UNIVERSIDADE FEDERAL DE MINAS GERAIS
Programa de Pós-Graduação em Engenharia Elétrica
Mestrado em Controle Automação e Robótica

Wilson Salomão Félix Júnior

APLICAÇÃO DE APRENDIZADO POR REFORÇO EM NAVEGAÇÃO DE ROBÔS

Belo Horizonte
2022

Wilson Salomão Félix Júnior

APLICAÇÃO DE APRENDIZADO POR REFORÇO EM NAVEGAÇÃO DE ROBÔS

Dissertação submetida ao Programa de Pós-Graduação em Engenharia Elétrica da Escola de Engenharia da Universidade Federal de Minas Gerais, como parte dos requisitos para obtenção do título de Mestre em Engenharia Elétrica.

Orientador: Prof. Dr. Vinícius Mariano Gonçalves

Belo Horizonte
2022

Félix Júnior, Wilson Salomão.
F316a Aplicação de aprendizado por reforço em sistemas robóticos [recurso eletrônico] / Wilson Salomão Félix Júnior. - 2022.
1 recurso online (87 f. : il., color.) : pdf.

Orientador: Vinícius Mariano Gonçalves.

Dissertação (mestrado) - Universidade Federal de Minas Gerais, Escola de Engenharia.

Bibliografia: f. 77-87.
Exigências do sistema: Adobe Acrobat Reader.

1. Engenharia Elétrica - Teses. 2. Aprendizado Profundo – Teses. 3. Robótica – Teses. 4. Programação Dinâmica – Teses. I. Gonçalves, Vinícius Mariano. II. Universidade Federal de Minas Gerais. Escola de Engenharia. III. Título.

CDU: 621.3(043)

Ficha catalográfica elaborada pela Bibliotecária Letícia Alves Vieira - CRB-6/2337
Biblioteca Prof. Mário Werneck - Escola de Engenharia da UFMG

UNIVERSIDADE FEDERAL DE MINAS GERAIS
ESCOLA DE ENGENHARIA
COLEGIADO DO CURSO DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

FOLHA DE APROVAÇÃO

"Aplicação de Aprendizado Por Reforço Em Sistemas Robóticos"

WILSON SALOMÃO FÉLIX JÚNIOR

Dissertação de Mestrado defendida e aprovada, no dia 29 de abril de 2022, pela Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Minas Gerais constituída pelos seguintes professores:

Prof. Dr. Adriano Veloso (DCC (UFMG))

Prof. Dr. Armando Alves Neto ((UFMG))

Prof. Dr. Vinícius Mariano Gonçalves (DEE (UFMG)) - Orientador

Belo Horizonte, 29 de abril de 2022.

Documento assinado eletronicamente por **Vinicius Mariano Goncalves**, **Membro**, em 29/04/2022, às 16:37, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade deste documento pode ser conferida no site https://sei.ufmg.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **1399203** e o código CRC **37283BA0**.

Referência: Processo nº 23072.222811/2022-35 SEI nº 1399203

Criado por fab2013, versão 4 por fab2013 em 27/04/2022 10:25:28.

Agradecimentos

Primeiramente, gostaria de agradecer a Deus, por todas as bençãos que Ele coloca em minha vida, desde muito antes de eu decidir fazer esse mestrado, até agora, que estou concluindo essa etapa.

A primeira pessoa que eu gostaria de agradecer é a minha linda esposa Isabela, que além de suportar os momentos em que estive ausente devido aos estudos, me apoiou em diversos momentos durante esses dois anos. Além disso, agradeço muito pela prestatividade para a leitura e avaliação dos textos escritos, porque realmente não foi fácil para ela corrigir tantos erros. Outra prestatividade que agradeço muito foi de cuidar de tudo que precisávamos nos momentos mais críticos, quando próximo de provas e entregas de trabalhos. Agradeço pelo amor e carinho que foram muito importante durante todo esse período de pandemia, pois foi difícil ficar em casa durante todo o tempo, com medo do que possa acontecer com os familiares, e graças a Deus, todo mundo passou por essa fase mesmo que distantes, juntos.

Em segundo lugar, gostaria de agradecer meus pais pelo apoio e carinho dado durante esse período. Apesar do fato de estar longe deles durante praticamente todo o mestrado, eles entenderam e me apoiaram sempre que tive que priorizar passar um domingo estudando, ao invés de ir visitá-los. Agradeço demais por tudo que eles sempre fizeram para mim, pois foi graças a eles que estou completando mais essa etapa.

Gostaria também de agradecer meu orientador Vinícius por todo o auxílio dado durante o mestrado, pela paciência em me explicar repetidas vezes os detalhes da teoria, pelas discussões produtivas que tivemos durante esse período, para desvendar um pouco sobre esse mundo chamado aprendizado por reforço e pelo apoio em buscar uma direção para minha carreira. O professor Vinícius também me influenciou, de forma direta e indireta, a melhorar dois pontos que serão muito importantes para a sequência da minha vida: a escrita de textos de forma correta, e a habilidade em matemática. Agradeço também os outros professores que tive durante o mestrado, pois aprendi muito com eles em suas respectivas disciplinas.

Resumo

O estudo e a utilização de robôs para atender as necessidades do ser humano tem sido estudada de forma profunda desde o último século. Umas das principais pesquisas é como realizar o movimento de robôs de forma autônoma, segura e eficiente, fazendo com que eles realizem tarefas que precisam de locomoção. No entanto, é comum que seja difícil construir ou seguir caminhos que respeitem as diversas restrições do ambiente, como por exemplo, presença de obstáculos, restrições no movimento ou limitações no sensoriamento do robô. Recentemente, uma das áreas que tem ganhado notoriedade dentro da comunidade científica é o aprendizado por reforço profundo, que reúne os conceitos de aprendizagem por reforço, uma sub-área da aprendizagem de máquina, com as últimas inovações produzidas pelo aprendizado profundo, outro campo muito estudado com diversos resultados expressivos obtidos. Ainda que inicialmente as primeiras aplicações tenham sido no setor de jogos virtuais, vários pesquisadores vem propondo aplicar essas técnicas em sistemas robóticos, para tarefas de manipulação, locomoção, entre várias outras. Neste sentido, esta dissertação irá apresentar algumas ferramentas e algoritmos propostas na área de aprendizagem por reforço profundo, que farão com um robô móvel seja capaz de aprender a se deslocar para um alvo em um cenário com obstáculos. Além disso, este trabalho irá propor um algoritmo que realiza o aprendizado do melhor caminho de acordo com a tarefa de forma contínua, melhorando o caminho percorrido a medida que o robô conclui as tarefas.

Palavras-chave: Aprendizado por reforço. Aprendizado profundo. Controle de movimento. Programação Dinâmica. Robótica.

Abstract

The study and the usage of robots to assist humanity has been studied deeply since the past century. One of the main researches is to perform the robot motion autonomously, safely and efficiently, in such a way that they perform tasks that may need locomotion. However, it is common that the desired path be complicated to build or follow, while some constraints of the environment have to be considered, such as, obstacle avoidance, moviment constraints or limitation on robot sensors. Recently, one of the areas that has achieved notoriety in the research community is deep reinforcement learning, which assembles concepts of reinforcement learning, one sub-area of machine learning, with the lastest breakthroughs of deep learning, another research field with several expressive results. Even considering that the first applications were in video games, many researchers have been proposing to apply these techniques in robot systems, for many tasks, for example, manipulation and locomotion. In this way, this dissertation will present some tools and algorithms recently proposed in deep reinforcement learning, which will make the robot capable of learning to move to a target in a scenario with obstacles. Besides that, this work will propose an algorithm that performs the learning of the best path according to the task continuously, improving the path travelled as the robot finalizes the tasks.

Keywords: Reinforcement Learning. Deep Learning. Motion Control. Dynamic Programming. Robotics.

Lista de Figuras

Sumário

Capítulo 1

Introdução

O presente trabalho demonstrará como utilizar aprendizado por reforço (AR) (*reinforcement learning - RL*) para controlar o movimento de robôs. A área de aprendizado por reforço é uma das que mais evoluem nos últimos anos. Ela é uma das ramificações de aprendizado de máquina, junto com aprendizado supervisionado e o não-supervisionado. Existem diferentes pesquisas para o controle de movimento de robôs, desde técnicas clássicas, como campos vetoriais [Rezende et al., 2021], controle por pseudo-inversa da jacobiana [Whitney, 1969], métodos de amostragem (RRT [Kuffner and RRT-Connect, 2000], PRM [Kavraki et al., 1996]), até técnicas mais modernas, como controle por otimização [Júnior and Gonçalves, 2018].

O aprendizado por reforço é uma área que busca promover um aprendizado à um sistema computacional através de estímulos que esse sistema recebe do ambiente em que ele está inserido. Ele difere dos outros dois tipos de aprendizado, aprendizado supervisionado e não supervisionado, visto que o primeiro consiste em treinar um sistema computacional com dados de entrada e com a saída esperada para aqueles dados de entrada, como por exemplo, a capacidade de identificar se uma pessoa está com gripe com base nos sintomas. Já o aprendizado não-supervisionado objetiva analisar e construir padrões a partir dos dados de entrada, como por exemplo, a partir de diversos sensores de uma máquina. Ele então tenta identificar se a mesma possui alguma falha. O aprendizado por reforço visa alterar o comportamento de uma determinada aplicação através dos estímulos recebidos desta. Esses estímulos são chamados de *recompensa*, e o sistema computacional que recebe os estímulos é chamado de *agente*, enquanto o resto do "mundo" é chamado de *ambiente* (a seção 3.2 irá apresentar definições formais de agente e ambiente). Vale ressaltar que, geralmente o ambiente é definido por um *modelo*. O agente tem uma *política*, que indica quais atuações ele deve fazer no ambiente com base nas informações recebidas do mesmo. A política é essencialmente o controlador em teoria de controle.

Existem diferentes ramificações em aprendizagem por reforço. Uma dessas ramificações estuda técnicas que aprendem uma política sem precisar do modelo do ambiente. Essas técnicas são chamadas *model-free* e geralmente são muito consideradas pelo baixo custo computacional para realizar a aprendizagem. Outra ramificação é o estudo de técnicas chamadas *model-based*,

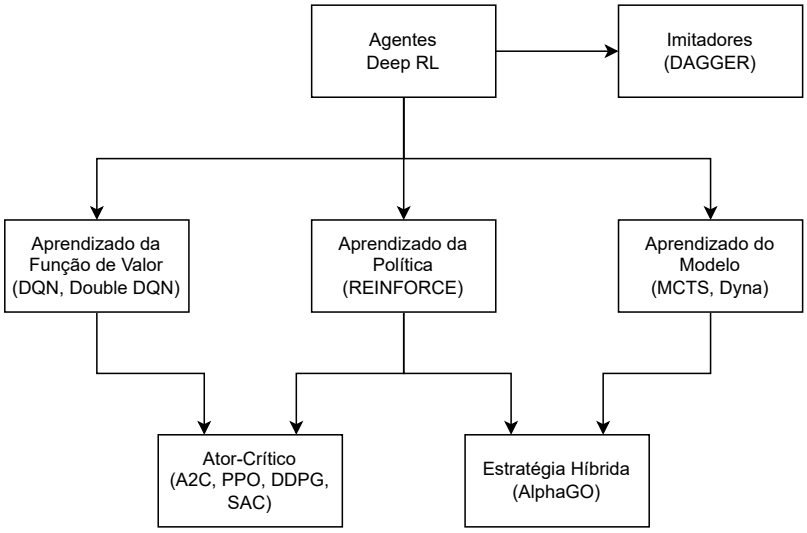

Figura 1.1: Taxonomia de aprendizado por reforço. Exemplos de algoritmos são apresentados entre parênteses. Imagem adaptada de https://pylessons.com/Beyond-DQN.

que são aquelas que visam aprender o modelo do ambiente, e utilizar esse modelo para encontrar a política que levará o agente a cumprir tarefas. Para que o agente possa encontrar uma política que seja satisfatória, ele precisa muitas vezes *explorar* o ambiente. Existe outra ramificação, dedicada a estudar diferentes técnicas de exploração, que serão apresentadas de forma abrangente no capítulo 2. Vale mencionar que existem muitas outras ramificações na área, que não serão exploradas no decorrer desta dissertação, mas que devem ser mencionadas, como por exemplo, o estudo do aprendizado por reforço *offline*, aprendizado por reforço inverso, transferência de aprendizado por reforço e meta-aprendizado por reforço. O capítulo 2 irá apresentar alguns dos estudos vistos nessa área. De forma resumida, a figura 1.1 apresenta as diferentes divisões de aprendizagem por reforço.

1.1 Motivação e Justificativa

A área da robótica possui muitas soluções para diferentes tipos de problemas [Spong et al., 2005], [Siciliano et al., 2010], [Choset et al., 2005]. Geralmente, técnicas clássicas de navegação de robôs têm limitações, como a dificuldade de lidar com os chamados *pontos de mínimos locais*. Esse é um dos principais problemas que é discutido em [Júnior, 2018], que sugere a utilização de otimização para realizar o controle de movimento do robô. Esse uso pode até lidar com eventuais mudanças do ambiente, por se tratar de uma estratégia local, mas quando o robô alcança um mínimo local, comumente é necessário utilizar uma estratégia de recuperação, que no trabalho foi sugerido a utilização de técnicas de amostragem como RRT e PRM. A grande desvantagem dessa técnica de amostragem é que elas geralmente requerem um conhecimento prévio do ambiente, o que inviabiliza utilizar essa solução em ambiente genéricos.

Portanto, não é comum ver técnicas clássicas que são realmente capazes de resolver tarefas

complexas com o mínimo de informação possível do ambiente. A solução muitas vezes assume um ambiente pré-processado ou um ambiente sem muitas dificuldades. Outras vezes, a solução encontrada é específica para uma determinada tarefa, sendo necessária uma outra solução para outro tipo de tarefa. Isso mostra que muitas vezes a solução não é generalizável, quando uma mesma proposta pode ser aplicada em diferentes ambientes e tarefas. Portanto, elaborar soluções gerais tem sido um desafio dentro da área.

Por outro lado, técnicas de inteligência computacional têm por característica a capacidade de modificar a solução com base nas entradas que são recebidas [James et al., 2013]. Como exemplo, através do uso de redes neurais, tornou-se possível fazer com que o computador realize um reconhecimento de imagens com uma alta taxa de acerto. Dentre as áreas da inteligência computacional, uma das que se destaca na aplicabilidade na robótica é a aprendizagem por reforço. Essa aprendizagem tem como característica fazer com que o robô aprenda na tentativa e erro, ganhando experiencia a cada tentativa e melhorando sua estratégia de controle.

Essa área obteve muito sucesso em fazer agentes capazes de jogar jogos de Atari, e nos últimos anos, alguns foram além, jogando jogos cada vez mais complexos [Shao et al., 2019]. Diversos trabalhos foram propostos na área, desde a apresentação do Deep Q-Network [Mnih et al., 2015], até os algoritmos mais novos como AlphaGo [Silver et al., 2017]. Portanto, essa área se apresenta promissora como solução para realizar tarefas difíceis em ambientes desafiadores na robótica.

1.2 Objetivos do Projeto

O principal objetivo desta dissertação é apresentar o problema de navegação de robôs, sob a ótica de aprendizado por reforço, de forma que o sistema inteligente possa levar o robô para uma posição e orientação alvo em um cenário com restrições. Essas restrições podem ser restrições físicas, como obstáculos, ou restrições no movimento, como a não-holonomia. O robô não terá qualquer conhecimento prévio do ambiente e o aprendizado do sistema computacional ocorrerá por meio do deslocamento do robô, através dos estímulos recebidos por ele.

Este trabalho apresentará duas soluções para esse problema, sendo uma para robôs sem a restrição de não-holonomia e a outra para robôs com a restrição de não-holonomia. As soluções serão apresentadas em uma ordem histórica, de acordo com o momento em que cada abordagem foi estudada. Esta dissertação também irá destacar as potenciais vantagens de cada solução, sob o ponto de vista teórico, visto que cada solução é aplicada em problemas distintos.

Em ambas abordagens, será notório que o robô será capaz de aprender a se deslocar no ambiente, mas somente a segunda abordagem tem o caráter de generalização descrito anteriormente. Também serão descritas as potenciais razões para esses comentários, além de apresentar resultados que confirmem tais proposições.

Como dito anteriormente e como será visto na seção de referências bibliográficas, essa área está em constante evolução e como valiosos trabalhos continuam sendo apresentados, se faz

necessário limitar o escopo a uma sub-área de aprendizado por reforço. Desta forma, existem muitas possibilidades de continuação dos estudos desta área. A seção de trabalhos futuros irá apresentar algumas delas.

1.3 Trabalhos Publicados

Como parte do processo do Mestrado, um dos trabalhos que serão discutidos nesta dissertação foi apresentado no Simpósio Brasileiro de Automação Inteligente - SBAi de 2021. Este trabalho consiste no segundo objetivo dessa dissertação descrito na seção anterior, que é a proposta de um agente que realiza seu aprendizado através da *iteração de valor por meio da álgebra max-plus*. Todos esses termos serão melhores explicados adiante, no capítulo de Metologia. O trabalho proposto se trata de uma pesquisa introdutória, podendo ser, em seguida, amplamente explorada.

1.4 Estrutura da Dissertação

O capítulo 1 contém uma breve introdução sobre a mesma, identificando as principais motivações, justificativas e objetivos da pesquisa realizada.

O capítulo 2 irá descrever um contexto histórico do aprendizado por reforço e programação dinâmica, desde os primeiros trabalhos apresentados na área até trabalhos clássicos que se tornaram marcos para os que estudam a área. Além disso, também será exposto diversos trabalhos apresentados recentemente sobre as várias sub-áreas de *aprendizado por reforço profundo*. Por fim, este capítulo se encerrará com aplicações dessas técnicas em robôs, sejam para tarefas de locomoção do robô ou tarefas de manipulação de objetos.

O capítulo 3 irá apresentar os conceitos fundamentais da área de aprendizado profundo, como Processo de Decisão de Markov, função de valor, programação dinâmica, políticas, entre vários outros conceitos.

O capítulo 4 irá apresentar duas formas diferentes de aplicar aprendizado por reforço em tarefas de navegação de robôs, a primeira para robôs sem a restrição de não-holonomia, enquanto a segunda será aplicada em robôs com essa restrição.

O capítulo 5 irá apresentar diferentes formas de validação das abordagens apresentadas no capítulo 4 e seus respectivos resultados obtidos a partir de simulações no ambiente de simulação Matlab. Além disso, este capítulo irá apresentar uma discussão a respeito das variadas simulações.

O capítulo 6 irá concluir a dissertação, com as observações sobre o trabalho apresentado e propostas de trabalhos futuros.

Capítulo 2

Revisão Bibliográfica

Neste capítulo será mostrado uma revisão bibliográfica sobre a teoria de aprendizagem por reforço e suas aplicações e potencialidades em robótica.

2.1 Início da aprendizagem por reforço

O livro [Sutton and Barto, 1998] traz uma narrativa de como surgiram as técnicas de aprendizagem por reforço. Segundo os autores, no início dessa abordagem existiam duas vertentes principais: aprendizado via tentativa e erro e aprendizado como um problema de controle ótimo. Essas vertentes começaram independentes e então se inter-relacionaram em torno de uma terceira vertente, que utilizava métodos de diferença temporal, no final dos anos 1980. Como descrito pelos autores, diversos pesquisadores influenciaram no desenvolvimento da área, dentre os quais serão destacados alguns.

Iniciando pela vertente de controle ótimo, um dos primeiros a tentar desenvolver controladores ótimos foi Richard Bellamn, na metade do século XX (na metade da década de 1950). Ele criou uma abordagem que utilizava a dinâmica do sistema e uma "função de retorno ótimo", para definir uma equação, que mais tarde ficou conhecida como *equação de Bellman*. Além disso, Bellman criou uma classe de métodos para resolver problemas de controle ótimos resolvendo essa equação, que acabou sendo conhecido como *programação dinâmica* [Bellman, 1957]. Essa classe de métodos é considerada como a única maneira factível de resolver problemas de controle ótimo estocástico de forma genérica, segundo [Sutton and Barto, 1998]. Apesar disso, ela tem uma desvantagem chamada "maldição da dimensionalidade", cunhada por Bellman, onde o custo computacional cresce exponencialmente com o número de estados. Os autores mencionam que apesar de programação dinâmica ter sido muito utilizada, sua relação com aprendizagem demorou para ser reconhecida. Alguns dos trabalhos que fizeram essa relação de forma mais evidente foram [Witten, 1977a], [Werbos, 1987], [Watkins, 1989], sendo esse último um trabalho utilizando aprendizagem por reforço e com o formalismo de MDPs muito adotado em pesquisas posteriores. Um outro trabalho de bastante relevância na época

foi [Bertsekas and Tsitsiklis, 1989], que criaram o termo “programação neurodinâmica” para a combinação de programação dinâmica e redes neurais artificiais.

A vertente de aprendizado via tentativa e erro, segundo [Sutton and Barto, 1998], começa quando Edward Thorndike cria a “Lei do Efeito”.

A implementação do aprendizado via tentativa e erro em um computador apareceu na mesma época dos primeiros pensamentos sobre inteligência artificial. Alguns dos trabalhos de destaque nessa vertente foram de dois autores em especial: Minsky com [Minsky, 1954] e [Minsky, 1961]; e Andreae com seus trabalhos [Andreae, 1963], [Andreae and Cashin, 1969], [Andreae, 1969], [Andreae, 1977]. Muitos outros trabalhos foram propostos para o aprendizagem via tentativa em erro, mas é interessante notar que durante as décadas de 1960 e 1970, poucos foram os trabalhos focados em aprendizagem por reforço, pois muitos pesquisadores estavam focados no aprendizado supervisionado. Até houve uma certa confusão na época, pois considerava-se que o aprendizado supervisionado também como tentativa e erro, o que foi corrigido mais a frente.

2.2 Aprendizagem por reforço profundo

Os trabalhos de aprendizagem por reforço profundo se iniciaram no artigo submetido por [Mnih et al., 2015], onde o algoritmo DQN (Deep Q-Network) foi proposto. Esse foi um grande *breakthrough* na área, uma vez que até então, ainda não existiam grandes casos de sucesso em aplicar redes neurais artificiais profundas em problemas de aprendizagem por reforço. Na sequência [Van Hasselt et al., 2016] propôs uma alteração no DQN que gerou o algoritmo DDQN (Double Deep Q-Network), que trouxe maior estabilidade ao DQN. Então, [Wang et al., 2016b] propôs uma nova abordagem para calcular a função de valor Q, separando-a na função de valor V e na função de vantagem A, que resultou no algoritmo D3QN (Dueling Double Deep Q-Network). Para utilizar o DQN em ambientes com ações de controle contínuas, [Lillicrap et al., 2015] propôs uma variação do DPG ([Silver et al., 2014]) que utiliza os mesmos truques do DQN para resultar no DDPG (Deep Deterministic Policy Gradient). Um outro exemplo de Q-Learning para ambientes contínuos é o NAF [Gu et al., 2016b].

Uma outra ala da aprendizagem por reforço profundo estuda aprender a política de forma direta e, nesse contexto, [Schulman et al., 2017b] e [Schulman et al., 2015a] foram dois trabalhos que se destacaram em apresentar algoritmos capazes de aprender a política de forma direta. Outro trabalho relevante é o [Levine and Koltun, 2013], que apresenta um gradiente de política com *importance sampling*. Já os trabalhos que consideram a arquitetura ator-crítico principais são [Mnih et al., 2016], que aplica essa arquitetura em formato de computação paralela, [Schulman et al., 2015b], que apresenta uma abordagem de cálculo da função de vantagem, e [Gu et al., 2016a], que descreve um ator-crítico com a função Q.

Para a ramificação da área que estuda a aprendizagem do modelo do ambiente, vários trabalhos foram propostos recentemente, como por exemplo [Tassa et al., 2012], [Levine and Abbeel,

2014]. Um dos principais artigos submetidos que utilizam redes neurais profundas para aprender a dinâmica do ambiente é [Nagabandi et al., 2018]. Na mesma direção, outros algoritmos foram sendo propostos em utilizar redes neurais artificiais para aprender a dinâmica do ambiente. Dentre eles são destacados: probabilistic ensembles with trajectory sampling (PETS) [Chua et al., 2018] e planning with deep dynamics models (PDDM) [Nagabandi et al., 2020]. Outros trabalhos recentes que utilizam deste conceito são [Feinberg et al., 2018] e [Buckman et al., 2018]. Existem trabalhos também que realizam a aprendizagem da dinâmica do ambiente a partir de imagens, através de variáveis latentes, como os descritos por [Watter et al., 2015], [Zhang et al., 2019], ou através de observações do espaço, como em [Finn and Levine, 2017] e [Ebert et al., 2017]. Um outra possibilidade é aprender a dinâmica e a política de forma conjunta. Para isso existem basicamente duas possibilidades. A primeira é utilizar técnicas que não consideram o modelo em conjunto com técnicas que aprendem o modelo e os principais trabalhos nessa área são [Parmas et al., 2018] e [Janner et al., 2019]. A segunda possibilidade é utilizar políticas mais simples que redes neurais profundas, como descrito nos artigos [Levine and Abbeel, 2014], [Levine et al., 2016], [Hinton et al., 2015], [Parisotto et al., 2015], [Ghosh et al., 2017], [Rusu et al., 2015], [Teh et al., 2017].

Um dos aspectos mais importantes nesse paradigma é a exploração do agente no ambiente. A seção 3.5 irá apresentar maiores detalhes sobre esse aspecto, mas seu estudo também proporcionou importantes trabalhos recentemente. Dentro do contexto clássico de aprendizagem por reforço, que pode ser encontrado em [Sutton and Barto, 2018], duas abordagens se destacam para realizar a exploração: exploração otimista e a exploração por meio da amostragem de Thompson. Uma outra abordagem bastante considerada é a exploração por ganho de informação [Russo and Van Roy, 2014]. Atualmente, diferentes autores têm buscado aplicar essas abordagens em aprendizado profundo, os quais são citados os seguintes trabalhos. Para a exploração otimista, alguns dos principais trabalhos são [Bellemare et al., 2016], [Tang et al., 2017], [Fu et al., 2017a] e [Burda et al., 2018]. Para exploração utilizando amostragem de Thompson, um dos artigos é [Osband et al., 2016]. E por fim, para a exploração que utiliza ganho de informação, foram propostos [Houthooft et al., 2016], [Stadie et al., 2015] e [Schmidhuber, 2010]. Uma outra maneira interessante de considerar a exploração é para ambientes que não retornam recompensas, onde o objetivo é aprender diferentes habilidades através da exploração do cenário. Alguns dos trabalhos nessa área são [Nair et al., 2018], [Pong et al., 2019], [Lee et al., 2019], [Hazan et al., 2019], [Eysenbach et al., 2018], [Gregor et al., 2016]. Uma das formas de estruturar a exploração de um agente através do controle da entropia da sua política. Essa é outra área muito estudada em aprendizagem por reforço, que teve como principais trabalhos [Levine, 2018], [Ziebart et al., 2010], [Haarnoja et al., 2018d], [Haarnoja et al., 2017], [Haarnoja et al., 2018b], [Haarnoja et al., 2018a], [Haarnoja et al., 2018c], [Schulman et al., 2017a], [Nachum et al., 2017].

Existe uma sub-área que estuda o aprendizagem por reforço *offline*, visando realizar o aprendizado a partir de amostras de interações do algoritmo com o ambiente, independentemente se as

amostras foram geradas pelo algoritmo, ou por qualquer outro mecanismo, além de possibilitar o uso de experiências muito antigas, que para a aprendizagem *online* geralmente são descartadas. Alguns dos mais importantes artigos nesse assunto são [Levine et al., 2020], [Kumar et al., 2019], [Jiang and Li, 2016], [Wu et al., 2019], [Peng et al., 2019], [Kostrikov et al., 2021], [Yu et al., 2020], [Kidambi et al., 2020], [Janner et al., 2021], [Kalashnikov et al., 2021], [Chebotar et al., 2021], [Kahn et al., 2021], [Kumar et al., 2021].

Por fim, ainda existem outras alas da pesquisa de aprendizagem por reforço. Uma delas por exemplo estuda como aprender a recompensa a partir de observações de um especialista. Essa área é chamada de aprendizagem por reforço inversa, e alguns dos trabalhos propostos são [Finn et al., 2016], [Ho and Ermon, 2016], [Fu et al., 2017b], [Silva et al., 2019]. Uma segunda ala estuda como utilizar o aprendizado obtido em uma tarefa como base para aprender outra tarefa. Duas possibilidades geralmente são estudadas: a transferência direta, onde o treinamento é realizado em uma tarefa, e então ele é usado como base para iniciar o treinamento para outra tarefa, ou a transferência multi-tarefa, onde várias tarefas são treinadas em conjunto e, então esse aprendizado é utilizado como base para aprender outra tarefa. Essa área é chamada de aprendizagem por transferência e alguns dos trabalhos propostos são [Tzeng et al., 2014], [Andreas et al., 2017], [Peng et al., 2018], [Parisotto et al., 2015], [Ghosh et al., 2017], [Barreto et al., 2017]. Uma terceira ala de pesquisa estuda como estruturar o aprendizado de uma política de forma a aprender como aprender a realizar tarefas. Essa área é chamada de meta-aprendizagem por reforço e alguns dos trabalhos propostos são [Wang et al., 2016a], [Finn et al., 2017], [Gupta et al., 2018], [Houthooft et al., 2018], [Zhao et al., 2020], [Belkhale et al., 2021].

2.3 Aplicações em robótica de técnicas de aprendizado

Existem várias técnicas clássicas para resolver o problema de navegação de robôs, como campos vetoriais [Rezende et al., 2021], controle por pseudo-inversa da jacobiana [Whitney, 1969], métodos de amostragem (RRT [Kuffner and RRT-Connect, 2000], PRM [Kavraki et al., 1996]), até técnicas mais modernas, como controle por otimização [Júnior and Gonçalves, 2018]. Como foi visto na introdução, essas técnicas geralmente têm limitações devido aos mínimos locais, ou para o caso de métodos de amostragem, são técnicas inviáveis em sistemas de tempo real, além de necessitarem do mapa do ambiente antes de serem executadas.

Com relação as aplicações em robótica de aprendizado por reforço, o artigo proposto por [Kober et al., 2013] lista diversos trabalhos nesse contexto. Para tarefas de manipulação, alguns dos trabalhos apresentados foram [Kaspar and Bock,], [Matas et al., 2018], [Rajeswaran et al., 2017], [Kalakrishnan et al., 2011]. Já os seguintes artigos são amostras de pesquisa realizadas em tarefas de locomoção [Zhu et al., 2017], [Bassani et al., 2020], [Qin et al., 2019], [Chaffre et al., 2020], [Pedersen, 2019], [Traoré et al., 2019]. Para outros tipos de tarefas, recentemente foram propostos [Witman et al., 2019], [Ding et al., 2020], [Nachum et al., 2019].

Esta dissertação utilizou como base os artigos [Tai et al., 2017] e [Jesus et al., 2019] para

propor o ambiente descrito na seção 4.2. Outros artigos estudados durante o Mestrado foram [Surmann et al., 2020], [Liu et al., 2020], [Ibarz et al., 2021].

2.4 Resumo do capítulo

Este capítulo apresentou uma revisão das principais referências bibliográficas na área de aprendizado por reforço. Ele iniciou destacando um contexto histórico do início dessa área, mostrando os diferentes autores que foram pioneiros na apresentação e uso dessas técnicas. O capítulo seguiu apresentando diversas referências para as muitas ramificações da área de aprendizado por reforço profundo, desde referências para estimar a função de valor e a política, até referências de meta-aprendizado por reforço, um tópico estudado mais recentemente. Por fim, o capítulo apresentou diferentes aplicações na robótica, desde tarefas de manipulação até tarefas de locomoção.

Capítulo 3

Fundamentos

Neste capítulo, serão apresentados os diversos conceitos nas áreas de aprendizagem por reforço, necessários para esta dissertação.

3.1 Agente e ambiente

A primeira terminologia a ser definida é o conceito de *agente* e *ambiente*. Em geral, os elementos que compõem uma malha de controle são os sensores, os controladores, os atuadores e o sistema a ser controlado [Dorf and Bishop, 2001]. Considerando no contexto de robótica, os mesmos elementos se mantêm, mas o sistema a ser controlado para realizar tarefas é o robô.

Essa terminologia não se aplica no contexto de aprendizagem por reforço. Nesse cenário, é comum na área considerar os termos *agente* e *ambiente*. Segundo [Sutton and Barto, 2018, p. 47] "quem aprende e faz a tomada de decisão é chamado de agente. Aquilo que interage com ele, compreendendo tudo que está fora do agente, é chamado de ambiente". O agente interage com o ambiente realizando ações no ambiente. Em contrapartida, o ambiente retorna o estado dele (ou observação) e um sinal de recompensa que tem o objetivo de qualificar a ação realizada pelo agente, com base no estado do ambiente. A figura 3.1 mostra essa interação entre o agente e o ambiente.

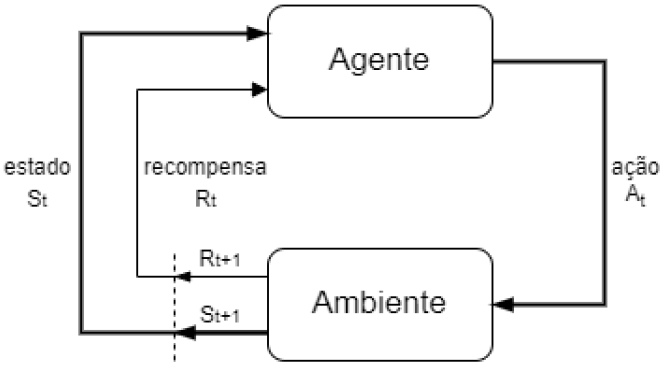

Figura 3.1: Diagrama de interface entre o agente e o ambiente. Imagem adaptada de [Sutton and Barto, 2018].

Considerando essas definições, o agente será *treinado* para *aprender* a interagir com o ambiente, de acordo com os objetivos da tarefa. Para entender como será feita essa interação, é necessário entender um processo chamado *Processo de Decisão de Markov* (*Markov Decision Process - MDP*).

3.2 Processo de Decisão de Markov

O *Processo de Decisão de Markov* (*Markov Decision Process - MDP*) é um processo estocástico utilizado em problemas de decisões sequenciais. De forma geral, os problemas de aprendizagem por reforço são formulados como um MDP. Existem diferentes variações de como formular um MDP, mas adaptando de [Sutton and Barto, 2018], ele considera a seguinte tupla:

$$\mathcal{M} = \{\mathcal{S}, \mathcal{A}, p(s'|s,a), r(s,a), \gamma\} \tag{3.1}$$

onde:

- $\mathcal{S}$ é conjunto de estados s do ambiente.
- $\mathcal{A}$ é o conjunto de ações a possíveis.
- $p(s'|s,a)$ é a função de transição de estados, sendo considerada estocástica, ou seja, probabilidade de ir para o próximo estado s', dado o estado s e a ação a atuais.
- $r : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ é a recompensa obtida após a transição.
- $\gamma \in [0, 1]$ é o *fator de desconto*.

É esperado que todos os estados de um MDP tenham a *propriedade de Markov*. A propriedade de Markov de um estado é matematicamente escrita como:

$$p(s_{k+1}|s_k, a_k) = p(s_{k+1}|s_1, s_2, ..., s_k, a_k) \tag{3.2}$$

ou seja, a definição do estado futuro independe dos estados passados, dependendo apenas do estado presente e ação presente (adaptado de [Sutton and Barto, 2018]).

Em um primeiro momento, o foco será nos casos em que os espaços $\mathcal{S}$ e $\mathcal{A}$ são discretos e finitos, com poucos elementos. Portanto, é possível construir tabelas para obter algumas garantias teóricas. A seção 3.9 irá estender os métodos para casos mais genéricos, através das funções de aproximação.

3.2.1 Processo de Decisão de Markov Parcialmente Observável

Existe uma variação desse processo chamada de *Processo de Decisão de Markov Parcialmente Observável* (*Partially Observed Markov Decision Process - POMDP*) que considera que o

agente não tem acesso ao estado s do ambiente, e sim uma observação o dentro de um espaço de observações $\mathcal{O}$ e com uma função de probabilidade de emissão $\epsilon(o|s)$. Portanto, a POMDP considera a tupla $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{O}, p(s'|s,a), r(s,a), \gamma, \epsilon(o|s)\}$. Geralmente, observações não são markovianas. A seção 4.2 irá apresentar um ambiente que retorna observações e não estados.

3.2.2 Objetivo da aprendizagem por reforço

Como descrito em [Sutton and Barto, 2018], os objetivos de um agente são codificados em um sinal chamado *recompensa*. A recompensa é um número real obtido a cada iteração pelo agente, e é possível definir informalmente que o objetivo de um agente é maximizar a recompensa total recebida pelo agente. É importante ressaltar que, a recompensa simboliza o objetivo do agente e não como atingi-lo, e isso muitas vezes é um ponto de confusão no momento de propor essa função.

A definição formal de objetivo em AR vem de [Sutton and Barto, 2018], e é chamada de *retorno*. Formalmente, o retorno, denotado por $G(s_0, a_0, s_1, a_1, ..., s_T, a_T)$, pode ser definido como a soma descontada das recompensas a partir do tempo k:

$$G(s_0, a_0, s_1, a_1, ..., s_T, a_T) \triangleq \sum_{k=0}^{T} \gamma^k r_k \tag{3.3}$$

onde $s_0, s_1, ..., s_T$ e $a_0, a_1, ..., a_T$ são os estados e as ações dos instantes 0, 1 até T e T é a ultima iteração.

De forma geral, existem dois tipos de tarefas em aprendizagem por reforço: as *tarefas episódicas* e as *tarefas contínuas*. As tarefas episódicas são aquelas que têm um estado final s_T, que finaliza a sequência de estados do ambiente, e geralmente é acompanhado de uma função que reinicia o ambiente a partir de um estado inicial s_0. Nesse caso, T é o instante de terminação da tarefa, e é definido como uma variável aleatória que se alterará dependendo do episódio. Já as tarefas contínuas não definem um estado terminal, e assim a interação agente-ambiente não se reinicia em nenhum estado, mas continua de forma constante. Então, T seria infinito. Nesta dissertação, serão considerados ambos os tipos de tarefas. O sistema da seção 4.2 consiste de uma tarefa episódica, no entanto, o sistema da seção 4.1 consiste de uma tarefa contínua.

O valor γ representa o conceito de *desconto*. Como descrito em [Sutton and Barto, 2018] esse parâmetro define o efeito no presente de recompensas recebidas no futuro. Quando $\gamma \approx 0$, então o agente se importará somente com as recompensas mais imediatas. Por outro lado, quando $\gamma \approx 1$, o agente se importará mais para as recompensas futuras no presente.

É possível, além de maximizar a recompensa recebida pelo agente, também maximizar um segundo objetivo: a *entropia* das ações geradas pelo agente, o que auxilia na exploração, como será visto na seção 3.5. O trabalho de [Ziebart et al., 2010] detalha os aspectos desse arcabouço, assim como a seção 4.2.1 irá apresentar um agente que se utiliza dessa teoria para realizar o

aprendizado. O conceito de entropia será apresentado na seção 3.3.

3.3 Políticas e Função de Valor

Utilizando a definição de [Sutton and Barto, 2018], a *política* é uma função que define qual ação o agente tomará dependendo do estado que o ambiente está. Existem dois tipos de políticas: as *políticas estocásticas* e as *políticas determinísticas*.

Uma **política estocástica** é o mapeamento $\pi(a|s) : \mathcal{S} \longmapsto P(\mathcal{A})$, em que $P(\mathcal{A})$ é o espaço de distribuições de probabilidade das ações possíveis no espaço $\mathcal{A}$. Quando o espaço $\mathcal{A}$ é discreto, esse mapeamento é direto, amostrando uma ação de acordo com a distribuição de probabilidade definida pela política. Porém, quando o espaço $\mathcal{A}$ é contínuo, deve-se utilizar alguma distribuição contínua. Geralmente, se utiliza a distribuição gaussiana.

Uma **política determinística** é o mapeamento $\mu(s) : \mathcal{S} \longmapsto \mathcal{A}$. Nesse caso, o mapeamento é feito de forma direta seja no caso contínuo, seja no caso discreto. Como será visto na seção 3.5, aprender uma política determinística pode resultar em mínimos locais. Esses mínimos locais podem ser evitados se utilizar uma política exploratória, que é, geralmente, estocástica. Porém, no decorrer desta dissertação, serão mostradas ferramentas capazes de solucionar esse problema. Por outro lado, a política ótima, como será explicada na seção 3.3.2 é uma política determinística.

É possível obter a política $\pi(a|s)$ em função de $\mu(s)$, como mostrado na equação:

$$\pi(a|s) = \begin{cases} 1, \text{ se } a = \mu(s) \\ 0, \text{ caso contrário} \end{cases} \tag{3.4}$$

e, como será visto mais adiante no texto, muitos conceitos serão apresentados considerando a política $\pi(a|s)$, e somente quando for necessário, $\mu(s)$ será descrito apropriadamente.

Em se tratando de políticas estocásticas, existe um conceito muito importante em aprendizagem por reforço chamado de *entropia*. Esse conceito é utilizado em diferentes áreas da ciência, mas no contexto de AR, a entropia $\mathcal{H}$ de uma política π em um estado s é dada por:

$$\mathcal{H}(\pi(a|s)) = -\sum_{a \in \mathcal{A}} \pi(a|s) \log \pi(a|s). \tag{3.5}$$

Logicamente, não há necessidade de considerar entropias para política determinísticas. De toda forma, ela é especialmente importante para garantir a exploração do ambiente, como será visto nas seções 3.5 e 4.2.1.

O objetivo de um agente é *aprender* uma *política* que resulte no maior retorno possível. Como a política é um mapeamento de estados em ações, devem existir estados e ações que resultem em mais recompensas do que outros. Para qualificar os estados e as ações que resultem em maiores retornos, dois conceitos muito importantes foram criados: a *função de valor de um*

estado dada uma política π, $V^{\pi}(s)$ e a *função de valor do par estado-ação* dada uma política π, $Q^{\pi}(s,a)$.

A função $V^{\pi}(s)$ é definida como a esperança matemática do *retorno* obtido utilizando uma política π partindo do estado inicial $s_0 = s$:

$$V^{\pi}(s) \triangleq \mathbb{E}_{\substack{s_{k+1}\sim p(\cdot|s_k,a_k) \\ a_k\sim\pi(\cdot|s_k)}}[G(s_0,a_0,...,s_T,a_T)|s_0 = s] \tag{3.6}$$

para todo $s \in \mathcal{S}$. Note que o valor de um estado terminal, se existir, é sempre zero.

Analogamente, a função $Q^{\pi}(s,a)$ é definido como a esperança matemática do *retorno* obtido utilizando a política π após tomar a ação inicial $a_0 = a$ no estado inicial $s_0 = s$:

$$Q^{\pi}(s,a) \triangleq \mathbb{E}_{\substack{s_{k+1}\sim p(\cdot|s_k,a_k) \\ a_k\sim\pi(\cdot|s_k)}}[G(s_0,a_0,...,s_T,a_T)|s_0 = s, a_0 = a] \tag{3.7}$$

para todo $s \in \mathcal{S}$ e $a \in \mathcal{A}$.

É possível interpretar a função $V^{\pi}(s)$ como uma função que qualifica o estado de um agente, ou em outras palavras, quanto de recompensa futura ele pode ter a partir daquele estado se ele seguir a política π. Logo, é necessário ver a importância da política na função de valor do estado. Da mesma maneira, a função $Q^{\pi}(s,a)$ é uma função que qualifica o efeito de uma ação específica em um estado específico, assumindo que uma política específica será seguida *após a primeira ação,* a. Aqui dois elementos são muito importantes: a ação a ser tomada no estado e a política que será seguida.

Quando a função de transição de estados do ambiente $p(s'|s,a)$ é conhecida, ou seja, o modelo do ambiente é conhecido, na teoria somente a função V seria necessária para encontrar uma política que resulta no máximo de recompensa possível. Em outras palavras, teoricamente, a função Q não é necessária quando se tem o modelo perfeito do ambiente a ser controlado, como geralmente acontece no contexto de controle ótimo. Então, [Watkins, 1989] propôs a função Q (apesar que [Sutton and Barto, 2018] indica a existência de outras funções de valor da ação anteriores a proposta por [Watkins, 1989]), como uma ferramenta para aprender uma boa política, mesmo sem ter o modelo do ambiente. A seção 3.3.2 irá trazer mais detalhes sobre como se encontrar a política tendo a função V e a função Q.

3.3.1 Equações de Bellman

Como descrito em [Sutton and Barto, 2018], a função de valor tem uma forma recursiva, que é chamada de *equação de Bellman*:

$$V^{\pi}(s) = \sum_{a\in\mathcal{A}} \pi(s|a) \sum_{s'\in\mathcal{S}} p(s'|s,a)[r(s,a) + \gamma V^{\pi}(s')]. \tag{3.8}$$

$$Q^{\pi}(s,a) = \sum_{s'\in\mathcal{S}} p(s'|s,a)[r(s,a) + \gamma \sum_{a'\in\mathcal{A}} \pi(s'|a')Q^{\pi}(s',a')]. \tag{3.9}$$

As equações de Bellman indicam que o valor de cada estado, ou o valor do par estado-ação, depende do valor de todos os outros estados, ações, da política π, da dinâmica da MDP $p(s'|s,a)$ e da recompensa $r(s,a)$. Uma propriedade importante das equações de Bellman é que elas têm uma única solução. Essas equações de Bellman têm muita relevância para diversos algoritmos de aprendizagem por reforço, que serão descritos mais a diante.

Além disso, é possível notar que existe um vínculo entre $V^\pi(s)$ e $Q^\pi(s,a)$. O valor de um estado depende do valor das ações possíveis naquele estado, modulado pela probabilidade de uma ação ser tomada, isto é a política $\pi(a|s)$. Matematicamente falando:

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) Q^\pi(s,a). \tag{3.10}$$

É possível escrever também $Q^\pi(s,a)$ em função de $V^\pi(s)$, como:

$$Q^\pi(s,a) = \sum_{s' \in \mathcal{S}} p(s'|s,a)[r(s,a) + \gamma V^\pi(s')]. \tag{3.11}$$

3.3.2 Otimalidade

A forma como as funções V^π e Q^π foram definidas considera políticas π arbitrárias, mas o objetivo principal em aprendizagem por reforço é conseguir encontrar a política que resulta no máximo de retorno, partindo de qualquer estado. Essa política é referida como *política ótima* π^*. Existe também uma *função de valor de estado ótima*, denotada por $V^*(s)$, e definida como:

$$V^*(s) \triangleq \max_\pi V^\pi(s), \tag{3.12}$$

para todo $s \in \mathcal{S}$. Da mesma maneira é possível encontrar a chamada *função de valor do par estado-ação ótima*, denotada por $Q^*(s,a)$, definida como:

$$Q^*(s,a) \triangleq \max_\pi Q^\pi(s,a), \tag{3.13}$$

para todo $s \in \mathcal{S}$ e $a \in \mathcal{A}$.

As equações de Bellman para as funções de valor ótimas são referidas como *equações de otimalidade de Bellman* e são escritas como:

$$V^*(s) = \max_{a \in \mathcal{A}} \sum_{s \in \mathcal{S}} p(s'|s,a)[r(s,a) + \gamma V^*(s')]. \tag{3.14}$$

$$Q^*(s,a) = \sum_{s' \in \mathcal{S}} p(s'|s,a)[r(s,a) + \gamma \max_{a' \in \mathcal{A}} Q^*(s',a')]. \tag{3.15}$$

Para um MDP finito, a equação de otimalidade de Bellman para $V^*(s)$ tem uma única solução e, uma vez obtendo-a, é possível encontrar política ótima simplesmente buscando para cada estado, as ações que levarão para próximo estado que tenha o maior valor. Matematica-

mente, a política ótima $\pi^*(a|s)$, que é sempre determinística, pode ser derivada tanto a partir da função de valor ótima de estado $V^*(s)$, quanto da função de valor ótima para o par estado-ação $Q^*(s,a)$. Para isso, seja $\mu^*(s)$ a política determinística ótima dada por:

$$\mu^*(s) = \arg\max_a \underbrace{\mathbb{E}_{s' \sim p(\cdot|s,a)}[r(s,a) + \gamma V^*(s')]}_{Q^*(s,a)} \tag{3.16}$$

e $\pi^*(\cdot|a)$ pode ser dada pela equação (3.4).

Nota-se que, de acordo com a equação (3.16), caso a função $Q^*(s,a)$ esteja disponível, encontrar a política ótima é ainda mais simples, basicamente, em cada estado, escolher a ação que maximiza $Q^*(s,a)$. Claro que isso vem com o custo de armazenar não somente os estados, mas também as ações.

Apesar do apelo teórico, as equações de otimalidade de Bellman raramente são utilizadas no contexto de aprendizagem por reforço, visto que a solução requer conhecer a dinâmica do ambiente de forma precisa, ter recurso computacional suficiente para fazer os cálculos e que os estados tenham a propriedade de Markov. Geralmente, as tarefas mais interessantes em AR não atendem tais requisitos e portanto, o que é feito, é encontrar mecanismos de resolver essas equações de forma aproximada.

3.4 Programação Dinâmica

Programação dinâmica (*Dynamic Programming - DP*), de acordo com [Sutton and Barto, 2018], são a coleção de algoritmos que podem ser usados para computar a política ótima dado o modelo perfeito do ambiente como um MDP. Os algoritmos clássicos de DP não têm muito uso na prática em aprendizagem por reforço, mas eles têm uma importância teórica. Geralmente, se assume que o problema tem espaço de ações e estados discreto, mas é possível aplicar DP em problemas contínuos específicos. A seção 4.1 irá mostrar uma dessas aplicações.

Programação dinâmica busca usar as funções de valor para encontrar políticas que resultem em bons retornos. Já foi descrito que uma vez tendo as funções de valor ótima, é possível encontrar a política ótima, de forma direta caso tenha função Q^*, ou de forma indireta, caso tenha V^* e o modelo do ambiente $p(s'|s,a)$. Para isso, será mostrado um algoritmo chamado *iteração de política* que pode ser visto como a base para os outros algoritmos de aprendizagem por reforço.

3.4.1 Iteração de política

O algoritmo chamado *iteração de política* (*policy iteration*) é composto por duas etapas que se alternam: *avaliação da política* (*policy evaluation*) e *melhoria da política* (*policy improvement*). A objetivo é, partindo de uma política determinística $\mu(s)$ qualquer, alternar essas duas etapas até encontrar a política ótima μ^*.

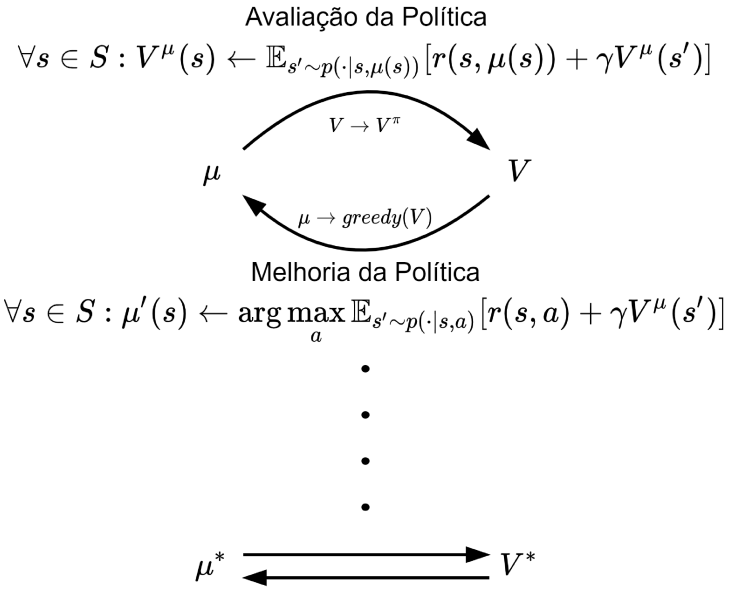

Figura 3.2: Esquemático que descreve o funcionamento do algoritmo *iteração de política*. Imagem adaptada de [Sutton and Barto, 2018]

A etapa de *avaliação da política* consiste em atualizar a função de valor de todos os estados de um MDP, utilizando a equação de Bellman:

$$\forall s \in \mathcal{S} : \quad V^{\mu}(s) \leftarrow \mathbb{E}_{s' \sim p(\cdot|s,\mu(s))}[r(s,\mu(s)) + \gamma V^{\mu}(s')], \tag{3.17}$$

sendo essa etapa executada até que as atualizações na função V^{μ} para todos os estados sejam irrelevantes, portanto, até o algoritmo convergir. Ao executar a *avaliação da política*, será possível obter a função de valor $V^{\mu}(s)$ para uma política μ. A próxima etapa, chamada de *melhoria da política*, irá atualizar para uma política μ' que será melhor ou igual a política anterior μ. Para isso, utiliza-se uma política gulosa em termos da função Q^{μ} dos estados:

$$\forall s \in \mathcal{S} : \quad \mu'(s) \leftarrow \arg\max_a Q^{\mu}(s,a) = \arg\max_a \mathbb{E}_{s' \sim p(\cdot|s,\mu(s))}[r(s,\mu(s)) + \gamma V^{\mu}(s')]. \tag{3.18}$$

Ao alternar as equações (3.17) e (3.18), lembrando de executar a equação (3.17) até que a função V se estabilize, levará a função de valor ótima V^* e a política ótima μ^*. Esse processo é mostrado na figura 3.2. Os autores de [Sutton and Barto, 2018] esclarecem que é possível estender o *iteração de política* para políticas estocásticas $\pi(a|s)$.

Muitos dos algoritmos de aprendizagem por reforço se baseiam nesse arcabouço, e na seção 4.2.1 será visto que é possível expandir essa ideia incluindo a maximização da entropia, que é o *core* do algoritmo Soft Actor-Critic, que será visto mais adiante. Outro agente muito famoso em aprendizagem por reforço, o REINFORCE, também pode ser visto como uma *iteração de política*.

3.4.2 Iteração de valor

Como explicado por [Sutton and Barto, 2018], é possível alterar a *iteração de política* de forma que a etapa de *avaliação da política* ocorra apenas uma vez para todos os estados. Essa modificação que faz o truncamento da *avaliação da política* para apenas uma execução para todos os estados, e a proposta de *melhoria da política*, foi proposta como o algoritmo *iteração de valor* (*value iteration*), fazendo com que a regra de atualização seja:

$$\forall s \in \mathcal{S}: \quad V(s) \leftarrow \max_a \mathbb{E}_{s' \sim p(\cdot|s,a)}[r(s,a) + \gamma V(s')]. \tag{3.19}$$

É possível visualizar a *iteração de valor* como a aplicação da equação (3.9) como uma regra de atualização. É necessário que $V(s) = 0$ se o estado s é um estado terminal. Após a aplicar a equação (3.19) sucessivas vezes em todos os estados, até que a atualização em $V(s)$ para todos os estados seja irrelevante, é possível encontrar a política $\mu(s)$, utilizando:

$$\forall s \in \mathcal{S}: \quad \mu(s) = \arg\max_a \mathbb{E}_{s' \sim p(\cdot|s,a)}[r(s,a) + \gamma V(s')]. \tag{3.20}$$

O algoritmo *iteração de valor* será utilizado como base para a proposta de agente que será descrita na seção 4.1.

3.5 Introdução à exploração

Explorar o ambiente é um dos passos mais importantes em aprendizagem por reforço. Ele é necessário para levar o agente a conhecer estados que podem não ser encontrados utilizando a política enquanto a mesma está atualizando. Por conta disso, existem dois modos em que o agente escolhe as ações a serem executadas: exploração (*exploration*) e aprimoramento (*exploitation*). O modo de exploração geralmente considera uma política que não está relacionada com a função de valor Q ou V, normalmente uma política aleatória. Já o modo de aprimoramento considera a política que está sendo aprendida pelo agente, podendo ser uma sequência de ações partindo de um estado inicial qualquer que se repete de forma sistemática.

Existe um dilema entre a exploração e o aprimoramento: se a política seleciona sempre as mesmas ações (aprimoramento), o agente nunca irá descobrir novos estados e então as estimativas irão convergir para mínimos locais. Se a política sempre considera ações aleatórias (exploração), a política avaliada nunca é a política atual π, mas sim uma política aleatória. A dosagem certa dessas duas abordagens é um desafio dentro de aprendizagem por reforço e uma boa estratégia em geral é considerar mais o exploração no início do episódio e mais aprimoramento no final do episódio.

Existem várias técnicas de exploração, mas uma estratégia muito simples e muito utilizada na área é a ϵ-*greedy*, que simplesmente gera as ações utilizando seguinte regra:

$$a = \begin{cases} \bar{a} \sim \pi(s|a) \text{ com probabilidade } (1-\epsilon) \\ \text{qualquer outra ação com probabilidade } \epsilon. \end{cases} \tag{3.21}$$

Há duas abordagens para tratar da exploração em aprendizagem por reforço:

1. ***On-policy*:** criação das trajetórias utilizando a política π, em conjunto com alguma estratégia de exploração.

2. ***Off-policy*:** criação das trajetórias com uma segunda política chamada de, enquanto é aprendida, a *política de comportamento alvo* para o aprimoramento.

Uma vantagem dos métodos *off-policy* é que o conhecimento do problema em si pode ser usada para restringir a escolha dos pares estado-ação a serem utilizados. Por outro lado, se a solução ótima não for explorada pela política de comportamento, então o agente não terá como aprendê-la.

3.6 Técnicas de Amostragens

Utilizar programação dinâmica implica em saber com exatidão o modelo do ambiente, o que nem sempre é viável. No entanto, é possível encontrar a política ótima, e por consequência, as funções V e Q ótimas, mesmo sem ter o modelo, utilizando técnicas que fazem *amostragens* do ambiente, realizando ações e coletando estados e recompensas. Esses métodos são chamados de *model-free*, porque eles não precisam do modelo para realizar as estimativas. Duas das principais amostragens são *amostragem Monte-Carlo* e *amostragem via diferença temporal*.

3.6.1 Amostragem Monte-Carlo

A amostragem Monte-Carlo (MC) é uma técnica de amostragem que usada para estimar quantidades definidas por esperança matemática. Supondo que o objetivo é calcular a função de valor de um estado específico s para uma política π, $V^{\pi}(s)$. A amostragem MC determina que realize-se um episódio completo, que é uma sequência finita de transições (s, a, r, s'), iniciando de s até um estado terminal s_T. Então computa-se o retorno obtido por aquele episódio e, $G^{(e)}$, que é uma amostra da variável aleatória G. Repete-se esse procedimento múltiplas vezes, até que o valor do estado s seja aproximado pela média dos retornos:

$$V^{\pi}(s) = \mathbb{E}_{\pi}[G|s_0 = s] \approx \frac{1}{M} \sum_{e=1}^{M} G^{(e)}. \tag{3.22}$$

Esse tipo de amostragem foi utilizada para derivar a regra de atualização dos pesos para o agente *policy gradient* / REINFORCE. O livro de [Sutton and Barto, 2018] descreve diferentes maneiras de estimar as funções de valor V e Q através dessa amostragem, o que é referido

como *predição*, e diversas formas de estimar a política, tanto *on-policy* quanto *off-policy*. Isso é referido como *controle*.

As maiores vantagens da amostragem MC são: é um estimador não-enviesado; ele funciona mesmo para POMDPs, visto que a estimativa considera todo episódio. As maiores desvantagens de métodos MC são: o horizonte da tarefa precisa ser finito; é um estimador de alta variância; deve existir uma exploração para garantir que as estimativas convirjam para os valores ótimos.

3.6.2 Amostragem via diferença temporal

A amostragem via diferença temporal (*temporal difference - TD*) busca utilizar as vantagens de programação dinâmica e amostragem Monte-Carlo para fazer a predição e o controle. Ao invés de esperar o episódio terminar igual MC, a amostragem TD considera somente a próxima transição para atualizar as estimativas. Portanto, a atualização da função de valor V é dada por:

$$V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha(r(s,a) + \gamma V^{\pi}(s') - V^{\pi}(s)) \tag{3.23}$$

onde α é a taxa de aprendizagem. É possível ver uma semelhança entre a equação (3.23) e (3.8). Isso é chamado *boostrapping* (ver [Sutton and Barto, 2018]).

A função Q também pode ser atualizada similarmente à equação (3.23):

$$Q^{\pi}(s,a) \leftarrow Q^{\pi}(s,a) + \alpha(r(s,a) + \gamma Q^{\pi}(s',a') - Q^{\pi}(s,a)). \tag{3.24}$$

Dentro das equações (3.23) e (3.24) é possível ver uma subtração que se assemelha com algum tipo de erro. Na realidade, essa quantidade, dada por $\delta = r(s,a) + \gamma V^{\pi}(s') - V^{\pi}(s)$ e $\delta = r(s,a) + \gamma Q^{\pi}(s',a') - Q^{\pi}(s,a)$, é chamada de *TD error* e ela define a surpresa entre a predição atual ($V^{\pi}(s)$ ou $Q^{\pi}(s,a)$) e a soma da recompensa imediata mais a predição para o próximo estado ou estado-ação.

δ pode representar três diferentes cenários:

- Se $\delta > 0$, então a transição foi positiva, ou seja, foi possível obter mais recompensa ou chegar a um estado de maior valor.
- Se $\delta < 0$, então a transição foi negativa, ou seja, a predição anterior foi melhor que a atual.
- Se $\delta = 0$, então a transição foi perfeitamente predita.

Esse tipo de amostragem será a base para dois algoritmos muito famosos em aprendizagem por reforço: *Q-Learning* e *SARSA*, que serão detalhados na seção 3.7.

As principais vantagens da amostragem TD sobre a amostragem MC são:

- Não tem necessidade de esperar o episódio terminar para computar as estimativas, ou seja, ela realiza uma *aprendizagem online*;

- Ela pode ser utilizada em tarefas contínuas, sem estado terminal;
- É um estimador de baixa variância.

Suas principais desvantagens são:

- Ela não tem garantia de funcionar em POMDPs;
- É um estimador enviesado;
- Ela depende mais fortemente das estimativas, ou seja, quando elas no início estão erradas, pode levar um tempo até que elas sejam corrigidas.

3.6.3 Outras técnicas de amostragem:

Na teoria as duas amostragens (MC e TD) podem ser consideradas como casos limites de uma amostragem mais geral chamada de *n-Step Bootstrapping*. O livro de [Sutton and Barto, 2018] detalha a relação entre essas amostragens, e mostra como a aprendizagem TD pode ser interpretada como *1-step booststrapping* e a amostragem MC pode ser interpretada como ∞*-step booststrapping*. Na prática, considerar outros números de passos aumentam significativamente a performance do algoritmo, além de ser uma maneira de controlar o *trade-off* viés-variância, um dos maiores desafios na aprendizagem de máquina.

Uma outra maneira de ajustar o viés e a variância é utilizando *elegibility traces*. Esta dissertação não irá entrar nos detalhes, visto que os agentes que serão descritos não utilizam dessa estratégia, mas ela é detalhada no livro de [Sutton and Barto, 2018].

3.7 Q-Learning e SARSA

No aprendizado da função Q pela amostragem TD, pode-se usar duas diferentes ações a': a ação que realmente será tomada na próxima transição (definida pela política $\pi(a'|s')$), ou uma outra ação qualquer, como por exemplo a ação que maximiza a estimativa atual da função Q, também chamada de ação *gulosa* ($a^* = \arg\max_{a'} Q^\pi(s', a')$). Isto está relacionado com as abordagens *on-policy* e *off-policy* visto na seção 3.5 e resulta em dois algoritmos muito famosos em aprendizagem por reforço:

- **On-policy TD:** é chamado de **SARSA** (*state-action-reward-state-action*). Essa abordagem utiliza a próxima ação amostrada da política $\pi(a'|s')$ para atualizar a transição atual. É necessário que essa próxima ação seja realmente tomada na próxima transição. A equação de atualização da função $Q^\pi(s, a)$ para esse método é:

$$Q^\pi(s, a) \leftarrow Q^\pi(s, a) + \alpha(r(s, a) + \gamma \underbrace{Q^\pi(s', a')} - Q^\pi(s, a)). \tag{3.25}$$

onde $a' \sim \pi(\cdot|s')$. É necessário que a política $\pi(a|s)$ tenha alguma estratégia de exploração, como por exemplo ser ϵ-greedy (ver seção 3.5).

- **Off-policy TD:** é chamado de **Q-Learning**. A ação gulosa no próximo estado (aquela com mais valor Q) será utilizada para atualizar a transição atual. Isso não significa que a ação gulosa tenha que ser utilizada na próxima transição. A equação de atualização da função $Q^\pi(s,a)$ para esse método é:

$$Q^\pi(s,a) \leftarrow Q^\pi(s,a) + \alpha(r(s,a) + \gamma \underbrace{\max_{a'} Q^\pi(s',a')} - Q^\pi(s,a)). \tag{3.26}$$

  A política $\pi(a|s)$ resultante desse método pode ser inclusive determinística, determinada pela máxima estimativa da função Q em cada estado. Para garantir a exploração, no Q-Learning a política de comportamento deve ser ter alguma estratégia de exploração associada a ela, como por exemplo, ser ϵ-greedy.

  Note que a diferença entre as atualizações da função $Q^\pi(s,a)$ utilizando o *Q-Learning* e o *SARSA* são somente os termos destacados.

3.8 Arquiteturas ator-crítico

A arquitetura ator-crítico é uma arquitetura que visa aprender tanto a função de valor quanto a política ótima de forma *simultânea*. Nessa arquitetura, o ator representa a política, porque ele é usado para selecionar as ações, enquanto o crítico representa a função de valor, porque ele avalia as ações feitas pelo ator. Quando foi proposta, por [Witten, 1977b], essa arquitetura considerava somente um aprendizado *on-policy*, mas nos trabalhos mais recentes já é possível encontrar aplicações *off-policy* como os agentes DDPG e SAC que serão vistos na seção 4.2.

A utilização do ator-critico foi inicialmente proposta para tentar utilizar o *TD error* para a função de valor V^π. Como foi visto anteriormente, os métodos que utilizam a amostragem TD geralmente usam a função Q, como o *Q-learning* e o SARSA. Isso acontece porque a função V^π sem o modelo do ambiente não é suficiente para encontrar a ação que levará para o próximo estado de maior valor V^π, ver equação (3.16). Para corrigir isso, foi proposto essa arquitetura, que, uma vez tendo a função V^π para um estado e a política aprendida até então, o ator será atualizado de acordo com o *TD error* $\delta = r(s,a) + \gamma V^\pi(s') - V^\pi(s)$ estimado pelo crítico. Por exemplo, uma ação com *TD error* positivos tem que ser utilizada mais vezes, porém ações com *TD error* negativos têm que ser evitadas sempre que possível. A figura 3.3 mostra a estrutura básica dessa arquitetura.

Portanto, algoritmos ator-crítico aprendem ao mesmo tempo dois aspectos do mesmo problema: a função $V^\pi(s)$ ou a função $Q^\pi(s,a)$ em um crítico; e a política π em um ator.

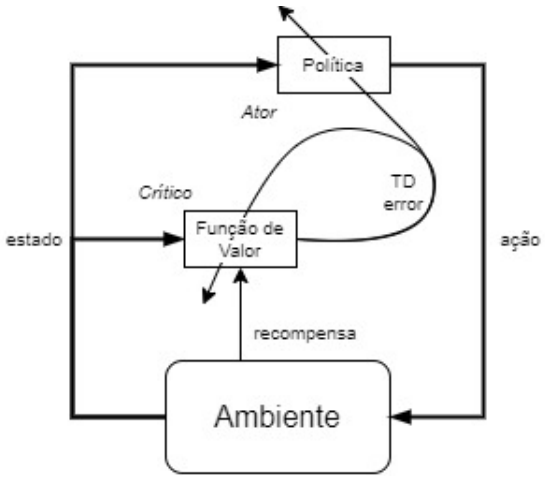

Figura 3.3: Arquitetura Ator-Crítico. Imagem adaptada de [Sutton and Barto, 1998].

3.9 Funções de aproximação

Todos os métodos apresentados até então são *métodos tabulares*, ou seja, funcionam para espaços $\mathcal{S}$ e $\mathcal{A}$ discretos e finitos. Porém, muitas aplicações requerem que ambos sejam contínuos, como o caso da robótica. Desta forma, é necessário utilizar algum tipo de ferramenta capaz de aproximar as soluções mostradas para ambientes contínuos, como por exemplo uma *função de aproximação*. Existem funções de aproximação que são parametrizadas, e o devido ajuste desses parâmetros fará com que seja possível aproximar, por exemplo, as funções $V^{\pi}(s)$ e $Q^{\pi}(s,a)$ para estados e ações contínuas. Da mesma maneira, é possível aproximar tanto a política estocástica $\pi(a|s)$ quanto a política determinística $\mu(s)$, também a partir do ajuste dos parâmetros de função de aproximação parametrizada.

Uma das principais características dessas funções é a capacidade de *generalização* para estados e ações não visitadas durante o treinamento. Nota-se que em diferentes aplicações, quando o espaço $\mathcal{S}$ é muito grande, nem sempre pequenos desvios irão resultar em grandes alterações nas ação a ser escolhida. Um exemplo simples é que a mudança de um pixel de uma imagem geralmente não indica que a ação a ser executada tem que ser totalmente diferente. Baseado nisso, as funções de aproximação são utilizadas para interpolar os estados nesses cenários, mesmo que alguns deles nunca seja visitados durante o treinamento.

O livro de [Sutton and Barto, 2018] destaca dois tipos de funções de aproximação: as funções lineares e as funções não-lineares. As funções lineares consideram um mapeamento linear entre o vetor de parâmetros e um outro vetor de elementos que é chamado de *features*. Por exemplo, seja $V^{\pi}(s;w) = w^T x(s)$ uma função de parâmetros w que aproxima a função de valor, $V^{\pi}(s;w) \approx V^{\pi}(s)$, utilizando um vetor de *features* que depende do estado s. Apesar do seu apelo teórico, definir essas *features* é uma tarefa que requer conhecimento prévio do ambiente estudado. O livro descreve maiores detalhes sobre como ajustar diferentes métodos de aprendizagem por reforço tabular para funções lineares. Já as funções de aproximação não-lineares consideram um mapeamento não-linear, porém diferenciável em relação os parâmetros

da função. Umas das funções de aproximação não-lineares mais comuns atualmente são as redes neurais artificiais, onde os parâmetros são os pesos e os vieses dos neurônios das redes. Essas redes têm a característica de *aprenderem* as *features* de forma automática de acordo com o problema proposto. Como diversos trabalhos recentes foram propostos para proporcionar esse aprendizado, além do avanço do poder computacional recente, as redes neurais profundas acabaram sendo uma das primeiras escolhas como funções de aproximação. Nesse sentido, a seção 4.2 irá apresentar dois algoritmos capazes de atualizar os parâmetros de funções de aproximação construídas a partir de redes neurais profundas.

3.10 Resumo do capítulo

Este capítulo apresentou diversos conceitos fundamentais dentro da área de aprendizado por reforço, iniciando com os conceitos de agente e ambiente. Então, foi apresentado o Processo de Decisão de Markov (MDP do inglês), e sua variação parcialmente observável (POMDP), além da formalização do objetivo de aprendizado por reforço. Em seguida, foram apresentados os conceitos de política e funções de valor, além do conceito de otimalidade e as equações de Bellman. O capítulo seguiu apresentando dois algoritmos de programação dinâmica clássicos, a *iteração de valor* e a *iteração de política*. Na sequência, foram apresentadas algumas técnicas de exploração de agentes, e técnicas de amostragem, que são alternativas interessantes à programação dinâmica. Em seguida foram apresentados dois agentes clássicos na área, os famosos *Q-Learning* e SARSA. O capítulo finalizou apresentado a arquitetura ator-crítico e as funções de aproximação. Estes conceitos serão importantes ferramentas para o próximo capítulo.

Capítulo 4

Metodologia

Neste capítulo serão apresentadas duas abordagens para realizar o controle de movimento de robôs utilizando aprendizagem por reforço.

4.1 Abordagem 1: Iteração de valor na álgebra max-plus

Como foi descrito na introdução, é um consenso que técnicas clássicas de navegação de robôs têm limitações, como visto em [Júnior, 2018]. Uma potencial solução para essas limitações consiste em adaptar essas técnicas fazendo uso das ferramentas de aprendizagem mencionadas no capítulo 3. Essa abordagem será proposta para robôs sem a restrição de não-holonomia, assumindo um modelo de um integrador simples.

Como definido na seção 3.1, o arcabouço de aprendizado por reforço pode ser dividido em dois componentes: o agente e o ambiente. Portanto, nesta seção será apresentado um tipo de agente que irá aprender uma política determinística $\mu(s)$, a partir do algoritmo *iteração de valor*, porém descrito na *álgebra max-plus*. Além disso, será formulado o ambiente através de um MDP.

Como será visto mais a diante, esse agente tem a particularidade de aprender ao se movimentar no ambiente de forma *online*, onde o robô irá aprender a se movimentar no ambiente a medida que ele se desloca no mesmo. Esse é um algoritmo *on-policy*, com uma *amostragem via diferença temporal* e *arquitetura ator-crítico*, onde o agente tem um ator determinístico e um crítico que aprende a função V. Como foi mencionado na seção 1.3, este trabalho foi apresentado no artigo [Júnior et al., 2021].

4.1.1 Contextualização

Ambiente:

Inicialmente, é necessário representar a tarefa de navegação de um robô sem a restrição de não-holonomia, em um cenário com obstáculos, como um MDP. O robô tem configuração q, sendo

o objetivo do agente, realizar ações a de forma que o robô alcance o conjunto de configurações alvo, que será definido como $\mathcal{S}_{tg}$. Considerando que o espaço de configurações tem regiões que indicam colisão do robô com obstáculos, define-se $\mathcal{S}_{free}$ como o espaço de configurações livre de colisões. O robô não tem a restrição de não-holonomia, tendo sua dinâmica determinada por $a = \dot{q}$. O MDP, como definido na seção 3.2, deste problema é dado por:

1. O espaço de estados do ambiente $\mathcal{S}$ é o espaço de configurações do robô , que neste caso será $\mathcal{S} = \mathbb{R}^n$;

2. O espaço de ações $\mathcal{A}$ é o espaço tangente do espaço de configurações ($\mathcal{A} \subseteq \mathbb{R}^n$) com a restrição que $\|a\|_2 = 1$. Esta restrição é usada por simplicidade na ação resultante, porém, uma vez obtida a solução ótima, ela permanece sendo ótima para o problema obtido removendo essa restrição. Por exemplo, mesmo se a política ótima for multiplicada por um fator positivo, ela ainda será ótima no problema modificado.

3. A função de transição de estados $p(s'|s, a)$ do ambiente é determinística, sendo dada pela versão discreta da dinâmica do robô ($\dot{s} = a$):

$$p(s'|s,a) = \begin{cases} 1, \text{ se } s' = s + a\Delta t \\ 0, \text{ caso contrário.} \end{cases} \tag{4.1}$$

4. A recompensa r será o oposto do custo por movimento no estado s codificada como $r : \mathcal{S} \times \mathcal{A} \longmapsto \mathbb{R}^-$, com:

$$r(s,a) = \begin{cases} -\Delta t, \text{ se } s + a\Delta t \in \mathcal{S}_{free} \\ -\infty, \text{ caso contrário.} \end{cases} \tag{4.2}$$

   Nota-se que essa recompensa depende da ação a. Essa formulação de $r(s, a)$ considera obter o caminho mais curto até o conjunto alvo, utilizando a métrica Euclidiana;

5. O fator de desconto é unitário ($\gamma = 1$), logo o retorno G é não descontado.

Essa formulação é equivalente àquela apresentada por [Júnior et al., 2021].

Proposta de solução:

Como descrito anteriormente, a política do agente $\mu(s)$ é determinística e contínua. Utilizando a equação de Hamilton-Jacobi-Bellman, é possível mostrar que a função de valor ótima $V^*(s)$ satisfaz a famosa *equação Eikonal* [Bardi and Capuzzo-Dolcetta, 2008]:

$$\begin{cases} \|\nabla V^*(s)\|^2 = 1, \text{ se } s \in \mathcal{S}_{free} \\ V^*(s) = 0, s \in \mathcal{S}_{tg}. \end{cases} \tag{4.3}$$

Resolvendo a equação Eikonal (4.3), é possível conseguir a política ótima $\mu^*(s)$:

$$\mu^*(s) = \nabla V^*(s). \tag{4.4}$$

Entretanto, a equação (4.3), sendo uma equação parcial não-linear, não é facilmente resolvida de forma analítica [Vavryčuk, 2012, Jahanandish, 2010], logo, será proposto um agente que, ao realizar a estimativa de $V^*(s)$, irá aprender a política ótima $\mu^*(s)$.

O algoritmo deste agente será composto por quatro passos:

1. **Etapa de discretização do espaço:** amostra-se estados no espaço $\mathcal{S}_{free}$ e associa a cada amostra uma função de valor e uma política. Elas deverão ser inicializadas sob certas condições para que o algoritmo funcione. A saída desta etapa são amostras do espaço de configurações, cuja função de valor e política são aproximações do verdadeira valor.

2. **Etapa de atualização das estimativas:** atualiza-se as estimativas da função de valor e da política de cada amostra, considerando uma regra de atualização que se baseia em programação dinâmica. A saída desta etapa são as funções de valor e as políticas discretas das amostras.

3. **Etapa de interpolação das amostras:** esta etapa irá interpolar as amostras próximas para gerar aproximações contínuas da função de valor e da política. O resultado desta camada é um política contínua para ser utilizada na geração da ação de controle do robô.

4. **Etapa de evitamento de obstáculos:** esta etapa será uma camada de proteção do robô que irá realizar o evitamento dos obstáculos. O resultado desta etapa é uma política contínua e segura, que irá desviar o robô dos obstáculos, objetivando preservar o máximo possível da política estimada na etapa anterior.

As próximas seções irão detalhar essas quatro etapas. Além disso, será apresentada uma estratégia de movimentação adicional para que o agente explore o espaço de configurações livre, enquanto aprende a função de valor ótima de certos pontos do espaço. Esse agente foi apresentado, de maneira equivalente, no artigo [Júnior et al., 2021].

4.1.2 Proposta para estimar a função de valor e a política das amostras

A maneira sugerida para estimar a função de valor $V^*(s)$ e a política $\mu^*(s)$ considera amostrar o espaço de configurações $\mathcal{S}_{free}$ gerando as amostras $\mathbb{s}_i$'s que estarão contidas no conjunto $\mathcal{S}_{dis}$. Pelo menos uma $\mathbb{s}_i$ precisa estar no conjunto alvo ($\mathbb{s}_i \in \mathcal{S}_{tg}$). Não existem outras restrições sobre como fazer essa amostragem, no entanto, a seção 4.1.5 apresentará um procedimento prático para gerar as amostras. Cada $\mathbb{s}_i$'s terá uma função de valor $\mathbb{V}_i \approx V^*(\mathbb{s}_i)$ e uma política $\mathbb{\mu}_i \approx \mu^*(\mathbb{s}_i)$ associada a ela.

Pode-se visualizar o resultado dessa amostragem como um Diagrama de Voronoi no espaço de configurações, a medida que mais amostras são geradas, esse diagrama irá se alterar. Cada células de Voronoi será centrada nos elementos do conjunto $\mathcal{S}_{dis}$. No entanto, vale ressaltar que a comparação com o Diagrama de Vonoroi foi feita apenas para efeitos de explicação, e que não há necessidade em construir esses diagramas explicitamente durante o movimento do robô, o que seria uma tarefa computacionalmente custosa. O que é necessário é que o robô saiba em qual célula de Voronoi ele se encontra, analisando qual centro de Voronoi ele está mais perto, o que pode ser computado de forma bem eficiente. A figura 4.1 mostra um exemplo desse diagrama.

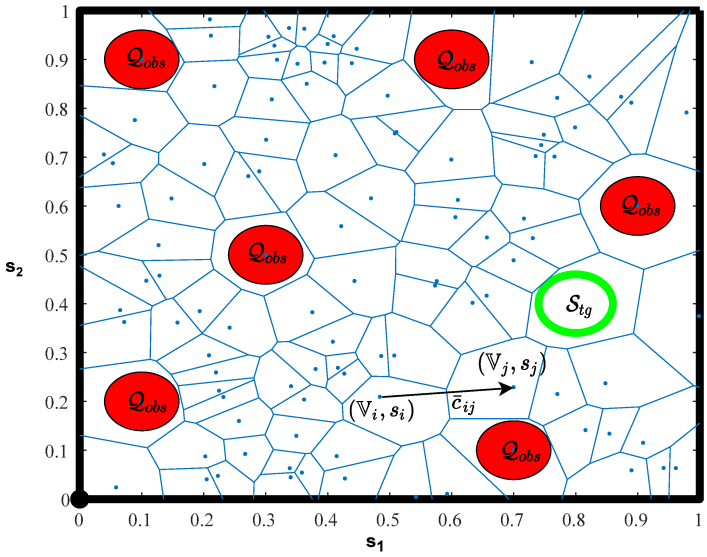

Figura 4.1: Representação de um diagrama de Voronoi de um ambiente com obstáculos. Os obstáculos estão representados círculos vermelhos $\mathcal{Q}_{obs}$, e os pontos azuis representam os centros de Voronoi denotados pela tupla $(\mathbb{V}_i, \mathbb{s}_i, \mathbb{\mu}_i)$. O conjunto alvo $\mathcal{S}_{tg}$ é representado pelo círculo verde. Fonte: autor.

O objetivo nesta proposta é estimar a função de valor $V^*(s)$ e a política $\mu^*(s)$ dessas amostras *dinamicamente*, ou seja, enquanto o robô se movimenta no espaço de configurações. Mais adiante, na seção 4.1.6, será mostrada uma estratégia para fazer a movimentação do robô, mas assumindo exista tal estratégia, fica claro que o robô sempre estará em uma célula de Voronoi durante todo o movimento, e eventualmente ele irá trocar de célula de Voronoi. Quando o robô trocar de célula de Voronoi, ou seja, sair de uma célula cujo centro é $\mathbb{s}_i$ para outra célula cujo centro é $\mathbb{s}_j$, a equação (4.5) será utilizada para atualizar a estimativa de $\mathbb{V}_i$:

$$\mathbb{V}_i \leftarrow \max_j(\mathbb{V}_i, \bar{r}_{ij} + \mathbb{V}_j), \tag{4.5}$$

no qual $\bar{r}_{ij}$ é uma estimativa do custo ótimo no caminho começando em $\mathbb{s}_i$ e terminando em $\mathbb{s}_j$, isto é, uma estimativa de $V(\mathbb{s}_i) - V(\mathbb{s}_j)$. Para a escolha particular de $r(s) = -\Delta t$ no

conjunto $\mathcal{S}_{free}$, é possível fazer $\bar{r}_{ij} = \|\mathbb{s}_i - \mathbb{s}_j\|$. Um caso particular dos resultado obtidos em [Gonçalves, 2021] mostra que essa iteração converge para a solução da equação de Bellman $\mathbb{V}_i = \max_j(\bar{r}_{ij} + \mathbb{V}_j)$, desde que o valor inicial de $\mathbb{V}_i$ sub-estime o seu valor real. Esse problema de programação dinâmica é uma aproximação do problema proposto. Nota-se que a equação (4.5) é a equação (3.23) para $\gamma = 1$ e α sendo:

$$\alpha = \begin{cases} 1, \text{ se } \mathbb{V}_j + \bar{r}_{ij} > \mathbb{V}_i \\ 0, \text{ caso contrário.} \end{cases} \tag{4.6}$$

Toda vez que $\mathbb{V}_i$ ser atualizado através da equação (4.5), se $\mathbb{V}_j + \bar{r}_{ij} > \mathbb{V}_i$, a política $\mathbb{\mu}_i$ será atualizada de acordo com a equação (4.7)

$$\mathbb{\mu}_i \leftarrow \frac{\mathbb{s}_j - \mathbb{s}_i}{\|\mathbb{s}_j - \mathbb{s}_i\|}. \tag{4.7}$$

Para que a função de valor das amostras $\mathbb{V}_i$'s seja estimada corretamente pela equação (4.5) e por consequência a política das amostras $\mathbb{\mu}_i$'s também seja estimada corretamente, é necessário que $\mathbb{V}_i$ seja inicializado sub-estimando o seu valor real. Para isso, foi proposta a seguinte inicialização:

- $\mathbb{V}_i = -Kd(\mathbb{s}_i)$, onde $d(\mathbb{s}_i)$ é a distância Euclidiana de $\mathbb{s}_i$ até $\mathcal{S}_{tg}$, desconsiderando os obstáculos (uma distância "*naive*") e $K \in \mathbb{R}^+$ sendo um fator de escala ($K \geq 1$).
- $\mathbb{\mu}_i = \frac{\mathbb{s}_{tg} - \mathbb{s}_i}{\|\mathbb{s}_{tg} - \mathbb{s}_i\|}$, sendo s_{tg} qualquer elemento do conjunto de amostras $\mathcal{S}_{dis}$ no conjunto alvo $\mathcal{S}_{tg}$.

É importante ressaltar que a condição necessária para o funcionamento da abordagem é somente uma inicialização sub-estimando a função de valor $\mathbb{V}_i$, além de assumir que $\mathbb{V}_i = 0 \ \forall \mathbb{s}_i \in \mathcal{S}_{tg}$, mas a inicialização da política das amostras $\mathbb{\mu}_i$ também tem relevância. Isso decorre do fato que, até que o robô atualize a política das amostras pela equação (4.7), ele estará utilizando a política inicial, e a forma como ela é definida influência diretamente nos resultados, pois a mesma será utilizada na estratégia de exploração que irá compor a ação a.

Destaca-se que o parâmetro K pode ser definido dependendo do número de obstáculos presentes no ambiente. Com mais obstáculos, a estimativa dos $\mathbb{V}_i$'s iniciais precisa ser mais conservadora, portanto maior, do que em um cenário com nenhum ou poucos obstáculos. Por exemplo, em um cenário com nenhum obstáculo, $K = 1$ é suficiente. Além disso, essa inicialização tem como vantagem o fato que a ação de controle inicial ser suficiente em vários cenários, como por exemplo, naqueles que não possuem obstáculos, ou ainda, para cenários com poucos obstáculos.

Concluindo, essa inicialização tem muita importância na eficiência do aprendizado, e como será visto na seção de resultados, uma inicialização aleatória acarreta em uma dificuldade maior dos campos vetoriais serem corrigidos, mesmo com o algoritmo garantindo que eventualmente eles serão.

4.1.3 Obtendo uma ação contínua

Até aqui, a função de valor e a política estimadas estão sendo consideradas apenas nas amostras do conjunto $\mathcal{S}_{dis}$, e seria necessário um número de amostras infinitas para conseguir ter todas as estimativas do espaço livre. Como isso é impraticável, é necessário construir um campo vetorial contínuo e suave para a geração da ação de controle.

Visando resolver isso, definindo a função $\tilde{V}^* : \mathcal{S} \longmapsto \mathbb{R}^-$ como uma estimativa da função de valor em um estado genérico $V^*(s)$. Seja $\mathcal{P}(N, s)$ o sub-conjunto de N estados mais próximos do conjunto $\mathcal{S}_{dis}$ do estado s do ambiente, computa-se $\tilde{V}^*(s)$ como a média dos elementos desse conjunto ponderado pelo inverso da distância de cada elemento desse conjunto a s:

$$\tilde{V}^*(s; \mathbb{V}, \mathbb{s}) = \frac{\sum_{i \in \mathcal{P}(N,s)} \frac{1}{\|s - \mathbb{s}_i\|} \mathbb{V}_i}{\sum_{i \in \mathcal{P}(N,s)} \frac{1}{\|s - \mathbb{s}_i\|}}. \tag{4.8}$$

Com $\tilde{V}^*(s)$ definido apropriadamente, é possível calcular a ação de controle como $\mu^*(s) = \nabla \tilde{V}^*(s)$. Considerando que quando $s = \mathbb{s}_i$ para alguns i, $\tilde{V}^*(s)$ é calculado utilizando o limite, que é $\lim_{s \to \mathbb{s}_i} \tilde{V}^*(s) = \mathbb{V}_i$.

No entanto, esse cálculo pode ter problemas se $\tilde{V}^*(s)$ tem ruído. Nesse caso, o gradiente não irá representar o real gradiente de $V(s)$ corretamente. Desta forma, é proposta uma solução para estimar a política diretamente. Definindo $\tilde{\mu}^* : \mathcal{S} \longmapsto \mathbb{R}^n$ como a estimativa da política $\mu^*(s)$ e, similarmente a, equação (4.8) é possível computar $\tilde{\mu}^*(s)$ como:

$$\tilde{\mu}^*(s; \mathbb{\mu}, \mathbb{s}) = \frac{\sum_{i \in \mathcal{P}(N,s)} \frac{1}{\|s - \mathbb{s}_i\|} \mathbb{\mu}_i}{\| \sum_{i \in \mathcal{P}(N,s)} \frac{1}{\|s - \mathbb{s}_i\|} \mathbb{\mu}_i \|} \tag{4.9}$$

e pelo menos a primeira vista, é possível utilizar essa estimativa diretamente como ação do agente, i.e. $a(s) = \tilde{\mu}^*(s)$. Usando mais uma vez o limite quando $s = \mathbb{s}_i$, resulta em $\lim_{s \to \mathbb{s}_i} \tilde{\mu}^*(s) = \mathbb{\mu}_i$.

Em cenários onde $\mathcal{S}_{dis}$ cobre todo o espaço livre $\mathcal{S}_{free}$, a função $\tilde{\mu}^*(s)$ converge para $\mu^*(s)$ quando os valores do conjunto $\mathbb{\mu}$ são estimados satisfatoriamente. Então, utilizando essa política seria possível levar o robô até $\mathcal{S}_{tg}$ movendo somente dentro de $\mathcal{S}_{free}$ e percorrendo o menor caminho. Porém, é infactível na prática amostrar todo o espaço livre $\mathcal{S}_{free}$ e esperar um perfeito aprendizado de $\mathbb{V}$. Nesse caso, a função $\tilde{\mu}^*(s)$ perde sua garantia de prevenção de colisão, tornando necessária a alternativa descrita na próxima seção.

4.1.4 Evitando os obstáculos

Até então, foi computada uma política contínua $\tilde{\mu}^*(s)$ que move o robô de um estado qualquer s até o conjunto alvo $\mathcal{S}_{tg}$. Essa política contínua irá aproximar a política ótima desse problema. No entanto, como foi descrito anteriormente, na prática essa política em si não é suficiente para evitar os obstáculos no espaço de configurações, uma vez que a mesma não é exatamente

a verdadeira $\mu^*(s)$. Visando resolver esse problema, é proposto uma alternativa que utiliza o seguinte problema de otimização para computar uma versão segura de $\tilde{\mu}^*(s)$ que considera desviar dos obstáculos:

$$\begin{aligned} \tilde{\mu}_c^*(s) = \arg\min_a \ & \|a - \tilde{\mu}^*(s)\|^2 \\ \text{sujeito à} \quad & A(s)a \leq b(s), \end{aligned} \tag{4.10}$$

no qual $A \in \mathbb{R}^{n \times m}$ e $b \in \mathbb{R}_+^{m \times 1}$ são partes das restrições que serão usadas para evitar os obstáculos. Esse problema de otimização se comporta como um "filtro" que irá retornar uma ação considerando os obstáculos do cenário.

Nota-se que é um problema de programação quadrática estritamente convexo, portanto pode ser resolvido eficientemente. Além disso, será garantido mais adiante, por construção, que o espaço factível é sempre não-vazio. Então, a solução a sempre existe e é sempre única e $\tilde{\mu}_c^*(s)$ é uma função bem definida.

Analisando a função objetivo, ela tenta aproximar $\tilde{\mu}_c^*(s)$ e $\tilde{\mu}^*(s)$. Realmente, a solução do problema de otimização quando as restrições não estão ativas é exatamente $\tilde{\mu}_c^*(s) = \tilde{\mu}^*(s)$. Com elas ativas, a solução é a mais próxima possível de $\tilde{\mu}^*(s)$ respeitando as restrições.

Para formular o evitamento de colisão como $A(s)a \leq b(s)$, definindo $\bar{G}(s) : \mathcal{S} \mapsto \mathbb{R}_-^m$ como uma função não-positiva que irá capturar o oposto da distância Euclidiana com os obstáculos. Ela não precisa ser uma distância precisa, mas se ela decrescer quando o robô se afastar dos obstáculos e for zero, se e somente se, o robô estiver colidindo com um obstáculo, ela pode ser utilizada nesse contexto. Como mostrado por [Kanoun et al., 2011], se $\dot{\bar{G}}(s) \leq -\eta \bar{G}(s)$, com η um escalar positivo, e $\bar{G}(s_0) \leq 0$, então $\bar{G}(s_t) \leq 0$ para $t > 0$, então o robô não irá colidir com obstáculos. Já que $\dot{\bar{G}}(s) = \frac{\partial \bar{G}}{\partial s}(s)\dot{s}$ e $\dot{s} = a$, conclui-se que $A(s) = \frac{\partial \bar{G}}{\partial s}(s)$ e $b(s) = -\eta \bar{G}(s)$. Nota-se que $b(s) = -\eta \bar{G}(s) \geq 0$, então o conjunto factível $A(s)a \leq b(s)$ é sempre não-vazio ($a = 0$ é sempre uma solução).

É importante ressaltar que, uma outra possibilidade também avaliada durante o desenvolvimento desse trabalho, foi utilizar campos repulsivos clássicos [Khatib, 1986] em vez de programação quadrática para computar $\tilde{\mu}_c^*(s)$, porém eles levaram a mínimos locais com mais frequência que os métodos usando otimização dados pela equação (4.10) [Panagou, 2014]. Além disso, vale destacar que existem outras alternativas para computar a distância do robô aos obstáculos, assim como obter o jacobiano de tais distâncias. Porém, não faz parte do escopo deste trabalho apresentar tais alternativas.

4.1.5 Estratégia de amostragem

Para realizar a estratégia de aprendizagem, como mencionado na seção 4.1.2, é necessário criar o conjunto $\mathcal{S}_{dis}$ amostrando o conjunto $\mathcal{S}_{free}$. Foi proposto repetir esse procedimento a medida que o robô se move dentro do ambiente por um procedimento de amostragem dentro de uma bola com raio D centrada no estado atual s a cada T segundos. Então, essa proposta considera que o robô tem uma percepção limitada do ambiente.

O problema com a proposta de amostrar pontos em $\mathcal{S}_{free}$ para o conjunto $\mathcal{S}_{dis}$ é *a maldição da dimensionalidade*, onde o problema passa a ter um custo computacional cada vez mais caro uma vez que ele terá que lidar com muitos pontos para a computação, especialmente em espaços de configurações com muitas dimensões. Então, para tornar o problema computacionalmente tratável e garantir uma amostragem abrangente nesse espaço, o processo de amostragem fará a seguinte verificação: amostrar somente se o número de pontos de $\mathcal{S}_{dis}$ dentro da bola é menor que um número fixo M. O agente irá amostrar P pontos, garantindo uma distância mínima h entre eles e entre todos os pontos de $\mathcal{S}_{dis}$. Fazendo esse procedimento repetidamente garante que o espaço livre será coberto de forma mais eficiente.

Nota-se que a cada momento que novos elementos são amostrados, os conjuntos $\mathbb{V}$ e $\mathbb{\mu}$ precisam também ser aumentados, inicializando seus novos elementos como definido na seção 4.1.2.

4.1.6 Proposta de exploração

Todo o arcabouço apresentado até aqui requer que o robô se mova dentro do ambiente para aprender o conjunto de elementos $\mathbb{\mu}_i$ corretamente. Esse movimento pode ser realizado de diferentes maneiras, mas foi proposto utilizar a seguinte abordagem. Seja $w \in \mathbb{R}^n$ uma entrada de perturbação que será adicionada a equação (4.10). Essa perturbação tem como objetivo mover o robô dentro do cenário, adicionando uma componente de *exploração* para o controle de movimento do robô. Então, é necessário modificar $\tilde{\mu}_c^*(s)$ para:

$$\begin{aligned} \tilde{\mu}_c^*(s) = \arg\min_{a} \ & \|a - (\tilde{\mu}^*(s) + w)\|^2 \\ \text{sujeito à} \quad & A(s)a \leq b(s). \end{aligned} \tag{4.11}$$

Como explicado para a equação (4.10), a função objetivo de (4.11) irá igualar $\tilde{\mu}_c^*$ a $\tilde{\mu}^* + w(t)$ quando não existem restrições ativas. Fazendo isso, o robô irá explorar o ambiente, dependendo do valor de w e não irá basear-se somente na política que pode não ser madura o suficiente no início.

A proposta considera as seguintes características de w. Primeiro, a norma máxima de w precisa decrescer com o tempo, uma vez que existe a expectativa de que o robô irá aprender a política ótima a medida que ele se move dentro do ambiente. Segundo, ela precisa reiniciar para a norma máxima se o robô parar em um mínimo local. Essa perturbação será a única maneira do robô sair do mínimo local caso necessário. Terceiro, ela precisará ser constante por um período de tempo, uma vez que o objetivo não é fazer o robô realizar movimentos erráticos, o que poderia acontecer se w variar com muita frequência, mas garantir que ele irá se mover em direções distintas, fazendo com que ele explore o cenário.

Utilizando a equação (4.11), uma vez que o robô chegar no alvo $\mathcal{S}_{tg}$, é possível que ele ainda não tenha aprendido a política ótima ainda. É necessário fazer com o robô ainda se mova

dentro de $\mathcal{S}_{free}$ para continuar a aprendizagem. Então, pode-se induzir um movimento que faça com que o robô vá para outra configuração, e então iniciar o algoritmo de aprendizagem a partir dessa configuração. Na prática, isso poderia ser feito fazendo o robô realizar outra tarefa ou então induzir artificialmente esse movimento fazendo movimentos aleatórios.

4.1.7 Comentários finais

O agente apresentado nesta seção será validado em uma tarefa de navegação de um robô sem a restrição de não holonomia. No entanto, existe interesse considerar robôs com esse tipo de restrição. Desta maneira, na próxima seção será apresentada uma abordagem para fazer navegação de robôs móveis com restrição de não-holonomia, também utilizando aprendizado por reforço. Isso será feito utilizando redes neurais profundas.

Por fim, vale destacar que, na visão do autor, existem vantagens da abordagem desta seção em comparação com aquela baseada em redes neurais que será apresentada na próxima seção, no caso que a aplicação seja feita em robôs sem a restrição de não-holonomia. A primeira vantagem é a existência de uma garantia de convergência, como foi descrito na seção 4.1.2. A segunda vantagem é a otimização do caminho em termos da distância percorrida, também descrita na seção 4.1.2. A terceira é que essa abordagem é mais simples de implementar, pois todos os passos do algoritmo requerem computações relativamente simples.

Existe ainda uma característica histórica por trás dessas abordagens, resultando em ideias apresentadas nesta seção que também serão utilizadas na próxima seção.

4.2 Abordagem 2: Aprendizado por reforço profundo

A principal desvantagem da abordagem descrita anteriormente é sua não aplicabilidade em robôs com restrições de não-holonomia. Além disso, existia também um interesse em estudar técnicas de aprendizado profundo. Portanto, serão mostrados nesta seção agentes modernos que utilizam redes neurais profundas para completar tarefas. Para isso, é necessário modelar a tarefa de navegação de um robô com restrição de não-holonomia com o arcabouço de aprendizagem por reforço.

Nesta seção serão apresentados os detalhes necessários para utilizar as técnicas de aprendizagem por reforço profundo (*Deep Reinforcement Learning - DRL*) visando aprender políticas que irão controlar o movimento de robôs de forma a realizar tarefas. Para isso, serão utilizadas redes neurais artificiais e técnicas de *Deep Learning*, assim como todo o contexto teórico descrito na seção 3. Novamente, será necessário formular dois componentes e suas interações: o agente e o ambiente.

Existem maneiras diferentes de construir agentes para aprender uma política *satisfatória* para o problema, e geralmente elas são classificadas em três diferentes categorias:

1. **Aprendendo somente a função de valor:** são abordagens que efetuam o aprendizado da função de valor do estado $V^{\pi}(s)$ ou do par ação-estado $Q^{\pi}(s, a)$ para inferir uma política de forma indireta. Os mais famosos exemplos utilizando essa abordagem são os já mencionados *Q-Learning* e *SARSA* para o caso tabular, mas também *Deep Q-Network* e suas variações (Double Deep Q-Network, Dueling Deep Q-Network, etc) com funções de aproximação.

2. **Aprendendo somente a política:** são métodos que aprendem uma política diretamente $\pi(a|s)$, mas não fazem nenhuma estimativa da função de valor. O exemplo mais famoso é o *REINFORCE*.

3. **Aprendendo a função de valor e a política:** são métodos chamados *ator-crítico* e realizam o aprendizado da função de valor e da política simultaneamente. Eles são os que têm mais variações, sendo alguns dos principais exemplos *Advantage Actor Critic* (A2C), *Asyncronous Advantage Actor-Critic* (A3C), *Proximal Policy Optimization* (PPO), *Trust Region Policy Optimization* (TRPO), *Soft Actor-Critic* (SAC), *Deep Deterministic Policy Gradient* (DDPG).

Os métodos baseados em função de valor têm suas principais aplicações em ambientes com estados e ações discretas, que não é o foco principal desta dissertação. Portanto, serão apresentados os agentes que realizam o aprendizado pelos outros métodos, visto que eles podem ser usados em ambientes com estados e ações contínuas.

Os ambientes precisam ser formulados no contexto de AR, como descrito anteriormente. Portanto, é necessário definir todos os elementos necessários do MDP (ou POMDP dependendo do problema) que são os já mencionados *espaço de observações, de estados, de ações, a recompensa e a função de transição de estados*. Nesta dissertação, será mostrada como modelar alguns exemplos de controle de movimento de robôs nesse arcabouço.

4.2.1 Agentes

Os agentes mencionados nesta seção foram os mais utilizados durante o trabalho. Vale destacar que existem muitos outros que não foram descritos e que não foram citados, mas que têm aplicações em diferentes problemas nessa área.

Deep Deterministic Policy Gradient (DDPG)

O DDPG [Lillicrap et al., 2015] é um dos agentes que realizam o aprendizado de uma política utilizando o gradiente de uma função objetivo que representa a recompensa esperada de longo prazo. O DDPG foi proposto como uma extensão do *Deterministic Policy Gradient* (DPG) [Silver et al., 2014] para funções de aproximação não-lineares como redes neurais profundas, visto que o DPG não funcionava muito bem com esse tipo de redes. A política que ambos

aprendem é contínua e determinística. Para garantir a exploração do ambiente, as trajetórias geradas pelo agente são provenientes de uma outra política, que é estocástica, o que faz com o que o DDPG seja um método *off-policy*. O espaço de observações pode ser discreto ou contínuo, porém o espaço de ações tem que ser contínuo. Ele é *model-free*, ou seja, ele não precisa do modelo do ambiente e, para executar o aprendizado, ele aprende tanto a política determinística $\mu(s;\theta)$, parametrizada por θ, quanto a função de valor do par ação-estado para essa política $Q^{\mu}(s,a)$. Portanto, ele utiliza a arquitetura ator-crítico.

O artigo do DPG explica como obter o gradiente de uma política determinística que maximiza a recompensa esperada de longo prazo G. Para isso, os autores enunciaram o *Teorema do gradiente de política determinística*, que é a versão determinística do *Teorema do gradiente de política estocástica*, proposto por [Sutton et al., 2000]. Através dele, é possível obter uma regra de atualização dos pesos de uma política determinística $\mu(s;\theta)$ maximizando a função objetivo,

$$J(\theta) = \mathbb{E}_{s\sim\rho[\mu(s;\theta)]}[r(s,\mu(s;\theta))] \tag{4.12}$$

onde $\rho[\mu(s;\theta)]$ é a distribuição dos estados descontados obtido pela política determinística $\mu(s;\theta)$ conforme definido por [Silver et al., 2014].

Os autores de [Silver et al., 2014] mostram que o gradiente dessa função objetivo com respeito aos parâmetros θ é,

$$\nabla_{\theta}J(\theta) = \mathbb{E}_{s\sim\rho[\mu(s;\theta)]}[\nabla_{\theta}\mu(s;\theta)\nabla_{a}Q^{\mu}(s,a)|_{a=\mu(s;\theta)}]. \tag{4.13}$$

Nota-se que equação acima está no formato da arquitetura ator-crítico, onde o ator está relacionado com $\nabla_{\theta}\mu(s;\theta)$ e o crítico está relacionado com $\nabla_{a}Q^{\mu}(s,a)$.

Existe a possibilidade de substituir $Q^{\mu}(s,a)$ por uma função de aproximação $Q(s,a;\varphi)$, parametrizada por φ, porém é necessário que essa função atenda os requisitos descritos no artigo do DPG. Baseado nisso, o DDPG propõe utilizar duas redes neurais como funções de aproximação do ator e do crítico. A rede do ator recebe a observação como entrada e retorna ação respectiva àquela observação. Já a do crítico recebe a observação e a ação realizada pelo ator como entradas e retorna a função Q do par ação-estado. Os pesos da rede do ator são atualizados utilizando a equação (4.13), enquanto os pesos do crítico são atualizados através do gradiente da função de perda $\mathcal{L}(\varphi)$, que é dada por:

$$\mathcal{L}(\varphi) = \frac{1}{N}\sum_{i=0}^{N}(y_i - Q(s_i,a_i;\varphi))^2 \tag{4.14}$$

onde y_i são os alvos das amostras de treinamento definidos pela equação de Bellman mostrada na seção 3.3, ou seja, $y_i = r_i + \gamma Q(s_{i+1},\mu(s_{i+1};\theta');\varphi')$ em que $\mu(s;\theta')$ e $Q(s,a;\varphi')$ são as *target networks* do ator e do crítico respectivamente. As *target networks* serão explicadas a seguir.

Os autores do DDPG viram que usar o algoritmo DPG com funções de aproximação não-

lineares não teria um bom resultado e, nesse sentido, inspirados nas alterações propostas no algoritmo Q-Learning para funções de aproximação não-lineares que resultaram no Deep Q Network (DQN) [Mnih et al., 2015], foram apresentadas as seguintes mudanças:

- **Experience replay:** aplicar o DPG puro em redes neurais não funcionaria porque as amostras de treinamento seriam temporalmente correlacionadas, o que não é uma boa entrada para o treinamento de uma rede neural. Para corrigir isso, foi proposto incluir um *experience buffer*, que armazenaria as amostras obtidas pela interação do agente com o ambiente (s, a, r, s'), e assim, extrair um *minibatch* desse *buffer*, escolhendo aleatoriamente amostras dele.

- **Target networks:** como é possível ver pela equação 4.14, a função de perda do crítico tem uma dependência dos parâmetros atuais da rede neural por conta do alvo da função $Q(s, a)$. Essa característica faz com que o alvo se altere por conta da atualização dos pesos, o que geralmente não traz bons resultados em redes neurais. Para solucionar esse problema, foi proposto criar uma cópia das redes do ator e do crítico, e variar os parâmetros dessas cópias de forma independente dos parâmetros das redes originais. Essa solução também foi considerada no algoritmo DQN, onde os autores atualizavam os parâmetros das cópias periodicamente, tornando-os fixos tempo suficiente para que os algoritmos de aprendizado profundo atuem de forma eficiente. Por outro lado, os autores do DDPG atualizam os parâmetros das cópias de uma maneira suavizada com relação as originais, através de um filtro de primeira ordem. Essas cópias são chamadas de *target networks*.

É importante mencionar que os autores também recomendam utilizar *batch normalization* [Ioffe and Szegedy, 2015] para estabilizar o treinamento.

Nota-se também que a equação (4.13) requer $\nabla_a Q(s, a; \varphi)$, porém o crítico retorna somente $Q(s, a; \varphi)$. Isso acaba não sendo um problema, visto que redes neurais profundas utilizam bibliotecas de diferenciação automática. Uma outra solução é utilizar o método da diferença de Euler finita:

$$\nabla_a Q(s, a; \varphi) \approx \frac{Q(s, a + \delta a; \varphi) - Q(s, a; \varphi)}{\delta a} \tag{4.15}$$

onde δa é um pequeno desvio da ação a a ser executada. Um outro tópico relevante é que apesar do DDPG ser um algoritmo *off-policy*, ele não precisa de *importance sampling* para correção do gradiente da função objetivo. Isso ocorre pois a política é determinística, logo não existe a necessidade de ponderar as probabilidades da política de comportamento e a política que está sendo aprendida.

Pelo fato do DDPG gerar uma política determinística, para garantir a exploração do ambiente, foi proposto adicionar um ruído na saída do ator. Isso é necessário, visto que não são todos

os ambientes que são naturalmente estocásticos. Nesse caso, a saída da rede do ator é dada por $\tilde{\mu}(s;\theta)$,

$$\tilde{\mu}(s;\theta) = \mu(s;\theta) + \mathcal{N} \tag{4.16}$$

onde $\mathcal{N}$ é ruído de exploração. No artigo, foi proposto utilizar o processo de Ornstein-Uhlenbeck (OU) [Uhlenbeck and Ornstein, 1930] que gera um ruído temporalmente correlacionado de média zero.

O DDPG se tornou rapidamente o estado da arte para métodos *model-free* em espaços de ações contínuas, até mesmo aqueles de dimensões elevadas, como em tarefas de robótica. Sua principal desvantagem é a necessidade de muitas amostras de treinamento, também chamada de *sample complexity*, e sua sensibilidade aos ajuste dos hiper-parâmetros do algoritmo.

Soft Actor-Critic (SAC)

O SAC [Haarnoja et al., 2018d] é um agente que busca maximizar a entropia esperada de uma política estocástica enquanto faz o mesmo para a recompensa esperada de longo prazo. Para isso, o objetivo em aprendizagem por reforço é aumentado para considerar um objetivo de máxima entropia (como em [Ziebart et al., 2010]):

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(s_t,a_t)\sim\rho[\pi(\cdot|s_t)]}[r(s_t,a_t) + \alpha\mathcal{H}(\pi(\cdot|s_t))] \tag{4.17}$$

onde α representa a *"temperatura da entropia"*, ou seja, o peso dada à entropia em relação à recompensa do ambiente e $\rho[\pi(a|s)]$ é a marginal do par estado-ação da distribuição da trajetória induzida pela política $\pi(a|s)$ como descrito em [Haarnoja et al., 2018d]. Enquanto o DDPG explora o ambiente através do ruído na saída do ator utilizando Ornstein-Uhlenbeck, o SAC faz o mesmo maximizando a entropia, e assim ele captura política próximas da política ótima, sendo mais robusto ao *overfitting*. Essa robustez ocorre porque a presença do ruído ajuda para que a rede não chegue mínimos locais, que são muito comuns neste tipo de problema. Pelo fato desse objetivo também considerar a recompensa, o agente também irá descartar as ações que não resultem em bons retornos. Ele é um método *off-policy*, ou seja, ele tem um *experience buffer* e aprende uma política para ambientes com espaços de estados e ações contínuos.

Para adicionar a entropia de forma explícita na formulação, os autores definiram uma variação do algoritmo *iteração de política* chamada de *iteração de política suave* (*Soft Policy Iteration*). Como visto na seção 3.4, a *iteração de política* é composta por duas etapas que se alternam, chamadas de *avaliação da política* e *melhoria da política*, fazendo com que uma política π convirja para uma política ótima π^* no caso tabular. A versão suave da *iteração de política* também é a alternância das versões suaves das etapas *avaliação da política* (*Soft Policy Evaluation*) e *melhoria da política* (*Soft Policy Improvement*). Essas versões consideram a entropia da política em conjunto com o retorno. Os autores mostram que a *iteração de política*

suave converge para uma política ótima no caso tabular, assim como a versão tradicional da *iteração de política*.

A *avaliação da política suave* irá computar o valor Q "suave" para uma política dada de forma iterativa, aplicando uma versão modificada do operador de Bellman que é dada por

$$\mathcal{T}Q(s,a) \triangleq r(s,a) + \gamma \mathbb{E}_{\substack{s' \sim p(\cdot|s,a) \\ a' \sim \pi(\cdot|s')}}[Q(s',a') - \log \pi(a'|s')]. \tag{4.18}$$

Já a etapa de *melhoria da política suave* busca atualizar a política na direção da exponencial da nova função Q. Os autores explicam que essa escolha de atualização resulta em uma política melhor em termos dos seus valores Q suaves. Para restringir a gama de políticas, eles definem que a política é restrita ao conjunto Π, que poderia ser por exemplo, uma família de distribuições como Gaussianas. Eles decidiram utilizar a *divergência de Kullback-Leibler* D_{KL} como ferramenta para computar a nova política como:

$$\pi_{new} = \arg\min_{\pi' \in \Pi} \mathbb{E}\left[D_{KL}\left(\pi'(\cdot|s) \middle\| \frac{\exp Q^{\pi_{old}}(s,\cdot)}{Z^{\pi_{old}}(s,\cdot)} \right) \right] \tag{4.19}$$

onde $Z^{\pi_{old}}$ é uma função de partição usada para normalizar a distribuição.

Para aplicar o arcabouço acima em ambiente com espaços de estados e ações contínuos, serão utilizadas funções de aproximação da função Q e da política π, dadas pelas redes neurais parametrizadas $Q(s,a;\varphi)$ para crítico e $\pi(a|s;\theta)$ para o ator. A rede neural do crítico irá receber a ação realizada pelo ator e o estado do ambiente, e irá retornar a função Q suave, enquanto a rede neural do ator irá receber o estado e sua saída serão os parâmetros de uma distribuição de probabilidade, por exemplo, a média e a variância de uma distribuição gaussiana. Os autores na realidade utilizam duas funções de aproximação da função de valor do par estado-ação Q_1 e Q_2, para evitar a sobre-estimativa do viés durante a *melhoria da política suave*, como feito no trabalho de [Fujimoto et al., 2018]. Os parâmetros φ_1 e φ_2 das funções Q_1 e Q_2 respectivamente são atualizados minimizando a chamada residual do *soft* Bellman:

$$J_Q(\varphi_i) = \mathbb{E}_{(s,a)\sim\mathcal{D}}[\frac{1}{2}(Q(s,a;\varphi_i) - (r(s,a) + \gamma \mathbb{E}_{s'\sim p(\cdot|s,a)}[V(s';\varphi_i')]))^2], \tag{4.20}$$

onde $V(s';\varphi_i') = \mathbb{E}_{a'\sim\pi(\cdot|s)}[Q(s',a';\varphi_i') - \alpha \log \pi(a'|s';\theta)]$ e $\mathcal{D}$ é o *experience buffer*. Nota-se que os autores também utilizam a **target network** para o crítico, denotada por $Q(s,a;\varphi_i')$, como feito no DDPG e no DQN. Já os parâmetros da política podem ser estimados utilizando o gradiente:

$$\nabla_\theta J_\pi(\theta) \approx \nabla_\theta \alpha \log(\pi(a|s;\theta)) + (\nabla_a \alpha \log(\pi(a|s;\theta)) - \nabla_a \min_i Q(s,a;\varphi_i)) \nabla_\theta f_\theta(\epsilon;s) \tag{4.21}$$

onde $f_\theta(\epsilon;s)$ é uma reparametrização da política utilizando a rede neural, ou seja, $a = f_\theta(\epsilon;s)$,

onde ϵ é um vetor de ruído de entrada amostrado a partir de uma distribuição fixa, como gaussiana esférica. Nota-se que esse gradiente tem o mesmo estilo do visto na seção do agente DDPG.

Os autores explicam que o hiper-parâmetro α é sensível ao ser ajustado para atingir a convergência do algoritmo, além de geralmente variar dependendo da tarefa, e portanto eles propuseram um problema de otimização que atualiza o parâmetro α de forma automática em conjunto com a função objetivo principal. Para isso, foi derivada a seguinte função objetivo:

$$J_\alpha = \mathbb{E}_{a\sim\pi(\cdot|s)}[-\alpha \log \pi(a|s) - \alpha\bar{\mathcal{H}}] \tag{4.22}$$

onde $\bar{\mathcal{H}}$ é a entropia desejada, que pode ser zero, por exemplo. Os autores indicam que é possível minimizar essa função objetivo em conjunto com a função objetivo principal aproximando o gradiente descendente dual. Eles destacam que, apesar de não ter garantias teóricas que esse método é aplicável com redes neurais, foi visto na prática que ele funciona.

Existem duas principais vantagens ao utilizar o objetivo de máxima entropia: a primeira é que a garantia de mais exploração do ambiente e a segunda é que a rede neural do ator auxilia nas atualizações do crítico. Em outros exemplos de agentes ator-crítico, o ator atua somente gerando amostras de transições do ambiente, enquanto no SAC o ator aparece de forma explícita no crítico como parte da parcela da entropia.

4.2.2 Ambientes

Como feito em 4.1, é necessário modelar o ambiente como um MDP. Para isso, serão definidos:

1. O espaço de estados $\mathcal{S}$, de observações $\mathcal{O}$ e de ações $\mathcal{A}$ do ambiente;
2. A recompensa $r(s, a)$;
3. A função de transição de estados $p(s'|s, a)$;
4. A condição de término de um episódio.

As próximas sub-seções irão descrever o desenvolvimento desses elementos do MDP descritos acima para um robô com restrição não-holonômica.

Espaço de estados, observações e ações:

Considere um robô móvel planar uniciclo de configuração $q = [x_r \quad y_r \quad \theta_r]^T$ onde x_r, y_r são as coordenadas da posição do robô e θ_r é orientação do robô, ambos com respeito a um referencial fixo, chamado *mundo*. A tarefa é levar o robô para uma configuração alvo fixa $q_d = [x_d \quad y_d \quad \theta_d]$, onde x_d, y_d são a posição do alvo e θ_d é orientação do alvo, ambos também com respeito ao referencial mundo. O cenário também pode conter obstáculos fixos que

o robô tenha que desviar. Por se tratar de um robô uniciclo, o mesmo tem uma restrição não-holonômica, logo, suas ações são $a = [v \quad \omega]$, onde v é a velocidade linear e ω é a velocidade angular do robô.

Para obter as informações dos obstáculos de forma prática, serão considerados sensores que tem *lasers* que podem ser distribuídos ao redor do robô, como mostra a figura 4.2, que possam medir a distancia de cada laser até o obstáculo mais próximo. O sensor tem a limitação no alcance que os lasers podem medir, mas é uma ferramenta prática para capturar a posição relativa dos obstáculos em relação ao robô.

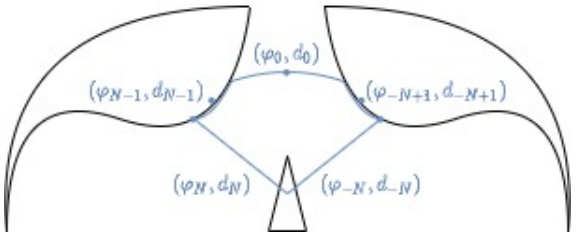

Figura 4.2: Distribuição dos sensores do robô. Sem perda de generalidade $\varphi_0 = 0$ (frente do robô). O robô é representado pelo triângulo, enquanto dois obstáculos estão sendo captados pelos sensores do robô. Fonte: autor.

Como é visto na figura 4.2, cada laser do sensor é indexado por um ângulo que varia de acordo com a direção do laser em relação ao eixo de coordenadas do robô. Considerando, sem perda de generalidade, que, $\varphi_0 = 0$ seja o feixe exatamente na frente no robô, então, todos os lasers estão no intervalo φ_{-N} até φ_N, onde N irá depender da aplicação. Além disso, a discretização do feixe de lasers do sensor também irá depender da aplicação. Cada laser terá uma distância associada, e portanto, será definido o par (φ_i, d_i), com $i = -N, ..., N$, que irá encapsular a distância medida pelo feixe e sua devida direção em relação ao robô. Nesta dissertação, d_i será normalizado entre 0 e 1.

O fato de utilizar esses sensores torna esse problema como parcialmente observável, trazendo a necessidade de considerar observações ao invés de estados. As distâncias medidas pelo *laser* serão consideradas *observações* que o ambiente irá externar, para que o agente consiga obter as informações dos obstáculos. Da mesma maneira, é necessário embutir nas observações as informações do alvo. Uma das possibilidades é computar a posição e a orientação do alvo com relação ao robô. Para isso, considere $\bar{p}$ e $\bar{\theta}$ como a posição e a orientação do alvo medidos a partir do referencial do robô, sendo calculados a partir de:

$$\begin{cases} \bar{p} = R(\theta)^T(p_d - p_r) \\ \bar{\theta} = \theta_r - \theta_d \end{cases} \tag{4.23}$$

onde $R(\theta)$ é uma matriz de rotação relativa a orientação θ do robô, $p_d = [x_d \quad y_d]^T$ é posição do alvo no referencial mundo e $p_r = [x_r \quad y_r]^T$ é a posição do robô no referencial mundo. É possível também considerar a posição do alvo em termos de coordenadas polares (distância e

fase).

Portanto, as observações deste sistema é dada pelo vetor o:

$$o = \begin{bmatrix} \bar{p} \\ \bar{\theta} \\ (\varphi_i, d_i) \end{bmatrix} \tag{4.24}$$

onde $-N \leq i \leq N$. Essa definição de observação é semelhante a definida nos trabalhos realizados por [Tai et al., 2017] e [Jesus et al., 2019]. As principais diferenças são: os autores consideram as informações do alvo como coordenadas polares ao invés de coordenadas cartesianas ($\bar{p}$); eles consideram apenas tarefas de posição, logo eles não incluem $\bar{\theta}$ como observação; e por fim, também consideram a ação de controle anterior também como observação.

Portanto, o agente receberia essas informações e, a partir delas, ele irá gerar uma ação, levando em consideração alguma política. Como será visto mais adiante, a recompensa acaba sendo função da observação, visto que a recompensa é uma função matemática que precisa ser formulada a partir de conhecimentos da tarefa e do sistema que o agente irá atuar. Portanto, depende das informações que o ambiente provê, que são as observações. O diagrama de blocos da figura 4.3 explicita essa relação.

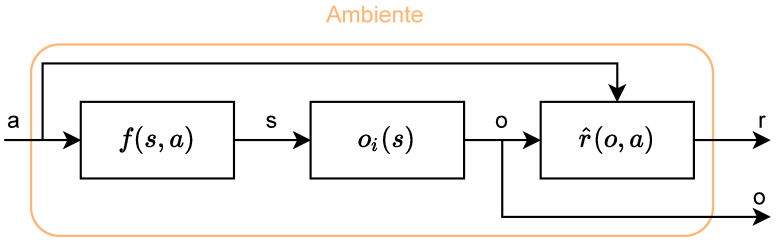

Figura 4.3: Diagrama de blocos que elucida os diferentes componentes do ambiente: função de transição de estados, recompensa e observação. A função de transição de estados e a recompensa não irão variar caso o cenário que o robô esteja inserido se modifique, porém a função da observação será alterada, ou pelo menos, re-parametrizada. Fonte: autor.

Como o alvo q_d é fixo e os obstáculos, se existem, são fixos, seria possível considerar como o estado do ambiente somente a configuração do robô $s = q$, porém ao fazer isso, toda vez que o cenário modificar, por exemplo, se a posição dos obstáculos for alterada, ou a pose do alvo for alterada, então o ambiente como um todo será alterado, uma vez que a função de observação irá alterar. Isso ocorre porque, como foi dito em 3.2, as observações são funções somente do estado, porém, se o alvo for modificado, então, a função da observação precisa ser *re-parametrizada* para essa modificação no alvo, o que resultará em um novo ambiente.

Por outro lado, é possível considerar as informações estruturais do ambiente no estado, juntamente com a configuração do robô. Para isso, a posição e orientação do alvo q_d precisam ser incluídas no vetor de estados, juntamente com as informações dos obstáculos. Assumindo que eles são círculos, e considerando esta premissa nesta dissertação, então a representação dos obstáculos é feita considerando o centro e o raio dos mesmos, sendo p_{obs1}, p_{obs2}, até p_{obsn} os

pontos centrais dos círculos e r_{obs1}, r_{obs2} até r_{obsn} o raio de cada círculo. Essa não é um premissa forte porque qualquer obstáculo pode ser arbitrariamente aproximado por círculos. Nota-se que os obstáculos inócuos para o sistema podem ser representados com o raio igual a zero. Outra premissa é que o sistema contém um número exato de n obstáculos, para que o vetor de estados seja finito e fixo. Neste caso, o estado do sistema pode ser definido como:

$$s = \begin{bmatrix} q & q_d & p_{obs_1} & r_{obs_1} & p_{obs_2} & r_{obs_2} & \dots & p_{obs_n} & r_{obs_n} \end{bmatrix}^T. \tag{4.25}$$

A definição apresentada de estado ajuda a explicar porque um agente treinado em um cenário, consegue resolver tarefas em outro cenário completamente diferente. Como dito anteriormente, cenários distintos podem ser vistos como estados, seguindo a equação (4.25). Como o arcabouço de aprendizado por reforço contém diversos fundamentos que propõem encontrar a política ótima, então, a descrição do ambiente com os estados contendo informações estruturais do cenário, são informações que serão utilizadas pelo agente, para chegar no alvo, mesmo em cenários não treinados. Além disso, caso o estado da equação (4.25) seja utilizado durante o treinamento, é esperado que o agente tenha uma maior capacidade de aprender a se deslocar em diferentes cenários. Para isso, a expectativa é que o tempo de treinamento necessário nesse caso seria muito maior do que o tempo de treinamento caso o agente fosse treinado em apenas um cenário.

Recompensa:

A recompensa proposta não é uma função contínua, visto que foi proposto recompensas extraordinárias caso o episódio termine com sucesso ou com falha. Em todos os outros casos, a recompensa $r(s,a)$ instantânea recebida pelo agente foi proposta como a soma de diferentes parcelas que estarão relacionadas com a aproximação do robô ao alvo e o evitamento de obstáculos pelo mesmo. Matematicamente, ela é definida como:

$$\begin{cases} r(s,a) = R_{sucesso}, \text{ se a tarefa for concluída com sucesso} \\ r(s,a) = R_{falha}, \text{ se o episódio terminar com falha} \\ r(s,a) = \alpha_1 \dot{d}_p(s,a) + \alpha_2 \dot{d}_o(s,a) + \alpha_3 \bar{R}_A(s,a) + \alpha_4 \bar{R}_B(s,a) + \alpha_5, \text{ caso contrário} \end{cases} \tag{4.26}$$

onde

- $\alpha_1 \dot{d}_p(s,a)$, com $\alpha_1 \in \mathbb{R}^-$, é parcela da recompensa relacionada com a tarefa de posição do robô. Definindo $d_p(s)$ como a distância entre a posição do robô e a posição alvo, sendo ela dada por

$$d_p(s) = \|\bar{p}\|_2 = \sqrt{(x_d - x_r)^2 + (y_d - y_r)^2} \tag{4.27}$$

então,

$$\dot{d}_p(s,a) = \frac{v\cos(\theta_r)(x_r - x_d) + v\sin(\theta_r)(y_r - y_d)}{\sqrt{(x_d - x_r)^2 + (y_d - y_r)^2}}. \tag{4.28}$$

Se $\dot{d}_p(s,a) < 0$, então a distância entre o robô e o alvo está diminuindo, o que resultará em uma recompensa positiva, porém, se $\dot{d}_p(s,a) > 0$, então a distância do robô e o alvo está aumentando, o que resultará em uma recompensa negativa. Portanto é necessário multiplicar $\dot{d}_p(s)$ por uma constante negativa.

- $\alpha_2\dot{d}_o(s,a)$, com $\alpha_2 \in \mathbb{R}^-$, é a parcela relacionada com a tarefa de orientação. Seja $d_o(s,a)$ como a varição na distância da orientação do robô à orientação alvo, medida no espaço de ângulos. Da mesma maneira que $\dot{d}_p(s,a)$, $\dot{d}_o(s,a)$ também precisa ser multiplicada por uma constante não-positiva, visto que uma redução na distância ($\dot{d}_o(s,a) < 0$), deve resultar em um aumento na recompensa. Definindo $d_o(s) = 1 - \cos(\theta_r - \theta_d)$, então,

$$\dot{d}_o(s,a) = \sin(\theta_r - \theta_d)\omega. \tag{4.29}$$

Nota-se que, $d_o = 0 \iff \theta_r = \theta_d$.

- $\alpha_3\bar{R}_A(s,a)$, com $\alpha_3 \in \mathbb{R}^+$, é uma penalização de proximidade do robô aos obstáculos, sendo dada por:

$$\bar{R}_A(s,a) = \left(\frac{sign(v)+1}{2}\right)\sqrt{|v|}\left[\frac{\sum_{i=-N_f}^{N_f}\cos(\varphi_i)d(\varphi_i)}{\sum_{i=-N_f}^{N_f}\cos(\varphi_i)} - 1\right] \tag{4.30}$$

onde, $[-N_f, N_f]$ é o intervalo de índices relativos ao feixe de lasers na direção frontal do robô e $sign(v) = \frac{v}{|v|}$. Nota-se que, por construção, $\bar{R}_A(s,a) \leq 0$. Quando não tem obstáculo no alcance dos sensores, a penalização da equação 4.30 é nula.

- $\alpha_4\bar{R}_B(s,a)$, com $\alpha_4 \in \mathbb{R}^+$, é definida de forma semelhante, porém considerando os índices relativos ao feixe de lasers na direção posterior do robô $[-N_r, N_r]$, sendo dada por:

$$\bar{R}_B(s,a) = \left(\frac{sign(v)-1}{2}\right)\sqrt{|v|}\left[\frac{\sum_{i=-N_r}^{N_r}\cos(\varphi_i)d(\varphi_i)}{\sum_{i=-N_r}^{N_r}\cos(\varphi_i)} - 1\right]. \tag{4.31}$$

Novamente, por construção, $\bar{R}_B(s,a) \leq 0$ e quando não tem obstáculo no alcance dos sensores, a penalização da equação 4.31 é igual a zero.

- $\alpha_5 \in \mathbb{R}^-$ é um escalar que irá penalizar o agente a cada passo de integração, caso ele fique parado longe de um obstáculo, pois neste cenário, com $\alpha_5 = 0$, o agente estando parado ($\dot{d}_p = \dot{d}_o = 0$) fora do alcance dos sensores do robô ($\bar{R}_A = \bar{R}_B = 0$), então a recompensa seria $R(s,a) = 0$, o que poderia representar uma vantagem para o agente em relação a ser penalizado por aumentar as distâncias. Portanto, com $\alpha_5 < 0$, então o agente

não terá esse conforto, sendo necessário explorar o ambiente, mesmo que isso represente aumentar as distâncias ao alvo.

- $R_{sucesso} \in \mathbb{R}^+$ e $R_{falha} \in \mathbb{R}^-$ são recompensas extraordinárias, para quando o episódio terminar. Ambas têm um valor alto em módulo, incentivando o agente a terminar as tarefas com sucesso, e penalizando os agente por terminar os episódios por falha.

A proposta de considerar a soma ponderada das variações das distâncias de posição e de orientação faz com que o agente busque resolver ambas tarefas concomitantemente. Naturalmente, pode existir um conflito entre essas variações, o que torna necessário priorizar uma das tarefas. De forma geral, é comum priorizar tarefas de posição, ainda mais que o modelo desse robô é de um uniciclo, o que possibilitaria fazer com que o robô seja capaz de girar em torno de si, e portanto corrigir eventuais erros de orientação sem prejudicar a posição. Para a recompensa proposta, essa priorização é feita por meio dos escalares α que multiplicam as variações. Em teoria, uma recompensa alternativa poderia realizar essa priorização de forma explícita, como por exemplo, desconsiderar a variação de orientação até que a tarefa de posição fosse concluída. No entanto, tal estratégia não foi considerada porque durante o trabalho existiu o interesse de aplicar esse agente também em sistemas cuja curvatura não é infinita, como automóveis. Desta forma, o ajuste dos pesos de cada variação pode viabilizar o agente também em tais robôs, o que não seria possível com a recompensa alternativa comentada.

As definições de $\bar{R}_A(s,a)$ e $\bar{R}_B(s,a)$ consideram a posição dos obstáculos em relação à direção na qual o robô está se deslocando. Caso o robô esteja se movendo para frente, ou seja $v > 0$, então $\bar{R}_A(s,a) \neq 0$ e $\bar{R}_B(s,a) = 0$. Caso o robô esteja se movendo para trás, ou seja $v < 0$, então $\bar{R}_A(s,a) = 0$ e $\bar{R}_B(s,a) \neq 0$. Um segundo aspecto relevante dessa definição é que, além de aumentar a penalização a medida que o robô se aproxima do obstáculo, ela pondera a penalização de acordo com a posição relativa do obstáculo em relação ao robô. Isso significa que a penalização para a aproximação de obstáculos na direção do movimento do robô é maior que a penalização pela aproximação de um obstáculo lateralmente ao robô, o que para um robô não-holonômico é uma característica relevante.

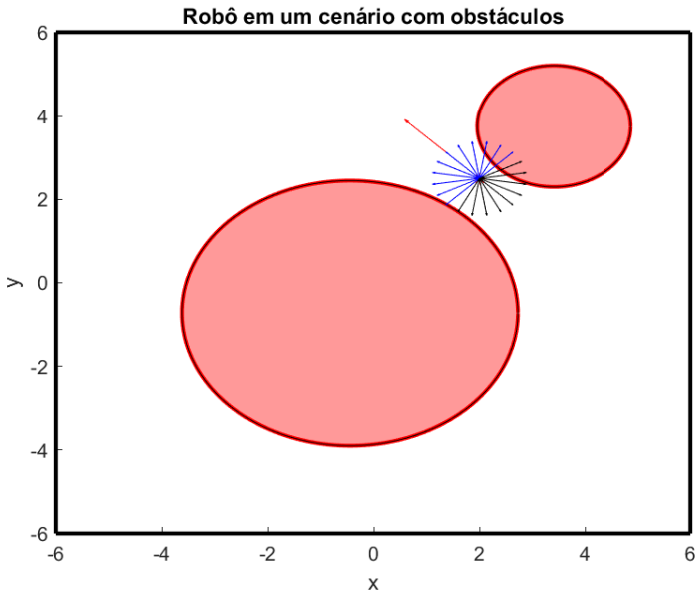

Figura 4.4: Robô próximo de obstáculos. Os obstáculos são representados pelos círculos vermelhos. A direção de movimento do robô é dada pela seta vermelha. Os lasers localizados na parte frontal do robô estão indicados de azul, enquanto os lasers que estão na parte traseira do robô estão indicados em preto. Cada seta tem um ângulo associado, onde a seta que está sobreposta pela seta vermelha representa o ângulo $\varphi_0 = 0$, e a seta em azul perpendicular à direita do robô representa o ângulo $\varphi_{-N} = -\pi/2$ e a seta em azul perpendicular à esquerda do robô representa o ângulo $\varphi_N = \pi/2$. Fonte: autor.

Analisando a figura 4.4, apesar do robô estar bem próximo do obstáculo menor, ele tem um caminho livre na direção do movimento, logo uma penalização considerando somente a leitura do sensor, como por exemplo proporcional à menor leitura obtida pelo sensor, é uma penalização muito dura para casos como esse, o que implicou em uma penalização ponderada pela posição (ângulo φ_i) do feixe do sensor que está fazendo a leitura, que representa a posição do obstáculo em relação a direção do movimento do robô.

Função de transição de estados:

A dinâmica deste sistema $\dot{s} = F(s, a)$ pode ser calculada de forma simples. As três primeiras componentes de $\dot{s}$ são dadas pela dinâmica de um uniciclo, e como todas as informações estruturais dos cenário (posição e orientação do alvo, posição e raio dos obstáculos) são constantes ao longo do tempo, então as derivadas temporais dessas componentes é igual a zero. Desta forma, o vetor $\dot{s}$ pode ser dado por,

$$\begin{bmatrix} \dot{x_r} \\ \dot{y_r} \\ \dot{\theta_r} \\ \dot{x_d} \\ \dot{y_d} \\ \dot{\theta_d} \\ \dot{x}_{obs_1} \\ \dot{y}_{obs_1} \\ \dot{r}_{obs_1} \\ \dot{x}_{obs_2} \\ \dot{y}_{obs_2} \\ \dot{r}_{obs_2} \\ \dots \\ \dot{x}_{obs_n} \\ \dot{y}_{obs_n} \\ \dot{r}_{obs_n} \end{bmatrix} = \begin{bmatrix} v\cos(\theta_r) \\ v\sin(\theta_r) \\ \omega \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \dots \\ 0 \\ 0 \\ 0 \end{bmatrix}. \tag{4.32}$$

A formulação do MDP da seção 3.2 implicitamente indica uma função de transição de estados discreta. Dado que a dinâmica do sistema é determinada pela equação (4.32) é necessário discretizar essa dinâmica para $s' = s + F(s,a)\Delta t$, onde Δt é o passo de integração. Portanto, a função de transição de estados do ambiente $p(s'|s,a)$ se trata de uma função determinística, dada por:

$$p(s'|s,a) = \begin{cases} 1, \text{ se } s' = s + F(s,a)\Delta t \\ 0, \text{ caso contrário.} \end{cases} \tag{4.33}$$

Condição de terminação do episódio:

Existem três condições para finalização de um episódio nesse ambiente:

- O robô chegar no alvo, com uma tolerância de λ_{pos} para a posição e λ_{ori} para a orientação. Nesse caso, o episódio terminou pelo agente ter completado a tarefa, resultando na recompensa $r(s,a) = R_{sucesso}$;
- O robô colidir com algum obstáculo do cenário, incluindo as paredes. Já nesse caso, o episódio terminou sem o agente ter completado a tarefa, ou seja, com uma falha, logo a recompensa será $r(s,a) = R_{falha}$;
- O robô parar em algum local por aproximadamente T_{stop_max} passos de integração. Essa condição de finalização do episódio visa penalizar o robô por ficar parado, assim como a parcela α_5 da recompensa descrita anteriormente. Portanto, o episódio terminou sem

o agente ter completado a tarefa, ou seja, novamente com uma falha, o que resulta na recompensa $r(s,a) = R_{falha}$.

Filtro na ação de controle:

É de se esperar que, à medida que o agente explore o ambiente, ele consiga aprender a desviar dos obstáculos, por conta da penalização proposta na função de recompensa. Todavia, até que isso aconteça, o agente tende a terminar episódios de forma muito rápida, o que pode ser prevenido caso existisse uma estratégia para realizar o evitamento de obstáculos para a rede neural. Essa ideia foi inspirada na camada de proteção proposta na seção 4.1.4 e ela objetiva ter a mesma característica para esses agentes baseados em redes neurais, para evitar o término do episódio de forma muito rápida, e assim buscar reduzir o número de episódios que o agente precisa ter para aprender uma política satisfatória. Portanto, para ajudar a rede neural na tarefa de desviar dos obstáculos, foi proposto um filtro que altera a ação a para uma ação segura a_{seg} que fará com que o robô desvie dos obstáculos.

O filtro consiste em modificar a velocidade angular ω, enquanto reduz a velocidade linear v, fazendo com que o desvio aconteça de forma mais suave possível. A ação segura $a_{seg}(s,a)$ é definida como:

$$a_{seg}(s,a) = \begin{bmatrix} h(s)v \\ \omega + \Delta\omega(s,a) \end{bmatrix}. \tag{4.34}$$

Com um $r \in \mathbb{R}^+$ e N é número de lasers que serão utilizados pela ponderação, a função $h(s)$ é dada por:

$$h(s) = \left(\prod_{i=-N}^{N} d_i \right)^r. \tag{4.35}$$

Como d_i é normalizado entre 0 e 1, é possível ver que $h(s) \in [0,1]$, o que fará com a velocidade v seja reduzida e nunca aumentada.

Já $\Delta\omega(s,a)$ é dado por:

$$\Delta\omega(s,a) = sign(v)\Delta w_{max}\frac{2}{\pi}\frac{\sum_{i=-N}^{N}(d_i)^2\varphi_i}{\sum_{i=-N}^{N}(d_i)^2} \tag{4.36}$$

onde Δw_{max} é o desvio máximo desejado, uma constante, e N é número de lasers que serão utilizados pela ponderação. Vale destacar que N para equação (4.36) pode ser diferente da equação (4.35).

Essa proposta da equação (4.34) resultará em um desvio na velocidade angular do robô ω enquanto a velocidade linear v é reduzida de forma a possibilitar o desvio do obstáculo. Esse desvio será computado a partir da leitura de distância que os lasers do sensor irão medir. Observando a figura 4.5, a distância medida pelos lasers em azul à esquerda do robô é menor que a distância medida pelos lasers em azul à direita do robô. A soma ponderada da equação (4.36) irá resultar em um ângulo médio ponderado, que resultaria em uma seta apontada para à direita do robô, que seria a direção do desvio a ser realizado pelo mesmo.

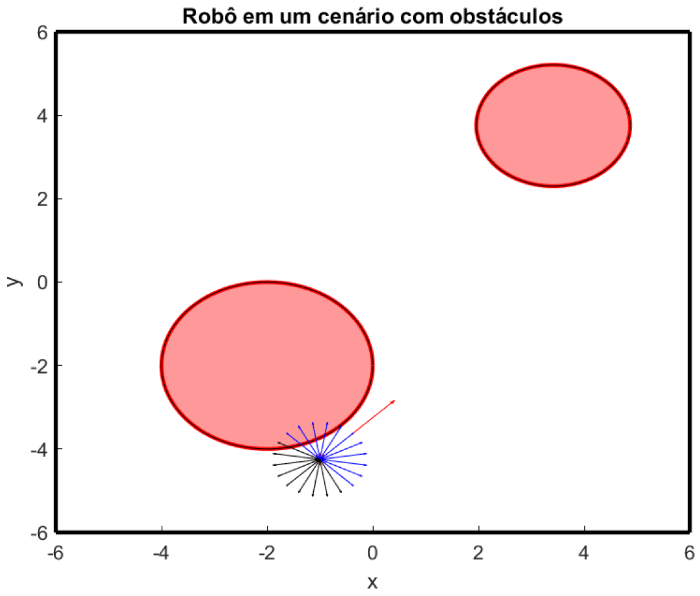

Figura 4.5: Leitura do sensor para o desvio. Novamente, os obstáculos são representados pelos círculos vermelhos. Fonte: autor.

Vale destacar que esse filtro é integrado na função de transição de estados do ambiente. Para isso, basta considerar

$$\dot{s} = F(s,a) \rightarrow \dot{s} = F(s, a_{seg}(s,a)) = \tilde{F}(s,a) \tag{4.37}$$

onde $F(s,a)$ é a dinâmica do sistema sem o filtro, descrita pela equação (4.32), e $\tilde{F}(s,a)$ é a dinâmica do sistema com o filtro, descrito pela equação (4.34).

Isso é importante porque o filtro proposto faz parte da descrição do ambiente, se tornando independente da escolha do agente que irá realizar o aprendizado. Além disso, não existe a necessidade de modificar a estrutura do agente para considerar o filtro. A figura 4.6 mostra em um diagrama de blocos a integração do filtro com a dinâmica do sistema, formando um novo sistema.

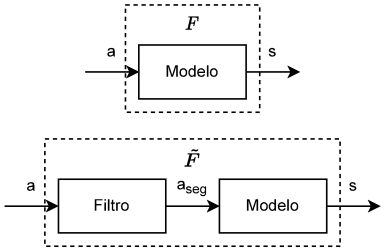

Figura 4.6: Diagrama de blocos indicando como o filtro pode ser incluído dentro do modelo do ambiente. Fonte: autor.

4.3 Resumo do capítulo

Este capítulo apresentou duas abordagens para realizar a navegação de robôs utilizando as ferramentas descritas no capítulo anterior. A primeira abordagem visa resolver o problema de

navegação de robôs sem a restrição de não-holonomia, propondo um agente que utiliza a *iteração de valor na álgebra max-plus* para estimar uma política à medida que o robô se movimenta no cenário. Todos os detalhes necessários para a implementação deste agente foram descritos, além de uma discussão sobre suas vantagens sobre a segunda abordagem. Essa segunda abordagem por outro lado, visa resolver o problema de navegação de robôs com restrição de não-holonomia. Para isso, é sugerido a utilização de agentes propostos na área de aprendizado por reforço profundo, no caso o DDPG e o SAC. Além disso, foi necessário formular o problema de navegação como um MDP, utilizando os conceitos apresentados no capítulo anterior. A formulação proposta tem a característica de mostrar como é possível um agente aprender a navegar em um cenário, e utilizar essa política também para navegar em outros cenários, com disposições diferentes de obstáculos e alvos.

Capítulo 5

Experimentos

Neste capítulo serão apresentadas as simulações das duas abordagens descritas anteriormente: por redes neurais profundas e por *iteração de valor*.

As simulações da seção 4.2 foram realizadas no MATLAB 2021a em um Intel Core i5-9300H CPU @ 2,40GHz, com 8GB de memória RAM, com sistema operacional Windows 10 e GPU GeForce GTX 1680 com 4GB de memória dedicada. Já as simulações da seção 4.1 foram retiradas do artigo [Júnior et al., 2021].

5.1 Resultados utilizando a abordagem 1

Para mostrar os resultados referentes ao agente proposto na seção 4.1, foram propostas três simulações diferentes:

1. A primeira foi feita em um cenário mais desafiador, com vários obstáculos dispostos no ambiente 2D, referenciada a seguir como *simulação 1*.

2. A segunda foi feita em um cenário mais simples, com poucos obstáculos no ambiente planar, referenciada a seguir como *simulação 2*.

3. A terceira foi feita em um cenário 3D, também desafiador, que será referenciada como *simulação 3*.

5.1.1 Parâmetros e configuração

Para as simulações 1 e 2, foi utilizada um robô móvel de configuração $q = [q_1 \;\; q_2]^T$, enquanto para a simulação 3 foi utilizada a configuração $q = [q_1 \;\; q_2 \;\; q_3]^T$. Essa configuração é a posição do robô em um sistema de coordenadas fixo. Em todos os cenários foi considerado um robô pontual porque foi assumido que os obstáculos estão aumentados. Todos os obstáculos têm formato circular/esférico, cada um com diferentes tamanhos. A função de distância para os obstáculos foi simplesmente calculada como distância de um ponto até um círculo e a distância

de um ponto até uma linha (para os limites dos cenários). Todos os cenários tem somente um alvo $\mathcal{S}_{tg}$ também representado por um círculo/esfera.

Foi utilizado $K = 10$ (sub-seção 4.1.2), $N = 5$ (sub-seção 4.1.3), $\eta = 0{,}1\ s^{-1}$ para obstáculos circulares/esféricos e $\eta = 1\ s^{-1}$ para os limites do cenário (sub-seção 4.1.4). O procedimento de amostragem foi realizado escolhendo $D = 0{,}5\ m$, $T = 1\ s$, $M = 1$, $P = 1$ e $h = 0{,}8\ m$ (sub-seção 4.1.5).

A perturbação w é gerada pela amostragem de um vetor a partir de uma distribuição normal multi-dimensional no $\mathbb{R}^2/\mathbb{R}^3$ com média igual ao vetor zero e matriz de covariância igual a matriz identidade. Então, esse vetor é normalizado e ponderado por um fator que inicialmente é igual a $2\ m/s$ e decresce linearmente com o tempo, reduzindo 20% a cada $5\ s$ (sub-seção 4.1.6). A ação de controle $\tilde{\mu}^*(q)$ é sempre normalizada, então precisa ser convertida para a velocidade real. Foi escolhido que essa velocidade é $1\ m/s$. Por fim, foi escolhido como método da integração a integração de Euler de primeira ordem com $dt = 0{,}05\ s$.

Toda vez que o robô chega em $\mathcal{S}_{tg}$, o robô é "teletransportado" para uma configuração livre de colisões aleatória, utilizando uma distribuição uniforme. Em uma aplicação real, seria necessário ter uma estratégia para garantir que o robô terá a oportunidade de explorar $\mathcal{S}_{free}$. Cada cenário foi simulado pela mesma quantidade de tempo.

5.1.2 Validação da Metodologia

De forma a validar a metodologia e mostrar que o robô está realmente aprendendo, para cada cenário, foram amostrados 100 pontos uniformemente em $\mathcal{S}_{free}$ e com eles, o seguinte procedimento de *validação* foi proposto: é simulada a trajetória percorrida pelo robô através da política $\mathbb{\mu}$ em diferentes instantes de tempo da simulação e considerando como configuração inicial cada um dos 100 pontos. Essas trajetórias consideram $w(t) = 0\ \forall\ t \geq 0$, ou seja, sem perturbação de exploração. Com essas trajetórias, foram computadas as distâncias percorridas para o conjunto alvo e essas distâncias são guardadas. Caso o robô não conseguir chegar no conjunto alvo (isto é, a ação de controle de cada instante de tempo falha em guiar o robô para $\mathcal{S}_{tg}$), a distância para aquela configuração inicial é considerada como ∞. Com as 100 distâncias computadas, duas métricas serão calculadas: a taxa de sucesso ao longo do tempo, que é o percentual das distâncias que não é ∞ (medindo a frequência com que a política estimada foi suficiente para finalizar a tarefa com sucesso) e a média log das distâncias, ou seja, quão ótimas foram as distâncias. A média log foi escolhida porque ela também considera "distâncias infinitas", enquanto uma média aritmética comum se tornaria infinita. A média log é dada por:

$$L = -\ln\left(\frac{\sum_i^{100} e^{-d_i}}{100}\right) \tag{5.1}$$

onde d_i é a i-ésima distância em metros, considerada sem dimensão quando usada na média. Ambas as métricas serão mostradas ao longo do tempo, porém é importante destacar que esses instantes de tempo não representam o tempo consumido pelo computador durante a simulação,

mas um *tempo "real" simulado*, que representaria o tempo real que o robô gastaria para produzir esses resultados.

5.1.3 Resultados

Cenário 1:

O cenário é mostrado na figura 5.1, junto com os pontos $\mathbb{s}_i$'s e vetores $\mathbb{\mu}_i$'s após duas horas de tempo simulado (tempo "real" do robô). É possível observar algumas políticas ainda apontando para o alvo, muitas vezes com obstáculos no caminho, porque o algoritmo ainda não foi capaz de corrigir todas as políticas do conjunto $\mathcal{S}_{dis}$ corretamente. Esse cenário visa validar a proposta em um ambiente bi-dimensional com diversos obstáculos, que pode ser desafiador para o robô, especialmente usando controladores por campos vetoriais tradicionais.

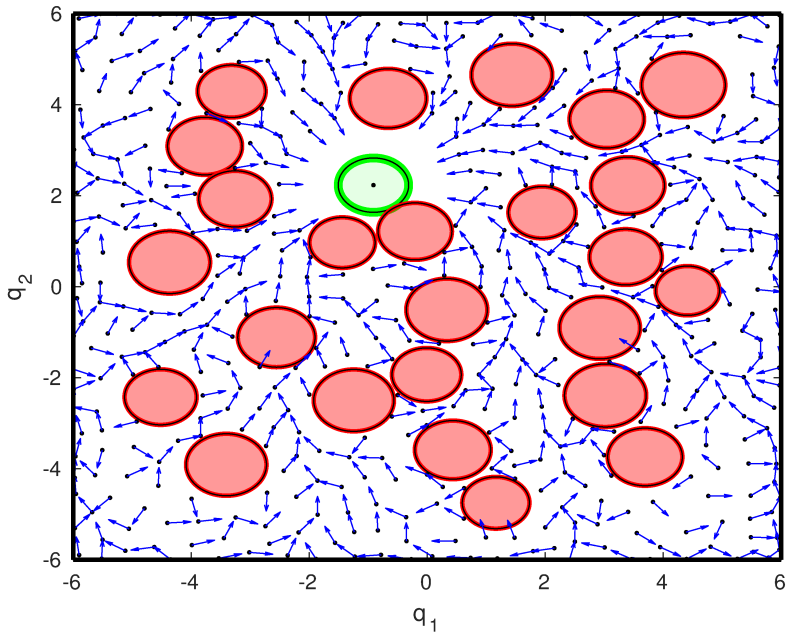

Figura 5.1: **Cenário 1**, com vários obstáculos ao redor do alvo. $\mathbb{s}_i$, $\mathbb{\mu}_i$ também são mostrados, com 496 elementos em cada conjunto. Fonte: autor.

A figura 5.2 mostra a evolução das métricas mencionadas anteriormente: a taxa de sucesso e a média log da distância. A taxa de sucesso é maior que 90% após 820 s, e é ainda melhor, acima de 95%, após 1860 s. Então, ele precisa de menos de 15 minutos para alcançar uma boa taxa de sucesso e ele precisa de somente 31 minutos para ser capaz de sair de praticamente qualquer configuração para o alvo.

Para destacar a importância da proposta de inicialização (seção 4.1.2), a figura 5.3 mostra a performance usando uma inicialização aleatória dos conjuntos $\mathbb{V}_i$, $\mathbb{\mu}_i$. Isso mostra que o robô precisa de muito mais tempo para alcançar bons resultados, uma vez que somente após aproximadamente 3000 s (quase uma hora) a taxa de sucesso é igual a 90%.

O tempo gasto para calcular a ação de controle é, em média, 5 ms.

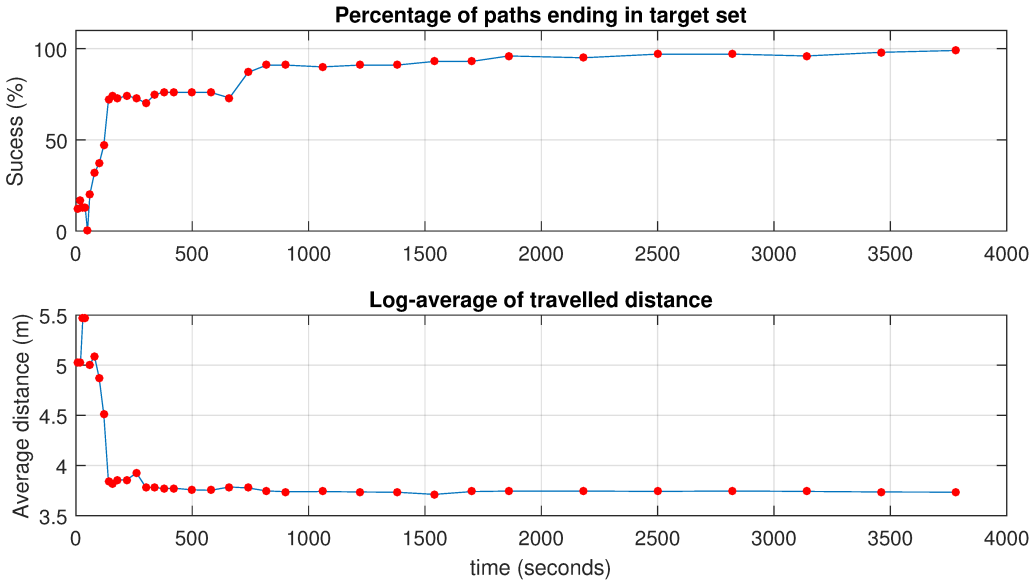

Figura 5.2: Performance do cenário 1. O gráfico superior mostra a taxa de sucesso enquanto o gráfico inferior mostra a média log da distância. Fonte: autor.

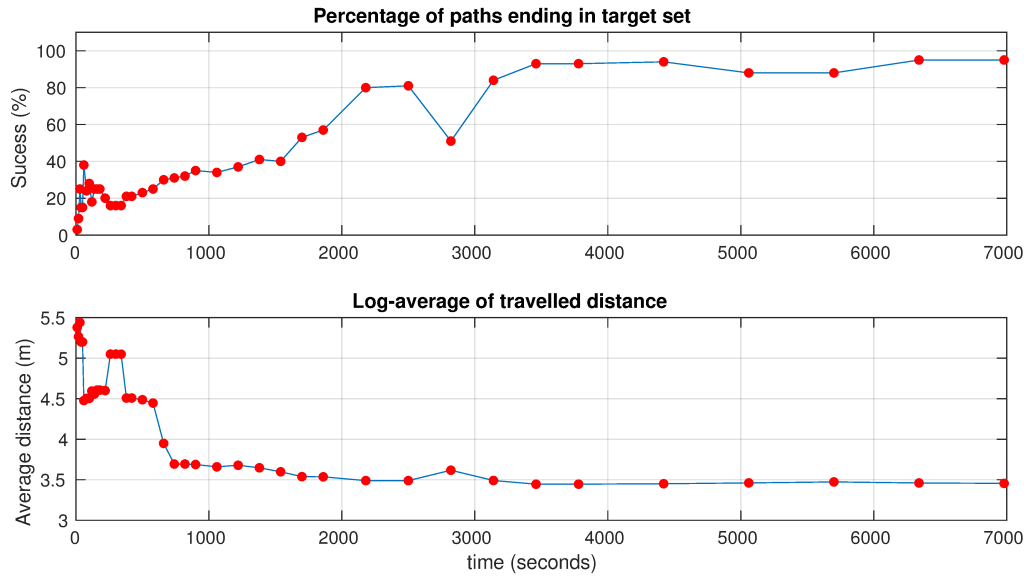

Figura 5.3: Performance do **cenário 1 com inicialização aleatória**. O gráfico superior mostra a taxa de sucesso enquanto o gráfico inferior mostra a média log da distância. Fonte: autor.

Cenário 2:

O segundo cenário simulado é um ambiente bi-dimensional com somente dois obstáculos, representando uma tarefa mais simples para o robô. Esse cenário é representado pela figura 5.4, junto com a política $\mathbb{\mu}$'s nos elementos de $\mathcal{S}_{dis}$, mostrados a partir de duas horas do tempo simulado (tempo "real" do robô).

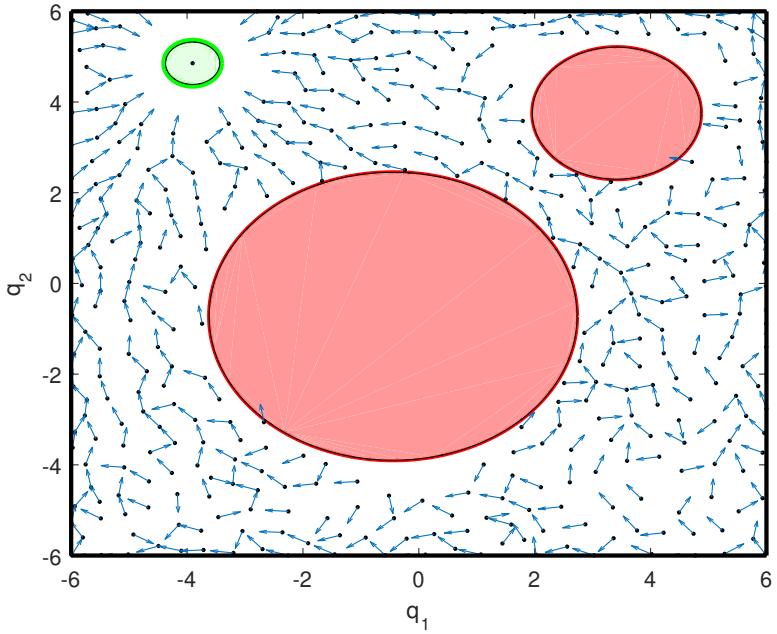

Figura 5.4: **Cenário 2**, com poucos obstáculos ao redor do alvo. $\mathbb{s}_i$, $\mathbb{\mu}_i$ também são mostrados, com 405 elementos em cada conjunto. Fonte: autor.

Pela figura 5.5, para esse cenário mais simples, a taxa de sucesso atinge 100% após 100 s, isto é, ele requer somente um minuto e quarenta segundos para ser capaz de ir de todas as configurações iniciais propostas até o alvo. A média log das distâncias requer um pouco mais de tempo para reduzir e alcançar o tempo ótimo, aproximadamente após 200 s, ou três minutos.

Com uma inicialização aleatória dos conjuntos $\mathbb{V}_i$, $\mathbb{\mu}_i$, a figura 5.6 mostra que, para esse cenário o tempo gasto para atingir 90% na taxa de sucesso é de aproximadamente 7000 s (quase duas horas), que é um tempo muito maior que o gasto pela inicialização proposta. Apesar disso, o agente foi capaz de corrigir a política inicial, o que mostra a efetividade da proposta.

O tempo gasto para calcular a ação de controle é, na média, 5 ms.

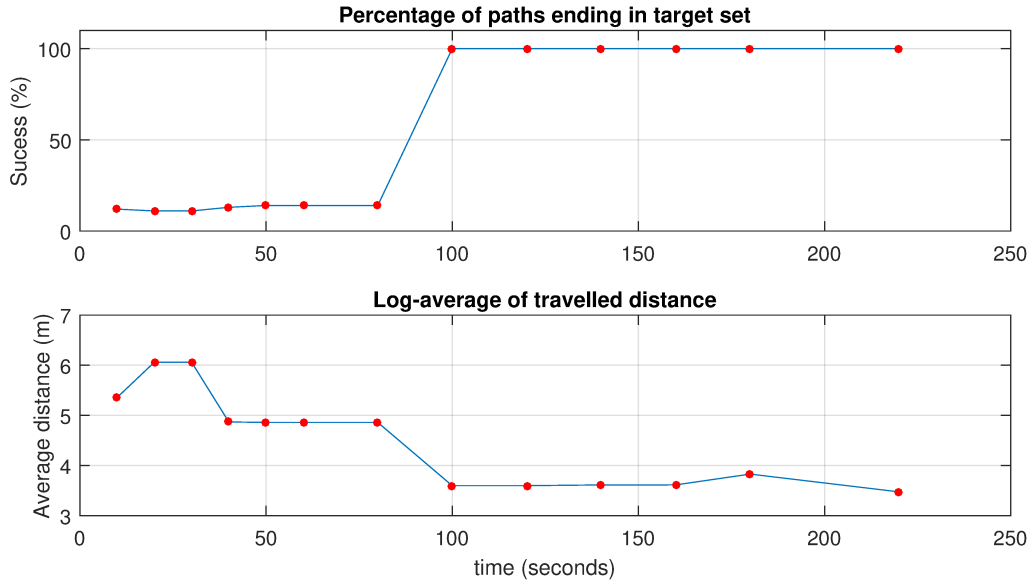

Figura 5.5: Performance do **cenário 2**. O gráfico superior mostra a taxa de sucesso enquanto o gráfico inferior mostra a média log da distância. Fonte: autor.

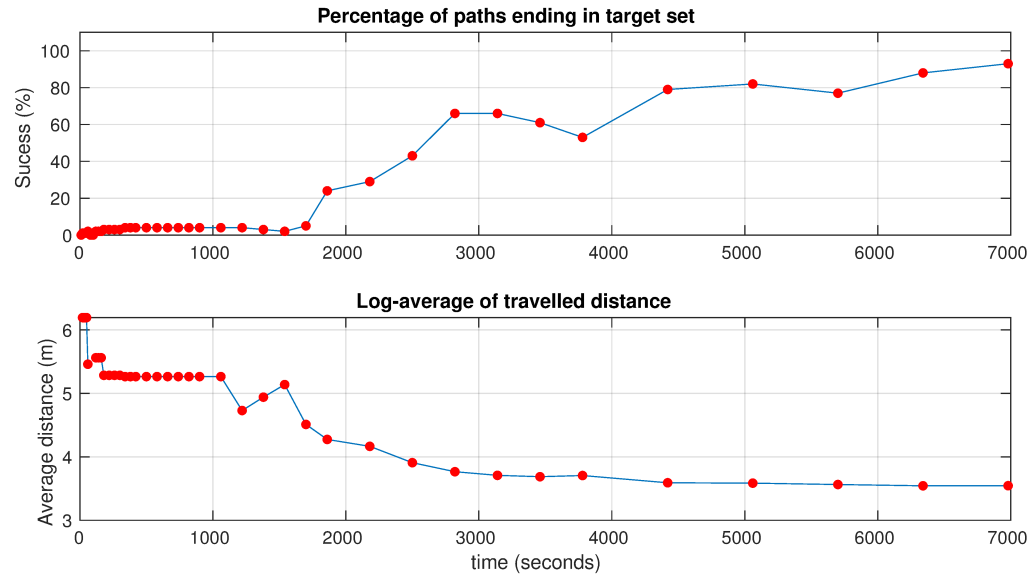

Figura 5.6: Performance do **cenário 2 com inicialização aleatória**. O gráfico superior mostra a taxa de sucesso enquanto o gráfico inferior mostra a média log da distância. Fonte: autor.

Cenário 3:

Este cenário é mostrado na figura 5.7. Ele é uma versão tri-dimensional do **cenário 1**. O objetivo de mostrar esse cenário é para atestar a capacidade do agente em problemas tri-dimensionais, que geralmente está associado a novos problemas.

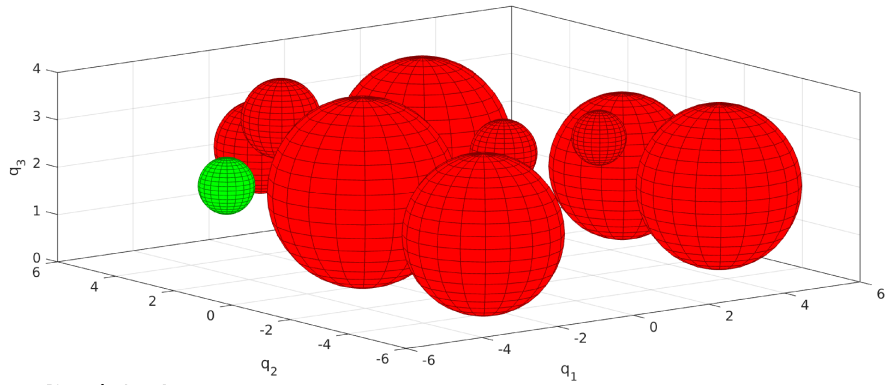

Figura 5.7: **Cenário 3**, com vários obstáculos ao redor do alvo. Os conjuntos $\mathbb{s}_i$, $\mathbb{\mu}_i$ não serão mostrados devido a dificuldade de vê-los claramente em 3D. Existem 1930 elementos em cada conjunto. Fonte: autor.

Ao analisar a figura 5.8, é possível observar que a taxa de sucesso atinge 100% após $50\ s$, o que é uma boa performance. Porém, a média log da distância mostra que é necessário $1200\ s$ (vinte minutos) para estabilizar.

O tempo gasto para calcular a ação de controle é, na média, $5\ ms$.

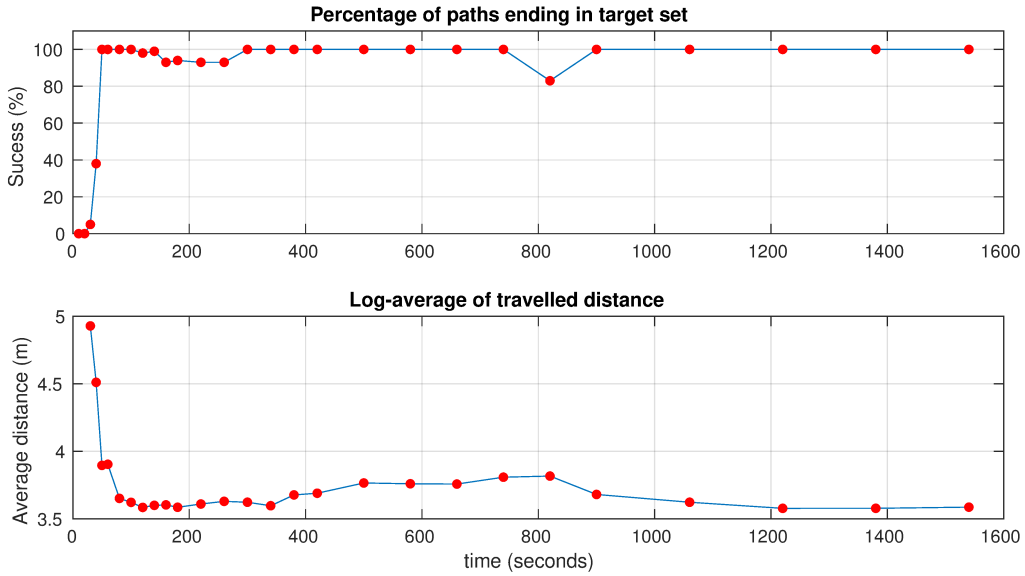

Figura 5.8: Performance do cenário 3. O gráfico superior mostra a taxa de sucesso enquanto o gráfico inferior mostra a média log da distância. Fonte: autor.

5.2 Resultados utilizando a abordagem 2

O próximo conjunto de testes é para o robô com a restrição de não-holonomia com agentes que utilizam redes neurais profundas como funções de aproximação, no caso o DDPG e o SAC apresentados na seção 4.2.

5.2.1 Parâmetros e configuração

Foi considerado um robô pontual com orientação, porque foi assumido que os obstáculos estão aumentados. Todos os obstáculos têm formato circular, cada um com diferentes tamanhos. A função de distância para os obstáculos foi simplesmente calculada como a distância de um ponto até um círculo e a distância de um ponto até uma linha (para os limites dos cenários). Os cenários têm dimensão 12×12 metros quadrados.

O sensor do robô tem 20 lasers de alcance de $1\ m$, tendo 18 graus de espaçamento entre eles. O passo de integração utilizado foi $\Delta t = 0{,}1\ s$. A velocidade máxima que o robô pode atuar é $v \in [-0{,}5\ 0{,}5]\ m/s$ e $\omega \in [-2{,}5\ 2{,}5]\ rad/s$. As tolerâncias para a conclusão da tarefa foram escolhidas como $\lambda_{pos} = 0{,}02\ m$ e $\lambda_{ori} = 0{,}02\ rad$. Os parâmetros da recompensa são $\alpha_1 = -2$, $\alpha_2 = -0{,}2$, $\alpha_3 = 0{,}8$, $\alpha_4 = 0{,}8$ e $\alpha_5 = -1$ (sub-seção 4.2.2). Para detectar que o robô parou, foi proposto o fluxograma mostrado na figura 5.9. Os parâmetros desse fluxograma escolhidos foram $\alpha_f = 0{,}97$, $a_{min} = 0{,}05$, $a_{tol} = 0{,}05$, $T_{stop\_max} = 15\ s$. As recompensas quando o episódio termina foram definidas como $R_{sucesso} = 100$ e $R_{falha} = -100$ (sub-seção 4.2.2). O filtro que realiza o evitamento de obstáculos tem os seguintes parâmetros selecionados $\Delta\omega_{max} = 0{,}5$ e $r = \frac{1}{3}$ (sub-seção 4.2.2).

Foi utilizado o *toolbox* de *Reinforcement Learning* do Matlab [The MathWorks, 2019] para realizar as simulações desta seção, que oferece um completo suporte para simulações de AR. Esse suporte inclui a pré-implementação de vários agentes, dentre eles o DDPG e o SAC, de alguns tipos de ambientes clássicos de AR, e de funções para o treinamento e validação do sistema. A pré-implementação dos agentes já contempla uma estrutura de redes neurais pré-definida, assim como os vários hiper-parâmetros das redes neurais e dos agentes, sendo possível customizar essas redes neurais e os hiper-parâmetros. Importante ressaltar que durante o mestrado ambos foram alterados, buscando entender o comportamento dos agentes com tais modificações. Porém, para os resultados apresentados, foi escolhido manter as estruturas das redes neurais pré-implementadas e variar os hiper-parâmetros que serão listados a seguir. Além disso, para essa aplicação, foi necessário customizar o ambiente para representar um robô com a restrição de não-holonomia, assim como, simular os sensores laser do robô e o filtro descrito na seção 4.2.2.

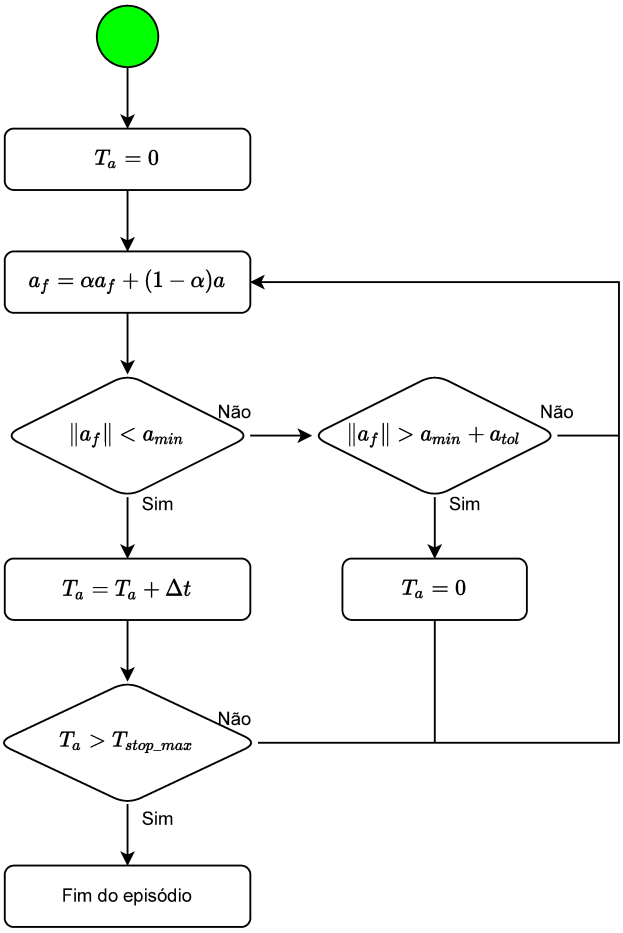

Figura 5.9: Fluxograma descrevendo a condição de finalização do episódio por parada do robô. A ação a passa por um filtro passa-baixa de primeira ordem e caso esteja abaixo de um limiar, então significa que o robô estará parado. Caso o robô fique parado (velocidade muito baixa) por T_{stop_max}, então o episódio termina. Fonte: autor.

Como descrito anteriormente, apenas os hiper-parâmetros a seguir foram alterados para os resultados apresentados. A taxa de aprendizagem para o ator dos agentes SAC e DDPG foi $0{,}0001$. Por outro lado, a taxa de aprendizagem para o críticos dos agentes foi $0{,}001$. O *experience buffer* para os agentes escolhido foi com 50000 transições, e o tamanho do *mini-batch* foi de 512 transições. Por fim, para o agente SAC, o número de passos antes de atualizar o ator e o crítico foi 512.

5.2.2 Redes neurais utilizadas

A seção 4.2.1 contém diversos detalhes a respeito da definição dos agentes DDPG e SAC. Esses agentes contêm funções de aproximação para os atores e críticos que são redes neurais profundas. O livro [Goodfellow et al., 2016] contém diversas informações sobre esses e vários outros tipos de redes neurais.

Para o DDPG, ambas as redes neurais do ator e do crítico estão apresentadas na figura 5.10. A rede do ator é composta por duas camadas escondidas densas, seguida da camada de saída. Essa rede recebe a observação como entrada e tem a ação que o agente irá realizar como saída. Já a rede do crítico tem duas entradas distintas, uma para a observação e outra para receber a ação do ator. Ambas entradas são seguidas de camada escondidas densas. Então, as saídas das camada densas são concatenadas, que sucede em mais uma camada densa, e finaliza em uma camada de saída que retornará um escalar, que é a função $Q^{\pi}(s, a)$. Todas as camadas escondidas e de concatenação tem função de ativação ReLU. As camadas escondidas têm 256 neurônios, enquanto a camada de concatenação tem 512 neurônios. Já as camadas de saída do ator e do crítico têm funções de ativação tangente hiperbólica e linear, respectivamente.

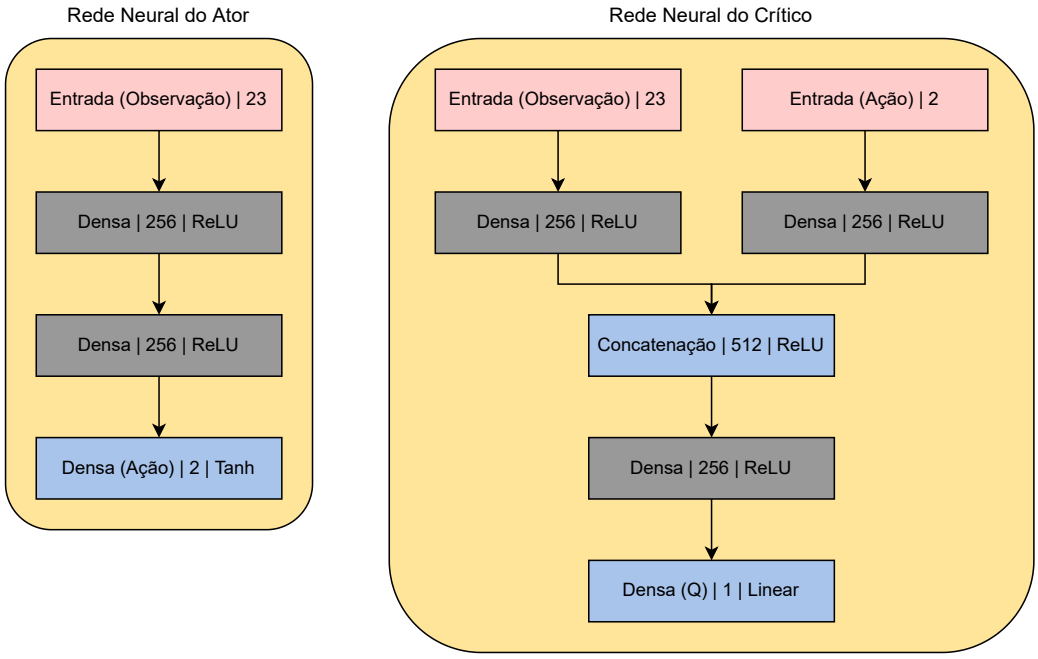

Figura 5.10: Redes neurais do ator e do crítico do DDPG. Fonte: autor.

As redes do SAC são mostradas na figura 5.11. A rede do crítico do SAC tem a mesma arquitetura, número de neurônios e mesma funções de ativação do crítico do DDPG apresentada na figura 5.10. Nota-se que ambas estimam a função $Q^{\pi}(s, a)$. Por outro lado, a rede do ator do SAC é mais complexa que a anteriormente descrita para o DDPG. A rede do ator irá receber as observações como entrada, que será seguida de duas camadas escondidas densas de 256 neurônios cada e função de ativação ReLU. Então, como o SAC é um ator estocástico, a implementação do Matlab considera que a saída da rede irá retornar os parâmetros de uma

gaussiana (média e desvio padrão) para cada elemento do vetor de ação $a = [v \quad \omega]$. Portanto, após as duas camadas densas, a rede do ator se divide em dois ramos, um contendo uma camada densa de 2 neurônios somente e função de ativação linear, para computar a média de cada gaussiana, e o outro ramo também contendo uma camada densa de 2 neurônios, mas com função de ativação softplus, para computar o desvio padrão de cada gaussiana. A camada softplus irá garantir que o desvio padrão sempre será maior que zero. Por fim, a rede tem uma camada de concatenação na saída que agrupa os pares de média e desvio padrão dos elementos de a. Essa camada de concatenação tem função de ativação linear.

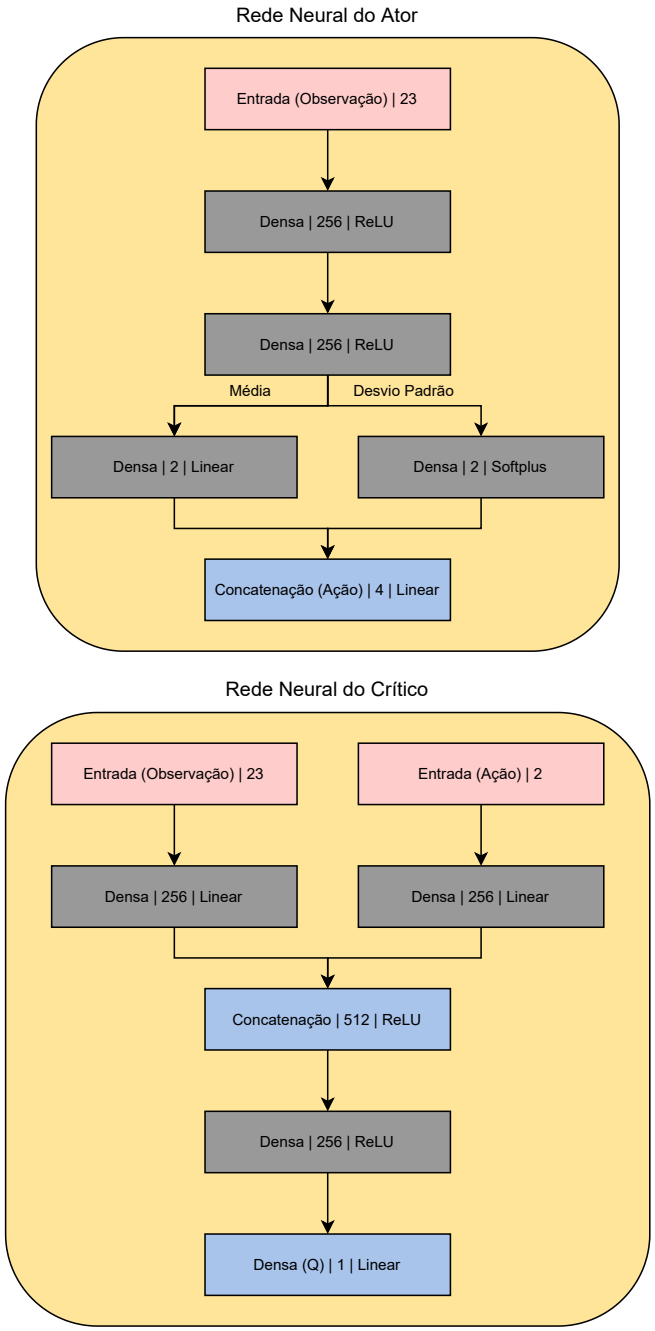

Figura 5.11: Redes neurais do ator e do crítico do SAC. Fonte: autor.

5.2.3 Validação da Metodologia

A proposta de validação dessa abordagem ocorrerá de duas formas: a primeira irá buscar entender o comportamento de agente caso o treinamento ocorra em somente um cenário, e a segunda buscará entender seu comportamento quando o treinamento ocorre em múltiplos cenários.

A primeira proposta de validação será:

- **Treinamento:** o treinamento ocorrerá em apenas um cenário. O agente será submetido a N_t episódios de treinamento, onde em cada episódio o robô irá iniciar de diferentes configurações do espaço livre. Em todos os episódios, os obstáculos e o alvo serão os mesmos, e eles ficarão fixos durante o episódio.
- **Validação:** a validação ocorrerá em seis diferentes cenários. Um dos seis cenários será o que foi utilizado durante o treinamento, mas os outros cinco serão totalmente novos para o agente. O agente será submetido a N_v episódios de validação para cada um dos cenários, onde novamente, somente a configuração inicial do robô irá se alterar a cada episódio, sendo aleatoriamente escolhida.

Desta forma, essa proposta de validação visará avaliar se o agente é capaz de, ao aprender a chegar no alvo desviando de obstáculos em um cenário, também ser capaz de desviar de obstáculos para chegar em alvos diferentes em outros cenários. As figuras 5.12 e 5.13 contém os seis cenários propostos. Quatro deles têm poucos obstáculos, que serão referenciados a seguir como "cenários simples", e os outros dois possuem diversos outros obstáculos, que serão referenciados como "cenários complexos". O cenário de treinamento dos agentes será o cenário simples 3. Para o SAC, foi escolhido $N_t = 100$, enquanto para o DDPG, foi escolhido $N_t = 300$. Como são seis cenários de validação, $N_v = 6$.

A segunda proposta de validação será:

- **Treinamento:** o agente terá N_t episódios, onde em cada episódio, o cenário que agente irá navegar será completamente diferente. Portanto, além da configuração inicial do agente variar a cada início de episódio, a disposição dos obstáculos, o tamanho dos obstáculos e a disposição do alvo também se alterará. Cada cenário será inicializado de forma aleatória, definindo uma instância do vetor de estados (4.25).
- **Validação:** o agente será submetido a N_{v1} episódios em N_{v2} cenários distintos. Portanto, a validação ocorrerá da mesma maneira que a primeira proposta, porém com uma quantidade muito maior de cenários de validação. Novamente, em cada um das N_{v1} simulações em cada cenário, somente a configuração inicial do robô irá se alterar no início de cada episódio.

Portanto, essa segunda proposta de treinamento irá explorar a capacidade do agente em visitar muitos outros estados do ambiente do que na primeira proposta. A expectativa é que após uma certa quantidade de episódios de treinamento, o agente seja capaz de alcançar qualquer

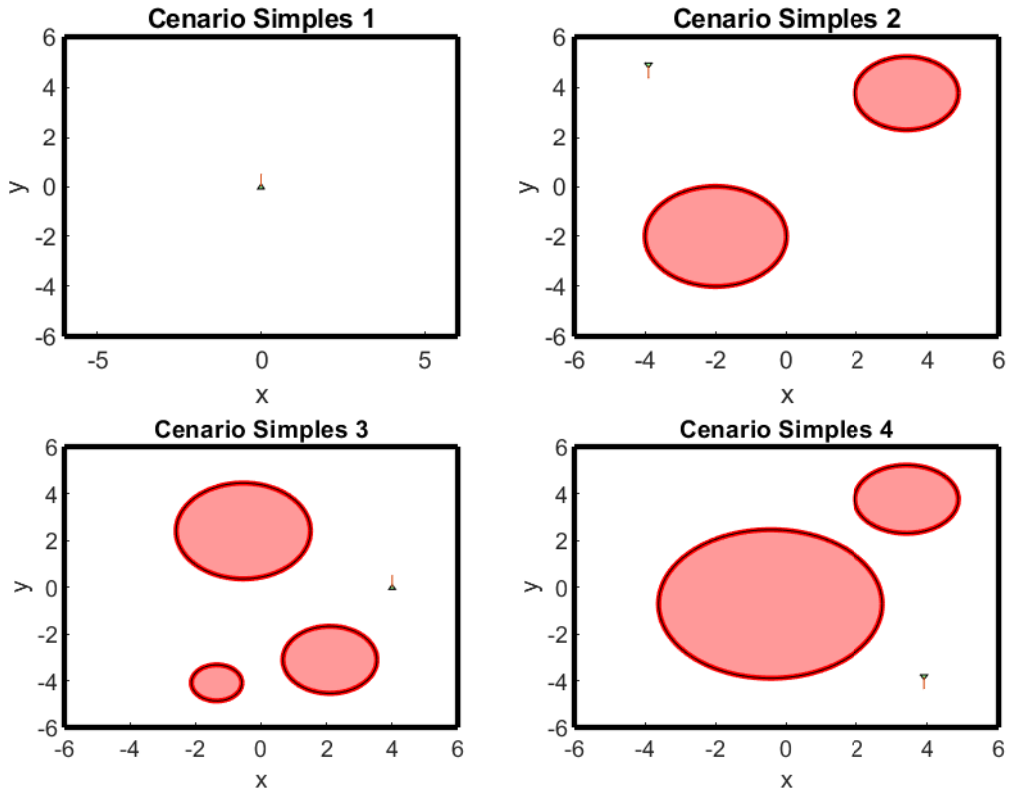

Figura 5.12: Cenários “simples” simulados na primeira proposta de validação. Os círculos vermelhos representam os obstáculos e os triângulos verdes representam os alvos. A orientação do alvo é representada pela seta no centro de cada triângulo. Fonte: autor.

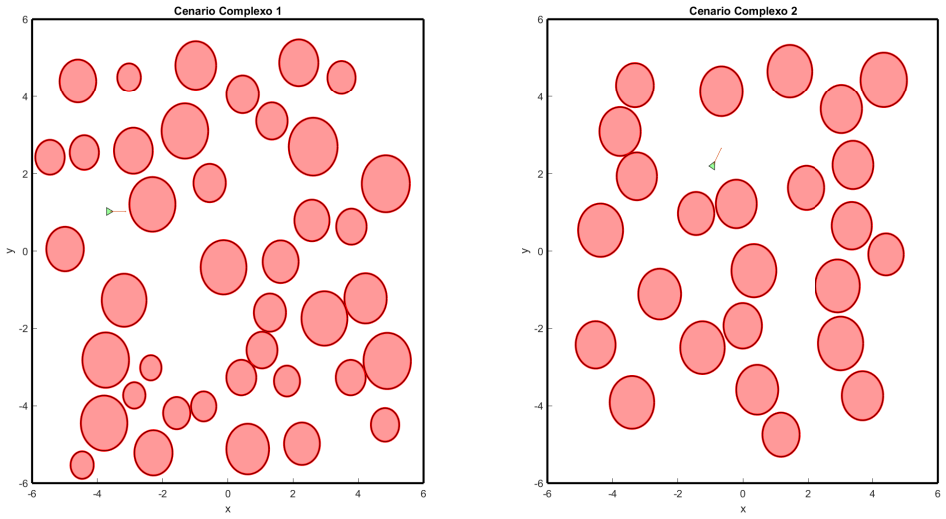

Figura 5.13: Cenários “complexos” simulados na primeira proposta de validação. Novamente, os círculos vermelhos representam os obstáculos e os triângulos verdes representam os alvos. A orientação do alvo é representada pela seta no centro de cada triângulo. Fonte: autor.

configuração desejada, partindo de diferentes configurações iniciais e desviando de qualquer tipo de obstáculos. Foi escolhido $N_t = 100$, $N_{v1} = 100$ e $N_{v2} = 300$.

A métrica da validação será o percentual de configurações iniciais em que o agente consegue completar a tarefa com sucesso, ou seja, em que o agente leva o robô ao alvo desviando dos obstáculos. Isso indicará se a política aprendida pelo agente é realmente capaz de completar tarefas. Como o treinamento da rede neural é feito através da função *train* do Matlab, assim como a implementação dos agentes DDPG e SAC, então não seria simples fazer uma comparação evolutiva do aprendizado dos agentes.

5.2.4 Resultados

Primeiro, serão apresentados os resultados para a primeira proposta de validação. A figura 5.14 mostra o percentual de vezes que os agentes SAC e o DDPG concluíram a tarefa chegando na pose desejada, partindo das N_v configurações iniciais descritas anteriormente. Esse percentual será chamado de percentual de sucesso.

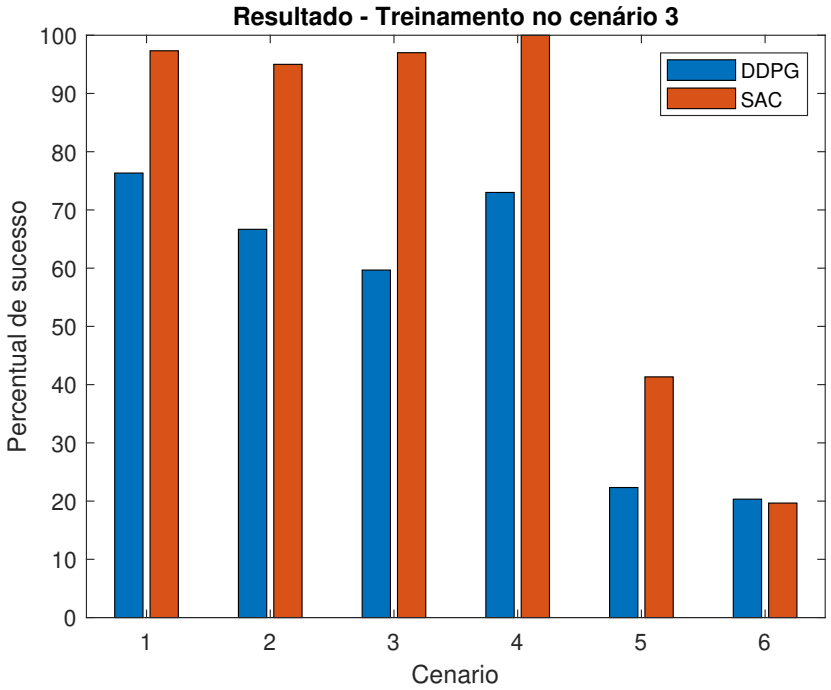

Figura 5.14: Primeira proposta de validação: percentual de sucesso do agente SAC em cada cenário para tarefas de pose completa. O gráfico em azul mostra o percentual para o DDPG, enquanto o gráfico vermelho mostra o resultado para o SAC. Fonte: autor.

A primeira impressão que o gráfico passa é que o SAC aparenta ter uma performance melhor que o DDPG, o que é comentado na seção 4.2. As possíveis explicações para isso são: primeiro porque o DDPG é um pouco mais complicado de se treinar, visto que o algoritmo é sensível à estrutura da rede neural e aos seus hiper-parâmetros. Além disso, função de recompensa também tem um fator muito importante no sucesso do algoritmo. Como é de conhecimento da

área, construir uma função de recompensa, também conhecido como *reward shapping*, é uma tarefa muito difícil, pois, além de precisar de um conhecimento prévio do problema, muitas vezes a parametrização da recompensa é relevante para alguns agentes. Pelo que foi observado, o DDPG é mais influenciado pelos parâmetros dessa função que o SAC.

A segunda impressão do gráfico 5.14 é o percentual de sucesso que o SAC tem para as tarefas simples, mesmo aquelas em que ele não foi treinado. Isso valida os comentários descritos na seção 4.2, de que um agente treinado em um ambiente pode ser usando em um outro ambiente similar. O mesmo é válido para o DDPG, possuindo uma performance inferior. Mesmo nos cenários complexos, o SAC tem um aproveitamento razoável, próximo de 40% em um cenário e próximo de 20% no outro cenário, mas vale destacar que o agente foi treinado em um ambiente simples. Já o DDPG, apesar de ele conseguir ainda chegar no alvo algumas vezes, o resultado realmente não é admirável. Porém, ressalta-se que o DDPG com os parâmetros devidamente ajustados e a função de recompensa melhor formulada deve ter resultados bastante interessantes.

Para essa tarefa de pose completa, o tempo de treinamento para o SAC foi de 46 minutos e 21 segundos, enquanto o treinamento do DDPG foi concluído em 1 hora, 22 minutos e 50 segundos.

Em seguida, serão mostrados os resultados para a segunda proposta de validação. A figura 5.15 mostra o percentual de sucesso em cada um dos N_{v1} cenários, partindo das N_{v2} configurações iniciais.

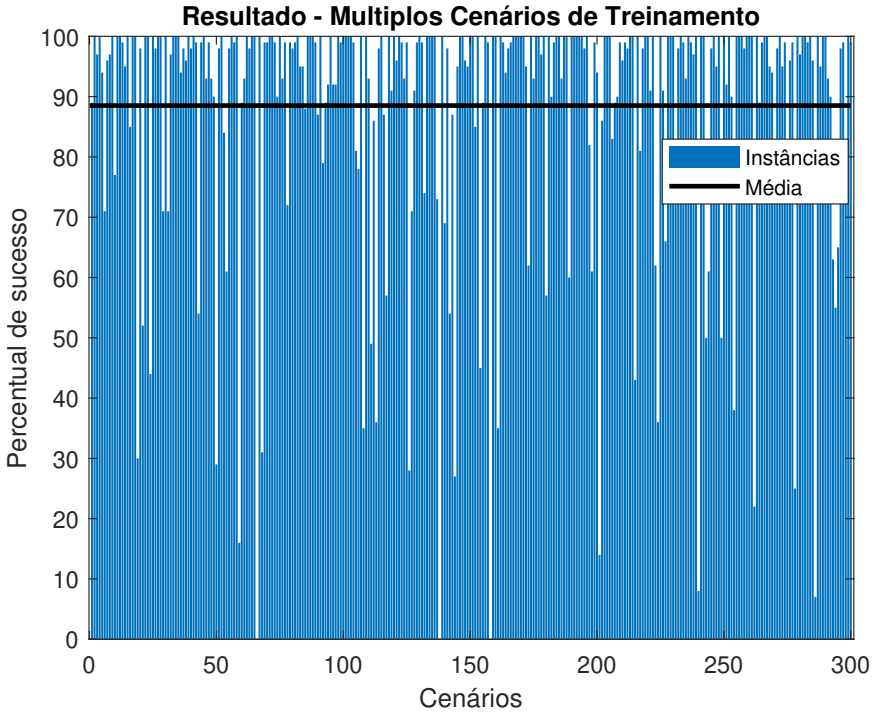

Figura 5.15: Segunda proposta de validação: percentual de sucesso do agente SAC em cada cenário de validação. A linha preta corresponde a média aritmética dos percentuais em cada cenário. Resultados somente para o agente SAC. Fonte: autor.

A figura 5.15 mostra que o agente teve um alto percentual de sucesso na maioria dos cenários, o que indica que o agente aprendeu de forma efetiva, após ter sido exposto a vários tipos de cenários. Na média, o agente atingiu um percentual de sucesso médio de $88{,}5\%$ em todos os cenários. Apesar de não ser o resultado perfeito, esse percentual de sucesso do agente SAC é bastante interessante para mostrar a capacidade deste agente para esse tipo de tarefas. O resultado mostra também que em três cenários, o agente não foi capaz de chegar no alvo em nenhuma das tentativas, o que pode ser explicado porque o alvo desejado pode ser muito difícil de ser alcançado, uma vez que a seleção do cenário foi feita de forma aleatória.

Não foi possível realizar essa validação no DDPG, visto que o resultado do agente teve um baixo percentual de sucesso, o que não justificou apresentar no texto.

Por fim, é importante destacar que foram obtidos bons resultados, mesmo com poucos episódios de treinamento. A principal explicação para esse resultado é que os primeiros episódios não se encerram de forma precoce, o que aconteceria facilmente sem a presença do filtro de desvio dos obstáculos proposto. Com esse filtro, o agente pode explorar por mais tempo o ambiente, o que resulta em mais tempo de treinamento por episódio, principalmente nos primeiros episódios.

5.3 Resumo do capítulo

Este capítulo apresentou um conjunto de simulações para validar os conceitos apresentados no capítulo de metodologia. Primeiro, foram apresentadas as simulações para mostrar o aprendizado do agente proposto nessa dissertação, mostrando a evolução do agente ao longo do tempo, de acordo com as métricas propostas. Em seguida, foram apresentadas simulações que validam a proposta de ambiente que utiliza informações locais para alimentar as redes neurais dos agentes. É visto que o agente, ao ser treinado em um cenário, pode ser aplicado em outros cenários e ainda assim conseguir desviar dos obstáculos até chegar no alvo.

Capítulo 6

Conclusões

Esta dissertação apresentou a utilização de aprendizagem por reforço em aplicações de controle de movimento de robôs. Inicialmente, uma revisão bibliográfica abrangente da área de aprendizado por reforço foi apresentada, desde os primeiros trabalhos na área até os trabalhos mais recentes, já em conjunto com a pesquisa de aprendizado profundo.

No capítulo seguinte, vários conceitos fundamentais foram apresentados, como MDPs, as funções de valor $V^{\pi}(s)$ e $Q^{\pi}(s, a)$, arquiteturas ator-crítico, técnicas de exploração, entre vários outros conceitos.

Em seguida, foram mostradas duas abordagens de realizar o controle de movimento de robôs utilizando aprendizagem por reforço. A primeira abordagem foi a proposta de um agente capaz de realizar o aprendizado da tarefa utilizando o algoritmo *iteração de valor na álgebra max-plus* para robôs sem a restrição de não holonomia. A proposta desse agente foi tal que não teria necessidade definir uma função de recompensa estruturada, ou definir um espaço de observações complexo, podendo utilizar o estado como a configuração do robô, e mesmo assim, o agente proposto seria capaz de aprender a finalizar as tarefas ao longo do tempo. Isso é importante, porque definir o ambiente muitas vezes é uma tarefa árdua, sendo necessário escolher os componentes do ambiente de forma minuciosa, o que é evitado ao considerar o agente proposto no terço final do capítulo.

A segunda abordagem foi considerar agentes que usam redes neurais profundas propostos nos últimos anos, com resultados já estáveis, como o DDPG e o SAC. Para isso, foi necessário descrever o cenário em que robô está inserido dentro do arcabouço de aprendizagem por reforço, definindo o ambiente com que o agente irá interagir. Portanto, conforme indicado na figura 4.3 foi necessário definir o espaço de estados, de observações e de ações do sistema, assim como a função de transição de estados, a função de observação e a função de recompensa para um robô móvel não-holonômico de modelo uniciclo. Foi mostrado que é possível descrever o estado do ambiente somente com a configuração do robô, ou descrever também capturando as informações estruturais do cenário, como as informações do alvo e dos obstáculos. Foi explicado que a segunda interpretação ajuda a entender como um agente treinado em apenas um cenário é capaz de completar tarefas em outro. Além disso, foi proposto um filtro que auxilia as redes neurais a

evitarem os obstáculos, fazendo com que os episódios não terminem de forma precoce.

No capítulo de resultados, foram apresentadas as simulações dos agentes descritos, para tarefas de pose completa, mostrando que o ambiente descrito realmente possibilita um agente como o DDPG ou o SAC aprender a movimentar em diferentes tipos de ambientes, mesmo sendo treinado em apenas um cenário. Foi mostrado que esse agente tem a habilidade de melhorar o desempenho ao longo do tempo, mesmo sem a necessidade de descrever uma recompensa estruturada, ou definir um espaço de observações complexo, que são elementos necessários para o agentes como DDPG e SAC. Foi mostrado, que para os hiper-parâmetros escolhidos, o agente SAC teve um resultado melhor que o DDPG, tanto quando treinado em um ambiente apenas, quanto quando treinado em diferentes cenários.

O presente trabalho foi bastante desafiador, pois se trata de uma área ainda pouco explorada, com muitas questões abertas. Por conta disso, o conhecimento adquirido foi sendo construído a partir de diversas fontes, que muitas vezes aplicaram as técnicas mostradas em outros tipos de ambientes que não são robôs, como aplicações em jogos de video-game, ou aplicações de mercado financeiro, entre várias outras. De toda a forma, o conhecimento adquirido durante este trabalho foi bastante satisfatório, mesmo que ainda haja muito a ser averiguado, devido a extensão da área.

6.1 Trabalhos Futuros

Como dito anteriormente, existem muitas pesquisas em andamento que não puderam ser incluídas na dissertação, mas que seriam possibilidades para expandir a pesquisa na área. Além das já mencionadas, aprendizagem por reforço offline, transferência de aprendizagem e meta-aprendizagem, existem ainda outras possibilidades como a utilização de imagens provenientes de uma câmera, e fazer o controle do robô de forma visual, utilizando redes neurais convolucionais e redes neurais recorrentes. Existe ainda a possibilidade de estudar melhor métodos baseados em entropia, um mesclar a pesquisa em aprendizagem com modelo e controle preditivo de modelo (*Model Predict Control* - MPC). Um outro trabalho sugerido é o estudo de técnicas para realizar tarefas mais complicadas, como a locomoção de um humanoide ou de um robô quadrúpede. Outro trabalho futuro é a modificação do agente proposto na seção 4.1 para ambientes multi-tarefa, onde dois ou mais alvos são aprendidos de forma simultânea. Como é possível observar, existem diversas possibilidades de expansão nesta área, que vem se ampliando de forma constante.

Referências Bibliográficas

[Andreae, 1963] Andreae, J. (1963). Stella: A scheme for a learning machine. *IFAC Proceedings Volumes*, 1.

[Andreae, 1969] Andreae, J. (1969). Learning machines: A unified view. *Encyclopaedia of Linguistics, Information and Control, Pergamon Press*, pages 261–270.

[Andreae, 1977] Andreae, J. (1977). *Thinking with the Teachable Machine*.

[Andreae and Cashin, 1969] Andreae, J. and Cashin, P. (1969). A learning machine with monologue. *International Journal of Human-computer Studies / International Journal of Man-machine Studies - IJMMS*, 1:1–20.

[Andreas et al., 2017] Andreas, J., Klein, D., and Levine, S. (2017). Modular multitask reinforcement learning with policy sketches. In *International Conference on Machine Learning*, pages 166–175. PMLR.

[Bardi and Capuzzo-Dolcetta, 2008] Bardi, M. and Capuzzo-Dolcetta, I. (2008). *Optimal control and viscosity solutions of Hamilton-Jacobi-Bellman equations*. Springer Science & Business Media.

[Barreto et al., 2017] Barreto, A., Dabney, W., Munos, R., Hunt, J. J., Schaul, T., van Hasselt, H. P., and Silver, D. (2017). Successor features for transfer in reinforcement learning. *Advances in neural information processing systems*, 30.

[Bassani et al., 2020] Bassani, H. F., Delgado, R. A., Junior, J. N. d. O. L., Medeiros, H. R., Braga, P. H., and Tapp, A. (2020). Learning to play soccer by reinforcement and applying sim-to-real to compete in the real world. *arXiv preprint arXiv:2003.11102*.

[Belkhale et al., 2021] Belkhale, S., Li, R., Kahn, G., McAllister, R., Calandra, R., and Levine, S. (2021). Model-based meta-reinforcement learning for flight with suspended payloads. *IEEE Robotics and Automation Letters*, 6(2):1471–1478.

[Bellemare et al., 2016] Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. (2016). Unifying count-based exploration and intrinsic motivation. *Advances in neural information processing systems*, 29.

[Bellman, 1957] Bellman, R. (1957). *Dynamic Programming*. Dover Publications.

[Bertsekas and Tsitsiklis, 1989] Bertsekas, D. P. and Tsitsiklis, J. N. (1989). *Parallel and Distributed Computation: Numerical Methods*. Prentice-Hall, Inc., USA.

[Buckman et al., 2018] Buckman, J., Hafner, D., Tucker, G., Brevdo, E., and Lee, H. (2018). Sample-efficient reinforcement learning with stochastic ensemble value expansion. *Advances in neural information processing systems*, 31.

[Burda et al., 2018] Burda, Y., Edwards, H., Storkey, A., and Klimov, O. (2018). Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*.

[Chaffre et al., 2020] Chaffre, T., Moras, J., Chan-Hon-Tong, A., and Marzat, J. (2020). Sim-to-real transfer with incremental environment complexity for reinforcement learning of depth-based robot navigation. *arXiv preprint arXiv:2004.14684*.

[Chebotar et al., 2021] Chebotar, Y., Hausman, K., Lu, Y., Xiao, T., Kalashnikov, D., Varley, J., Irpan, A., Eysenbach, B., Julian, R., Finn, C., et al. (2021). Actionable models: Unsupervised offline reinforcement learning of robotic skills. *arXiv preprint arXiv:2104.07749*.

[Choset et al., 2005] Choset, H., Lynch, K., Hutchinson, S., Kantor, G., Burgard, W., Kavraki, L., Thrun, S., Latombe, J., and Arkin, R. (2005). *Principles of Robot Motion: Theory, Algorithms, and Implementation*. A Bradford book. A Bradford Book.

[Chua et al., 2018] Chua, K., Calandra, R., McAllister, R., and Levine, S. (2018). Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *Advances in neural information processing systems*, 31.

[Ding et al., 2020] Ding, Z., Lepora, N. F., and Johns, E. (2020). Sim-to-real transfer for optical tactile sensing. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1639–1645. IEEE.

[Dorf and Bishop, 2001] Dorf, R. and Bishop, R. (2001). Sistemas de controle moderno; 8ª. *Ed, LTC, Rio de Janeiro*.

[Ebert et al., 2017] Ebert, F., Finn, C., Lee, A. X., and Levine, S. (2017). Self-supervised visual planning with temporal skip connections. In *CoRL*, pages 344–356.

[Eysenbach et al., 2018] Eysenbach, B., Gupta, A., Ibarz, J., and Levine, S. (2018). Diversity is all you need: Learning skills without a reward function. *arXiv preprint arXiv:1802.06070*.

[Feinberg et al., 2018] Feinberg, V., Wan, A., Stoica, I., Jordan, M. I., Gonzalez, J. E., and Levine, S. (2018). Model-based value expansion for efficient model-free reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning (ICML 2018)*.

[Finn et al., 2017] Finn, C., Abbeel, P., and Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR.

[Finn and Levine, 2017] Finn, C. and Levine, S. (2017). Deep visual foresight for planning robot motion. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2786–2793. IEEE.

[Finn et al., 2016] Finn, C., Levine, S., and Abbeel, P. (2016). Guided cost learning: Deep inverse optimal control via policy optimization. In *International conference on machine learning*, pages 49–58. PMLR.

[Fu et al., 2017a] Fu, J., Co-Reyes, J., and Levine, S. (2017a). Ex2: Exploration with exemplar models for deep reinforcement learning. *Advances in neural information processing systems*, 30.

[Fu et al., 2017b] Fu, J., Luo, K., and Levine, S. (2017b). Learning robust rewards with adversarial inverse reinforcement learning. *arXiv preprint arXiv:1710.11248*.

[Fujimoto et al., 2018] Fujimoto, S., Hoof, H., and Meger, D. (2018). Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1587–1596. PMLR.

[Ghosh et al., 2017] Ghosh, D., Singh, A., Rajeswaran, A., Kumar, V., and Levine, S. (2017). Divide-and-conquer reinforcement learning. *arXiv preprint arXiv:1711.09874*.

[Gonçalves, 2021] Gonçalves, V. M. (2021). Max-plus approximation for reinforcement learning. *Automatica*, 129:109623.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[Gregor et al., 2016] Gregor, K., Rezende, D. J., and Wierstra, D. (2016). Variational intrinsic control. *arXiv preprint arXiv:1611.07507*.

[Gu et al., 2016a] Gu, S., Lillicrap, T., Ghahramani, Z., Turner, R. E., and Levine, S. (2016a). Q-prop: Sample-efficient policy gradient with an off-policy critic. *arXiv preprint arXiv:1611.02247*.

[Gu et al., 2016b] Gu, S., Lillicrap, T., Sutskever, I., and Levine, S. (2016b). Continuous deep q-learning with model-based acceleration. In *International conference on machine learning*, pages 2829–2838. PMLR.

[Gupta et al., 2018] Gupta, A., Mendonca, R., Liu, Y., Abbeel, P., and Levine, S. (2018). Meta-reinforcement learning of structured exploration strategies. *Advances in neural information processing systems*, 31.

[Haarnoja et al., 2018a] Haarnoja, T., Ha, S., Zhou, A., Tan, J., Tucker, G., and Levine, S. (2018a). Learning to walk via deep reinforcement learning. *arXiv preprint arXiv:1812.11103*.

[Haarnoja et al., 2018b] Haarnoja, T., Pong, V., Zhou, A., Dalal, M., Abbeel, P., and Levine, S. (2018b). Composable deep reinforcement learning for robotic manipulation. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 6244–6251. IEEE.

[Haarnoja et al., 2017] Haarnoja, T., Tang, H., Abbeel, P., and Levine, S. (2017). Reinforcement learning with deep energy-based policies. In *International Conference on Machine Learning*, pages 1352–1361. PMLR.

[Haarnoja et al., 2018c] Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018c). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR.

[Haarnoja et al., 2018d] Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., et al. (2018d). Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*.

[Hazan et al., 2019] Hazan, E., Kakade, S., Singh, K., and Van Soest, A. (2019). Provably efficient maximum entropy exploration. In *International Conference on Machine Learning*, pages 2681–2691. PMLR.

[Hinton et al., 2015] Hinton, G., Vinyals, O., Dean, J., et al. (2015). Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2(7).

[Ho and Ermon, 2016] Ho, J. and Ermon, S. (2016). Generative adversarial imitation learning. *Advances in neural information processing systems*, 29.

[Houthooft et al., 2016] Houthooft, R., Chen, X., Chen, X., Duan, Y., Schulman, J., De Turck, F., and Abbeel, P. (2016). Vime: Variational information maximizing exploration. In Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc.

[Houthooft et al., 2018] Houthooft, R., Chen, Y., Isola, P., Stadie, B., Wolski, F., Jonathan Ho, O., and Abbeel, P. (2018). Evolved policy gradients. *Advances in Neural Information Processing Systems*, 31.

[Ibarz et al., 2021] Ibarz, J., Tan, J., Finn, C., Kalakrishnan, M., Pastor, P., and Levine, S. (2021). How to train your robot with deep reinforcement learning: lessons we have learned. *The International Journal of Robotics Research*, 40(4-5):698–721.

[Ioffe and Szegedy, 2015] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR.

[Jahanandish, 2010] Jahanandish, M. (2010). A geometric-based numerical solution of eikonal equation over a closed level curve. *Iranian Journal of Science and Technology (Sciences)*, 34(1):47–58.

[James et al., 2013] James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013). *An Introduction to Statistical Learning: with Applications in R*. Springer Texts in Statistics. Springer New York.

[Janner et al., 2019] Janner, M., Fu, J., Zhang, M., and Levine, S. (2019). When to trust your model: Model-based policy optimization. *Advances in Neural Information Processing Systems*, 32.

[Janner et al., 2021] Janner, M., Li, Q., and Levine, S. (2021). Offline reinforcement learning as one big sequence modeling problem. *Advances in neural information processing systems*, 34.

[Jesus et al., 2019] Jesus, J. C., Bottega, J. A., Cuadros, M. A., and Gamarra, D. F. (2019). Deep deterministic policy gradient for navigation of mobile robots in simulated environments. In *2019 19th International Conference on Advanced Robotics (ICAR)*, pages 362–367. IEEE.

[Jiang and Li, 2016] Jiang, N. and Li, L. (2016). Doubly robust off-policy value evaluation for reinforcement learning. In *International Conference on Machine Learning*, pages 652–661. PMLR.

[Júnior, 2018] Júnior, W. S. F. (2018). Validação de técnicas de controle em sistemas robóticos utilizando teoria de otimização. Monografia. Universidade Federal de Minas Gerais.

[Júnior and Gonçalves, 2018] Júnior, W. S. F. and Gonçalves, V. M. (2018). Formalismo para resoluçao de tarefas em robótica utilizando otimizaçao. *Anais do XXII Congresso Brasileiro de Automática (CBA) - 2018*.

[Júnior et al., 2021] Júnior, W. S. F., Monteiro, N. S., and Gonçalves, V. M. (2021). Learning vector fields through persistent exploration for robot motion control. *Anais do XV Simpósio Brasileiro de Automação Inteligente (SBAI) - 2021*.

[Kahn et al., 2021] Kahn, G., Abbeel, P., and Levine, S. (2021). Badgr: An autonomous self-supervised learning-based navigation system. *IEEE Robotics and Automation Letters*, 6(2):1312–1319.

[Kalakrishnan et al., 2011] Kalakrishnan, M., Righetti, L., Pastor, P., and Schaal, S. (2011). Learning force control policies for compliant manipulation. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4639–4644. IEEE.

[Kalashnikov et al., 2021] Kalashnikov, D., Varley, J., Chebotar, Y., Swanson, B., Jonschkowski, R., Finn, C., Levine, S., and Hausman, K. (2021). Mt-opt: Continuous multi-task robotic reinforcement learning at scale. *arXiv preprint arXiv:2104.08212*.

[Kanoun et al., 2011] Kanoun, O., Lamiraux, F., and Wieber, P. (2011). Kinematic control of redundant manipulators: Generalizing the task-priority framework to inequality task. *IEEE Transactions on Robotics*, 27(4):785–792.

[Kaspar and Bock,] Kaspar, M. and Bock, J. Reinforcement learning with cartesian commands and sim to real transfer for peg in hole tasks.

[Kavraki et al., 1996] Kavraki, L. E., Svestka, P., Latombe, J.-C., and Overmars, M. H. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12(4):566–580.

[Khatib, 1986] Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. In *Autonomous robot vehicles*, pages 396–404. Springer.

[Kidambi et al., 2020] Kidambi, R., Rajeswaran, A., Netrapalli, P., and Joachims, T. (2020). Morel: Model-based offline reinforcement learning. *Advances in neural information processing systems*, 33:21810–21823.

[Kober et al., 2013] Kober, J., Bagnell, J. A., and Peters, J. (2013). Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274.

[Kostrikov et al., 2021] Kostrikov, I., Nair, A., and Levine, S. (2021). Offline reinforcement learning with implicit q-learning. *arXiv preprint arXiv:2110.06169*.

[Kuffner and RRT-Connect, 2000] Kuffner, J. and RRT-Connect, S. L. (2000). An efficient approach to single-query path planning ieee international conference on robotics and automation. *San Francisco*, pages 473–479.

[Kumar et al., 2019] Kumar, A., Fu, J., Soh, M., Tucker, G., and Levine, S. (2019). Stabilizing off-policy q-learning via bootstrapping error reduction. *Advances in Neural Information Processing Systems*, 32.

[Kumar et al., 2021] Kumar, A., Singh, A., Tian, S., Finn, C., and Levine, S. (2021). A workflow for offline model-free robotic reinforcement learning. *arXiv preprint arXiv:2109.10813*.

[Lee et al., 2019] Lee, L., Eysenbach, B., Parisotto, E., Xing, E., Levine, S., and Salakhutdinov, R. (2019). Efficient exploration via state marginal matching. *arXiv preprint arXiv:1906.05274*.

[Levine, 2018] Levine, S. (2018). Reinforcement learning and control as probabilistic inference: Tutorial and review. *arXiv preprint arXiv:1805.00909*.

[Levine and Abbeel, 2014] Levine, S. and Abbeel, P. (2014). Learning neural network policies with guided policy search under unknown dynamics. *Advances in neural information processing systems*, 27.

[Levine et al., 2016] Levine, S., Finn, C., Darrell, T., and Abbeel, P. (2016). End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373.

[Levine and Koltun, 2013] Levine, S. and Koltun, V. (2013). Guided policy search. In *International conference on machine learning*, pages 1–9. PMLR.

[Levine et al., 2020] Levine, S., Kumar, A., Tucker, G., and Fu, J. (2020). Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*.

[Lillicrap et al., 2015] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

[Liu et al., 2020] Liu, L., Dugas, D., Cesari, G., Siegwart, R., and Dubé, R. (2020). Robot navigation in crowded environments using deep reinforcement learning. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5671–5677. IEEE.

[Matas et al., 2018] Matas, J., James, S., and Davison, A. J. (2018). Sim-to-real reinforcement learning for deformable object manipulation. In *Conference on Robot Learning*, pages 734–743. PMLR.

[Minsky, 1954] Minsky, M. (1954). *Theory of Neural-analog Reinforcement Systems and Its Application to the Brain-model Problem*. Princeton University.

[Minsky, 1961] Minsky, M. (1961). Steps toward artificial intelligence. *Proc. IRE*.

[Mnih et al., 2016] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR.

[Mnih et al., 2015] 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. (2015). Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533.

[Nachum et al., 2019] Nachum, O., Ahn, M., Ponte, H., Gu, S., and Kumar, V. (2019). Multi-agent manipulation via locomotion using hierarchical sim2real. *arXiv preprint arXiv:1908.05224*.

[Nachum et al., 2017] Nachum, O., Norouzi, M., Xu, K., and Schuurmans, D. (2017). Bridging the gap between value and policy based reinforcement learning. *Advances in neural information processing systems*, 30.

[Nagabandi et al., 2018] Nagabandi, A., Kahn, G., Fearing, R. S., and Levine, S. (2018). Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7559–7566. IEEE.

[Nagabandi et al., 2020] Nagabandi, A., Konolige, K., Levine, S., and Kumar, V. (2020). Deep dynamics models for learning dexterous manipulation. In *Conference on Robot Learning*, pages 1101–1112. PMLR.

[Nair et al., 2018] Nair, A. V., Pong, V., Dalal, M., Bahl, S., Lin, S., and Levine, S. (2018). Visual reinforcement learning with imagined goals. *Advances in neural information processing systems*, 31.

[Osband et al., 2016] Osband, I., Blundell, C., Pritzel, A., and Van Roy, B. (2016). Deep exploration via bootstrapped dqn. *Advances in neural information processing systems*, 29.

[Panagou, 2014] Panagou, D. (2014). Motion planning and collision avoidance using navigation vector fields. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2513–2518. IEEE.

[Parisotto et al., 2015] Parisotto, E., Ba, J. L., and Salakhutdinov, R. (2015). Actor-mimic: Deep multitask and transfer reinforcement learning. *arXiv preprint arXiv:1511.06342*.

[Parmas et al., 2018] Parmas, P., Rasmussen, C. E., Peters, J., and Doya, K. (2018). Pipps: Flexible model-based policy search robust to the curse of chaos. In *International Conference on Machine Learning*, pages 4065–4074. PMLR.

[Pedersen, 2019] Pedersen, O.-M. (2019). Sim-to-real transfer of robotic gripper pose estimation-using deep reinforcement learning, generative adversarial networks, and visual servoing. Master's thesis, NTNU.

[Peng et al., 2018] Peng, X. B., Andrychowicz, M., Zaremba, W., and Abbeel, P. (2018). Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 3803–3810. IEEE.

[Peng et al., 2019] Peng, X. B., Kumar, A., Zhang, G., and Levine, S. (2019). Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *arXiv preprint arXiv:1910.00177*.

[Pong et al., 2019] Pong, V. H., Dalal, M., Lin, S., Nair, A., Bahl, S., and Levine, S. (2019). Skew-fit: State-covering self-supervised reinforcement learning. *arXiv preprint arXiv:1903.03698*.

[Qin et al., 2019] Qin, B., Gao, Y., and Bai, Y. (2019). Sim-to-real: Six-legged robot control with deep reinforcement learning and curriculum learning. In *2019 4th International Conference on Robotics and Automation Engineering (ICRAE)*, pages 1–5. IEEE.

[Rajeswaran et al., 2017] Rajeswaran, A., Kumar, V., Gupta, A., Vezzani, G., Schulman, J., Todorov, E., and Levine, S. (2017). Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*.

[Rezende et al., 2021] Rezende, A. M., Goncalves, V. M., and Pimenta, L. C. (2021). Constructive time-varying vector fields for robot navigation. *IEEE Transactions on Robotics*.

[Russo and Van Roy, 2014] Russo, D. and Van Roy, B. (2014). Learning to optimize via information-directed sampling. *Advances in Neural Information Processing Systems*, 27.

[Rusu et al., 2015] Rusu, A. A., Colmenarejo, S. G., Gulcehre, C., Desjardins, G., Kirkpatrick, J., Pascanu, R., Mnih, V., Kavukcuoglu, K., and Hadsell, R. (2015). Policy distillation. *arXiv preprint arXiv:1511.06295*.

[Schmidhuber, 2010] Schmidhuber, J. (2010). Formal theory of creativity, fun, and intrinsic motivation (1990–2010). *IEEE transactions on autonomous mental development*, 2(3):230–247.

[Schulman et al., 2017a] Schulman, J., Chen, X., and Abbeel, P. (2017a). Equivalence between policy gradients and soft q-learning. *arXiv preprint arXiv:1704.06440*.

[Schulman et al., 2015a] Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015a). Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR.

[Schulman et al., 2015b] Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. (2015b). High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*.

[Schulman et al., 2017b] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017b). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

[Shao et al., 2019] Shao, K., Tang, Z., Zhu, Y., Li, N., and Zhao, D. (2019). A Survey of Deep Reinforcement Learning in Video Games. *arXiv e-prints*, page arXiv:1912.10944.

[Siciliano et al., 2010] Siciliano, B., Sciavicco, L., Villani, L., and Oriolo, G. (2010). *Robotics: Modelling, Planning and Control*. Advanced Textbooks in Control and Signal Processing. Springer London.

[Silva et al., 2019] Silva, J. A., Grassi, V., and Wolf, D. F. (2019). Continuous deep maximum entropy inverse reinforcement learning using online pomdp. In *2019 19th International Conference on Advanced Robotics (ICAR)*, pages 382–387. IEEE.

[Silver et al., 2014] Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. PMLR.

[Silver et al., 2017] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. (2017). Mastering the game of go without human knowledge. *nature*, 550(7676):354–359.

[Spong et al., 2005] Spong, M., Hutchinson, S., and Vidyasagar, M. (2005). *Robot Modeling and Control*. Wiley.

[Stadie et al., 2015] Stadie, B. C., Levine, S., and Abbeel, P. (2015). Incentivizing exploration in reinforcement learning with deep predictive models. *arXiv preprint arXiv:1507.00814*.

[Surmann et al., 2020] Surmann, H., Jestel, C., Marchel, R., Musberg, F., Elhadj, H., and Ardani, M. (2020). Deep reinforcement learning for real autonomous mobile robot navigation in indoor environments. *arXiv preprint arXiv:2005.13857*.

[Sutton and Barto, 1998] Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. The MIT Press, first edition.

[Sutton and Barto, 2018] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. The MIT Press, second edition.

[Sutton et al., 2000] Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063.

[Tai et al., 2017] Tai, L., Paolo, G., and Liu, M. (2017). Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 31–36. IEEE.

[Tang et al., 2017] Tang, H., Houthooft, R., Foote, D., Stooke, A., Xi Chen, O., Duan, Y., Schulman, J., DeTurck, F., and Abbeel, P. (2017). # exploration: A study of count-based exploration for deep reinforcement learning. *Advances in neural information processing systems*, 30.

[Tassa et al., 2012] Tassa, Y., Erez, T., and Todorov, E. (2012). Synthesis and stabilization of complex behaviors through online trajectory optimization. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4906–4913. IEEE.

[Teh et al., 2017] Teh, Y., Bapst, V., Czarnecki, W. M., Quan, J., Kirkpatrick, J., Hadsell, R., Heess, N., and Pascanu, R. (2017). Distral: Robust multitask reinforcement learning. *Advances in neural information processing systems*, 30.

[The MathWorks, 2019] The MathWorks, I. (2019). *Reinforcement Learning Toolbox*. Natick, Massachusetts, United State.

[Traoré et al., 2019] Traoré, R., Caselles-Dupré, H., Lesort, T., Sun, T., Díaz-Rodríguez, N., and Filliat, D. (2019). Continual reinforcement learning deployed in real-life using policy distillation and sim2real transfer. *arXiv preprint arXiv:1906.04452*.

[Tzeng et al., 2014] Tzeng, E., Hoffman, J., Zhang, N., Saenko, K., and Darrell, T. (2014). Deep domain confusion: Maximizing for domain invariance. *arXiv preprint arXiv:1412.3474*.

[Uhlenbeck and Ornstein, 1930] Uhlenbeck, G. E. and Ornstein, L. S. (1930). On the theory of the brownian motion. *Physical review*, 36(5):823.

[Van Hasselt et al., 2016] Van Hasselt, H., Guez, A., and Silver, D. (2016). Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30.

[Vavryčuk, 2012] Vavryčuk, V. (2012). On numerically solving the complex eikonal equation using real ray-tracing methods: A comparison with the exact analytical solution. *Geophysics*, 77(4):T109–T116.

[Wang et al., 2016a] Wang, J. X., Kurth-Nelson, Z., Tirumala, D., Soyer, H., Leibo, J. Z., Munos, R., Blundell, C., Kumaran, D., and Botvinick, M. (2016a). Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*.

[Wang et al., 2016b] Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., and Freitas, N. (2016b). Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. PMLR.

[Watkins, 1989] Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards*. PhD thesis, King's College, Oxford.

[Watter et al., 2015] Watter, M., Springenberg, J., Boedecker, J., and Riedmiller, M. (2015). Embed to control: A locally linear latent dynamics model for control from raw images. *Advances in neural information processing systems*, 28.

[Werbos, 1987] Werbos, P. J. (1987). Building and understanding adaptive systems: A statistical/numerical approach to factory automation and brain research. *IEEE Transactions on Systems, Man, and Cybernetics*, 17(1):7–20.

[Whitney, 1969] Whitney, D. E. (1969). Resolved motion rate control of manipulators and human prostheses. *IEEE Transactions on man-machine systems*, 10(2):47–53.

[Witman et al., 2019] Witman, M., Gidon, D., Graves, D. B., Smit, B., and Mesbah, A. (2019). Sim-to-real transfer reinforcement learning for control of thermal effects of an atmospheric pressure plasma jet. *Plasma Sources Science and Technology*, 28(9):095019.

[Witten, 1977a] Witten, I. H. (1977a). An adaptive optimal controller for discrete-time markov environments. *Information and Control*, 34(4):286–295.

[Witten, 1977b] Witten, I. H. (1977b). An adaptive optimal controller for discrete-time markov environments. *Information and control*, 34(4):286–295.

[Wu et al., 2019] Wu, Y., Tucker, G., and Nachum, O. (2019). Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361*.

[Yu et al., 2020] Yu, T., Thomas, G., Yu, L., Ermon, S., Zou, J. Y., Levine, S., Finn, C., and Ma, T. (2020). Mopo: Model-based offline policy optimization. *Advances in Neural Information Processing Systems*, 33:14129–14142.

[Zhang et al., 2019] Zhang, M., Vikram, S., Smith, L., Abbeel, P., Johnson, M., and Levine, S. (2019). Solar: Deep structured representations for model-based reinforcement learning. In *International Conference on Machine Learning*, pages 7444–7453. PMLR.

[Zhao et al., 2020] Zhao, T. Z., Nagabandi, A., Rakelly, K., Finn, C., and Levine, S. (2020). Meld: Meta-reinforcement learning from images via latent state models. *arXiv preprint arXiv:2010.13957*.

[Zhu et al., 2017] Zhu, Y., Mottaghi, R., Kolve, E., Lim, J. J., Gupta, A., Fei-Fei, L., and Farhadi, A. (2017). Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3357–3364. IEEE.

[Ziebart et al., 2010] Ziebart, B. D., Bagnell, J. A., and Dey, A. K. (2010). Modeling interaction via the principle of maximum causal entropy. In *ICML*.