JONATHAS MARCELO PEREIRA FIGUEIREDO
RODRIGO PEREIRA ABOU REJAILI

APLICAÇÃO DE ALGORITMOS DE APRENDIZAGEM POR REFORÇO PARA CONTROLE DE NAVIOS EM ÁGUAS RESTRITAS

São Paulo
2018

JONATHAS MARCELO PEREIRA FIGUEIREDO
RODRIGO PEREIRA ABOU REJAILI

APLICAÇÃO DE ALGORITMOS DE APRENDIZAGEM POR REFORÇO PARA CONTROLE DE NAVIOS EM ÁGUAS RESTRITAS

Trabalho apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro Mecânico de Automação e Sistemas.

São Paulo
2018

JONATHAS MARCELO PEREIRA FIGUEIREDO
RODRIGO PEREIRA ABOU REJAILI

APLICAÇÃO DE ALGORITMOS DE APRENDIZAGEM POR REFORÇO PARA CONTROLE DE NAVIOS EM ÁGUAS RESTRITAS

Trabalho apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro Mecânico de Automação e Sistemas.

Área de Concentração:

Automação e Sistemas

Orientador:

Prof. Dr. Fábio Cozman

Co-orientador:

Prof. Dr. Lucas Moscato

São Paulo
2018

Catalogação-na-publicação

Figueiredo, Jonathas Marcelo Pereira
Aplicação de algoritmos de aprendizagem por reforço para controle de navios em águas restritas / J. M. P. Figueiredo, R. P. A. Rejaili -- São Paulo, 2018.
86 p.

Trabalho de Formatura - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos.

1.Aprendizagem por reforço 2.Navegação marítima 3.Redes neurais 4.Aprendizado Computacional I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos II.t. III.Rejaili, Rodrigo Pereira Abou

AGRADECIMENTOS

Ao nosso orientador Prof. Dr. Fábio Cozman, pela orientação, apoio e confiança.

Ao Prof. Dr. Eduardo Aoun Tannuri pela oportunidade de realização e elaboração deste trabalho em conjunto com o TPN.

Ao Mestrando José Amendola pela orientação apoio e cooperação na realização deste trabalho.

Ao corpo docente do curso de Engenharia Mecatrônica da Escola Politécnica da Universidade de São Paulo que nos proporcionou real aprendizado e pensamento crítico apesar das limitações e dificuldades presentes no contexto da Universidade.

RESUMO

A navegação de águas restritas ainda é uma tarefa onde a Aprendizagem por Reforço foi pouco explorada na comunidade científica. No entanto, tais algoritmos poderiam potencialmente trazer leis de controle mais robustas e eficazes para o campo, onde a automação possui diversos benefícios no evitamento de acidentes e redução da necessidade da formação intensiva de pessoal (práticos). Este trabalho apresenta então a utilização de algoritmos de Aprendizado por Reforço para o controle automático dos movimentos de manobra de navios em águas restritas. A aprendizagem de uma lei de controle foi realizada utilizando-se métodos de Deep Q Network e de Gradiente de Política (Deep Deterministic Policy Gradient) em conjunto com um simulador numérico para manobras de navios. A lei de controle aprendida pelos dois métodos apresentou boa resposta nas simulações realizadas de navegação em um canal.

Palavras-Chave – Aprendizado por Reforço, Navegação Interior, Redes Neurais, Aprendizado Computacional.

ABSTRACT

The navigation on restricted waters using Reinforcement Learning methods is still too little explored in the scientific community. However, such algorithms could potentially result in more robust and efficient control laws on the field. This work thus presents the use of Reinforcement Learning algorithms for the automated control of vessels maneuvering movements in restricted waters. Control laws are learned using Deep Q Network and Deep Deterministic Policy Gradient methods coupled with a numerical simulator for ship maneuvers. The control law learned by both methods presented good responses on channel navigation simulations.

Keywords – Reinforcement Learning, Inland Navigation, Neural Networks, Machine Learning.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

APR Aprendizagem por Reforço

CNN Convolutional Neural Network

DDPG Deep Deterministic Policy Gradient

DQN Deep Q Network

DT Diferenças Temporais

EQM Erro Quadtrático Médio

GP Gradiente de Política

LSTM Long Short-Term Memory

MC Monte Carlo

MPI Métodos de Ponto Inicial

PD Programação Dinâmica

PDM Processo de Decisão de Markov

PRM Processo de Recompensa de Markov

RNA Rede Neural Artificial

RNN Recurrent Neural Network

SARSA State-Action-Reward-State-Action

SSN Simulador Simples de Navegação

SVM Support Vector Machine

TA Transferência de Aprendizado

TPN Tanque de Provas Numérico

1 INTRODUÇÃO

1.1 Contexto

A automação vem sendo utilizada para aumentar a eficiência de sistemas e processos, e tem como um de seus objetivos diminuir ou substituir a intervenção humana em processos nos quais a ocorrência de erros não é tolerada. O controle de navegação de sistemas de transporte é um exemplo no qual a automação é pretendida para diminuir acidentes ocasionados pela influência humana nos processos de pilotagem e controle.

Recentemente, a aplicação de redes neurais e outros métodos de aprendizagem por máquina tem apresentado bons resultados na automação de sistemas de transportes tais como automóveis, drones e helicópteros - os artigos de Gerla et al.(1) e Cutler e How(2) são exemplos bem sucedidos dessa aplicação. Por outro lado, existem problemas nos quais soluções concretas de automação ainda não foram plenamente desenvolvidas e aplicadas, como é o caso do controle de navios em águas restritas.

Atualmente, a navegação em águas restritas é realizada por um comandante de navios especializado nesse tipo de tarefa. O controle do navio é basado em seu conhecimento das condições ambientais e meteorológicas locais e em sua experiência em atracamento e transporte nessas regiões, exigindo, assim, profissionais experientes e específicos para cada local.

Contudo, esse processo ainda apresenta riscos humanos, os quais são causa recorrente de acidentes marítimos como mostra Hetherington, Flin e Mearns(3). Busca-se então, na automação, uma alternativa para diminuição dos riscos associados ao processo de manobras e navegação em águas restritas. Como mostra Ahmed e Hasegawa(4), tal tema ainda está em aberto, sendo objeto de pesquisa pertinente para o setor náutico.

Este trabalho visa então a aplicação de Redes Neurais Artificiais (RNA) e algoritmos de Aprendizagem por Reforço (APR) em conjunto com o simulador de manobras do Tanque de Provas Numérico (TPN), descrito no trabalho de Filho, Zimbres e Tannuri(5),

para o desenvolvimento de um sistema de controle de direção de um navio. O desempenho de tais algoritmos é avaliado pela sua capacidade de navegar uma embarcação através de uma determinada trajetória.

1.2 Estado da arte

A utilização de redes neurais aplicadas ao transporte em geral é recorrente na literatura desde a década de 1990, como vê-se no trabalho de Dougherty(6). No artigo de Hafner e Riedmiller(7), apresenta-se sistemas de controle em que a aplicação de APR foi bem sucedida, e demonstra-se que algumas adaptações são viáveis para a concepção de controladores usando APR. Porém, como descreve Amendola(8), a aplicação na automação da manobra de navios ainda é assunto pouco explorado na literatura científica. Um exemplo é o trabalho de Ahmed e Hasegawa(4), no qual utiliza-se um controlador de navios em águas restritas assumindo trajetórias de navegação conhecidas.

A aplicação de APR para essas tarefas começou a ser investigada recentemente. Um dos primeiros resultados de relevância presente na literatura é o de Stamenkovich(9), que realizou um experimento utilizando um agente ator-crítico semelhante a um neurônio e simulou a navegação de um navio através de um canal com o auxílio de sensores que fornecem o rumo do navio, o ângulo entre o rumo e o grupo de boias mais próximo e a distância até esse grupo. Lacki(10) comparou em seu estudo a aplicação dos algoritmos SARSA e Q-learning com um modelo discreto de estado para controlar o ângulo de ataque da embarcação, na navegação em águas restritas com velocidade constante e pequenos obstáculos. Mais recentemente, Rak e Gierusz(11) compararam a aplicação do método Q-learning (on-line) usando uma discretização de estados e a Least Squares Policy Iteration (LSPI) para estados contínuos usando aproximadores de função (off-line). Neste estudo, o objetivo foi gerar a trajetória-guia de navegação usando APR, a partir de uma determinada configuração do canal (disposição de obstáculos) que apresentava uma posição final como objetivo. Amendola(8) utilizou uma estratégia de APR usando Fitted Q-Iteration com batchs de simulações geradas com o simulador de manobras do TPN, através de uma discretização das ações de controle e usando uma velocidade de navegação variável, com o objetivo de seguir uma linha-guia em um canal. Os resultados obtidos, no entanto, não foram satisfatórios e a embarcação obteve movimentos oscilatórios ao redor da linha-guia a partir da política de navegação aprendida pelo algoritmo.

1.3 Objetivos e especificações

O objetivo principal desse trabalho é aplicar métodos de APR que sejam capazes de controlar a navegação de um navio em águas restritas. Para tanto, utiliza-se o algoritmo Q-learning (model-free) em conjunto com o simulador fast-time para criar um controlador de direção que seja capaz de manter o navio em uma trajetória de navegação através de um canal.

O interesse nessa tarefa é avaliar a capacidade do algoritmo APR no controle de direção (leme) e de velocidade (propulsão) do navio. Como objetivo secundário potencial pretende-se desenvolver outros tipos de algoritmos APR e compará-los ao algoritmo Q-learning.

1.4 Métodos

Este trabalho segue as tendências recentes na comunidade de machine learning e desenvolve um modelo de Aprendizado por Reforço do tipo Q-learning, aplicado ao controle de trajetória de um navio. Exemplos bem-sucedidos de Q-learning aplicados à controle são apresentados por Nagendra et al.(12) e Kiumarsi et al.(13).

Atualmente a difusão de técnicas de aprendizagem por máquinas tem sido incentivada por comunidades de desenvolvimento de software e por grandes corporações, como o Google. Através do suporte e fornecimento de ferramentas em código-aberto (open-source), essas empresas buscam fomentar o desenvolvimento rápido dessa tecnologia.

Nesse contexto, as bibliotecas de aprendizagem de máquina e aprendizagem profunda Keras e Tensorflow foram selecionadas para o uso nesse trabalho em razão de sua potencialidade, flexibilidade de uso e grande comunidade de colaboradores. Outra biblioteca também utilizada foi a OpenAI Gym, a qual propõe uma estrutura para o modelo de formulação de problemas de APR.

Como mencionado anteriormente, os dados utilizados para treinar o modelo são obtidos através de uma simulação de navegação fast-time. Em resumo, a simulação reproduz um modelo complexo do navio desenvolvido pelo TPN, a qual recebe como entradas as ações de comando de leme e propulsão, em seguida calcula a evolução do espaço de estados do navio para um intervalo de tempo desejado, utilizando o método de Runge-Kutta de ordem 4 e fornece como saída os estados atualizados no fim da integração. Os comandos passados para o simulador podem ser decididos em modo offline ou em modo online de

aprendizagem.

O desenvolvimento de um modelo de aprendizado por máquina do tamanho e da complexidade do modelo Q-learning requer uma série de escolhas de projeto envolvendo múltiplos hiperparâmetros que definem a arquitetura do modelo e o algoritmo de treinamento. Encontrar o conjunto ideal desses parâmetros não é viável devido ao grande espaço de busca e ao custo computacional da execução do sistema. Portanto, extraímos intuições dos trabalhos existentes (Ahmed e Hasegawa(4), Xu et al.(14)) e seguimos algumas heurísticas do campo náutico para desenhar o modelo de APR. Detalhes sobre as heurísticas e intuições utilizadas são apresentados no capítulo 6.

1.5 Organização

Essa monografia está organizada em oito capítulos. O capítulo 1 introduz o tema a ser abordado, faz uma revisão do estado da arte, traça objetivos e especificações gerais do projeto e discute os métodos que serão testados. O capítulo 2 faz uma análise da missão da solução exposta nesta monografia, definindo tanto seus requisitos funcionais e de desempenho, bem como seus modos de operação. O capítulo 3 faz uma introdução teórica à Aprendizagem de Máquina e suas vertentes, em especial às Redes Neurais e ao paradigma da Aprendizagem por Reforço. O capítulo 4 mostra o modelo dinâmico de embarcação utilizado neste estudo e o simulador numérico que realiza a integração deste modelo para obter a resposta do sistema. O capítulo 5 aborda trabalhos relacionados que serviram de base para construção da solução. O capítulo 6 descreve a solução proposta para o problema exposto na Introdução. O capítulo 7 expõe os resultados obtidos com a solução descrita. Finalmente, o capítulo 8 discute as possíveis conclusões e perspectivas que decorrem dos resultados.

2 ANÁLISE E REQUISITOS

Neste capítulo, o projeto é analisado a partir da perspectiva funcional e construtiva do sistema à partir da missão proposta. Primeiro, define-se formalmente a missão do sistema, em seguida, define-se os requisitos necessários para o cumprimento da missão proposta, por fim, realiza-se uma análise funcional do sistema proposto e classifica-se seus principais blocos construtivos.

2.1 Missão

A missão deste trabalho é criar uma política de controle de navegação inteligente que seja capaz de guiar um navio ao longo de um canal de águas restritas, guiando-o através de uma trajetória pré-definida. A missão do nosso sistema é traduzida pela formulação abaixo:

Seja $S(t)$ o vetor de estados de interesse para navegação do navio.

Seja $A(t)$ um conjunto de ações do agente para controle do navio.

Seja $P_r(S(t+1) = s'|S(t) = s, A(t) = a)$ a probabilidade de transição do estado s para o estado s' sob uma ação a.

Seja $R(s, a, s')$ uma função de recompensa que traduz o acerto do navio em relação à uma trajetória-guia pré-definida (equivalente à penalizar o desvio).

O objetivo é treinar um agente para afim de obter o conjunto de ações $A(t)$ que maximizem a soma de recompensas coletadas durante uma trajetória resultante desse conjunto de ações.

Para tanto, o agente deve ser treinado para otimizar uma política de decisão $A(t) = \pi(S(t))$ à partir de um ambiente de simulação de navegação T que forneça a transição de estados $S(t+1) = f(S(t), A(t))$, ou seja P_r.

2.2 Definição de requisitos

As especificações definidas para realização da tarefa principal e secundária são resumidas a seguir:

2.2.1 Requisitos Primários (RP)

Os requisitos definidos que o sistema deve seguir para atingir a missão proposta são os seguintes.

RP1 (Natureza do Sistema): O sistema deve ser do tipo modelo de aprendizagem por reforço (APR), utilizando como proposta primária um algoritmo Q-Learning.

RP2 (Transição de estados): O sistema deve basear-se na transição de estados fornecida pelo simulador fast-time de navegação em águas restritas do TPN.

RP3 (Variáveis de controle): A política de controle aprendida pelo APR deve atuar nos comandos de leme e propulsor principal.

RP4 (Estados do navio): Os estados do navio, utilizados para construção da política de ação, devem se limitar à: Posição espacial do navio (X_{abs}, Y_{abs}), velocidade do navio (V_x, V_y), ângulo de aproamento (θ) e velocidade angular $(\dot{\theta})$.

RP5 (Step de simulação): Utilização do tempo de integração do simulador fixo de 0.5 s.

RP6 (Implementação): Utilização do algoritmo Q-Learning no ambiente de desenvolvimento python-Keras seguindo os modelos de implementação de interfaces do OpenAI gym.

RP7 (Trajetória e ambiente): A trajetória-guia deve ser definida como sendo o conjunto de coordenadas que definem um segmento de reta $(X_{inicial}, Y_{inicial}))$; (X_{final}, Y_{final}), e deve estar contida um canal de largura definida, o qual, por sua vez, deve ser definido como sendo o espaço E_{canal} contido entre dois segmentos de reta r_{sup} e r_{inf}.

RP8 (Função de recompensa): A função recompensa deve punir o desvio do navio em relação à trajetória-guia pré-definida (ou seja, recompensar a procissão da trajetória). A função recompensa deve utilizar somente os estados do navio citados em RP4 e as coordenadas da trajetória-guia e dos segmentos de reta que definem o canal.

RP9 (Intervalo de ação): O intervalo entre o envio de duas ações consecutivas para o simulador deve fixo e ser superior ao tempo de resposta do algoritmo.

RP10 (Desvio da trajetória): O desvio entre o centro do navio e a linha-guia (erro de tracking) deve ser $e_{desvio} < 0.5W$, onde W é a largura do navio.

RP11 (Desvio de velocidade): O desvio do setpoint de velocidade deve ser inferior à 10%.

2.2.2 Requisitos Secundários (RS)

Os requisitos secundários podem ser realizados para complementar o desenvolvimento deste trabalho, mas que não são necessários para o cumprimento da missão proposta.

RS1 (Comparação): Desenvolvimento e aplicação de outros algoritmos de APR para comparação com o método Q-learning.

RS2 (Complexidade): Geração de trajetórias-guia mais complexas após validação em trajetória linear inicial.

2.2.3 Fatores de performance (FP)

Os fatores de performance são indicadores da qualidade de execução e aprendizagem do sistema desenvolvido, os quais podem ser utilizados como métricas de avaliação de diferentes algoritmos, no caso do comprimento do requisito RS2. Os fatores de interesse para esse trabalho são descritos a seguir.

FP1 Número de iterações necessárias na fase de aprendizado para atingir os níveis de performance requisitados.

FP2 Número de iterações necessárias para levar o navio até a linha guia (que se traduz no tempo de resposta do controlador de APR).

2.3 Análise funcional

A análise funcional consiste em definir os principais blocos de construção do sistema projetado. Essa divisão, além de ser o suporte para a criação do modelo de APR, também facilita a compreensão do mesmo, exibindo as funcionalidades de cada bloco do sistema. As funcionalidades definidas são apresentadas a seguir, cada uma delas atua em diferentes modos de operação, como mostrado na seção 2.4.

F1 (Agente): O agente é o nome dado à parte do sistema que recebe o estado atual $S(t)$ do ambiente e, baseado em uma política de ação $A(t) = \pi(S(t))$, realiza uma ação de comando no navio para o controle de ângulo de leme $\beta(t)$ e da potência do propulsor $P_u(t)$.

F2 (Ambiente): Ambiente é nome dado à parte do sistema que recebe um conjunto de ações de controle leme e propulsor $A(t) = [\beta(t), P_u(t)]^t$ vindas do agente, e as envia, através de uma interface, ao simulador TPN. O ambiente obtém como resposta do simulador um conjunto de estados $S(t+1)$ vindos do simulador após o termino da iteração e armazena o estado como variável interna.

F3 (Recompensa): Recompensa é o nome dado à parte do sistema que recebe um conjunto de estados atuais do navio $S(t)$, que foram resultado de uma ação $A(t-1)$ tomada pelo agente em um estado $S(t-1)$, e calcula um valor para de recompensa $R(t) = \psi_r(S(t), A(t-1), S(t-1))$ para a ação tomada. Essa recompensa é baseada no cumprimento ou não dos objetivos propostos, qual sejam, seguir uma trajetória-guia proposta.

F4 (Aprendizado): Aprendizado é o nome dado à parte do sistema que recebe do sistema que recebe um conjunto de estados atuais do navio $S(t)$, que foram resultado de uma ação $A(t-1)$ tomada pelo agente em um estado $S(t-1)$, resultando em uma recompensa $R(t)$. A partir desses dados, o Aprendizado é responsável por atualizar a política de tomada de decisão $\pi(S(t))$, de forma a maximizar a soma de futuras recompensas $R(t+1)$.

F7 (Interface-simulador): Interface-simulador é o nome dado à parte do sistema responsável pela comunicação entre o ambiente de funcionamento do nosso sistema de APR (Python-shell) e o ambiente de simulação de navegação (Software Comercial TPN). Essa interface é utilizada para comunicação entre o Ambiente e o simulador.

F6 (Parâmetros-simulador): Parâmetros-simulador é o nome dado à parte do sistema responsável por configurar os parâmetros do simulador TPN no instante antes do início da simulação, tais como tempo de integração, condições ambientais para o navio e número de iterações.

F7 (Parâmetros-Aprendizado): Parâmetros-Aprendizado é o nome dado à parte do sistema responsável por configurar os parâmetros de aprendizado. Os parâmetros de aprendizado são a taxa de exploração ε, que é responsável pela parte aleatória da política de decisão $\pi(S(t))$, pela taxa de aprendizado α, que está relacionado com a

importância dada a novos eventos durante a fase de aprendizagem, e pelo desconto γ, que é a taxa de desconto dada à experiência obtida em iterações passadas.

F8 (Interface Gráfica): Interface-gráfica é o nome dado à parte do sistema responsável pela visualização da navegação gráfica do navio, e dos estados de interesse $S(t)$ do navio.

O diagrama a seguir resume a funcionalidade de cada uma das partes do sistema.

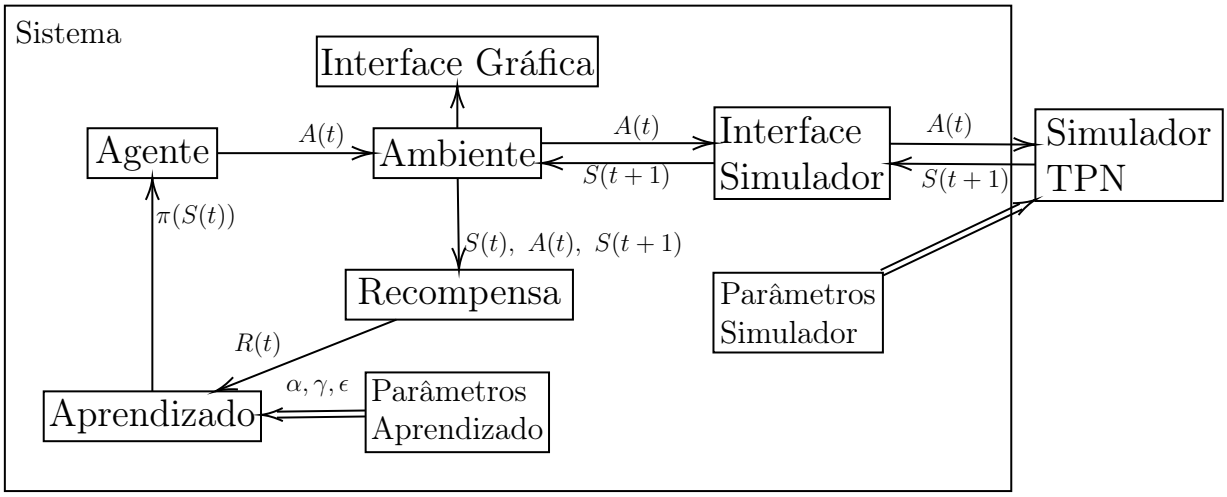

Figura 1: Diagrama de funcionalidades das partes do sistema

2.4 Modos de operação

Os modos de operação do sistema desenvolvidos são divididos em 3 partes:

- Aprendizagem: Nesse modo o sistema realiza a aprendizagem da política de ação a partir da iteração do algoritmo de APR com o simulador TPN. Durante essa etapa busca-se uma política ótima que maximize a recompensa recebida pelo agente.

- Ajuste: Esse modo é alternada com o modo de aprendizagem e consiste na escolha de hiper-parâmetros, tais como como a função de recompensa e os parâmetros de Aprendizagem, que são utilizados para avaliar o desempenho de aprendizagem do algoritmo

- Inferência: Esse modo é o modo de inferência em que o Agente já sabe controlar a navegação do navio, e não aprende mais com suas ações. Esse modo é o modo final no qual o sistema opera para navegar em águas restritas.

3 MACHINE LEARNING E APRENDIZAGEM

Este capítulo tem como objetivo apresentar ao leitor a teoria básica da aprendizagem de máquina que foi considerada relevante para a compreensão deste e de outros trabalhos apresentados como o estado da arte utilizado para a estruturação e desenvolvimento do projeto.

Primeiramente apresenta-se uma breve introdução sobre Machine Learning e os modelos de aprendizagem que são usualmente utilizados. Posteriormente apresenta-se o conceito de Redes Neurais Artificiais e como tais são usualmente utilizadas para a aprendizagem profunda. Em seguida, introduz-se os processos de Markov e sua relação com o problema de aprendizagem. Finalmente, apresenta-se o paradigma de Aprendizagem por Reforço e os métodos Q-learning, Q-learning aproximado, Deep Q Network e Deep Deterministic Policy Gradient.

3.1 Modelos de aprendizagem automática

O aprendizado de máquina é um campo de inteligência artificial que usa técnicas estatísticas para dar aos sistemas de computador a capacidade de "aprender"a partir de um conjunto de dados (por exemplo, melhorar progressivamente o desempenho de uma tarefa específica), sem ser explicitamente programado para isso.

Segundo a definição de M. Mitchell, a aprendizagem de máquina (machine learning) pode ser definida como: "Diz-se que um programa de computador aprende com a experiência E em relação a alguma classe de tarefas T, e medida de desempenho P, se seu desempenho nas tarefas em T, como medido por P, melhora com a experiência E."

Para "aprender", os métodos de aprendizagem utilizam técnicas estatísticas para construir modelos que permitem fazer previsões ou decisões guiadas a partir de um conjunto de dados utilizados para a construção dos parâmetros de decisão do algoritmo. O conjunto de dados em questão é genérico e pode referir-se à diversos tipos e estruturas, como por

exemplo, imagens de um tipo de objeto, trechos de texto, vetores de espaço de estados de um sistema simulado, entre outros.

A classificação é um exemplo de um dos tipos de tarefas realizadas frequentemente utilizando-se a aprendizagem automática. Alguns dos problemas que são abordados por algoritmos de classificação são a filtragem de e-mails de spam, reconhecimento de caracteres manuscritos, diagnósticos médicos, visão computacional, etc.

O aprendizado de máquina, porém, abrange temas genéricos e possui muitas subáreas de aplicação. Tipicamente, para fins de organização, os métodos de aprendizagem são divíduos em algumas grandes categorias, as quais dependem dos dados utilizados e do modo de aprendizagem empregado pelo algoritmo. As principais categorias são as seguintes:

- Aprendizado supervisionado: Realizado quando os dados utilizados possuem "etiquetas", que são fornecidas por algum "professor"(dados etiquetados previamente, podendo ser feitos à mão ou por outro método). A tarefa de aprendizado consiste então em determinar um modelo que leve as entradas (dados) às saídas (etiquetas) a partir de um treinamento realizado com um conjunto de dados de treino. Esse modelo é então utilizado para etiquetar novos dados. Exemplos tipo são as Máquinas de Vetores de Suporte (SVM), regressão linear, árvores de decisão, k vizinhos mais próximos, redes neurais artificiais e naive Bayes.

- Aprendizado não supervisionado: Nesse caso, utiliza-se dados não etiquetados, e o objetivo do aprendizado é encontrar padrões ou características latentes dos dados que permitam regrupá-los em categorias não definidas previamente.

- Aprendizado por reforço: Baseia-se na interação de um agente e um ambiente. O agente toma decisões seguindo uma política nesse ambiente, que influenciam sua transição de estados e a recompensa ganha a cada ação. O objetivo então é de encontrar uma política ótima, ou seja, que maximize a recompensa ganha pelo agente.

Este último tipo de aprendizagem é apropriado para a utilização no desenvolvimento de uma lei de controle para navios, visto que a situação se enquadra na interação entre um agente (controle do navio) e um ambiente (mar), e as decisões são os diferentes comandos possíveis (propulsão e ângulo de leme).

3.2 Aprendizagem supervisionada

Nesta seção são apresentados os métodos de Aprendizagem Supervisionada que foram utilizados neste trabalho.

3.2.1 Redes Neurais Artificiais

3.2.1.1 Estrutura de uma Rede Neural

As Redes Neurais Artificiais (RNA) são modelos computacionais inspirados nas redes neurais biológicas, originalmente inspirando-se no Sistema Nervoso Central dos animais, notadamente no cérebro e no funcionamento dos neurônios e suas conexões.

O modelo de redes neurais artificiais é construído através da interconexão de neurônios artificiais para a construção de uma rede neural (neural network), a qual é posteriormente treinada e utilizada para resolver um problema específico.

Nesse sentido, o neurônio artificial é a entidade central na concepção de Redes Neurais, pois é o elemento fundamental no qual a arquitetura dessas redes se baseia. A estrutura de um neurônio artificial com entrada $\mathbf{x} \in \mathbb{R}^d$ é composta de um vetor de pesos $\mathbf{w} = [w_1, w_2, ..., w_d]$, um viés b e uma função de ativação, que pode ser linear, tangente hiperbólica, ReLU, etc. O funcionamento do neurônio no passo direto (forward pass) pode ser dividido em duas partes: pré-ativação e ativação. A pré-ativação consiste na multiplicação da entrada $\mathbf{x} = [x_1, ..., x_d]^t$ pelo vetor de pesos $\mathbf{w}$ e subsequente soma so viés b, ou seja, $\mathbf{w} \times \mathbf{x} + b$. A ativação consiste então na passagem desse resultado pela função de ativação para obter a saída do neurônio. O diagrama da Figura 2 ilustra esse funcionamento.

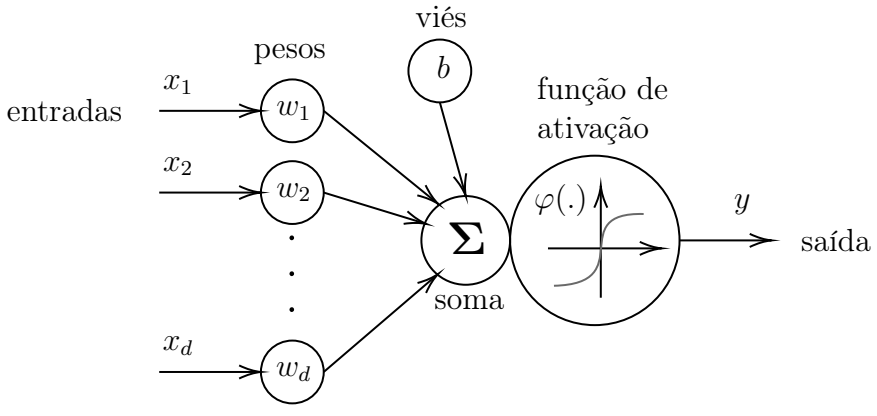

Figura 2: Estrutura de um neurônio artificial

As redes neurais são modelos flexíveis e os neurônios artificiais podem ser organizados de várias maneiras, dependendo da aplicação específica. No entanto, a maioria das redes neurais é estruturada em camadas, que são conjuntos de neurônios que não compartilham nenhuma conexão. Nesse caso, o cálculo flui sequencialmente de uma camada para a próxima, da entrada x para a saída y, interconectando-se os neurônios (N_{jk}) em configurações como exemplificado na Figura 3, onde o número de camadas l e de neurônios por camada n_i varia dependendo da aplicação e da complexidade do problema. Quando temos mais que duas camadas em uma RNA costuma-se falar então em Aprendizagem Profunda (deep learning, em inglês).

Como esse tipo de rede não inclui nenhum ciclo em seu gráfico computacional, ele é chamado de rede feedforward. Cada camada em uma rede neural feedforward pode ser interpretada como uma função em si e em toda a rede como composição dessas funções. Como exemplo, se $f^{(l)}$ é a função implementada pela camada l e temos três camadas na rede, podemos escrever $f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$.

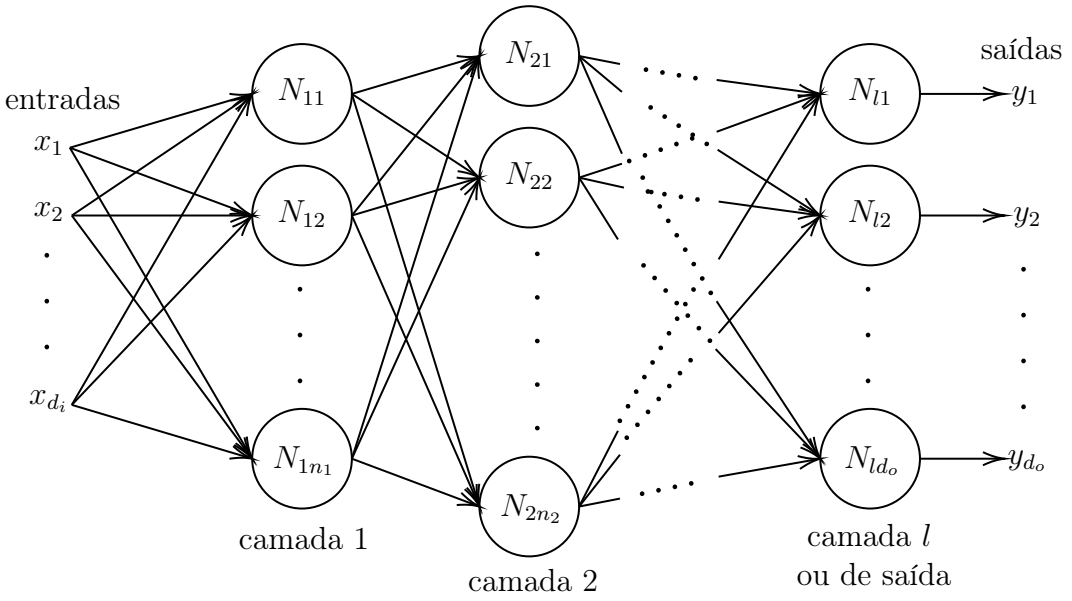

Figura 3: Arquitetura de uma Rede Neural

Além das redes feedfowards, existem redes neurais de variados tipos: convolucionais (CNN), Long Short-Term Memory (LSTM), Recurrent Neural Networks (RNN), entre outras, as quais fogem do escopo deste trabalho. As RNAs podem ser utilizadas para resolver problemas de aprendizado supervisionado, não-supervisionado e como parte da solução de aprendizado por reforço, como detalhado nas seções 3.4.4.3 e 3.4.5.1.

Na próxima seção os detalhes do mecanismo de aprendizado de RNAs são abordados.

3.2.1.2 Aprendizagem

De maneira geral, uma rede neural que mapeia uma entrada x para uma saída y tem como objetivo aproximar uma função $y = f^*(x)$. A saída da RNA pode ser escrita com um conjunto de parâmetros $\theta : \hat{y} = f(x; \theta)$, e o objetivo da etapa de aprendizado é encontrar o conjunto de parâmetros θ que melhor aproxima $f^*(x)$. Esse objetivo pode ser traduzido como a minimização de uma função de perda (ou função de custo, loss function em inglês) $L(x, y, \theta)$, que fornece uma medida da diferença entre $f(x, \theta)$ e $f^*(x)$.

Um exemplo recorrente utilizado como função de custo é o erro quadrático médio (EQM, ou MSE em inglês):

$$L(x, y, \theta) = (y - f(x, y, \theta)^2) \ . \tag{3.1}$$

Entretanto, a avaliação do modelo é realizada geralmente através da esperança da função de custo afim de avaliar o melhor fit de um conjunto de dados com o modelo. Dessa maneira, define-se a função de custo como sendo a esperança matemática da perda:

$$J(\theta) = E[L(x, y, \theta)] \ . \tag{3.2}$$

A RNA "aprende" a aproximar-se de $f^*(x)$ atualizando seus parâmetros θ, geralmente utilizando o método de otimização chamado gradiente descendente. Tal método atualiza θ gradualmente na direção inversa do gradiente da função de custo em relação à θ, para buscar a minimização de tal custo.

$$\theta \leftarrow \theta - \epsilon \nabla J(\theta) \ . \tag{3.3}$$

Existem também métodos derivados do gradiente descendente como o método de Adam (15), o qual é utilizado posteriormente nesse trabalho.

Esse método pode ser aplicado sem perda de generalidade em casos onde a função $y = f^*(x)$ não é conhecida, mas é conhecida uma função custo, ou função objetivo, a qual é função dos estados e deseja-se minimizar. Detalhes sobre o processo de aprendizagem por redes neurais utilizando APR são apresentados nas seções 3.4.4.3 e 3.4.5.1.

3.3 Processos de Markov

3.3.1 Cadeias de Markov

Os processos de Markov descrevem processos estocásticos caracterizados pela propriedade markoviana: "O futuro é independente do passado dado o presente". Os processos de Markov são utilizados para descrever modelos estocásticos de transições de estados em que a probabilidade de transição de um estado presente para um estado futuro só depende do estado presente. Tem-se então:

$$P(S_{t+1} \mid S_t) = P(S_{t+1} \mid S_1, S_2, S_3, \ldots, S_t), \tag{3.4}$$

onde $P(S_{t+1} \mid S_t)$ representa a probabilidade de transição do S_t para o estado S_{t+1}. A propriedade markoviana diz então que um estado S_t captura toda informação relevante do histórico de estados anteriores $(S_1, S_2, ..., S_{t-1})$ e é suficiente para determinar a probabilidade de transição para o próximo estado S_{t+1}. Um processo de Markov (ou cadeia de Markov) é então definido pela dupla $< \mathrm{S}, \mathrm{P} >$, onde:

- S é o conjunto de estados possíveis;
- P é a matriz de transição de estados, onde $P_{ss'} = P(S_{t+1} = s' \mid S_t = s)$.

A matriz de transição de Markov para n estados possíveis é definida como:

$$P = \begin{pmatrix} P_{11} & P_{12} & \ldots & P_{1n} \\ & \ldots & & \\ P_{n1} & P_{n2} & \ldots & P_{nn} \end{pmatrix} . \tag{3.5}$$

3.3.2 Processos de Recompensa de Markov

Por extensão, um Processo de Recompensa de Markov (PRM) é um processo markoviano onde as transições de estados estão associadas à uma recompensa R. Tal processo é definido então pela quádrupla $< \mathrm{S}, \mathrm{P}, \mathrm{R}, \gamma >$, onde.

- S, P são como descritos anteriormente (vide seção 3.3.1);
- R é a função de recompensa, $R_s = \mathbb{E}[R_t + 1 \mid S_t = s]$;
- $\gamma \in [0, 1]$ é um fator de desconto.

Um PRM visa avaliar um conjunto de transições segundo uma métrica de recompensas para cada transição. Para tanto, uma função conhecida como retorno (G_t), avalia o conjunto de recompensas para cada etapa de transição:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \ . \tag{3.6}$$

O valor de γ pondera as recompensas futuras em uma avaliação no presente, ou seja a recompensa R recebida na (k+1)-ésima transição tem valor $\gamma^k R$. Ele garante a convergência de recompensas acumuladas em processos que possuem horizonte infinito (não tem fim previsto). De uma forma geral, se γ é próximo de 1 temos uma avaliação ponderada no futuro muito longo, se γ é próximo de 0 temos avaliação pautada no futuro muito próximo.

Cada estado de uma cadeia de Markov associada a um PRM é associado a uma função de valor de estados $V(s)$ que dá o valor de longo-prazo do estado, e é definida por:

$$V(s) = \mathbb{E}[G_t \mid S_t = s] \ . \tag{3.7}$$

O valor de $V(s)$ pode pode ser reescrito como função da recompensa imediata e da recompensa futura:

$$V(s) = \mathbb{E}[R_{t+1} + \gamma V(S_{t+1}) \mid S_t = s] \ . \tag{3.8}$$

Como a transição de estados está associada à uma probabilidade de transição, podemos reescrever a função de valor como se segue:

$$V(s) = R_s + \gamma \sum_{s' \in S} P_{ss'} V(s') \ . \tag{3.9}$$

A figura a seguir ilustra essa transição a recompensa associada e a função de valor correspondente.

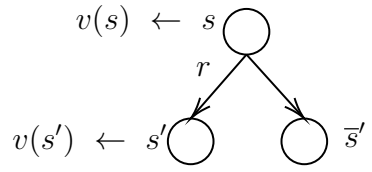

Figura 4: Representação de um PRM

A equação de Bellman é utilizada para definir o valor de V(s) associados para todos os estados de uma cadeia markoviana de um PRM:

$$v = R + \gamma P v \ . \tag{3.10}$$

A equação pode ser resolvida no caso de PRM, sendo definida como:

$$v = (I - \gamma P)^{-1} R \ . \tag{3.11}$$

3.3.3 Processo de Decisão de Markov

Um processo de decisão de Markov (PDM) é um processo de controle estocástico em tempo discreto. Ele fornece uma estrutura matemática para modelar a tomada de decisões em situações em que os resultados são parcialmente aleatórios.

PDMs são utilizados na modelagem e formulação de problemas de APR onde o ambiente é totalmente observável. PDMs são uma extensão de um PRM onde as transições estão associadas à ações tomadas antes da transição. Um Processo de Decisão de Markov (PDM) é definido como a quíntupla $< \mathrm{S}, \mathrm{A}, \mathrm{P}, \mathrm{R}, \gamma >$, onde:

- S é o conjunto de estados possíveis;
- A é o conjunto de ações que o agente pode tomar;
- P é a probabilidade de transição de estado $p(s, a, s')$;
- R é a recompensa recebida pelo agente, definida pela função $r(s, a) : \ \mathrm{S} \times \mathrm{A} \rightarrow \mathrm{R}$;
- γ é o fator de desconto.

Um processo de decisão de Markov possui a propriedade markoviana, ou seja todas as transições de estado dependem somente do estado atual e da ação tomada pelo agente, ou seja o processo não possui memória (vide seção 3.3.1). No escopo de um PDM, temos então:

$$P(s_{t+1} \mid s_t, a_t) = P(s_{t+1} \mid s_1, s_2, \ldots, s_t, a_1, a_2, ..., a_t) \ . \tag{3.12}$$

O objetivo é escolher uma política π que maximize algumas funções cumulativas das recompensas aleatórias, tipicamente uma função G_t que computa a soma de recompensas esperada em um horizonte potencialmente infinito. Uma política π é uma distribuição de ações realizadas para um dado estado:

$$\pi(a \mid s) = P(A_t = s \mid S_t = s) \ . \tag{3.13}$$

Uma política define totalmente o comportamento do Agente em um determinado ambiente.

Dado um PDM $M = < \mathrm{S}, \mathrm{A}, \mathrm{P}, \mathrm{R}, \gamma >$ e uma política π, a sequência de estados S é um processo de Markov em $< \mathrm{S}, \mathrm{P}^{\pi} >$, e a sequência de estados e recompensas $< \mathrm{S}, \mathrm{P}^{\pi}, \mathrm{R}^{\pi}, \gamma >$ define um PRM. Onde

$$P^{\pi}_{ss'} = \sum_{a \in A} \pi(a) P^{a}_{ss'} \ , \tag{3.14}$$

$$R^{\pi}_{s} = \sum_{a \in A} \pi(a) R^{a}_{s} \ . \tag{3.15}$$

No caso do PDM, a função de valor, $V(s)$ apresentada na seção anterior, é redefinida como sendo a esperança do retorno G_t começada no estado s e seguindo a política π:

$$V_{\pi}(s) = \mathbb{E}_{\pi}[G_t \mid S_t = s] \ . \tag{3.16}$$

Uma nova função que avalia as ações tomadas em um determinado é definida, a função de valor da ação $Q_{\pi}(s, a)$ é a esperança dos retornos começando no estado s e realizando a ação a, quando seguindo a política π:

$$Q_{\pi}(s, a) = \mathbb{E}_{\pi}[G_t \mid S_t = s, A_t = a] \ . \tag{3.17}$$

Ambas as funções de valor da ação e a de valor do estado podem ser reescritas de forma recursiva, como foi feito na seção anterior:

$$V_{\pi}(s) = \mathbb{E}_{\pi}[R_{t+1} + \gamma V_{\pi}(S_{t+1}) \mid S_t = s] \ , \tag{3.18}$$

$$Q_{\pi}(s, a) = \mathbb{E}_{\pi}[R_{t+1} + \gamma Q_{\pi}(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a] \ . \tag{3.19}$$

As equações de Bellman podem ser reescritas para cada uma das funções, e representam o valor para as funções para um estado genérico s. As figuras a seguir ilustram o processo para o cálculo das equações de Bellman. Na figura da esquerda, o valor de V_{π} é associado à probabilidade da tomada de uma ação a que é função de uma distribuição de $\pi(a \mid s)$. Na figura da direita, o valor de $Q_{\pi}(s, a)$ é dependente da recompensa imediata r e da função de valor associada ao novo estado de transição, $v_{\pi}(s')$.

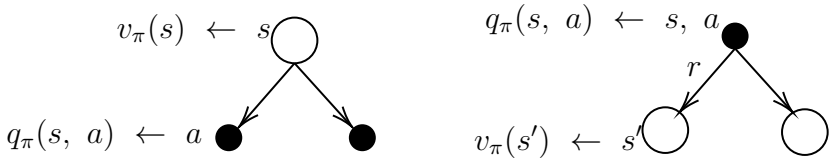

Figura 5: Representação de um PDM, à esquerda, a dependência do valor de V_{π} , à direita, a dependência do valor de $Q_{\pi}(s, a)$.

Em síntese tem-se que:

$$V_\pi(s) = \sum_{a \in A} \pi(a \mid s) Q_\pi(s, a) \ , \tag{3.20}$$

$$Q_\pi(s, a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V_\pi(s') \ . \tag{3.21}$$

Usando as duas equações anteriores pode-se escrever as funções de valor de estados e de valor de ações de forma recursiva:

$$V_\pi(s) = \sum_{a \in A} \pi(a \mid s)(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V_\pi(s')) \ , \tag{3.22}$$

$$Q_\pi(s, a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \sum_{a' \in A} \pi(a' \mid s') Q_\pi(s', a') \ . \tag{3.23}$$

Pode-se então definir que a função de valor de estados e de ações ótima é aquelas cujos valores de $V_*(s)$ e $Q_*(s)$, respectivamente, são máximos sobre todas as políticas:

$$V_*(s) = \max_\pi V_\pi(s) \ , \tag{3.24}$$

$$Q_*(s, a) = \max_\pi Q_\pi(s, a) \ . \tag{3.25}$$

Toda política ótima alcança a melhor performance em termos das duas funções de valores. E a busca pela política ótima é o que define a motivação da aprendizagem, pois não existe uma solução geral para a equação de Bellman, e a maioria dos problemas é solucionado por soluções iterativas como nos algoritmos de Iteração de Valor, Iteração de Política, Q-learning e SARSA.

3.4 Aprendizagem por Reforço

A Aprendizagem por Reforço (APR) tem como objetivo geral ensinar a realização de uma tarefa a um Agente por meio de decisões sequenciais tomadas por ele (Sutton e Barto(16)). A cada iteração, o agente observa um estado ($s \in S$) que o Ambiente fornece, toma uma ação ($a \in A$) nesse Ambiente que causa uma transição desse estado para um outro ($s' \in S$) e recebe uma Recompensa ($r \in R$) em virtude dessa ação.

Dessa maneira, quando os problemas de APR são formulados com probabilidades de transição bem definidas, eles constituem um PDM, que se baseia na quíntupla
$< \mathrm{S}, \mathrm{A}, \mathrm{P}, \mathrm{R}, \gamma >$ (cf. item 3.3.3).

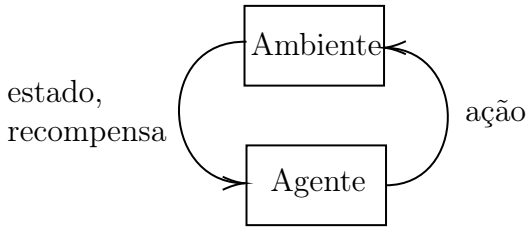

Figura 6: Cenário geral de um problema de APR

3.4.1 Objetivo principal da APR

Dada a estrutura do problema, o objetivo dos algoritmos de APR é encontrar políticas ótimas que maximizem a recompensa acumulada. Essas políticas definem a ação que o agente deve tomar dado o estado que ele se encontra ($\pi(s) : \mathrm{S} \rightarrow \mathrm{A}$). As recompensas dadas pelo ambiente servem para moldar o comportamento do agente, então a simples concepção das recompensas já constitui sozinha um problema complexo a ser resolvido, e diversas heurísticas podem ser usadas para definí-las, pois a política ótima a ser encontrada pelo agente depende fortemente delas.

Como mostrado na seção anterior, o PDM possui métricas de valor de estados e de ações, que traduzem a esperança da recompensa seguindo certa política. Ora, encontrar a política ótima π^* pode se resumir então a encontrar a política que maximize tais métricas. Quando tem-se o conhecimento completo do PDM, inclusive das probabilidades de transição de estados (ou se ela é determinística), pode-se aplicar métodos de Programação Dinâmica para encontrar π^*. Outra opção são os métodos que aproximam a função Q_π durante a aprendizagem, que pode ser usada posteriormente para definir a política, escolhendo-se sempre a ação que maximiza Q no estado atual. Métodos bastante utilizados e que tem isso como base são o Q-learning e suas variantes.

3.4.2 Programação Dinâmica

Quando tem-se um modelo definido que descreve o comportamento do ambiente como um PDM (geralmente suposto finito) e o conhecimento das probabilidades de transição de estados, existe uma coleção de algoritmos que podem ser usados para encontrar políticas ótimas, a qual é chamada de Programação Dinâmica (PD). Sua aplicação, no entanto, é limitada em APR dada sua premissa de um modelo descritivo perfeito do ambiente, e seu custo computacional elevado (16).

A PD, e mesmo a APR em geral, utiliza as funções de valor (equações 3.16 e 3.17) para

organizar e estruturar a busca por políticas melhores. Isto ocorre pois ao encontrar-se as funções de valor ótimas - equações 3.24 e 3.25 - a política ótima π^* também é encontrada:

$$\pi^*(s) = \arg\max_{\pi}(V_\pi(s)),\ \forall s \in \mathrm{S}. \tag{3.26}$$

3.4.2.1 Avaliação de Política

Primeiramente considera-se o problema de computar a função de valor do estado V_π para uma política arbitrária π. Toma-se então a equação 3.22. Se a dinâmica do ambiente é conhecida, isso quer dizer que R_s^a e $P_{ss'}^a$ são conhecidos. Portanto, encontrar V_π se resume a resolver um sistema de equações lineares com o número de equações e de incógnitas iguais ao número de estados. Porém, o número de estados frequentemente é elevado, o que torna inviável a resolução desse sistema. Prefere-se então uma solução iterativa para esse problema, a qual aproveita a equação 3.22 para atualizar a estimativa da função de valor do estado. Tem-se então:

$$\begin{aligned} V_{k+1}(s) &\doteq \mathbb{E}_\pi[R_{t+1} + \gamma V_k(S_{t+1}) \mid S_t = s] \\ &= \sum_{a \in A} \pi(a \mid s)(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V_k(s'))\ . \end{aligned} \tag{3.27}$$

Mostra-se que $V_k \to V_\pi$ para $k \to \infty$ com as mesmas condições que garantem a existência de V_π (16). Para obter-se a estimação, deve-se então definir um limite de precisão θ como critério de parada do algoritmo iterativo de Avaliação da Política mostrado a seguir:

Algoritmo 1: Avaliação de Política iterativa (16)

Entrada: π, a política a ser avaliada

Dados: um pequeno limite $\theta > 0$ delimitando a precisão da estimação

Saída: V, estimação de V_π

Inicializar $V(s)$ arbitrariamente para todos estados, exceto o terminal onde $V = 0$

repita
  $\Delta \leftarrow 0$
  para cada $s \in$ S faça
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s)(R_s^a + \gamma \sum_{s'} P_{ss'}^a V(s'))$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
  fim
até $\Delta < \theta$;

3.4.2.2 Iteração de Política

Uma vez obtida a estimação de V_π, e sabendo-se as probabilidades de transição de estados, a política atual pode ser melhorada simplesmente buscando-se as ações que levam a estados s com o maior $V_\pi(s)$ possível. O algoritmo que se aproveita dessa estratégia para melhorar a política é mostrado abaixo:

Algoritmo 2: Iteração de Política (usando avaliação de política iterativa) (16)

Saída: $\pi \approx \pi^*$ e $V \approx V_*$

Inicialização: $V(s) \in \mathbb{R}$ e $\pi(s) \in$ A arbitrariamente $\forall s \in$ S

repita

1. Avaliação de Política:

repita

$\Delta \leftarrow 0$

para cada $s \in$ S faça

$v \leftarrow V(s)$

$V(s) \leftarrow \sum_a \pi(a|s)(R_s^a + \gamma \sum_{s'} P_{ss'}^a V(s'))$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

fim

até $\Delta < \theta$;

2. Melhorar Política:

$politica\_estável \leftarrow verdadeiro$

para cada $s \in$ S faça

$última\_ação \leftarrow \pi(s)$

$\pi(s) \leftarrow \arg\max_a(R_s^a + \gamma \sum_{s'} P_{ss'}^a V(s'))$

Se $última\_ação \neq \pi(s)$, então $politica\_estável \leftarrow falso$

fim

até $politica\_estável$;

3.4.2.3 Iteração de Valor

A Iteração de Política, apesar de convergir para a política ótima no limite, pode ser demasiadamente custosa em processamento, uma vez que cada iteração realiza uma Avaliação de Política, o que acaba tornando o algoritmo ineficiente em certos casos. Porém, se ao invés de se aguardar a convergência da Avaliação de Política, para-se o algoritmo nas primeiras iterações, pode-se haver uma redução significativa no custo computacional do algoritmo, sem perder a garantia de convergência da Iteração de Política (16). É o caso da Iteração de Valor, onde para-se a Avaliação da Política após apenas uma atualização

de cada estado. O algoritmo detalhado é mostrado a seguir:

Algoritmo 3: Iteração de Valor (16)

Dados: um pequeno limite $\theta > 0$ delimitando a precisão da estimação

Saída: $\pi \approx \pi^*$

Inicializar $V(s)$ arbitrariamente para todos estados, exceto o terminal onde $V = 0$

repita
  $\Delta \leftarrow 0$
  para cada $s \in$ S faça
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a(R_s^a + \gamma \sum_{s'} P_{ss'}^a V(s'))$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
  fim
até $\Delta < \theta$;

Retornar $\pi(s) = \arg\max_a(R_s^a + \gamma \sum_{s'} P_{ss'}^a V(s'))$

3.4.3 Métodos de Monte Carlo

Os métodos de Monte Carlo (MC)(16), contrariamente aos de Programação Dinâmica, não assumem conhecimento completo do ambiente (são ditos Model-Free), e requerem somente amostras experimentais de sequências de estados, ações e recompensas de interações com o ambiente (simuladas ou reais). Tais métodos são interessantes pois não requerem conhecimento prévio da dinâmica do ambiente, porém conseguem ainda assim alcançar políticas ótimas. Além disso, aprender a partir de experiências simuladas (amostras de transições) permite o emprego de tais métodos em uma vasta gama de situações onde gerar as distribuições completas do ambiente de forma explícita é impossível ou extremamente complexo.

Tais métodos resolvem o problema de APR baseado em médias de amostras de retorno. As tarefas consideradas para os métodos de Monte Carlo são episódicas (a experiência é dividida em episódios e todos episódios eventualmente terminam independentemente das ações selecionadas), para assegurar que retornos bem definidos são disponíveis. Os estimadores de valor e a política são atualizados então somente no final dos episódios. Portanto, os métodos MC são offline, incrementando a cada episódio, ao contrário dos métodos online, que atualizam-se a cada passo da simulação.

Para aprender, utiliza-se uma adaptação da generalização da Iteração de Política. Ao contrário da PD, que usa o conhecimento do PDM para computar funções de valor, MC utiliza amostras de retorno do PDM para aprender as funções de valor. Como descrito

na seção de Programação Dinâmica (3.4.2), considera-se primeiramente o problema da Avaliação de Política (computando-se V_π e Q_π para uma política arbitrária e fixa π), para posteriormente aprimorar a política e utilizá-la no problema de controle. Cada uma dessas ideias extraídas da PD são estendidas para o caso onde somente amostras de experiência estão disponíveis (Monte Carlo).

3.4.4 Aprendizagem por Diferenças Temporais

A aprendizagem por Diferenças Temporais (DT)(16) combina ideias dos métodos de Monte Carlo com Programação Dinâmica. Os métodos de DT aprendem diretamente da simples experiência sem um modelo da dinâmica do ambiente, como os métodos de MC. Porém, assim como na PD, a atualização dos estimadores em DT é feita em parte baseada em outros estimadores aprendidos (online), sem esperar um resultado final episódico. Alguns métodos clássicos de DT são a Previsão por Diferenças Temporais (TD(0), TD(λ), ...), SARSA e Q-learning. Este último é descrito em detalhes a seguir.

3.4.4.1 Q-learning

O algoritmo Q-learning (17) é um algoritmo de controle por DT dito off-policy, ou seja, que não segue a política sendo otimizada na fase de aprendizagem (possui política própria durante essa fase). A fórmula de atualização da estimação da função de valor de ações é a seguinte:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]. \tag{3.28}$$

Prova-se que neste caso, a função de valor de ações aprendida, Q, aproxima diretamente Q_*, a função de valor de ações ótima, independentemente da política sendo seguida (16). A condição requerida para essa convergência é a de que o valor de todos os pares de estado-ação continue a ser atualizado durante a aprendizagem. Por isso, a política tem um efeito que determina quais pares são visitados e atualizados. Geralmente utiliza-se uma política chamada ε-greedy, que com probabilidade ε escolhe aleatoriamente uma ação a ser tomada, e com probabilidade $1 - \varepsilon$ escolhe a ação que maximiza a função de valor de ações ("ambiciosa"). Esta política equilibra exploração do conhecimento já adquirido (representado pela função de valor) e experimentação de novas ações (escolhidas

aleatoriamente). O algoritmo detalhado de Q-learning é mostrado abaixo:

Algoritmo 4: Q-learning (Controle por DT off-policy) (16)

Dados: taxa de aprendizagem $\alpha \in [0, 1]$, $\varepsilon \in]0, 1]$ pequeno

Saída: $\pi \approx \pi^*$

Inicializar $Q(s, a)$ arbitrariamente para todos estados e ações, exceto para o estado terminal onde $Q = 0$

para cada episódio faça

  Inicializar S

  para cada passo do episódio faça

    Escolher A no estado S usando política derivada de Q (p.ex. ε-greedy)

    Tome a ação A, observe R, S'

    $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$

    $S \leftarrow S'$

  fim

fim

Retornar $\pi(s) = \arg\max_a Q(s, a)$

O algoritmo mostrado funciona muito bem quando tem-se estados e ações discretos de dimensão não muito elevada, utilizando-se Q tabular. Em casos contínuos, existe ainda a possibilidade de discretizar estados e ações. No entanto, conforme a quantidade de estados e ações aumenta, a complexidade do algoritmo também cresce. A atualização de Q pela visita a todos pares de estado-ação torna-se custosa, e a convergência do algoritmo, lenta. Além disso, quando tem-se estados com várias dimensões, o tamanho da tabela que armazena Q torna-se excessivamente grande, e o problema torna-se difícil de resolver. Possíveis soluções para tais problemas utilizando princípios do Q-learning são mostradas a seguir (Q-learning aproximado e Deep Q Network).

3.4.4.2 Q-learning aproximado

O Q-learning aproximado(16) utiliza uma aproximação de função para a função de valor de ações Q. Ao invés de se utilizar uma tabela para armazená-la, aproxima-se Q por uma forma funcional parametrizada com vetor de pesos $\mathbf{w} \in \mathbb{R}^d$. Denota-se tal aproximação por $\hat{Q}(s, a, \mathbf{w}) \approx Q_\pi(s, a)$. Essa aproximação pode ser a mais variada possível, podendo ser:

- um modelo de regressão linear, onde $\mathbf{w}$ denomina os pesos da função linear;
- uma árvore de decisão, onde $\mathbf{w}$ define os nós e valores das folhas;

- ou até mesmo uma RNA multi-camadas (vide seção 3.2.1), onde $\mathbf{w}$ é o vetor dos pesos dos neurônios em todas as camadas.

Tipicamente, o número de pesos (dimensão de $\mathbf{w}$) é bem menor que o número de estados: $d \ll |\mathrm{S}|$. Portanto, mudar um peso afeta o valor estimado de vários estados, o que por consequência, faz com que a atualização de um estado seja generalizada pelo modelo e afete o valor de vários outros. Tal generalização é interessante pois torna o aprendizado potencialmente mais poderoso porém também mais difícil de controlar e entender. Outro aspecto interessante é que tais métodos de APR permitem sua aplicação para problemas parcialmente observáveis (nos quais o estado completo não é disponível para o agente).

3.4.4.3 Deep Q Network

Deep Q Network (DQN) é uma arquitetura de APR proposta por Mnih et al.(18) para aprender a jogar 49 jogos clássicos de Atari 2600, utilizando uma observação do estado pelo agente por meio somente da visualização do jogo (pixels que constituem a tela do jogo). Para tal, os autores aproximam a função de valor de ação ótima por meio de uma Rede Neural Convolucional (CNN) que recebe a observação do estado como entrada e retorna os valores de ação como saída. Os autores propõem também uma rotina de treinamento por replay de experiência, ou seja, pegam amostras aleatórias de um conjunto de passos experimentais para utilizar no treinamento da RNA, eliminando a correlação na sequência das observações (que é prejudicial para o treinamento da RNA) e suavizando as mudanças na distribuição de dados. Portanto, durante a aprendizagem, as atualizações do Q-learning são feitas com amostras (ou minibatches) de experiência (s, a, r, s'), retirados uniformemente de maneira aleatória das amostras experimentais armazenadas em $U(D)$ (replay). A função custo utilizada na atualização dos pesos θ_i da Rede Q (Q Network, em inglês) na i-ésima iteração do algoritmo é a seguinte:

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim U(D)}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i)\right)^2\right]; \tag{3.29}$$

onde:

- γ é o fator de desconto;
- θ_i são os parâmetros da Rede Q na i-ésima iteração;
- θ_i^- são os parâmetros da RNA utilizada para computar o alvo - $\max_{a'} Q(s', a'; \theta_i^-)$ - na i-ésima iteração.

O treinamento da RNA é feito com o método de Adam (15). Os parâmetros da RNA que computa o alvo (θ_i^-) são somente atualizados com os parâmetros da Rede Q (θ_i) a cada C passos, sendo fixos entre atualizações individuais, o que é denominado de Atualização Dura do Modelo Alvo (Hard Target Model Update, em inglês). Outra opção que pode ser utilizada, a Atualização Mole do Modelo Alvo (Soft Target Model Update) atualiza os parâmetros a cada iteração do algoritmo, por meio de uma atualização que pondera a importância dada ao modelo anterior e o novo segundo a equação a seguir (C é o parâmetro que define essa ponderação):

$$\theta_i^- = C \times \theta_i^- + (1 - C) \times \theta_i \ . \tag{3.30}$$

Mesmo se a DQN atinge bons resultados em sistemas com dimensões elevadas, o espaço de ações do método ainda é discreto. No entanto, muitas tarefas de interesse, especialmente de controle (inclusive a tarefa analisada neste trabalho), possuem espaço de ações contínuo. Se a discretização de tal espaço de ações é muito fina, o espaço de ações torna-se muito grande e a complexidade do problema aumenta, dificultando a convergência do método.

O interesse da aplicação desse algoritmo para a esse trabalho é que ele pode ser utilizado para a construção de uma política de controle de leme e de propulsão discreta, utilizando uma aproximação para a função com relativamente poucos parâmetros da rede neural. A contrapartida desse método é que ele deve ser aplicado em conjunto com a discretização das ações de leme e propulsão.

3.4.5 Gradiente de Política

Os métodos de Gradiente de Política (GP) são diferentes dos expostos anteriormente na medida em que eles não utilizam mais uma estimativa da função de valor de ações para constituir uma política (selecionando-se ações que maximizem a função). Ao invés disso, tais métodos aprendem uma política parametrizada que pode selecionar ações sem consultar uma função de valor. No entanto, uma função de valor pode ainda ser usada para aprender o parâmetro da política (16). Similarmente ao procedimento de gradiente descendente descrito na seção 3.2.1.2, que fala sobre o processo de otimização das RNAs, os métodos de GP realizam o gradiente ascendente, utilizando uma métrica de performance ao invés de custo. Por isso, para maximizar o desempenho, sobe-se o gradiente ao invés de descê-lo, como é feito com o custo. Analogamente à equação 3.3, o GP segue então a

seguinte regra geral de atualização (com $J(\theta)$ sendo agora uma medida de performance):

$$\theta \leftarrow \theta + \epsilon \nabla J(\theta) \ . \tag{3.31}$$

Costuma-se denominar a arquitetura ator-crítico como métodos que aprendem aproximações para ambas a política e função de valor. Ator refere-se à política aprendida, enquanto crítico é uma referência à função de valor estimada, geralmente uma função de valor de estado.

3.4.5.1 Deep Deterministic Policy Gradient

O método Deep Deterministic Policy Gradient (DDPG), proposto por Lillicrap et al.(19), é comumente aplicado para ambientes em que o espaço de ações é contínuo, como é o caso do problema de controle de ações de leme e propulsão em navegação.

Tal método apoia-se na arquitetura ator-crítico, a qual é usado para representar a função de política independentemente da função de valor. A estrutura da função de política é conhecida como o ator e a estrutura da função de valor é referida como crítico. O ator produz uma ação, dado o estado atual do ambiente, e o crítico produz um sinal de erro DT (Diferença Temporal), dado o estado e a recompensa resultante. A saída do crítico impulsiona a aprendizagem tanto no ator quanto no crítico. O diagrama a seguir ilustra essa estrutura.

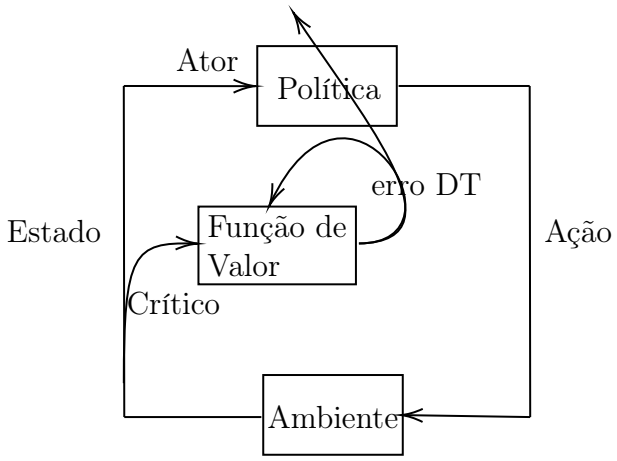

Figura 7: Diagrama ator-crítico

No caso do DDPG utilizamos um algoritmo de política determinística, ou seja,

$$a = \mu_\theta(s_t|\theta^\mu) \tag{3.32}$$

ao invés de $\pi_\theta(a|s) = P(a|s,\theta)$ como nos casos estocásticos anteriores. Dessa forma o

DDPG o gradiente é calculado apenas sobre o espaço de ações, o que exige um número de amostras menor que no caso estocástico. Entretanto uma política determinística não explora plenamente o espaço de estados, dessa forma, para superar essa limitação, utiliza-se a adição de um processo de ruído $\mathcal{N}_t$. Dessa forma tem-se:

$$a = \mu_\theta(s_t|\theta^\mu) + \mathcal{N}_t \ . \tag{3.33}$$

Um crítico é usado para avaliar a política estimada pelo ator segundo o erro de DT:

$$y_i = r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \ . \tag{3.34}$$

Assim como no caso do DQN é utilizado a técnica de replay de experiências para tirar a correlação das sequências de observações.

Atualizar diretamente o seu ator e os pesos críticos da rede neural com os gradientes obtidos a partir do sinal de erro de DT calculado faz com que seu algoritmo de aprendizado divirja (ou seja, não aprenda). Para contornar esse problema utiliza-se um conjunto de redes-alvo (target, em inglês) para gerar os alvos para o cálculo de erro de DT o qual regulariza o algoritmo de aprendizado e aumenta a estabilidade da solução, similarmente como é feito para o método DQN. As equações para o alvo DT yi e a função de perda para a rede do crítico são as seguintes, respectivamente:

$$y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'}) \ , \tag{3.35}$$

$$L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q)^2) \ . \tag{3.36}$$

Nessa equação, um minibatch de tamanho N foi amostrado a partir do buffer de reprodução, com i o índice referente à amostra. O alvo para o cálculo do erro de DT, y_i, é calculado a partir da soma da recompensa imediata e das saídas das redes-alvo do ator e do crítico, tendo pesos $\theta^{\mu'}$ e $\theta^{Q'}$ respectivamente. Então, a saída do crítico pode ser computada como sendo $Q(s_i, a_i|\theta^Q)$.

O pseudo-código que define o DDPG é demonstrado a seguir.

Algoritmo 5: Deep Deterministic Policy Gradient (19)

Inicializar $\tau << 1$

Inicializar a rede do crítico $Q(s, a|\theta^Q)$ e a rede do ator $\mu(s|\theta^\mu)$ arbitrariamente com pesos θ^Q e θ^μ

para cada episódio = 1, M faça

- Inicializar o processo aleatório de exploração $\mathcal{N}_t$
- Receber o estado inicial de observação s_1
- para cada iteração t= 1, T faça
  - Escolher $a_t = \mu_\theta(s_t|\theta^\mu) + \mathcal{N}_t$ usando política de exploração e ruído.
  - Tome a ação a_t, observe r_t, s_{t+1}
  - Armazenar a transição (s_t, a_t, r_t, s_{t+1}) no minibatch B_s
  - Extrair de B_t uma amostra aleatória de contendo N transições (s_i, a_i, r_i, s_{i+1})
  - Defina: $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
  - Atualizar o crítico minimizando $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q)^2)$
  - Atualizar a política do ator utilizando o gradiente da amostra de política: $\nabla_{\theta^\mu} J \approx \mathbb{E}_{s_t \tilde{\rho}^\beta}[\nabla_a Q(s, a)|_{s=s_t, a=\mu(s_t)} \cdot \nabla_{\theta^\mu}\mu(s)|_{s=s_t}]$
  - Atualizar as redes "targets":
  - $\theta^{Q'} \leftarrow \theta^Q + (1-\tau)\theta^{Q'}$
  - $\theta^{\mu'} \leftarrow \theta^\mu + (1-\tau)\theta^{\mu'}$
- fim

fim

O interesse do algoritmo de DDPG para esse trabalho reside no fato de que ele pode ser utilizado para construir uma política de decisão contínua, a qual pode ser utilizada para a construção de uma lei de controle de propulsão e leme.

3.5 Transferência de Aprendizado

A Transferência de Aprendizado (TA), ou Transfer Learning (em inglês), é uma técnica de Aprendizado de Máquina onde um modelo já treinado em uma tarefa é reaproveitado para uma outra tarefa relacionada(20). A técnica é muito útil em casos onde o treinamento dos modelos do zero é longo e demanda muitos recursos, como por exemplo no treinamento de Redes Neurais Convolucionais para reconhecimento de imagens, pois muitas vezes é possível reduzir drasticamente o número de iterações da fase de treino simplesmente ao

utilizar-se o modelo já treinado para aprender a nova tarefa.

Na Aprendizagem por Reforço, a TA também se preocupa com acelerar o processo de aprendizagem, uma vez que os agentes de APR podem demorar muitos episódios realizando exploração aleatória antes de conseguir desenvolver uma política razoável. A aplicação da TA torna-se então interessante para problemas onde os episódios são demorados (simulações complexas e custosas), acelerando-se assim consideravelmente a aprendizagem da política. Duas das principais formas de transferir o aprendizado em APR são mostradas a seguir.

3.5.1 Métodos de Ponto Inicial

Os Métodos de Ponto Inicial (MPI) utilizam um modelo já aprendido como solução inicial do aprendizado da nova tarefa. Tal solução é relativamente simples visto que o modelo se atualiza para a nova tarefa através da experiência (treino) por meio da própria APR. Comparado à configuração inicial aleatória ou em zero que é geralmente utilizada pelos algoritmos de APR, tais métodos podem iniciar a aprendizagem em um ponto muito mais próximo de uma boa solução para o novo problema.

Existem algumas maneiras de aproveitar o modelo aprendido como solução inicial, porém em geral o algoritmo de APR utilizado é o mesmo em ambos os problemas (da solução encontrada e o que se quer resolver). Algumas abordagens podem requerer um mapeamento dos estados e ações entre as tarefas, quando há diferenças entre os ambientes analisados.

3.5.2 Métodos de Imitação

Os Métodos de Imitação induzem escolhas de políticas aprendidas durante o aprendizado de outras tarefas. Apesar de não causar mudanças diretas na solução encontrada para a tarefa a aprender como os MPI, a Imitação afeta a aprendizagem produzindo diferentes atualizações na política ou função de valor. As decisões por Imitação podem levar o agente mais rapidamente a áreas mais promissoras do Ambiente se comparado à exploração aleatória geralmente realizada pelos algoritmos de APR. Uma opção de Método de Imitação, por exemplo, substitui as ações aleatórias de uma política ε-greedy pela escolha da política aproveitada.

4 MODELO DINÂMICO E SIMULAÇÃO

Neste capítulo descreve-se brevemente a dinâmica do navio considerado no estudo e o Simulador TPN, para o qual propõe-se o controle.

4.1 Dinâmica do navio

Nesta seção apresenta-se um modelo dinâmico simplificado que descreve o movimento de navegação de uma embarcação. A análise desse modelo é essencial para a compreensão dos resultados obtidos pelo simulador numérico, serve de base teórica para validação da coerência dos resultados obtidos pela APR, e é relevante para a dedução de parâmetros de design do algoritmo.

O modelo de navio apresentado possui 3 graus de liberdade, deve navegar no plano horizontal, e tem como hipótese principal a de que o navio possui baixa velocidade de até 3m/s (hipótese coerente com a navegação em águas restritas). Um modelo mais detalhado de navegação pode ser encontrado em (5) e em (21).

Seja um sistema de coordenadas fixo na terra $OXYZ$ e um sistema de coordenadas fixa no navio $oxyz$, com a origem o fixa no ponto central da seção média da quilha da embarcação. O centro de gravidade G está à distância x_G à frente do ponto o, ox é o eixo longitudinal do navio direcionado para a proa, e oy é o eixo transversal, apontando para o bombordo. O rumo da embarcação ψ define o ângulo entre os eixos de proa e OX. A figura 9 apresenta um diagrama simplificado do navio e seus sistemas de coordenadas.

As 3 equações diferenciais que definem a dinâmica desse modelo são:

$$\begin{aligned}(M + M_{11})\dot{u} - (M + M_{22})vr - (Mx_g + M_{26})r^2 &= X_{ext},\\ (M + M_{22})\dot{v} + (Mx_g + M_{26})\dot{r} + (M + M_{11})ur &= Y_{ext},\\ (I_z + M_{66})\dot{r} + (Mx_g + M_{26})(\dot{v} + ur) + (M_{11} - M_{22})uv &= N_{ext},\end{aligned} \tag{4.1}$$

onde M é a massa do navio, I_z é o momento de inércia do navio, u e v são as velocidades

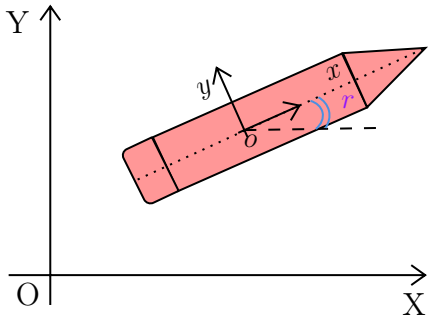

Figura 8: Diagrama do navio e seus sistemas de coordenadas

longitudinal e transversal, respectivamente, e r é a velocidade angular da guinada. Os termos M_{11} e M_{22} são as massas adicionadas ao navio nas direções ox e oy, M_{66} é o momento de inércia adicional do navio e M_{26} é a inércia acoplada adicional. N_{ext} é o momento de Munks. O subscrito ext representa as cargas externas que podem ser expressas em termos de diferentes fatores.

X_{ext} é decomposto como sendo:

$$X_{ext} = X_h + X_w + X_{wv} + X_p + X_{tug} + X_M, \tag{4.2}$$

onde X_h representa as forças hidrodinâmicas de não potencial, incluindo as forças de manobra e corrente, X_w, X_{wv} representam as forças de vento e de onda, respectivamente X_p representa os impulsores, forças de hélice e leme, finalmente, X_{tug} e X_M representam respectivamente a ação externa dos rebocadores e as forças devido a linhas de ancoragem. Analogamente, podemos decompor Y_{ext} e N_{ext}, sem perda de generalidade.

No escopo desse trabalho o navio foi simulado em condições de canal, ou seja na ausência de vento, ondas e corrente, e com forças de navegação e controle definidas apenas pelas ações de hélice e leme (sem rebocadores e amarras), simplificando as forças externas para as forças de impulsores e forças hidrodinâmicas de manobra e resistência (velocidade de corrente nula):

$$\begin{aligned} X_{ext} &= X_h + X_p, \\ Y_{ext} &= Y_h + Y_p, \\ N_{ext} &= N_h + N_p. \end{aligned} \tag{4.3}$$

As equações físicas que representam os esforços hidrodinâmicos de X_h, Y_h, eN_h fogem do escopo deste trabalho, mas podem ser encontradas nos trabalhos em detalhes em (21) e (22).

A força dinâmica do leme F_L (a qual as forças externas de índice p são funções dependentes), responsável pelo controle de guinada do navio é dada por:

$$F_L(\phi) = 0.5\rho A_r C_L(\phi) V_r^2, \tag{4.4}$$

onde ρ é a densidade da água, A_r é a área do leme, C_L é um coeficiente adimensional, ϕ o ângulo efetivo do leme e V_r a velocidade relativa de escoamento da água sobre no leme (Figura 9). Como observado, as forças do leme dependem da velocidade de escoamento, que está diretamente relacionada à rotação da hélice. Dessa forma, é intuitivo notar que a embarcação possui um potencial de controle de leme proporcional à rotação da hélice. Ou seja, as forças do leme são reduzidas quando o motor está inoperante.

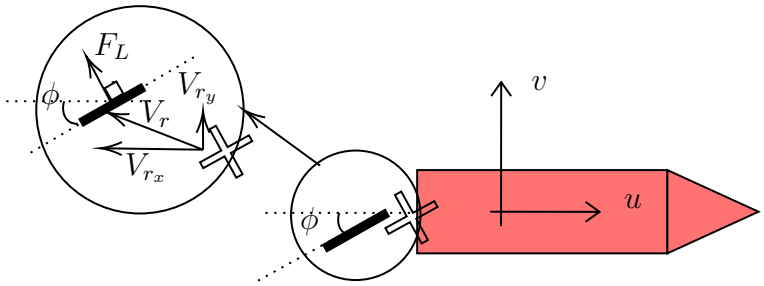

Figura 9: Diagrama de forças de leme

Já a força de propulsão (empuxo) de um propulsor azimutal fixo é definida como sendo:

$$T_p(J_A) = \rho n_p^2 D_p^4 K_T(J_A), \tag{4.5}$$

onde ρ é a densidade da água, n_p é a velocidade de rotação da hélice do propulsor, D_p é o diâmetro da hélice e K_T é a constante adimensional de torque do propulsor, que depende do coeficiente de avanço J_A e normalmente pode ser obtido através do gráfico K_T versus J_A fornecido pelo fabricante do propulsor.

O detalhamento de como as funções externas de índice p são funções das forças de leme e propulsão pode ser encontrado em (22).

4.2 Simulador TPN

O Centro de Simulação de Manobras TPN-USP é o maior Centro Brasileiro para simulação de Manobras de Navios. Este centro é composto por seis simuladores, três deles classificados como de missão completa (sistema imersivo com projeção virtual em

270º). A figura 10 ilustra uma sala do centro de manobras do TPN.

Figura 10: TPN: Sala de manobras de navios

A simulação pode ser executada em modo de tempo real em uma ou várias cabines simultaneamente (simulação de um ou vários pilotos). O simulador é usado para avaliação de novos portos, operações, análise de risco, treinamento de pilotos e capitães. O mesmo software de simulação também pode ser executado no modo de tempo rápido (fast-time) para propósito de analisar a trajetória percorrida por embarcações pilotadas por controladores automáticos, situações nas quais não existe a necessidade de simulação em tempo real.

O simulador pode ser resumido como um sistema de integração Runge-Kutta de 4ª ordem, que integra um conjunto de 6 equações diferenciais que regem a dinâmica do navio em seus 6 graus de liberdade (X, Y, Z, r_x, r_y, r_z).

O simulador é configurável através de um arquivo externo, o qual define as características do navio a ser pilotado tais como as matrizes de massa, matrizes de massa adicional, dimensões espaciais, localização e tipo de atuadores, etc. O arquivo externo define também as condições ambientais externas de navegação, tais como a presença e características dos ventos, correntes, ondas, etc. Para um maior detalhamento da influência das forças externas referir-se a (22) e para maior detalhamento das equações em 6 graus de liberdade referir-se a (21).

Para este trabalho o navio considerado é conhecido como ANGRA, e cujas caracte-

rísticas são detalhadas no apêndice A.1. O Navio apresenta apenas um leme azimutal como superfície de controle, e um propulsor do tipo azimutal fixo de rotação variável. As características do leme e do propulsor são descritas também nas seções A.4 e A.3, respectivamente.

O simulador fast-time recebe como entrada os comandos de leme e de propulsão, realiza a integração numérica nos 6 graus de liberdade e fornece como saída, a posição e vetor de velocidades do do navio em relação ao ponto de origem fixo. Para este trabalho, entretanto, como definido na seção 2.2.1, apenas a posição absoluta do navio (X_{abs}, Y_{abs}), a velocidade absoluta (V_x, V_y), o ângulo de aproamento (θ), e a velocidade angular $(\dot{\theta})$ são extraídos do simulador.

Os parâmetros de entrada tem a forma vetorial do tipo $A_V = [A_l, A_p]$, onde A_l é o comando adimensional de leme e A_p o comando adimensional de propulsão, de tal forma que $A_l \in [-1, 1]$ e $A_p \in [0, 1]$. Tais parâmetros tem uma relação proporcional direta com o ângulo de leme e a propulsão, de tal forma que:

$$\begin{aligned} A_p = 1 &\rightarrow T_p = T_p^{\max}, \\ A_p = 0 &\rightarrow T_p = T_p^{\min}, \\ A_l = 1 &\rightarrow \phi_L = \phi_L^{\max}, \\ A_l = 0 &\rightarrow \phi_L = 0\ , \\ A_l = -1 &\rightarrow \phi_L = \phi_L^{\min}. \end{aligned} \tag{4.6}$$

Os valores de $T_p^{\max}$, $T_p^{\min}$, $\phi_L^{\max}$ e $\phi_L^{\min}$, são definidos nos apêndices A.3 e A.4.

5 TRABALHOS RELACIONADOS

Nesta seção, os principais trabalhos que foram utilizados como base para elaboração da solução proposta são apresentados, e os tópicos de maior relevância para elaboração da solução são citados e descritos brevemente.

5.1 Ambiente de Simulação

Nos trabalhos de Ahmed e Hasegawa(4), o objetivo é controlar as ações de voo de um helicóptero. Neste contexto, o ambiente é o espaço de estados do helicóptero durante o voo. Para modelizar o ambiente, os autores utilizam uma etapa de identificação do modelo através de um voo manual sensoreado. Em seguida utiliza-se uma rede neural para parametrizar as transições de estados a partir dos dados coletados. Finalmente, um agente é treinado utilizando o ambiente virtual (modelo parametrizado) e posteriormente testado no ambiente físico real (voo).

No trabalho de Abbeel et al.(23), o objetivo foi controlar um helicóptero em voo invertido. A metodologia para modelização do ambiente foi parecida - primeiramente, modelizou-se o ambiente através de dados captados de um voo. Nesse caso, porém, utilizou-se um modelo paramétrico físico na etapa de identificação, e os parâmetros do modelo foram identificados a partir de uma regressão linear dos dados de voo. O ambiente (modelo) foi então utilizado para o treinar o agente, que foi posteriormente validado em voo.

Em Lau(24), busca-se controlar um simulador de corrida de carros a partir de uma rede neural usando APR profunda e DDPG. Nesse caso o ambiente não necessita ser identificado pois o próprio simulador é virtual e suficientemente rápido para realizar a aprendizagem do algoritmo.

Finalmente, em Brockman et al.(25) propõe-se uma forma genérica para modelizar ambientes virtuais utilizados em técnicas de aprendizado por reforço utilizando a bibli-

oteca OpenAI gym. O objetivo é a padronização de conceitos relacionados à aplicação de algoritmos de aprendizagem. Busca-se também o compartilhamento dos ambientes em uma plataforma global para que diversos membros da comunidade OpenAI gym possam facilmente aplicar e comparar técnicas de aprendizado, fomentando assim o desenvolvimento de algoritmos eficientes para um determinado problema.

5.2 Design da recompensa

A definição da recompensa é um aspecto fundamental na APR, como relata (24), diferentes funções de recompensa podem significar uma diferença substancial no número de etapas necessárias para a convergência do algoritmo, seja ele de APR Profunda ou Q-learning aproximado. Em (23) e em (7), propõe-se a aplicação de uma função de custo semelhante à utilizada nos problemas de controle clássico utilizando LQR. Tal utilização apresentou bons resultados em ambos os trabalhos. Para fins de convergência, entretanto, como cita (24), recompensas com valores normalizados entre -1 e 1 tendem uma convergência mais rápida para redes neurais em DQN E DDPG.

5.3 Redes Neurais e Hiper-parâmetros

Como demonstrado em (24), uma rede neural 2 camadas, utilizando o método de DDPG, com a estrutura ator-crítico-target, e foi capaz de controlar a condução de um veículo após 200 mil iterações. Para isso utilizou-se o método estocástico reversível Ornstein-Uhlenbeck. Os comandos abordados eram do tipo contínuo em aceleração, freio e direção. Observou-se como resultado final que o veículo seguia uma trajetória-guia com pequenas oscilações de direção.

Já em Plappert(26) apresenta-se o controle de diversos ambientes de controle continuo e discreto estruturados utilizando os moldes definidos por (25). Nesse contexto, um pêndulo invertido foi controlado utilizando o método DQN e uma rede neural de 3 camadas, em associação à uma discretização das ações para a formulação e usando replay de experiência para garantir a convergência . A partir de (26) e de (24) foram coletados os hiper-parâmetros para a definição da rede neural. Eles são detalhados na tabela a seguir.

Tabela 1: Hiper-parâmetros para a definição da RNA

	CartPole	Pêndulo invertido	Simulador Carro
Espaço de Estados	Número: 4 Tipo: Contínuo	Número: 3 Tipo: Contínuo	Número: 8 Tipo: Contínuo
Espaço de Ações	Número: 2 Tipo: Discreto	Número: 1 Tipo: Contínuo	Número:3 Tipo: Contínuo
Tipo de RNA	DQN com Replay de Experiência e RNA Alvo	DDPG: Ator-Crítico, Replay de Experiência e RNA Alvo	DDPG: Ator-Crítico, Replay de Experiência e RNA Alvo
Arquitetura da RNA	Sequencial, Entrada: Estados Camada 1: 16 Camada 2: 16 Camada 3: 16 Saída: Ação	Ator: Sequencial, Entrada: Estados Camada 1: 16 Camada 2: 16 Camada3: 16 Saída: Direção Crítico: Sequencial, Entrada: Ações, Estados Camada 1: 32 Camada 2: 32 Camada3: 32 Saída: Booleana (dimensão = Ações)	Ator:Sequencial, Entrada: Estados Camada 1: 600 Camada 2: 300 Saída: Direção Crítico:Sequencial, Entrada: Ações, Estados Camada 1: 600 Camada 2: 300 Saída: Booleana (dimensão = Ações)

Além disso, a partir de outros ambientes que podem ser encontrados em (26) observa-se que para modelos em que o número de estados é elevado o número de neurônios por camada é mais elevado. O número de camadas está associado ao número de decomposições espaciais da entrada de uma rede, enquanto o número de neurônios está associado ao número de permutações espaciais e ambos são configurados de acordo com a complexidade da tarefa a ser resolvida.

6 SOLUÇÃO PROPOSTA

Este capítulo detalha a solução proposta, tanto no lado da simulação do sistema dinâmico, quanto na parte da estruturação do problema de Aprendizagem por Reforço

6.1 Simulação para aprendizagem

Nesta seção os mecanismos utilizados de simulação para a elaboração da solução são discutidos e os principais aspectos da solução são apresentados.

6.1.1 Simulador TPN: Uso e limitações

Como descrito no capitulo 4 o simulador TPN é um software para simulação de navegação que já foi extensivamente validado, e que é usado como suporte para simulação de manobras e para o aprendizado de novos pilotos. Para tanto o software apresenta uma dinâmica complexa, que considera não só a dinâmica do navio em 6 graus de liberdade como também as equações fluidodinâmicas na iteração navio-ambiente (vento, correntes) e na iteração do propulsor e leme.

Se por um lado essa complexidade é essencial para a reprodução verossímil do comportamento do navio, por outro, ela impõe dificuldades para a aprendizagem por reforço.

Além disso o simulador do TPN se comunica com o ambiente de desenvolvimento em python utilizando a interface simulador-python desenvolvida pela equipe do TPN e que utiliza um protocolo local através de uma comunicação via soquetes. Tal arquitetura de comunicação também impõe dificuldades para o aprendizado.

As dificuldades apresentadas pela complexidade do modelo e pela arquitetura de comunicação são apresentadas a seguir.

D1 - Tempo de simulação: A primeira delas está relacionada ao tempo de simulação, utilizando a versão fast-time do simulador, o número de iterações por segundo alcan-

çado nas configurações de teste apresentadas em 6.1.3 , é em torno de 5 (iterações/s). Ou seja para um processo de aprendizagem com 300 mil iterações como o usado em (24), o tempo de simulação seria em torno de 138 horas. Diminuindo a tolerância de integração e desconsiderando os efeitos externos como o ventos e correntes, o tempo de simulação gira em torno de 30 iterações/s (parâmetro de treinamento TPN apresentado em 6.1.3), ou seja 2h40 de treino para 300 mil iterações. Parte da lentidão imposta pelo simulador é devida à estrutura de comunicação.

D2 - Falhas de comunicação: Um número elevado de iterações pode favorecer erros de comunicação. Tal erro pode prejudicar a simulação, uma vez que neste caso a resposta do ambiente de simulação (estado do navio) é divergente do real valor do estado do navio.

D3 - Falha de reinicialização: Foi identificada ao longo do projeto um erro na velocidade do navio quando o simulador era exposto a um número elevado de reinicializações da propulsão, leme e posição (realiza-se uma reinicialização a cada início de episódio). O problema deve-se ao fato do simulador não ser projetado para muitas reinicializações seguidas em uma mesma instância. Este problema foi contornado iniciando-se uma nova instância de simulação a cada 10 episódios, o que induziu um aumento no tempo de simulação.

Tais dificuldades fizeram com que o requisito de projeto primário RP2 fosse reformulado afim de aumentar a velocidade de simulação e construir uma solução que convergisse para o cumprimento da missão proposta.

Nesse sentido, uma solução alternativa que foi proposta e desenvolvida foi a realização de um simulador simples de navegação (SSN) em python, o qual foi utilizado para ensinar o agente de APR as dinâmicas básicas de navegação. Em seguida para que o agente fosse capaz de cumprir a missão proposta no simulador TPN, foi proposto o método de Transferência de Aprendizado. Os detalhes sobre essa etapa são detalhada na seção seguinte.

6.1.2 Simulador simples: Uso e desenvolvimento

O Simulador Simples de Navegação (SSN) tem como finalidade reproduzir a dinâmica de navegação de uma embarcação afim de ser utilizada como suporte para o método de aprendizagem. Sua utilização foi inspirada nos trabalhos relacionados como os citados em 5.1. O uso de tal simulador representa a reformulação do RP2 e é descrito a seguir:

- O sistema de APR deve basear-se na transição de estados fornecida pelo SSN afim de aprender uma dinâmica de navegação simples.
- Utilizando o agente aprendido no SSN, deve-se proceder para uma etapa de transferência de aprendizagem (TA).
- Após a TA o agente deve apresentar uma performance compatível com as descritas na seção 2.2.1.

O diagrama a seguir ilustra a estrutura de aprendizado desenvolvida a partir do SSN.

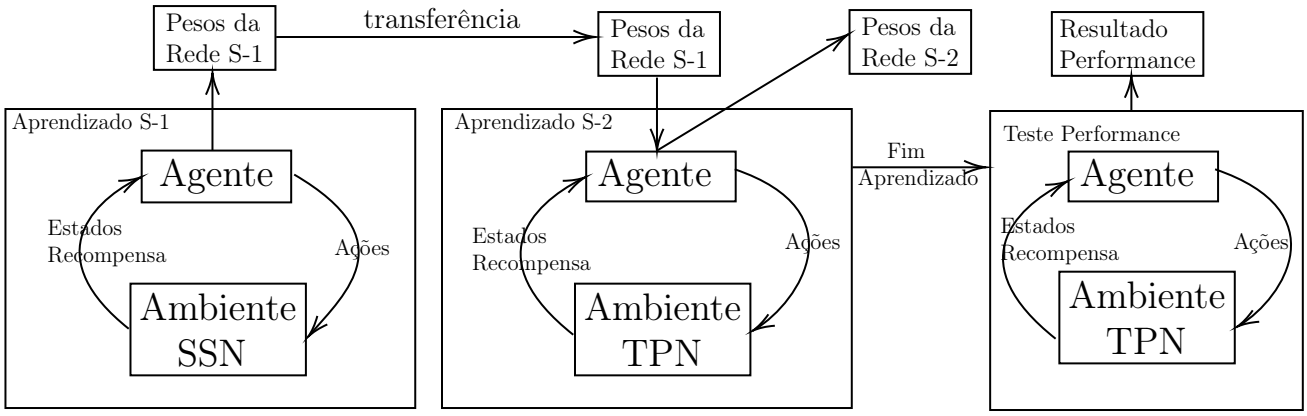

Figura 11: TA usando SSN e simulador TPN

O simulador desenvolvido é um simulador em 3 graus de liberdade, utiliza as equações apresentadas na seção 4, e implementa um método de integração Runge-Kutta de ordem 5. O tempo de simulação utilizando o simulador simples gira em torno de 200 iterações/s e não há erros de comunicação, uma vez que o SSN foi desenvolvido em python.

Além disso o SSN utiliza a embarcação com as mesmas características do simulador TPN, e o comportamento dinâmico de simulação foi ajustado para ser similar ao do simulador TPN. Tal ajuste foi realizado através uma identificação não linear de parâmetros utilizando-se como ferramenta auxiliar o software MATLAB.

O SSN está atualmente disponível online sob o nome de ShipAI para contribuição com a comunidade OpenAI Gym, assim como para contribuição científica.

6.1.3 Parâmetros de simulação

Nesta seção os parâmetros de simulação que influenciam o design da solução são apresentados assim como suas justificativas.

P1 Tempo de integração (t_{int}): O tempo de integração é o tempo que o método de integração numérico (Runge-Kutta) utiliza como limite superior para integrar as equações dinâmicas que compõem o sistema físico simulado. A cada iteração do simulador, o sistema dinâmico avança t_{int}.

P2 Passo de integração: O passo de integração é o passo de tempo avançado a cada subetapa de integração numérica, este é função do tempo de integração e da tolerância de integração.

P3 Tolerância global ($atol$) e relativa ($rtol$): Tolerâncias relativas e absolutas são utilizadas explicitamente no SSN. O solver do SSN utiliza essas medidas para manter as estimativas de erros locais menores que $atol + rtol \cdot \|(s_i)\|$, onde s_i é o estado estado integrado. No simulador TPN a tolerância é controlado pelo parâmetro TOL_{CONV}.

P4 Tempo de ação (t_{acao}): O tempo de ação define o intervalo entre o envio de duas ações consecutivas ao simulador (uma iteração do algoritmo de aprendizagem). Geralmente é definido como sendo $t_{acao} = K_{int} \cdot t_{int}$. Ou seja uma iteração terá K_{int} etapas de integração numérica.

Primeiramente definiu-se fixo $t_{acao} = 10s$, como requisitado em 2.2.1 tanto para o ambiente utilizando o simulador TPN quanto para o SSN. Esse valor foi definido com base em dados empíricos de navegação. Em seguida definiu-se as configurações de treinamento e teste para o simulador TPN e SSN como se segue:

- Configuração de treinamento e teste SSN:

$$\begin{aligned} t_{int} = t_{acao} &= 10 \text{ s}, \\ atol &= 1\text{e}{-4}, \\ rtol &= 1\text{e}{-6}. \end{aligned} \tag{6.1}$$

- Configuração de treinamento (TA) TPN:

$$\begin{aligned} t_{int} = t_{acao} &= 10 \text{ s}, \\ TOL_{CONV} &= 1\text{e}{-3}. \end{aligned} \tag{6.2}$$

- Configuração de teste TPN:

$$\begin{aligned} t_{acao} &= 10 \text{ s}, \\ t_{int} &= 0.5, \\ TOL_{CONV} &= 1\text{e}{-3}. \end{aligned} \tag{6.3}$$

6.2 Estruturação do problema de APR

Nessa seção é descrita a estrutura dada ao problema de APR, desde a definição de parâmetros até os algoritmos utilizados. O problema foi modelado afim de atingir a missão descrita em 2.1, e respeitando os requisitos descritos em 2.2.1.

6.2.1 Algoritmos

Dadas as soluções propostas na literatura, escolheu-se para a análise a comparação do método variante de Q-learning aproximado - DQN[1] e o DDPG[2].

6.2.2 Estados utilizados

O problema da manobra de embarcações possui várias variáveis de estado, seja em posição, rotação, velocidade, etc. Para simplificar o problema e permitir sua aplicação em APR, foram selecionadas aquelas consideradas essenciais na manutenção de uma trajetória dada durante a navegação. O estado escolhido foi

$$s = (d, \theta, v_x, v_y, \dot{\theta}), \tag{6.4}$$

onde:

- d: Distância do centro de massa do navio à linha-guia
- θ: Ângulo entre o eixo longitudinal do navio e a linha-guia
- v_x: Velocidade horizontal do navio no seu centro de massa (na direção da linha-guia)
- v_y: Velocidade vertical do navio no seu centro de massa (perpendicular à linha-guia)
- $\dot{\theta}$: Velocidade angular do navio

[1] cf. 3.4.4.3
[2] cf. 3.4.5.1

Essas variáveis de estado estão ilustradas na figura 12.

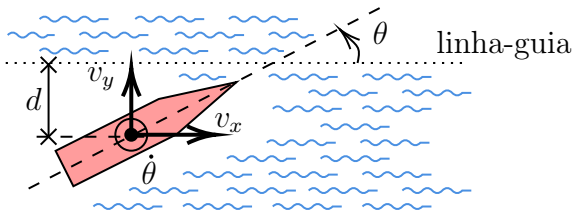

Figura 12: Esquema do navio com estados utilizados na APR

6.2.2.1 Limites admitidos

Os limites admitidos para os estados no algoritmo de APR são descritos a seguir. Caso o agente ultrapasse tais limites, o episódio é encerrado.

$$\begin{aligned} d &\in [0, 150] \text{ m}, \\ \theta &\in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right] \text{ rad}, \\ v_x &\in [0, 4] \text{ m/s}, \\ v_y &\in [-4, 4] \text{ m/s}, \\ \dot{\theta} &\in \left[-\frac{\pi}{9}, \frac{\pi}{9}\right] \text{ rad/s}. \end{aligned} \tag{6.5}$$

O limite máximo de d decorre do ambiente utilizado na análise, como descrito mais a frente neste capítulo, na seção 6.2.5.2.

6.2.2.2 Inicialização dos estados

A inicialização do estado de cada episódio é feita aleatoriamente, porém com limites menores que os admitidos, para garantir a navegabilidade e manobrabilidade do navio no estado inicial, dando estabilidade ao algoritmo de aprendizagem. Buscou-se utilizar estados iniciais compatíveis com aqueles que ocorrem durante a entrada de uma embarcação em um canal. Tais limites são mostrados abaixo:

$$\begin{aligned} d_0 &\in [0, 30] \text{ m}, \\ \theta_0 &\in \left[-\frac{\pi}{15}, \frac{\pi}{15}\right] \text{ rad}, \\ v_{x0} &\in [1, 2] \text{ m/s}, \\ v_{y0} &\in [-0.4, 0.4] \text{ m/s}, \\ \dot{\theta}_0 &= 0 \text{ rad/s}. \end{aligned} \tag{6.6}$$

6.2.3 Ações de comando

As ações de comando foram desenvolvidas para controlar o ângulo de leme e a propulsão do navio. Como descrito na seção 4.2, as ações de comando tem a forma $A_V = [A_l, A_p]$, onde A_l é o comando adimensional de leme e A_p o comando adimensional de propulsão, de tal forma que $A_l \in [-1, 1]$ e $A_p \in [0, 1]$.

Para o DQN, as ações foram discretizadas afim de se adequar ao modelo, para tanto, optou-se por dividir o espaço de ações de leme em 21 ações discretas, e o espaço de ações de propulsão em 3. Tal divisão foi feita com base em técnicas empíricas de navegação de práticos. Sendo assim tem-se que:

$$\begin{aligned} A_V^{\text{DQN}} &= [A_l^{\text{DQN}}, A_p^{\text{DQN}}], \\ \text{SSN}: A_l^{\text{DQN}} &\in \{-1, -0.9, -0.8, ..., 0.9, 1\}, \\ \text{TPN}: A_l^{\text{DQN}} &\in \{-1, -0.9, -0.8, ..., 0.9, 1\}/3, \\ A_p^{\text{DQN}} &\in \{0, 0.1, 0.2\}. \end{aligned} \tag{6.7}$$

Optou-se por reduzir os ângulos de leme no DQN pois notou-se um melhor comportamento desta maneira no simulador TPN. Para o DDPG, as ações são do tipo contínuo, porém limitou-se as ações de leme e propulsão como se segue:

$$\begin{aligned} A_V^{\text{DDPG}} &= [A_l^{\text{DDPG}}, A_p^{\text{DDPG}}], \\ A_l^{\text{DDPG}} &\in [-1/3, 1/3], \\ \text{SSN}: A_p^{\text{DDPG}} &\in [0, 0.2], \\ \text{TPN}: A_p^{\text{DDPG}} &\in [0, 0.24]. \end{aligned} \tag{6.8}$$

A heurística da limitação de leme deu-se pois a partir de simulações preliminares identificou-se que com o comando de leme entre -10º e 10º é possível controlar a direção da embarcação, começando com os estados citados em 6.2.2.2. Tal procedimento pode ser entendido como uma adaptação simples entre os simuladores SSN e TPN.

Para o controle de leme optou-se por controlar a propulsão com empuxos positivos, e o limite definido em 0.2 é suficiente para acelerar o navio até a velocidade de setpoint no SSN, para o simulador TPN aumentou-se esse limite em 20%, após testes empíricos que mostraram uma melhor performance com essa configuração.

6.2.4 Definição das recompensas

As recompensas, apesar do nome, comumente são valores negativos dados ao agente, pois devem punir estados/ações indesejáveis proporcionalmente ao quão indesejáveis eles são. Em alguns casos, no entanto, pode-se dar recompensas positivas quando o agente atinge um objetivo, por exemplo. No caso de navegação considerado, decidiu-se utilizar uma recompensa negativa para os desvios do navio em relação ao setpoint de estados desejado, com uma certa tolerância. Optou-se por dividir os desvios pelos desvios máximos tolerados no treinamento, para facilitar a ponderação das importâncias de cada desvio. A recompensa utilizada segue a fórmula abaixo:

$$r(s) = k_{tol} - k_d \times \frac{|d|}{d_{\max}} - k_\theta \times \frac{|\theta|}{\theta_{\max}} - k_{v_x} \times \frac{|v_x - v_{sp}|}{v_{x_{\max}}} - k_{v_y} \times \frac{|v_y|}{v_{y_{\max}}} - k_{\dot{\theta}} \times \frac{|\dot{\theta}|}{\dot{\theta}_{\max}} \ , \quad (6.9)$$

onde:

- k_{tol}: Constante de tolerância da recompensa, define a margem de erro que o navio pode cometer ainda possuindo uma recompensa positiva;
- $\{k_d, k_\theta, k_{v_x}, k_{v_y}, k_{\dot{\theta}}\}$: Constantes de proporcionalidade das recompensas, ajustam a importância dada a cada um dos desvios;
- v_{sp}: Setpoint de velocidade do navio;
- $\{d_{\max}, \theta_{\max}, v_{x_{\max}}, v_{y_{\max}}, \dot{\theta}_{\max}\}$: Limites máximos admitidos para os estados (vide equação 6.5).

Nos testes realizados, utilizou-se $k_{tol}, k_{v_x}, k_{v_y}, k_{\dot{\theta}} = 1$ e $k_d, k_\theta = 8$.

6.2.5 Parametrização

6.2.5.1 Hiper-parâmetros

Para a parametrização do modelo, buscou-se utilizar parâmetros coerentes com o que se encontra nos trabalhos relacionados[3] e com a complexidade do problema a ser resolvido. Os hiper-parâmetros utilizados em cada método são mostrados na tabela abaixo:

[3] cf. 5.3

Tabela 2: Hiper-parâmetros utilizados nos métodos aplicados ao problema do navio

Métodos	DQN	DDPG
Espaço de Estados	Dimensão: 5, Tipo: Contínuo	Dimensão: 5, Tipo: Contínuo
Espaço de Ações	Dimensão: 2, Tipo: Discreto (3 intensidades de propulsão e 20 ângulos de leme)	Dimensão: 2, Tipo: Contínuo
Tipo de RNA	DQN com Replay de Experiência e RNA Alvo	DDPG: Ator-Crítico, Replay de Experiência, RNA Alvo
Arquitetura da(s) RNA(s)	Sequencial, Entrada: Estados, 4 camadas: [256,128,64,33] neurônios, Ativação ReLU nas camadas ocultas, linear na de saída, Saída: Valores de ação, Otimização: Adam, taxa de aprendizagem 1e−3	Ator: Sequencial, Entrada: Estados, 3 camadas: [400,300,2] neurônios, Ativação ReLU nas camadas ocultas, Softsign na de saída, Saída: Ação, Otimização: Adam, taxa de aprendizagem 1e−4 Crítico: Sequencial, Entrada: Ações, Estados, 3 camadas: [400,300,1] neurônios, Ativação ReLU nas camadas ocultas, linear na de saída, Saída: Booleana (dimensão = Ações), Otimização: Adam, taxa de aprendizagem 1e−3
Treinamento	Iterações: 400000, $\gamma = 0.99$ Política: ε-greedy, decaimento linear de ε de 1 a 0,1 Memória do Replay de Experiência: 20000 transições Atualização Dura do Modelo Alvo: $C = 1000$	Iterações: 600000, $\gamma = 0.99$ Processo Aleatório: Ornstein Uhlenbeck ($\theta = 0.3$, $\mu = 0$, $\sigma = 0.3$), Memória do Replay de Experiência: 20000 transições Atualização Mole do Modelo Alvo: $C = 1e{-}2$
Transferência de Aprendizagem	MPI com 50000 iterações, Decaimento linear de ε de 0,1 a 0,01	MPI com 100000 iterações, Restante dos parâmetros iguais aos do treinamento

6.2.5.2 Ambiente

O ambiente utilizado para aprendizagem tem sua estrutura baseada nos modelos de ambientação OpenAI Gym. Tanto para o SSN como no caso do TPN utilizou-se a mesma interface gráfica e ambientação. Tal estruturação foi baseada nos trabalhos relacionados como citado em 5.1 .

O cenário utilizado nos experimentos realizados foi inspirado no canal de acesso do Porto de Suape, localizado em Recife, Pernambuco, no Nordeste brasileiro. O canal foi dimensionado como sendo retangular de 5000 metros de comprimento e 300 metros de largura. A linha-guia que o navio deve seguir é a linha-média longitudinal do canal. Temos então os pontos inicial e final da trajetória-guia definidos como:

$$(X_{inicial}, Y_{inicial}) = (0, 0);\ (X_{final}, Y_{final}) = (5000, 0)\ \text{m}; \tag{6.10}$$

e o espaço entre as duas retas que delimitam a largura do canal:

$$E_{canal} = 300\ \text{m}. \tag{6.11}$$

O setpoint de velocidade do navio é de 2 m/s.

7 RESULTADOS

Esta seção visa apresentar o resultado obtido no âmbito do treinamento, teste e controle final do navio utilizando os métodos DQN e DDPG.

7.1 Treinamento

Nesta seção avalia-se os fatores de aprendizagem e de TA nos processos de DQN e DDPG.

7.1.1 DQN

7.1.2 Recompensa acumulada treino SSN

Uma das maneiras de avaliar um algoritmo de aprendizagem é através da análise da recompensa acumulada durante o treinamento como mostrado na figura 13. É possível observar que após o episódio 2600 há um salto no valor da recompensa do navio, pode-se inferir que a partir de então o navio deixou de colidir frequentemente com as bordas do canal e passou a alcançar 5000 m de navegação sem colisão.

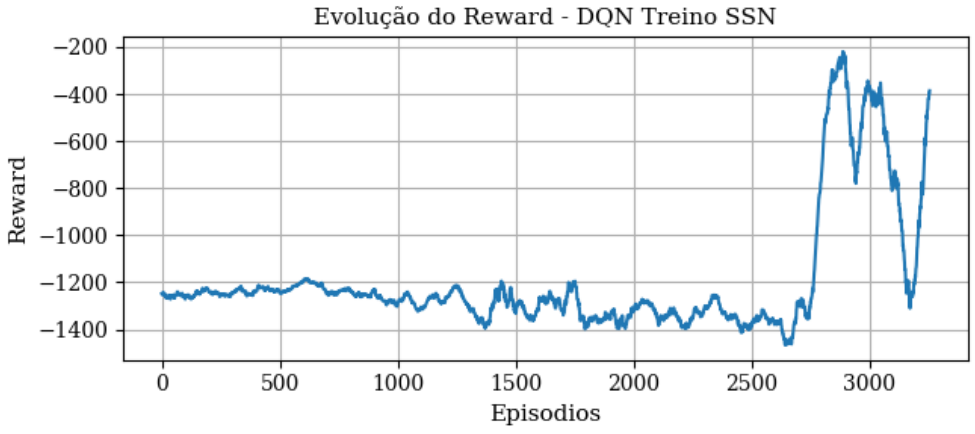

Figura 13: Recompensa acumulada treino no SSN - DQN

É notável também a presença de ruído na evolução da recompensa acumulada, isso pode ser explicado pelo caráter aleatório das inicializações de estados, o que, por sua vez, gera diferenças expressivas no desempenho geral do episódio. Ou seja, quando o navio inicia suas condições de navegação com baixo ângulo de aproamento e baixa distância da linha-guia é mais fácil para o agente controlar o navio do que quando o ambos são elevados.

Apesar de não ser visível um grande aumento na recompensa após o salto por volta dos 2600 episódios, o navio continua seu aprendizado a cada episódio, porém de forma menos visível pelo gráfico de recompensa acumulada.

7.1.3 Recompensa acumulada TA TPN

O gráfico de recompensa acumulada para o processo de TA é mostrado na figura 14, a primeira parte da TA consiste em uma etapa de warm-up (aquecimento), em que as ações do agente do SSN são registradas na memória dos batch de estados, a região até 50 episódios apresenta portanto alto valor de reward, logo em seguida a TA é inicializada com uma política com ε-greedy descendente, portanto, o gráfico apresenta uma região de baixa recompensa seguida por um aumento no final do episódio.

Durante os experimentos de TA não houve melhora expressiva no desempenho da navegação. Diversas grandezas para o número de iterações foram estudadas, porém não observou-se melhorias com o aumento do número de iterações, portanto chegou-se a conclusão de que 50 mil iterações de TA eram suficientes para uma navegação de desempenho análogo às demais grandezas.

Atribui-se esse resultado à limitação de controle da política DQN associada ao sistema de Hard Target Update, o qual pode apresentar oscilação na otimização de políticas para alguns modelos de aprendizagem.

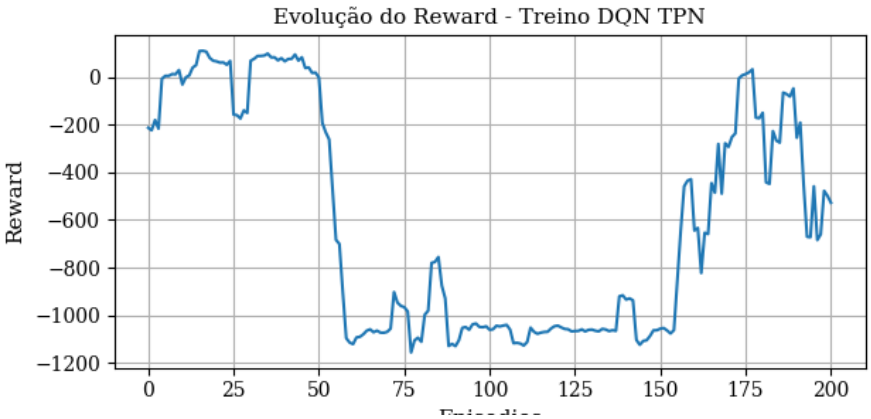

Figura 14: Recompensa acumulada treino no TPN - DQN

7.1.4 DDPG

7.1.5 Recompensa acumulada treino SSN

Observa-se que após o episódio 500 há um salto no valor da recompensa do navio, pode-se inferir que a partir de então o navio deixou de colidir frequentemente com as bordas do canal e passou a alcançar 5000 m de navegação sem colisão.

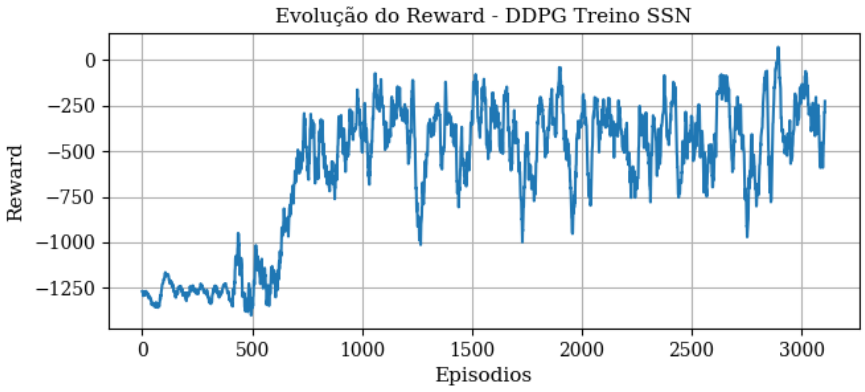

Figura 15: Recompensa acumulada treino no SSN - DDPG

Assim como no caso anterior, observa-se a presença de ruído na evolução da recompensa acumulada, isso pode ser explicado pelo caráter aleatório das inicializações de estados.

7.1.6 Recompensa acumulada TA TPN

A recompensa acumulada pode ser usada para analisar-se também a TA. Nesse caso porém, não se observa um grande aumento no acúmulo de recompensas, porém, como observado na seção 7.2.5 o desempenho da navegação é melhorado.

Figura 16: Recompensa acumulada TA no TPN - DDPG

7.2 Análise de Performance

Nessa seção uma análise da performance da política de navegação é realizada. Busca-se avaliar a qualidade da navegação em relação à convergência da distância (d) e velocidade horizontal (v_x), assim como analisa-se os demais estados observáveis, tais como o ângulo de ataque (θ), velocidade de rotação ($\dot{\theta}$) e velocidade de aproximação (v_y).

Para analisar a performance do agente durante a tarefa de controle da embarcação usando o DDPG e o DQN foram realizados testes de performance em três etapas:

1 Performance do agente em navegação no SSN com os pesos da rede neural treinada no SSN

2 Performance do agente em navegação no TPN com os pesos da rede neural treinada no SSN

3 Performance do agente em navegação no TPN com os pesos da rede neural após a TA no TPN

Com os resultados de 1 pode-se justificar o treinamento inicial no agente DDPG para posterior TA. Com os resultados de 2 pode-se avaliar a viabilidade do TA assim como a coerência do modelo SSN em relação ao TPN em relação à tarefa de aprendizado. Com

os resultados de 3 pode-se avaliar o resultado da TA, assim como o cumprimento dos requisitos de projeto e performance.

Para cada uma das três análises, utilizou-se dois cenários para avaliação de performance:

- Cenário 1:

  100 episódios com inicialização aleatória assim como descrito em 6.2.2.2. O objetivo é avaliar a sobrevivência da navegação, ou seja a ausência de colisão com o canal durante os episódios. Além disso pode-se avaliar de forma geral a convergência da distância ($d \to 0$) e a convergência de velocidade ($v_x \to 2$ m/s) .

- Cenário 2:

  10 episódios com inicialização definida, de tal modo que os estados iniciais sejam $s_i = [d_i, \theta_i, vx_i, vy_i, \dot{\theta}_i]$, onde:

$$\begin{aligned} d_i &= 30 \text{ m}, \\ \theta &= \frac{k}{10} \cdot \frac{\pi}{15} \text{ rad}, \ k = 0, 1, 2, ..., 9 \ , \\ va_i &= 1.5 \text{ m/s}, \\ v_{xi} = va_i \cdot cos(\theta) &\text{ e } v_{yi} = va_i \cdot sin(\theta), \\ \dot{\theta}_i &= 0. \end{aligned} \tag{7.1}$$

  O objetivo do cenário 2 é observar a evolução dos estados observáveis para cada uma das inicializações, assim como avaliar a performance das ações. Dessa forma é possível avaliar a convergência e o caráter da ação de leme e propulsão.

7.2.1 DQN

7.2.2 Performance no SSN

- Resultado cenário 1:

  Nenhuma colisão. Convergência oscilatória para a distância com $d < 20$ m, não convergência da velocidade, apesar de haver uma rampa ascendente para a velocidade.

- Resultado cenário 2:

  Nos 10 episódios houve a convergência da distância $d < 25$ m, com oscilação do navio e um erro estacionário de 20 m como observado na figura 17.

Além disso observa-se que não há convergência para a velocidade desejada, apesar da velocidade apresentar perfil ascendente.

O DQN não é capaz de controlar o ângulo de navegação para suavizar a direção do navio e não consegue otimizar a velocidade.

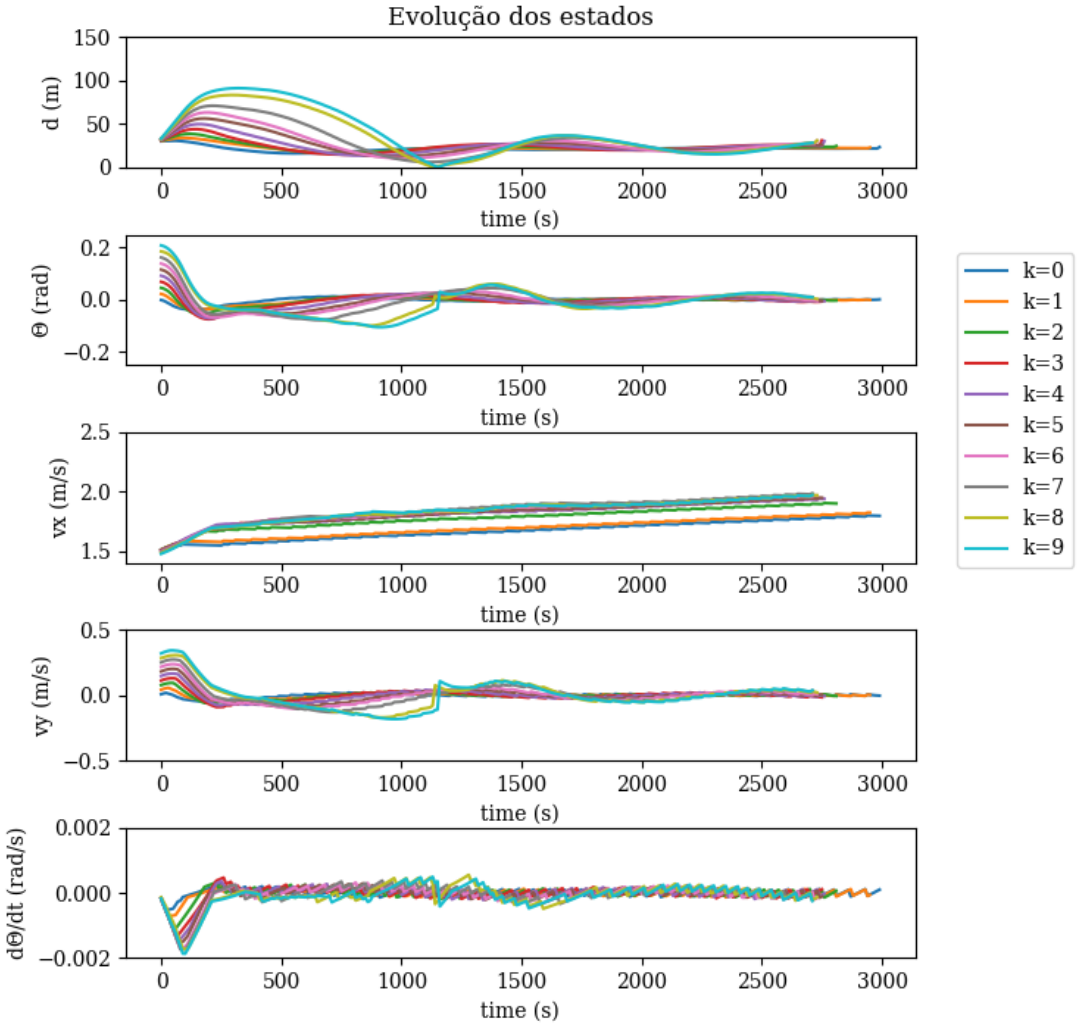

Figura 17: Evolução de estados observáveis DQN SSN

7.2.3 Performance no TPN antes da TA

- Resultado cenário 1:

  Nenhuma Colisão. Convergência oscilatória para a distância com $d < 18$ m, não convergência da velocidade.

- Resultado cenário 2:

Em 9 dos 10 episódios houve a convergência da distância, com oscilação inferior à 18 metros como observado na figura 18. O episódio mais crítico levou à colisão do navio, mostrando o ambiente SSN não é capaz de representar a dinâmica do TPN para o caso do agente DQN.

Além disso observa-se que não há convergência para a velocidade desejada, pelo contrário, houve uma queda para vy em torno de 1.2 m/s.

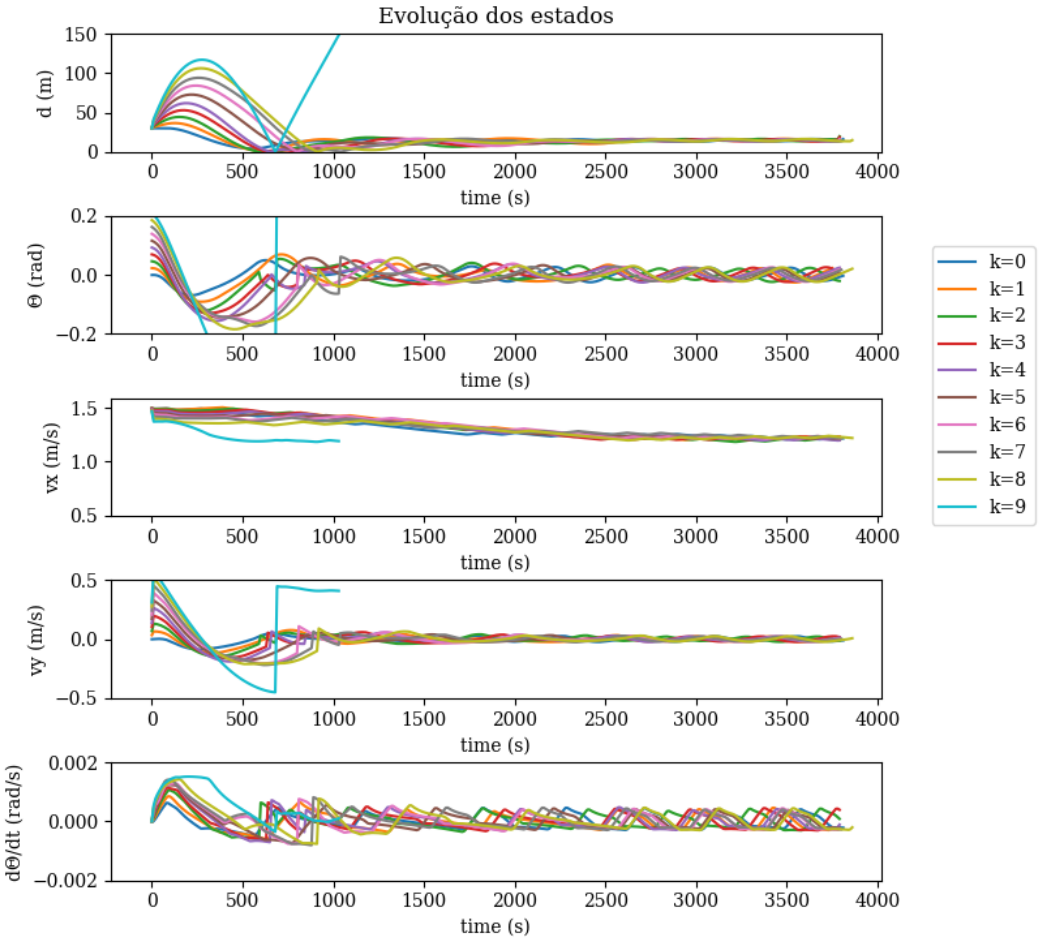

Figura 18: Evolução de estados observáveis DQN SSN

7.2.4 Performance no TPN após a TA

- Resultado cenário 1:

  Nenhuma Colisão. Convergência pouco oscilatória para a distância com $d < 15$ m, não convergência da velocidade.

- Resultado cenário 2:

Para os 10 episódios houve a convergência da distância, com oscilação inferior à 25 metros como observado na figura 19.

Além disso observa-se que não há convergência para a velocidade desejada.

Observa-se que o DQN apresenta limitações para o controle de velocidade e direção do navio.

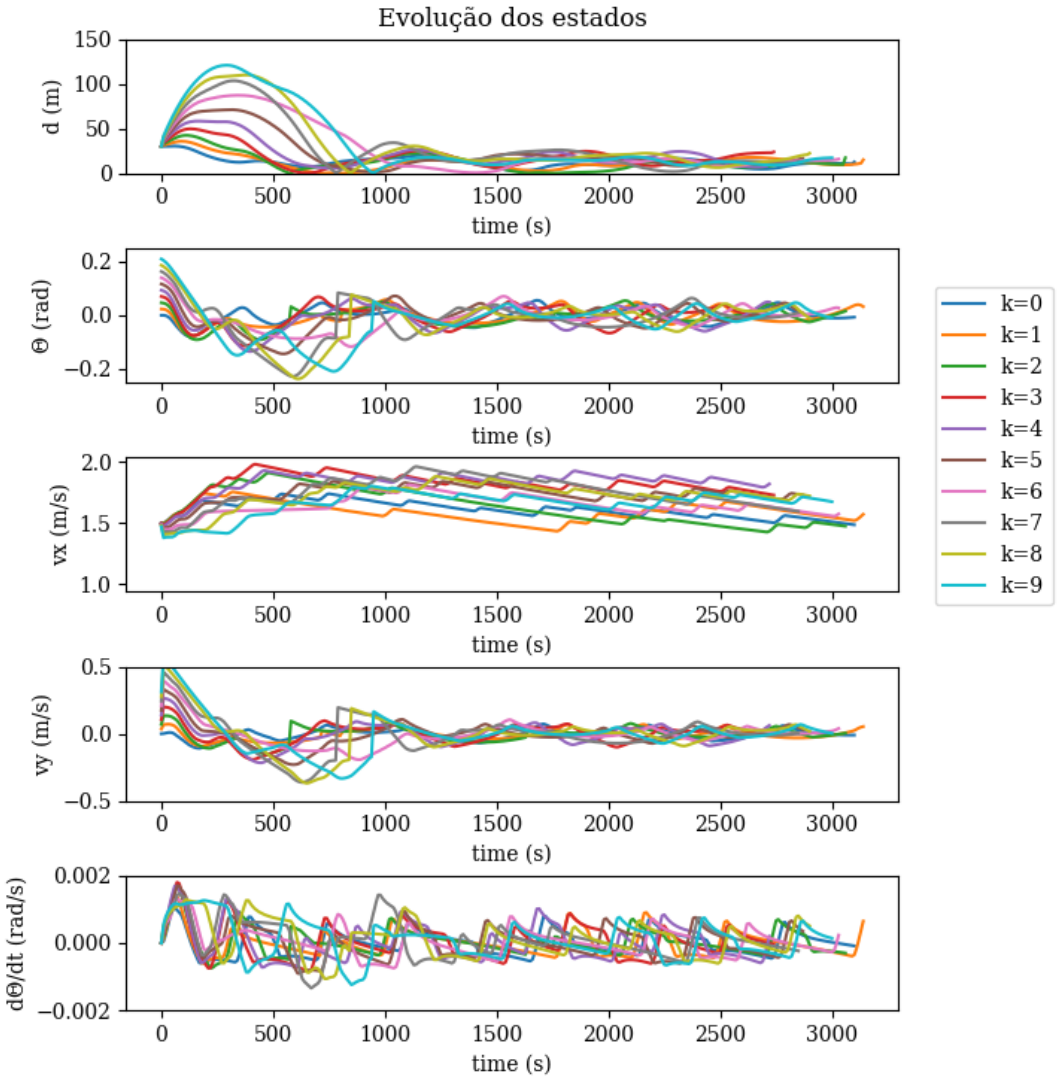

Figura 19: Evolução de estados observáveis DQN SSN

7.2.5 DDPG

7.2.6 Performance no SSN

- Resultado cenário 1:

  Nenhuma Colisão. Convergência geral da distância, não convergência da velocidade.

- Resultado cenário 2:

  Durante os 10 episódios houve a convergência da distância, com pouca ou nenhuma oscilação do navio como observado na figura 20

  Além disso observa-se que não há convergência para a velocidade desejada, entretanto, isso não torna-se um problema pois espera-se convergir uma vez que haja mais episódios de treinamento durante a TA.

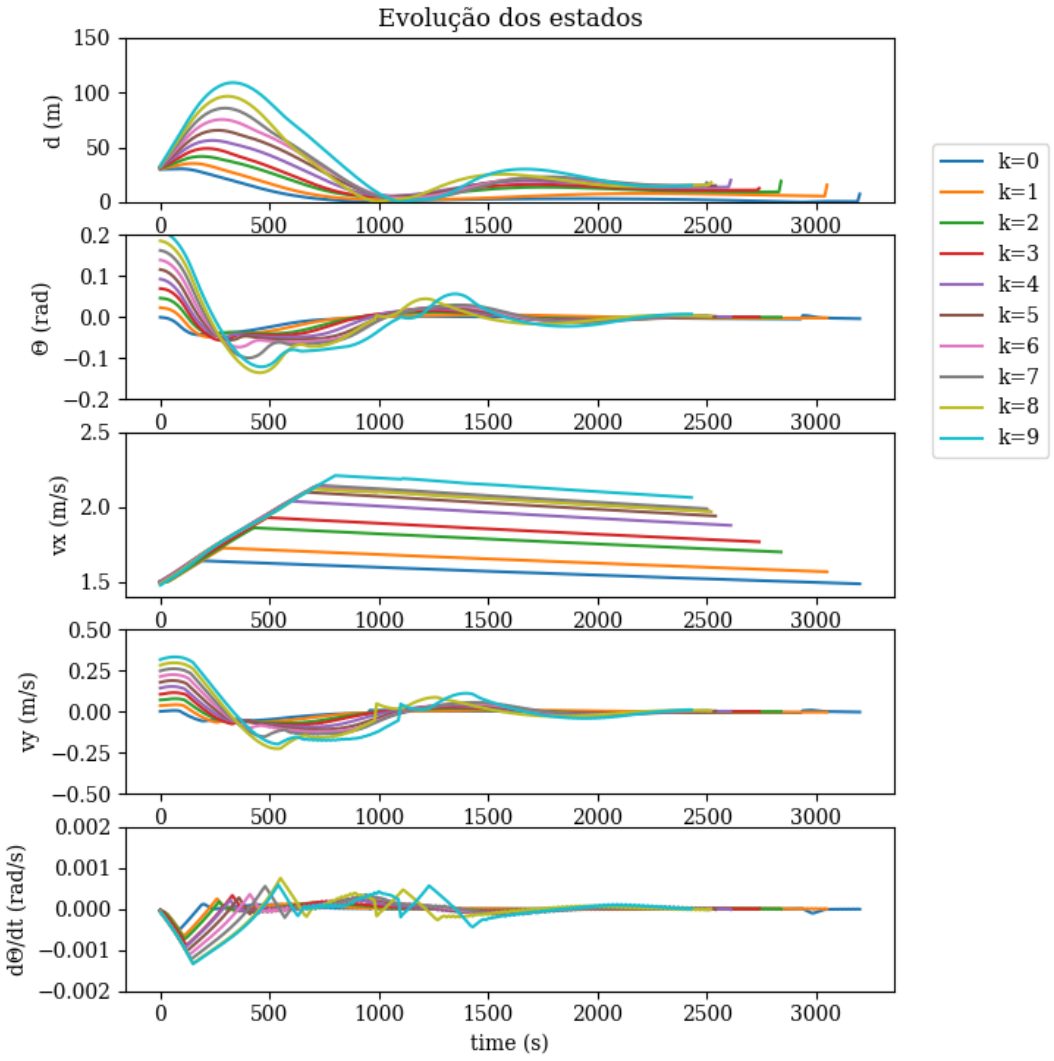

Figura 20: Evolução de estados observáveis DDPG SSN

7.2.7 Performance no TPN antes da TA

- Resultado cenário 1: Nenhuma Colisão. Convergência geral da distância, não convergência da velocidade.

- Resultado cenário 2:

Em 9 dos 10 episódios houve a convergência da distância (1 colisão), com leve oscilação do navio como observado na figura 21.

Além disso observa-se que não há convergência para a velocidade desejada, entretanto, como já citado anteriormente espera-se em velocidade após a TA.

É notável que o agente seguindo a política aprendida no SSN é suficiente para controlar a posição do navio durante a navegação sem que haja nenhuma colisão. Ainda que a velocidade não tenha convergido, e que haja oscilações, é esperado que a TA supere esses problemas.

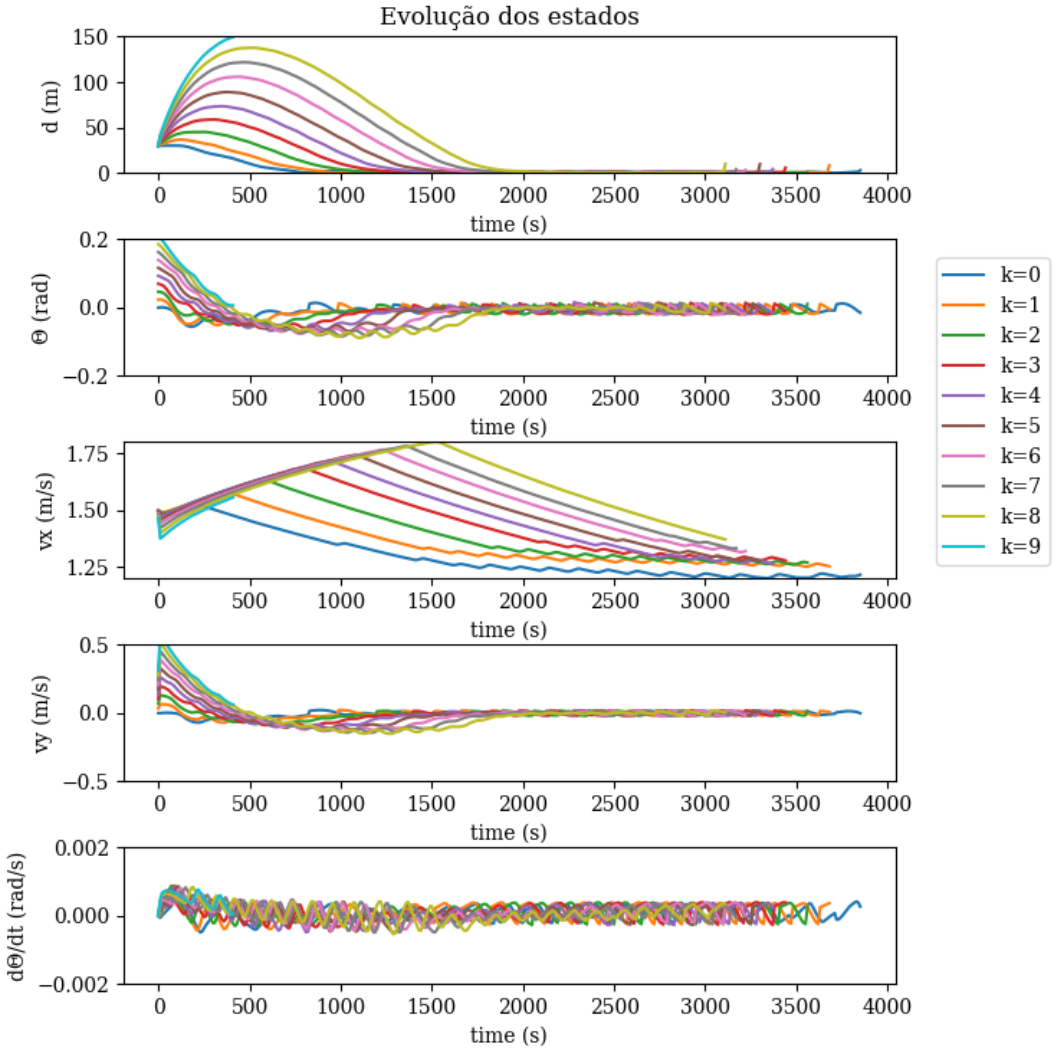

Figura 21: Evolução de estados observáveis DDPG TPN antes TA

7.2.8 Performance no TPN após a TA

- Resultado cenário 1: Nenhuma Colisão. Convergência geral da distância, convergência da velocidade.

- Resultado cenário 2:

  Durante os 10 episódios houve a convergência da distância, com uma leve ou nenhuma oscilação do navio como observado na figura 22. Além disso, observa-se que após a TA houve uma tendência de convergência para a velocidade desejada no fim dos episódios. A velocidade de 1.8m/s (10% do setpoint) é atingida para todos as inicializações por volta da iteração 138, ou seja $t = 1380s$. Isso indica um aperfeiçoamento da política de controle no ambiente de simulação TPN.

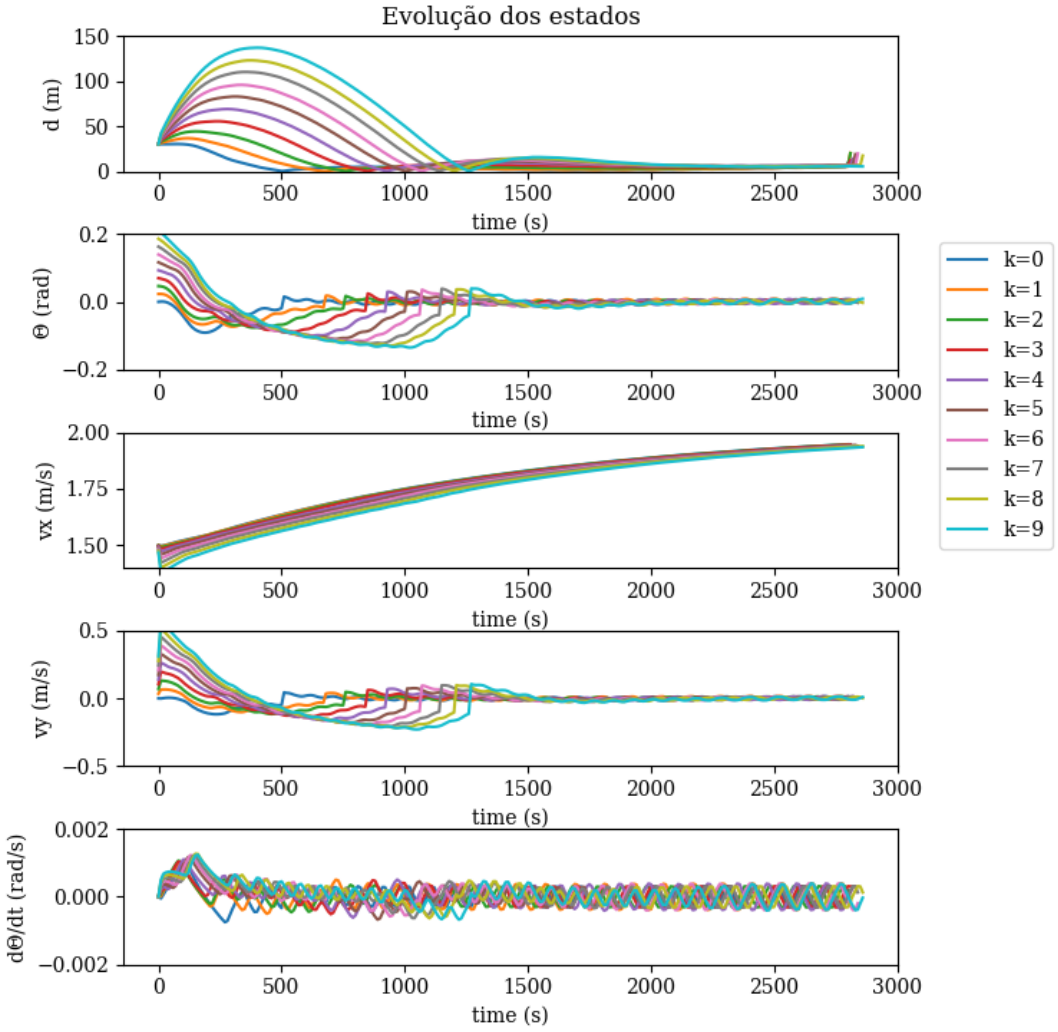

Figura 22: Evolução de estados observáveis DDPG TPN após TA

7.3 Performance do ponto de vista de Controle

Em métodos de Controle Clássico uma estrutura de controle é predefinida para realizar uma tarefa de controle bem estabelecida em um sistema físico que normalmente é modelado e utilizado para o design do controlador, como é o caso dos controles PID, LQR e Controle Robusto.

Tal estrutura após desenvolvida é analisada do ponto de vista do cumprimento de sua tarefa em termos de robustez, tempo de assentamento, erro estacionário, entre outros. Analisa-se também o acionamento e gasto energético do controlador.

Do ponto de vista da APR, porém, o sistema de controle pode ser entendido como a política aprendida pelo agente durante a aprendizagem, e nesse caso não se realiza diretamente o design das saídas das ações de controle. Sendo assim é essencial analisar-se o desempenho do controle assim como sua viabilidade.

7.3.1 DQN

7.3.2 Tempo de Subida (Rise Time)

Para o caso do DQN não é possível analisar o tempo de assentamento em 10%, uma vez que a margem de 10% não é alcançada devido às oscilações. Pode-se, porém, observar que para todos os testes do cenário 2 descrito em 7.2, o tempo de subida (rise time) é em torno de 1000 s, comparando-se com o DDPG, o qual possui um tempo de subida entre 500 e 1200 s, observa-se que o DQN possui uma um tempo de resposta com menor variância neste experimento.

7.3.3 Ações de controle

As ações de leme e propulsão para o caso em que $k = 4$ descrito em 7.2, cenário 2 é apresentado na figura 23.

Para as ações de leme, observa-se um caráter oscilatório entre os extremos de controle [-1/3, 1/3], com poucas graduações de amplitude, o agente utiliza-se apenas dos níveis 0.3, 0.2 0.1 e -0.3. Esse resultado é fisicamente inviável, pois ainda que o tempo de transição entre as posições do leme seja fisicamente realizável, o gasto energético e a possível perda de estabilidade de controle limitariam uma aplicação prática desse tipo de controle.

Para as ações de propulsor, observa-se uma amplitude variável entre 0, 0.1 e 0.2, ainda

que não haja pleno controle alcance do controle de velocidade adequada.

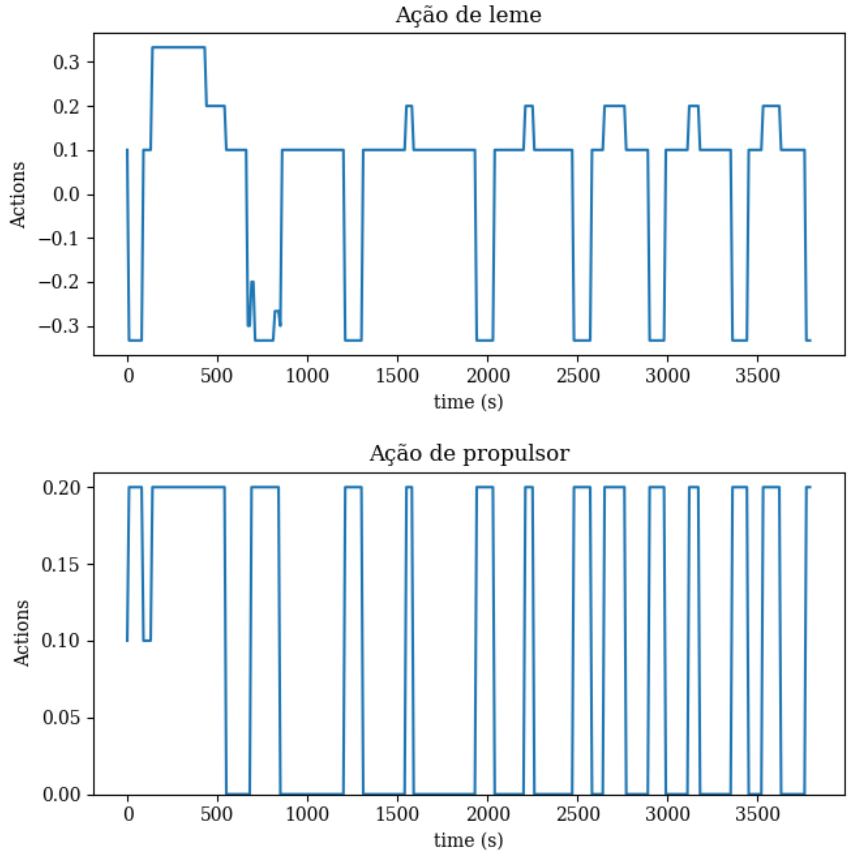

Figura 23: Evolução da ação, para o cenário 2 com k=4 DQN TPN após TA

7.3.4 DDPG

7.3.5 Tempo de assentamento (Settling Time)

A figura a seguir apresenta o tempo de assentamento para os 10 experimentos descritos em 7.2, cenário 2.

O tempo de assentamento (10%) para a distância é crescente com relação ao número de episódios, uma vez que o ângulo de aproamento inicial aumenta com k, como descrito no experimento. No caso mais favorável obtém-se um tempo de cerca de 250 s e no mais desfavorável de cerca de 1100 s.

O tempo de assentamento da velocidade é levemente decrescente, mas é em torno de 1400 s.

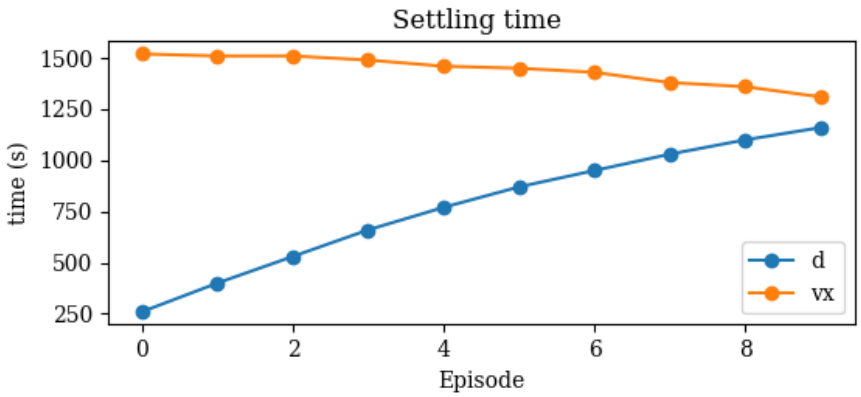

Figura 24: Tempo de assentamento DDPG

7.3.6 Ações de controle

As ações de leme e propulsão para o caso em que $k = 10$ (cenário 2) descrito em 7.2 é apresentado na figura 25.

Para as ações de leme, observa-se um caráter oscilatório entre os extremos de controle [-1/3, 1/3], com poucas variações de amplitude. Esse resultado é fisicamente inviável, pois ainda que o tempo de transição entre as posições do leme seja fisicamente realizável, o gasto energético e a possível perda de estabilidade de controle limitariam uma aplicação prática desse tipo de controle. A explicação para esse comportamento pode estar ligada à função de ativação da última camada da rede neural ($softsign$), a qual possui uma pequena zona de transição entre os extremos (+1, -1).

Para as ações de propulsor, observa-se uma amplitude quase constante em torno de 0.24, o limite máximo da ação de propulsão.

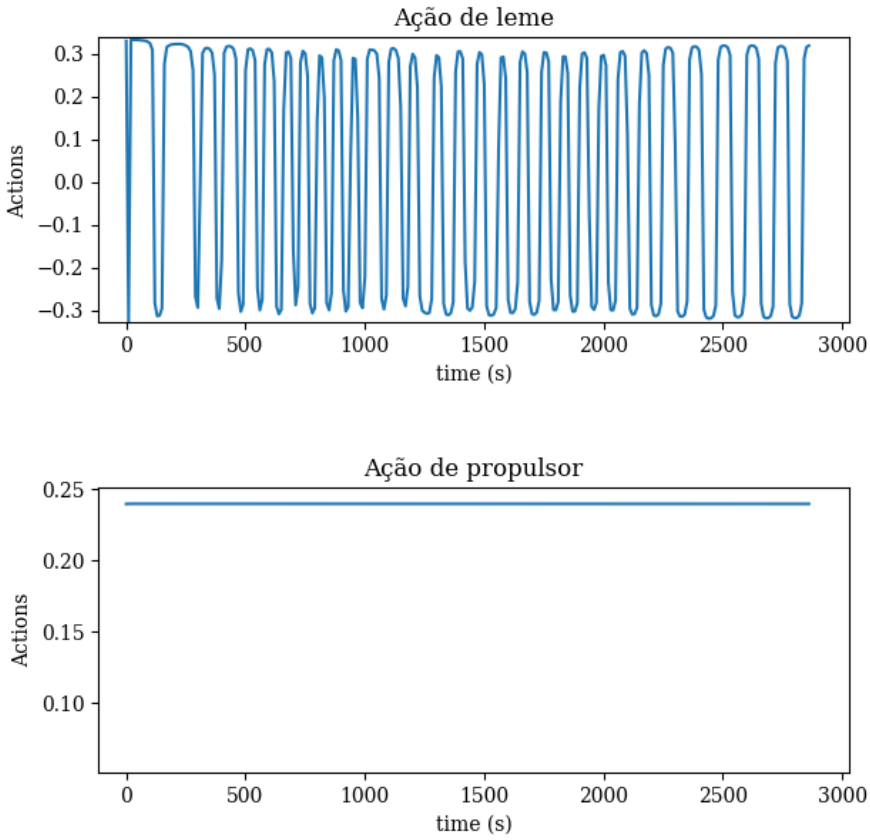

Figura 25: Evolução da ação, para o cenário 2 com k=10 DDPG TPN após TA

7.4 Avaliação de requisitos

7.4.1 Requisitos Primários

Os requisitos primários foram cumpridos pela aplicação do método de DDPG, porém não foram cumpridos pelo DQN no que se refere ao setpoint de velocidade e posição.

O requisito RP2 foi adaptado para que a solução pudesse implementar a TA, como explicado em 6.1.2.

7.4.2 Requisitos Secundários

o RS1 foi cumprido uma vez que realizou-se uma comparação entre os métodos DQN e o DDPG.

o RS2 não foi explorado no escopo do trabalho.

8 CONCLUSÃO

Dados os resultados mostrados no capítulo anterior, pode-se concluir que é possível desenvolver uma lei de controle para o problema de navegação em águas restritas utilizando-se métodos de Aprendizagem por Reforço sem uma complexidade muito elevada.

O método DDPG mostrou um bom desempenho no cumprimento da tarefa, e o método DQN mostrou-se insuficiente para controlar a velocidade e a direção do navio, apresentando oscilações de distância.

Apesar de cumprir seus objetivos, os métodos apresentam limitações para apelações práticas tais como: a avaliação da robustez do sistema perante situações não conhecidas pelo agente; a adaptação para a implementação de uma lei de controle menos custosa do ponto de vista energético; a viabilidade de navegação dado os padrões de navegação oscilante na trajetória.

8.1 Perspectivas futuras

Dada a criticidade do sistema abordado (acidentes não são tolerados pois podem ter consequências drásticas), para a consolidação do uso de Aprendizagem por Reforço no controle de embarcações, é necessário um estudo extensivo dos parâmetros utilizados no treinamento. Para tanto sugere-se a verificação da influência da atualização do modelo alvo e da parametrização dos processos estocásticos no comportamento da resposta dada pelo APR. Tal linha de estudos pode revelar influências e correlações entre o comportamento de navegação e os de hiper-parâmetros, as quais não foram abordadas no escopo desse trabalho.

Uma outra linha de trabalho possível é a aplicação de um filtro de tipo passa-baixa nas ações de comando geradas pelo APR afim de suavizar a resposta do sistema e evitar as oscilações inviáveis apresentadas no comando de leme, por exemplo. Visto que a função

de ativação $softsign$ na RNA utilizada pelo ator DDPG possui uma região de transição pequena, isso pode justificar a transição abrupta das ações de controle, que resulta nas oscilações indesejáveis no leme. A utilização de outra função de ativação, principalmente na camada de saída, de tipo linear (ou outro que possua uma transição gradual) também poderia ser uma linha de estudo para solucionar esse inconveniente.

Como uma opção extra para que o controle se torne mais contínuo tem-se a possibilidade de incluir o ângulo de leme nos estados observáveis na APR e de se modificar a ação do comando de leme para que ela se torne incremental, afim de que as ações do agente sejam sempre viáveis fisicamente e visando uma redução energética. Finalmente, pode-se também estudar o desacoplamento das ações de leme e de propulsão do afim de analisar-se o cumprimento dos objetivos de maneira independente.

REFERÊNCIAS

1 GERLA, M. et al. Internet of vehicles: From intelligent grid to autonomous cars and vehicular clouds. In: 2014 IEEE World Forum on Internet of Things (WF-IoT). IEEE, 2014. Disponível em: <https://doi.org/10.1109/wf-iot.2014.6803166>.

2 CUTLER, M.; HOW, J. P. Efficient reinforcement learning for robots using informative simulated priors. In: 2015 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2015. Disponível em: <https://doi.org/10.1109/icra.2015.7139550>.

3 HETHERINGTON, C.; FLIN, R.; MEARNS, K. Safety in shipping: The human element. Journal of Safety Research, Elsevier BV, v. 37, n. 4, p. 401–411, jan 2006. Disponível em: <https://doi.org/10.1016/j.jsr.2006.04.007>.

4 AHMED, Y. A.; HASEGAWA, K. Experiment results for automatic ship berthing using artificial neural network based controller. IFAC Proceedings Volumes, Elsevier BV, v. 47, n. 3, p. 2658–2663, 2014. Disponível em: <https://doi.org/10.3182/20140824-6-za-1003.00538>.

5 FILHO, A. N. Q.; ZIMBRES, M.; TANNURI, E. A. Development and validation of a customizable DP system for a full bridge real time simulator. In: Volume 1A: Offshore Technology. ASME, 2014. Disponível em: <https://doi.org/10.1115/omae2014-23623>.

6 DOUGHERTY, M. A review of neural networks applied to transport. Transportation Research Part C: Emerging Technologies, Elsevier BV, v. 3, n. 4, p. 247–260, aug 1995. Disponível em: <https://doi.org/10.1016/0968-090x(95)00009-8>.

7 HAFNER, R.; RIEDMILLER, M. Reinforcement learning in feedback control. Machine learning, Springer, v. 84, n. 1-2, p. 137–169, 2011.

8 AMENDOLA, J. Batch reinforcement learning of feasible trajectories in a ship maneuvering simulator. In: Anais do XV Encontro Nacional de Inteligência Artificial e Computacional (ENIAC). [S.l.: s.n.], 2018.

9 STAMENKOVICH, M. An application of artificial neural networks for autonomous ship navigation through a channel. In: IEEE PLANS 92 Position Location and Navigation Symposium Record. IEEE. Disponível em: <https://doi.org/10.1109/plans.1992.185865>.

10 LACKI, M. Reinforcement learning in ship handling. TransNav: International Journal on Marine Navigation and Safety of Sea Transportation, v. 2, n. 2, 2008.

11 RAK, A.; GIERUSZ, W. Reinforcement learning in discrete and continuous domains applied to ship trajectory generation. Polish Maritime Research, Walter de Gruyter GmbH, v. 19, n. Special, p. 31–36, oct 2012. Disponível em: <https://doi.org/10.2478/v10012-012-0020-8>.

12 NAGENDRA, S. et al. Comparison of reinforcement learning algorithms applied to the cart-pole problem. In: 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI). IEEE, 2017. Disponível em: <https://doi.org/10.1109/icacci.2017.8125811>.

13 KIUMARSI, B. et al. Reinforcement -learning for optimal tracking control of linear discrete-time systems with unknown dynamics. Automatica, Elsevier BV, v. 50, n. 4, p. 1167–1175, apr 2014. Disponível em: <https://doi.org/10.1016/j.automatica.2014.02.015>.

14 XU, Q. et al. Deep convolutional neural network-based autonomous marine vehicle maneuver. International Journal of Fuzzy Systems, Springer Nature, v. 20, n. 2, p. 687–699, sep 2017. Disponível em: <https://doi.org/10.1007/s40815-017-0393-z>.

15 KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

16 SUTTON, R. S.; BARTO, A. G. Reinforcement Learning: An Introduction. Second. The MIT Press, 2018. Disponível em: <http://incompleteideas.net/book/the-book-2nd.html>.

17 WATKINS, C. J. C. H. Learning from delayed rewards. Tese (Doutorado) — King's College, Cambridge, 1989.

18 MNIH, V. et al. Human-level control through deep reinforcement learning. Nature, Nature Publishing Group, v. 518, n. 7540, p. 529, 2015.

19 LILLICRAP, T. P. et al. Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971, 2015.

20 TORREY, L.; SHAVLIK, J. Transfer learning. In: Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques. [S.l.]: IGI Global, 2010. p. 242–264.

21 FOSSEN, T. I. Handbook of Marine Craft Hydrodynamics and Motion Control. John Wiley & Sons, Ltd, 2011. Disponível em: <https://doi.org/10.1002/9781119994138>.

22 TANNURI, E. A. Desenvolvimento de metodologia de projeto de sistema de posicionamento dinâmico aplicado a operações em alto-mar. Tese (Doutorado). Disponível em: <https://doi.org/10.11606/t.3.2002.tde-04082003-173204>.

23 ABBEEL, P. et al. An application of reinforcement learning to aerobatic helicopter flight. In: SCHöLKOPF, B.; PLATT, J. C.; HOFFMAN, T. (Ed.). Advances in Neural Information Processing Systems 19. MIT Press, 2007. p. 1–8. Disponível em: <http://papers.nips.cc/paper/3151-an-application-of-reinforcement-learning-to-aerobatic-helicopter-flight.pdf>.

24 LAU, B. Using keras and deep deterministic policy gradient to play torcs. Ben Lau, 2016. Disponível em: <https://yanpanlau.github.io/2016/10/11/Torcs-Keras.html>.

25 BROCKMAN, G. et al. Openai gym. CoRR, abs/1606.01540, 2016.

26 PLAPPERT, M. keras-rl. [S.l.]: GitHub, 2016. <https://github.com/keras-rl/keras-rl>.

APÊNDICE A – PARÂMETROS DO SIMULADOR

A.1 Parâmetros do Navio

Tabela 3: Coeficientes da matriz de massa e massa adicional

M	115000	10^6 Kg
Iz	414000000	10^6 Kg.m^2
M11	14840.4	10^6 Kg
M22	174050	10^6 Kg
M26	38369.6	10^6 Kg.m
M62	36103	10^6 Kg.m
M66	364540000	10^6 Kg.m^2

Tabela 4: Dimensões do Navio

Comprimento (L)	244.745	m
Boca (B)	42	m
Calado [Draft] (D)	15.3	m
Centro de Massa (Xg, Yg, Zg)	(2.223 , 0.000, 12.300)	m
Posição do leme (Xl, Yl, Zl)	(-115, 0, 6)	m
Posição do Propulsor (Xp, Yp, Zp)	(-112.4, 0, 3.5)	m
Área molhada (S)	27342	m^2
Centro de Cross-Flow (lp)	7.65	m

A.2 Parâmetros Hidrodinâmicos

Tabela 5: Constantes hidrodinâmicas da água e parâmetros aproximados de arrasto do navio

Densidade da água (ρ)	1.025	10^3 Kg·m^-3
Viscosidade Dinâmica(μ)	1.002	10^-3 kg(m·s)^-1
Coeficiente de arrasto lateral (Cy)	0.06	adimensional
Coeficiente de Bloco (Cb)	0.85	adimensional

A.3 Parâmetros do propulsor

Tabela 6: Parâmetros dimensionais do propulsor

Rotação Máxima (np)	1.6	Hz
Limites de Empuxo (Tpmin, Tpmax)	(-1377, 2500)	kN
Diâmetro (Dp)	7.2	m

A.4 Parâmetros Do Leme

Tabela 7: Parâmetros dimensionais do leme

Limites de ângulo (ϕ_{max}, ϕ_{min})	(-30, 30)	º (graus)
Área efetiva (A_{rud})	68	m^2
Razão de aspecto (Λ)	2	adimensional