UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

RENIÊ DE AZEVEDO DELGADO

APRENDIZAGEM DE COMPORTAMENTOS EM ROBÔS ATRAVÉS DE APRENDIZAGEM POR REFORÇO

Recife
2019

Reniê de Azevedo Delgado

Aprendizagem de Comportamentos em Robôs Através de Aprendizagem por Reforço

Trabalho apresentado ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: Inteligência Computacional
Orientador: Hansenclever de França Bassani

Recife
2019

Catalogação na fonte
Bibliotecário Cristiano Cosme S. dos Anjos, CRB4-2290

D352a Delgado, Reniê de Azevedo
Aprendizagem de comportamentos em robôs através de aprendizagem reforço/ Reniê de Azevedo Delgado. – 2019.
87 f.: il., fig., tab.

Orientador: Hansenclever de França Bassani.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2019 .
Inclui referências.

1. Inteligência computacional. 2. Robótica. 3. Aprendizagem de Máquina. 4. Aprendizagem por reforço. I. Bassani, Hansenclever de França (orientador). II.Título.

006.31 CDD (22. ed.) UFPE-CCEN 2021-67

Reniê de Azevedo Delgado

“Aprendizagem de Comportamentos em Robôs Através de Aprendizagem por Reforço”

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Aprovado em: 30 de agosto de 2019.

BANCA EXAMINADORA

_____

Prof. Dr. Paulo Salgado Gomes de Mattos Neto
Centro de Informática/UFPE

_____

Prof. Dr. João Fausto Lorenzato de Oliveira
Escola Politécnica/UPE

_____

Prof. Dr. Hansenclever de França Bassani
Centro de Informática/UFPE
(**Orientador)**

Dedico este trabalho primeiramente a minha família, namorada e amigos que, mesmo que de longe, foram meus pilares durante todo o percurso sempre os carreguei comigo nessa trajetória. Gostaria de dedicar também a meu orientador Hansenclever que me ajudou e aconselhou por toda a jornada e sem o qual este trabalho não seria possível.

AGRADECIMENTOS

Agradeço a todos que contribuíram, mesmo que indiretamente, para a conclusão deste trabalho. Em especial deixo o agradecimento à minha família por sempre ter me dado uma base sólida para que eu pudesse conseguir seguir o meu caminho sem preocupações, aos meus amigos e namorada que me deram apoio e foram também muito importantes nesta jornada e ao meu orientador Hansenclever, sem todo o esforço e ajuda prestada por ele este trabalhado não seria possível.

RESUMO

A sociedade vem passando por mudanças radicais nas últimas décadas. Cada vez mais, aparelhos inteligentes surgem no nosso dia-a-dia com o intuito de nos trazer comodidades. Empresas que atuam em diversas áreas de mercado tem investido cada vez mais em algoritmos de inteligência artificial. Apesar dos enormes avanços da última década, os algoritmos mais modernos ainda estão longe de construir, generalizar e inferir conhecimentos como humanos. Essas limitações por muitas vezes limitam o escopo que esses algoritmos podem atuar e trazem vulnerabilidades neles. Para que máquinas possam realmente estar presentes nos mais diversos ambientes do cotidiano elas precisam aprender a interagir com o mundo e se adaptar a ele. Robôs inteligentes são agentes que conseguem inferir conhecimentos a partir das observações retiradas do seu ambiente que garantam a autonomia do robô em executar a tarefa. O controle do robô do seu próprio corpo de forma adequada é uma característica fundamental, que deve ser aprendida por qualquer agente que precise atuar em um ambiente. Um agente com essas características pode ser aplicado em diversas tarefas. Esta dissertação utiliza aprendizagem de máquina, prioritariamente com o paradigma de aprendizagem por reforço, para estudar como agentes se comportam em ambientes dinâmicos e complexos para realizar uma tarefa comum a todos. O intuito é, posteriormente, aplicar a melhor técnica estudada em robôs reais e participar de uma competição real para avaliar o desempenho da estratégia aprendida. Este trabalho visa investigar e contribuir para o avanço da área de aprendizagem de comportamentos para o mundo real, construindo um ambiente de aprendizagem por reforço fiel à realidade e analisando sempre o *tradeoff* entre dificuldade de simulação e velocidade de aprendizagem. Utilizando o ambiente criado treinar agentes simulados capazes de performar bem no jogo de futebol de robôs e conseguir transferir o comportamento aprendido para um robô real de forma que seu comportamento fique fiel ao aprendido em simulação.

Palavras-chaves: Robótica. Aprendizagem de Máquina. Aprendizagem por Reforço. Futebol de Robôs. Redes Neurais.

ABSTRACT

Society has been undergoing radical changes in recent decades with smart devices continuously emerging in our daily lives. Companies operating in the most broad market areas are increasingly investing in artificial intelligence algorithms. Despite last decade advances, the most modern algorithms are still far from building, generalizing and inferring human knowledge. These limitations often limit the scope in which these algorithms can act on and bring vulnerabilities to them. For machines to really be present in the most diverse environments of everyday life they need to learn to interact with and adapt to the world. Intelligent robots are agents able to infer knowledge from environment observations that guarantees their autonomy to perform the task. Properly controlling your own robotic body is a key feature for any agent who needs to work in an environment. An agent with these characteristics can be applied to several tasks. This dissertation uses machine learning, primarily with the reinforcement learning paradigm, to study how agents behave in dynamic and complex environments to accomplish a common task. The aim is to apply the best technique studied in real robots and participate in a real competition to evaluate the performance of the chosen strategy. This work aims to investigate and contribute to the advance of applied behavioral learning, building a learning environment for reinforcement that is true to reality and analyzing the tradeoff between simulation difficulty and learning speed. Using the environment created, train simulated agents capable of performing well in the robot soccer game and being able to transfer the learned behavior to a real robot guaranteeing that its behaviour is faithful to the simulated.

Keywords:Robotics Machine learning. Reinforcement Learning.Cognition. Action Policy. Robot Soccer. Neural networks. DQN. DDPG.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AGI	Inteligência Geral Artificial (do inglês Artificial General Intelligence)
DDPG	Deep Deterministic Policy Gradients
DDQN	Double Deep Q Networks
DNN	Redes Neurais Profundas (do inglês, Deep Neural Networks)
DQN	Deep Q Networks
IEEE-VSSS	IEEE Very Small Size Soccer
MDP	Markovian Decision Process
MLP	Multilayer Perceptron
MSE	Mean Squared Error
Q	Função Q
RDD	Robô de Direção Diferencial
RL	Aprendizagem por Reforço (do inglês, Reinforcement Learning)
RNA	Redes Neurais Artificiais
RobôCIn	Equipe de Robótica do Centro de Informática - UFPE
TC	Camada de Neurônios Totalmente Conectada
V	Função Valor

LISTA DE SÍMBOLOS

x, y	Coordenadas no plano cartesiano.
θ	Ângulo que determina orientação do robô.
ξ	Pose do robô.
R	Matriz Rotacional.
v, ω	Velocidade linear e angular.
$k_\rho, k_\alpha, k_\beta$	Constantes de Controle.
ρ, α, β	Posição em coordenadas polares.
ϕ	Velocidade angular das rodas do robô.
π	Política de ações de um agente.
s, o	Estado e observação em um processo de decisão markoviano.
a	Ação do agente.
r	Recompensa recebida ao aplicar uma ação.
γ	Fator de desconto da recompensa.
ϵ	Fator de aleatoriedade da política para DDQN.
ψ	Conjunto de parâmetros ajustáveis.
E	Esperança.
J	Função objetivo DDPG.
$\mathcal{N}$	Ruído normal.
σ, γ	Constantes do *Ornstein-Uhlenbeck Process*.
W	Processo de Wiener.
t_x, t_y	Coordenadas cartesianas da posição objetivo do robô.
m_R, gbp_R, g_R	Componentes que formam a recompensa total do ambiente.
$\omega_m, \omega_g bp, \omega_g$	Pesos associados às componentes da recompensa.
R_t	Recompensa total recebida por uma ação no ambiente no tempo t.

SUMÁRIO

1 INTRODUÇÃO

Inteligência artificial está cada vez mais se tornando presente em produtos do cotidiano da nossa sociedade. Serviços inteligentes estão surgindo com o intuito de nos trazer comodidades, seja nos propor um filme para assistir, um produto para comprar, uma viagem de final de ano ou até um restaurante novo para comer. Esses serviços usam algoritmos de inteligência artificial junto com os dados coletados de seus usuários para se adaptar cada vez melhor a seus gostos e padrões para oferecer um produto melhor e mais personalizado. Empresas que atuam em diversas áreas de mercado tem investido cada vez mais em algoritmos de inteligência artificial, entre alguns casos mais notórios estão os algoritmos de buscas e recomendações da Netflix (GOMEZ-URIBE; HUNT, 2015), os avanços na construção de agentes inteligentes do Google Deep Mind (SILVER et al., 2016) e (HASSELT; GUEZ; SILVER, 2016a) ou em visão do Facebook AI Research (SERMANET et al., 2013). Estes e outros avanços mostram como a inteligência artificial deixou de ser uma matéria de apenas estudos para trazer soluções práticas para nosso dia-a-dia.

Apesar dos enormes avanços da última década, os algoritmos mais modernos ainda estão longe de construir, generalizar e inferir conhecimentos como humanos. Esse fato por muitas vezes limita o escopo no qual esses algoritmos podem atuar e trazem vulnerabilidades a eles. Para que máquinas possam realmente estar presentes nos mais diversos ambientes do cotidiano elas precisam aprender a interagir com o mundo e se adaptar a ele. O principal problema está em quanto a mesma situação pode mudar e como se adaptar a isto, por exemplo: um ato simples, como lavar a louça, envolve percepções muito complexas para uma máquina, que passam desapercebidas de tão naturais para elas são para nós, como perceber a forma do prato, entender que ele está mais escorregadio por causa do sabão e entender quando um prato está realmente limpo.

O Capítulo 1 começa introduzindo conceitos sobre o que é o conhecimento e porque os agentes que temos hoje ainda não podem ser considerados realmente inteligentes. Esta discussão serve para guiar o objetivo final de qualquer um que queira ajudar na construção de uma inteligência artificial geral. A Seção 1.2 descreve o problema escolhido para este trabalho e as dificuldades que acompanham a criação de agentes inteligentes. A Seção 1.3 introduz as contribuições esperadas desta Dissertação. Por último, na Seção 1.4, será informada como esta Dissertação foi dividida e o que foi incluído em cada capítulo de forma geral.

1.1 CONSTRUÇÃO DO CONHECIMENTO

Embora os algoritmos de aprendizagem de máquina estejam distantes de obter uma inteligência generalista o suficiente para estar presente em todos os aspectos da nossa sociedade,

é importante definir o que esperamos de um agente generalista.

Cientistas estudam como replicar a forma com que os humanos aprendem, representam e repassam o conhecimento há séculos. Mesmo antes do surgimento da computação, algoritmos e modelos eram criados para descrever os comportamentos observados em humanos e animais. Em *Computing Machines and Intelligence*, (**??**), Alan Turing reflete sobre a capacidade de máquinas de pensar. Turing define duas orientações a serem seguidas na busca pela inteligência artificial. A primeira orientação refere-se ao estudo de comportamentos humanos, como ensinar algo a uma criança. A segunda orientação endereça o processo do pensamento abstrato, como o ato de jogar xadrez.

Em Inteligência artificial, máquinas devem agir racionalmente para alcançar um objetivo. O objetivo da inteligência artificial pode, então, ser definido como: criar um agente racional cujas ações gerem as melhores soluções ou melhores soluções esperadas, caso haja incertezas. Então, podemos inferir que, para nós, inteligência necessita tanto de ação quanto de raciocínio.

Stuart Russel e Peter Norvig, (RUSSELL; NORVIG, 2016), se referem a indução como a fonte do conhecimento. Indução consiste em a partir de observações indiciosas e gerar uma generalização que permita uma conclusão. A Figura 1 mostra o fluxo do processo indutivo.

Figura 1 – Processo Indutivo. Conhecimento particular é generalizado para criar hipóteses que darão sustentação a teorias.

Fonte: Autor (2019).

Aprendizagem de Máquina, a maior vertente atual da inteligência artificial, utiliza indução para criar seu conhecimento. O problema é que indução pode levar a criação de conhecimentos incorretos, caso suas observações não ofereçam todo conhecimento necessário. A parábola da galinha, do filósofo e matemático Bertrand Russel, explica isso: uma galinha vê todo dia o fazendeiro trazê-la comida, logo, a galinha induz que o fazendeiro vai sempre lhe trazer comida. Então a teoria da galinha vai morrer junto com ela no dia que o fazendeiro decidir comê-la. Assim como a galinha, vários algoritmos de aprendizagem de máquina falham ao reconhecer esse viés em seus dados. Karl Popper, um importante filósofo da ciência do conhecimento, defende que os esforços sejam sempre direcionados em achar evidências que refutem a teoria e não em evidências que a sustentem.

1.2 CONTEXTUALIZAÇÃO

Robôs inteligentes podem ser definidos como agentes que inferem conhecimento a partir de observações do ambiente do problema a fim de executar a tarefa da melhor forma conseguem. Um robô inteligente deve ser capaz de utilizar as ferramentas disponíveis do ambiente para melhorar a sua iteração, e consequentemente, os seus resultados no ambiente. Um agente com essas características pode ser aplicado em diversas tarefas. Ainda não se conhece como o aprendizado dessas capacidades em seres vivos ocorre, mas modelos recentes que tentam recriar a capacidade biológica de aprender estão sendo propostos e conseguindo sucesso em situações específicas.

Outras capacidades esperadas de um agente inteligente capaz de desempenhar tarefas no nosso dia-a-dia são: cooperação, planejamento, previsão, etc. Todas as capacidades mencionadas até agora são devidamente exploradas por um humano em diversos momentos da sua vida, por exemplo em diversos jogos. Normalmente, em um jogo é necessário que o agente raciocine sobre o ambiente para tomar a melhor ação possível, a fim de ganhar pontos que podem significar garantir a autonomia do agente no ambiente do jogo. O cenário de jogos é um ambiente controlado, com objetivos bem definidos e realizáveis, em que, normalmente as tarefas podem ser subdivididas em tarefas menores mais fáceis. Por essas características os jogos são muito utilizados para avaliar a capacidade de aprendizagem de um agente.

Para um agente desempenhar bem em um ambiente de futebol de robôs real como o da categoria *Very Small Size Soccer*, explicado em 2.1, ele precisará ter desenvolvido bem todas capacidades cognitivas mencionadas anteriormente. Este ambiente requer que 3 agentes trabalhem cooperativamente contra outros 3 adversários para colocar a bola dentro do gol adversário, e evitar que a bola seja levada ao seu gol. Os times são livres para escolher as funções dos seus agentes e como eles devem se comportar em jogo, desde que eles obedeçam às regras da competição (COMPETITION, 2008). Por ser um ambiente altamente dinâmico e complexo, esta tarefa possui um grau de dificuldade elevado para as técnicas de estado da arte de aprendizagem de máquina atual.

A área de robótica naturalmente apresenta dificuldades quanto a disponibilidade de dados, já que coletar dados para o treinamento pode ser uma difícil tarefa devido às características e limitações físicas de qualquer robô. Os algoritmos atuais de aprendizagem precisam de muitos dados para convergirem, por isso é necessário a interação do robô com o ambiente repetidas vezes para coletar dados. A não disponibilidade de robôs físicos e as dificuldades de se operar com estes no mundo real de forma adequada faz com que seja necessária a criações de simuladores ou modelos para ajudar na tarefa.

Aprender através de simuladores, que normalmente são simplificações do cenário real, introduz, porém, a dificuldade de transferir o conhecimento aprendido no cenário simulado para ser aplicado ao cenário real com eficácia. Fazer com que agentes aprendam a se comportar perante a uma dada dificuldade é uma habilidade com muitas aplicações

no mundo real, se bem sucedida, por exemplo: tarefas domésticas, trabalho em lugares perigosos, como minas, trabalhos repetitivos, trabalhos fisicamente desgastantes, etc. Nos anos recentes o paradigma por Reforço vem sendo muito utilizado para a tarefa de aprendizagem de comportamentos e tem obtido bons resultados(MNIH et al., 2013; PLAPPERT et al., 2018a; SILVER et al., 2018). Porém ainda é escasso o número ambientes de estudo dinâmicos e complexos disponíveis para estudar o comportamento agentes que possam ser aplicados em cenários reais. Até o presente conhecimento do autor, poucos trabalhos já publicados na área têm ambientes que se equiparam ao mundo real em dinâmica e complexidade. Algumas simplificações do ambiente de futebol de robôs são feitas. Em (BALCH et al., 1997) os agentes aprendem a escolher entre comportamentos previamente definidos para cada robô móvel real, como o comportamento de ir na direção da bola. O agente neste trabalho não precisa aprender a controlar seu corpo diretamente. Em (BRAIN, 2019), humanoides simulados com conjuntos de ações abstratos, como chutar a bola ou correr em determinada direção, tentam trabalhar em times para conseguir desempenhar bem. Neste ambiente não há preocupação em evoluir para um cenário de futebol no mundo real. Em (LIU et al., 2019), os robôs controlam suas velocidades lineares e angulares para jogar futebol, neste ambiente a física do mundo real não é uma preocupação e os robôs podem, inclusive, sobrepor suas posições.

Esta dissertação utiliza aprendizagem de máquina, com o paradigma de aprendizagem por reforço, para estudar como agentes se comportam em ambientes dinâmicos e complexos para realizar uma tarefa comum a todos. O intuito é, posteriormente, aplicar a melhor técnica estudada em robôs reais e participar de uma competição real e avaliar o desempenho da estratégia aprendida.

1.3 OBJETIVOS E CONTRIBUIÇÕES

Esta dissertação visa investigar e contribuir para o avanço da área de aprendizagem de comportamentos para o mundo real, construindo um ambiente simulado de aprendizagem por reforço e analisando sempre o *tradeoff* entre dificuldade de simulação e velocidade de aprendizagem. Achar um equilíbrio entre essas duas variáveis para que um agente consiga aprender um bom comportamento rapidamente e consiga que este conhecimentos seja extrapolado e aplicável para um cenário do mundo real.

O ambiente construído deve ser estável, rápido e confiável suficiente para que qualquer pessoa consiga treinar seu agente no ambiente de maneira eficiente. O ambiente construído pode ser utilizado para o estudo de diversas tarefas de aprendizagem por reforço, como: estudo de recompensas esparsas, cooperação e competição com múltiplos agentes, etc. No futuro, o ambiente deverá ser utilizado por diversas equipes de robótica para treinar e estudar os agentes em diversos ambientes diferentes competições de robótica.

Este trabalho visa estudar o treinamento de diferentes agentes simulados no ambiente e entender as características adquiridas pelos agentes durante a aprendizagem, assim como

seus pontos fortes e fracos. Os agentes criado precisam ser robustos e ter comportamentos inteligentes esperados para um robô no ambiente de futebol de robôs.

1.4 ORGANIZAÇÃO

Esta dissertação foi dividida da seguinte forma:

O Capítulo 2 introduz os conceitos mais importantes utilizados durante a dissertação. Este capítulo apresenta o ambiente de jogo ao qual os agentes serão submetidos durante sua aprendizagem, assim como as equações de cinemática e controle, e como tradicionalmente é estruturada a formação de estratégia para estes agentes. Além disso, o capítulo trata de maneira breve de aprendizagem profunda em robótica através de redes neurais artificiais, fala de como são os ambientes simulados mais famosos atualmente disponíveis na literatura que se assemelham ao proposto nesta dissertação e porque eles não lidam com todos os problemas tratados pelo ambiente proposto.

O Capítulo 3 é dedicado a discutir aprendizagem de comportamentos utilizando o paradigma por reforço. Nele são discutidas as principais técnicas utilizadas em aprendizagem por reforço, assim como seus conceitos e fórmulas. O capítulo também apresenta os algoritmos utilizados para o treinamento de agentes nesta dissertação, assim como uma discussão do porquê estes algoritmos foram escolhidos no primeiro momento deste trabalho.

O Capítulo 4 explica como foram implementadas as contribuições deste trabalho e como foram testadas, assim como os aspectos mais importantes de cada uma das decisões tomadas.

No Capítulo 5 serão apresentados todos os resultados do trabalho tanto para o ambiente proposto quanto para os agentes treinados. Neste capítulo, serão discutidos os resultados através de gráfico e serão explicados os comportamentos aprendidos pelos robôs nos jogos simulados.

O Capítulo 6 traz um resumo do que foi feito no trabalho com as principais conclusões que foram possíveis tirar a partir dos resultados e elenca uma série de oportunidades de trabalhos futuros que podem ser construídos a partir desta dissertação.

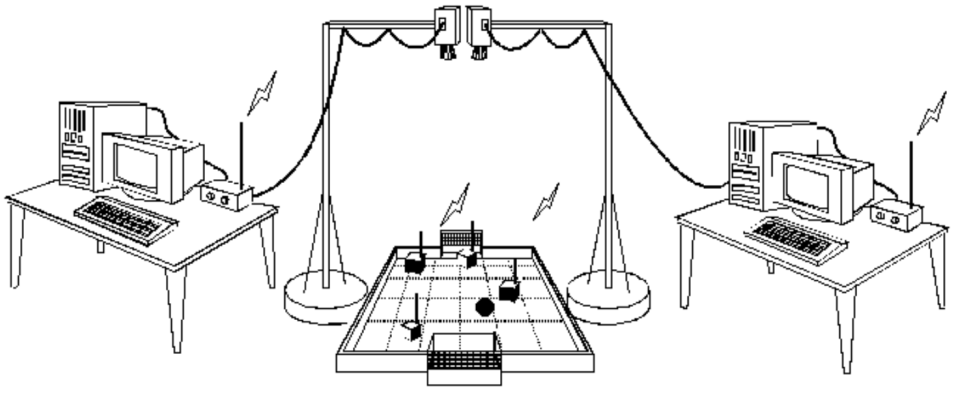
2 REFERENCIAL TEÓRICO

Este Capítulo fornece conceitos importantes sobre a área e problema abordado nesta Dissertação, assim como algumas contribuições recentes que mostram como a área está sendo explorada pela comunidade e sua relevância em trabalhos muito recentes. O Capítulo começa detalhando a competição de robótica, na Seção 2.1, que servirá como cenário básico explorado nesse trabalho, apresentando as dificuldades e , na Seção 2.2, complexidades em se traçar uma estratégia nesse cenário e como a dinâmica do robô pode ser modelada para seu controle. Em seguida, na Seção 2.3, a relevância e atualidade desta Dissertação são destacadas através da demonstração de alguns trabalhos recentes da comunidade em aprendizagem profucanda e elenca, na Seção 5.1, o uso de ambientes simulados para robótica que se assemelham ao criado nesta dissertação.

2.1 IEEE - *VERY SMALL SIZE SOCCER*

A categoria de futebol de robôs IEEE Very Small Size Soccer (IEEE-VSSS) constitui-se de um jogo de futebol entre dois times de três robôs. Cada robô deve ter no máximo 7.5x7.5x7.5 de dimensões. Os robôs devem ser controlados por computador sem intervenção humana. O computador deve processar o fluxo de vídeo recebido de uma câmera posicionada acima do campo e enviar comandos aos robôs, como mostrado na Figura 2.

Figura 2 – Jogo de IEEE-VSSS.



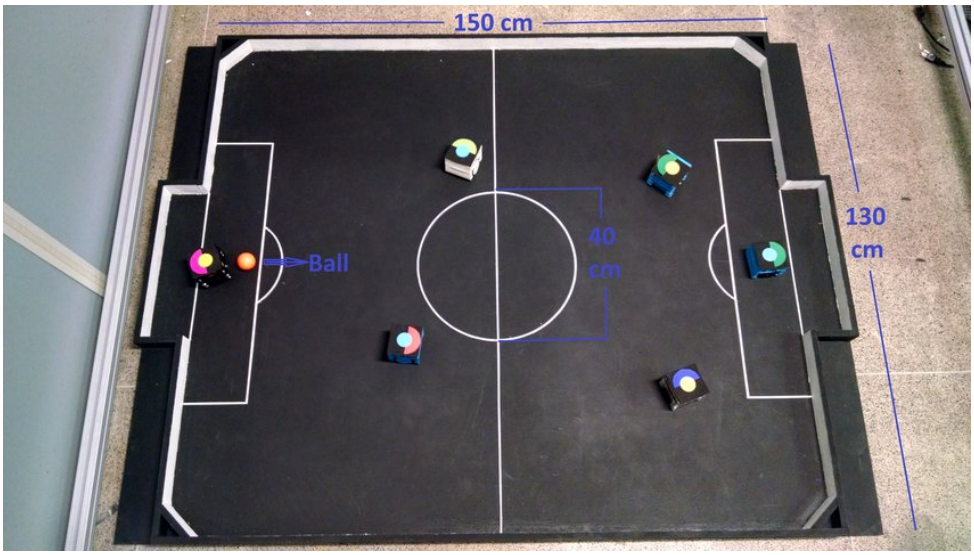
Fonte: (COMPETITION, 2008).

O campo dessa categoria é composto por uma chapa plana e ríigida de madeira medindo 150cm×130cm, em cor preta fosca com laterais medindo 5cm de altura por 2.5cm de largura. A parte superior das bordas de contenção laterais deverá ser da mesma cor do campo, e as paredes deverão ser na cor branca. Quatro triângulos isósceles são fixados às quinas do campo para evitar que a bola fique retida nestes locais. Os gols tem dimensões de 40cm de largura por 10cm de profundidade. Deverá ser utilizada uma bola de golfe

laranja, com 42.7mm de diâmetro e 46g de massa. Cada partida terá dois períodos iguais, cada qual com 5 minutos, com um intervalo de meio tempo de 10 minutos.

Um gol deverá ser computado quando a bola atravessar completamente a linha do gol. Como em uma partida tradicional de futebol, o time vencedor de um jogo é determinado com base no número de gols computados (saldo). Mais regras e detalhes são encontrados no livro oficial de regras da categoria (COMPETITION, 2008). A Figura 3 ilustra um campo da categoria com dois times completos diferentes.

Figura 3 – Campo para a categoria VSSS com dois times completos diferentes em campo.



Fonte: (SANTOS et al., 2015).

2.1.1 Definição da Estratégia de Jogo

A estratégia de cada equipe de robôs é uma grande parte das contribuições de cada participante da competição, cada equipe faz ajustes finos nas suas estratégias buscando antever cada um dos possíveis cenários de jogo. Isto torna o processo de construir uma estratégia um processo muito árduo e frenquentemente há estados de jogo que não foram previstos pelas equipes, resultando em comportamentos muitas vezes nocivos aos objetivos. Nesta parte da Dissertação será descrita a estratégia construída pela equipe Equipe de Robótica do Centro de Informática - UFPE (RobôCIn) em seu *Team Description Paper* (ROBôCIN, 2018a) para a competição LARC 2018 (outras equipes de ponta dividem suas estratégias em processos semelhantes).

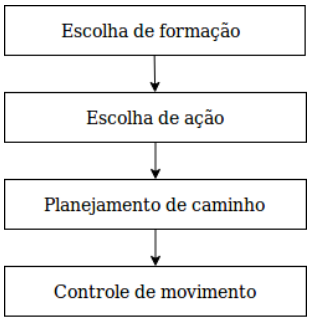
Cada equipe deve ter 4 módulos principais: detecção de objetos, agente físico, comunicação e planejamento de ações. O módulo de detecção de objetos deve fornecer as informações do estado do ambiente como entrada para o sistema. Robôs capazes de desempenhar cada ação conforme foi comandado. Um módulo de comunicação entre o computador que

estará executando a política de ações e cada robô controlado. O módulo de planejamento de ações define a política de ações de cada agente em campo.

O módulo de planejamento de ações é responsável por determinar as melhores ações e caminho à serem tomados pelos robôs, utilizando as informações fornecidas pelo módulo de detecção de objetos, como a posição, orientação e velocidade dos robôs e bola presentes em campo. É importante que o módulo de estratégia seja rápido e preditivo, de forma a prover uma reação rápida dos robôs em função das situações de jogo e assim maximizar a probabilidade de vitória.

A equipe do RobôCIn tradicionalmente separa este módulo em camadas, nomeadas: escolha de formação, escolha de ação, planejamento de caminho e controle de movimento. Tais camadas são executadas em sequência e tem como saída final as velocidades de roda que devem ser enviadas para cada robô de forma a executar o plano de jogo, como pode ser observado na Figura 4.

Figura 4 – Fluxo das camadas do módulo de estratégia.



Fonte: (ROBôCIN, 2018b).

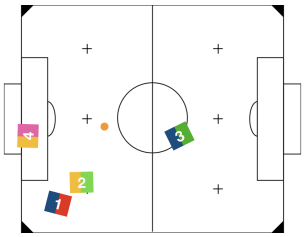
2.1.1.1 Escolha de Formação

Para explicar a camada de escolha de formação, é necessário primeiramente introduzir o conceito de comportamento. O comportamento de um robô é definido como o conjunto de relações entre as ações que o robô pode tomar e as situações em que cada ação é tomada. Sua definição é análoga a definição de função de um jogo de futebol. Por exemplo, semelhante a um goleiro de um time de futebol que, ao perceber que a bola está se aproximando de seu gol, deve ir em direção à bola com objetivo de desviar sua trajetória, um robô programado com o comportamento de goleiro deve executar ação semelhante em campo. O software do RobôCIn apresenta os seguintes comportamentos atribuídos para cada um dos três robôs em campo a cada momento do jogo: um será Atacante, outro Defensor e outro Goleiro.

A camada de escolha de formação tem como função analisar as posições dos robôs e da bola no campo e assim decidir qual comportamento é o mais adequado para cada robô executar. É importante observar que a escolha de formação é executada *frame* a

frame, ou seja, um robô que começou o jogo sendo goleiro pode terminar sendo atacante, e vice-versa. Tal característica possibilita uma maior adaptabilidade ao time de robôs. Essa adaptabilidade pode ser observada na Figura 5. Nessa situação, o robô atacante, marcado pelo identificador 1, não está em boa posição para chegar na bola pois está sendo marcado pelo robô inimigo de identificador 2. Já o robô defensor, marcado com o identificador 3, tem um caminho quase retilíneo em direção ao gol. Nessa situação, a troca de funções faz com que este defensor tome o comportamento de atacante e aproveite da chance para fazer o gol. Para decidir que robô é o melhor para cada posição, são utilizadas as distâncias dos robôs para a bola e para o gol aliado, priorizando a escolha do goleiro em detrimento das outras.

Figura 5 – Situação de jogo onde a troca de funções entre os jogadores é vantajosa



Fonte: (ROBôCIN, 2018b).

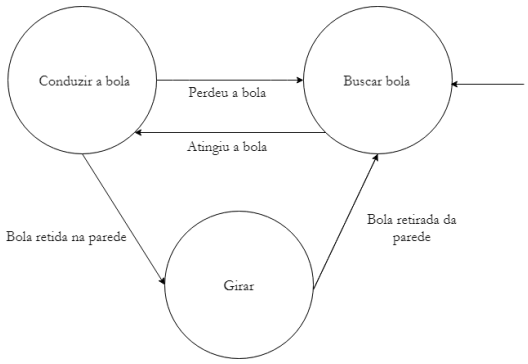
2.1.1.2 Escolha de Ação

A camada de escolha de ação, como é sugerido pelo nome, diz respeito à avaliação da melhor ação a ser tomada, dado o comportamento de cada robô. A escolha de ações é feita através de uma máquina de estados, onde os estados são ações e as arestas são situações de jogo. O exemplo de uma máquina de estados para um comportamento de atacante pode ser observada na Figura 6. Atualmente no módulo de estratégia em questão, as seguintes ações principais podem ser tomadas: buscar a bola, conduzir a bola, Girar e acompanhar a bola. Cada um dessas ações é subdividas em várias formas de fazê-las dependendo do estado do jogo.

2.1.1.3 Planejamento de Caminho

A escolha da ação a ser executada implica na definição de uma coordenada e orientação definidas como objetivo, que deve ser alcançado o mais rápido possível para que a ação tenha sucesso. É trabalho da camada de planejamento de caminhos definir qual a melhor trajetória a ser seguida de forma a chegar no objetivo evitando colisões. Para o planejamento da rota é utilizado o algoritmo de campos potenciais, em que o valor do potencial em cada ponto é definido pela Equação 2.1. Nesta abordagem, o robô é abstraído para um ponto que sofre a influência de um campo potencial gerado pelos outros robôs, pela

Figura 6 – Máquina de estados que representa o comportamento do atacante



Fonte: (ROBôCIN, 2018b).

bola, pelo seu objetivo final e pelos limites do campo. Os pontos com maiores potenciais são considerados pontos repulsivos, enquanto os pontos com os menores valores são considerados atrativos. Para definir o alcance do campo, de cada obstáculo e do objetivo, são geradas gaussianas (positivas ou negativas) que utilizam a equação 2.1, onde os parâmetros σ_x e σ_y definem o alcance da Gaussiana, A é a amplitude máxima ou mínima da Gaussiana, x_o e y_o são as coordenadas do obstáculo ou do objetivo, e x e y são as coordenadas do ponto que está sendo analisado.

$$f(x,y) = A \times exp(-(\frac{(x-x_o)^2}{2\sigma_x^2} + \frac{(y-y_o)^2}{2\sigma_y^2})) \tag{2.1}$$

Para a estimação da trajetória em cima da superfície criada pelos campos potenciais é utilizado o algoritmo A*. A utilização do A* é interessante pois é fácil mudar as características da trajetória desejada através da mudança de parâmetros da heurística de busca. Além disso, o algoritmo A* é robusto à mínimos locais presentes no campo, isto é, posições que apresentam valores menores que sua vizinhança, porém não representam o menor potencial do campo, que ocorre no o local onde está bola.

O resultado do processamento do módulo de planejamento de caminho será um conjunto ordenado de pontos no plano que descrevem a trajetória a ser executada. A partir dessa trajetória, é gerado um objetivo e angulação instantâneos, que são pontos do caminho alcançáveis num intervalo de *frame*.

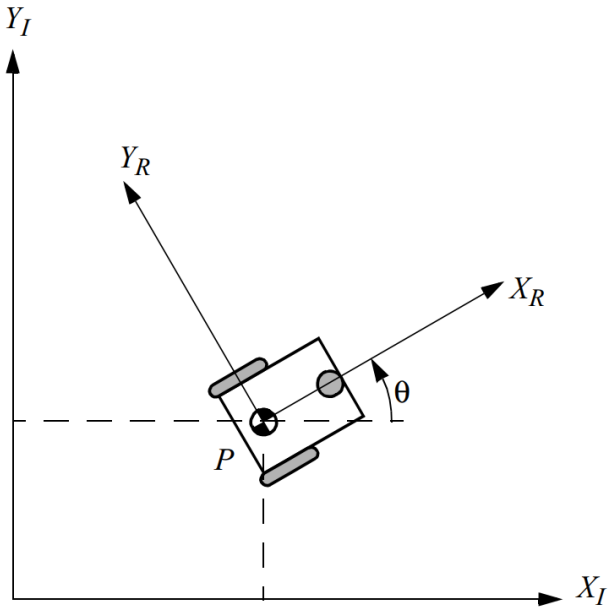
2.2 CINEMÁTICA E CONTROLE DO ROBÔ DIFERENCIAL

Para a categoria IEEE-VSSS são utilizados robôs móveis de direção diferencial. Robô de Direção Diferencial (RDD) são robôs móveis cujos movimentos são baseados em duas rodas ativas posicionadas em lados opostos do corpo do robô. Estes robôs controlam seus movimentos através da diferença de rotação entre as rodas. Para se obter um controle preciso dos RDD suas restrições cinemáticas precisam ser bem modeladas e entendidas.

Cinemática é a ciência que trata a movimentação do objeto sem considerar as forças que o causam. As equações a seguir fornecem um modelo para representar a movimentação de um robô de direção diferencial em função da velocidade de rotação de suas rodas. Essas equações ignoram certos aspectos físicos como atrito, energia e inércia do sistema. SIEGWART(2011) apresenta o conjunto de equações que descrevem como varia a posição do robô em função da velocidade das suas rodas e vice-versa, as quais são apresentadas nas seções seguintes.

2.2.1 Representação da Posição do Robô

Figura 7 – Robô Diferencial e Quadros de Referêcia.



Fonte: (SIEGWART, 2011).

Para especificar a posição de um robô em um plano, estabelece-se uma relação entre o sistema de referência global do plano I e o sistema de referência local do robô, como mostrado na Figura 7. Um ponto P é escolhido no robô como seu ponto de referência, geralmente esse ponto se localiza no centro do eixo que passa pelas duas rodas. O par X_R, Y_R define dois eixos relativos a P, estes eixos definem o sistema de coordenadas local do Robô. A posição de P no quadro de referência global é definida pelas suas coordenadas x e y, e pela diferença angular entre o quadro global e o quadro local dado por θ. A pose desse elemento é definida como um vetor dessas três características:

$$\xi_I = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \tag{2.2}$$

Para mapear entre o sistema de referência do robô e o global utiliza-se a matriz rotacional:

$$R(\theta) = \begin{bmatrix} cos\theta & sin\theta & 0 \\ -sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

Então:

$$\dot{\xi_R} = R(\theta)\dot{\xi_I}, \text{ e} \tag{2.4}$$

$$\dot{\xi_I} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} \tag{2.5}$$

O ponto em cima das variáveis indica a derivada delas em relação ao tempo.

2.2.2 Cinemática Direta

É interessante saber como o robô se movimenta, sendo que são conhecidas sua geometria e a velocidade das suas rodas. Um modelo matemático para cinemática direta deve ser capaz de prever a velocidade do robô no sistema de referência global em função destes parâmetros:

$$\dot{\xi_I} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = f(l, r, \theta, \phi_1, \phi_2) \tag{2.6}$$

$\dot{\xi_I}$ indica a velocidade instantânea no sistema de referência I, r significa o raio de cada roda, l é a distância entre cada roda e o ponto de referência da posição P, ϕ_1 é a velocidade angular da roda esquerda e ϕ_2 é a velocidade angular da roda direita.

A componente $\dot{x_R}$ pode ser obtida através da combinação das contribuições da velocidade de cada roda, $\dot{x_R} = \frac{r\dot{\phi_1}}{2} + \frac{r\dot{\phi_2}}{2}$. A componente $\dot{y_R}$ para cada instante terá valor zero, já que nenhuma das rodas tem capacidade de se locomover na direção lateral do robô. Para a última componente $\dot{\Theta}$ podemos novamente computar as contribuições de cada roda e somá-las, $\dot{\theta} = \frac{r\dot{\phi_1}}{2l} + \frac{-r\dot{\phi_2}}{2l}$. Assim, assumindo um mundo ideal sem ruídos, temos:

$$\dot{\xi_R} = \begin{bmatrix} \frac{r\dot{\phi_1}}{2} + \frac{r\dot{\phi_2}}{2} \\ 0 \\ \frac{r\dot{\phi_1}}{2l} + \frac{-r\dot{\phi_2}}{2l} \end{bmatrix} \tag{2.7}$$

Combinando as Equações 2.4 e 2.7:

$$\dot{\xi_I} = R(\theta)^{-1} \begin{bmatrix} \frac{r\dot{\phi_1}}{2} + \frac{r\dot{\phi_2}}{2} \\ 0 \\ \frac{r\dot{\phi_1}}{2l} + \frac{-r\dot{\phi_2}}{2l} \end{bmatrix} \tag{2.8}$$

2.2.3 Cinemática Inversa

Enquanto cinemática direta é útil para saber em que posição o robô estará, dado o comando nos seus atuadores, em cinemática inversa o objetivo é exatamente o contrário. Cinemática inversa interessa-se em saber quais os comandos necessários para que os atuadores levem o robô para determinada posição. O modelo de cinemática inversa é muito utilizado para gerar um controle para sistemas robóticos.

2.2.4 Controle de Movimentação

O problema de modelar o controle para robôs de direção diferencial pode ser resumido a encontrar a matriz K tal que o controle de $v(t)$ e $w(t)$, velocidades lineares e angulares, respectivamente, sejam feitos da seguinte forma:

$$\begin{bmatrix} v(t) \\ w(t) \end{bmatrix} = K \cdot e \tag{2.9}$$

De forma que o erro e convirja para zero no tempo infinito.

$$lim_{t \to \infty} e(t) = 0.$$

Uma possível solução em coordenadas polares para esse sistema pode ser obtida através da equação:

$$v(t) = k_\rho \rho \tag{2.10}$$

$$w(t) = k_\alpha \alpha + k_\beta \beta \tag{2.11}$$

$$k_\rho > 0; k_\beta < 0; k_\alpha - k_\beta > 0, \tag{2.12}$$

Em que k_ρ, k_α e k_β são parâmetros que configuram as características do movimento e α, β e ρ são definido por:

$$\rho = \sqrt{\Delta x^2 + \Delta y^2} \tag{2.13}$$

$$\alpha = -\theta atan2(\Delta y, \Delta x) \tag{2.14}$$

$$\beta = -\theta - \alpha \tag{2.15}$$

As velocidades angulares de cada roda individualmente podem ser derivadas das velocidades angular e linear do robô através das Equações 2.16 e 2.17.

$$\phi_1 = \frac{2v(t) + lw(t)}{\frac{r}{2}} \tag{2.16}$$

$$\phi_2 = \frac{2v(t) - lw(t)}{\frac{r}{2}} \tag{2.17}$$

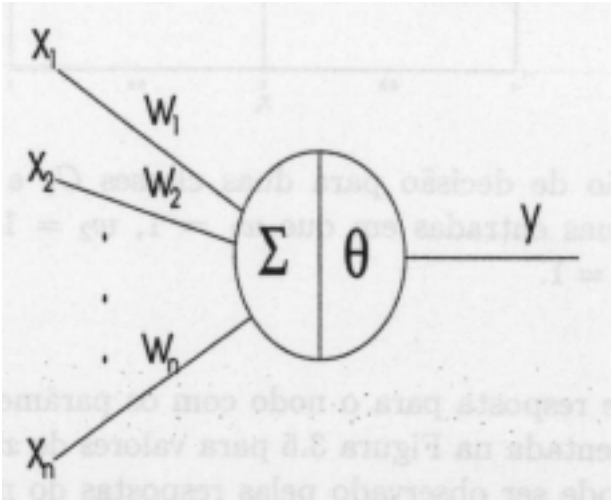
As provas e derivações desses conceitos fogem ao escopo desta Dissertação, em SIEGWART(2011).

2.3 APRENDIZAGEM PROFUNDA EM ROBÓTICA

As Redes Neurais Artificiais (RNA) são técnicas computacionais de aprendizagem de máquina inspiradas pelo funcionamento básico dos nerônios de animais. RNAs são sistemas paralelos compostos por nodos interconectados, esses nodos resolvem funções matemáticas, normalmente não-lineares. Estes nodos normalmente são organizados em camadas, cada camada e responsável por processar uma informação e repassar seu resultado na rede.

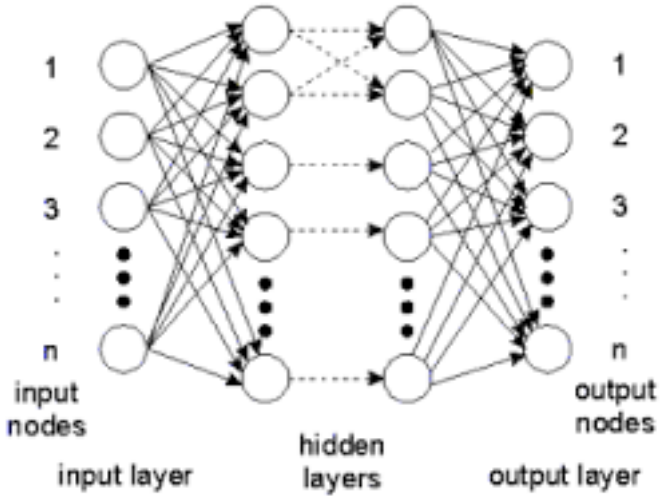
O primeiro modelo artificial de um neurônio biológico foi proposto por Warren McCulloch e Walter Pitts em 1943 (MCCULLOCH, 86). Esse trabalho se concentrou em descrever um modelo artificial de neurônio e apresentar suas capacidades computacionais. Em 1949, Donald Hebb publicou um trabalho que relacionava as redes neurais com a tarefa de aprendizagem (HEBB, 1949). Nesse trabalho ele propôs uma teoria para explicar o aprendizado em nodos biológicos baseado no reforço das ligações sinápticas entre nodos excitados. Apenas em 1958 Frank Rosenblatt (ROSENBLATT, 1958) propôs o modelo perceptron simples. O trabalho de Rosenblatt mostrou que, se acrescidas de sinapses ajustáveis, RNAs usando o modelo de McCulloch e Pitts poderiam ser treinadas para classificar alguns tipos de padrões. O perceptron simples, mostrado na Figura 8, recebe como entrada os valores que vão ser utilizados pelo nodo, as entradas se conectam ao nodo através de conexões que possuem coeficientes de transmissão (pesos) diferentes e ajustáveis. O neurônio fará a computação da soma das entradas ponderadas pelos seus respectivos pesos e aplicará uma função θ para gerar um sinal de saída. Por fim a saída, recebe e propaga o resultado das operações no nodo. Embora o modelo do perceptron não consiga resolver problemas não-linearmente separáveis, ele se tornou um importante modelo para o progresso das RNAs e pode ser considerado pai das redes Multilayer Perceptron (MLP).

Figura 8 – Modelo de Neurônio Artificial: Peceptron Simples.



Fonte: (BRAGA; FERREIRA; LUDERMIR, 2007).

Figura 9 – Modelo MLP.



Fonte: (BRAGA; FERREIRA; LUDERMIR, 2007).

O modelo das MLPs se caracterizam principalmente pela inserção de uma ou mais camadas intermediárias ou escondidas, vide figura 9, para solução de problemas não-linearmente separáveis. Com as técnicas tradicionais de RNAs apenas arquiteturas simples eram capazes de convergir eficientemente.

2.3.1 *Deep Neural Networks* (DNN)

Nas últimas décadas Redes Neurais Profundas (do inglês, Deep Neural Networks) (DNN) foram responsáveis por avanços significativos em inteligência computacional nas mais diversas áreas de pesquisa e aplicações. Esta nomenclatura engloba diversos tipos de técnicas utilizadas em RNAs para garantir uma melhor convergência delas quando a complexidade

da suas arquiteturas aumentam. As Redes Neurais Profundas (do inglês, Deep Neural Networks) (DNN)s, principalmente devido aos sucessos das camadas convolucionais se mostraram muito eficiente em tarefas com imagens (RUSSAKOVSKY et al., 2015; LONG; SHELHAMER; DARRELL, 2015; SZEGEDY et al., 2017; HANSON et al., 2018; ZHANG et al., 2018). Existem também muitos casos de sucesso das DNNs nas áreas de previsão de séries temporais (YANG et al., 2015; ZHENG et al., 2014; CHE et al., 2018) e robótica (KAHN et al., 2018; LEVINE et al., 2018). As DNNs se mostram particularmente especialistas em construir boas representações dos dados de entrada que proporcionam ao modelo obter melhores resultados em frente a uma determinada tarefa, antes do sucesso das DNNs o trabalho de engenharia de *features* era modelado manualmente por um humano especialista. As DNNs não só facilitaram este processo como conseguiram resultados muito superiores aos antes obtidos. Essas representações criadas são robustas e podem ser extrapoladas para diversas tarefas através das técnicas de *transfer learning*. O *transfer learning* é muito importante para a área de robótica (BOUSMALIS et al., 2018; JAMES et al., 2019) já que comumente os treinos precisam ser feitos em simuladores para serem aplicados em cenários reais.

As rede neurais artificiais profundas proporcionaram avanços nos paradigmas, supervisionado, reforço e não supervisionado. Os avanços em aprendizagem por reforço são particularmente importantes na área de robótica e aprendizagem de comportamentos. Quando se trata de aprendizagem de comportamentos é difícil se obter dados rotulados, visto que podem existir inúmeros estados e que um mesmo estado pode possuir diversos comportamentos que são aceitáveis e igualmente ótimos a longo prazo. Recentemente, a área de aprendizagem por reforço ganhou muita importância em reder neurais artificiais, alcançando resultados sobre-humanos em jogos de Atari (MNIH et al., 2013). Desde então, muita atenção foi trazida para a área e muitos novos trabalhos foram sendo produzidos. O DeepMind, grupo de inteligência artificial, em seu trabalho denominado AlphaStar (VINYALS et al., 2019), foi capaz de derrotar por cinco jogos a zero um profissional em Starcraft 2, um jogo altamente dinâmico de estratégia em tempo real, em cada jogador controlando uma das raças deve traçar uma estratégia de como utilizar seus recursos para anular seu oponente e destruir o centro da base adversária. O OpenAI recentemente apresentou um trabalho em que aprendizagem por reforço é utilizada para aprender a jogar Dota 2, um jogo em tempo real de ação e estratégia em que dois times de cinco jogadores cada lutam entre si em uma arena para ganhar recursos e destruir o centro da base adversária, e conseguiu vencer um time de profissionais (OPENAI, 2019).

Apesar da aprendizagem por reforço parecer ser muito indicada para a área de robótica, por não precisar de exemplos supervisionados e por fazerem com o que o robô consiga aprender iterativamente com seu ambiente, o trabalho de KOBER; BAGNELL; PETERS(2013) mostra que a sua aplicação nesta área tem algumas complicações que não estão presentes nos jogos de computadores. Entre estes desafios que precisam ser lidados estão: alta dimensionalidade dos estados e ações que os robôs normalmente têm que

lidar, dificuldade de coletar experiências do mundo real devido a limitações e desgaste do hardware, custos computacionais e incertezas de ambientes simulados, dificuldades em estabelecer um sinal de reforço adequado, etc.

2.4 AMBIENTES SIMULADOS DE ROBÓTICA

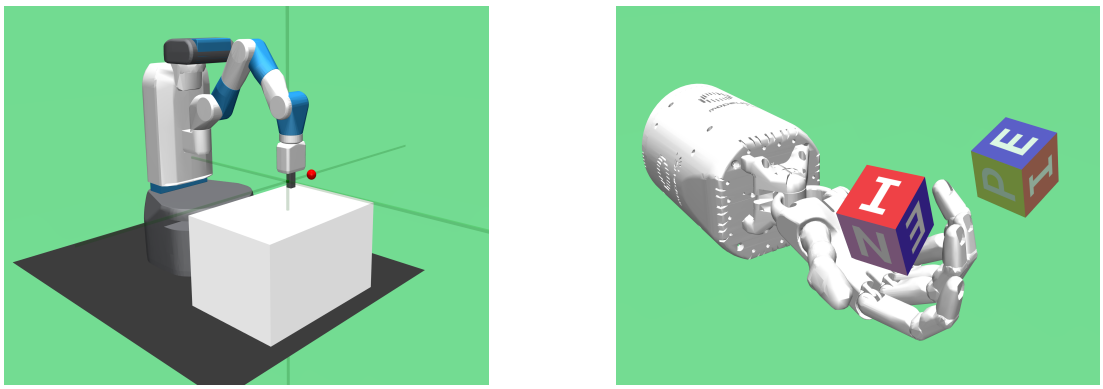
Nos últimos anos aprendizagem de máquina vem sendo cada vez mais utilizada em robótica. Os sucessos obtidos nas pesquisas em aprendizagem por reforço fez com que vários ambientes fossem sendo construídos para representar as mais diversas tarefas. Esses ambientes englobam jogos abstratos sem aplicações diretas no mundo real, simplificações de tarefas do mundo real e simulação do mundo real.

OpenAI é uma organização que visa criar ou auxiliar na criação de uma inteligência artificial de propósito geral (Inteligência Geral Artificial (do inglês Artificial General Intelligence) (AGI)). Para isso, ela tem colocado esforços na padronização da construção de inteligência. A OpenAI propôs um padrão adotado pela a comunidade de robótica para ambientes de simulação. Este padrão, introduzido pela biblioteca Gym, consiste de uma classe simples que deve implementar métodos específicos e padronizados que servem para treinar qualquer algoritmo de aprendizagem por reforço. Esta classe então pode ser acessada pelas interfaces implementadas pela biblioteca Gym, o que faz com que qualquer ambiente possa ser facilmente exportado para projetos que adotem o padrão.

Os ambientes de robótica envolvendo braços e manipuladores robóticos (PLAPPERT et al., 2018a), em Figura 10 disponibilizados pela organização OpenAI são utilizados como base de muitas pesquisas para aprendizagem, como: PLAPPERT et al.(2018b), SEHGAL et al.(2019), RAPISARDA(2019) e ANDRYCHOWICZ et al.(2018). Eles simulam o funcionamento de braços robóticos em algumas tarefas simples de manipulação e deslocamento de objetos. A dinâmica é feita para recriar a realidade da forma mais fiel possível. Porém, as tarefas definidas, apesar de muito importantes, por não serem dinâmicas e ruidosas suficientes, não são suficientemente gerais para representar todos os tipos de cenário que um robô possa encontrar na vida real e não envolve a cooperação entre agentes.

Apresentados recentemente, os ambientes Google Research Football (BRAIN, 2019) e o MuJoCo Soccer (LIU et al., 2019), são dois ambientes que surgiram no decorrer dos estudos deste trabalho. Esses dois ambiente, mostrados em Figura 11, apresentam o tema de futebol de robôs como *playground*. Esses ambientes são facilmente configuráveis e leves computacionalmente. Por isso, são rápidos nas interações com o ambiente e consequentemente acelerando o aprendizado dos agentes. O futebol é uma tarefa que inerentemente apresenta competição e cooperação, sendo um bom e desafiador cenário de estudos para aprendizagem de comportamentos por agentes. Porém, tanto o ambiente Google Research Football quanto o Deep Mind soccer não tem preocupação em manter fidelidade ao mundo real e, por isso, não estão preparados nem são adequados para que o conhecimento aprendido neles seja aplicado em robôs reais.

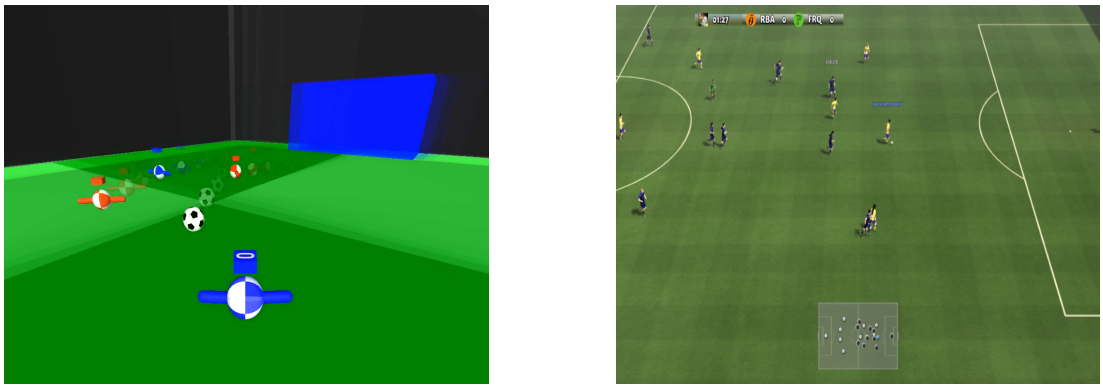
Figura 10 – Ambientes de robóticas do OpenAI. 10a: Ambiente *Fetch-Reach* utiliza braço robótico para aprender a mover objetos. 10b: Ambiente *Hand-Block* utiliza manipulador robótico para manejar objetos.



(a) Ambiente *Fetch-Reach* (b) Ambiente *Hand-Block*

Fonte: (OPENAI, 2018)

Figura 11 – Ambientes de Simulação de Futebol.



(a) Ambiente *MuJoCo Soccer* (b) Ambiente *Google Research Football*

Fonte: 11a: (LIU et al., 2019). 11b: (BRAIN, 2019)

Gazebo (V-REP, 2002) e V-Rep (V-REP, 2010) são dois simuladores de propósito geral que possuem uma simulação física que procuram ser fieis ao mundo real. Essas ferramentas são estáveis e possuem flexibilidade para modelar uma grande quantidade de ambientes robóticos simulados para os mais diversos problemas. Embora essas ferramentas sejam extremamente poderosas, por serem de propósito geral, elas falham em entregar alguns requisitos que são necessários para o treino rápido e eficiente de uma solução. A simulação nessas ferramentas é extremamente custosa computacionalmente, a comunicação com métodos externos são de difícil configuração, controlar o ambiente da simulação externamente é difícil. Por tudo isso elas se tornaram inviáveis para o uso no treinamento de comportamentos em robôs utilizando aprendizagem por reforço.

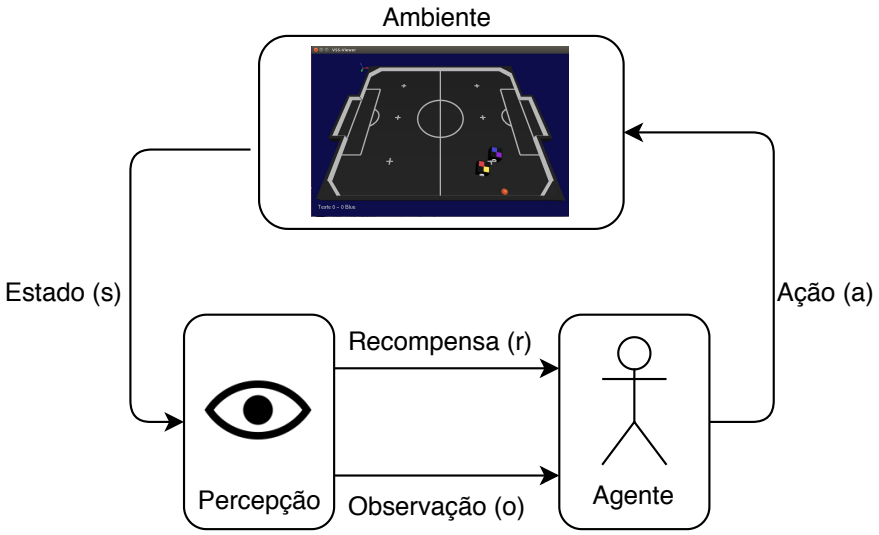
Apesar de todos esses agentes terem sido criados para satisfazer diversos nichos de pesquisas e interesses, nenhum deles apresenta todas as características que realmente re-

presentam um ambiente que pode ser utilizado para propósitos gerais na mundo real. Um ambiente como esse precisa ser fiel a realidade, ser dinâmico e cheio de obstáculos, imprevisível de certa maneira, permitir facilmente que o conhecimento aprendido no simulado seja transferido para o real, seja estável e capaz de executar por muito tempo sem interrupções, rápido e leve para ser executado de forma simples deixando o poder computacional para o o treinamento, seja cooperativo para o estudo dos comportamentos em sociedade, de preferência seja também competitivo para se assemelhar a vários cenários do mundo real de competição e facilmente configurável.

3 APRENDIZAGEM POR REFORÇO PARA COMPORTAMENTOS

Na Aprendizagem por Reforço (do inglês, Reinforcement Learning) (RL) o agente deve se adaptar a partir de recompensas que ele eventualmente receba conforme explora o ambiente. Este paradigma de aprendizagem se concentra em aprender como o agente deve agir em cada situação. A Figura 12 ilustra o ambiente ao qual o agente será submetido neste trabalho. Neste ambiente, o agente deve tomar ações com base nas suas observações e na forma como o ambiente muda e afeta o próprio agente. Enquanto a Aprendizagem por Reforço (do inglês, Reinforcement Learning) fornece o formalismo matemático necessário para escolher ações complexas a partir de estados, as DNN viabilizam o processamento de dados complexos sobre o ambiente.

Figura 12 – Ambiente Markoviano



Fonte: Autor (2019).

Atualmente a Aprendizagem por Reforço (do inglês, Reinforcement Learning) se mostra adequada para aprender políticas de ações ótimas governadas por regras simples e conhecidas, como em jogos de tabuleiro (SILVER et al., 2018); aprender habilidades simples a partir de dados de sensores(GU et al., 2017), como manipulação de objetos (RAPISARDA, 2019; ANDRYCHOWICZ et al., 2018); e aprender a partir da política demonstrada por um especialista(BOJARSKI et al., 2016). Porém, algoritmos de RL ainda não são tão bons quanto humanos em aprender e se adaptar em situações com poucos exemplos, aprender qual a recompensa do ambiente deve ser, aprender e entender comportamentos não apenas ações.

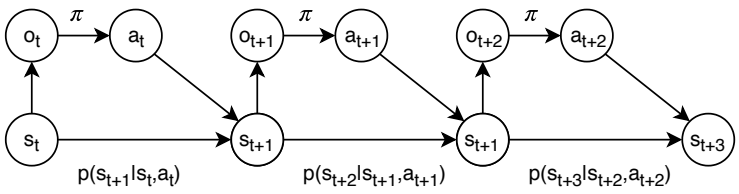
Este capítulo fornece uma base sólida sobre Markovian Decision Process (MDP) e descreve métodos de aprendizagem de comportamentos com ênfase em RL, de forma a subsidiar a compreensão das próximas etapas do trabalho. A primeira seção, Seção 3.1, do capítulo apresenta as definições do Processo de Decisão Markoviano e sua importân-

cia para a área de aprendizagem de comportamentos. Em seguida, Seção 3.2, introduz aprendizagem por imitação para aprendizagem de comportamentos e os problemas que fazem ela não ter sido aplicada neste trabalho. A Seção 3.3 faz uma descrição da área de Aprendizagem por Reforço, descrevendo as principais técnicas e algoritmos importante para a área e principalmente para esta dissertação.

3.1 PROCESSO DE DECISÃO MARKOVIANO

O ambiente de Aprendizagem por Reforço (do inglês, Reinforcement Learning) é tipicamente formulado como um Markovian Decision Process (MDP). O MDP provê um formulação matemática para modelar tomadas de decisões em ambientes estocásticos que são parcialmente controlados por um agente. Em um dado instante t, o processo a ser controlado deve estar em um estado, denominado s_t. O agente deve então escolher uma ação a_t a partir do estado atual ambiente, ação esta que influenciará na transição do ambiente para um novo estado s_{t+1}, retornando para o agente um sinal de reforço $r_t = R_{a_t}(s_t, s_{t+1})$. A probabilidade do processo transicionar de s_t para s_{t+1} é dependente da ação escolhida pelo agente e é definida pela função de transição de estados $P_a(s_t, s_{t+1})$. Então, o próximo estado deve ser dependente apenas do estado atual do sistema e da ação determinada pela política de escolha de ações π utilizada pelo agente no momento e independente de tudo que houver ocorrido antes disso. Este processo é ilustrado na Figura 13.

Figura 13 – Processo de Decisão Markoviano



Fonte: Autor (2019).

É importante remarcar que ao longo deste trabalho s_t pode significar, sem perda de generalidade, tanto o estado do processo como a observação do processo feita pelo agente. A observação de um agente sobre um processo poder ser igual ao estado do processo ou pode ser um subconjunto ou uma transformação das informações que definem o estado do processo.

Logo, os Markovian Decision Process são formalmente definidos pela tupla (S, A, P_a, R_a), onde:

- S é um conjunto de estados possíveis do processo,
- A é o conjunto de ações possíveis,

- $P_a = p(s_{t+1}|s_t, a_t)$ é a probabilidade da ação a_t no estado s_t levar o processo ao estado s_{t+1},
- R_a é a recompensa imediata recebida ao transicionar de s_t para s_{s+1} depois de aplicar a ação a_t.

O principal problema dos MDPs é encontrar uma política de ações π que especifica uma ação $a_t = \pi(s_t)$ a partir de uma estado qualquer s_t. O objetivo é encontrar uma política de ações que maximize alguma função das recompensas obtidas. Por exemplo:

$$\sum_{t=0}^{\infty} \lambda^t R_{a_t}(s_t, s_{t+1}), \tag{3.1}$$

onde λ é um fator de desconto que satisfaz a condição $0 \leq \lambda \leq 1$. O fator λ é utilizado para que o agente priorize ações que levam a recompensas mais próximas no tempo em detrimento ações que geram recompensas distantes no tempo, portanto mais incertas.

Uma política é dita ótima, e denotada por π^*, se ela escolhe a melhor ação para cada um dos estados possíveis da MDP. A política ótima pode ser obtida de de diversas maneiras, como: programação dinâmica, Aprendizagem por Reforço (do inglês, Reinforcement Learning) ou até mesmo imitando o comportamento de um especialista. Aprender um comportamento observando diretamente as ações de um especialista é conhecido como Aprendizagem por Imitação.

3.2 APRENDIZAGEM POR IMITAÇÃO

Por quê não aprender a partir de um especialista e construir a política a partir de suas ações? Esta parece ser uma boa solução mas nem sempre é factível e leva a outros problemas.

Aprendizagem por imitação conta com comportamentos classificados, e, a partir deles, aprende a política representada pelos dados como correta. Ela não depende diretamente do ambiente e por isso pode ser treinada de forma *offline*. Este tipo de aprendizagem possui a característica de levar a uma política que se comporta de maneira esperada, pelo menos para os casos vistos em treinamento, o que pode evitar imprevistos na execução do agente que causem sérios danos. Por exemplo, no trabalho (BOJARSKI et al., 2016) publicado pela companhia Nvidia , um carro autônomo aprende a se comportar em estradas através de aprendizagem por imitação, utilizando apenas as câmeras como sensores e as ações humanas no volante o carro.

Apesar da aprendizagem por imitação obter excelentes resultados em diversas tarefas, quando se trata de aprendizagem de comportamentos genéricos ela traz vários problemas que muitas vezes impedem sua simples aplicação. Primeiramente, por ser supervisionada, é necessário classificar cada ação como boa no conjunto de treinamento, o que é muito custoso para sua aplicação em robótica e muitas vezes estas classificações podem estar

erradas. Aprender com um especialista não significa que o agente se tornará um especialista. Não se pode garantir que um agente treinado para imitar o comportamento construa um entendimento real do mundo no lugar de apenas decorar certas ações em certas situações. Um agente que decore a política de um especialista será ruim em generalizar para situações que não foram encontradas nos exemplos de treino. Outro problema que surge é que, como todo treinamento tem imperfeições, a política imperfeita vai levar o agente a estados que podem não ter sido vistos no treino, gerando uma distribuição de dados diferentes da que o agente treinou. Esses erros tendem a se acumular com o tempo já que o agente não aprendeu a reagir a situações não vistas. Aprendizagem por imitação não permite ao agente aprender autonomamente, como observado em animais.

Uma forma intermediária de abordar o problema de aprendizagem de comportamentos que se situa entre a aprendizagem por imitação e aprendizagem por reforço, sem enviesar o comportamento do agente a decorar as ações do especialista consiste em aprender a função de reforço ideal a partir do comportamento de um especialista, está área de estudos se chama *Inverse Reinforcement Learning*. Ainda assim, este tipo de aprendizagem limita o agente ao comportamento do especialista que pode não ser ótimo em todas as situações.

Apesar destas técnicas serem úteis para muitos problemas, gerar dados supervisionados suficientes para um jogo tão dinâmico e complexo como futebol de robôs seria algo impraticável devido a enorme combinação de estados possíveis. Por isso, Aprendizagem por Reforço (do inglês, Reinforcement Learning) se torna uma alternativa mais viável para este tipo de problema de aprendizagem de comportamentos.

3.3 APRENDIZAGEM POR REFORÇO (RL)

Aprendizagem por Reforço (do inglês, Reinforcement Learning) (RL) tem o objetivo de aprender o melhor conjunto de parâmetros ψ^* de uma política de ações $\pi_\psi(a|s)$. O conjunto de parâmetros ψ^* deve ser escolhido de forma a maximizar a recompensa esperada ao longo de todo tempo, ou seja:

$$\psi^* = \arg\max_{\psi} E_\psi[\sum_t r(s_t, a_t)], \tag{3.2}$$

em que $r(s_t, a_t)$ é a recompensa imediata recebida pelo agente ao aplicar ação a_t no estado s_t.

A partir da ideia que Aprendizagem por Reforço (do inglês, Reinforcement Learning) otimiza a recompensa esperada do processo, duas funções importantes para auxiliar uma política a tomar decisões são: a Função Q (Q) e a Função Valor (V). A função Q é muito importante para vários algoritmos de RL. Ela estima a qualidade de se tomar uma ação a_t estando no estado s_t e é definida como a soma da recompensa esperada em cada passo de tempo, desde o tempo considerado na função até o fim do processo, aplicando-se a ação a_t e seguindo determinada política de escolha de ações π. Logo, ela pode ser escrita

como:

$$Q^{\pi}(s_t, a_t) = \sum_{t'=t}^{T} E_{\pi}[r(s_{t'}, a_{t'})|s_t, a_t]. \tag{3.3}$$

Se a verdadeira Função Q for sempre conhecida é fácil adaptar a política para priorizar ações que retornem a máxima recompensa. Assim como a função Q, a função V é também muito importante para vários algoritmos de aprendizagem. Ela estima o valor de se estar em um determinado estado e é definida como a recompensa total esperada de agente seguindo determinada política a partir de um estado s_t até o fim do processo. Ela pode ser escrita como:

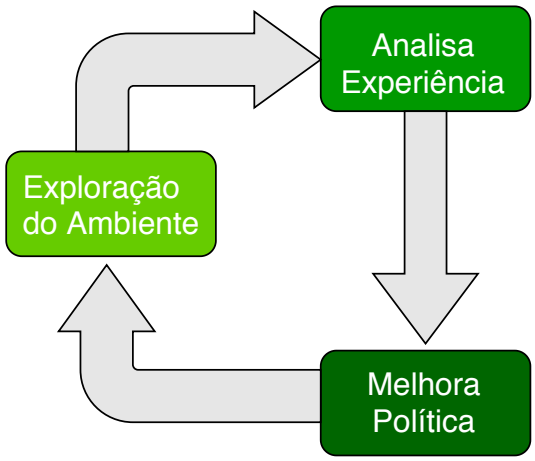
$$V^{\pi}(s_t) = \sum_{t'=t}^{T} E_{\pi}[r(s_{t'}, a_{t'})|s_t], \text{ logo,} \tag{3.4}$$

$$V^{\pi}(s_t) = E_{a_t \sim \pi}[Q^{\pi}(s_t, a_t)]. \tag{3.5}$$

Porém, supor que estas funções sejam conhecidas, ou mesmo fixas, não é algo plausível para a maioria dos algoritmos que otimizam suas políticas, uma vez que eles adaptam a política conforme descobre os valores Q ou V.

Embora nem todos os algoritmo de RL funcionem da mesma maneira, exite um fluxo básico seguidos por todos que consiste de três etapas principais indicadas na Figura 14, que podem ser mais ou menos complexas dependendo do algoritmo escolhido. Primeiro, o algoritmo deve gerar exemplos de interações com o mundo na etapa de Exploração do Ambiente, por exemplo, executando a política de ações no ambiente. Em seguida, o algoritmo analisa os exemplos coletados para aprender algo útil sobre eles. E no último passo, o conhecimento aprendido na etapa anterior é utilizado para melhorar a política de alguma forma, a depender do algoritmo e do ambiente estudado.

Figura 14 – Principais etapas do fluxo de RL.



Fonte: Autor (2019).

Cada etapa ilustrada na Figura 14 apresenta dificuldades diferentes dependendo do ambiente e algoritmo utilizado. Por exemplo, a exploração do ambiente pode ser mais

fácil ou mais difícil dependendo do ambiente em que o agente existe. Vários paradigmas de algoritmos de RL existem para lidar com cada problema de um modo diferente.

3.3.1 Tipos de Algoritmos de Aprendizagem por Reforço e Seus Problemas

O trabalho de SUTTON; BARTO(2018) divide os algoritmos de Aprendizagem por Reforço (do inglês, Reinforcement Learning) em: *Model Based*, *Value Based Policy Gradients* e *Actor-Critic*.

Algoritmos *Model Based* focam na iteração do agente com o ambiente para o aprendizado de um modelo acurado do ambiente. Este modelo é a tentativa do agente de criar uma representação fiel da MDP que rege o ambiente. Este modelo então deve, a partir do estado atual e de uma política de ações, ser capaz de informar as probabilidades da matriz de transição associada. A partir deste modelo, uma política pode ser definida ou até algo mais simples como algum algoritmo de busca para planejamento pode ser utilizado para estabelecer ações ótimas no problema. O problemas dos métodos *Model Based* normalmente se encontram na complexidade de gerar um modelo acurado do ambiente que seja capaz de generalizar bem. Em ambientes muito dinâmicos, muitas vezes encontrados em experimentos reais, a utilização deste tipo de abordagem se torna muito difícil e ineficiente. Pequenos erros no modelos podem gerar comportamentos indesejados (ARULKUMARAN et al., 2017).

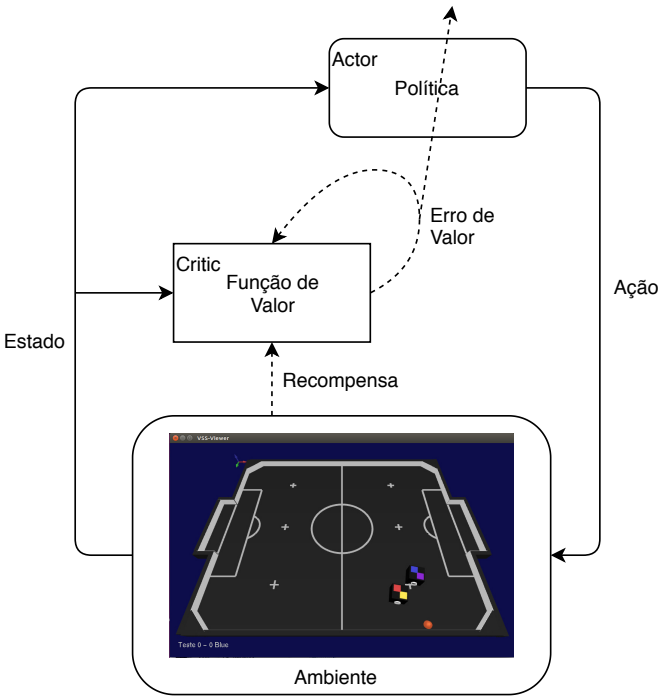
Algoritmos do tipo *Value Based* focam em aprender alguma função de valor dos estados, , como as funções Q ou V explicadas em 3.3. Através dessa função, uma política de ações pode ser otimizada. Se o algoritmo for capaz de convergir para uma função de valor ótima, então pode ser gerada uma política ótima a partir dela. Porém, possíveis erros na função de valor ou até a divergência dela fazem com que a política derivada possa apresentar problemas. Normalmente, esta família de algoritmos possuem muito parâmetros configuráveis e o resultado do algoritmo é muito sensível ao ajuste destes parâmetros (ARULKUMARAN et al., 2017).

Algoritmos do tipo *Policy Gradients*, ao contrário dos *Value Based*, trabalham diretamente na política de ações do agente. A política de ações é otimizada utilizando o gradiente da própria política com base nas experiências obtidas pelo agente executando tal política.

Algoritmos *Actor-Critic* tentam combinar o melhor das abordagens *Value Based* e *Policy Gradients*. Eles possuem dois módulos principais: o *Critic* aproxima alguma função de valor com base nas experiências do agente, enquanto que o *Actor* é responsável por executar uma política que será otimizada utilizando das informações fornecidas pelo *Critic*. A Figura 15 ilustra a arquitetura básica deste tipo de algoritmo.

Todos os algoritmos de Aprendizagem por Reforço (do inglês, Reinforcement Learning) buscam otimizar de alguma forma a Equação 3.2. Cada algoritmo tem suas particularidades e é especialista em resolver algum nicho de problemas de RL. Estes problemas

Figura 15 – Arquitetura de Algoritmos *Action-Critic*.



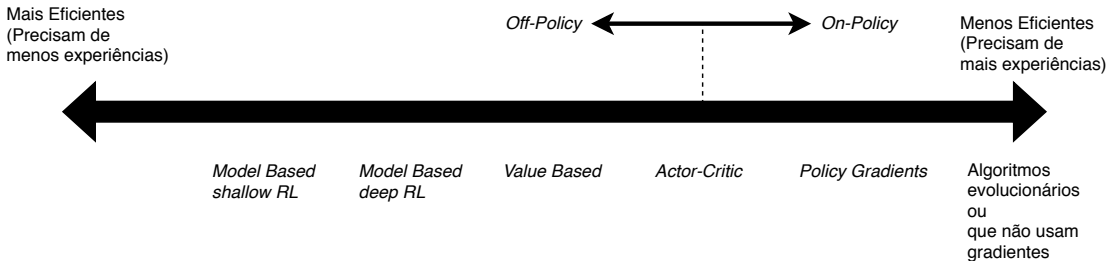
Fonte: Autor (2019).

envolvem: eficiência das experiências obtidas, estabilidade, se o domínio do problema é discreto ou contínuo, se modelo do mundo é fácil ou difícil de ser aprendido, etc.

Um dos principais problemas que afetam RL é a baixa eficiência no uso de experiências pelo agente, o que é muito importante especialmente para robótica. Se um agente pode rapidamente e sem restrições interagir sempre com o ambiente, esse problema não é muito importante, como um agente jogando um jogo simulado como xadrez. Já para robôs executando em mundo real, a quantidade de interações é limitada tanto pelo tempo real, quanto pelos limites do corpo do Robô. Deste problema se deriva uma das principais classificações em RL, na qual os algoritmos podem ser *On-Policy* ou *Off-Policy*. Algoritmos *Off-Policy* aprendem a partir de experiências que não necessariamente precisam ser geradas pela política atual. Esta característica faz com que as experiências passadas possam ser aproveitadas no treinamento da política atual, o que torna menos importante a necessidade de coletar novas experiências com a politica mais recente no ambiente. De forma contrária, algoritmos *On-Policy* atuam diretamente sobre a política de ações atual e aprendem diretamente a partir dela. Algoritmos *On-Policy* precisam gerar um conjunto de experiências novas para avaliar e otimizar a política atual. A Figura 16 mostra a relação dos tipos de métodos apresentados com a sua eficiência em usar as experiências obtidas.

Outro grande problema de Aprendizagem por Reforço (do inglês, Reinforcement Learning) é quanto à estabilidade. Ao contrário do que acontece em aprendizagem supervisionada, em RL a convergência não é garantida. Pelo contrário, muitos algoritmos populares

Figura 16 – Relação dos Tipo de Algoritmos de RL com sua Eficiência de Exemplos.



Fonte: Autor com base em (LEVINE; FINN, 2018)

de RL que aproximam alguma função de valor como Q ou V não possuem garantia de convergência. Por isso, muitas técnicas foram criadas para melhorar a convergência destes algoritmos na prática. Os algoritmos que procuram criar modelos de mundo possuem a garantia de sua convergência, devido ao fato deles utilizarem aprendizagem supervisionada para aprender o modelo de mundo. Porém, aprender modelo de mundo não é o mesmo que aprender políticas, pequenos erros no modelo aprendido podem levar a recompensas esperadas erradas. Algoritmos que aprendem a partir dos gradientes da própria política possuem garantias de convergir para algum ótimo local, já que aplicam gradiente descendente. Porém, na prática, eles são os mais ineficientes quanto a utilização das experiências, pois não conseguem reaproveitar experiências passadas obtidas(LEVINE; FINN, 2018).

Além desses, existem outros problemas que devem ser considerados para a escolha de um tipo de algoritmo, como: se o domínio é discreto ou contínuo, se a função de recompensa é esparsa ou não, se o problema é de um único agente ou são vários em cooperação, etc.

O ambiente estudado no trabalho é principalmente um ambiente simulado rápido e com um uma dinâmica complexa e muito variável. Métodos *Model Based* tendem a não funcionar bem neste caso, pois seria difícil aprender com eficácia um ambiente com uma dinâmica complexa e tão imprevisível quanto o ambiente estudado. Por exemplo, se o modelo fosse capaz de aproximar bem o ambiente para um determinado oponente, ou com qualquer outra pequena variação, como baixo nível de bateria do robô real, o modelo estaria defasado e passaria a retornar previsões incorretas. Apesar do treinamento acontecer primeiramente em simulador, o principal objetivo é que o conhecimento seja aplicado e adaptado para um cenário real de jogo. Como o simulador não apresenta uma física muito acurada em relação ao real para que ele seja computacionalmente mais eficiente, espera-se que um retreinamento em cenário real possa ser necessário para otimizar o comportamento dos agentes depois de aprendido em simulação, a fim de alcançar um desempenho ótimo. Por isso, é importante que o algoritmo utilizado no nosso estudo tenha boa eficiência com as experiências obtidas, o que torna os métodos *Policy Gradients* mais difíceis de serem executados. Para esta Dissertação, métodos *Value Based* ou *Actor-Critic* são alternativas que se encaixam melhor ao ambiente escolhido.

Apesar das ações do agente no ambiente escolhido serem naturalmente contínuas, elas também pode ser estudadas no domínio discreto através de métodos de discretização e isto será feito neste trabalho. Além disso, optou-se por estudar o problema na perspectiva de controle de um único agente. Os algoritmo escolhidos para estudo foram o Deep Q Networks (DQN) e Deep Deterministic Policy Gradients (DDPG), para o domínio discreto e contínuo, respectivamente. Além de serem algoritmos conhecidos e alvo de muitos trabalhos na literatura, ambos podem desfrutar de uma técnica chamada *replay* de memória, entre várias outras para garantir melhor convergência.

A técnica de *Replay* de memória, introduzida em (LIN, 1993), permite um aumento da eficiência de experiências dos algoritmos e permite com que experiências de cenários de jogos diferentes sejam misturadas e aprendidas em conjunto. A tupla $e_t = (s_t, a_t, r_t, s_{t+1})$ é armazenada num *buffer* de memória para cada iteração do agente no ambiente. As experiências contidas neste *buffer* são aleatoriamente escolhidas para treinar o algoritmo. Isso faz com que o treino e a execução da política sejam independentes podendo ser executadas em paralelo e melhora a estabilidade da optimização, já que os exemplos escolhidos aleatoriamente na etapa de testes são independentes e não enviesam a política rapidamente para um ótimo local causando *overfiting*.

Estas e outras técnicas são utilizadas tanto no DQN quanto no DDPG para melhorar sua convergência e gerar comportamentos mais eficientes do agente.

Alguns outros algoritmos muito importantes para a área de robótica como *Proximal Policy Optimization* ou *Trust Region Policy Optimization* são estados da arte em diversos problemas. Estes algoritmos são muito promissores para o problema abordado nesta Dissertação. Por seguirem o paradigma *On-Policy* e naturalmente precisarem mais experiências para convergir (LEVINE; FINN, 2018), esses algoritmos foram adiados para um segundo momento em que o ambiente estará mais estável em relação aos comportamentos obtidos e com os agentes convergindo mais rapidamente.

3.3.2 *Deep Q Networks*

O algoritmo *Q-learning* ganhou muita atenção na última década devido a seus grandes sucessos quando combinado com DNNs. A junção dos dois ganhou o nome de Deep Q Networks (DQN). As DQNs foram alvo de muitos estudos e obtiveram grande sucesso em jogos de computadores com ações discretas (MNIH et al., 2013). É uma técnica *Off-Policy* que possui boa eficiências no aproveitamento dos exemplos obtidos por iteração. Várias técnicas surgiram a partir do algoritmo básico para garantir melhor convergência em diversos tipos de problemas (HASSELT; GUEZ; SILVER, 2016b; WANG et al., 2015). Entre as mais importantes para este trabalho estão *Replay* de Memória e Double Deep Q Networks.

Q-learning é um algoritmo simples de se implementar. Sua ideia fundamental é utilizar a função Q_ψ com parâmetros ψ para aproximar a função Q (Equação 3.3). A partir da função Q, uma política pode ser aprendida. Esta política estará sempre limitada a quão

bom é a aproximação atual da função Q. Um aproximação perfeita levará a uma política ótima. As Equações 3.6 e 3.7 resumem o funcionamento geral do algoritmo Q-learning.

$$y_i \leftarrow r(s_i, a_i) + \gamma E[V_\psi(s'_i)], \text{sendo } E[V_\psi(s'_i)] \approx \max_{a'} Q_\psi(s'_i, a'_i) \tag{3.6}$$

$$\psi \leftarrow \arg\min_{\psi} \frac{1}{2} \sum_i \|Q_\psi(s_i, a_i) - y_i\|^2 \tag{3.7}$$

A recompensa esperada y_i é primeiramente estimada com base na recompensa obtida em algum passo do agente no ambiente no instante i somado à recompensa esperada máxima estimada pela função Q_ψ a partir do estado alcançado pelo agente no instante $i + 1$ (Equação 3.6). Os parâmetros ψ da função Q_ψ são então otimizados para que ela se assemelhe melhor à experiência obtida pelo agente no instante i (Equação 3.7). Os parâmetros serão atualizados com base na diferença entre função Q_ψ no tempo t e a Equação 3.6 para otimizar a Equação de Bellman (minimizando a Equação 3.8). Se o erro for zero, na Equação 3.8, então $Q_\psi(s, a) = r(s, a) + \gamma \max_{a'} Q_\psi(s', a')$ é ótimo, dito Q^*, e pode ser usado para gerar uma política ótima ψ^* como na Equação 3.9.

$$erro = \frac{1}{2} E_{s,a}[(Q_\psi(s, a) - [r(s, a) + \gamma \max_{a'} Q_\psi(s', a')])^2] \tag{3.8}$$

$$\pi^*(a_t|s_t) = \begin{cases} 1, & \text{se } a_t = \arg\max_{a_t} Q^*_\psi(s_t, a_t) \\ 0, & \text{caso contrário.} \end{cases} \tag{3.9}$$

O algoritmo de treinamento do Q-learning, descrito no Algoritmo 1. Depois da inicialização o algoritmo deve coletar um conjunto D_N de experiências define alguns parâmetros importantes para o treinamento e a convergência do algoritmos. O parâmetro N define se o algoritmo será *online* (caso seja igual a um) ou *offline*. Em seguida, o algoritmo vai estimar a recompensa esperada e atualizar os parâmetros da Função Q a partir de K experiências previamente coletadas. O parâmetro K define quantos passos de aprendizagem serão feitos até que novos exemplos sejam coletados novamente. γ define quão instantaneamente guloso será o algoritmo (maior γ mais importantes as recompensas futuras serão consideradas).

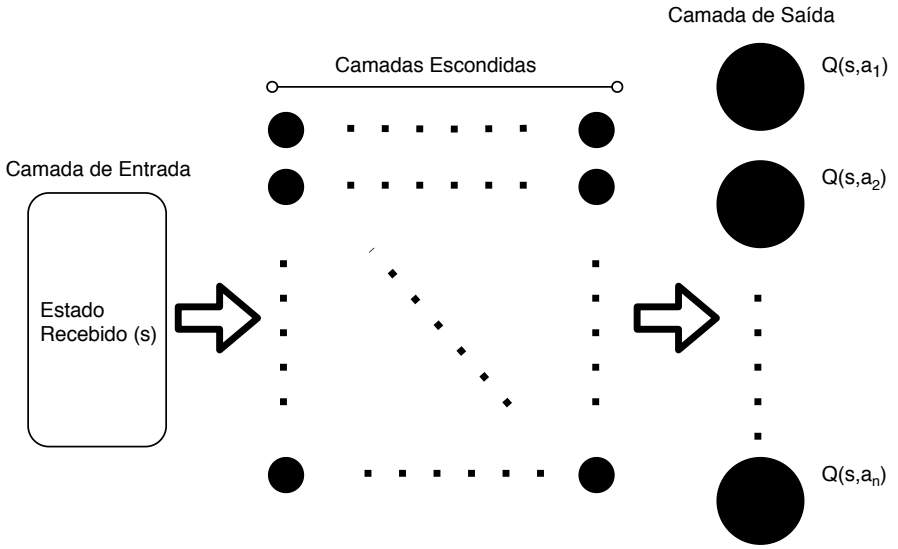
Deep Q Networks (DQN) nada mais é que utilizar o algoritmo de Q-Learning juntamente com DNN como função aproximadora da Função Q (Q). A rede neural da Figura 17 mostra uma rede genéria em que cada neurônio da camada de saída corresponde a uma das possíveis ações, fornecendo como saída o valor de Q para a ação a_i no estado s. O problema de utilizar redes neurais é que a sua convergência não é garantida, na verdade, isto adiciona muitas instabilidades ao processo. O que tornou DQN, e redes neurais em geral, tão eficientes para *Off-policy* RL foram as diversas novas técnicas de estabilização criadas para fazer estas abordagens funcionar bem juntas. Deep Q Networks foi introduzida por MNIH et al.(2013) mas ficou realmente em evidência após o artigo de MNIH et al.(2015).

Algorithm 1 Q-Learning

Defina os parâmetros:tamanho do conjunto de dados (N)
Defina número de iterações por conjunto de dados (K)
Defina parâmetro de desconto ($0 \leq \gamma \leq 1$)
Inicialize os parâmetros ψ da Função Q
enquanto não convergir **faça**
    Colete base de dados $D_N = (s_t, a_t, r_t, s_{t+1})$ seguindo uma política qualquer
    para i $\leftarrow$ 1 **até** K **faça**
        Colete um experiência (s_i, a_i, r_i, s_{i+1}) da base de dados
        $y_i \leftarrow r_i + \gamma \max_{a'} Q_\psi(s_{i+1}, a'_i)$
        $\psi \leftarrow \arg\min_\psi \frac{1}{2} \sum_i \|Q_\psi(s_i, a_i) - y_i\|^2$
    fim para
fim enquanto

O Algoritmo 2 mostra o funcionamento das DQNs apenas com a técnica de *replay* de memória e uma política de exploração ϵ-*greedy*. Na exploração ϵ-*greedy*, o agente tem uma probabilidade ϵ de realizar uma ação completamente aleatória para explorar o ambiente fora da política aprendida. Além do importante algoritmo de *Replay* de memória já apresentado, algumas técnicas importantes para garantir mais estabilidade a DQN são: usar uma rede auxiliar chamada de *target* e limitar as recompensas recebidas para intervalos adequados.

Figura 17 – Rede neural genérica para a aproximação da função Q.



Fonte: Autor (2019).

Uma rede separada chamada de *target* que tem a mesma estrutura da rede responsável por estimar o valor da função Q é utilizada para dar mais estabilidade ao treinamento de DQNs. A cada número de iterações fixos C a rede *target* é sobre-escrita com os pesos da rede neural responsável por estimar a função Q. Esta técnica é tão importante para DQNs que ganhou uma nomenclatura diferente Double Deep Q Netwok (DDQN). O algoritmo

Algorithm 2 *Deep Q-Learning* com *Replay* de Memória

Inicializar a memória D de tamanho N
Inicializar a rede neural Q para os valores de ações com parâmetros aleatório ψ
enquanto não convergir **faça**
    Inicialize o ambiente para o estado s_0
    para i $\leftarrow$ 1 **até** K **faça**
        Com probabilidade ϵ selecione uma ação aleatória a_t, caso contrário selecione $a_t = \max_a Q(s_i, a|\psi)$
        Execute ação a_t, compute recompensa r_t e receba novo estada s_{t+1}
        Armazene a transição (s_t, a_t, r_t, s_{t+1}) em D
        Colete *mini-batch* de transições de tamanho J da memória D
        para j $\leftarrow$ 1 **até** J **faça**

$$y_j = \begin{cases} r_j, & \text{se } s_{j+1} \text{ for terminal} \\ r_j + \gamma \max_{a'} Q(s_{j+1}, a'|\psi), & \text{caso contrário.} \end{cases}$$

            Compute o gradiente da função $(y_j - Q(s_j, a_j|\psi))^2$
        fim para
        Faça gradiente descendente com os gradientes computados para o *mini-batch*
    fim para
fim enquanto

DDQN funciona de maneira muito semelhante ao DQN original, com apenas uma mudança na computação dos gradientes, como mostrado em Algoritmo 3. Outra técnica importante para manter a estabilidade em DQNs é ajustar o limiar das recompensas evitando que os gradientes da rede se descontrolem e causem um *overfit* cedo no treinamento ou que sejam insignificantes e que o agente não consiga aprender. Depois de ter caído em ótimos locais é difícil para o algoritmo DQN achar uma função Q ótima pois as experiências recebidas pela política no ambiente podem se tornar muito semelhantes.

3.3.3 Deep Deterministic Policy Gradients (DDPG)

O algoritmo Deep Deterministic Policy Gradients (DDPG), proposto em (LILLICRAP et al., 2015), foi escolhido por motivos similares ao DQN. Ele pode operar de maneira *off-policy* e também é capaz de usufruir das técnicas de *Replay* de Memória e de redes *target* para garantir melhor convergência. A sua principal diferença para este trabalho comparado com DQN é que ele mais indicado para o domínio de ações contínuo por seguir o paradigma *Actor-Critic* e lidar diretamente com a política. Ou seja, valores contínuos são fornecidos como saída para o controle do agente.

Como já foi dito para algoritmos do tipo *Actor-Critic*, e que não é diferente para DDPG, é que eles aprendem tanto uma função de valor quanto um função da política. O DDPG aprende a função Q de forma muito similar ao DQN e uma função de política π é atualizada a partir destes. O DDPG utiliza também duas redes neurais adicionais para funcionar como *target*, a rede Q' para a função Q e a rede π' para a política π.

Algorithm 3 *Double Deep Q-Learning* com *Replay* de Memória

Inicializar a memória D de tamanho N
Inicializar a rede neural Q para os valores de ações com parâmetros aleatório ψ
Inicializar a rede neural *target* Q' para os valores de ações com parâmetros $\psi' \leftarrow \psi$
enquanto não convergir **faça**
    Inicialize o ambiente para o estado s_0
    para i $\leftarrow$ 1 **até** K **faça**
        Com probabilidade ϵ selecione uma ação aleatória a_t, caso contrário selecione $a_t = \max_a Q(s_i, a|\psi)$
        Execute ação a_t, compute recompensa r_t e receba novo estado s_{t+1}
        Armazene a transição (s_t, a_t, r_t, s_{t+1}) em D
        Colete *mini-batch* de transições de tamanho J da memória D
        para j $\leftarrow$ 1 **até** J **faça**

$$y_j = \begin{cases} r_j, & \text{se } s_{j+1} \text{ for terminal} \\ r_j + \gamma \max_{a'} Q'(s_{j+1}, a'|\psi'), & \text{caso contrário.} \end{cases}$$

            Compute o gradiente da função $(y_j - Q(s_j, a_j|\psi))^2$
        fim para
        Faça gradiente descendente com os gradientes computados para o *mini-batch*
        A cada C iterações faça $\psi' = \psi$
    fim para
fim enquanto

Para o aprendizado da função Q, de forma semelhante ao que ocorre em DQN, o valor de referência para a recompensa esperada é obtido conforme a Equação 3.10 e a otimização da rede é feita minimizando o Mean Squared Error (MSE) descrito na Equação 3.11.

$$y_i = r_i + \gamma Q'(s_{i+1}, \pi'(s_{i+1}|\psi^{\pi'})|\psi_{Q'}) \tag{3.10}$$

$$MSE = \frac{1}{N}\sum_{i}^{N}(y_i - Q(s_i, a_i|\psi^Q))^2 \tag{3.11}$$

Já a aprendizagem da política ocorre para tentar maximizar a recompensa esperada dada a função Q como parâmetro. A função objetivo $J(\psi)$ que se busca otimizar é definida na Equação 3.12. O erro da política é então calculado baseando-se no derivativo da função $J(\psi)$ com respeito aos parâmetros ψ^π (Equação 3.13). Como o DDPG atualiza a política de maneira *off-policy* em *mini-batches* coletados da memória, o gradiente será dado pela média da soma dos gradientes calculados para o *mini-batch*, Equação 3.14, e a otimização da política seguirá o gradiente ascendente deste valor.

$$J(\psi) = E(Q(s, a)|_{s=s_t, a=\pi(s_t)}) \tag{3.12}$$

$$\nabla_{Q^\pi} J(\psi) \approx \nabla_a Q(s, a|\psi^Q) \nabla_{\psi^\pi} \pi(s|\psi^\pi) \tag{3.13}$$

$$\nabla_{Q^\pi} J(\psi) \approx \frac{1}{N} \sum_{i}^{N} \nabla_a Q(s_i, \pi(s_i)|\psi^Q) \nabla_{\psi^\pi} \pi(s_i|\psi^\pi) \tag{3.14}$$

A atualização das redes *target* funciona de maneira um pouco diferente do que acontece em DQN. No DQN, as redes *target* são atualizadas de maneira brusca a cada C iterações, enquanto para DDPG, com o objetivo de melhorar a estabilidade, as redes *targets* são atualizadas a cada iteração por uma proporção $\tau \ll 1$ dos parâmetros das redes principais, como indicado nas Equações 3.15 e 3.16.

$$\psi^{Q'} \leftarrow \tau\psi^Q + (1-\tau)\psi^{Q'} \tag{3.15}$$

$$\psi^{\pi'} \leftarrow \tau\psi^\pi + (1-\tau)\psi^{\pi'} \tag{3.16}$$

Outra importante diferença é quanto a exploração do ambiente. Enquanto na DQN utiliza exploração ϵ-*greedy*, o algoritmo DDPG realiza exploração adicionando ruído às ações, logo a ação a_t passa a ser calculada como:

$$a_t = \pi(s_t) + \mathcal{N}, \tag{3.17}$$

onde $\mathcal{N}$ é o ruído adicionado.

Os autores do DDPG utilizam o *Ornstein-Uhlenbeck Process* (UHLENBECK; ORNSTEIN, 1930) para gerar um ruído que estará correlacionado a com ruídos anteriores, como indicado na (3.18) que define o processo de Ornstein-Uhlenbeck x_t com base no processo de Wiener W_t (3.19). Isto provoca uma continuidade no comportamento do agente, necessário para uma exploração mais adequada do ambiente, ao evitar que comandos consecutivos terminem se anulando. O Algoritmo 4 mostra o pseudo-código para implementação do DDPG.

$$dx_t = -\sigma x_t dt + \gamma dW_t,\ \sigma > 0 \text{ e } \gamma > 0. \tag{3.18}$$

$$\begin{cases} W_0 = 0 \\ W_{t+u} - W_t\ \mathcal{N}(0, u) \end{cases} \tag{3.19}$$

Algorithm 4 *Deep Deterministic Gradients Policy* com *Replay* de Memória

Inicializar a memória D de tamanho N
Inicializar a rede neural Q para o *Critic* com parâmetros aleatórios ψ^Q e a rede neural π para o *Actor* com parâmetros aleatórios ψ^π
Inicializar as redes neurais *target* Q' e π' para os valores de ações com parâmetros $\psi^{Q'} \leftarrow \psi^Q$ e $\psi^{pi'} \leftarrow \psi^\pi$
enquanto não convergir **faça**
    Inicialize o ambiente para o estado s_0
    Inicialize um processo aleatório $\mathcal{N}$ para a exploração
    para i $\leftarrow$ 1 **até** K **faça**
        Selecione uma ação $a_t = \pi(s_i|\psi^\pi) + \mathcal{N}$ de acordo com a política e ruido de exploração atual
        Execute ação a_t, compute recompensa r_t e receba novo estado s_{t+1}
        Armazene a transição (s_t, a_t, r_t, s_{t+1}) em D
        Colete *mini-batch* de transições de tamanho J da memória D
        para j $\leftarrow$ 1 **até** J **faça**

$$y_j = \begin{cases} r_j, & \text{se } s_{j+1} \text{ for terminal} \\ r_j + \gamma Q'(s_{j+1}, \pi'(s_{j+1|\psi^{\pi'}})|\psi^{Q'}), & \text{caso contrário.} \end{cases}$$

            Compute o gradiente da função $(y_j - Q(s_j, a_j|\psi^Q))^2$
        fim para
        Atualize o *Critic* usando gradiente descendente com os gradientes computados para o *mini-batch*
        Atualize a rede *Actor* utilizando o gradiente da política:

$$\nabla_{Q^\pi} J(\psi) \approx \frac{1}{J} \sum_i^J \nabla_a Q(s_i, \pi(s_i|\psi^\pi)|\psi^Q) \nabla_{\psi^\pi} \pi(s_i|\psi^\pi)$$

        A cada iteração atualize os parâmetros das redes *target*:

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

$$\psi^{\pi'} \leftarrow \tau\psi^\pi + (1-\tau)\psi^{\pi'}$$

    fim para
fim enquanto

4 METODOLOGIA

Neste capítulo o leitor entenderá os caminhos tomados nesta dissertação para a aplicação dos conhecimentos anteriormente introduzidos neste trabalho. Serão apresentados os detalhes das implementações dos algoritmos, as ferramentas utilizadas e os hiper-parâmetros escolhidos para os testes, assim como o porquê de cada escolha. Primeiramente, na Seção 4.1, será introduzido o ambiente de simulação que foi utilizado como base e as mudanças que precisaram ser feitas nele para que se adequasse aos propósitos desta dissertação para aprendizagem por reforço. Em seguida, na Seção 4.3 serão definidos as recompensas dadas aos agentes. As formas de controle utilizadas em cada agente será discutido na Seção 4.4. Na Seção 4.5 será introduzido da arquitetura das redes neurais utilizadas pelos agentes Por último, na Seção 4.6, serão apresentadas as medições que foram consideradas como importantes para definir um bom comportamento do agente no ambiente IEEE-VSSS.

4.1 AMBIENTE DE SIMULAÇÃO - VSS-SDK

O ambiente de Simulação utilizado (SIRLAB, 2019) é um ambiente próprio para a competição IEEE Very Small Size Soccer (IEEE-VSSS). O VSS-SDK é um projeto *open source* feito em C++, utilizando o CMake para compilar o projeto. Este ambiente não foi feito com o objetivo de ser utilizado com aprendizagem por reforço, ele tem o propósito de ajudar equipes na construção das estratégias de seus times da categoria. Este projeto é divido nos seguintes módulos:

- VSS-Vision - Sistema de visão computacional para ser utilizado em conjunto com uma câmera ligada ao computador para computar as posições cartesianas de robôs reais.
- VSS-Samples - Exemplos de estratégias de robôs pré-prontas.
- VSS-Joystick - Sistema de controle manual dos robôs via controle ligado ao computador através das interfaces usb ou bluetooth
- VSS-Simulator - Simulador de partidas de futebol
- VSS-Viewer - Sistema de visualização do estado das partidas simuladas e também fornece informações para depuração das estratégias

Neste trabalho, dois módulos foram utilizados: o VSS-Viewer e o VSS-Simulator. Estes dois módulos funcionam de forma independente entre si e se comunicam através de *sockets*, o que permite que eles sejam executados até em computadores diferentes. Isto permite que a simulação fique mais eficiente, podendo esta ser executada em um *cluster*

de computadores enquanto a visualização do experimento pode rodar localmente. O módulo VSS-Simulator dá a capacidade, de maneira estável e rápida, que times da categoria consigam testar e ajustar suas estratégias. Neste trabalho, foram feitas modificações no simulador para que um agente possa treinar sobre ele e aprender autonomamente novas estratégias. Este módulo do projeto oferece uma simulação tri-dimensional das partidas de futebol feita utilizando a biblioteca Bullet Physics (SIRLAB, 2015) capaz de simular colisões de objetos e restrições dinâmicas e cinemáticas. Este módulo também implementa um árbitro para detectar gols e reposicionar objetos para fazer a simulação prosseguir sem intervenção humana. Neste módulo a aceleração da simulação é controlada por um parâmetro passado ao simulador. O VSS-Viewer foi outro módulo muito importante neste projeto para a avaliação visual das estratégias aprendidas e das suas evoluções. Este módulo apresenta uma imagem em tempo real do que está acontecendo na simulação, como visto na Figura 18.

Figura 18 – Campo simulado visto através do VSS-Viewer.

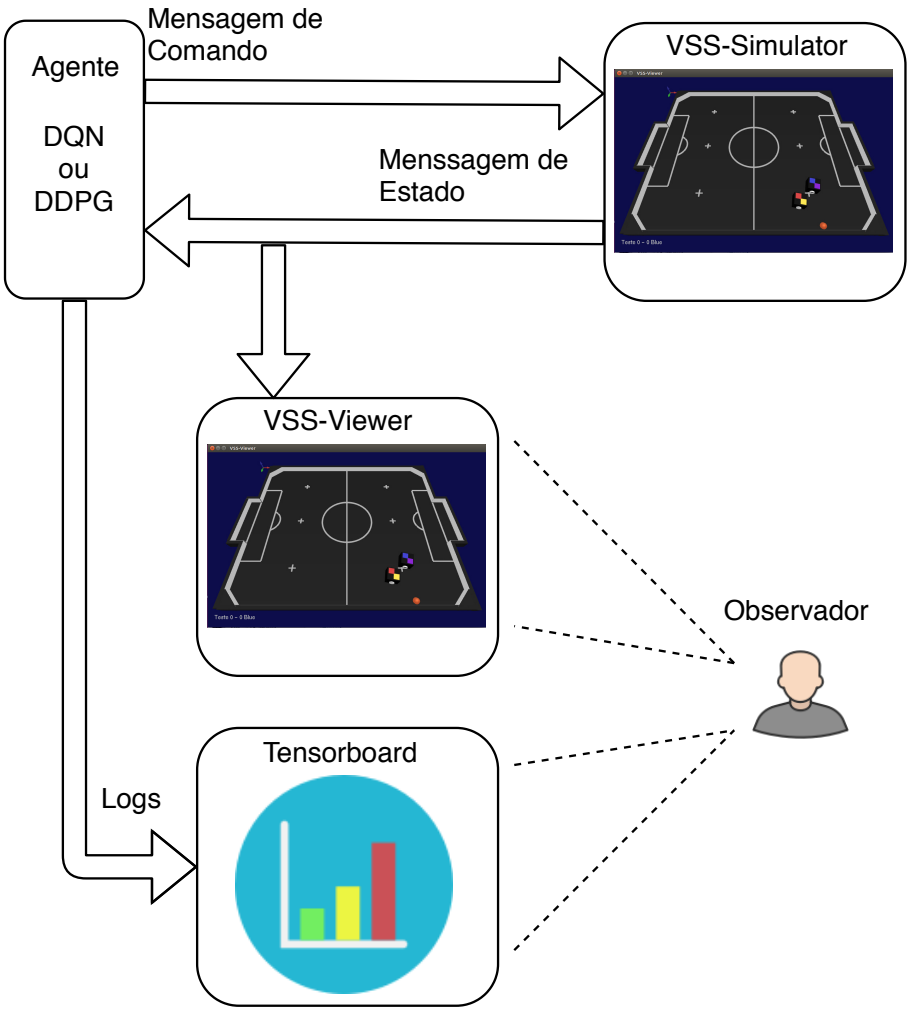
Fonte: Autor (2019).

A comunicação entre os módulos do VSS-SDK é feita através de sockets tcp e segue o protocolo de definição de mensagens Protobuf (DEVELOPERS, 2019). Protobuf é um método de serialização de dados estruturados em mensagens. O objetivo do Protobuf é trazer simplicidade e rapidez na comunicação. Ao utilizar este método o VSS-SDK permite que se construa módulos adjacentes em qualquer linguagem comunicando-se por socket e que vários de seus módulos possam ser instanciados em paralelo utilizando para cada um uma porta diferente do protocolo.

Para a comunicação entre os módulos do VSS-SDK são definidos três tipos de mensagem que podem ser utilizadas. A mensagem de estado (*State*) é utilizada para enviar informações de posição, angulação e velocidade dos robôs e da bola para os diferentes

módulos. A mensagem de comando (*Command*) deve conter as velocidades lineares desejadas da roda esquerda e direita de cada robô comandado pelo time. A mensagem de depuração (*Debug*) consiste em posições de objetos fantasmas que devem ser plotados no VSS-Viewer, estas posições podem ser utilizadas para indicar a posição futura desejada do robô ou analisar o atraso entre a percepção e ação, por exemplo. Neste trabalho, o agente de aprendizagem por reforço foi implementado em python e se comunicou com o VSS-SDK por sockets. A cada iteração do agente com o ambiente ele recebe uma mensagem de estado, calcula uma recompensa com relação ao estado e ação no passo anterior e envia uma mensagem de comando de volta ao ambiente para ser executado. O estado do ambiente e do agente podem ser acompanhados tanto pelos gráficos iterativos produzidos e disponibilizados através da ferramenta de visualização Tensorboard ou pelo estado da partida mostrado pelo VSS-Viewer. O fluxo dessa iteração está mostrado em Figura 19.

Figura 19 – Fluxo completo de um agente treinando em ambiente simulado.



Fonte: Autor (2019).

O funcionamento do simulador VSS-Simulator é originalmente assíncrono, ou seja, cada mensagem que chegar vai ser tratada enquanto o simulador continua executando o

ambiente e a simulação continua a acontecer mesmo que não chegue mensagem alguma. Esta assincronicidade, embora ocorra no mundo real, pode trazer muitos ruídos para o treinamento do agente quando a execução do simulador é acelerada, por isso, para esta dissertação o simulador foi modificado para funcionar de forma síncrona. Esta modificações pode ter afetado o comportamento no mundo real, mas não chegou ao ponto de ser um problema constatado nesse trabalho. O funcionamento síncrono do simulador significa que cada passo de simulação só vai ocorrer com a chegada de uma mensagem de comando vinda da estratégia. Por isso, o agente sempre vai perceber uma variação de estado de período igual. Esta mudança permite tanto eliminar ruídos de simulação quanto acelerar a simulação à máxima velocidade possível da máquina, sem perder eficiência do agente. Este ajuste adaptativo da velocidade de simulação, conforme a capacidade da máquina em treinar o agente e simular o ambiente, foi uma mudança necessária para se ter o treinamento mais rápido possível. Originalmente, o simulador era apenas capaz de definir velocidades fixas, o que poderia ser pouco ou muito rápido para o agente, a depender das configurações da máquina em que ele está executando, introduzindo ineficiências de recursos de computador e introduzindo ruídos de percepção ao agente, respectivamente.

Outra mudança no simulador que foi necessário nesse trabalho foi introduzir a possibilidade de inicialização aleatória das posições e orientações dos robôs e da bola ao início da simulação. Assim, as experiências do agente se tornam mais variadas, evitando um aprendizado de comportamentos enviesados a um mesmo estado inicial da simulação.

4.2 AMBIENTE DE APRENDIZAGEM

A Figura 20 mostra como foram organizados os módulos implementados e como eles se comunicam. No diagrama “Processo de Experimentação” estão os módulos pertencentes ao processo de exploração dos agentes no meio ambiente. Cada agente independente executa uma *thread* para escolher através de sua política e da rede neural aprendida qual ação deve executar, enviá-la para a Classe *Wrapper* e esperar por uma resposta do ambiente da sua ação, armazenando-a numa fila para posteriormente ser passada ao *Buffer* de Memória do sistema. A classe *wrapper* é responsável por coletar os comandos de todos os agentes, empacotar estas ações para enviar para a ambiente de jogo que irá desempacotar e retornar o estado do jogo, conforme a experiência recebida por cada agente. O ambiente Gym (BROCKMAN et al., 2016) faz a interface do agente com o ambiente. Este módulo realiza toda a comunicação com o jogo, calculando as recompensas das ações dadas e retornando a experiência percebida no passo simulação executado para o sistema de aprendizagem. No diagrama “Processo de Treinamento” estão os módulos responsáveis pelo treinamento da política. O módulo de Treinamento é uma interface de controle para toda a execução. Primeiramente, ele controla a Fila de Experiências para esvaziá-la quando estiver na metade da sua capacidade máxima. Em seguida, coleta experiências do *Buffer* de Memória para avaliá-las no estado atual da política, em seguida calculando o erro associado à

experiência coletada para ajustar a rede neural. A Rede Neural *Target* será atualizada em períodos de tempo definidos no início da simulação. Esta arquitetura implementada funciona tanto para o algoritmo DDQN quanto para o DDPG e pode ser estendida para outros algoritmos.

O controle das experiências é definido no ambiente Gym. Este ambiente implementa métodos de comunicação para aplicar ações aos agentes, início e reinício de simulações, configurações dos parâmetros de simulação e calculo de recompensas. O agente utiliza esse ambiente para coletar experiências na simulação.

Ao iniciar, o agente deve instanciar um objeto do ambiente para se conectar à simulação. A primeira atribuição do ambiente Gym é inicializar o processo do VSS-Simulator e logo em seguida inicializar os *sockets* de comunicação com a simulação nas portas informadas como parâmetro para o agente. Quando o agente precisa reiniciar a simulação, o ambiente deve encerrar o processo do VSS-Simulator, fechar as portas de comunicação e reconfigurar uma nova simulação.

O ambiente de aprendizagem recebe o pacote de estado cru da simulação e executa um processamento no pacote, criando um vetor de estado adequado para uso pelo agente em seu aprendizado. O pacote fornece as informações de posição e velocidade de todos os objetos móveis da simulação (robôs e bola), além do tempo de simulação e se houve gol. Cada experiência será descrita em uma vetor de valores normalizados que serão passados para treinar o agente. O vetor de valores será composto por cinquenta e dois valores. Os primeiros quatro valores deste vetor representam as posições (x, y) e velocidades vetoriais (v_x, v_y) da bola, $b = x, y, v_x, v_y$. Em seguida o vetor terá 9 valores para cada robô do time controlado, estes valores são divididos em:

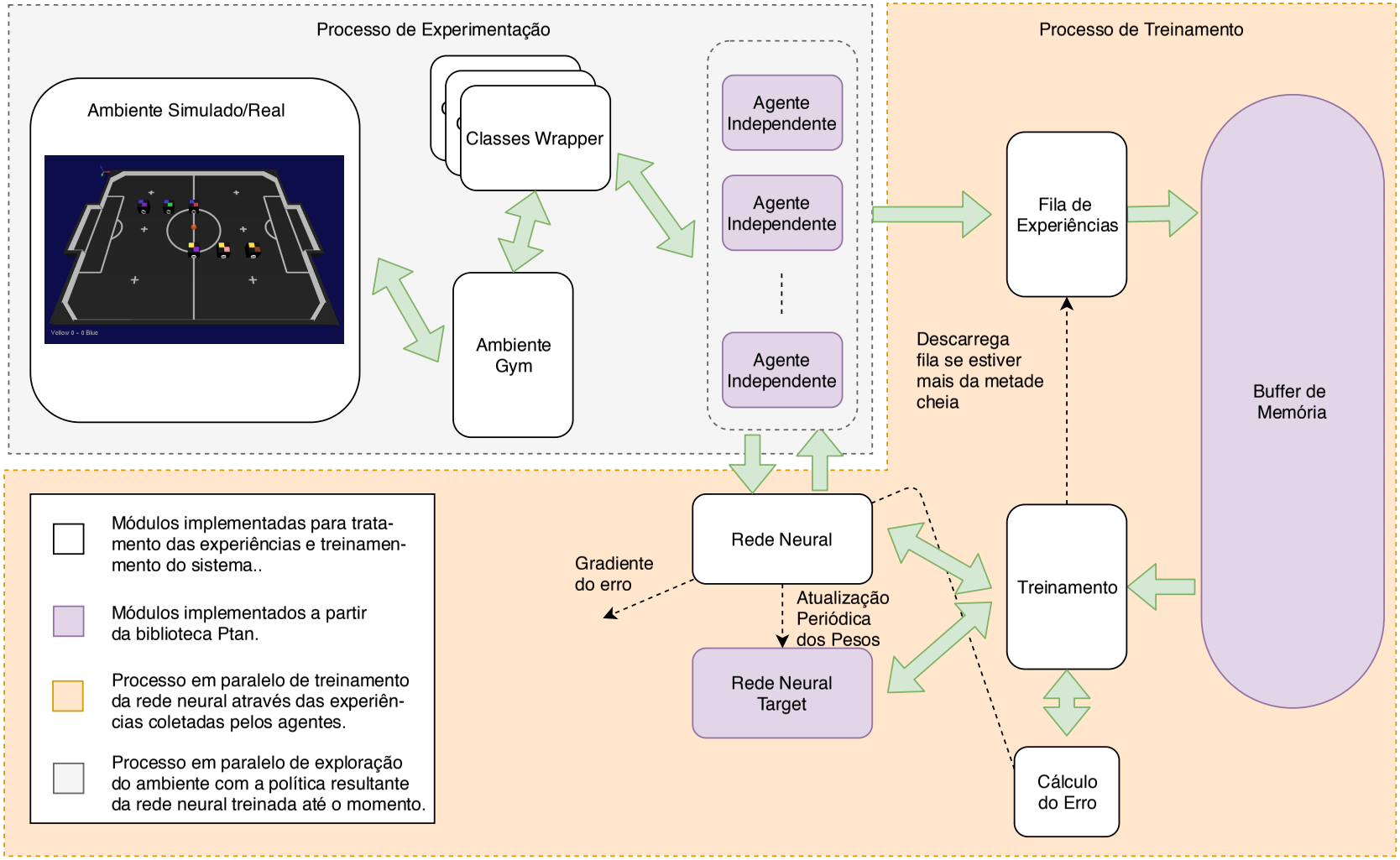
- 2 valores correspondentes à posição desejada do robô quando esta estiver disponível, caso contrário, a posição do atual robô (t_x, t_y),
- 4 valores para posição e orientação do robô $(x, y, \sin\theta, \cos\theta)$,
- 3 valores representando as velocidades linear e angular do robô (v_x, v_y, v_θ).

Os 21 valores restantes do vetor serão divididos entre cada robô da equipe adversária da seguinte forma:

- 4 valores para posição e orientação do robô $(x, y, \sin\theta, \cos\theta)$,
- 3 valores representando as velocidades linear e angular do robô (v_x, v_y, v_θ).

Os agentes contínuo e discreto possuem espaço de ações diferentes que serão explicados mais adiante neste capítulo. O fim de um jogo no nosso ambiente é definido para acontecer depois de cinco minutos em tempo de simulação, a partir do início do seu início.

Figura 20 – Diagrama de classes utilizadas e como elas se comunicam durante o treinamento de um agente.



Fonte: Autor (2019).

4.3 TRATAMENTO DE RECOMPENSAS ESPARSAS

Recompensas esparsas são a maneira mais simples de se modelar as recompensas em um ambiente. O agente recebe recompensas esporádicas, quando realmente atingem um estado objetivo. Em muitos cenários simples utilizar esta função mais simples funciona, como nos jogos de Atari (MNIH et al., 2013). Porém em ambiente complexos como o deste estudo recompensar o agente esporadicamente, apenas em estados de gols para o ambiente de testes desta dissertação, não funciona bem. Este tipo de recompensa é muito esparsa, o agente precisaria executar uma quantidade de passos muito grande até receber algum sinal de recompensa, tornando a tarefa de exploração uma tarefa bem árdua para o agente que não tem pistas sobre o valor de seu comportamento intermediário. O aprendizado através de sinais de reforço esparsos é um problema em aberto e alvo de estudos atualmente (RIEDMILLER et al., 2018; VEČERÍK et al., 2017), logo, comumente, os agentes precisam utilizar métodos específicos para aprender eficientemente e convergir para uma boa política. *Curriculum Learning* (HEESS et al., 2017; GHOSH et al., 2017), *Model Based RL* (MONTGOMERY; LEVINE, 2016), *inverse RL* (ZIEBART et al., 2008) ou *Reward Shapping* (GU et al., 2017; NG; HARADA; RUSSELL, 1999). Dentre estes, *Reward Shapping* e *Curriculum Learning* se encaixam bem no escopo desta dissertação. *Curriculum Learning* consiste em fazer o agente aprender pequenas habilidades por vez e avançar o seu aprendizado a partir das habilidades já adquiridas, porém sua utilização no escopo desta dissertação foi deixada para trabalhos futuros por requerer a definições de tarefas intermediárias específicas para o agente performar em sequência, o que seria custoso para o escopo deste trabalho. *Reward Shapping*, como foi explicado no Capítulo 3, é o processo de destrinchar a recompensa principal (o gol para nossos experimentos) em recompensas intermediárias que guiam o agente para a obtenção do objetivo principal.

O *Reward Shapping* feito neste trabalho consiste em adicionar dois valores de recompensas por comportamentos desejados específicos do agente além do gol em específico.

A primeira componente de recompensa é nomeada de Componente de Movimento, ela faz com o agente se aproxime da bola (m_R). Ela é definida como a variação da posição do agente em relação à posição da bola 4.1 no tempo t comparado ao tempo $t-1$.

$$m_R = \min(\max((d(agente, bola)_t - d(agente, bola)_{t-1}) * c_1, -1.0), 1.0), \tag{4.1}$$

onde, $d(a, b)$ é a distância euclidiana entre as posições do objeto a e do objeto b, c_1 é a constante que ajusta a escala de tempo, definida de forma a gerar valores entre 0.0 e 1.0. .

A outra componente é (gbp_R), nomeada Componente de Gradiente da Posição da Bola, é definida a partir do deslocamento da bola no campo. Esta componente vai ensinar ao agente a manter a interagir com a bola de maneira correta para que ela vá na direção do campo adversário. Esta componente é definida como o gradiente da variável bp_R 4.2, que

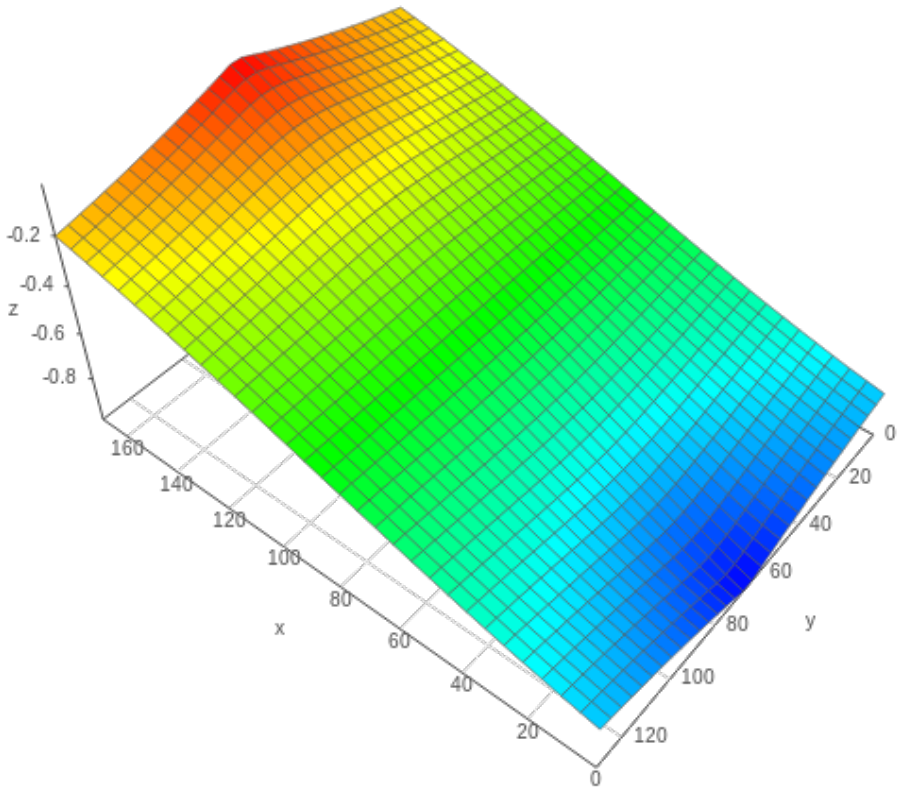
é uma função determinada pelo posicionamento da bola respectivo ao próprio gol e ao gol adversário, a superfície da função bp_r pode ser vista na Figura 21.

$$bp_R = \frac{\frac{-d(golAdversário,bola)+d(golPróprio,bola)}{170} - 1}{2}, \tag{4.2}$$

$$gbp_R = \min(\max((bp_{R_t} - bp_{R_{t-1}})\frac{c_2}{dt}, -1.0), 1.0) \tag{4.3}$$

onde, 170 é o comprimento do campo em centímetros, $d(a, b)$ é a distância euclidiana entre as posições do objeto a e do objeto b, c_2 é a constante que ajustam a escala de tempo, definidas de forma a gerar valores entre 0.0 e 1.0 e d_t é o intervalo de tempo entre as experiências nos passos t e $t - 1$.

Figura 21 – Campo potencial relacionado à posição da bola para um agente que atacará na direção de $x = 170$ para $x = 0$.



Fonte: Autor (2019).

A recompensa final dada para o agente R depois de sua experiência do tempo t será dada pela seguinte combinação linear das mini-tarefas:

$$R_t = \omega_m m_R + \omega_{gbp} gbp_R + \omega_g g_R \tag{4.4}$$

em que ω_m, ω_{bp} são os pesos atribuídos a cada mini-tarefa e ω_g é o peso atribuído a recompensa de gol g_R.

É importante notar que, segundo NG; HARADA; RUSSELL(1999), utilizar um tipo de função de *shaping* chamada *potetial-based shaping function* garante que toda política ótima definida no domínio de recompensas inicial (esparsa) também será ótima no novo domínio com *shaping*. Uma função F é nomeada *potencial-based* se existir uma função ϕ $S \rightarrow R$ tal que:

$$F(s_t, a_t, s_{t-1}) = \phi(s_t)\gamma - \phi(s_{t-1})$$

É fácil notar que cada componente de *shaping* proposto nesta seção segue esta formulação, logo a soma destas componentes também seguirá. Com isso, podemos afirmar que o *Reward Shaping* aplicado não terá influência na política ótima do problema.

Para o domínio contínuo há uma componente a mais na recompensa para evitar que o agente caia em ótimos locais de comportamentos pouco úteis. Esta componente penaliza o gasto de energia pelo agente, ela evita que o agente se mova de forma ineficiente, por exemplo: rotacionando sobre o próprio eixo. Esta rotação é um atalho que o agente encontrou em algumas situações para ter mais sucesso na sua tarefa, pois lhe permite chutar a bola para o gol apenas rotacionando no sentido adequado. Porém, repetir este comportamento continuamente não é um comportamento esperado de um agente inteligente por demorar mais que o necessário para chegar na bola. A componente de energia e_R é dada pela soma dos módulos das velocidades comandadas de cada uma das rodas como indicado na (4.5).

$$e_R = |v_l| + |v_r| \tag{4.5}$$

Com isso a recompensa final no domínio contínuo será adicionada da componente de energia multiplicada a um peso associado:

$$R_t = \omega_m m_R + \omega_g g_R + \omega_{gbp} gbp_R + \omega_e e_R \tag{4.6}$$

Isto não se mostrou necessário no domínio discreto, pois a forma como a discretização das ações foi realizada já reduz a possibilidade do agente desenvolver e manter altas velocidades de rotação como será visto na Seção 4.4.

4.4 CONTROLE DOS AGENTES

Cada agente é controlado de uma forma diferente devido a suas características únicas. O agente que executa o algoritmo DDQN possui um controle mais complexo pois precisou ser definido um espaço de ações discreto para o agente de forma que ele consiga ainda obter bons comportamentos em jogo. Este espaço discreto deve ser pequeno para melhorar a convergência do algoritmo. Para resolver este problema, este trabalho opta por trabalhar com o controle de movimentos descrito no Capítulo 2. Então o papel do agente será em controlar o posicionamento alvo do robô e o papel de levar o robô ao posicionamento será inteiramente do controle de movimentos. Uma analogia para entender melhor o funcionamento do controle é pensar que há uma mola invisível conectando o robô à "mão"do

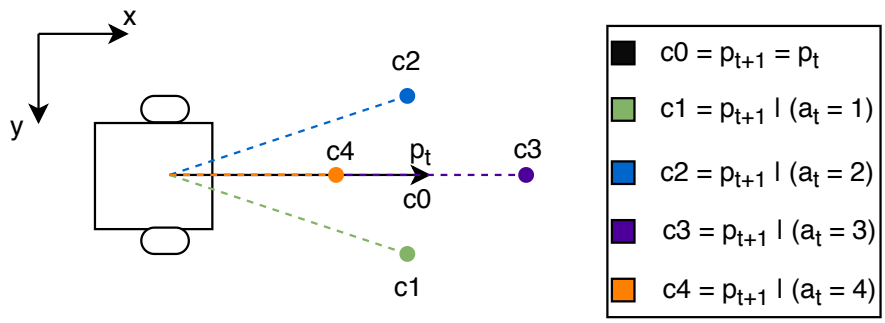
agente (que será a posição desejada do robô). Então o agente move sua "mão"(posição alvo) no campo para guiar o robô na trajetória aprendida desejada que vai interagir de maneira correta com o ambiente.

O espaço de ações do agente com o algoritmo DDQN é definido por cinco valores que comandam o deslocamento do alvo com referência ao respectivo robô, estes valores são:

- c0: Posição alvo permanece inalterada,
- c1: Posição do alvo é rotacionada em -15 graus,
- c2: Posição do alvo é rotacionada em +15 graus,
- c3: Posição do alvo é deslocada em +12 centímetros,
- c4: Posição do alvo é deslocada em -12 centímetros,

Dado um posicionamento p_t qualquer do alvo de um robô no tempo t, a Figura 22 mostra o resultado no tempo $t + 1$ no posicionamento do alvo ao aplicar cada uma das possíveis ações a_t do agente discreto (considerando que, por alguma força externa, o robô fique parado no mesmo lugar).

Figura 22 – Deslocamento do alvo através dos comandos discretos do agente.



Fonte: Autor (2019).

Apesar desta forma de controle não parecer ótima pois abstrai como o robô deve chegar a posição desejada, este controle indireto feito pelo agente retira instabilidades que poderiam acontecer ao comandar o robô diretamente pelas velocidades das rodas, por exemplo: uma mudança brusca de velocidade errada pelo agente geraria um resultado no caminho muito diferente do desejado, enquanto que através do controle um erro gera apenas um leve deslocamento do posicionamento alvo. Isto foi muito importante principalmente nas fases mais embrionárias de todo o sistema construído neste trabalho quando a convergência dos algoritmos era muito difícil de se obter e os parâmetros do ambiente ainda estavam sendo ajustados. O agente discreto com este tipo de controle consegue convergir para comportamentos esperados de um agente inteligente do ambiente IEEE-VSSS que serão demonstrados no próximo capítulo.

Já o agente contínuo executando o algoritmo DDPG possui um espaço de ações mais fácil de modelar por já trabalhar no mesmo domínio das ações do robô. Este agente comanda o robô diretamente através das velocidades linear e angular desejadas do robô. As velocidades linear e angular fornecidas são então traduzidas pelo ambiente Gym para as velocidades de cada roda do robô através das equações de cinemática do robô explicadas no Capítulo 2. Foi escolhido controlar o agente contínuo dessa forma ao invés de controlar diretamente as velocidades de cada roda pois dessa segunda forma o agente possui uma grande dificuldade em andar em linha reta. Controlar diretamente a velocidade das rodas do agente exige que o agente coordene bem as velocidades aplicadas devido a alta interdependência entre elas, enquanto o controle através da velocidade linear e angular faz com que cada ação seja uma componente "independente"que controla parte do movimento.

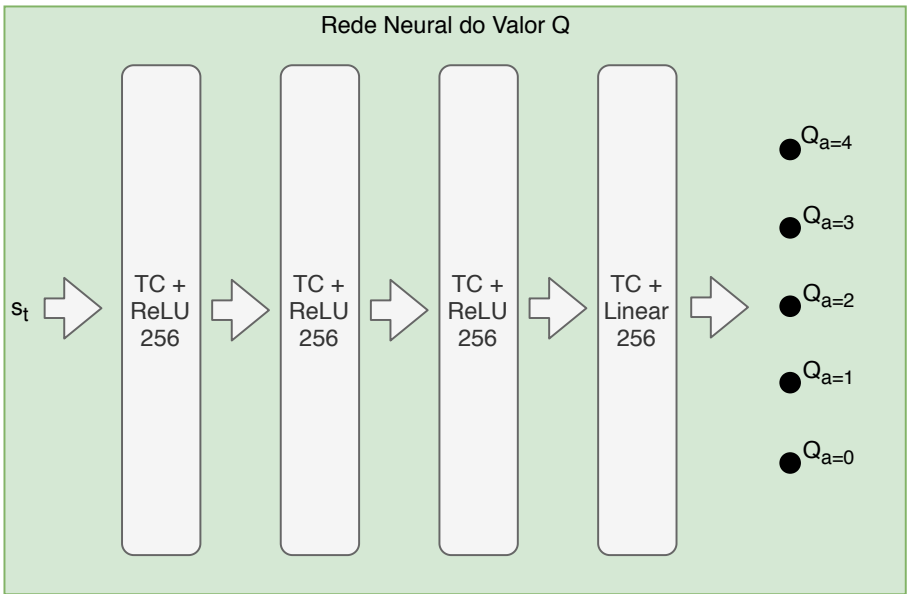
4.5 ARQUITETURA DAS REDES NEURAIS

A arquitetura das redes neurais de cada agente foi escolhida experimentalmente de forma que sejam simples e consigam aprender comportamentos no nosso ambiente de teste, porém não são garantidas de serem ótimas, pois um estudo mais extenso precisaria ser realizado.

Para o agente com o algoritmo DDQN apenas uma rede neural é necessária para aproximar a Q da melhor forma o possível. A Figura 23 mostra como essa função vai ser utilizada para determinar a política, escolhendo a ação com maior valor Q predito. A figura também mostra a arquitetura da rede neural utilizada. A rede neural é formada por três camadas escondidas totalmente conectadas de duzentos e cinquenta e seis neurônios da Camada de Neurônios Totalmente Conectada (TC) com a função de ativação ReLU (GLOROT; BORDES; BENGIO, 2011), uma quarta camada de duzentos e cinquenta e seis neurônios e ativação linear. A camada de saída consiste no Q para cada uma das cinco ações explicadas na próxima seção. A ação escolhida pelo agente será a ação que maximize o Q com probabilidade $1 - \epsilon$ ou uma ação aleatória para incentivar exploração com probabilidade ϵ.

O algoritmo DDPG, por se tratar de um modelo *Actor-Critic*, utiliza duas redes neurais em seu funcionamento. A primeira rede neural é utilizada para aproximar o *actor* e a segunda rede aproxima o *critic*. Neste algoritmo, as ações contínuas necessárias (velocidade linear, v_l e angular, v_a) são diretamente fornecidas pela rede *actor*. A Figura 24 mostra a arquitetura do *critic*. O *critic* consiste de uma rede de uma camada totalmente conectada de 400 neurônios TC, com função de ativação ReLU para processar os dados de observação de entrada, a saída dessa camada junto com um sinal das ações para serem avaliadas são processadas então por uma rede com uma camada totalmente conectada com função de ativação ReLU e uma camada linear que fornece a saída da rede. Já a Figura 25 mostra a arquitetura do *actor*. O *actor* é formado por uma rede de duas camadas totalmente conectadas com funções de ativação ReLU seguidas por uma camada de

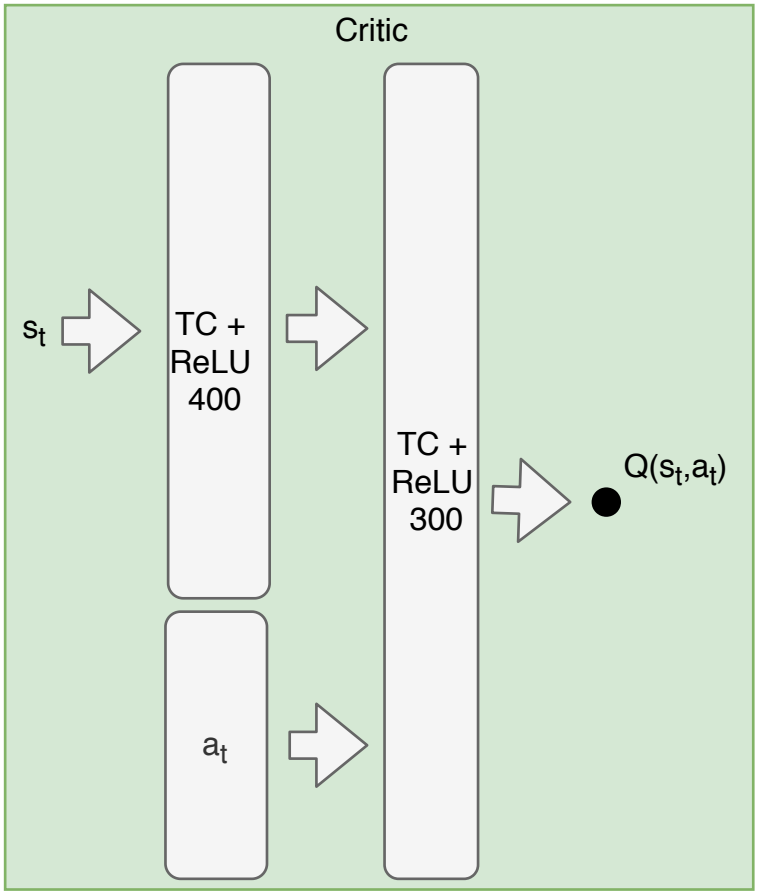
Figura 23 – Arquitetura da rede neural e sua utilização para o algoritmo DDQN na definição de uma política.



Fonte: Autor (2019).

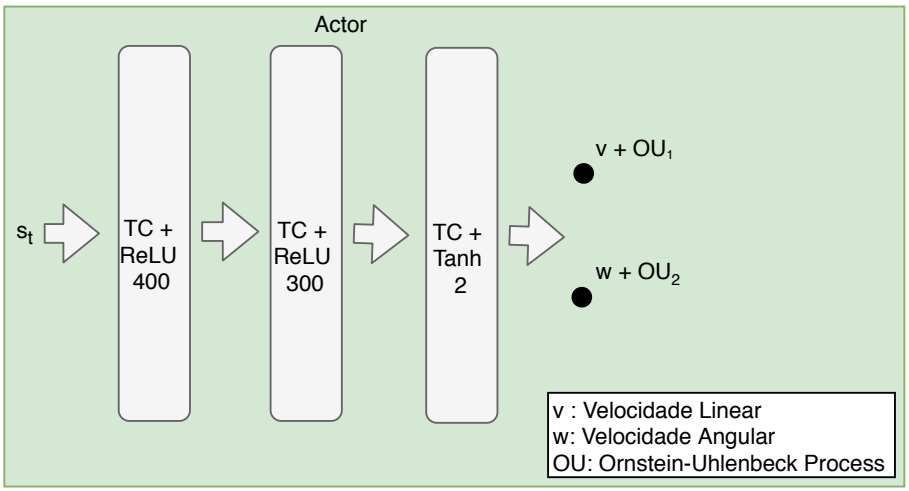
saída totalmente conectada com função de ativação tangente hiperbólica que será responsável por indicar as ações escolhidas pela política. A essas ações são acrescidas o ruído de Ornstein-Uhlenbeck OU para incentivar exploração ao agente.

Figura 24 – Arquitetura da rede utilizada no *critic* do algoritmo DDPG



Fonte: Autor (2019).

Figura 25 – Arquitetura da rede utilizada no *actor* do algoritmo DDPG



Fonte: Autor (2019).

4.6 MÉTRICAS DE AVALIAÇÃO

A avaliação do simulador se dará através da latência das principais operações para o treinamento do agente, da estabilidade nas experiências, e uma análise qualitativa da confiabilidade e reprodutibilidade da simulação.

Normalmente, a avaliação da capacidade de um agente aprendendo via aprendizagem por reforço se dá simplesmente pelas recompensas recebidas por ele durante a sua execução e pela análise visual de seu comportamento. Embora isto seja bem relevante para o problema abordado, o principal interesse está em investigar mais puramente a quantidade de gols feitos ou sofridos pelo agente, já que este é o único objetivo que realmente importa para o jogo. Por isso, a qualidade do agente será avaliada pelas recompensas recebidas por cada parte do *reward shapping*, pela diferença entre a quantidade de gols feitos e levados por ele em cada simulação. Além disso, o agente será avaliado qualitativamente, por seu comportamento no jogo.

5 RESULTADOS

No Capítulo 5 serão mostrados os resultados de testes do ambiente de simulação e dos agentes treinados nele. Ele será separado em três seções. Cada seção será constituída dos resultados quantitativos e qualitativos obtidos durante os testes da solução completa proposta nessa dissertação. A análise qualitativa é extremamente necessária, pois vários aspectos do comportamento do agente não são obtidos diretamente dos gráficos de resultados, como serão vistas nas seções deste capítulo. Primeiro, na Seção 5.1 serão apresentados alguns resultados e discussões sobre o desempenho do ambiente de simulação utilizado. Em seguida, nas Seções 5.2 e 5.3, serão apresentados os desempenhos dos agentes treinados, tanto no domínio discreto com DDQN quanto no domínio contínuo com DDPG, respectivamente, focando na importância do *Reward Shaping* para a obtenção de bons resultados. Serão apresentados os resultados de estudos com um único agente aprendendo no ambiente simulado. Estudos de cooperação em cenários multiagentes estão no escopo de trabalhos futuros com este ambiente.

5.1 RESULTADOS DO AMBIENTE DE SIMULAÇÃO

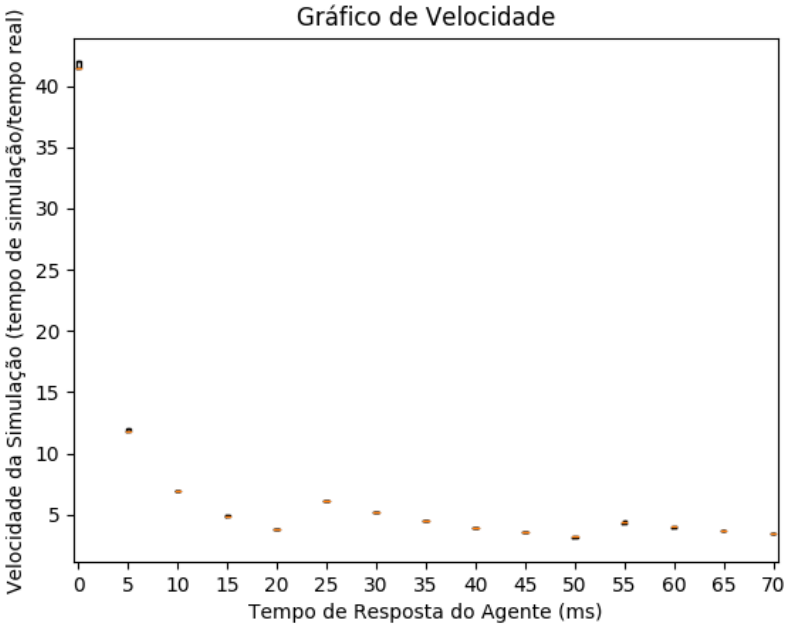
O ambiente de simulação deve ser:

- Estável para que um agente experiencie o ambiente sempre da mesma forma com poucos ruídos,
- Rápido para que um agente consiga obter o máximo número de experiências no menor intervalo de tempo,
- Confiável para que um treinamento não seja prejudicado ou interrompido por quebras na simulação.

A estabilidade e velocidade do simulador deve ser tal que um agente consiga em seu aprendizado obter a melhor experiência, de forma que variações e ruídos não gerem instabilidades nos algoritmos de aprendizagem e sejam impedimentos à convergência destes a uma solução satisfatória. A Figura 26 mostra a adaptação do simulador ambiente de simulação a situação para a qual ele está sendo empregado. À medida que se reduz a velocidade de resposta do agente a simulação também vai diminuindo a sua velocidade para fazer com que esse mesmo agente tenha experiências mais uniformes o possível e liberar capacidade computacional da máquina para o agente.

Porém, visando o funcionamento da simulação em casos assíncronos com mais de um agente se comunicando com o simulador, caso um deles se torne muito lento ou pare de responder, um tempo limite de espera faz com que a simulação prossiga mesmo

Figura 26 – A velocidade do simulador se adapta às capacidades dos agentes visando manter experiências uniformes. Quando o agente demora mais a responder o simulador também desacelera esperando o agente enviar comandos.



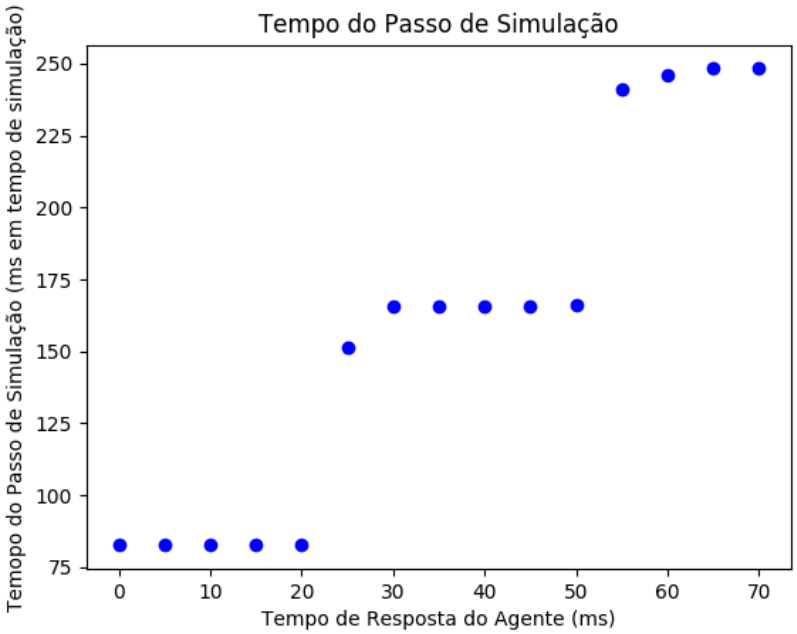
Fonte: Autor (2019).

sem receber resposta de alguns dos agentes, evitando que a simulação fique estagnada por algum motivo externo. Este mecanismo também evita eventuais *deadlocks* entre o simulador e o agente devido a eventuais perdas de mensagens. Com isso, um agente muito lento passa a perceber a simulação em intervalos de tempo maiores e algumas experiências serão automaticamente perdidas por ele. Na Figura 27 mostra a adaptação do passo de simulação percebido pelo agente. Os picos no gráfico em vinte e cinco mili-segundos (ms) e cinquenta e cinco mili-segundos (ms) são referentes aos casos em que o *timeout* é atingido e ocorre uma perda de sincronia entre agente e simulador, a partir de 25ms uma experiência é perdida por passo e de 55ms, dois *frames* são perdidos por passo.

A Tabela 1 mostra a média e desvio padrão de tempo que uma mensagem leva desde que o agente envia um comando até o recebimento da sua reposta. Este processo, mostrado na Figura 28, consiste dos seguintes passos: comando ser empacotado e enviado ao simulador através do nosso ambiente de aprendizagem, o simulador processar o pacote e executar um passo e enviar de volta o novo estado da simulação, o ambiente de aprendizagem desempacotar e processar o pacote retornando ao agente o novo vetor de estado e sua recompensa medida.

Cada reinicio de simulação se dá com o término e reinício do processo do simulador e todas as portas de comunicação. Fazer isto melhora a confiabilidade do processo, evitando que possíveis erros se perpetuem, prejudicando as experiências do agente por longos pe-

Figura 27 – Quando o agente demora mais que o limiar (de aproximadamente 25ms), a simulação executa um passo mesmo sem receber um comando e o agente acaba percebendo a simulação em janelas maiores de tempo.



Fonte: Autor (2019).

Tabela 1 – Tempo médio (em tempo real) que o ambiente leva desde um envio de comando pelo agente até o recebimento da resposta do comando.

Tempo para Executar Comando (ms)	
Média	Desvio Padrão
2.07	0.37

Fonte: Autor (2019).

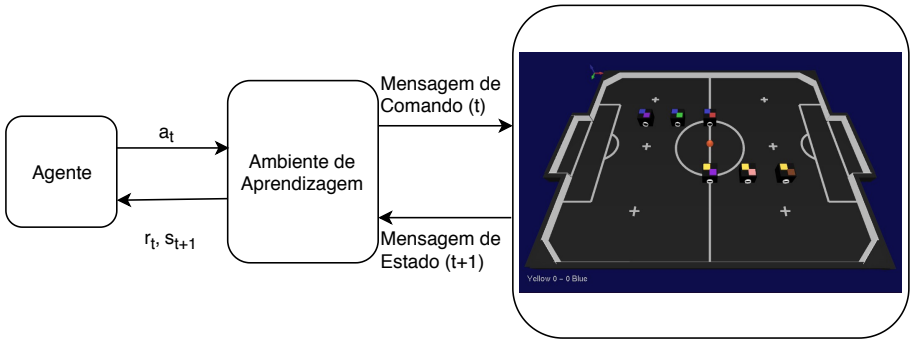
ríodos de treinamento. A Tabela 2 mostra a média e variância do tempo de reiniciar o ambiente de simulação medidos durante dez execuções seguidas de um agente.

Nos experimentos realizados, a velocidade de simulação se manteve em geral entre 250x e 350x, dependendo do hardware utilizado, o que garante que praticamente não ocorreu perda de experiências, algo que seria observado apenas abaixo de 10x de velocidade de simulação, de acordo com a Figura 26.

5.1.1 Análise Qualitativa do Ambiente de Simulação

No geral, o ambiente de simulação se mostrou muito estável, rápido e confiável. A execução de agentes pôde persistir por dias seguidos de computação, durante este tempo o simulador não apresentou quebras ou problemas de comunicação. Os tempos de execução das ações

Figura 28 – Fluxo de comunicação entre agente e simulação.



Fonte: Autor (2019).

Tabela 2 – Tempo médio entre acabar com uma simulação e ela reiniciar até responder um novo comando.

Tempo para Reiniciar Simulação (ms)	
Média	Desvio Padrão
181.06	33.59

Fonte: Autor (2019).

são muito baixos o que faz com que a simulação ocorra fluidamente e que os recursos computacionais fiquem concentrados no aprendizado do agente.

A reinicialização da simulação representa um tempo elevado em comparação com as outras tarefas do simulador, mas ela não chega a representar um gargalo, pois ocorre com muito menos frequência que as outras tarefas. Apesar disso ela pode ser melhorada evitando-se terminar e reiniciar o processo novamente. Um processo mais suave através do reposicionamento dos robôs e reiniciando apenas as variáveis de simulação como tempo e números de gols traria uma economia de tempo na reinicialização da simulação. Por não ser um gargalo muito importante no aprendizado, esta tarefa foi mapeada para atividades futuras.

Outro problema da simulação é que em alguns momentos as colisões entre os objetos eventualmente não são identificadas. Por exemplo, a bola aravessa o robô ficando presa, em outras situações foi observado que o robô sai do campo. Atualmente, como a simulação é reinicializada com frequência e como esse problema ocorre muito raramente. ele não se tornou um empecilho à aprendizagem do agente. A física da simulação é outro aspecto que pode ser melhorado, pode-se observar que por muitas vezes a bola fica muito nos cantos do campo e massa do robô não interfere adequadamente na simulação. Porém, melhorar a física do simulador significa aumentar o custo computacional de cada passo, o que torna o treinamento mais lento.

Apesar destes problemas menores, o ambiente de aprendizagem se mostrou muito robusto, sendo capaz de convergir para agentes executando dois tipos de algoritmo diferen-

tes. É um ambiente que representa um jogo real, altamente versátil podendo ser utilizado com um único agente ou com múltiplos agentes. É um ambiente em que se pode estudar também competição por recursos intra e entre equipes. Com o ambiente implementado, é relativamente simples fazer a política aprendida por um agente simulado controlar um robô real. O ambiente criado se mostra um ótimo cenário de testes para diversos estudos de aprendizagem de comportamentos, *Transfer Learning* e *Sim2Real* em um ambiente altamente dinâmico e competitivo.

5.2 TESTES NO DOMÍNIO DISCRETO

Os agentes que atuam com um domínio de ações discreto foram treinados utilizando o algoritmo DDQN com *replay* de memória. O controle do agente e suas recompensas recebidas se dão como foi explicado no Capítulo 4.

A Tabela 3 mostra os principais hiper-parâmetros de referência escolhidos para o algoritmo DDQN. Estes parâmetros foram experimentalmente ajustados e representam os melhores parâmetros encontrados levando em consideração a convergência do agente para melhores soluções.

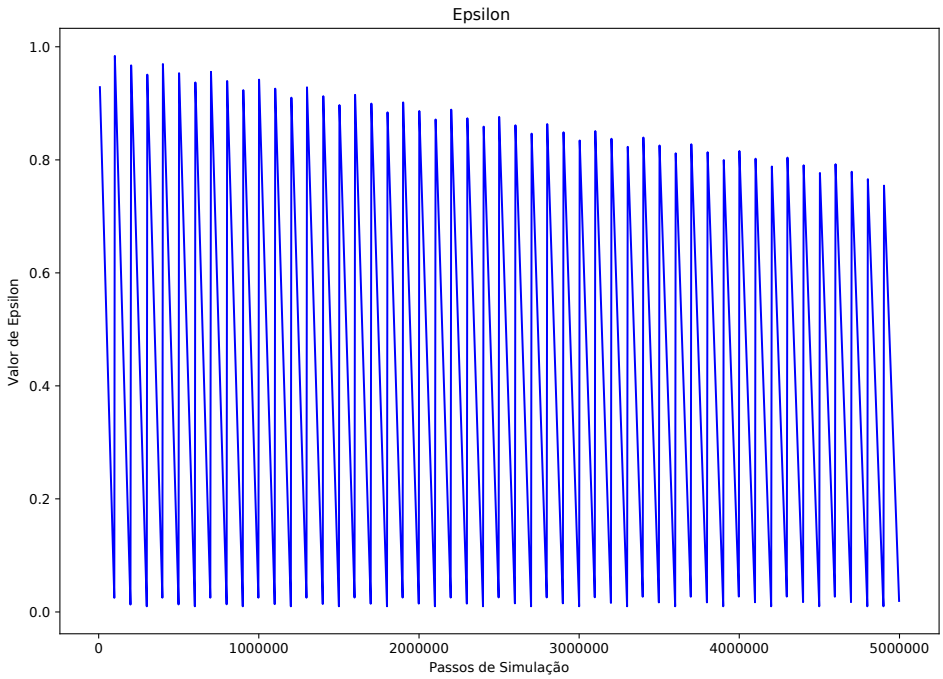
Tabela 3 – Tabela de hiper-parâmetros utilizados com o agente DDQN.

Hiper-parâmetros utilizados com o agente DDQN	
Parâmetro	Valor
Tamanho da Memória de Experiências	5000000
Período de Atualização da Rede *Target*	500000
ϵ Máximo Inicial *Target*	1.0
ϵ Máximo Final *Target*	0.4
ϵ Mínimo *Target*	0.01
γ	0.95
Tamanho do *Batch* de Treinamento	64

Fonte: Autor (2019).

O valor ϵ, que controla o nível de exploração da política do agente, foi estabelecido como variante dentro do intervalo, como mostrado em Figura 29. Quando o valor de ϵ é igual a 1 o agente atua de forma completamente aleatória. Quando o valor de ϵ é 0 o agente segue integralmente a política aprendida. A melhor definição do valor de ϵ da política encontrado durante os testes para o agente é um valor que fica variando em cilos entre um máximo e mínimo determinado. Ao longo do aprendizado o máximo possível do valor ϵ vai decaindo para inibir uma alta exploração do agente conforme ele vai se especializando. Isto faz que com uma alta frequência o agente alterne entre exploração e explotação do espaço de ações.

Figura 29 – Valor *epsilon* alterna periodicamente com alta frequência entre exploração e explotação. O valor máximo possível vai diminuindo durante a aprendizagem do agente até um máximo mínimo de 0.4.



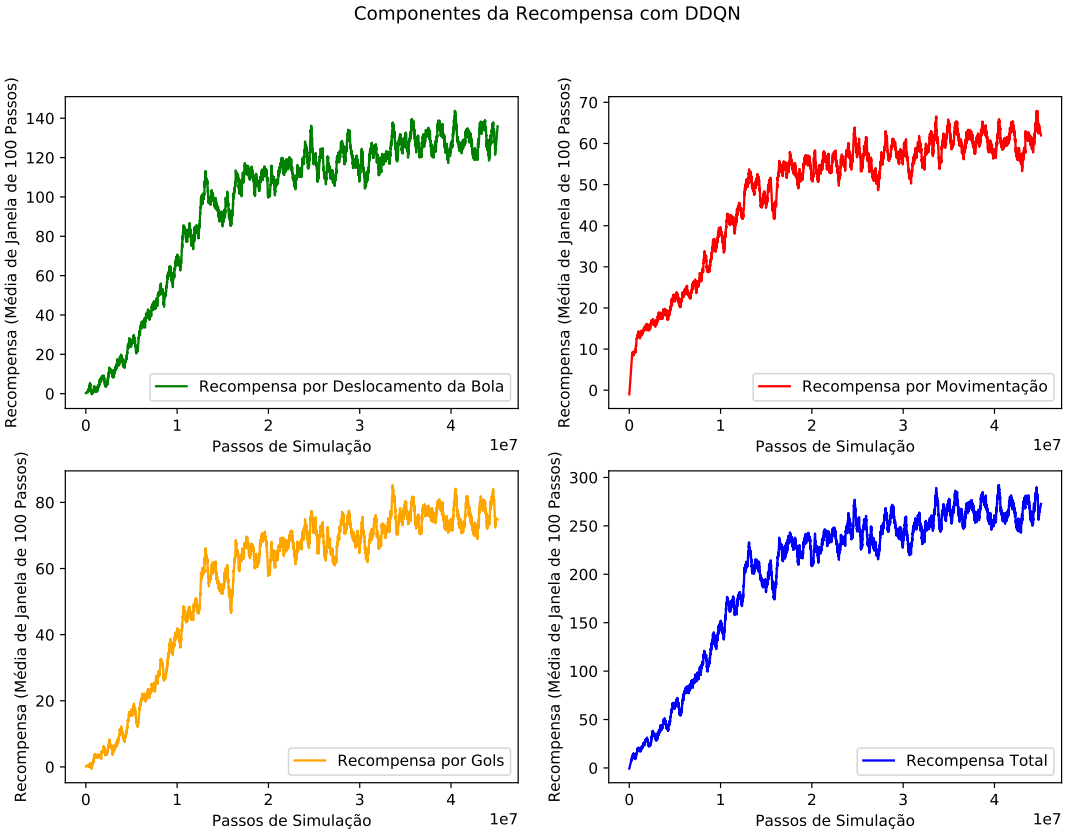
Fonte: Autor (2019).

Através desta política o agente consegue aprender a otimizar cada uma das componentes de sua recompensa, explicadas na Seção 4.3, ao mesmo tempo. Na Figura 30, vê-se que cada componente da recompensa é aprendida em conjunto desde o início do aprendizado tendo formas de curva parecidas apenas com magnitudes diferentes, ou seja, nenhuma componente atrapalha o resultado final em nenhum momento durante o aprendizado e o agente não mostra possuir nenhum *tradeoff* entre elas.

O algoritmo DDQN mostra uma convergência estável na presença de pelo menos a componente de recompensa de movimentação. Porém, o *Reward Shaping* completo possibilita uma melhora no comportamento final do agente Figura 31.

Ao analisar o comportamento do agente através dos gols feitos subtraído dos gols sofridos mostra que o agente consegue aprender um comportamento de guiar a bola eficientemente para a direção do gol adversário. A figura 32 mostra que além de aprender a guiar a bola na direção correta, mostrado anteriormente, o agente aprende a fazer isso de maneira cada vez mais eficiente, necessitando de cada vez menos passos de simulação para chegar até o gol.

Figura 30 – Gráficos mostram a convergência de cada uma das componentes da recompensa. A forma das componentes segue exatamente a forma da curva da recompensa total, indicando que nenhuma das componentes é prejudicial à recompensa total durante a aprendizagem.



Fonte: Autor (2019).

5.2.1 Importância do emprego de *Reward Shaping*

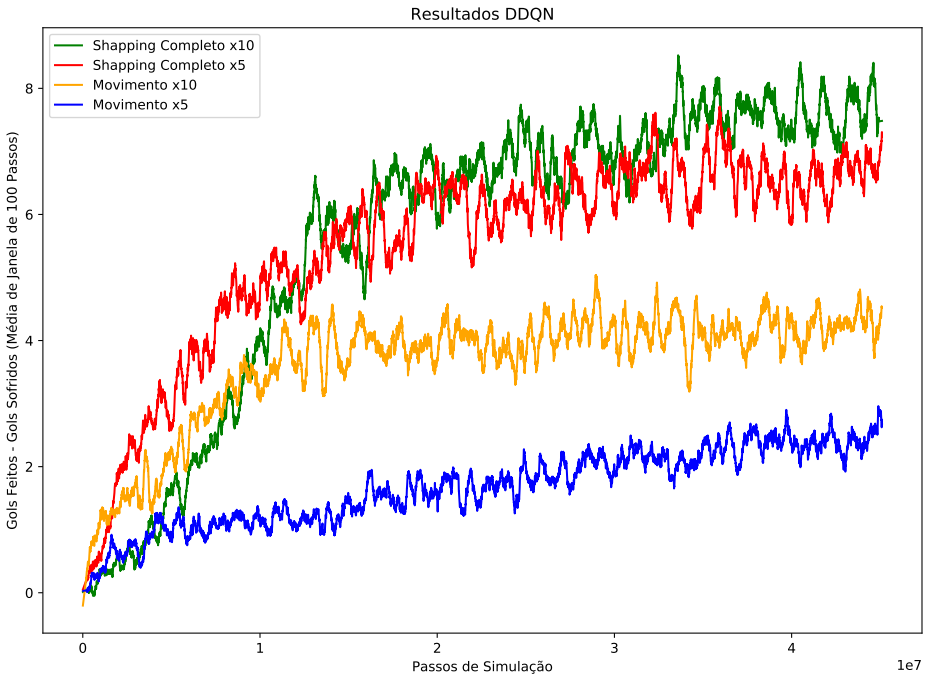
A Figura 33 mostra os resultados de gols dos agentes treinados por aproximadamente dois dias com cada componente do *Reward Shaping* e com sua combinação final. Os pesos finais encontrados experimentalmente de cada componente da recompensa (g_R, gbp_R e m_R na Equação 4.6) assim como o multiplicador da recompensa final são os indicados na Tabela 4.

Tabela 4 – Tabela dos Pesos de cada componente do *Reward Shaping*

Pesos das Componente de *Reward Shaping*			
Gol(g_R)	Deslocamento da Bola(gbp_R)	Movimentação(m_R)	Multiplicador
1.0	0.08	0.02	10.0

Fonte: Autor (2019).

Figura 31 – Média em uma janela de cem passos do valor de gols feitos subtraído de gols sofridos.

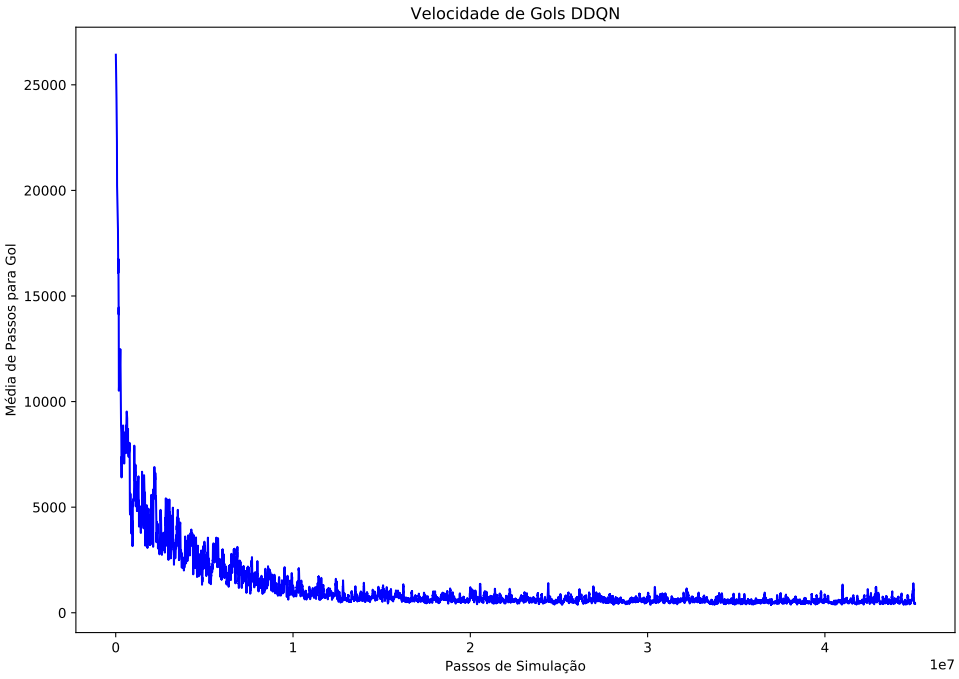


Fonte: Autor (2019).

Como foi explicado no Capítulo 4 fornecer recompensas para o agente apenas quando ele faz gols torna o aprendizado do agente muito difícil, devido a grande esparsidade das recompensas. Esta esparsidade faz com que oos agentes escolhidos não consiga diferir entre as experiências obtidas por ele quais foram benéficas aos seus objetivos e quais foram prejudiciais. Apenas com recompensas esparsas o agente fica estagnado em recompensas baixas com alguns gols esporádicos feitos de maneira aleatória. Apesar do agente receber alguma informação no momento do gol é difícil a ele associar o gol a ações que efetivamente geraram um gol. Por exemplo: um agente que empurra a bola na direção do próprio gol mas a bola bate na trave e acaba fazendo um gol a favor do time do agente, perceberá a ação de empurrar a bola para o próprio gol como positiva e ao tentar novamente repetir a ação ele terá recompensas negativas fazendo gols contra o próprio time, assim levará mais várias experiências parecidas até que este agente perceba que a probabilidade de ter recompensa negativa levando a bola pro próprio gol é muito alta. Situações como esta deixam o treinamento muito lento e instável, para o caso de DDQN esta situação pode nunca convergir.

A primeira preocupação então é fazer com que o agente interaja com a bola e receba recompensas mais comumente. Para isso, além das recompensas para o gol, a agente será recompensado ao aproximar-se da bola e penalizado a distanciar-se dela. Depois de

Figura 32 – Média de passos até que haja um gol.



Fonte: Autor (2019).

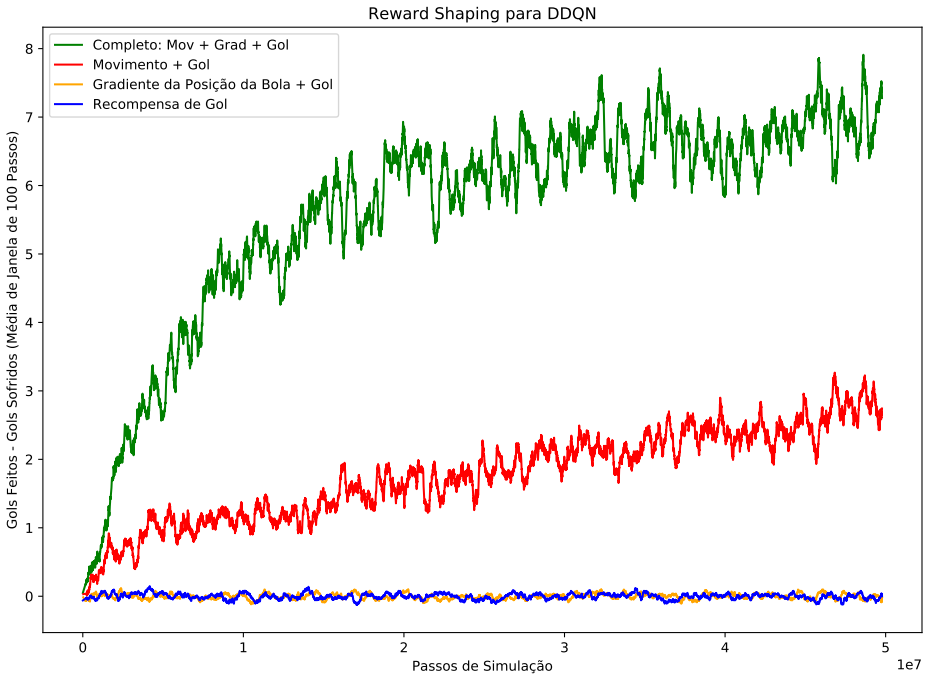
introduzir esta recompensa o agente consegue sair de não aprender a fazer gols para um comportamento interessante de marcar mais gols a favor do que gols contra.

Para melhorar ainda mais o comportamento do nosso agente vamos introduzir a ele o conhecimento do deslocamento da bola do campo. Esta recompensa visa fazer o agente manter a bola perto do gol oponente e manter o deslocamento da bola para o gol oponente. Esta componente sozinha não faz com que o agente consiga sair do seu comportamento inicial mas, quando combinada com a componente de movimento do agente, faz com que o comportamento final do agente melhore e aumenta quantidade de gols feitos pelo agente.

A Figura 34 mostra os resultados de adicionar um fator multiplicador a recompensa final e como o agente reage a isso. O agente com escalonamento de um mostrou um comportamento bem aquém dos outros comparados, escalonamentos significam componentes de gradiente também mais baixo para o treinamento, o que pode levar a um convergência lenta do agente. O agente com escalonamento de dez apresentou o melhor comportamento em termos de gols feitos. Este comportamento não ficou longe do agente com escalonamento de cinco, o que indica que aumentar este fator não traria melhoras significativas ao modelo e poderia fazer o gradiente explodir muito precocemente levando o agente a convergir para ótimos locais ruins ou gerando instabilidade na aprendizagem do agente.

É importante salientar que os resultados obtidos nesta seção, embora utilizem DDQN

Figura 33 – Aprendizagem da DDQN para cada cenário de recompensas analisados.



Fonte: Autor (2019).

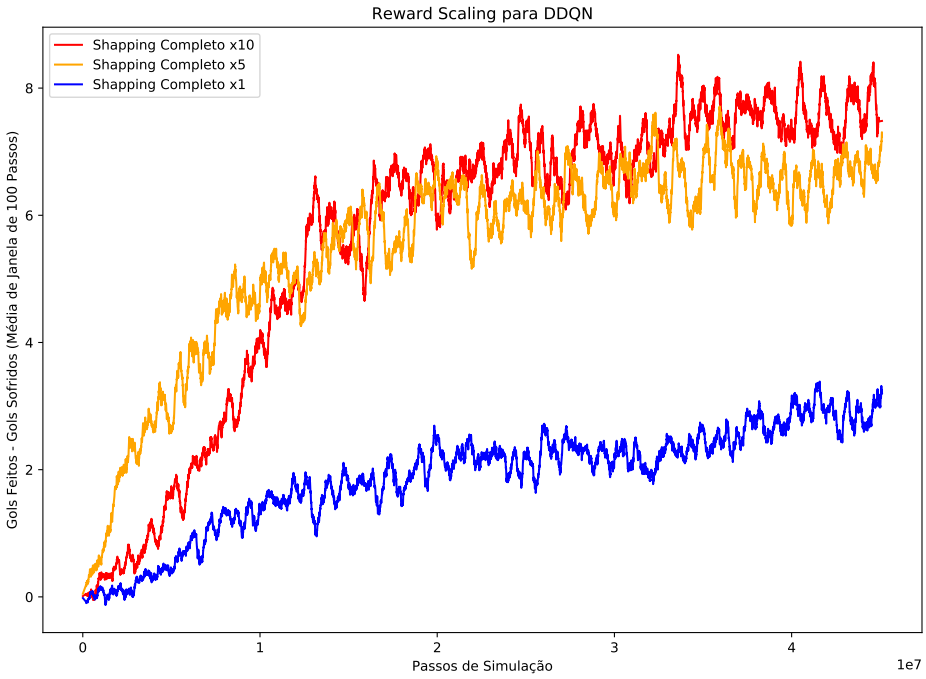
como algoritmo de aprendizagem, foram assumidos como verdadeiros também para o agente com DDPG já que os algoritmos tratam de forma semelhantes as experiências e nenhum deles lidam diretamente com o problema de esparsidade.

Durante a formulação do *Reward Shaping* foi tentado inserir uma penalização para as colisões desnecessárias do agente a fim de preservar um possível agente real no futuro, mas esta componente impedia a convergência adequada dos agentes e foi adiada para trabalhos futuros.

A forma com que as componentes da recompensa foram inseridas faz com que um agente não ganhe recompensa nenhuma ao ficar executando ações ineficientes no ambiente, como ir e voltar para a mesma posição sem interagir com a bola. Isto faz indiretamente que o agente tente interagir com a bola e fazer gols da maneira mais rápida que ele encontrar.

Apesar da componente de deslocamento da bola sozinha não ser suficiente para garantir aprendizagem, ela é muito importante para o comportamento final do agente. Logo, fica claro que a componente de movimentação do agente consegue extrair informação no início do aprendizado, levando o agente até a bola, e em seguida a atuação da componente de deslocamento da bola ensina a melhor forma do agente interagir com a bola para melhorar seu comportamento e, consequentemente, sua quantidade de gols feitos. Estas recompensas adicionadas se tornaram vitais para a convergência do agente até o presente estado deste

Figura 34 – Aprendizagem da DDQN para recompensa total escalonada por valores fixos.



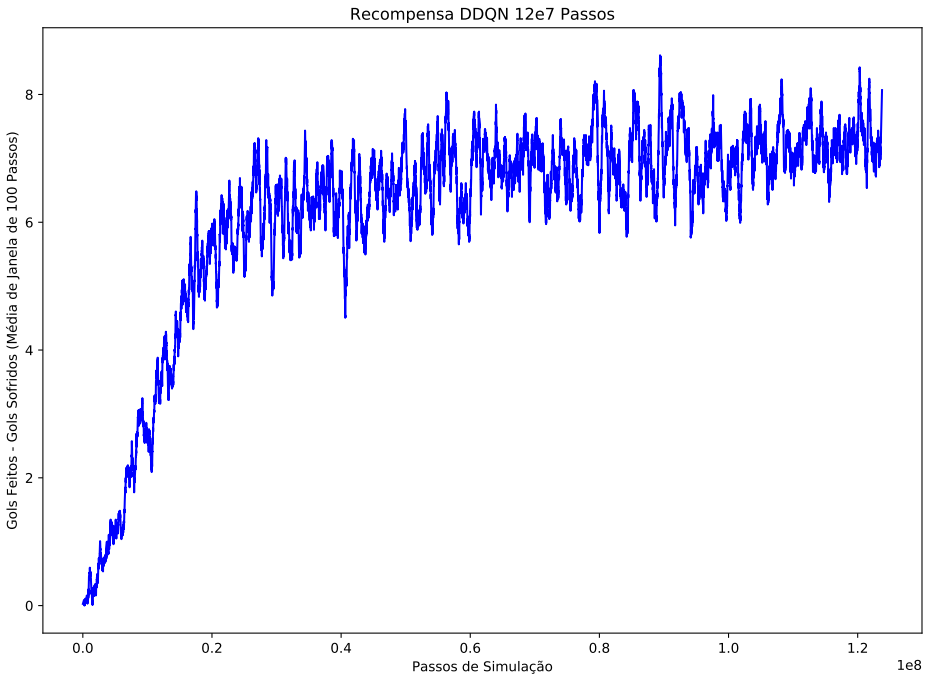
Fonte: Autor (2019).

trabalho.

O escalonamento da recompensa final é outro ponto vital para o aprendizado rápido do agente. O escalonamento serve para controlar a magnitude dos gradientes utilizados para modificar os parâmetros da rede. Deve haver aqui uma preocupação para estes gradientes não explodirem e ficarem descontrolados, o que gera um aprendizado instável para o agente. Durante os experimentos a curva dos gradientes foram monitoradas para garantir que eles permanecessem estáveis.

Dois dias de treino para cada agente utilizando GPU foram suficientes para que eles convergissem para comportamentos próximos ao máximo observado. Quando treinado por mais tempo, após os quarenta milhões de passos, o agente sofre ainda uma pequena melhora no seu comportamento convergindo para um valor máximo a partir do qual não há melhoras aparentes, como mostrado na Figura 35. Devido às limitações de tempo para este trabalho os testes não foram feitos até que os agentes chegassem ao seu ótimo mas até um ponto que os autores consideraram relevante para demonstrar as diferenças entre os resultados.

Figura 35 – Aprendizagem do agente com DDQN para cento e vinte milhões de passos de simulação.



Fonte: Autor (2019).

5.2.2 Análise Qualitativa do Agente Discreto

Com o *reward shapping* o agente executando no domínio discreto consegue convergir para bons comportamentos. A utilização de recompensas contínuas no espaço de estados se mostrou vital para a convergência do agente.

A forma escolhida para controle do o robô com ações discretas fornece boa flexibilidade permitindo uma grande quantidade de movimentos diferentes que o robô pode executar, porém o número de passos que ele precisa para realizar uma manobra aumenta com sua complexidade. Por terem um controle indireto, que funciona integrando os comando recebidos ao longo do tempo, esses agentes são invariantes a ruídos esporádicos nos comandos e, então, são estáveis mesmo na presença de alguma incerteza no agente. Por outro lado, este tipo de controle abordado impede que o agente faça uso completo das capacidades dos motores do robô, já que em poucos casos o agente consegue atuar na sua velocidade máxima ou fazer mudanças ríspidas em seus movimentos.

Outros controles testados, como controlar as velocidades diretas das rodas ou controlar as velocidades linear e angular, se mostraram muito instáveis nos treinamentos dos agentes no estado inicial deste trabalho. Por apresentarem mudanças mais rápidas, os agentes treinados cometiam muitos erros.

Não foi possível para os agentes aprender a evitar completamente as colisões desnecessárias entre robôs. Ao configurar uma penalidade neste cenário um ruído muito alto é acrescentado à aprendizagem do robô.

O comportamento do agente treinado em jogo é de levar a bola para o gol o mais rápido o possível, porém em alguns momentos raros o agente toma ações erradas batendo na bola no sentido errado chegando até a fazer gols no seu próprio gol.

Nos comportamentos do agente treinado fica claro que o robô consegue adquirir comportamentos esperados para um jogador de IEEE-VSSS, por exemplo: ele vai até a bola pelo lado certo, espera a bola em sua posição futura para fazer um gol, evita obstáculos e até as vezes usa os obstáculos a seu favor, gira para remover a bola das quinas e demais situações de bola presa.

5.3 TESTES NO DOMÍNIO CONTÍNUO

Assim como foi mostrado no caso contínuo, a Tabela 5 mostra os principais hiper-parâmetros de referência escolhidos para o algoritmo DDPG. Estes parâmetros foram experimentalmente ajustados e representam os melhores parâmetros encontrados levando em consideração a convergência do agente para melhores soluções.

Tabela 5 – Tabela de hiper-parâmetros utilizados com o agente DDPG.

Hiper-parâmetros utilizados com o agente DDPG	
Parâmetro	Valor
Tamanho da Memória de Experiências	5000000
Taxa de Atualização da Rede *Target*	0.001
ϵ Máximo Inicial *Target*	1.0
ϵ Máximo Final *Target*	0.4
ϵ Mínimo *Target*	0.01
γ	0.95
Tamanho do *Batch* de Treinamento	128

Fonte: Autor (2019).

O algoritmo DDPG dá ao agente a possibilidades de controlar diretamente suas velocidades, porém isso aumenta muito a gama de movimentos possíveis, ampliando a quantidade de comportamentos não ótimos, como por exemplo se locomover girando sobre o próprio eixo. Por isso, a componente de energia foi adicionada ao *Reward Shaping*, com peso ω_e mostrado em Tabela 6, fazendo com que o agente privilegie aprender uma movimentação mais eficiente desde o início do treinamento, penalizando movimentos ineficientes. A Figura 36 mostra como o agente se comporta com e sem a componente de energia. Assim como no domínio discreto, o agente executando apenas recebendo recom-

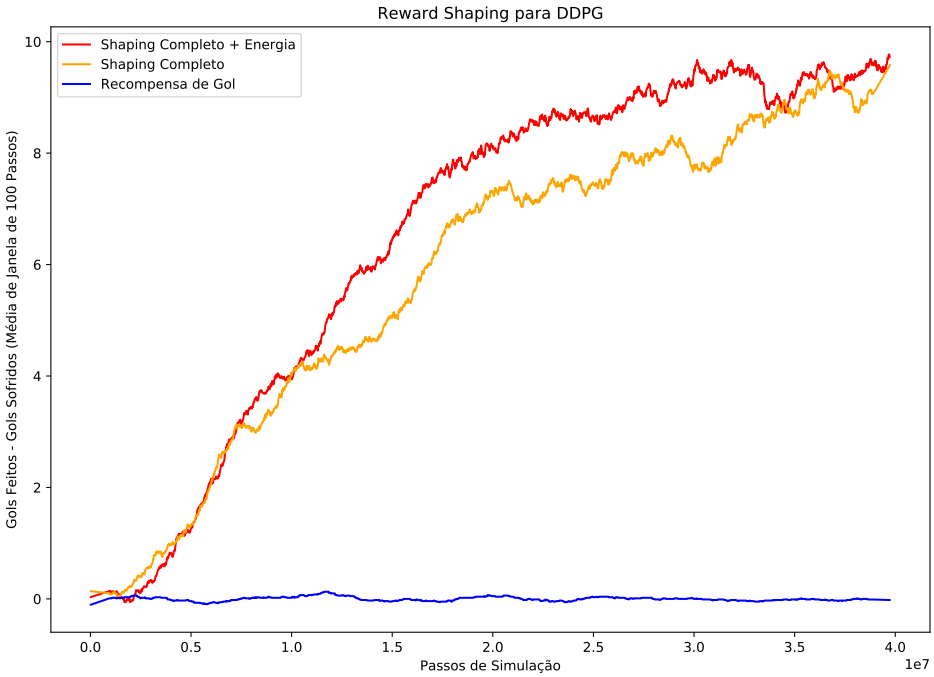
pensas esparsas não consegue aprender um comportamento. Embora para o resultado final aparente que a componente de energia não tenha sido tão importante, é essencial observar que com a componente de energia, o agente chega ao patamar final cerca de dez milhões de passos antes, quando comparado ao agente com o *Reward Shaping* usado na DDQN.

Tabela 6 – Tabela dos Peso da Componente de Energia para o DDPG.

Peso da Componente de Energia para o DDPG	
Peso Energia (ω_e)	0.00005

Fonte: Autor (2019).

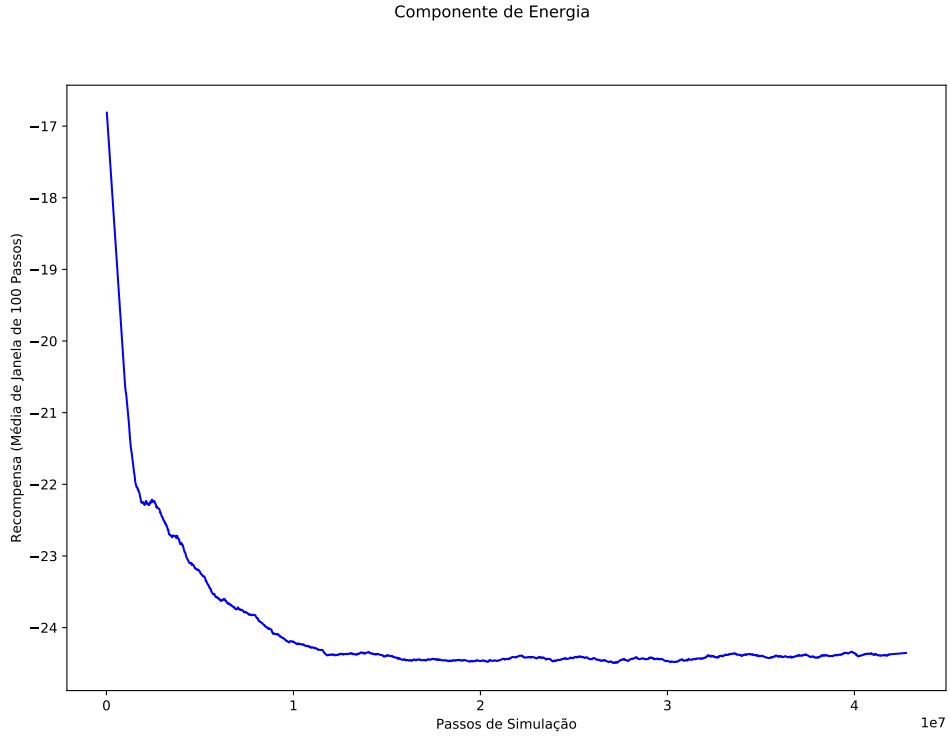
Figura 36 – Comportamento do agente com a adição da componente de energia à recompensa recebida.



Fonte: Autor (2019).

As componentes de recompensa se comportam de maneira similar ao agente no domínio discreto. Mas a nova componente de energia se comporta diferente, ao contrário de todas as outras ela é uma componente de penalização. Esta componente é multiplicada por um peso experimentalmente ajustado de um centésimo de milésimo. A componente de energia faz com que haja um *tradeoff* entre o agente se movimentar no ambiente e a sua penalidade de energia. Esse *tradeoff* faz com que o agente aprenda uma movimentação mais efetiva. A figura 37 mostra que no início o agente passa a ser cada vez mais

Figura 37 – Comportamento da componente de energia durante execução do agente.



Fonte: Autor (2019).

penalizado, provavelmente devido ao grande ganho das recompensas de movimentação, mas logo em seguida a componente se estabiliza enquanto o agente continua a melhorar as recompensas recebidas através das outras componentes.

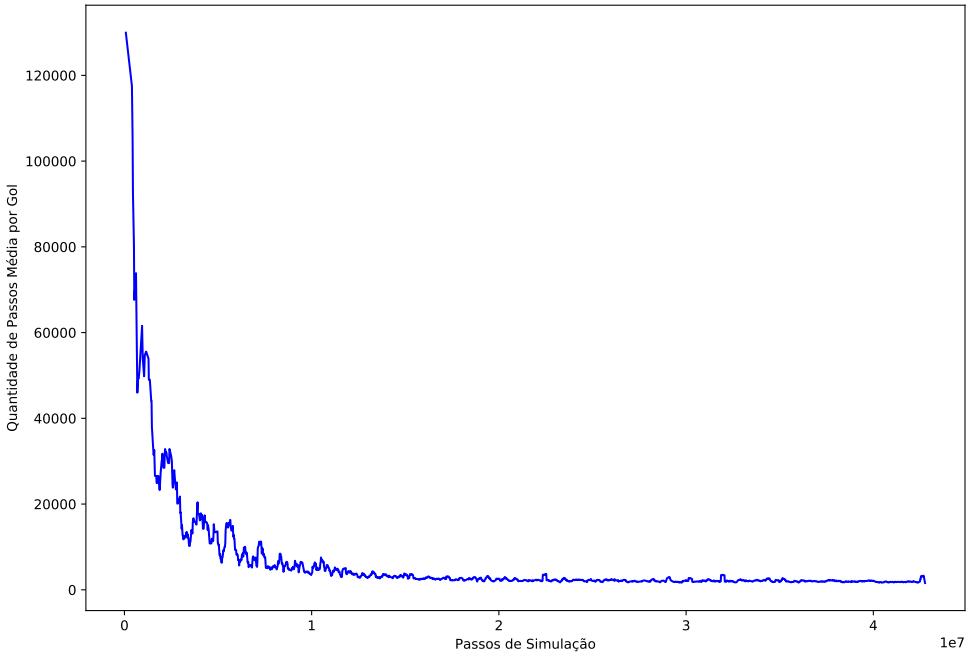
O agente no domínio contínuo, assim como no domínio discreto, mantém o comportamento de diminuir o número de passos até chegar ao gol durante seu aprendizado Figura 38.

5.3.1 Análise Qualitativa

O agente controlado no domínio contínuo utilizou de muitos avanços estudados no domínio discreto. Estes avanços foram muito importantes na convergência dos agentes no domínio contínuo, pois diversos tipos de ruído presentes no ambiente inicial foram eliminados, através dos ajustes de gradientes para não explodir ou desaparecer e sincronização do simulador, por exemplo.

Para este agente é mais fácil tomar ações ineficientes e pouco estáveis no começo do seu treinamento e convergir para ótimos locais ruins para seu comportamento, como girar indefinidamente ou ir girando para a bola. Acrescentar a ideia de penalizar a energia gasta garantiu ao agente conseguir evitar comportamentos radicais que eram sub-ótimos no início dos testes e mais posteriormente se mostrou bom para agilizar a convergência

Figura 38 – Quantidade da passos até haver um gol.



Fonte: Autor (2019).

do agente para as melhores soluções encontradas. Ao considerar o consumo de energia o agente aprender a ir mais diretamente para o seu objetivo. Isto pode também representar uma politica mais adequada para o controle de robôs reais, onde o consumo de bateria e o desgaste das peças são fatores importantes.

Através do seu controle contínuo, o agente com DDPG consegue rapidamente modificar seu comportamento da forma que ele quiser, por isso, o agente tem capacidades de executar ações com muito mais destreza que o agente no domínio discreto. Este diferencial se reflete diretamente no resultado obtido pelo agente, que consegue uma taxa de gols feitos maior. Esta diferença também é perceptível visualmente ao observar-se a movimentação do agente no ambiente. Além de conseguir um desempenho melhor o agente contínuo precisa de aproximadamente a mesma quantidade de passos para chegar ao seu melhor nível.

6 CONCLUSÕES E TRABALHOS FUTUROS

Este capítulo, na Seção 6.1 vai retomar os principais resultados obtidos neste trabalho e suas importâncias. Na Seção 6.2 serão explicados os objetivos alcançados neste trabalho. Na Seção 6.3, serão elencados todas as contribuições que esta dissertação traz para a comunidade científica de robótica, aprendizagem por reforço e aprendizagem de comportamento. Por último, na Seção 6.4, serão listados todos os possíveis estudos de diversas áreas que podem ser feitos a partir das contribuições desta dissertação.

6.1 RESUMO DOS RESULTADOS OBTIDOS

Aprendizagem por reforço tem uma aplicação muito especial no campo de robótica por conseguir aprender comportamentos em tarefas complexas que seriam difíceis de modelar. Esta dissertação de mestrado introduz uma nova ferramenta para auxiliar na construção de uma inteligência robótica em ambientes altamente dinâmicos e em tarefas competitivas que exijam colaboração entre agentes. Este ambiente pode ser útil para diversas áreas de pesquisa em aprendizagem por reforço e robótica, como: *transfer learning*, cooperação, competição, controle robótico, ambientes multiagentes, etc. Todas estas características em conjunto destacam o ambiente criado de todos os outros estudados neste trabalho.

O ambiente criado mostrou-se computacionalmente eficiente durante seus experimentos tomando aproximadamente dois mili-segundos de tempo real para processar o efeito de um comando feito pelo agente em um computador comum de mesa. Cada comando corresponde a setenta e cinco mili-segundos de simulação, então, em um dia o simulador é capaz de executar mais de dez mil partidas de cinco minutos, este cálculo desconsidera o tempo de reinicialização do ambiente. Além disso, o ambiente se mostra adaptável para agentes executando em máquinas diferentes desacelerando sua execução quando o agente não consegue enviar comandos rápido o suficiente. O ambiente criado foi executado por dois dias seguidos para este trabalho (e por semanas em outros testes) sem interrupção ou quebra da execução, esta características mostra a confiabilidade do sistema como um todo.

O trabalho de ajustar as recompensas do ambiente mostrou-se muito importante na convergência e resultado final para todos os algoritmos testados. Adicionar componentes contínuas à recompensa esparsa inicial fez com que os agentes saíssem de um estado em que não conseguiam aprender para um comportamento inteligente evidente de fazer gols o mais rápido possível. Na área de aprendizagem por reforço o aprendizado através de recompensas esparsas é um problema em aberto muito estudado que ainda carece de soluções gerais.

Utilizando deste ambiente, o trabalho mostra que é possível treinar agentes inteligentes

tanto no domínio contínuo quanto no domínio discreto para jogar futebol no campo. É evidente dos resultados do trabalho que o agente controlado no domínio contínuo tem mais facilidade em aprender a tarefa de fazer gols do que sua alternativa no domínio discreto. Isto se deve pela essência contínua da tarefa, essência essa que está muito presente nas tarefas de robóticas que lidam diretamente com os atuadores.

Com as políticas para o domínio discreto treinadas no ambiente simulado criado neste trabalho, Lima Jr.(2019) foi capaz de transferir o comportamento aprendido para um robô real. O que mostra que os comportamento aprendidos podem ser generalizados para outros ambientes similares e que o *transfer learning* pode ser aplicado com sucesso para competições reais de IEEE-VSSS.

6.2 OBJETIVOS ALCANÇADOS

Nesta dissertação, foi feito um estudo da área de aprendizagem por reforço aplicado a robótica. Foram abordados vários algoritmos e técnicas comumente utilizadas nos trabalhos mais recentes. Além disso, foi feito um estudo dos simuladores que representavam um ambiente parecido ao implementado durante este trabalho, focando nas diferenças do ambiente proposto.

O trabalho apresentado traz contribuições importantes para o estudo da aprendizagem por reforço aplicada à robótica. O ambiente de robótica criado pode ser utilizado tanto para robôs simulados como robôs reais com velocidade, estabilidade e confiabilidade. Este ambiente produzido pode servir como futuro padrão para construção de inteligência das equipes de IEEE-VSSS mudando as formas como as equipes programam suas estratégias atualmente para aplicar RL.

O ambiente foi utilizado com sucesso para treinar agentes tanto no domínio discreto de ações quanto no domínio contínuo. A partir do trabalho de *Reward Shaping* aplicado durante este trabalho os agentes foram melhorando seus resultados a ponto de se tornarem, no final de suas execuções, extremamente eficientes em fazer gols no ambiente. Chegando a um comportamento de fazer mais de dois gols por minuto. Isto introduz à categoria IEEE-VSSS uma função de recompensa que pode ser usada para treinar os agente da categoria.

O ambiente foi utilizado com sucesso para treinar um agente e em seguida aplicar o conhecimento aprendido em um robô real de forma que o comportamento aprendido se mantenha (Lima Jr., 2019). Com isso, o ambiente se mostra preparado para construir estratégias que serão utilizadas por times reais nas competições da categoria IEEE-VSSS.

6.3 CONTRIBUIÇÕES

A principal contribuição do trabalho foi inserir para a ciência um ambiente de testes altamente dinâmico com todas as características apresentadas e que visa o *sim2real*. Com

ele comportamentos de agentes podem ser estudados e diversas teorias de aprendizagem, cooperação, atribuição de recompensa podem ser demonstrados tanto em simulação como em extrapolados para o mundo real. O ambiente foi validado tanto em simulação como em robôs reais com resultados promissores. O ambiente pode ser facilmente expandido para outras atividades em cenários de robótica parecidos para os mais diversos objetivos como controle de agente em armazéns como o da Amazon. O ambiente será disponibilizado para a sociedade em breve, assim que o time jogar sua primeira competição real de futebol de robôs, planejado para a segunda metade do ano 2019.

6.4 TRABALHOS FUTUROS

O trabalho apresentado nesta dissertação é apenas um início de diversas pesquisas que podem ser feitas utilizando o sistema aqui criado. Alguns trabalhos já estão sendo realizados, como o estudo do treinamento de um time completo ou fazer o agente jogar contra si mesmo (*self-play*) para que ele se desenvolva. Alguns outros trabalhos que podem ser aplicados no ambiente criado aqui são:

- Publicar na conferência do IEEE International Conference on Robotics and Automation o ambiente criado e os resultados coletados nesta Dissertação para que qualquer pessoa possa usar o ambiente para desenvolver seu próprio time, estratégia e estudo em aprendizagem por reforço.
- Aplicar estas técnicas em outras categorias de robótica para mostrar que o ambiente pode ser facilmente extensível a outras aplicações de robótica.
- Propor o ambiente criado como padrão de competições de RL para a evolução dos algoritmos e modelos de robóticas em ambientes dinâmicos e competitivos.
- Aplicar *Imitation learning* e *Inverse Reinforcement Learning* utilizando a política de ações modeladas manualmente para um time ou através de gravações de jogos da categoria IEEE-VSSS.
- Estudos de cooperação entre agentes independentes para que eles funcionem como um time organizado em campo e como atribuir as recompensas a cada agente separado.
- Melhorar o simulador para eliminar alguns cenários de ruído, como: a bola se prende embaixo de um agente ou ela cai do campo.
- Expandir os ambiente para novos desafios, como: treinamento específico de goleiros, pênaltis, etc.

- Aplicar algoritmos online ou model-based de aprendizagem por reforço que são muito promissores na área de robótica, como: *Proximal Policy Optimization*, *Advantage Actor-Critic*, *Trust Region Policy Optimization*.
- Estudar o aprendizado de agentes a partir diretamente das imagens do jogo através de redes convolucionais.
- Adicionar um sistema de torneio para o treinamento de agentes como o utilizado em (VINYALS et al., 2019).
- Utilizar um time completo para aprender a tarefa de jogar futebol (sendo feito).
- Aplicar *self-play* para que o agente evolua jogando contra si mesmo (sendo feito).

Estas e outras tarefas fazem do sistema proposto nesta dissertação um ambiente muito versátil para o aprendizado de robótica através de aprendizagem por reforço. Tudo criado neste trabalho está sendo lapidado e terá o código disponibilizado junto com a participação de um time completo treinado na plataforma criada em breve.

REFERÊNCIAS

ANDRYCHOWICZ, M.; BAKER, B.; CHOCIEJ, M.; JOZEFOWICZ, R.; MCGREW, B.; PACHOCKI, J.; PETRON, A.; PLAPPERT, M.; POWELL, G.; RAY, A. et al. Learning dexterous in-hand manipulation. *arXiv preprint arXiv:1808.00177*, 2018.

ARULKUMARAN, K.; DEISENROTH, M. P.; BRUNDAGE, M.; BHARATH, A. A. A brief survey of deep reinforcement learning. *arXiv preprint arXiv:1708.05866*, 2017.

BALCH, T. et al. Learning roles: Behavioral diversity in robot teams. *College of Computing Technical Report GIT-CC-97-12, Georgia Institute of Technology, Atlanta, Georgia*, v. 73, 1997.

BOJARSKI, M.; TESTA, D. D.; DWORAKOWSKI, D.; FIRNER, B.; FLEPP, B.; GOYAL, P.; JACKEL, L. D.; MONFORT, M.; MULLER, U.; ZHANG, J. et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

BOUSMALIS, K.; IRPAN, A.; WOHLHART, P.; BAI, Y.; KELCEY, M.; KALAKRISHNAN, M.; DOWNS, L.; IBARZ, J.; PASTOR, P.; KONOLIGE, K. et al. Using simulation and domain adaptation to improve efficiency of deep robotic grasping. In: IEEE. *2018 IEEE International Conference on Robotics and Automation (ICRA)*. [S.l.], 2018. p. 4243–4250.

BRAGA, A. de P.; FERREIRA, A. C. P. de L.; LUDERMIR, T. B. *Redes neurais artificiais: teoria e aplicações*. [S.l.]: LTC Editora Rio de Janeiro, Brazil:, 2007.

BRAIN, G. *Google Rearch Football Ambiente Simulado de Futebol do Google Brain*. 2019. Disponível em: <https://github.com/google-research/football/blob/master/paper.pdf>.

BROCKMAN, G.; CHEUNG, V.; PETTERSSON, L.; SCHNEIDER, J.; SCHULMAN, J.; TANG, J.; ZAREMBA, W. *OpenAI Gym*. 2016.

CHE, Z.; PURUSHOTHAM, S.; CHO, K.; SONTAG, D.; LIU, Y. Recurrent neural networks for multivariate time series with missing values. *Scientific reports*, Nature Publishing Group, v. 8, n. 1, p. 6085, 2018.

COMPETITION, L. A. I. S. R. *VSSS Livro de Regras*. 2008. Disponível em: <http://www.cbrobotica.org/wp-content/uploads/2014/03/VerySmall2008_en.pdf>.

DEVELOPERS, G. *Protocol Buffers*. 2019. <https://developers.google.com/protocol-buffers/>.

GHOSH, D.; SINGH, A.; RAJESWARAN, A.; KUMAR, V.; LEVINE, S. Divide-and-conquer reinforcement learning. *arXiv preprint arXiv:1711.09874*, 2017.

GLOROT, X.; BORDES, A.; BENGIO, Y. Deep sparse rectifier neural networks. In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. [S.l.: s.n.], 2011. p. 315–323.

GOMEZ-URIBE, C. A.; HUNT, N. The netflix recommender system: Algorithms, business value, and innovation. *ACM Trans. Manage. Inf. Syst.*, ACM, New York, NY, USA, v. 6, n. 4, p. 13:1–13:19, dez. 2015. ISSN 2158-656X. Disponível em: <http://doi.acm.org/10.1145/2843948>.

GU, S.; HOLLY, E.; LILLICRAP, T.; LEVINE, S. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In: IEEE. *2017 IEEE international conference on robotics and automation (ICRA)*. [S.l.], 2017. p. 3389–3396.

HANSON, J.; PALIWAL, K.; LITFIN, T.; YANG, Y.; ZHOU, Y. Accurate prediction of protein contact maps by coupling residual two-dimensional bidirectional long short-term memory with convolutional neural networks. *Bioinformatics*, Oxford University Press, v. 34, n. 23, p. 4039–4045, 2018.

HASSELT, H. V.; GUEZ, A.; SILVER, D. Deep reinforcement learning with double q-learning. In: *Thirtieth AAAI Conference on Artificial Intelligence*. [S.l.: s.n.], 2016.

HASSELT, H. V.; GUEZ, A.; SILVER, D. Deep reinforcement learning with double q-learning. In: *Thirtieth AAAI conference on artificial intelligence*. [S.l.: s.n.], 2016.

HEBB, D. O. *The Organization of Behavior*. [S.l.]: New York: Wiley, 1949.

HEESS, N.; SRIRAM, S.; LEMMON, J.; MEREL, J.; WAYNE, G.; TASSA, Y.; EREZ, T.; WANG, Z.; ESLAMI, S.; RIEDMILLER, M. et al. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017.

JAMES, S.; WOHLHART, P.; KALAKRISHNAN, M.; KALASHNIKOV, D.; IRPAN, A.; IBARZ, J.; LEVINE, S.; HADSELL, R.; BOUSMALIS, K. Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2019. p. 12627–12637.

KAHN, G.; VILLAFLOR, A.; DING, B.; ABBEEL, P.; LEVINE, S. Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation. In: IEEE. *2018 IEEE International Conference on Robotics and Automation (ICRA)*. [S.l.], 2018. p. 1–8.

KOBER, J.; BAGNELL, J. A.; PETERS, J. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, SAGE Publications Sage UK: London, England, v. 32, n. 11, p. 1238–1274, 2013.

LEVINE, J. S. S.; FINN, C. *CS 294: Deep Reinforcement Learning*. 2018. <http://rll.berkeley.edu/deeprlcourse/>.

LEVINE, S.; PASTOR, P.; KRIZHEVSKY, A.; IBARZ, J.; QUILLEN, D. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, SAGE Publications Sage UK: London, England, v. 37, n. 4-5, p. 421–436, 2018.

LILLICRAP, T. P.; HUNT, J. J.; PRITZEL, A.; HEESS, N.; EREZ, T.; TASSA, Y.; SILVER, D.; WIERSTRA, D. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Lima Jr., J. N. de O. *Sim-to-Real: Adapting the Control forRobots Playing Soccer in the Real World*. [S.l.], 2019.

LIN, L.-J. *Reinforcement learning for robots using neural networks*. [S.l.], 1993.

LIU, S.; LEVER, G.; MEREL, J.; TUNYASUVUNAKOOL, S.; HEESS, N.; GRAEPEL, T. Emergent coordination through competition. *CoRR*, abs/1902.07151, 2019. Disponível em: <http://arxiv.org/abs/1902.07151>.

LONG, J.; SHELHAMER, E.; DARRELL, T. Fully convolutional networks for semantic segmentation. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2015. p. 3431–3440.

MCCULLOCH, W. Pitts, wh (1943). a logical calculus of ideas im-manent in nervous activity. *Bull. math. Biophys*, v. 5, p. 115–33, 86.

MNIH, V.; KAVUKCUOGLU, K.; SILVER, D.; GRAVES, A.; ANTONOGLOU, I.; WIERSTRA, D.; RIEDMILLER, M. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

MNIH, V.; KAVUKCUOGLU, K.; SILVER, D.; RUSU, A. A.; VENESS, J.; BELLEMARE, M. G.; GRAVES, A.; RIEDMILLER, M.; FIDJELAND, A. K.; OSTROVSKI, G. et al. Human-level control through deep reinforcement learning. *Nature*, Nature Publishing Group, v. 518, n. 7540, p. 529, 2015.

MONTGOMERY, W. H.; LEVINE, S. Guided policy search via approximate mirror descent. In: *Advances in Neural Information Processing Systems*. [S.l.: s.n.], 2016. p. 4008–4016.

NG, A. Y.; HARADA, D.; RUSSELL, S. Policy invariance under reward transformations: Theory and application to reward shaping. In: *ICML*. [S.l.: s.n.], 1999. v. 99, p. 278–287.

OPENAI. *Ambientes de Robótica Descrição dos Ambientes de Robóticas do OpenAI*. 2018. Disponível em: <https://gym.openai.com/envs/#robotics>.

OPENAI. *How to Train Your OpenAI Five*. 2019. <https://openai.com/five/>.

PLAPPERT, M.; ANDRYCHOWICZ, M.; RAY, A.; MCGREW, B.; BAKER, B.; POWELL, G.; SCHNEIDER, J.; TOBIN, J.; CHOCIEJ, M.; WELINDER, P.; KUMAR, V.; ZAREMBA, W. *Multi-Goal Reinforcement Learning: Challenging Robotics Environments and Request for Research*. 2018.

PLAPPERT, M.; ANDRYCHOWICZ, M.; RAY, A.; MCGREW, B.; BAKER, B.; POWELL, G.; SCHNEIDER, J.; TOBIN, J.; CHOCIEJ, M.; WELINDER, P. et al. Multi-goal reinforcement learning: Challenging robotics environments and request for research. *arXiv preprint arXiv:1802.09464*, 2018.

RAPISARDA, C. Reinforcement learning for dexterity transfer between manipulators. 2019.

RIEDMILLER, M.; HAFNER, R.; LAMPE, T.; NEUNERT, M.; DEGRAVE, J.; WIELE, T. Van de; MNIH, V.; HEESS, N.; SPRINGENBERG, J. T. Learning by playing-solving sparse reward tasks from scratch. *arXiv preprint arXiv:1802.10567*, 2018.

ROBôCIN. *ROBÔCIN*. 2018. <http://www.cbrobotica.org/mostravirtual/interna.php?id=25531>.

ROBôCIN. *RobôCIn*. 2018. <https://robocin.github.io/>.

ROSENBLATT, F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, American Psychological Association, v. 65, n. 6, p. 386, 1958.

RUSSAKOVSKY, O.; DENG, J.; SU, H.; KRAUSE, J.; SATHEESH, S.; MA, S.; HUANG, Z.; KARPATHY, A.; KHOSLA, A.; BERNSTEIN, M. et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, Springer, v. 115, n. 3, p. 211–252, 2015.

RUSSELL, S. J.; NORVIG, P. *Artificial intelligence: a modern approach*. [S.l.]: Malaysia; Pearson Education Limited„ 2016.

SANTOS, G. C.; MERCES, R.; PIMENTEL, F.; SAPUCAIA, F.; LARANJEIRA, C.; SOUZA, J.; SIMõES, M.; FRIAS, D. Evoluindo o projeto de robôs para liga de futebol very small size soccer. In: . [S.l.: s.n.], 2015. p. 39–48.

SEHGAL, A.; LA, H.; LOUIS, S.; NGUYEN, H. Deep reinforcement learning using genetic algorithm for parameter optimization. In: IEEE. *2019 Third IEEE International Conference on Robotic Computing (IRC)*. [S.l.], 2019. p. 596–601.

SERMANET, P.; EIGEN, D.; ZHANG, X.; MATHIEU, M.; FERGUS, R.; LECUN, Y. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

SIEGWART, I. R. N. R. *Introduction to Autonomous Mobile Robots*. [S.l.]: The MIT Press, 2011.

SILVER, D.; HUANG, A.; MADDISON, C. J.; GUEZ, A.; SIFRE, L.; DRIESSCHE, G. van den; SCHRITTWIESER, J.; ANTONOGLOU, I.; PANNEERSHELVAM, V.; LANCTOT, M.; DIELEMAN, S.; GREWE, D.; NHAM, J.; KALCHBRENNER, N.; SUTSKEVER, I.; LILLICRAP, T.; LEACH, M.; KAVUKCUOGLU, K.; GRAEPEL, T.; HASSABIS, D. Mastering the game of go with deep neural networks and tree search. *Nature*, Springer Science and Business Media LLC, v. 529, n. 7587, p. 484–489, jan. 2016. Disponível em: <https://doi.org/10.1038/nature16961>.

SILVER, D.; HUBERT, T.; SCHRITTWIESER, J.; ANTONOGLOU, I.; LAI, M.; GUEZ, A.; LANCTOT, M.; SIFRE, L.; KUMARAN, D.; GRAEPEL, T. et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, American Association for the Advancement of Science, v. 362, n. 6419, p. 1140–1144, 2018.

SIRLAB. *Bullet Physics*. 2015. <https://github.com/bulletphysics/bullet3>.

SIRLAB. *VSS-SDK*. 2019. <https://github.com/VSS-SDK/>.

SUTTON, R. S.; BARTO, A. G. *Reinforcement learning: An introduction*. [S.l.]: MIT press, 2018.

SZEGEDY, C.; IOFFE, S.; VANHOUCKE, V.; ALEMI, A. A. Inception-v4, inception-resnet and the impact of residual connections on learning. In: *Thirty-First AAAI Conference on Artificial Intelligence*. [S.l.: s.n.], 2017.

UHLENBECK, G. E.; ORNSTEIN, L. S. On the theory of the brownian motion. *Physical review*, APS, v. 36, n. 5, p. 823, 1930.

V-REP. *Plataforma V-Rep Ambiente Simulado de Propósito Geral*. 2010. Disponível em: <http://www.coppeliarobotics.com/>.

V-REP, C. R. *Plataforma Gazebo Ambiente Simulado de Propósito Geral*. 2002. Disponível em: <http://gazebosim.org/>.

VEČERÍK, M.; HESTER, T.; SCHOLZ, J.; WANG, F.; PIETQUIN, O.; PIOT, B.; HEESS, N.; ROTHÖRL, T.; LAMPE, T.; RIEDMILLER, M. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817*, 2017.

VINYALS, O.; BABUSCHKIN, I.; CHUNG, J.; MATHIEU, M.; JADERBERG, M.; CZARNECKI, W. M.; DUDZIK, A.; HUANG, A.; GEORGIEV, P.; POWELL, R.; EWALDS, T.; HORGAN, D.; KROISS, M.; DANIHELKA, I.; AGAPIOU, J.; OH, J.; DALIBARD, V.; CHOI, D.; SIFRE, L.; SULSKY, Y.; VEZHNEVETS, S.; MOLLOY, J.; CAI, T.; BUDDEN, D.; PAINE, T.; GULCEHRE, C.; WANG, Z.; PFAFF, T.; POHLEN, T.; WU, Y.; YOGATAMA, D.; COHEN, J.; MCKINNEY, K.; SMITH, O.; SCHAUL, T.; LILLICRAP, T.; APPS, C.; KAVUKCUOGLU, K.; HASSABIS, D.; SILVER, D. *AlphaStar: Mastering the Real-Time Strategy Game StarCraft II*. 2019. <https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/>.

WANG, Z.; SCHAUL, T.; HESSEL, M.; HASSELT, H. V.; LANCTOT, M.; FREITAS, N. D. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.

YANG, J.; NGUYEN, M. N.; SAN, P. P.; LI, X. L.; KRISHNASWAMY, S. Deep convolutional neural networks on multichannel time series for human activity recognition. In: *Twenty-Fourth International Joint Conference on Artificial Intelligence*. [S.l.: s.n.], 2015.

ZHANG, X.; ZHOU, X.; LIN, M.; SUN, J. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2018. p. 6848–6856.

ZHENG, Y.; LIU, Q.; CHEN, E.; GE, Y.; ZHAO, J. L. Time series classification using multi-channels deep convolutional neural networks. In: SPRINGER. *International Conference on Web-Age Information Management*. [S.l.], 2014. p. 298–310.

ZIEBART, B. D.; MAAS, A.; BAGNELL, J. A.; DEY, A. K. Maximum entropy inverse reinforcement learning. figshare, 2008.