximo de execução do experimento, ne_{max} é o número máximo de episódios e ns_{max} é o número máximo de passos (*steps*). Os valores em negrito correspondem aos melhores resultados obtidos durante as etapas de experimentos em cada ambiente de treinamento.

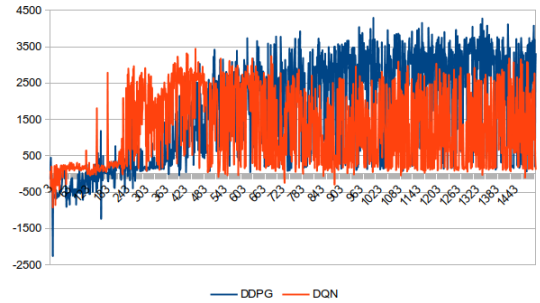
Métricas avaliadas	Ambiente 01		Ambiente 02		Ambiente 03		Ambiente 04	
	SAN-DDPG	DQN	SAN-DDPG	DQN	SAN-DDPG	DQN	SAN-DDPG	DQN
Sucesso (g)	**850**	691	**2656**	1563	**3167**	1780	**3021**	1282
Colisões (c)	53	**46**	**705**	949	**631**	882	**1179**	1482
Tempo excedido (t)	75	**72**	55	**9**	36	**14**	**9**	21
Taxa de sucesso (%)	**56,67**	46,07	**35,41**	20,84	**42,23**	23,73	**30,21**	12,82
Taxa de colisão (%)	10,60	**9,20**	**47,00**	63,27	**42,07**	58,80	**58,95**	74,10
Taxa p/tempo excedido (%)	15,00	**14,40**	3,67	**0,60**	2,40	**0,93**	**0,45**	1,05

Tabela 6 – Resultados do Treinamento

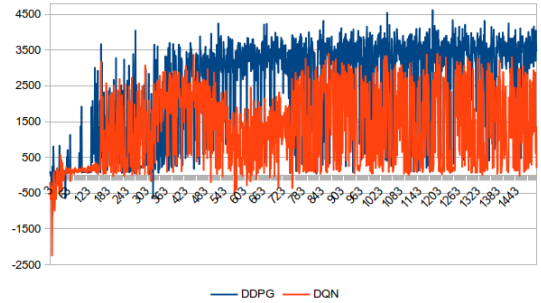
Um episódio corresponde a uma etapa de treinamento, conforme definido na tabela 3. O número máximo de passos (*steps*) é utilizado para definir o tempo de duração de um episódio, que pode encerrar de duas formas: quando o robô realiza uma colisão ou quando excede o tempo máximo de execução do experimento.



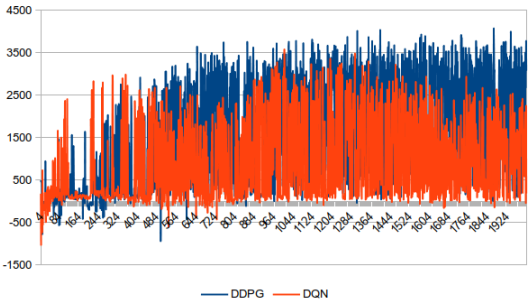
(a) Etapa 01



(b) Etapa 02



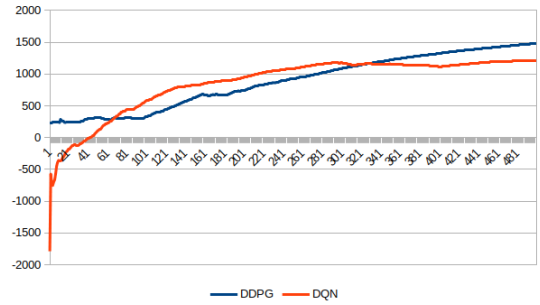
(c) Etapa 03



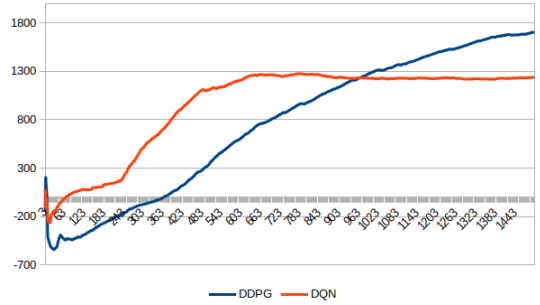
(d) Etapa 04

Figura 19 – Recompensas por episódio - Os valores correspondentes ao eixo Y (à esquerda) apresentam as recompensas obtidas por episódio, o eixo X corresponde aos episódios de treinamento.

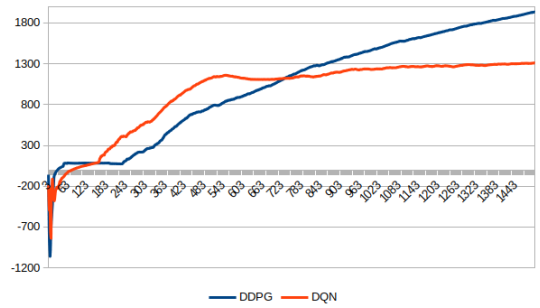
A Figura 19 apresenta a evolução das recompensas obtidas por episódio em cada estágio de treinamento.



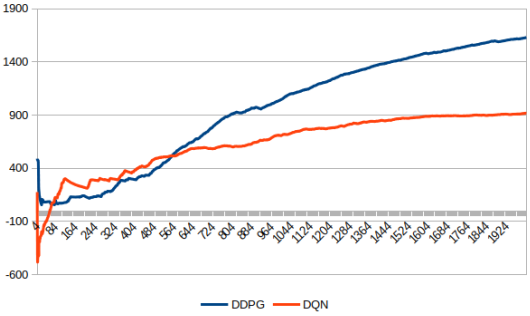
(a) Etapa 01



(b) Etapa 02



(c) Etapa 03



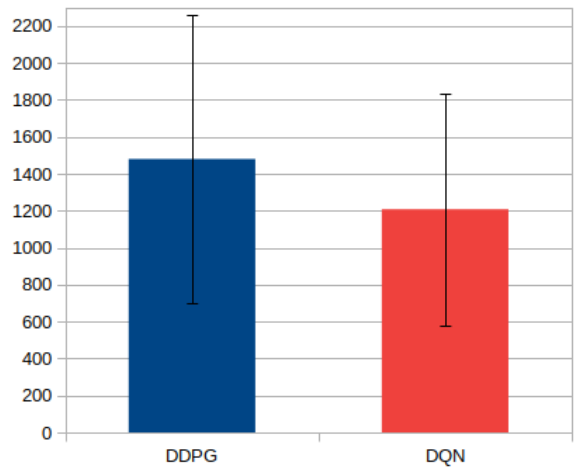
(d) Etapa 04

Figura 20 – Recompensa Média - O eixo Y (à esquerda) apresenta os valores correspondentes à média de recompensas obtidas ao longo do treinamento, o eixo X corresponde aos episódios de treinamento.

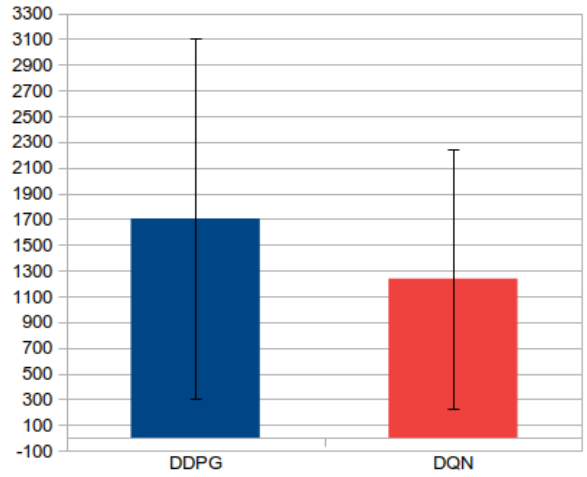
A recompensa por episódio é a soma das recompensas obtidas pelo agente durante todo um episódio de interação com o ambiente. Essa métrica permite avaliar como o agente está progredindo em direção ao seu objetivo, analisando a eficácia das políticas aprendidas e a capacidade de maximizar as recompensas acumuladas ao longo do tempo.

Ao monitorar as recompensas por episódio, é possível identificar tendências de melhoria ou piora no desempenho do agente à medida que o treinamento progride. Essa análise possibilita identificar problemas, ajustar os hiperparâmetros do algoritmo, refinar as funções de recompensa ou melhorar a política de exploração do agente durante as etapas de treinamento.

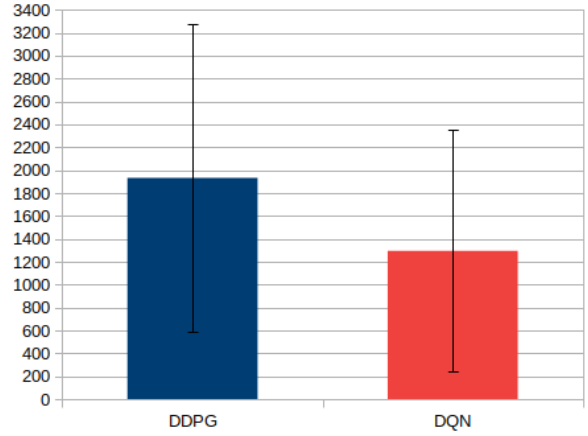
A Figura 20, apresenta uma medida consolidada do desempenho médio do agente em cada etapa de treinamento e permite visualizar de forma mais clara o avanço do aprendizado ao longo do tempo.



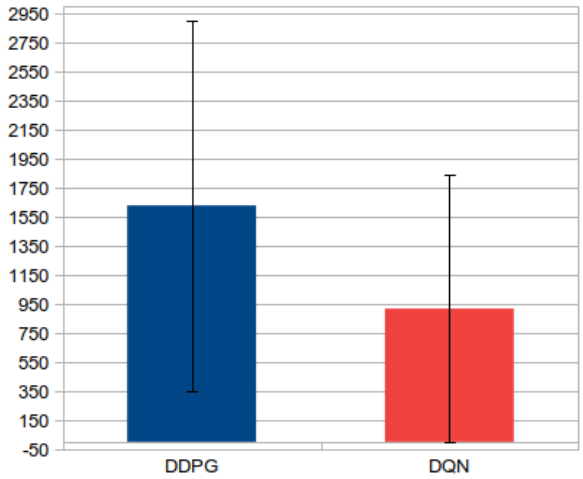
(a) Etapa 01



(b) Etapa 02



(c) Etapa 03



(d) Etapa 04

Figura 21 – Desvio padrão sobre a média de recompensas recebidas - O eixo Y (à esquerda) apresenta os valores correspondentes à média de recompensas obtidas ao longo do treinamento.

A Figura 21, apresenta o desvio padrão sobre as recompensas recebidas. Essa medida indica a variabilidade dos valores de recompensa obtidos pelo agente durante o treinamento. Um desvio padrão mais amplo indica que as recompensas estão mais dispersas em torno da média, enquanto um desvio padrão menor indica maior consistência e estabilidade nas recompensas.

5.1.2 Análise dos resultados

- Etapa 01: Nesta etapa a abordagem proposta e o DQN apresentaram desempenhos próximos, com baixas taxas de colisão e altas taxas de sucesso. A abordagem SAN-DDPG apresentou maior estabilidade nas recompensas recebidas, enquanto o DQN obteve recompensas baixas mesmo após 300 episódios de treinamento (Fig. 19a e 20a). Como pode ser visto na tabela 6, a taxa de colisão apresentada pelo DQN foi inferior à taxa apresentada pela abordagem SAN-DDPG, no entanto, esse resultado pode estar relacionado a uma prospecção menor, resultando em recompensas mais baixas e taxa de sucesso inferior.

- Etapa 02: Na etapa 02 ambos os algoritmos apresentaram dificuldades no início do treinamento, resultando na obtenção de recompensas baixas durante os primeiros 250 episódios. O DQN obteve recompensas melhores no início do treinamento, enquanto SAN-DDPG demorou mais para convergir. Apesar disso, os resultados da tabela 6 demonstram que a abordagem SAN-DDPG apresentou uma taxa de sucesso cerca de 15% superior ao DQN, além de melhor capacidade de aprendizado, adquirindo maior estabilidade na obtenção de recompensas positivas a partir de 400 episódios de treinamento (Fig. 20b). Por outro lado, o DQN apresentou uma elevada taxa de colisão, o que demonstra que teve dificuldades em tomar decisões em um ambiente com obstáculos em seu entorno, resultando na baixa taxa de sucesso obtida.

- Etapa 03: Nesta etapa os algoritmos passaram a ter um comportamento mais exploratório após 180 episódios de treinamento (Fig. 20b). A solução proposta por este trabalho atingiu estabilidade na obtenção de recompensas positivas após 350 episódios. O DQN por sua vez apresentou uma alta taxa de colisão, resultando em recompensas negativas, mesmo após 700 episódios de treinamento. Como pode ser visto na figura 20c, essa característica menos exploratória do DQN resultou em um aprendizado mais demorado para o agente, demonstrando dificuldade em maximizar as recompensas obtidas ao longo do tempo.

- Etapa 04: Nesta etapa a abordagem SAN-DDPG apresentou uma taxa de sucesso cerca de 17% superior em relação ao DQN, com poucas colisões no início do treinamento, passando a maximizar as recompensas obtidas de forma

mais significativa após 300 episódios de treinamento. O gráfico de recompensas (Fig. 19d) demonstra que ambos os algoritmos obtiveram recompensas negativas mesmo após 500 episódios de treinamento. No entanto percebe-se que a abordagem SAN-DDPG manteve o aprendizado constante, maximizando o valor das recompensas durante todo o treinamento, obtendo resultados significativamente melhores que o DQN ao longo do tempo.

De forma geral a análise dos experimentos realizados demonstra que ambos os algoritmos obtiveram bons resultados, proporcionando que o agente aprenda e ao final do tempo de treinamento navegue de forma autônoma apresentando baixas taxas de colisão nos cenários em que foi treinado. No entanto, é possível perceber que a abordagem SAN-DDPG possui características que a tornam superior ao DQN, como maior capacidade exploratória, aprendizado constante, maior estabilidade e capacidade de maximizar as recompensas obtidas, convergindo para soluções melhores em todos as etapas.

Os gráficos demonstram que no início dos experimentos o algoritmo DQN foi capaz de obter recompensas melhores que o SAN-DDPG em todas as etapas (Figuras 20a, 20b, 20c e 20d). No entanto, percebe-se que o DQN tende a estabilizar o aprendizado, enquanto o SAN-DDPG apresenta uma evolução contínua, obtendo resultados significativamente melhores ao longo do tempo. Essa característica pode ser percebida por meio da análise da recompensa média, que fornece uma medida consolidada do desempenho médio do agente ao longo do tempo.

Assim como a recompensa por episódio, a recompensa média é utilizada para monitorar o progresso, identificar problemas de aprendizado e ajustar a política de ação durante o treinamento. Os experimentos demonstram que a recompensa média utilizando o algoritmo DDPG foi significativamente maior em comparação ao DQN, evidenciando que a abordagem SAN-DDPG foi mais eficaz em maximizar as recompensas ao longo do treinamento em todas as etapas (Figuras 20a, 20b, 20c e 20d).

Percebe-se ainda que a solução proposta por este trabalho apresentou um desvio padrão mais amplo em relação à média de recompensas quando comparada ao DQN, no entanto, também obteve recompensas positivas significativamente maiores, resultando em um desempenho geral melhor em termos de recompensas totais recebidas (Fig. 21).

Um desvio padrão mais amplo pode estar relacionado ao comportamento mais exploratório do algoritmo DDPG, não necessariamente resultando em um desempenho inferior. Isso sugere que o agente está explorando diferentes estratégias e ações em busca de uma política ótima, resultando em maior variabilidade nos resultados.

O DQN obteve um desvio padrão mais elevado em termos de recompensas negativas, principalmente nas etapas 2, 3 e 4, onde os cenários são mais desafiadores e consequente sujeitam o agente à um número maior de colisões.

Ao analisar a plataforma robótica durante as etapas de treinamento com o algoritmo DQN é possível perceber uma maior instabilidade nos movimentos executados, resultando em movimentos angulares mais bruscos o que, consequentemente, pode ter resultado na elevada taxa de colisões. Por outro lado, o SAN-DDPG apresentou uma navegação mais suave, proporcionando o atingimento dos objetivos de forma mais rápida e melhor capacidade para evitar colisões com obstáculos próximos.

5.2 Validação e Testes

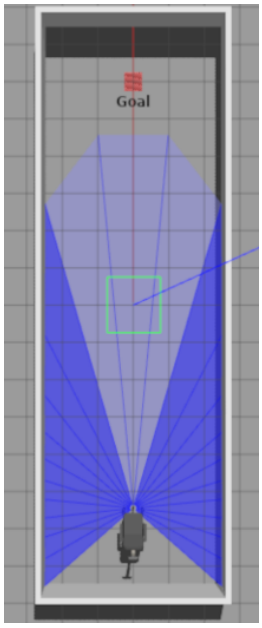
Como forma de validar a eficiência e robustez da solução proposta, foram aplicados diversos testes em cenários distintos aos que o agente foi treinado. Esta seção apresenta as configurações utilizadas durante a etapa de testes, a avaliação dos resultados obtidos e as considerações acerca desses resultados. Cumpre destacar que os experimentos com simulação de humanos não envolveram qualquer tipo de colaboração durante a navegação. Cabendo, portanto, apenas ao robô, com base no aprendizado adquirido, evitar colisões enquanto busca atingir seu objetivo.

5.2.1 Ambientes de Teste

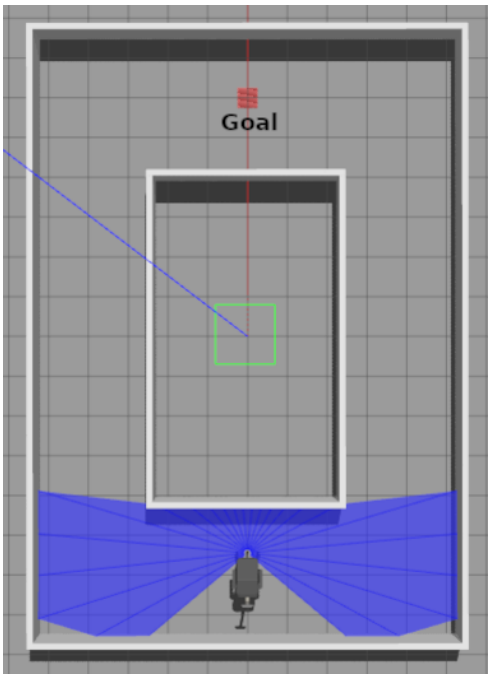
Os ambientes de teste foram configurados com tamanhos diferentes e com níveis de complexidade superiores aos ambientes de treinamento. O objetivo é avaliar a robustez da solução proposta e verificar se o agente é capaz de generalizar o conhecimento aprendido em um ambiente específico para situações novas e desconhecidas, aplicando o conhecimento aprendido anteriormente em novos contextos.

Além disso, os testes realizados em ambientes diferentes, permitem coletar e analisar novos dados e *feedback* que podem ser utilizados para fornecer uma validação mais abrangente do seu desempenho.

5.2.1.1 Etapa 01: Ambientes sem obstáculos



(a) Corredor sem obstáculos

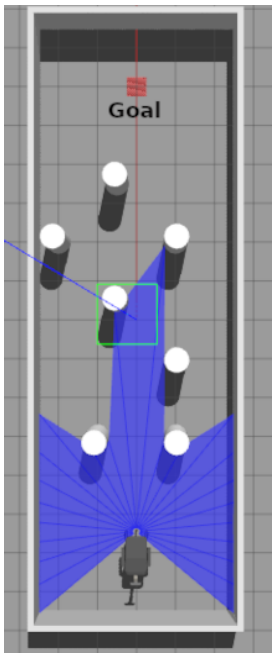


(b) Circuito 10x15 sem obstáculos

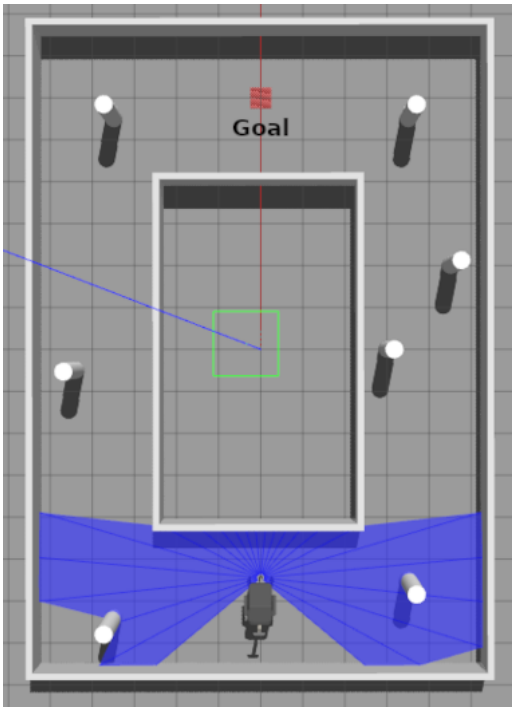
Figura 22 – Etapa de testes 01: Ambientes sem obstáculos

Fonte: Autor

5.2.1.2 Etapa 02: Ambientes com obstáculos estáticos



(a) Corredor com obstáculos estáticos

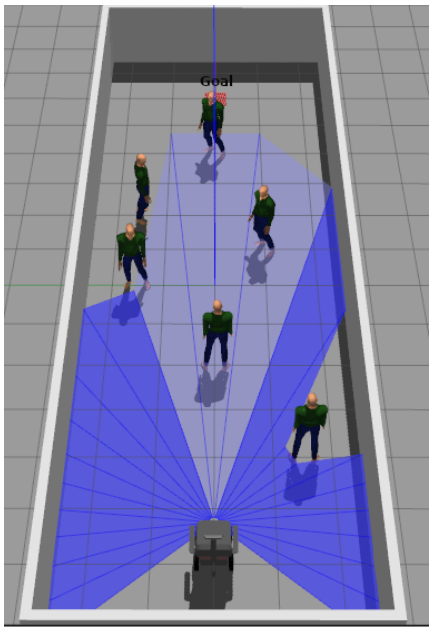


(b) Circuito 10x15 com obstáculos estáticos

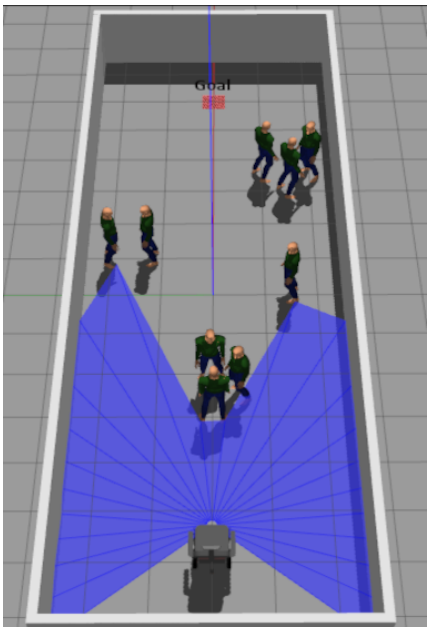
Figura 23 – Etapa de testes 02: Ambientes com obstáculos estáticos

Fonte: Autor

5.2.1.3 Etapa 03: Ambientes com pessoas paradas



(a) Corredor com pessoas paradas individualmente

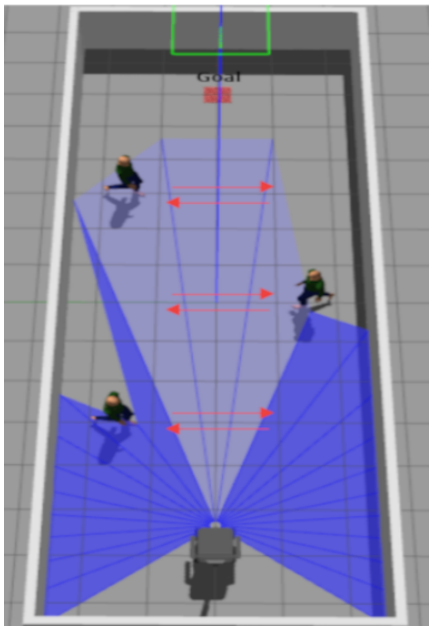


(b) Corredor com grupos de pessoas paradas

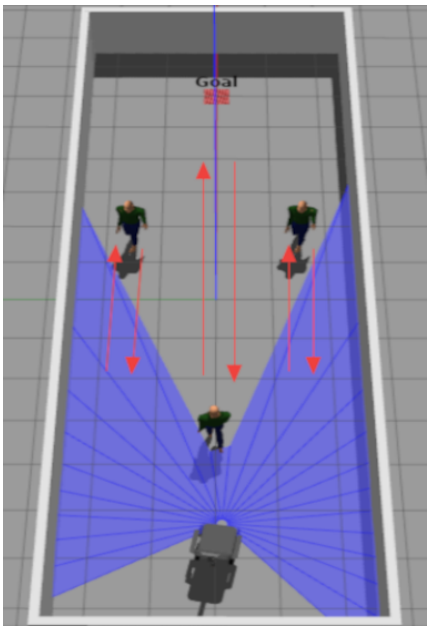
Figura 24 – Etapa de testes 03: Corredor com pessoas paradas individualmente e em grupos.

Fonte: Autor

5.2.1.4 Etapa 04: Ambientes com pessoas dinâmicas



(a) Corredor com pessoas em movimento - obstrução horizontal em relação ao alvo



(b) Pessoas em movimento - obstrução vertical em relação ao alvo

Figura 25 – Etapa de testes 04 - Pessoas em movimento

Fonte: Autor

5.2.2 Configurações

Para realização dos testes, com exceção da etapa 01, foram utilizadas as mesmas configurações adotadas durante as etapas de treinamento, de acordo com cada ambiente, conforme definido na seção 4.6.5.

5.2.2.1 Etapa de testes 01

A etapa de testes 01 foi realizada em dois cenários distintos: corredor sem obstáculos (Fig. 22a) e um circuito 10x15 sem obstáculos (Fig 22b).

Esta etapa objetiva avaliar o comportamento do agente em diferentes situações relacionadas à alterações na dimensão do ambiente. O circuito 10x15 é mais estreito nas laterais, submetendo o robô a uma área de travessia que corresponde à metade da largura do corredor e menos de 1/3 em relação à área de treinamento. Além disso, o circuito possui um obstáculo central, como uma parede, dessa forma são necessárias informações adicionais para que o robô possa navegar nesse ambiente.

Portanto, a entrada da rede é dada por: $[\ l_r, \theta, d_c, r_{min}, \theta_{obs}\ ]$ onde:

- l_r: representa o número de amostras do sensor LiDAR;
- θ: é o ângulo do robô em relação ao objetivo;
- d_c: é a distância atual até o objetivo;
- r_{min}: corresponde à distância do obstáculo mais próximo;
- θ_{obs}: corresponde ao ângulo do obstáculo mais próximo em relação ao robô;

5.2.2.2 Etapa de testes 02

Nesta etapa as configurações de entrada da rede foram as mesmas anteriormente descritas para a etapa de testes 01.

Os testes foram realizados em dois cenários distintos: corredor com obstáculos estáticos (Fig. 23a) e um circuito 10x15 com obstáculos estáticos (Fig. 23b). Como pode ser visto nas figuras 23a e 23b, o tamanho dos obstáculos também é diferente, havendo obstáculos maiores no corredor.

O objetivo nesta etapa é avaliar se o agente é capaz de lidar com cenários complexos, repletos de obstáculos e ter uma avaliação mais abrangente do seu desempenho.

5.2.2.3 Etapa de testes 03

A etapa de testes 03 foi realizada em dois cenários distintos: corredor com pessoas paradas individualmente (Fig. 22a) e corredor com grupos de pessoas paradas (Fig 22b). O objetivo é avaliar se o robô é capaz de navegar em meio à pessoas paradas,

abordando situações diversas daquela para a qual foi treinado e em ambientes mais complexos.

Conforme pode ser visto na Figura 17, o robô foi treinado em um ambiente mais amplo e com apenas 04 pessoas em seu entorno. Entretanto, para lidar com cenários da vida real, ele deve ser capaz de generalizar o conhecimento adquirido para lidar com situações inesperadas, como grupos de pessoas ou pessoas paradas individualmente durante o percurso até seu objetivo. Dessa forma, os ambientes de teste foram implementados considerando essas situações.

Nas etapas 03 e 04 a entrada da rede recebe um parâmetro referente ao número de pessoas detectadas, e é dada por: $[\, l_r, \theta, d_c, r_{min}, \theta_{obs}, h_d \,]$ onde:

- l_r: representa o número de amostras do sensor LiDAR;
- θ: é o ângulo do robô em relação ao objetivo;
- d_c: é a distância atual até o objetivo;
- r_{min}: corresponde à distância do obstáculo mais próximo;
- θ_{obs}: corresponde ao ângulo do obstáculo mais próximo em relação ao robô;
- h_d: corresponde ao número de pessoas detectadas.

5.2.2.4 Etapa de testes 04

A etapa de testes 04 foi realizada em dois cenários, com pessoas em movimento em diferentes direções: no primeiro cenário as pessoas movimentam-se transversalmente em relação à pose do robô (Fig. 25a) e no segundo cenário o movimento das pessoas segue a trajetória entre o robô e o ponto alvo (Fig 25b). Assim como na etapa 03, o objetivo é avaliar se o robô é capaz de navegar em meio à pessoas em movimento, abordando situações diversas daquela para a qual foi treinado e em ambientes de maior complexidade.

5.2.3 Métricas

A seguir são apresentadas as métricas utilizadas para avaliar o desempenho do agente em cada etapa de testes:

- Taxa de Sucesso (SR): A porcentagem de vezes em que o robô concluiu com sucesso a etapa de testes, dada por $SR = \frac{g}{nt_{max}} \times 100$, onde g é o número de vezes que o robô alcançou o objetivo sem colidir, e nt_{max} representa o número máximo de testes realizados.
- Taxa de Colisão (CR): A porcentagem de vezes em que o robô colidiu, dada por $CR = \frac{c}{nt_{max}} \times 100$, onde c é o número de vezes que o robô colidiu, e nt_{max} é o número máximo de testes realizados.

- Tempo médio (AT): A média de tempo que o robô levou para alcançar o objetivo, considerando todas as etapas de testes que resultaram em sucesso, dado por $AT = \frac{t}{nt_{max}} \times 100$, onde t é o tempo que o robô levou para atingir o objetivo durante uma execução, e nt_{max} é o número total de testes realizados.

5.2.4 Resultados dos testes

Esta seção apresenta os resultados obtidos durante cada etapa de testes. Os experimentos foram conduzidos em ambiente simulado e demonstram a eficiência da solução proposta, considerando a capacidade de generalização e adaptação do agente em diferentes cenários e variados níveis de complexidade.

5.2.4.1 Etapa 01

A tabela 7 apresenta os resultados da primeira etapa de testes. Cada coluna representa uma métrica específica, e os valores correspondem às estatísticas obtidas após a execução dos testes.

Tabela 7 – Resultados da etapa de testes 01

Métricas avaliadas	**Etapa 01 (a)**		**Etapa 01 (b)**	
	SAN-DDPG	DQN	SAN-DDPG	DQN
Taxa de sucesso (%)	100%	100%	**98%**	16%
Taxa de colisão (%)	0%	0%	**2%**	84%
Taxa de tempo excedido (%)	0%	0%	0%	0%
Média de tempo em segundos (tr)	**40,97**	41,31	58,85	**51,65**

Tanto o SAN-DDPG quanto o DQN alcançaram uma alta taxa de sucesso na Etapa 01(a), atingindo o objetivo alvo em todas as execuções. No entanto, na Etapa 01(b), o SAN-DDPG registrou uma taxa de sucesso ligeiramente menor, atingindo 98%, enquanto o DQN obteve um desempenho significativamente inferior, com apenas 16% de taxa de sucesso.

Com relação à taxa de colisão, ambos os algoritmos apresentaram uma taxa de 0% na Etapa 01(a), o que indica que conseguiram evitar colisões com sucesso. No entanto, na Etapa 01(b), o SAN-DDPG teve uma taxa de colisão de 2%, enquanto o DQN apresentou uma taxa alarmantemente alta, colidindo 84% das vezes, apresentando dificuldades significativas em evitar colisões, mesmo em um ambiente sem obstáculos.

Em relação ao tempo médio para atingir o objetivo, na Etapa 01(a) o SAN-DDPG obteve um desempenho de 40,97 segundos, enquanto o DQN registrou um tempo ligeiramente maior, com 41,31 segundos. Na Etapa 01(b), o SAN-DDPG apresentou um aumento significativo no tempo médio, com 58,85 segundos, enquanto o DQN teve um tempo médio de 51,65 segundos. Embora a abordagem SAN-DDPG tenha

tido tempos médios mais altos na Etapa 01(b), apresentou uma taxa de sucesso 82% superior em relação ao DQN.

Os resultados da etapa 01 demonstram que a abordagem SAN-DDPG possui boa capacidade de generalização, proporcionando que o robô navegue de forma eficiente em cenários com dimensões diferentes do ambiente de treinamento, atingindo o objetivo com quase 100% de sucesso nos testes realizados.

5.2.4.2 Etapa 02

Tabela 8 – Resultados da etapa de testes 02

Métricas avaliadas	**Etapa 02 (a)**		**Etapa 02 (b)**	
	SAN-DDPG	DQN	SAN-DDPG	DQN
Taxa de sucesso (%)	**85%**	0%	**93%**	1%
Taxa de colisão (%)	**15%**	99%	**7%**	99%
Taxa de tempo excedido	**0%**	1%	0%	0%
Média de tempo em segundos (tr)	**52,08**	0,00	**67,17**	68,24

Na Etapa 02(a) os testes foram realizados em um ambiente extremamente mais complexo que o ambiente de treinamento. Além da redução no espaço, foram colocados 07 obstáculos na área de travessia do robô. Nesta etapa o SAN-DDPG obteve uma taxa de sucesso de 85%, enquanto o DQN não obteve sucesso em nenhuma execução. Na Etapa 02(b), o SAN-DDPG teve um desempenho ainda melhor, alcançando uma taxa de sucesso de 93%, enquanto o DQN manteve a taxa de sucesso extremamente baixa, com apenas 1%.

Em relação a taxa de colisão, na Etapa 02(a), a abordagem SAN-DDPG obteve uma taxa de 15%, enquanto o DQN apresentou um valor extremamente alto, colidindo 99% das vezes, indicando que quase todas as execuções resultaram em colisões. Na Etapa 02(b), o SAN-DDPG teve uma taxa de colisão ainda menor, com apenas 7%, enquanto o DQN manteve a taxa de 99%. Esses resultados mostram que o DQN enfrentou dificuldades significativas em evitar colisões em ambas as etapas, enquanto o SAN-DDPG teve um desempenho consideravelmente melhor.

Com relação ao tempo, na Etapa 02(a), o SAN-DDPG obteve um tempo médio de 52,08 segundos, enquanto o DQN não teve sucesso em nenhuma execução. Na Etapa 02(b) o SAN-DDPG apresentou um tempo médio de 67,17 segundos, e o DQN teve um tempo médio de 68,24 segundos, decorrente da única execução realizada com sucesso.

Os resultados da etapa 02 demonstram que a solução implementada utilizando a abordagem SAN-DDPG é capaz de lidar com ambientes complexos, proporcionando que o robô navegue de forma eficiente em meio à obstáculos estáticos de diferentes dimensões, em um espaço significativamente menor que o ambiente de treinamento

e obtendo uma taxa média de sucesso superior a 90% nos testes realizados.

5.2.4.3 Etapa 03

Tabela 9 – Resultados da etapa de testes 03

Métricas avaliadas	**Etapa 03 (a)**		**Etapa 03 (b)**	
	SAN-DDPG	DQN	SAN-DDPG	DQN
Taxa de sucesso (%)	**100%**	35%	**96%**	2%
Taxa de colisão (%)	**0%**	65%	**4%**	98%
Taxa de tempo excedido	0%	0%	0%	0%
Média de tempo em segundos (tr)	**58,62**	85,90	**53,42**	92,49

Na Etapa 03(a), a abordagem SAN-DDPG alcançou 100% de sucesso, concluindo todas as execuções sem colisões ou tempo excedido. Por outro lado, o DQN obteve uma taxa de sucesso de apenas 35%. Na Etapa 03(b) o SAN-DDPG manteve um bom desempenho, com 96% de sucesso, enquanto o DQN obteve um desempenho muito baixo, com apenas 2% de sucesso.

Em relação à taxa de colisão, na Etapa 03(a), o SAN-DDPG registrou uma taxa de 0%, evitando colisões em todas as execuções, enquanto o DQN obteve uma taxa de colisão de 65%. Na Etapa 03(b) o SAN-DDPG manteve uma taxa de colisão baixa, com apenas 4%, enquanto o DQN apresentou uma taxa de colisão extremamente alta, de 98%. Esses resultados mostram que o DQN continuou a ter dificuldades significativas em evitar colisões em ambas as etapas, enquanto a abordagem SAN-DDPG apresentou um desempenho consideravelmente melhor.

Com relação ao tempo médio, na Etapa 03(a), o SAN-DDPG levou em média 58,62 segundos para atingir o objetivo, enquanto o DQN teve um tempo médio mais elevado de 85,90 segundos. Na Etapa 03(b) o SAN-DDPG apresentou um tempo médio de 53,42 segundos, enquanto o DQN teve um tempo médio significativamente maior, com 92,49 segundos.

Os resultados da Etapa 03 mostram mais uma vez que a abordagem proposta por este trabalho foi superior ao DQN. O SAN-DDPG registrou altas taxas de sucesso, baixa taxa de colisão e tempos médios razoáveis, enquanto o DQN, apesar de ter apresentado resultados melhores do que na etapa 02, continuou enfrentando dificuldades significativas, obtendo taxas de sucesso baixas e altas taxas de colisão.

Conforme descrito anteriormente, nas etapas 03 e 04 a entrada da rede recebe um parâmetro referente ao número de pessoas detectadas. Essa informação é disponibilizada pelo pacote DarknetRos a partir das imagens detectadas pela câmera RGB do sensor kinect e atua como um mecanismo de atenção social. Dessa forma, ao invés de utilizar apenas dados de distância do sensor laser, o número de pessoas detectadas é fornecido como um parâmetro adicional para o algoritmo que utiliza essas

informações para calcular a recompensa, conforme descrito na seção 4.5.3.

Os resultados obtidos na etapa 03 demonstram que ambos os algoritmos obtiveram melhor desempenho ao receber informações de detecção de pessoas a partir de imagens de vídeo. A taxa de punição atribuída ao robô por se aproximar demais das pessoas contribuiu para reduzir o número de colisões, demonstrando que o robô manteve uma distância maior dos obstáculos em relação à etapa 02. Esse resultado é evidenciado tanto nos experimentos de treinamento (Tabela 6) quanto nos testes de navegação (Tabelas 8 e 9).

5.2.4.4 Etapa 04

Tabela 10 – Resultados da etapa de testes 04

Métricas avaliadas	**Etapa 04 (a)**		**Etapa 04 (b)**	
	SAN-DDPG	DQN	SAN-DDPG	DQN
Taxa de sucesso (%)	**80%**	30%	**92%**	26%
Taxa de colisão (%)	**20%**	70%	**6%**	73%
Taxa de tempo excedido	0%	0%	2%	**1%**
Média de tempo em segundos (tr)	45,20	**38,43**	43,04	**41,64**

Na Etapa 04(a), o SAN-DDPG alcançou sucesso em 80% das execuções, enquanto o DQN obteve sucesso em apenas 30% delas. Na Etapa 04(b) o SAN-DDPG obteve um desempenho ainda melhor, com 92% de sucesso, enquanto o DQN obteve apenas 26% de sucesso.

Em relação à taxa de colisão, na Etapa 04(a), o SAN-DDPG registrou uma taxa de 20%, enquanto o DQN teve uma taxa mais alta, resultando em 70% de colisões. Na Etapa 04(b) o SAN-DDPG manteve uma baixa taxa de colisão, com apenas 6%. O DQN por sua vez, apresentou uma taxa de colisão de 73%, o que demonstra que o algoritmo continuou enfrentando dificuldades significativas em evitar colisões, enquanto o SAN-DDPG manteve um bom desempenho.

Com relação ao tempo médio, na Etapa 04(a), o SAN-DDPG levou em torno de 45,20 segundos para atingir o objetivo, enquanto o DQN, apesar de apresentar uma taxa de sucesso significativamente inferior, registrou um tempo médio um pouco menor, com 38,43 segundos. Na Etapa 04(b), o SAN-DDPG apresentou um tempo médio de 43,04 segundos, e o DQN teve um tempo médio de 41,64 segundos.

Os resultados da etapa 04 demonstram que a abordagem SAN-DDPG é capaz de lidar com ambientes complexos, proporcionando que o robô navegue de forma eficiente em meio à pessoas em movimento, obtendo uma taxa média de sucesso superior a 80% nos testes realizados. Necessário observar que os testes foram aplicados em ambientes mais complexos que o ambiente de treinamento e sem qualquer cooperação dos demais agentes para evitar colisões.

5.3 Considerações finais

Testar o desempenho do agente em cenários diferentes do ambiente de treinamento é fundamental para avaliar sua capacidade de generalização. Se o agente só consegue tomar boas decisões no ambiente em que foi treinado, mas falha quando submetido a novos cenários, pode ser arriscado submetê-lo à aplicações do mundo real, onde poderá encontrar situações diferentes. Além disso, treiná-lo novamente, demanda tempo e uso de recursos computacionais que podem não estar disponíveis em determinadas situações, como por exemplo, simular o novo ambiente em que o robô deverá atuar.

Os resultados dos testes realizados demonstram a superioridade da abordagem SAN-DDPG em relação ao DQN. A abordagem SAN-DDPG obteve altas taxas de sucesso e baixa taxa de colisão em todas as etapas. Por outro lado, o DQN enfrentou dificuldades para atingir o objetivo em praticamente todos os cenários, registrando uma taxa de colisão consideravelmente maior em todas as etapas de testes.

É importante observar que o desempenho dos algoritmos pode variar dependendo da natureza da tarefa e de outros fatores, como hiperparâmetros, arquitetura de rede neural e estratégias de exploração empregadas. Apesar disso, os experimentos realizados neste trabalho evidenciam que a abordagem SAN-DDPG supera o DQN, demonstrando que a solução proposta é robusta e eficiente, proporcionando que o robô navegue de forma autônoma em ambientes dinâmicos, compartilhados com humanos, mesmo quando exposto à cenários distintos do ambiente de treinamento.

6 CONCLUSÃO

Este trabalho apresentou o desenvolvimento de uma solução baseada em aprendizado por reforço profundo e visão computacional para navegação autônoma de robôs móveis em ambientes dinâmicos compartilhados com humanos. Nesse sentido, a abordagem proposta considerou questões de segurança, como o distanciamento entre o robô e as pessoas em seu entorno, além de adaptação do aprendizado e generalização do conhecimento adquirido à diferentes situações.

Foram conduzidos testes comparativos entre a abordagem proposta por este trabalho, denominada *Social Attention Navigation - DDPG* e o algoritmo *Deep Q-Network* (DQN). Nos experimentos realizados a abordagem SAN-DDPG demonstrou ser mais eficiente e estável que o algoritmo DQN, apresentando taxas médias de sucesso superiores em todas as etapas analisadas: 98% (Etapa 01), 89% (Etapa 02), 86% (Etapa 03) e 86% (Etapa 04), demonstrando excelente capacidade de generalização e resultados consistentemente melhores em ambientes diferentes do ambiente de treinamento.

Apesar da variedade de soluções que abordam o uso do algoritmo DQN como uma solução viável para navegação autônoma em ambientes internos (Tabela 1), os resultados dos testes demonstram que o agente treinado pelo DQN não foi capaz de generalizar o conhecimento para aplicação em ambientes diferentes daquele no qual foi treinado. Além disso, a análise dos experimentos (Figuras 20a, 20b, 20c e 20d) demonstra que o DQN tende a estabilizar o aprendizado, além de necessitar de mais tempo de treinamento para convergir para uma boa solução.

Essas descobertas destacam a superioridade da abordagem SAN-DDPG e demonstram que a solução proposta é promissora, contribuindo para o avanço da pesquisa na área, possibilitando a análise de experimentos em ambiente simulado e realização de testes para posterior implantação de sistemas robóticos em cenários do mundo real.

6.1 Contribuições

Neste trabalho foi desenvolvida uma abordagem para navegação autônoma de robôs móveis em ambientes internos compartilhados com humanos, denominada Social Attention Navigation - DDPG. A arquitetura dessa abordagem é baseada em DRL e tem como base o algoritmo DDPG combinado com técnicas de visão computacional para detecção de pessoas.

Os experimentos foram conduzidos em ambiente simulado, permitindo que os modelos aprendidos sejam salvos e posteriormente utilizados em aplicações do mundo real. Os resultados demonstram que a abordagem proposta é eficaz, apresentando estabilidade e capacidade de aprendizado para lidar com diferentes cenários e situações. Além disso destacam-se as seguintes contribuições:

- Este trabalho contribui para o desenvolvimento de uma abordagem que combina técnicas de visão computacional e aprendizado por reforço profundo para o treinamento de robôs em ambiente simulado. Os experimentos realizados foram validados e demonstram que a abordagem é promissora, podendo ser estendida à diferentes plataformas robóticas. Além disso, o código fonte e os ambientes simulados implementados neste trabalho serão disponibilizados publicamente, podendo ser aprimorados e aplicados à outras soluções, contribuindo para o avanço do estado-da-arte;

- Além das vantagens óbvias de independência e mobilidade, o desenvolvimento de um sistema para navegação autônoma de cadeiras de rodas motorizadas tem o potencial de contribuir significativamente para a inclusão social. Ao permitir que pessoas com mobilidade reduzida participem ativamente de ambientes compartilhados, promove-se a normalização da interação entre indivíduos com diferentes capacidades, melhorando a qualidade de vida de pessoas com deficiência;

- O estudo e análise do comportamento do robô durante a navegação em meio a testes simulados oferece *insights* valiosos para aprimorar o aprendizado e avaliar sua capacidade de adaptação a ambientes em constante mudança, além de permitir identificar melhorias para a realização de experimentos futuros.

6.2 Publicações

P. de Almeida Afonso and P. R. Ferreira, “Autonomous robot navigation in crowd,” in 2022 Latin American Robotics Symposium (LARS), 2022 Brazilian Symposium on Robotics (SBR), and 2022 Workshop on Robotics in Education (WRE), 2022, pp. 139–144.

Artigo submetido: Autonomous Robots (Springer) Resarch Square - Autonomous Navigation of Wheelchairs in Indoor Environments using Deep Reinforcement Learning and Computer Vision, 28 August 2023, PREPRINT (Version 1). Available at Research Square [https://doi.org/10.21203/rs.3.rs-3287103/v1];

Artigo aceito: Autonomous Navigation of Wheelchairs in Indoor Environments using Deep Reinforcement Learning and Computer Vision. SBR-LARS 2023 (15th Brazilian Symposium on Robotics / 20th Latin American Robotics Symposium).

6.3 Trabalhos Futuros

Além de testar a abordagem proposta em cenários do mundo real, algumas questões foram identificadas e apontam direções para pesquisas futuras que podem contribuir para melhorar a solução apresentada neste trabalho:

- Um ponto fundamental a ser observado são as características particularmente associadas à plataforma robótica utilizada. Uma cadeira de rodas motorizada, por exemplo, pode chegar a 55cm de largura e 47cm de comprimento, exigindo maior espaço para interação no ambiente do que plataformas robóticas originalmente desenvolvidas para atuar como robôs de serviço. Nesse aspecto destaca-se uma das dificuldades enfrentadas neste trabalho, onde foi possível perceber que o ângulo de rotação (*yaw*) da cadeira de rodas pode levar a um número de colisões relativamente alto, constituindo um desafio para o algoritmo convergir para uma solução que proporcione uma navegação mais suave e segura. Dessa forma, devem ser estudados mecanismos relacionados aos movimentos diferenciais para obtenção de velocidades compatíveis com o ambiente de aplicação, especialmente em ambientes compartilhados com humanos;

- Durante os experimentos realizados foi possível perceber que o robô evitou colisões com pessoas em movimento de forma eficiente. No entanto, não foram respeitadas normas sociais, como por exemplo, evitar passar pela frente das pessoas, interrompendo sua passagem. Como o sistema implementado não previu colaboração por parte de outros agentes, essa foi uma das principais causas de colisões identificada, resultando na menor taxa de sucesso durante os testes realizados com o DDPG, conforme pode ser visto na tabela 10. Portanto, são necessários estudos complementares, capazes de identificar a orientação da pessoa em relação ao robô, constituindo mais uma informação para o algoritmo de aprendizado;

- A estrutura da cadeira de rodas também é um fator limitante para os sensores. Neste trabalho, o sensor LiDAR foi posicionado na frente da plataforma robótica,

reduzindo o ângulo de sensoriamento devido às dimensões da cadeira. Esse problema pode ser agravado em situações reais, onde haverá uma pessoa sentada na cadeira. Dessa forma, entende-se que devem ser utilizados sensores LiDAR também nas laterais da plataforma robótica. Para isso devem ser estudados meios de realizar a fusão dos dados obtidos a partir de vários sensores, objetivando ampliar a percepção do robô em relação ao ambiente de navegação que ele deverá atuar.

REFERÊNCIAS

ADIWAHONO, A. H. et al. Human tracking and following in dynamic environment for service robots. In: TENCON 2017 - 2017 IEEE REGION 10 CONFERENCE, 2017. **Anais...** [S.l.: s.n.], 2017. p.3068–3073.

ALAHI, A. et al. Social lstm: Human trajectory prediction in crowded spaces. In: IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, 2016. **Proceedings...** [S.l.: s.n.], 2016. p.961–971.

ALOM, M. Z. et al. A state-of-the-art survey on deep learning theory and architectures. **Electronics**, [S.l.], v.8, n.3, p.292, 2019.

ARULKUMARAN, K.; DEISENROTH, M. P.; BRUNDAGE, M.; BHARATH, A. A. Deep reinforcement learning: A brief survey. **IEEE Signal Processing Magazine**, [S.l.], v.34, n.6, p.26–38, 2017.

ARULKUMARAN, K.; DEISENROTH, M. P.; BRUNDAGE, M.; BHARATH, A. A. Deep Reinforcement Learning: A Brief Survey. **IEEE Signal Processing Magazine**, [S.l.], v.34, n.6, p.26–38, nov 2017.

BAREISS, D.; BERG, J. van den. Generalized reciprocal collision avoidance. **The International Journal of Robotics Research**, [S.l.], v.34, n.12, p.1501–1514, 2015.

BERA, A.; RANDHAVANE, T.; MANOCHA, D. Aggressive, Tense or Shy? Identifying Personality Traits from Crowd Videos. In: IJCAI, 2017. **Anais...** [S.l.: s.n.], 2017. p.112–118.

BERA, A.; RANDHAVANE, T.; PRINJA, R.; MANOCHA, D. Sociosense: Robot navigation amongst pedestrians with social and psychological constraints. In: INTELLIGENT ROBOTS AND SYSTEMS (IROS), 2017 IEEE/RSJ INTERNATIONAL CONFERENCE ON, 2017. **Anais...** [S.l.: s.n.], 2017. p.7018–7025.

BERG, J. Van den; LIN, M.; MANOCHA, D. Reciprocal velocity obstacles for real-time multi-agent navigation. In: IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION, 2008., 2008. **Anais...** [S.l.: s.n.], 2008. p.1928–1935.

BEYER, L.; HERMANS, A.; LEIBE, B. DROW: Real-Time Deep Learning-Based Wheelchair Detection in 2-D Range Data. **IEEE Robotics and Automation Letters**, [S.l.], v.2, n.2, p.585–592, 2017.

Beyer, L. et al. Deep Person Detection in Two-Dimensional Range Data. **IEEE Robotics and Automation Letters**, [S.l.], v.3, n.3, p.2726–2733, 2018.

BJELONIC, M. **YOLO ROS**: Real-Time Object Detection for ROS. `https://github.com/leggedrobotics/darknet_ros`.

Bresson, R.; Saraydaryan, J.; Dugdale, J.; Spalanzani, A. Socially Compliant Navigation in dense crowds. In: IEEE INTELLIGENT VEHICLES SYMPOSIUM (IV), 2019., 2019. **Anais...** [S.l.: s.n.], 2019. p.64–69.

Cadena, C. et al. Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age. **IEEE Transactions on Robotics**, [S.l.], v.32, n.6, p.1309–1332, 2016.

Cao, C.; Trautman, P.; Iba, S. Dynamic Channel: A Planning Framework for Crowd Navigation. In: INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION (ICRA), 2019., 2019. **Anais...** [S.l.: s.n.], 2019. p.5551–5557.

CESA-BIANCHI, N.; GENTILE, C.; LUGOSI, G.; NEU, G. Boltzmann exploration done right. In: ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS, 2017. **Anais...** [S.l.: s.n.], 2017. p.6284–6293.

CHEN, Y. F.; EVERETT, M.; LIU, M.; HOW, J. P. Socially aware motion planning with deep reinforcement learning. In: IEEE/RSJ INTERNATIONAL CONFERENCE ON INTELLIGENT ROBOTS AND SYSTEMS (IROS), 2017., 2017. **Anais...** [S.l.: s.n.], 2017. p.1343–1350.

CHEN, Y. F.; LIU, M.; EVERETT, M.; HOW, J. P. Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning. In: IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION (ICRA), 2017., 2017. **Anais...** [S.l.: s.n.], 2017. p.285–292.

Chen, Z. et al. Autonomous Social Distancing in Urban Environments Using a Quadruped Robot. **IEEE Access**, [S.l.], v.9, p.8392–8403, 2021.

CHIANG, H.-T. et al. Path-guided artificial potential fields with stochastic reachable sets for motion planning in highly dynamic environments. In: IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION (ICRA), 2015., 2015. **Anais...** [S.l.: s.n.], 2015. p.2347–2354.

Choi, J.; Park, K.; Kim, M.; Seok, S. Deep Reinforcement Learning of Navigation in a Complex and Crowded Environment with a Limited Field of View. In: INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION (ICRA), 2019., 2019. **Anais...** [S.l.: s.n.], 2019. p.5993–6000.

CIOU, P.-H. et al. Composite Reinforcement Learning for Social Robot Navigation. In: IEEE/RSJ INTERNATIONAL CONFERENCE ON INTELLIGENT ROBOTS AND SYSTEMS (IROS), 2018., 2018. **Anais...** [S.l.: s.n.], 2018. p.2553–2558.

DOUTHWAITE, J. A.; ZHAO, S.; MIHAYLOVA, L. S. A Comparative Study of Velocity Obstacle Approaches for Multi-Agent Systems. In: UKACC 12TH INTERNATIONAL CONFERENCE ON CONTROL (CONTROL), 2018., 2018. **Anais...** [S.l.: s.n.], 2018. p.289–294.

DRUZHKOV, P. N.; KUSTIKOVA, V. D. A survey of deep learning methods and software tools for image classification and object detection. **Pattern Recognition and Image Analysis**, [S.l.], v.26, p.9–15, 2016.

EVERETT, M.; CHEN, Y. F.; HOW, J. P. Motion planning among dynamic, decision-making agents with deep reinforcement learning. In: IEEE/RSJ INTERNATIONAL CONFERENCE ON INTELLIGENT ROBOTS AND SYSTEMS (IROS), 2018., 2018. **Anais...** [S.l.: s.n.], 2018. p.3052–3059.

FAN, T. et al. CrowdMove: Autonomous Mapless Navigation in Crowded Scenarios. **arXiv preprint arXiv:1807.07870**, [S.l.], 2018.

FENJIRO, Y.; BENBRAHIM, H. Deep Reinforcement Learning Overview of the state of the Art. **Journal of Automation, Mobile Robotics and Intelligent Systems**, [S.l.], v.2012, n.3, p.20–39, Dec. 2018.

FERRER, G.; SANFELIU, A. Anticipative kinodynamic planning: multi-objective robot navigation in urban and dynamic environments. **Autonomous Robots**, [S.l.], p.1–16, 2018.

FIORINI, P.; SHILLER, Z. Motion planning in dynamic environments using velocity obstacles. **The International Journal of Robotics Research**, [S.l.], v.17, n.7, p.760–772, 1998.

FUKUSHIMA, K. Neocognitron: A hierarchical neural network capable of visual pattern recognition. **Neural networks**, [S.l.], v.1, n.2, p.119–130, 1988.

GAYA, J. O. et al. Vision-based obstacle avoidance using deep learning. In: XIII LATIN AMERICAN ROBOTICS SYMPOSIUM AND IV BRAZILIAN ROBOTICS SYMPOSIUM (LARS/SBR), 2016., 2016. **Anais...** [S.l.: s.n.], 2016. p.7–12.

GIRSHICK, R.; DONAHUE, J.; DARRELL, T.; MALIK, J. Rich feature hierarchies for accurate object detection and semantic segmentation. In: IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, 2014. **Proceedings...** [S.l.: s.n.], 2014. p.580–587.

GU, X. et al. Intelligent Vehicle Path Planning Based on Improved Artificial Potential Field Algorithm. In: INTERNATIONAL CONFERENCE ON HIGH PERFORMANCE BIG DATA AND INTELLIGENT SYSTEMS (HPBD&IS), 2019., 2019. **Anais...** [S.l.: s.n.], 2019. p.104–109.

GUERRERO-HIGUERAS, Á. M. et al. Tracking people in a mobile robot from 2d lidar scans using full convolutional neural networks for security in cluttered environments. **Frontiers in neurorobotics**, [S.l.], v.12, p.85, 2019.

GUPTA, A. et al. Social gan: Socially acceptable trajectories with generative adversarial networks. In: IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, 2018. **Proceedings...** [S.l.: s.n.], 2018. p.2255–2264.

GYENES, Z.; SZADECZKY-KARDOSS, E. G. Motion planning for mobile robots using the safety velocity obstacles method. In: INTERNATIONAL CARPATHIAN CONTROL CONFERENCE (ICCC), 2018., 2018. **Anais...** [S.l.: s.n.], 2018. p.389–394.

HADFIELD-MENELL, D.; RUSSELL, S. J.; ABBEEL, P.; DRAGAN, A. Cooperative inverse reinforcement learning. In: ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS, 2016. **Anais...** [S.l.: s.n.], 2016. p.3909–3917.

HARIS, M.; GLOWACZ, A. Road Object Detection: A Comparative Study of Deep Learning-Based Algorithms. **Electronics**, [S.l.], v.10, n.16, 2021.

HASSELT, H. V. Double Q-learning. In: ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS, 2010. **Anais...** [S.l.: s.n.], 2010. p.2613–2621.

HE, K.; GKIOXARI, G.; DOLLÁR, P.; GIRSHICK, R. Mask r-cnn. In: IEEE INTERNATIONAL CONFERENCE ON COMPUTER VISION, 2017. **Proceedings...** [S.l.: s.n.], 2017. p.2961–2969.

HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. **Neural computation**, [S.l.], v.9, n.8, p.1735–1780, 1997.

KAELBLING, L. P.; LITTMAN, M. L.; MOORE, A. W. Reinforcement learning: A survey. **Journal of artificial intelligence research**, [S.l.], v.4, p.237–285, 1996.

KHATIB, O. Real-time obstacle avoidance for manipulators and mobile robots. In: **Autonomous robot vehicles**. [S.l.]: Springer, 1986. p.396–404.

KIM, S. et al. BRVO: Predicting pedestrian trajectories using velocity-space reasoning. **The International Journal of Robotics Research**, [S.l.], v.34, n.2, p.201–217, 2015.

KOUBÂA, A. et al. **Robot Operating System (ROS).** [S.l.]: Springer, 2017. v.1.

KRETZSCHMAR, H.; SPIES, M.; SPRUNK, C.; BURGARD, W. Socially compliant mobile robot navigation via inverse reinforcement learning. **The International Journal of Robotics Research**, [S.l.], v.35, n.11, p.1289–1307, 2016.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. **nature**, [S.l.], v.521, n.7553, p.436–444, 2015.

LECUN, Y.; BOTTOU, L.; BENGIO, Y.; HAFFNER, P. Gradient-based learning applied to document recognition. **Proceedings of the IEEE**, [S.l.], v.86, n.11, p.2278–2324, 1998.

LEE, D.; JEONG, J.; KIM, Y. H.; PARK, J. B. An improved artificial potential field method with a new point of attractive force for a mobile robot. In: INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION ENGINEERING (ICRAE), 2017., 2017. **Anais...** [S.l.: s.n.], 2017. p.63–67.

LEE, I.-K.; KIM, M.-S.; ELBER, G. The Minkowski sum of 2D curved objects. In: ISRAEL-KOREA BI-NATIONAL CONFERENCE ON NEW THEMES IN COMPUTERIZED GEOMETRICAL MODELING, 1998. **Proceedings...** [S.l.: s.n.], 1998. v.5, p.155–164.

LILLICRAP, T. P. et al. Continuous control with deep reinforcement learning. **arXiv preprint arXiv:1509.02971**, [S.l.], 2015.

LIN, T.-Y. et al. Focal loss for dense object detection. In: IEEE INTERNATIONAL CONFERENCE ON COMPUTER VISION, 2017. **Proceedings...** [S.l.: s.n.], 2017. p.2980–2988.

Lisotto, M.; Coscia, P.; Ballan, L. Social and Scene-Aware Trajectory Prediction in Crowded Spaces. In: IEEE/CVF INTERNATIONAL CONFERENCE ON COMPUTER VISION WORKSHOP (ICCVW), 2019., 2019. **Anais...** [S.l.: s.n.], 2019. p.2567–2574.

LIU, W. et al. Ssd: Single shot multibox detector. In: EUROPEAN CONFERENCE ON COMPUTER VISION, 2016. **Anais...** [S.l.: s.n.], 2016. p.21–37.

LIU, Y.; LIU, H.; WANG, B. Autonomous exploration for mobile robot using Q-learning. In: INTERNATIONAL CONFERENCE ON ADVANCED ROBOTICS AND MECHATRONICS (ICARM), 2017., 2017. **Anais...** [S.l.: s.n.], 2017. p.614–619.

LIU, Y.; XU, A.; CHEN, Z. Map-based Deep Imitation Learning for Obstacle Avoidance. In: IEEE/RSJ INTERNATIONAL CONFERENCE ON INTELLIGENT ROBOTS AND SYSTEMS (IROS), 2018., 2018. **Anais...** [S.l.: s.n.], 2018. p.8644–8649.

LONG, P.; LIU, W.; PAN, J. Deep-learned collision avoidance policy for distributed multi-agent navigation. **IEEE Robotics and Automation Letters**, [S.l.], v.2, n.2, p.656–663, 2017.

LONG, P. et al. Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning. In: IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION (ICRA), 2018., 2018. **Anais...** [S.l.: s.n.], 2018. p.6252–6259.

LORENTE, M.-T.; OWEN, E.; MONTANO, L. Model-based robocentric planning and navigation for dynamic environments. **The International Journal of Robotics Research**, [S.l.], v.37, n.8, p.867–889, 2018.

MALONE, N. et al. Hybrid dynamic moving obstacle avoidance using a stochastic reachable set-based potential field. **IEEE Transactions on Robotics**, [S.l.], v.33, n.5, p.1124–1138, 2017.

MNIH, V. et al. Playing atari with deep reinforcement learning. **arXiv preprint arXiv:1312.5602**, [S.l.], 2013.

MNIH, V. et al. Human-level control through deep reinforcement learning. **Nature**, [S.l.], v.518, n.7540, p.529, 2015.

MOHANTY, P. K.; SAH, A. K.; KUMAR, V.; KUNDU, S. Application of deep Q-learning for wheel mobile robot navigation. In: INTERNATIONAL CONFERENCE ON COMPUTATIONAL INTELLIGENCE AND NETWORKS (CINE), 2017., 2017. **Anais...** [S.l.: s.n.], 2017. p.88–93.

MUJAHED, M.; MERTSCHING, B. The admissible gap (AG) method for reactive collision avoidance. In: IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION (ICRA), 2017., 2017. **Anais...** [S.l.: s.n.], 2017. p.1916–1921.

OKUYAMA, T.; GONSALVES, T.; UPADHAY, J. Autonomous driving system based on deep q learnig. In: INTERNATIONAL CONFERENCE ON INTELLIGENT AUTONOMOUS SYSTEMS (ICOIAS), 2018., 2018. **Anais...** [S.l.: s.n.], 2018. p.201–205.

PANCHPOR, A. A.; SHUE, S.; CONRAD, J. M. A survey of methods for mobile robot localization and mapping in dynamic indoor environments. In: SIGNAL PROCESSING AND COMMUNICATION ENGINEERING SYSTEMS (SPACES), 2018 CONFERENCE ON, 2018. **Anais...** [S.l.: s.n.], 2018. p.138–144.

PATHAK, A. R.; PANDEY, M.; RAUTARAY, S. Application of deep learning for object detection. **Procedia computer science**, [S.l.], v.132, p.1706–1717, 2018.

PATIL, A. **Nuric Wheelchair Model**. Accessed: 2021-09-30, `https://github.com/patilnabhi/nuric_wheelchair_model_02`.

PFEIFFER, M. et al. Predicting actions to act predictably: Cooperative partial motion planning with maximum entropy models. In: IEEE/RSJ INTERNATIONAL CONFERENCE ON INTELLIGENT ROBOTS AND SYSTEMS (IROS), 2016., 2016. **Anais...** [S.l.: s.n.], 2016. p.2096–2101.

PFEIFFER, M. et al. A data-driven model for interaction-aware pedestrian motion prediction in object cluttered environments. In: IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION (ICRA), 2018., 2018. **Anais...** [S.l.: s.n.], 2018. p.1–8.

PRABHA, M. I.; UMARANI SRIKANTH, G. Survey of Sentiment Analysis Using Deep Learning Techniques. In: INTERNATIONAL CONFERENCE ON INNOVATIONS IN INFORMATION AND COMMUNICATION TECHNOLOGY (ICIICT), 2019., 2019. **Anais...** [S.l.: s.n.], 2019. p.1–9.

QIANG, L.; NANXUN, D.; HUICAN, L.; HENG, W. A model-free mapless navigation method for mobile robot using reinforcement learning. In: CHINESE CONTROL AND DECISION CONFERENCE (CCDC), 2018., 2018. **Anais...** [S.l.: s.n.], 2018. p.3410–3415.

REDMON, J.; DIVVALA, S.; GIRSHICK, R.; FARHADI, A. You only look once: Unified, real-time object detection. In: IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, 2016. **Proceedings...** [S.l.: s.n.], 2016. p.779–788.

REN, S.; HE, K.; GIRSHICK, R.; SUN, J. Faster r-cnn: Towards real-time object detection with region proposal networks. **Advances in neural information processing systems**, [S.l.], v.28, p.91–99, 2015.

REYES, E.; GÓMEZ, C.; NORAMBUENA, E.; SOLAR, J. Ruiz-del. Near real-time object recognition for pepper based on deep neural networks running on a backpack. In: ROBOCUP 2018: ROBOT WORLD CUP XXII 22, 2019. **Anais...** [S.l.: s.n.], 2019. p.287–298.

RIBEIRO, T. et al. Q-Learning for Autonomous Mobile Robot Obstacle Avoidance. In: IEEE INTERNATIONAL CONFERENCE ON AUTONOMOUS ROBOT SYSTEMS AND COMPETITIONS (ICARSC), 2019., 2019. **Anais...** [S.l.: s.n.], 2019. p.1–7.

BEN-ARI, M.; MONDADA, F. **Robots and Their Applications**. Cham: Springer International Publishing, 2018. p.1–20.

RODERICK, M.; MACGLASHAN, J.; TELLEX, S. Implementing the deep q-network. **arXiv preprint arXiv:1711.07478**, [S.l.], 2017.

RODRÍGUEZ-TEILES, F. G. et al. Vision-based reactive autonomous navigation with obstacle avoidance: Towards a non-invasive and cautious exploration of marine habitat. In: IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION (ICRA), 2014., 2014. **Anais...** [S.l.: s.n.], 2014. p.3813–3818.

RUAN, X.; REN, D.; ZHU, X.; HUANG, J. Mobile Robot Navigation based on Deep Reinforcement Learning. In: CHINESE CONTROL AND DECISION CONFERENCE (CCDC), 2019., 2019. **Anais...** [S.l.: s.n.], 2019. p.6174–6178.

RUBIO, F.; VALERO, F.; LLOPIS-ALBERT, C. A review of mobile robots: Concepts, methods, theoretical framework, and applications. **International Journal of Advanced Robotic Systems**, [S.l.], v.16, n.2, p.1729881419839596, 2019.

SALLAB, A. E.; ABDOU, M.; PEROT, E.; YOGAMANI, S. Deep reinforcement learning framework for autonomous driving. **Electronic Imaging**, [S.l.], v.2017, n.19, p.70–76, 2017.

Sasaki, Y.; Matsuo, S.; Kanezaki, A.; Takemura, H. A3C Based Motion Learning for an Autonomous Mobile Robot in Crowds. In: IEEE INTERNATIONAL CONFERENCE ON SYSTEMS, MAN AND CYBERNETICS (SMC), 2019., 2019. **Anais...** [S.l.: s.n.], 2019. p.1036–1042.

SCHAUL, T.; QUAN, J.; ANTONOGLOU, I.; SILVER, D. Prioritized experience replay. **arXiv preprint arXiv:1511.05952**, [S.l.], 2015.

SCHULMAN, J. et al. Proximal policy optimization algorithms. **arXiv preprint arXiv:1707.06347**, [S.l.], 2017.

SILVER, D. et al. Deterministic policy gradient algorithms. In: INTERNATIONAL CONFERENCE ON MACHINE LEARNING, 2014. **Anais...** [S.l.: s.n.], 2014. p.387–395.

SNAPE, J.; VAN DEN BERG, J.; GUY, S. J.; MANOCHA, D. The hybrid reciprocal velocity obstacle. **IEEE Transactions on Robotics**, [S.l.], v.27, n.4, p.696–706, 2011.

SUN, L.; ZHAI, J.; QIN, W. Crowd Navigation in an Unknown and Dynamic Environment Based on Deep Reinforcement Learning. **IEEE Access**, [S.l.], v.7, p.109544–109554, 2019.

SURMANN, H. et al. Deep reinforcement learning for real autonomous mobile robot navigation in indoor environments. **arXiv preprint arXiv:2005.13857**, [S.l.], 2020.

SUTTON, R. S. Introduction: The challenge of reinforcement learning. In: **Reinforcement Learning**. [S.l.]: Springer, 1992. p.1–3.

SUTTON, R. S. Reinforcement learning: Past, present and future. In: ASIA-PACIFIC CONFERENCE ON SIMULATED EVOLUTION AND LEARNING, 1998. **Anais...** [S.l.: s.n.], 1998. p.195–197.

TAI, L.; LI, S.; LIU, M. A deep-network solution towards model-less obstacle avoidance. In: IEEE/RSJ INTERNATIONAL CONFERENCE ON INTELLIGENT ROBOTS AND SYSTEMS (IROS), 2016., 2016. **Anais...** [S.l.: s.n.], 2016. p.2759–2764.

TAI, L.; LIU, M. A robot exploration strategy based on q-learning network. In: IEEE INTERNATIONAL CONFERENCE ON REAL-TIME COMPUTING AND ROBOTICS (RCAR), 2016., 2016. **Anais...** [S.l.: s.n.], 2016. p.57–62.

TAI, L.; LIU, M. Towards cognitive exploration through deep reinforcement learning for mobile robots. **arXiv preprint arXiv:1610.01733**, [S.l.], 2016.

VAN DEN BERG, J.; GUY, S. J.; LIN, M.; MANOCHA, D. Reciprocal n-body collision avoidance. In: **Robotics research**. [S.l.]: Springer, 2011. p.3–19.

VEMULA, A.; MUELLING, K.; OH, J. Modeling cooperative navigation in dense human crowds. In: ROBOTICS AND AUTOMATION (ICRA), 2017 IEEE INTERNATIONAL CONFERENCE ON, 2017. **Anais...** [S.l.: s.n.], 2017. p.1685–1692.

WANG, H.; BAN, X. Research on Autonomous Collision Avoidance Method of Unmanned Surface Vessel in the Circumstance of Moving Obstacles. In: CHINESE CONTROL CONFERENCE (CCC), 2018., 2018. **Anais...** [S.l.: s.n.], 2018. p.501–506.

WANG, R. et al. Convolutional recurrent neural networks for text classification. In: INTERNATIONAL JOINT CONFERENCE ON NEURAL NETWORKS (IJCNN), 2019., 2019. **Anais...** [S.l.: s.n.], 2019. p.1–6.

WANG, X.; SHRIVASTAVA, A.; GUPTA, A. A-fast-rcnn: Hard positive generation via adversary for object detection. In: IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, 2017. **Proceedings...** [S.l.: s.n.], 2017. p.2606–2615.

WATKINS, C. J.; DAYAN, P. Q-learning. **Machine learning**, [S.l.], v.8, n.3-4, p.279–292, 1992.

WU, J.; SHIN, S.; KIM, C.-G.; KIM, S.-D. Effective lazy training method for deep q-network in obstacle avoidance and path planning. In: IEEE INTERNATIONAL CONFERENCE ON SYSTEMS, MAN, AND CYBERNETICS (SMC), 2017., 2017. **Anais...** [S.l.: s.n.], 2017. p.1799–1804.

WU, P. et al. A novel obstacle avoidance strategy of nonholonomic mobile robot based on virtual simulation platform. In: IEEE INTERNATIONAL CONFERENCE ON INFORMATION AND AUTOMATION, 2015., 2015. **Anais...** [S.l.: s.n.], 2015. p.185–190.

WULFMEIER, M. et al. Large-scale cost function learning for path planning using deep inverse reinforcement learning. **The International Journal of Robotics Research**, [S.l.], v.36, n.10, p.1073–1087, 2017.

XIAO, Y. et al. Deep learning for occluded and multi-scale pedestrian detection: A review. **IET Image Processing**, [S.l.], v.15, n.2, p.286–301, 2021.

XUE, X.; LI, Z.; ZHANG, D.; YAN, Y. A Deep Reinforcement Learning Method for Mobile Robot Collision Avoidance based on Double DQN. In: IEEE 28TH INTERNATIONAL SYMPOSIUM ON INDUSTRIAL ELECTRONICS (ISIE), 2019., 2019. **Anais...** [S.l.: s.n.], 2019. p.2131–2136.

YAN, Z.; DUCKETT, T.; BELLOTTO, N. Online learning for 3D LiDAR-based human detection: Experimental analysis of point cloud clustering and classification methods. **Autonomous Robots**, [S.l.], v.44, n.2, p.147–164, 2020.

YANG, S.; LI, C. Behavior Control Algorithm for Mobile Robot Based on Q-Learning. In: INTERNATIONAL CONFERENCE ON COMPUTER NETWORK, ELECTRONIC AND AUTOMATION (ICCNEA), 2017., 2017. **Anais...** [S.l.: s.n.], 2017. p.45–48.

ZENG, F.; WANG, C.; GE, S. S. A Survey on Visual Navigation for Artificial Agents With Deep Reinforcement Learning. **IEEE Access**, [S.l.], v.8, p.135426–135442, 2020.

Zhang, D.; Han, X.; Deng, C. Review on the research and practice of deep learning and reinforcement learning in smart grids. **CSEE Journal of Power and Energy Systems**, [S.l.], v.4, n.3, p.362–370, 2018.

ZHANG, D. et al. Real-time navigation in dynamic human environments using optimal reciprocal collision avoidance. In: IEEE INTERNATIONAL CONFERENCE ON MECHATRONICS AND AUTOMATION (ICMA), 2015., 2015. **Anais...** [S.l.: s.n.], 2015. p.2232–2237.

ZUO, G.; DU, T.; LU, J. Double DQN method for object detection. In: CHINESE AUTOMATION CONGRESS (CAC), 2017., 2017. **Anais...** [S.l.: s.n.], 2017. p.6727–6732.

Apêndices

APÊNDICE A – Links para acesso aos vídeos de treinamento e testes

A seguir são disponibilizados os links para acesso a alguns vídeos de treinamento e de testes. Devido ao tempo necessário para treinamento e execução dos experimentos, os vídeos disponibilizados apresentam demonstrações parciais dos experimentos realizados.

Tabela 11 – Vídeos de treinamento

Etapa de treinamento		
Etapa	DQN	DDPG
02	https://youtu.be/xCCiAXp4n5U	https://youtu.be/-n2vzooPZCo
03	https://youtu.be/nsUqLHYcNCE	https://youtu.be/v4lNtb0nZBo
04	https://youtu.be/BXsDgBBzpeE	https://youtu.be/VqqsS1RywMI

Tabela 12 – Vídeos de testes

Testes utilizando o algoritmo DDPG	
Etapa 02a	https://youtu.be/yKeqTDyQClQ
Etapa 02b	https://youtu.be/mZbnFQYCNX4
Etapa 03a	https://youtu.be/fVLRNMI-VCg
Etapa 03b	https://youtu.be/2LDr_yqhIn8
Etapa 04a	https://youtu.be/2qPFWKNTdgE
Etapa 04b	https://youtu.be/UGKHEvG3vc8

Link para acesso ao código fonte: https://github.com/poolafonso/deepbot