UNIVERSIDADE FEDERAL DE UBERLÂNDIA

TÉRCIO DE MELO ALVES JÚNIOR

IMPLEMENTAÇÃO DE ALGORITMOS DE APRENDIZADO POR REFORÇO NO CONTROLE DE UM CARRO PÊNDULO

Uberlândia, Brasil

2023

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

TÉRCIO DE MELO ALVES JÚNIOR

IMPLEMENTAÇÃO DE ALGORITMOS DE APRENDIZADO POR REFORÇO NO CONTROLE DE UM CARRO PÊNDULO

Trabalho de conclusão de curso apresentado à Faculdade de Engenharia Mecânica da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Engenharia Mecatrônica.

Orientador: Pedro Augusto Queiroz de Assis

Universidade Federal de Uberlândia – UFU

Faculdade de Engenharia Mecânica

Bacharelado em Engenharia Mecatrônica

Uberlândia, Brasil

2023

TÉRCIO DE MELO ALVES JÚNIOR

IMPLEMENTAÇÃO DE ALGORITMOS DE APRENDIZADO POR REFORÇO NO CONTROLE DE UM CARRO PÊNDULO

Trabalho de conclusão de curso apresentado à Faculdade de Engenharia Mecânica da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Engenharia Mecatrônica.

Trabalho aprovado. Uberlândia, Brasil, 23 de Agosto de 2023:

Pedro Augusto Queiroz de Assis
Orientador

Fran Sergio Lobato

Leonardo Rosa Ribeiro da Silva

Uberlândia, Brasil
2023

Resumo

No *Reinforcement Learning* (RL) ou aprendizado baseado em reforço, pretende-se fazer com que o agente aprenda a realizar uma tarefa por meio de interações com o ambiente. Sendo um ramo da Inteligência Artificial não supervisionado, nos métodos de RL o aprendizado é realizado com base em tentativa e erro. O objetivo é determinar a sequência de ações que maximizam o somatório de recompensas a longo prazo, o que é denominado retorno. Neste trabalho serão avaliados quatro algoritmos de aprendizado baseado em reforço para o controle de um carro pêndulo. Em particular são implementados SARSA, *Q-Learning*, *Deep Q-Learning* e *Double Deep Q-Learning*. O objetivo dos métodos consiste em treinar um agente para manter uma haste equilibrada e o carro dentro de um certo intervalo em torno da origem. Com esse propósito é definida uma função que retorna recompensas positivas, caso esses objetivos sejam alcançados, e recompensas negativas, caso contrário. Para tais objetivos, será utilizado a linguagem Python, tanto para implementação, quanto para simulação e exibição dos resultados. Os resultados da simulação no modelo não linear do sistema demonstram a capacidade de aprendizado do agente, pois foi possível realizar a tarefa proposta utilizando todos os métodos implementados. Mais ainda, usando essas simulações, avaliam-se efeitos de variações nos parâmetros de ajuste dos métodos no aprendizado do agente. Esses resultados podem orientar outros projetistas na implementação dos métodos de RL considerados. Constatou-se que o método *Double Deep Q-Learning* proporcionou um aprendizado mais rápido, tornando-se assim o melhor entre os métodos testados.

Palavras-chave: SARSA, *Reinforcement Learning*, *Q-Learning*, *Deep Q-Learning*, *Double Deep Q-Learning*, *Artificial Inteligence*, Inteligência Artificial, Carro pêndulo.

Abstract

In Reinforcement Learning (RL), the goal is to enable an agent to learn how to perform a task through interactions with the environment. As a branch of unsupervised Artificial Intelligence, RL methods facilitate learning by means of trial and error. The objective is to determine a sequence of actions which maximize the cumulative long-term rewards, referred to as the return. Herein four RL methos are employed for controlling a pendulum cart. Specifically, SARSA, Q-Learning, Deep Q-Learning, and Double Deep Q-Learning are implemented. The methods aim to train an agent to balance the pendulum and keep the cart within a certain range around its starting position. To achieve these objectives, a reward function is defined, providing positive rewards when these goals are met and negative rewards otherwise. Python is utilized for both implementation and simulation, as well as displaying the results. Simulation results in the nonlinear model of the system demonstrate the agent's learning capabilities, as all implemented methods successfully accomplished the proposed task. Furthermore, by using these simulations, the effects of variations in method tuning parameters on the agent's learning are evaluated. This can guide other developers in implementing the considered RL methods. It was observed that the Double Deep Q-Learning method resulted faster learning for the particular application considered in this work.

Palavras-chave: SARSA, *Reinforcement Learning*, *Q-Learning*, *Deep Q-Learning*, *Double Deep Q-Learning*, *Artificial Inteligence*.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

DP	*Dynamic Programming*, Programação Dinâmica
DQN	*Deep Q-Network*
GPI	*Gradient Policy Iteration*, Iteração de Política por Gradiente
IA	Inteligência Artificial
MDP	Markov Decision Process, Processo de deisão de Markov
RL	*Reinforcement Learning*, Aprendizado por reforço
RNA	Rede Neural Artifical
RELU	*Rectified Linear Unit*, Unidade Linear Retificada
SARSA	*State-Action-Reward-State-Action*, Estado-Ação-Recompensa-Estado-Ação

Sumário

1 Introdução

A interação entre indivíduo e ambiente é crucial no aprendizado de uma certa tarefa. Desde aprender a engatinhar ou a andar, crianças instintivamente correlacionam ações a recompensas negativas, como a dor de uma queda, ou positivas, no caso do êxito de suas vontades. Na psicologia, o aprendizado por tentativa e erro foi investigado pela por Conway Lloyd Morgan em 1894, descrevendo o aprendizado animal. Na computação, o sistema de prazer-dor proposto por Alan Turing pode ser considerado pioneiro no ramo de inteligência artificial denominado aprendizado por reforço (RL, do inglês *Reinforcement Learning*). No RL, objetiva-se determinar uma sequência de decisões por um agente para atingir um objetivo (SUTTON; BARTO, 2018). Especificamente no aprendizado por reforço, considera-se um agente em um ambiente desconhecido. Em cada instante de tempo t, baseando-se no estado atual $s(t)$, o agente escolhe uma ação $a(t)$ de acordo com uma política $\pi(a(t)|s(t))$. Então o ambiente retorna uma observação $s(t+1)$ e uma recompensa $r(t)$. Os algoritmos de RL visam modificar a política de decisões do agente, de modo a maximizar o somatório das recompensas ao longo do tempo(SCHULMAN, 2016). A ideia fundamental do RL é ilustrada na Figura 1.

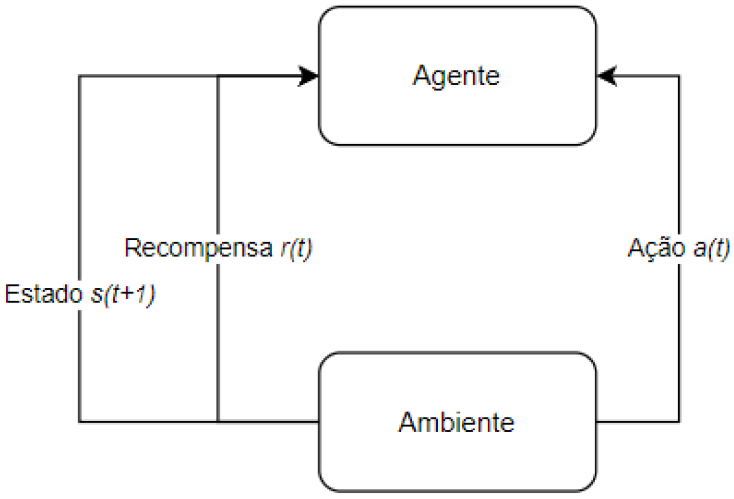

Figura 1 – Ilustração da interação entre agente e ambiente em um algoritmo de aprendizado por reforço.

Existem alguns desafios na implementação do aprendizado por reforço. Por exemplo, a necessidade de grande quantidade de experiências para um aprendizado adequado. Outra dificuldade é relacionada ao sinal de recompensa, que pode ser ruidoso, atrasado ou até mesmo difícil de se caracterizar (MNIH et al., 2013). Por outro lado, quando comparados aos métodos tradicionais de controle, o RL apresenta a vantagem de aprender através da interação com o ambiente, não necessitando das etapas de análise do sistema, construção do modelo matemático e projeto do controlador. Por esse motivo, o RL pode ser aplicado aos mais diversos sistemas: lineares e não lineares, com múltiplas variáveis de entrada e/ou de saída. Consequentemente, o número de aplicações de RL também cresce, tendo usos desde jogos eletrônicos (MNIH et al., 2013) até em controle de helicópteros e carros (HAFNER; RIEDMILLER, 2011).

Os métodos de RL utilizam dois elementos principais: política e recompensa. A política define como o agente interage com o ambiente. Em outras palavras, a política rege como é escolhida a ação a partir de um determinado estado. A recompensa, por sua vez, caracteriza o objetivo do agente no aprendizado por reforço. Como mostrado na Figura 1 em cada instante de tempo, o ambiente retorna o sinal de recompensa. O objetivo do agente é maximizar o somatório das recompensas ao longo da execução da tarefa (SUTTON; BARTO, 2018). Esse somatório é denominado retorno.

Existem duas abordagens principais de aprendizado por reforço: métodos de otimização de política (denominados *Policy Gradient*) e os métodos de programação dinâmica. Os primeiros funcionam modificando diretamente a política do agente para maximizar o retorno. Com esse propósito, adotam-se técnicas para estimar o gradiente da função retorno em termos dos parâmetros da política (ACHIAM, 2018). Já os métodos de programação dinâmica tentam prever o retorno esperado para cada ação possível em um estado. A partir dessa previsão, as ações são tomadas de modo a maximizar o retorno (SCHULMAN, 2016). Como uma introdução à área de aprendizado por reforço, o presente trabalho envolve a implementação de técnicas de Programação Dinâmica no controle de um carro-pêndulo. Em específico, as conhecidas técnicas SARSA, Q-learning, Deep Q-learning e Double Deep Q-learning serão adotadas.

O sistema do carro-pêndulo é constituído por um carro de movimento retilíneo, no qual encontra-se fixada uma haste que pode rotacionar livremente (vide Figura 2). O estado do sistema é composto por posições e velocidades da haste e do carro. Já a ação manipulada é uma força horizontal aplicada ao carro. Os métodos de RL serão empregados para manter a haste na posição vertical independentemente da condição inicial e também manter o carro parado na posição inicial.

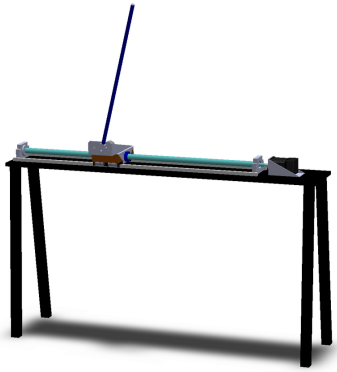

Figura 2 – Ilustração de um carro-pêndulo. (MARTINS; VIEIRA, 2015)

Para atingir esse objetivo, o restante do trabalho foi dividido e será apresentado no decorrer do presente documento do seguinte modo:

- O Capítulo 2 abordará as definições dos elementos essenciais para a construção de um sistema de aprendizado por reforço, e também introduzirá os métodos utilizados. Especificamente, SARSA, Q-Learning, Deep Q-Learning e Double Deep Q-Learning.
- No Capítulo 3 descrever-se-á o carro-pêndulo, detalhando estado, ação, tarefa de controle e função de recompensa.
- No Capítulo 4 serão apresentados os resultados obtidos, bem como discussões sobre o desempenho e o aprendizado do agente. Nesse capítulo também será apresentada uma comparação entre os métodos de RL adotados.
- O Capítulo 5 trará as conclusões do trabalho, e as sugestões de trabalhos futuros.

2 Aprendizado por Reforço

Neste capítulo são apresentados os conceitos básicos de aprendizado por reforço bem como os algoritmos adotados no controle do carro-pêndulo.

2.1 Visão Geral

No aprendizado por reforço objetiva-se maximizar o retorno de um agente na execução de uma tarefa. Considera-se que em cada instante de tempo discreto t, o agente percebe o estado em que se encontra $s(t)$ e escolhe uma ação $a(t)$ de acordo com a política $\pi(a(t)|s(t))$. Essa ação é aplicada ao ambiente dando origem a uma recompensa $r(t)$ e a um novo estado $s(t+1)$. Assim o ciclo se reinicia (SUTTON; BARTO, 2018).

O aprendizado por reforço é tido como um método de aprendizado não supervisionado, uma vez que, por mais que o agente tenha acesso a recompensa imediata, não é por essa que o mesmo se guia para tomar ações. A experiência é um fator decisivo para o aprendizado, pois quanto mais se dispõe de um conjunto de ações, recompensas e novos estados, melhor se consegue tomar uma ação que maximize o retorno (KAELBLING; LITTMAN; MOORE, 1996).

2.1.1 Processo de Decisão de Markov

No RL, consideram-se sistemas que possam ser descritos como um processo de decisão de Markov, MDP do inglês *Markov Decision Process*. Um processo é um MDP caso (FUKUSHIMA, 2020):

$$P(s(t+1)|s(t)) = P(s(t+1)|s(1),s(2),\ldots,s(t)) \tag{2.1}$$

em que $P(A|B,C)$ significa a probabilidade do estado A acontecer visto que os estados B e C aconteceram. Dessa forma, pode-se falar que a probabilidade de $s(t+1)$ ocorrer dado $s(t)$ é igual a probabilidade de $s(t+1)$ ocorrer dados $s(i)$ para $i = 1, 2, \ldots, t$. Assim um sistema pode ser considerado como um MDP quando $s(t+1)$ pode ser inteiramente descrito por $s(t)$. Nesse trabalho, considerar-se-ão MDPs completamente descritos por $P(s(t+1), r(t)|s(t), a(t))$ indicando probabilidade de atingir um novo estado $s(t+1)$ e a recompensa $r(t)$ estando no estado $s(t)$ e tomando a ação $a(t)$.

2.2 Retorno

A recompensa (ou reforço) caracteriza o objetivo do agente no aprendizado por reforço. A escolha de uma ação visa maximizar o acúmulo desses reforços a longo prazo, o

que é chamado de retorno. Uma primeira possibilidade para calcular o retorno é a seguinte:

$$G(t) = \sum_{t=0}^{h} r(t) \tag{2.2}$$

em que h indica o tempo final da tarefa. Já no caso de tarefas contínuas, isto é, aquelas que não têm tempo final definido, pode-se adotar

$$G(t) = \sum_{t=0}^{\infty} \gamma^t r(t) \tag{2.3}$$

sendo $0 \leq \gamma < 1$ um fator de desconto. Esse fator é adotado para ponderar a importância de recompensas futuras. Dessa forma, um γ próximo a 0 representa que as recompensas a curto prazo valem muito mais. Já com γ próximo a 1, tem-se uma importância similar entre recompensas de curto e longo prazo. Esse parâmetro de ajuste influencia no tempo de aprendizado do agente. Isso será investigado nesse trabalho.

2.3 Política

Por definição a política π é o conjunto de regras que rege como é escolhida a ação a partir de um determinado estado. Exemplos de políticas são redes neurais, leis de controle e tabelas relacionando estado e entrada de um sistema. Quanto à classificação de políticas temos (PELLEGRINI; WAINER, 2007):

- Total, se todos os estados possíveis são mapeados em ações pela política;
- Parcial, se alguma de suas regras não são aplicáveis a alguns estados do MDP.
- Estacionária, se a ação independe da época de decisão;
- Não-estacionária, se a ação tomada depende da época de decisão;
- Markoviana (ou sem memória), quando a escolha da ação depende apenas do estado corrente;
- Não-Markoviana, quando a escolha da ação depende de todo o histórico de ações e estados do sistema até o momento.

Com o propósito de conciliar o quanto o agente deve explorar outras oportunidades e o quanto o agente deve aproveitar o conhecimento já adquirido, uma política bastante empregada na área de RL é conhecida como ε-greedy. Nessa política sorteia-se um número aleatório entre 0 e 1. Caso tal número seja menor do que um parâmetro escolhido pelo projetista, o agente tomará ações aleatórias. Caso contrário, será escolhida a ação que maximiza o retorno de acordo com o conhecimento já adquirido. Dessa forma, através da escolha desse parâmetro, é possível balancear entre exploração e aproveitamento (SUTTON; BARTO, 2018).

2.4 Função Valor

A função valor define, seguindo a política π, a partir do estado atual $s(t)$ até o estado terminal, qual a esperança do valor do retorno $G(t)$. Matematicamente,

$$V_\pi(s(t)) = E_\pi[G(t)|s(t) = s] \tag{2.4}$$

2.4.1 Equação de Bellman

A equação de Bellman permite estimar o valor de um estado $s(t)$ por meio de um procedimento recursivo e de experiências adquiridas junto ao ambiente. Essa equação é composta pela função valor decomposta em duas partes, a recompensa imediata $r(t)$ e o retorno descontado do estado seguinte $\gamma V_\pi(s(t+1))$. A equação de Bellman é importante uma vez que separa o problema em duas partes, podendo-se escrever o valor do estado $s(t)$ em função da recompensa imediata e mais uma parte recursiva de $s(t+1)$. Matematicamente, essa equação pode ser obtida conforme demonstrado a seguir (SUTTON; BARTO, 2018):

$$V_\pi(s(t)) = E_\pi[G(t)|S(t) = s] \tag{2.5}$$

$$V_\pi(s(t)) = E_\pi\left[r(t+1) + \gamma G(t+1)|S(t) = s\right] \tag{2.6}$$

A equação anterior pode ser ainda mais desenvolvida do seguinte modo:

$$V_\pi(s) = \sum_a \pi(a(t)|s(t)) \sum_{s(t+1)} \sum_r P(s(t+1),r(t)|s(t),a(t))[r(t) + \gamma E_\pi[G(t)|S(t) = s]] \tag{2.7}$$

$$V_\pi(s) = \sum_a \pi(a(t)|s(t)) \sum_{s(t+1),r} P(s(t+1),r(t)|s(t),a(t))[r(t) + \gamma V_\pi(s(t+1))] \tag{2.8}$$

Da mesma forma, pode-se definir a função valor da ação, que estima o retorno esperado ao tomar uma ação a (divergente à politica π) e após, então, passar a seguir a política π. Isto é

$$Q_\pi(s(t), a(t)) = E_\pi\left[G(t)|S(t) = s, A(t) = a\right] \tag{2.9}$$

Doravante $s(t)$ será denotado por simplesmente s e $s(t+1)$ por s'. O mesmo se aplica às variáveis a e r.

2.5 *Programação Dinâmica*

O termo *Programação Dinâmica* (DP) se refere a um conjunto de métodos de RL que utiliza poder computacional para caracterizar um MDP. Por essa razão, os algoritmos clássicos de DP têm utilidade limitada a sistemas mais simples devido ao alto custo computacional (SUTTON; BARTO, 2018). Por exemplo, o DP convencional não pode ser aplicado à sistemas que tenham estados variando continuamente em um intervalo.

Para contornar o problema, pode-se assumir que o ambiente é um MDP finito. Ou seja, que existem conjuntos finitos de estados, ações e recompensas.Para isso é possível discretizar estados e ações.

Com o ambiente discretizado, utiliza-se uma estimativa da função valor para guiar a tomada de decisão. Nos próximos tópicos serão descritos métodos de DP usados para realizar tal estimativa usando como base a equação de Bellman. Todos os métodos a serem apresentados partem da ideia do *Generalized Policy Iteration*, que será detalhado na sequência.

2.5.1 Avaliação de Política

O objetivo desse método é avaliar uma política π. A ideia é determinar o retorno que será será obtido em cada estado seguindo-se a política π. O Algoritmo 1 descreve como isso pode ser realizado. Nesse algoritmo recebe-se uma política π, um erro máximo permitido entre atualizações θ e um conjunto com todos os estados possíveis $\mathcal{S}$. Inicializam-se os valores de $V_\pi(s)$, $\forall s \in \mathcal{S}$, aleatoriamente. Então, tais valores são atualizados com base no modelo do sistema até que a o incremento de atualização seja menor do que o θ previamente estipulado.

Algoritmo 1: Avaliação de Política.

Política π a ser avaliada, θ erro aceitável entre iterações, $\mathcal{S}$, conjunto de estados possíveis **Saída:** $V_\pi(s)$
Inicializar $V_\pi(s)$ para todos os estados possíveis com exceção dos estados terminais, nos quais se assume $V_\pi(s) = 0$
while $\Delta > \theta$ **do**
  while $s \in \mathcal{S}$ **do**
    $v \leftarrow V_\pi(s)$
    $V_\pi(s) \leftarrow \sum \pi(a|s) \sum P(s',r|s,a)[r + \gamma v(s')]$
    $\Delta \leftarrow \max(\Delta, |v - V_\pi(s))$
  end
end

No Algoritmo 1, em todos os estados, atualiza-se a estimativa da função valor com base na política π e no modelo conhecido do ambiente. Isso é repetido até que a diferença entre a função valor para esse estado antes e após a iteração seja menor que um θ definido pelo usuário. Essa repetição é necessária uma vez que assume-se um ambiente estocástico, ou seja, para um mesmo conjunto estado-ação pode-se ter diferentes $s(t+1)$ e por consequência diferentes recompensas. Tem-se garantia que tal algoritmo converge uma vez que $\gamma < 1$ e assumindo-se que o número de estados e ações seja finito (KAELBLING; LITTMAN; MOORE, 1996).

2.5.2 Aprimoramento de Política

De posse da função valor de uma política π, pode-se prosseguir para o aprimoramento da mesma. Isto é, é encontrar uma política modificada π', a partir de $V_\pi(s)$, que resulte em um retorno maior ou igual ao obtido com π. Para cada estado s, o algoritmo de Aprimoramento de política determina qual a melhor ação a tomar (ou seja, determina uma nova política π'), considerando que as ações seguintes serão tomadas de acordo com a política original π. Com esse propósito, para cada $s(t) \in \mathcal{S}$, utiliza-se a equação de Bellman para testar todas as ações possíveis (o conjunto de ações possíveis será denotado por $\mathcal{A}$) e definir se alguma ação diferente da previamente estabelecida pela política π oferece um retorno maior. Caso isso seja verdade, modifica-se a política aprimorando-a.

Algoritmo 2: Aprimoramento de Política (PELLEGRINI; WAINER, 2007).

Um MDP conhecido, $\mathcal{S}$ um conjunto de estados, $\mathcal{A}$ um conjunto de ações, $V_\pi(s)$ **Saída:** $\pi'(s)$

for *each* $s \in \mathcal{S}$ **do**

  while $a \in \mathcal{A}$ **do**

    Executar a ação $a$recebendo do ambiente s' e r

    $Q_\pi(s,a) \leftarrow \sum_{s',r} P(s',r|s,a)[r + \gamma V_\pi(s')]$

  end

  $\pi'(s) \leftarrow argmax_a Q_\pi(s,a)$

end

Devolva π'

Observando o Algoritmo 2, entende-se que esse método depende de $V_\pi(s)$ que é gerado por π. Logo, trata-se de um método dito *on-policy*, que são métodos que utilizam da política π tanto para avaliar $V_\pi(s)$ quanto para aprimorar a própria política (SUTTON; BARTO, 2018). Na sequência, descrever-se-á como os Algoritmos 1 e 2 podem ser combinados para gerar a política ótima. Por definição, uma política π é melhor que uma política π' caso $\pi \geq \pi' \iff \forall\ s \in \mathcal{S}\ V_\pi(s) \geq V'_\pi(s)$. Caso exista uma política π que que satisfaça a condição anterior para todas as outras políticas possíveis, trata-se da política ótima.

2.5.3 *Generalized Policy Interation*

Utilizando o que foi apresentado nas Seções 2.5.1 e 2.5.2 tem-se uma metodologia base que descreve o funcionamento dos algoritmos de DP: a Iteração Geral de Política (*Generalized Policy Iteration, GPI*). A ideia fundamental é juntar os métodos de avaliação e de aprimoramento de políticas descritos anteriormente. Especificamente, primeiro avalia-se a política π adotada, gerando-se $V_\pi(s)$. Então, seguindo a política ε-*greedy*, exploram-se novas ações em cada estado. Como resultado, é possível melhorar a política conhecida

seguindo o Algoritmo 2. Esse processo é repetido até que não seja mais possível melhorar a ação em cada estado. Como resultado, pode-se afirmar que obteve-se a política ótima para a tarefa em questão (SUTTON; BARTO, 2018).

Uma dificuldade em implementar o GPI é a necessidade de se conhecer totalmente o ambiente. Isto é, para aplicar o Algoritmo 1, deve-se conhecer $P(s', r|s,a)$, $\forall s \in \mathcal{S}$ e $\forall a \in \mathcal{A}$ em que $P(s', r|s, a)$ denota o conjunto de probabilidades que descrevem a dinâmica do ambiente. Para contornar esse limitante, pode-se utilizar experiências adquiridas interagindo com o ambiente. Isto é, aplicando-se uma ação e recebendo-se um novo estado s' e uma recompensa r. Na sequência, descreve-se como isso pode ser feito.

2.6 SARSA

Seguindo a abordagem de GPI, a primeira técnica de RL que será tratada nesse trabalho é o SARSA. O método recebe esse nome por utilizar todos os elementos de uma experiência no aprendizado. Isto é, todos os elementos do conjunto $\{s, a, r, s', a'\}$ são utilizados para cálculo de uma estimativa para $Q_\pi(s, a)$. Como resultado, tem-se um do acrônimo da seguinte frase em inglês: *State-Action-Reward-State-Action* (SARSA). Comparada ao GPI, no SARSA estimam-se diretamente os valores de $Q_\pi(s,a)$.

No SARSA (descrito no Algoritmo 3), recebe-se uma política π (que pode ser a ϵ-greedy) e inicializa-se o estado aleatoriamente. Na sequência, escolhe-se e executa-se uma ação no ambiente de acordo com tal política. Então, o ambiente devolve o novo estado s' e a recompensa r. Nesse novo estado, define-se a ação a' de acordo com π. Com base nesses valores uma nova estimativa para $Q_\pi(s,a)$, é calculada. Esse processo é repetido até que se tenha chegado a um estado terminal $s \in \mathcal{T}$ onde o episódio é encerrado(note que $\mathcal{T}$ aponta o conjunto de estados terminais). Um novo episódio é começado por $\mathcal{N}$ vezes até que todos os $Q_\pi(s,a)$, $\forall s \in \mathcal{S}$ $\forall a \in \mathcal{A}$, tenham convergido.

O SARSA utiliza da equação (2.10) para atualizar o valor de $Q_\pi(s,a)$. Para estados s' terminais $Q_\pi(s',a')$ é definido como 0. A equação $r + \gamma Q_\pi(s',a')$ é familiar e denota uma das notações de $G(t)$ definido em 2.3. Subtraindo $Q_\pi(s,a)$ obtém-se o erro de estimativa $[r + \gamma Q_\pi(s',a') - Q_\pi(s,a)]$. Dessa forma, atualiza-se $Q_\pi(s,a)$ através do valor anterior $Q_\pi(s,a)$ mais um erro de estimativa que diminui a cada iteração. Utiliza-se também de um fator α, que pode ser visto como o tamanho do passo, para ponderar o quão sensível ao erro de estimativa será a atualização do valor de $Q_\pi(s,a)$.

$$Q_\pi(s,a) \leftarrow Q_\pi(s,a) + \alpha[r + \gamma Q_\pi(s',a') - Q_\pi(s,a)] \tag{2.10}$$

A utilização da política π, tanto para definição da ação a dado s quanto na estimativa de $Q_\pi(s,a)$, fazem com que o SARSA seja um método *on-policy*, ou seja, dependente

Algoritmo 3: SARSA.

Entrada: Política π, $\mathcal{S}$ conjunto de estados possíveis, $\mathcal{T}$ conjunto de estados terminais, $\mathcal{N}$ Número de episódios
Saída: $Q_\pi(s,a)$
Iniciar aleatoriamente os valores de $Q_\pi(s,a)$, $\forall s \in \mathcal{S}$ e $\forall a \in \mathcal{A}$
$n \leftarrow 1$
while $n < \mathcal{N}$ **do**
  s $\leftarrow random(s) \in \mathcal{S}$
  $a \leftarrow \pi(s)$
  while $s \notin \mathcal{T}$ **do**
    Tomar a ação a e receber do ambiente o novo estado s' e a recompensa r
    $r \leftarrow r(t)$
    $s' \leftarrow s(t+1)$
    $a' \leftarrow \pi(s')$
    $Q_\pi(s,a) \leftarrow Q_\pi(s,a) + \alpha[r + \gamma Q_\pi(s',a') - Q_\pi(s,a)]$
    $s \leftarrow s'$
    $a \leftarrow a'$
  end
  $n \leftarrow n+1$
end

de política. Logo, uma política adequada, é essencial para o bom funcionamento do algorítimo. Para políticas ε-greedy, a convergência dos valores de $Q_\pi(s,a)$ é garantida após um número suficientemente elevado de iterações (SUTTON; BARTO, 2018).

Seguindo o Algoritmo 3, será possível estimar os valores de $Q_\pi(s,a)$ $\forall s \in \mathcal{S}$ e $\forall a \in \mathcal{A}$. Logo, utilizando tais estimativas, será possível escolher a melhor ação em um estado visando maximizar o retorno. Ou seja, ter-se-á caracterizado a política ótima de acordo com as recompensas fornecidas.

2.6.1 *Q-Learning*

Ao contrário do SARSA, no *Q-Learning*, realiza-se o aprendizado independentemente da política utilizada. Portanto trata-se de um método *Off-Policy* (SUTTON; BARTO, 2018).

A implementação do *Q-learning* (descrita no Algoritmo 4) é similar à do SARSA, com a diferença na forma de atualizar o valor de $Q_\pi(s,a)$. No algoritmo abaixo, em cada episódio de um total de $\mathcal{N}$, inicia-se em um estado aleatório s. A partir desse estado, a política π é usada para escolher define a ação a ser tomada a. Tendo sido tomada a ação, o ambiente retorna um estado s' e uma recompensa r. A partir desses valores atualiza-se $Q_\pi(s,a)$. Por fim, o s' é tomado como o estado atual e repete-se o ciclo.

Pode-se perceber que, a diferença advém da substituição de $Q_\pi(s',a')$ por $\max_{a\in\mathcal{A}} Q_\pi(s',a)$. Isto é, ao invés de seguir a política π, no *Q-Learning* escolhe-se a ação que ma-

Algoritmo 4: *Q-Learning.*

Política π, $\mathcal{S}$ conjunto de estados possíveis, $\mathcal{T}$ conjunto de estados terminais, $\mathcal{N}$ Número de episódios **Saída:** $Q_\pi(s,a)$

Iniciar aleatoriamente os valores de $Q_\pi(s,a)$, $\forall s \in \mathcal{S}$ e $\forall a \in \mathcal{A}$

$n \leftarrow 1$

while $n < \mathcal{N}$ **do**

  s $\leftarrow random(s) \in \mathcal{S}$

  while $s \notin \mathcal{T}$ **do**

    $a \leftarrow \pi(s)$

    $s' \leftarrow s(t+1)$

    $r \leftarrow r(t)$

    $Q_\pi(s,a) \leftarrow Q_\pi(s,a) + \alpha[r + \gamma \max_{a \in \mathcal{A}} Q_\pi(s',a) - Q_\pi(s,a)]$

    $s \leftarrow s'$

  end

  $n \leftarrow n+1$

end

ximiza o retorno no estado seguinte. Dessa forma, os algoritmos de *Q-learning* funcionam aproximando $Q_\pi(s,a)$ da função valor ótima. Como resultado o *Q-Learning* converge mais rapidamente que o SARSA para a política ótima. Isso será demonstrado nos resultados.

2.6.2 *Deep Q-Learning*

Deep Q-learning (DQN -*Deep Q-netowrk*) é um método de RL baseado em *Q-learning*. Logo, também utiliza as experiências para estimar $Q_\pi(s,a)$. A diferença é que no DQN se emprega uma Rede Neural Artificial (RNA) para estimar $Q_\pi(s,a)$. Logo, torna-se possível considerar que os estados variam continuamente em um certo intervalo. Além disso, é possível considerar estados em diferentes formas. Por exemplo, os estados poder ir desde leitura de sensores até *frames* de um jogo, como foi feito em Mnih et al. (2013). O desafio consiste em treinar a rede neural para estimar de maneira adequada $Q_\pi(s,a)$, $\forall s \in \mathcal{S}$ e $\forall a \in \mathcal{A}$. Como proposto em Mnih et al. (2013), isso pode ser feito como detalhado no Algoritmo 6.

A implementação do *Deep Q-learning* é similar à do *Q-Learning*, com a diferença na forma de estimar o valor de $Q_\pi(s,a)$. No algoritmo abaixo, em um número $\mathcal{N}$ de episódios, inicia-se em um estado aleatório s. A partir desse estado, a política π é usada para escolher define a ação a ser tomada a. Tendo sido tomada a ação, o ambiente retorna s'e r. A partir desses valores atualiza-se $Q_\pi(s,a)$. A diferença é que cada transição é armazenada na memória de repetição $\mathcal{D}$, que será utilizada para treinar a RNA empregada para estimar $Q_\pi(s,a)$. Por questões computacionais, para realizar o treinamento não utiliza-se de todas as transições armazenadas em $\mathcal{D}$, escolhem-se de forma aleatórias $\mathcal{M}$ transições por passo para serem utilizadas no treinamento.

Algoritmo 5: *Deep Q-Learning.*

Entrada: Política π, $\mathcal{S}$ conjunto de estados possíveis, $\mathcal{T}$ conjunto estados terminais, $\mathcal{N}$ Número de episódios, $\mathcal{D}$ memória de repetição, $\mathcal{M}$ Capacidade da memória de repetição

Iniciar aleatoriamente os valores de $Q_\pi(s,a)$, $\forall s \in \mathcal{S}$ e $\forall a \in \mathcal{A}$
$n \leftarrow 1$
while $n < \mathcal{N}$ **do**
  while $s \notin T$ **do**
    $a \leftarrow \pi(s)$
    Executar a ação a e obter a recompensa r e o próximo estado s'
    Armazenar a transição (s,a,r,s') em $\mathcal{D}$
    Sortear aleatoriamente um subconjunto de $\mathcal{D}$ *com um tamanho* $\mathcal{M}$
    while $j<M$ **do**
      $y_j \leftarrow Q_\pi(s,a) + \alpha[r + \gamma \max_{a \in \mathcal{A}} Q_\pi(s',a) - Q_\pi(s,a)]$
      IF $s \in \mathcal{T}$ ***Then*** $\boldsymbol{y_j} \leftarrow \boldsymbol{r}$
      $\boldsymbol{j} \leftarrow j + 1$
    end
    Realizar o treinamento da RNA para minimizar o erro $(y_j - Q_\pi(s,a))^2$ *em todo subconjunto* $\mathcal{M}$.
    $s \leftarrow s'$
  end
  $n \leftarrow n + 1$
end

2.6.3 *Double Deep Q-Learning*

Nos algoritmos *Q-learning* e *Deep Q-learning*, a política é escolher a melhor ação dado estado s, para isso, utiliza-se da presunção de que quanto maior o Q-Value melhor seria a ação no longo prazo. O problema é que por mais que haja convergência, é necessário que o modelo estime e atualize o *Q-value* a cada iteração, assim, em ambientes ruidosos não é possível mostrar que um *Q-value* maior representa necessáriamente um retorno $G(t)$ maior. O problema acima, no qual o valor da ação ótima é menor que as demais, e por consequência o agente toma ações divergentes da ação ótima, é denominado superestimação do valor da ação (HASSELT; GUEZ; SILVER, 2015).

Uma alternativa para mitigar o problema da superestimação de $Q_\pi(s,a)$ do Q-learning foi utilizar duas RNAs (HASSELT; GUEZ; SILVER, 2015) . Uma é empregada para estimar o *Q-value* para cada ação em um certo estado, e a outra seria para dentre os *Q-values* escolher a melhor ação. O motivo pelo qual essa abordagem funciona é que, por mais que existam ruídos nas duas previsões, podem-se considerar os ruídos como uma distribuição uniforme, resolvendo assim o valor da superestimação.

Algoritmo 6: *Double Deep Q-Learning.*

Entrada: Política π, $\mathcal{S}$ conjunto de estados possíveis, $\mathcal{T}$ conjunto estados terminais, $\mathcal{N}$ Número de episódios, $\mathcal{D}$ memória de repetição, $\mathcal{M}$ Capacidade da memória de repetição

Iniciar aleatoriamente os valores de $Q_\pi(s,a)$, $\forall s \in \mathcal{S}$ e $\forall a \in \mathcal{A}$
$n \leftarrow 1$
while $n < \mathcal{N}$ **do**
  while $s \notin T$ **do**
    $Q_\pi(s,a)_A \leftarrow$ *Valor da ação estimado pela RNA A*
    $a \leftarrow argmax_a Q_\pi(s,a)_A$
    Executar a ação a e obter a recompensa r e o próximo estado s'
    Armazenar a transição (s,a,r,s') em $\mathcal{D}$
    Sortear aleatoriamente um subconjunto de $\mathcal{D}$ com um tamanho $\mathcal{M}$
    while $j<M$ **do**
      $Q_\pi(s',a)_B \leftarrow$ Valor da ação estimado pela RNA B
      $y_j \leftarrow Q_\pi(s,a) + \alpha[r + \gamma \max_{a \in \mathcal{A}} Q_\pi(s',a)_B - Q_\pi(s,a)_A]$
      IF $s \in \mathcal{T}$ **Then** $\mathbf{y}_j \leftarrow \mathbf{r}$
      $\mathbf{j} \leftarrow j + 1$
    end
    Realizar o treinamento da RNA para minimizar o erro $(y_j - Q_\pi(s,a))^2$ em todo subconjunto $\mathcal{M}$.
    $s \leftarrow s'$
  end
  $n \leftarrow n + 1$
end

3 Descrição do Sistema

O sistema do carro-pêndulo é constituído por um carro de movimento retilíneo, no qual encontra-se fixada uma haste que pode rotacionar livremente (vide Figura 3). A tarefa de controle consiste em equilibrar verticalmente a haste (isto é manter o ângulo $\theta \approx 0$) e manter o carro parado o mais perto possível do centro (i.e. manter a posição $x \approx 0$). Para isso, manipula-se a força horizontal F aplicada ao carro. Neste trabalho, considerou-se o carro-pêndulo descrito em (NETTO, 2016). Esse equipamento é ilustrado na Figura 3

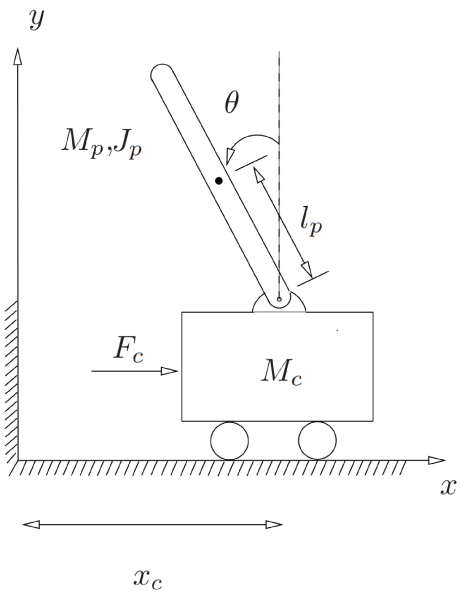

Figura 3 – Ilustração de um carro-pêndulo. (NETTO, 2016)

Nesse sistema, as variáveis de estado são

- A posição linear, que determina a distância que o carro se encontra do centro da bancada, denotada por x;
- A velocidade linear do carro, como derivada do item anterior;
- A posição angular da haste, o ângulo que ela se encontra com o referencial na posição neutra vertical para cima, denotada por θ;
- Velocidade ângular da haste denotada por $\omega = \dot{\theta}$.

Assumiram-se os seguintes intervalos admissíveis para x e θ:

- $-1 \leq x \leq 1$;

- $-30° \leq \theta \leq 30°$

Caso, tais variáveis saiam do intervalo admissível, termina-se o episódio.

Como já mencionado, adotou-se a força como variável de controle. Para isso, desconsiderou-se a curva tensão x torque do motor, e até mesmo o torque x aceleração.

Todas as variáveis descritas acima, nos algoritmos de SARSA e *Q-Learning*, precisaram ser discretizadas. Isso acontece pois nesses métodos os valores de $Q_\pi(s,a)$, $\forall s \in \mathcal{S}$ e , $\forall a \in \mathcal{A}$, são armazenados em formato matricial. Assim, criou-se uma matriz Q de dimensões (720;720;200;5), sendo as duas primeiras dimensões relativas à posição e à velocidade angular com passos de 1° e $1°/s$, respectivamente. Isto é, para a posição θ, inicia-se em -360° (para a posição 0 do array) e indo até 360° (para a posição 719). Algo similar vale para a posição angular. Já a terceira dimensão, relativa a posição x, representando 200 passos de 1 cm de -100 cm até 100 cm. A última dimensão indica a quantidade de ações possíveis de serem tomadas. Em particular, considerou-se que a força pode assumir algum dos seguintes valores: -60, -20, 0, 20, 60 N. Note que, em cada instante de tempo discreto, o agente deve escolher uma dessas cinco ações.

Além disso, para que o agente aprenda a realizar a tarefa proposta, uma função de recompensa precisa ser definida. Tal função é detalhada na sequência.

3.1 Função Recompensa

A função recompensa foi escolhida variando de -1 a 1 nas seguintes condições: Caso a haste se mantenha entre $-2°$ e 2° do equilíbrio e caso e $|x| \leq 0{,}5$ m, o agente recebe uma recompensa numérica de +1. Para estados terminais, ou seja, se $|\theta| \geq 30°$ ou se $|x| \geq 1$ m, a recompensa é -1. Para as demais situações a recompensa é -0,25. Matematicamente,

$$r(\theta, x) = \begin{cases} 1, & |\theta| \leq 2° \quad \text{e} \quad |x| \leq 0{,}5 \text{ m} \\ -1, & |\theta| \geq 30° \quad \text{ou} \quad |x| \geq 1 \text{ m} \\ -0{,}25, & \text{outros valores} \end{cases} \tag{3.1}$$

De posse dessa função de recompensa e da discretização descrita anteriormente, já é possível testar os algoritmos de RL descritos no Capítulo 2. Tais testes foram realizados em ambiente de simulação. Particularmente no presente trabalho, as simulações foram realizadas com base no modelo não linear do carro-pêndulo. Esse modelo é apresentado na seção seguinte e pode ser encontrado em diversos trabalhos da literatura. O modelo a ser apresentado foi retirado de (NETTO, 2016).

3.2 Modelagem Matemática

Os valores e a descrição das constantes do modelo,(NETTO, 2016) encontram-se detalhadas na Tabela 1.

Tabela 1 – Tabela com a descrição e os valores das constantes utilizadas.

Constantes do Modelo	
Variável [Símbolo]	Valor
Gravidade [g]	9,81 $\frac{\text{m}}{\text{s}^2}$
Massa do pêndulo [M_p]	0,16985 kg
Massa do carro [M_c]	0,5676 kg
Comprimento da haste [l_p]	0,0475 m
Momento de inércia do pêndulo [J_p]	0,0032 kgm^2
Amortecimento rotacional [B_p]	0,002 $\frac{\text{Nsm}}{\text{rad}}$
Amortecimento translacional [B_{eq}]	5,4 $\frac{\text{Ns}}{\text{m}}$

É possível iniciar a modelagem considerando as forças horizontais agindo no carro. Em particular, considerando a força manipulada F, uma força de atrito viscoso $B_{eq}\dot{x}(t)$ e a força que a haste exerce no carro N, tem-se

$$M_c\ddot{x}(t) + N(t) + B_{eq}(t)\dot{x}(t) = F \tag{3.2}$$

Doravante a dependência com o tempo das variáveis será omitida, por claridade das equações. Considerando x como a posição do carro, l_p como comprimento do centro de massa da haste pode-se considerar a equação da primeira lei conforme abaixo

$$M_p\frac{d^2}{dt^2}(x - l_p \operatorname{sen}\theta) = N \tag{3.3}$$

Derivando (3.3) e substituindo em (3.2) tem-se:

$$(M_p + M_c)\ddot{x} + B_{eq}\dot{x} - M_p l_p \cos\theta\ddot{\theta} + M_p l_p \operatorname{sen}\theta\dot{\theta}^2 = F \tag{3.4}$$

Adicionalmente, para o movimento rotacional da haste, é possível escrever

$$(J_p + M_p l_p^2)\ddot{\theta} - M_p l_p \cos\theta\ddot{x} - M_p g l_p \operatorname{sen}\theta + B_p\dot{\theta} = 0 \tag{3.5}$$

As duas equações acima definem o comportamento da planta, sendo possível ainda isolar $\ddot{\theta}$ em (3.5) e substituir em (3.4), e são obtidas as duas equações que serão usadas para a simulação do sistema. Especificamente, (NETTO, 2016)

$$\ddot{x} = \frac{M_p l_p \cos(\theta)(M_p l_p g \operatorname{sen}\theta - B_p\dot{\theta}) - (M_p l_p^2 + J_p)(B_{eq}\dot{x} + M_p l_p \operatorname{sen}\theta\dot{\theta}^2) + (M_p l_p^2 + J_p)F}{M_p M_c l_p^2 + J_p(M_p + M_c) + M_p^2 l_p^2 \operatorname{sen}\theta} \tag{3.6}$$

$$\ddot{\theta} = \frac{M_p l_p \cos\theta\ddot{x} - B_p\dot{\theta} + M_p g l_p \operatorname{sen}\theta}{M_p l_p^2 + J_p} \tag{3.7}$$

3.2.1 Espaço de Estados

Toda a implementação e testes que serão apresentados no próximo capítulo foram desenvolvidos por simulação computacional. Para tal, foi utilizado a linguagem de programação Python, sendo que a função *odeint* da biblioteca *scipy*, cuja a documentação pode ser encontrada em <https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.odeint.html> foi empregada para integrar numericamente o modelo do sistema entre dois instantes de amostragem consecutivos. Com esse propósito, faz-se necessário a representação do modelo no espaços de estado conforme definido abaixo

$$P : \begin{cases} \dot{x} = Ax + Bu + E \\ y = Cx + Du \end{cases}$$

Nesse caso, o vetor de estados é representado por $x = [x \quad \theta \quad \dot{x} \quad \dot{\theta}]^T$ e a entrada manipulada de força será dada por u. A partir das equações (3.4) e (3.5) é possível escrever

$$\phi\dot{x} + \Gamma x + \Omega = \Psi u \tag{3.8}$$

Sendo por comparação:

$$\Phi = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & M_p + M_c & -M_p l_p \cos(x_2) \\ 0 & 0 & 0 - M_p l_p \cos(x_2) & J_p + M_p l_p^2 \end{bmatrix}, \Gamma = \begin{bmatrix} 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & B_{eq} & M_p l_p \operatorname{sen}(x_2) x_4 \\ 0 & 0 & 0 & B_p \end{bmatrix}$$

$$\Omega = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -M_p l_p g \operatorname{sen}(x_2) \end{bmatrix}, \Psi = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

Isolando $\dot{x}$

$$\dot{x} = -\Phi^{-1}\Gamma x + \Phi^{-1}\Psi u - \Phi^{-1}\Omega \tag{3.9}$$

Por comparação, obtêm-se $A = -\Phi^{-1}\Gamma$, $B = \Phi^{-1}\Psi$ e $E = \Phi^{-1}\Omega$. Matematicamente,

$$A = \frac{1}{J_{Tn}} \begin{bmatrix} 0 & 0 & \frac{1}{J_{Tn}} & 0 \\ 0 & 0 & 0 & \frac{1}{J_{Tn}} \\ 0 & 0 & -B_{eq}(M_p l_p^2 + J_p) & -B_{eq} M_p l_p \cos(x_2) - M_p l_p x_4 \operatorname{sen}(x_2)(M_p l_p^2 + J_p) \\ 0 & 0 & -B_{eq} M_p l_p \cos(x_2) & -B_p(M_c + M_p) - M_p^2 l_p^2 \cos(x_2) \operatorname{sen}(x_2) x_4 \end{bmatrix}$$

$$B = \frac{1}{J_{Tn}} \begin{bmatrix} 0 \\ 0 \\ M_p l_p^2 + J_p \\ M_p l_p \cos(x_2) \end{bmatrix}, \quad E = \frac{1}{J_{Tn}} \begin{bmatrix} 0 \\ 0 \\ M_p^2 l_p^2 g \operatorname{sen}(x_2) \cos(x_2) \\ M_p l_p g \operatorname{sen}(x_2)(M_p + M_c) \end{bmatrix}$$

sendo $J_{tn} = -M_p^2 l_p^2 (\cos(x_2))^2 + (M_p + M_c)(M_p l_p^2 + J_p)$. Os resultados da aplicação dos algoritmos descritos no Capítulo 2 no controle do carro-pêndulo são mostrados no capítulo a seguir.

4 Resultados

Foi criado e utilizado um ambiente de simulação em *Python* usando o *Google Colab*. Nesse ambiente, inicia-se o sistema no estado $x(0) = \dot{x}(0) = \ddot{x}(0) = \dot{\theta}(0) = 0$ e $\theta(0) \in [-6°,\ 6°]$. A partir desse estado inicial, o agente decide uma ação, de acordo com sua política. Essa ação é aplicada ao ambiente gerando um novo estado e uma recompensa. A geração de um novo estado é realizada integrando-se numericamente o modelo do sistema empregando a função *odeint* da biblioteca *scipy.integrate* considerando um tempo de amostragem de 10 ms. Então, o ciclo se repete até que um estado seja terminal (i.e. até que x ou θ saia de um dos conjuntos admissíveis) ou que o tempo de simulação, definido como 10 s, acabe.

Para avaliar a influência dos parâmetros fator de ganância ϵ, fator de desconto γ e fator de aprendizagem α no aprendizado, a média de retorno por episódio será adotada como métrica. Esse indicador avalia a média do somatório das recompensas obtidas em um episódio. Logo, dado o sistema de recompensas desse trabalho, quanto menos o agente demorar para manter θ entre $\pm 2°$ e $|x| < 0{,}5$ m, maior será tal média. Ou seja, a métrica adotada é diretamente proporcional ao aprendizado do agente. Para os testes, será variado um dos parâmetros de aprendizagem por vez, e analisada a influência dessa alteração na média do retorno por episódio. Isso será realizado para todos os algoritmos descritos no Capítulo 2.

4.1 SARSA

Para a simulação com SARSA, foi realizado o treinamento do agente por 10000 episódios. Os parâmetros de ajuste foram variados no seguinte intervalo:

- $\epsilon \in [0{,}1,\ 0{,}95]$.
- $\gamma \in [0{,}1,\ 0{,}95]$.
- $\alpha \in [0{,}1,\ 0{,}95]$.

4.1.1 Fator de Ganância ϵ

O fator ϵ é empregado para ponderar entre o aproveitamento do conhecimento já adquirido e a exploração de novas possibilidades. Quanto maior for essa variável maior a chance de aproveitar o conhecimento adquirido para a escolha das ações. Por outro lado, quanto menor for ϵ, mais se exploram novas ações aleatoriamente. Para essa variável foram testados os valores 0,1, 0,3125 e 0,525, 0,7375, 0,95, os demais valores foram fixados

em $\gamma = 0{,}9$ e $\alpha = 0{,}9$. Para cada um desses valores foi repetida a simulação três vezes. A Figura 4 mostra evolução da média móvel do retorno considerando uma janela de de 200 episódios.

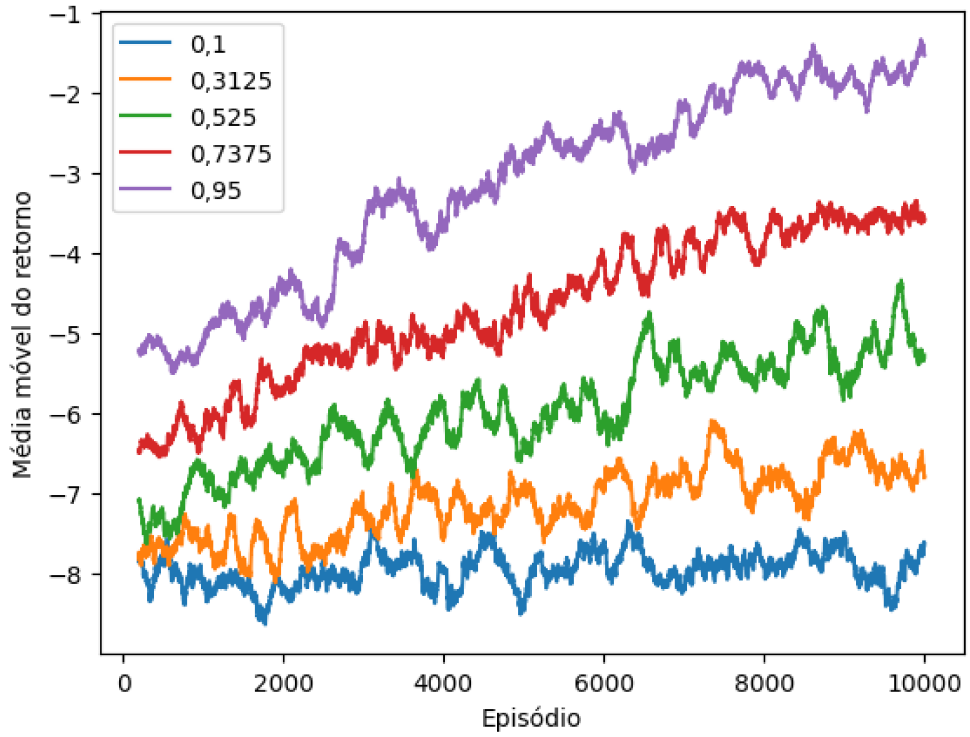

Figura 4 – Evolução da média móvel do retorno por episódio.

Percebe-se que o aprendizado é realizado mais rapidamente quanto maior o valor de ϵ. Em particular, nota-se que, com $\epsilon = 0{,}95$, obtiveram-se os melhores resultados. Isso pode ser explicado porque a probabilidade do agente seguir uma ação aleatória, que pode levar a estados terminais, é menor. Isso interfere também no desvio padrão mostrado na Figura 5 nota-se que ao diminuir a aleatoriedade (i.e. aumentar ϵ) reduz o desvio padrão.

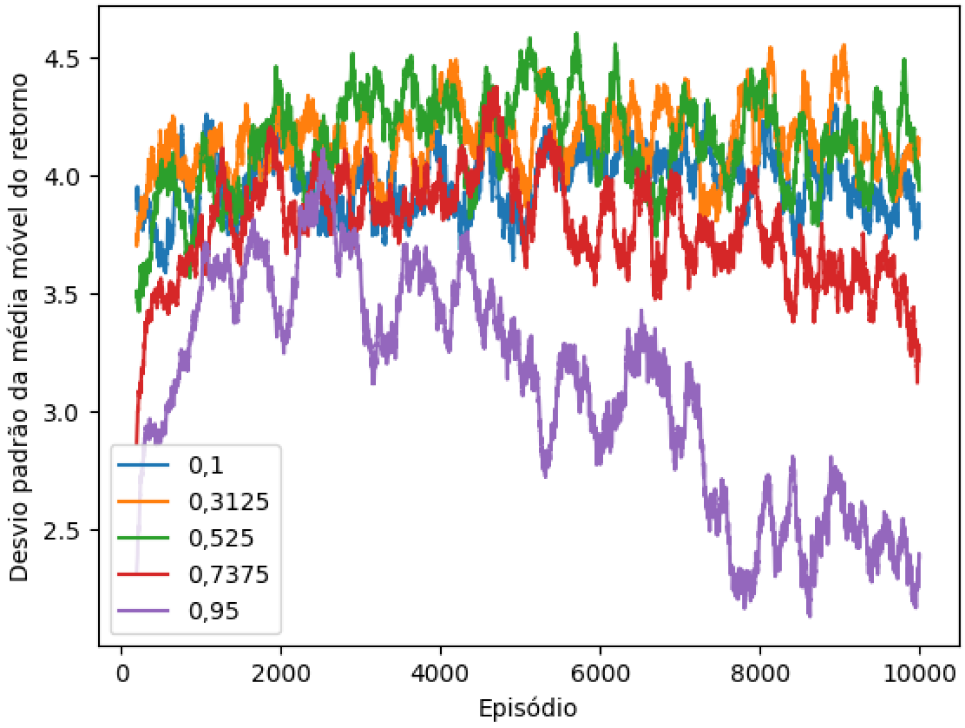

Figura 5 – Evolução dos desvios padrão entre valores de ϵ.

Deixar o ϵ fixo ao longo do aprendizado pode não ser a melhor opção. Isso ocorre porque, principalmente em episódios iniciais, é interessante que o agente explore novas ações para melhores ações. Assim, iniciar o experimento com um $\epsilon \approx 0$ e incrementá-lo até que $\epsilon \approx 1$ pode ser vantajoso. Em particular, pode-se evoluir o valor de ϵ do seguinte modo:

$$\epsilon = 1 - e^{\beta N} \tag{4.1}$$

sendo $\beta > 0$ um parâmetro de ajuste e N o número do episódio.

Para $\beta = -0{,}0005$ obtiveram-se os resultados das Figuras 6 e 8, em que "EXP" indica a curva resultante de se adotar ϵ evoluindo como (4.1). Pode-se notar que a longo prazo um ϵ variável foi mais vantajoso.

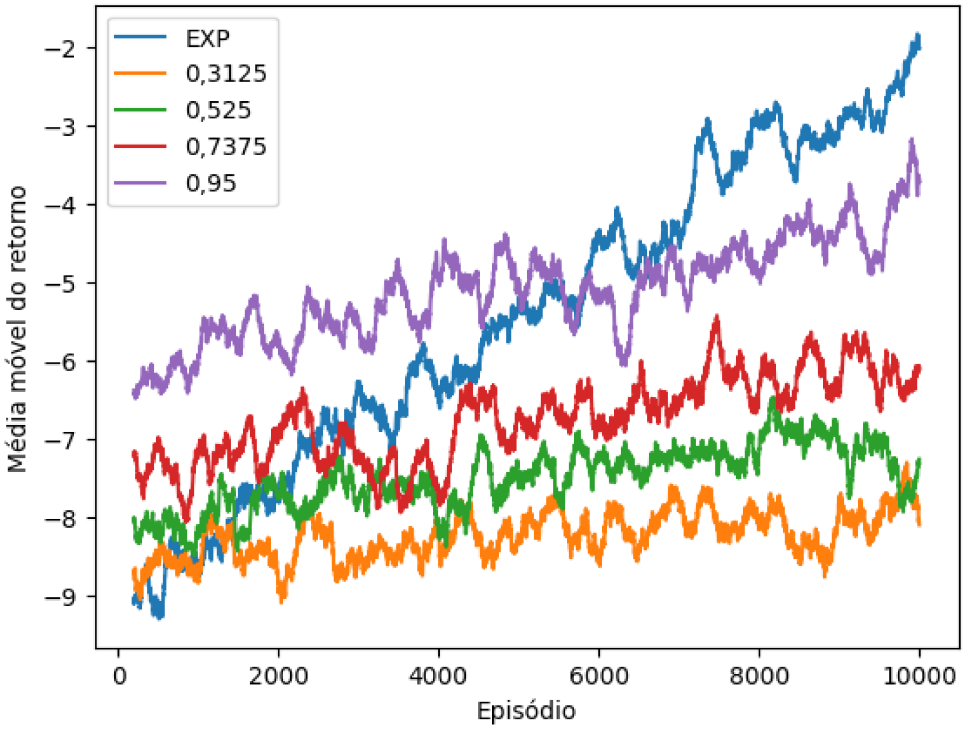

Figura 6 – Comparação do retorno do episódio com ϵ variável.

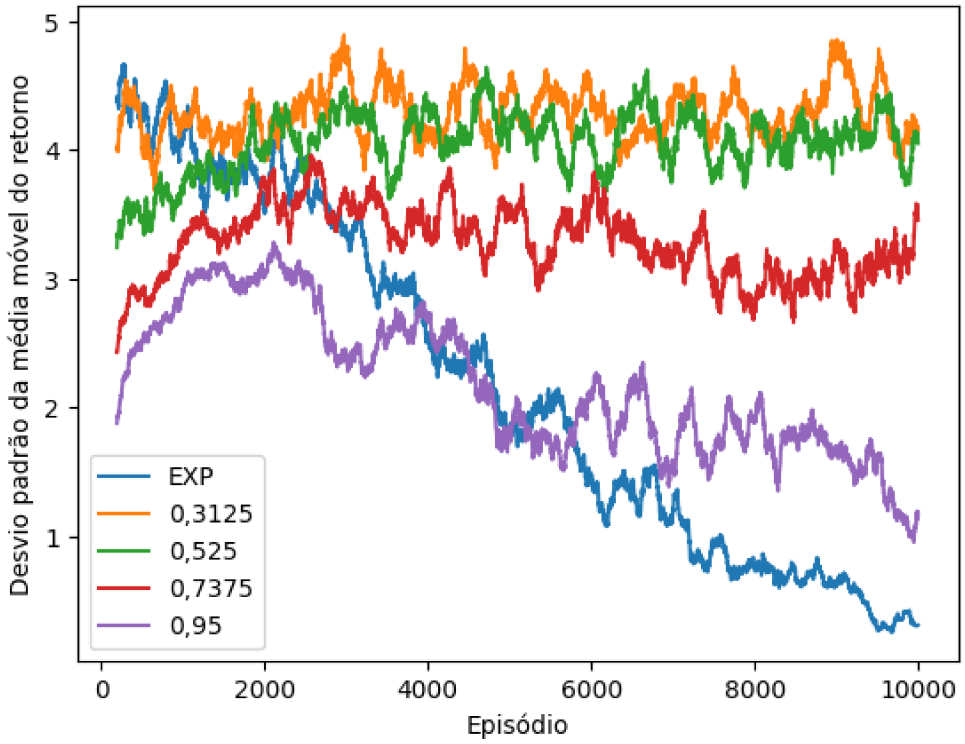

Figura 7 – Comparação do desvio padrão do retorno por episódio com ϵ variável.

Ainda considerando ϵ variável, pode-se variar o fator β e verificar como isso afeta o aprendizado. A Figura 8 apresenta os resultados para $\beta \in \{-0{,}0005, -0{,}0006, -$

0,0007, −0,0008, −0,0009}. Constata-se que, nesse caso em particular, para $\beta = -0{,}0009$ o aprendizado ocorre mais rapidamente.

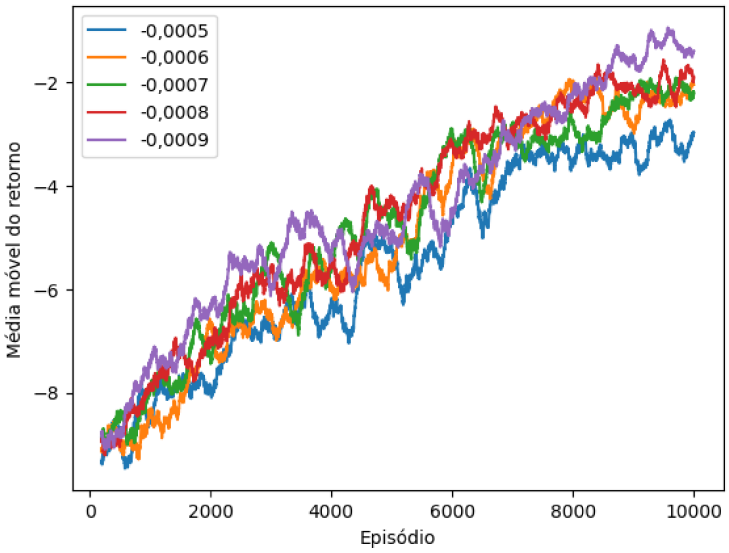

Figura 8 – Comparação do desempenho do sistema para β variável.

4.1.2 Fator de aprendizagem α

O parâmetro α é o fator de aprendizagem do agente. Esse fator determina o quanto a atualização do *Q-value* é sensível ao erro de estimativa. Nesse teste, fixaram-se $\epsilon = 0{,}85$ e $\gamma = 0{,}9$. Além disso, considerou-se $\alpha \in \{0{,}1,\ 0{,}3125,\ 0{,}525,\ 0{,}7375,\ 0{,}95\}$. Os resultados são mostrados nas Figuras 9 e 10.

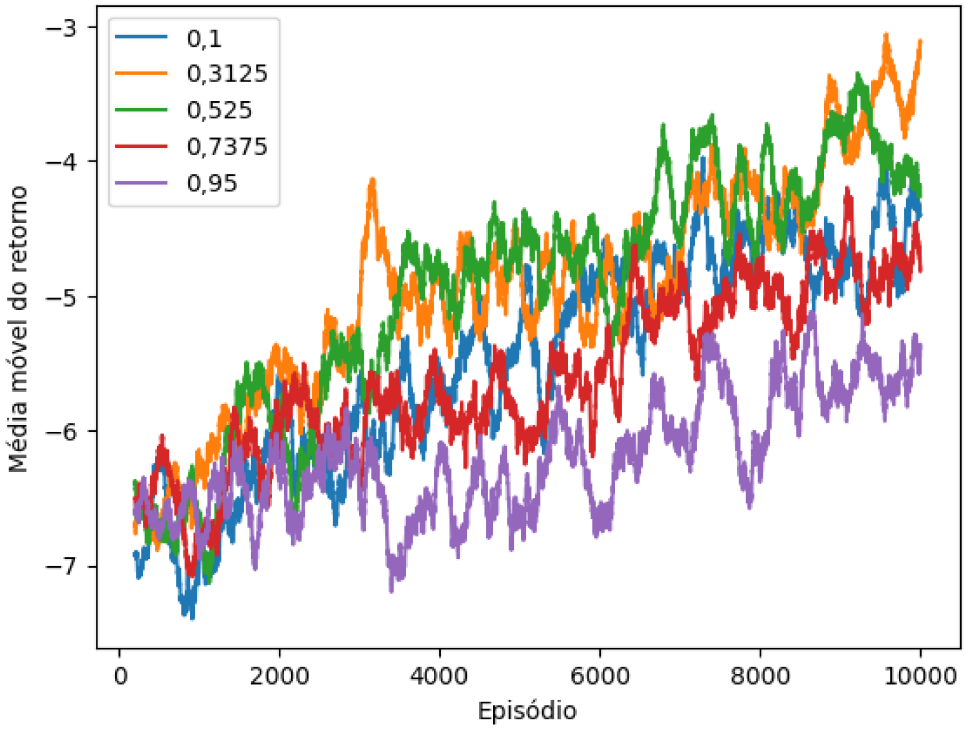

Figura 9 – Comparativo da média do retorno por episódio para diferentes valores de α.

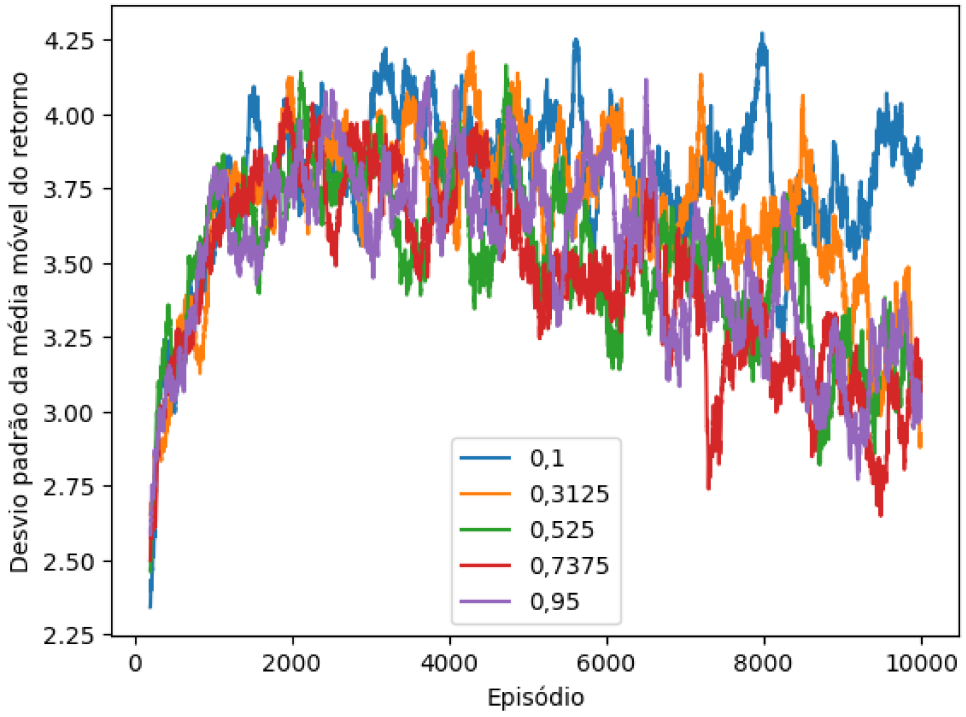

Figura 10 – Comparativo do desvios padrão do retorno por episódio para diferentes valores de α.

Pode-se perceber que um valor mediano está associado com os melhores resulta-

dos (maior média do retorno com menores desvios padrão). Para valores abaixo de 0,4 apresentaram decréscimo na velocidade de aprendizado. Nesse cenário em particulara, verifica-se que o agente aprende mais rapidamente com $\alpha = 0{,}3125$.

4.1.3 Fator de desconto γ

O fator de desconto γ permite ponderar recompensas de curto e longo prazo. Um fator de desconto 0 significa que o agente se preocupará somente com recompensas imediatas. Por outro lado, um fator de desconto 1 permite com que os retornos de longo e curto prazo tenham o mesmo peso. Ao avaliar a média de retorno por episódio, utilizaram-se os valores para γ iguais a 0,1, 0,525, 0,95, para as outras constantes, adotaram-se $\epsilon = 0{,}85$ e $\alpha = 0{,}9$. Os resultados são mostrados na Figura 11. Pode-se verificar que para $\gamma = 0{,}1$ obteve-se um resultado melhor ao fim dos 10000 episódios, se destacando de todos os outros valores ao final do gráfico. Por outro lado, esse valor está associado a uma maior variância (Figura 12).

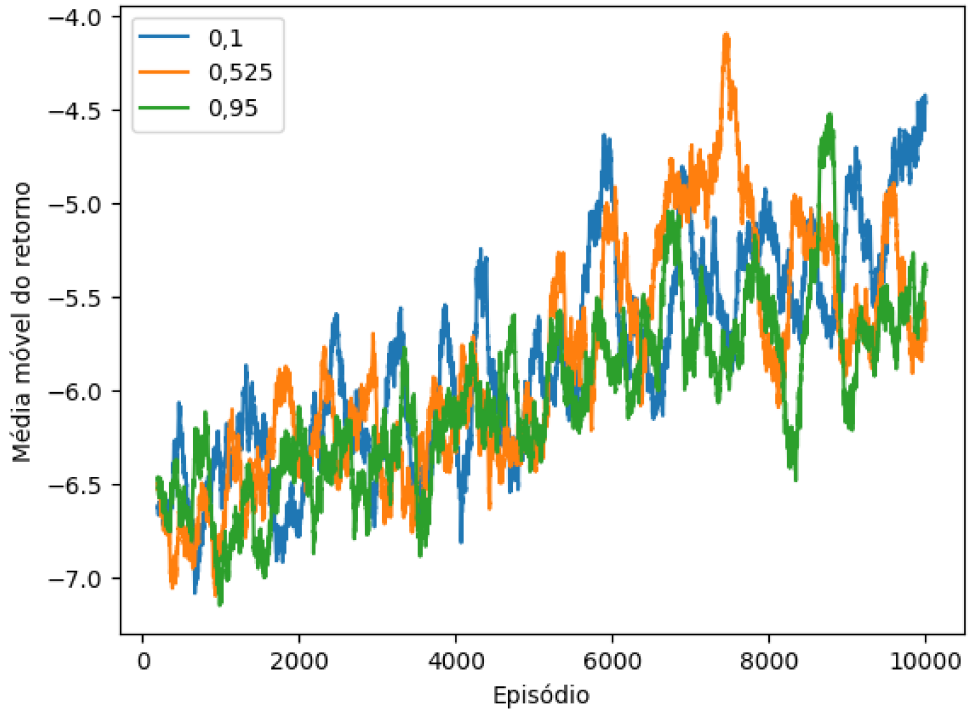

Figura 11 – Comparativo da média do retorno por episódio para diferentes valores de γ.

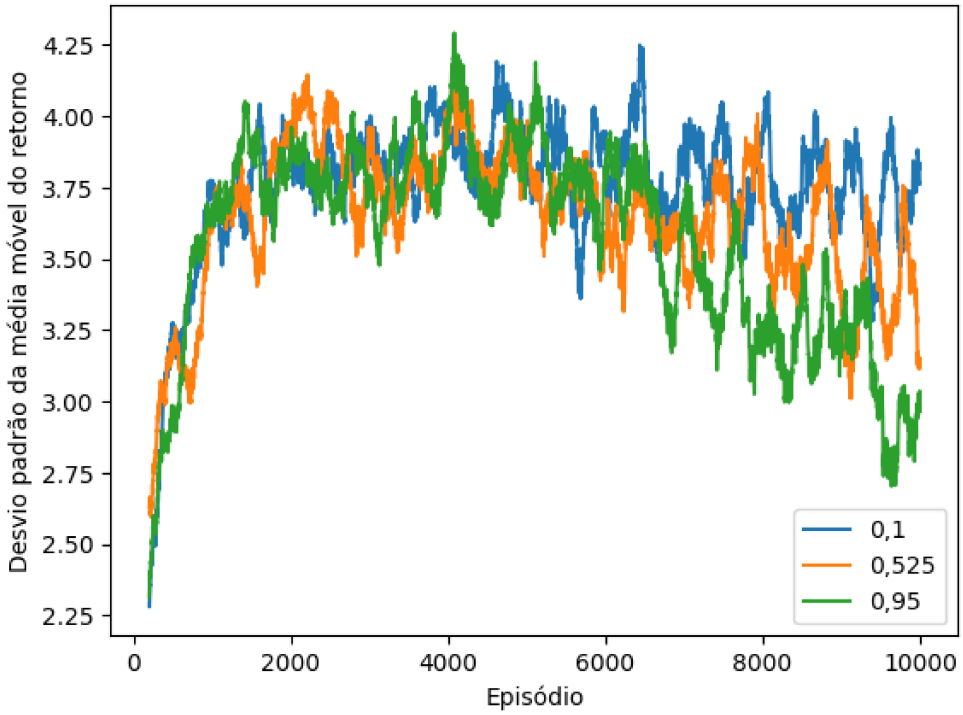

Figura 12 – Comparativo do desvios padrão do retorno por episódio para diferentes valores de γ.

Por fim, na Figura 13, mostra-se o sistema equilibrando o pêndulo a partir de um ângulo inicial. Pode-se notar que a haste é levada para a posição invertida por meio da aplicação de forças ao carro, permanecendo assim até que a posição seja maior que 0,5. Nesse caso, o agente causa um deslocamento no veículo para recuperação da recompensa. Porém, a simulação é interrompida pois o ângulo θ adquire uma posição não aceitável. Para os gráficos temporais, será denotado a velocidade do carro, $\dot{x}$, como x'.

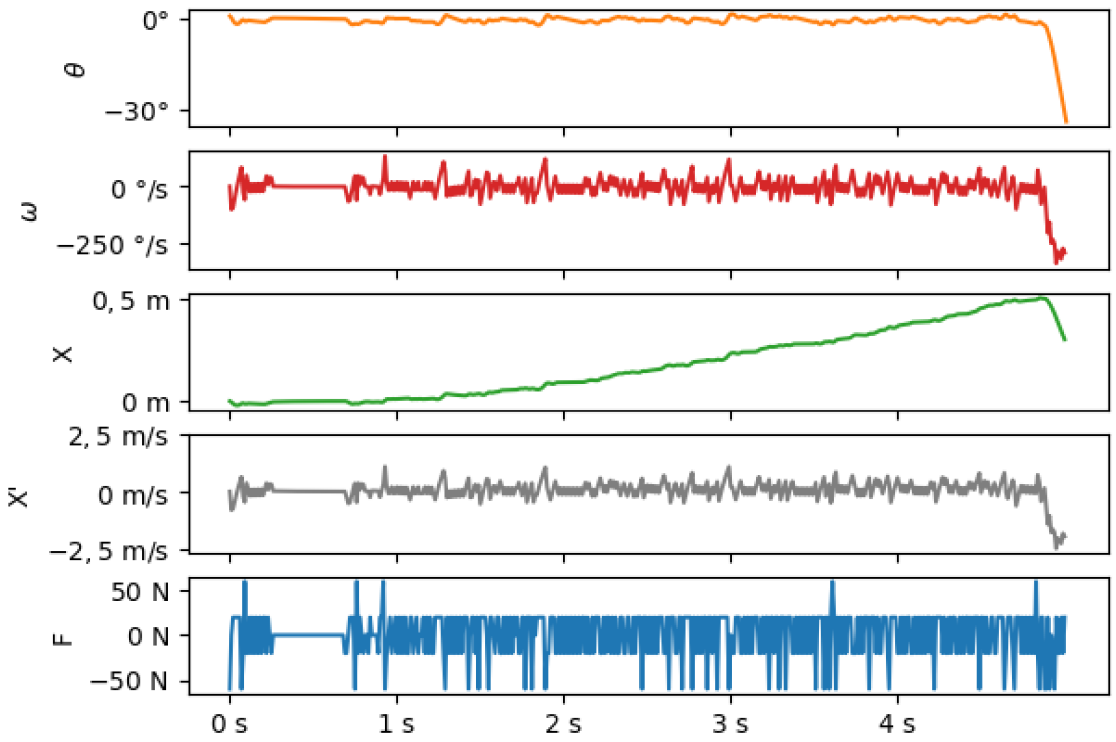

Figura 13 – Evolução dos estados (posições e velocidades angulares e lineares) e da ação de controle (força) em um teste após o aprendizado utilizando SARSA.

4.2 *Q-Learning*

Para algoritmo de *Q-Learning* foram realizados testes similares àqueles que foram feitos como SARSA.

4.2.1 Fator de Ganância ϵ

Os valores adotados de ϵ foram iguais aos adotados para o SARSA, os resultados (média e desvio padrão) são apresentados nas Figuras 14 e 15 respectivamente. Além disso, também foram gerados resultados com ϵ variando conforme (4.1) considerando $\beta = -0{,}0005$. Os resultados (média e desvio padrão) encontram-se nas Figuras 16 e 17.

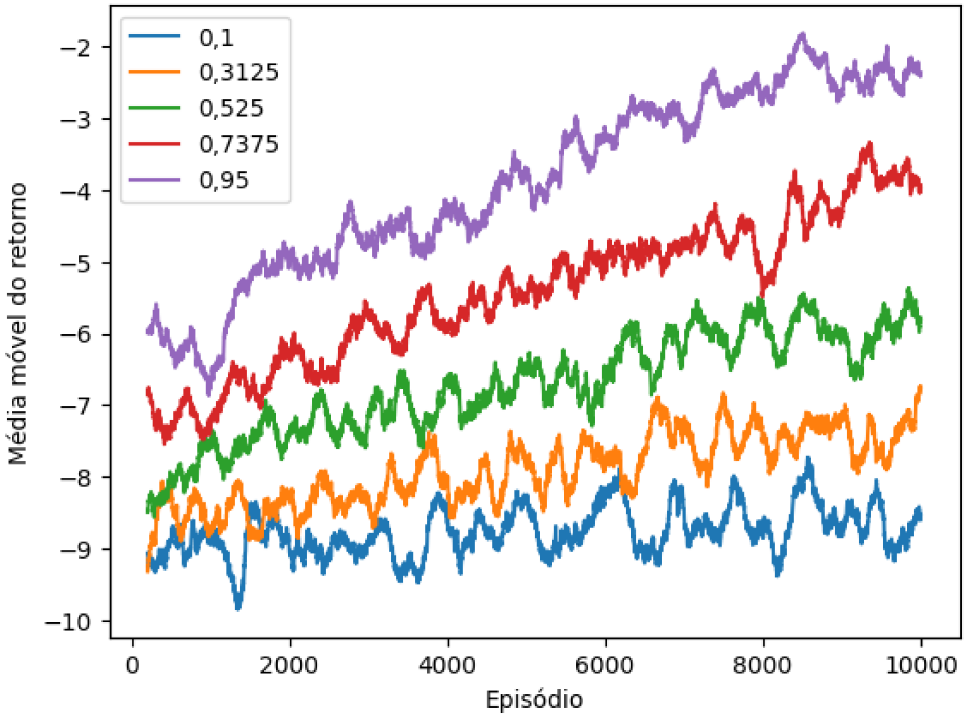

Figura 14 – Evolução da média móvel do retorno por episódio.

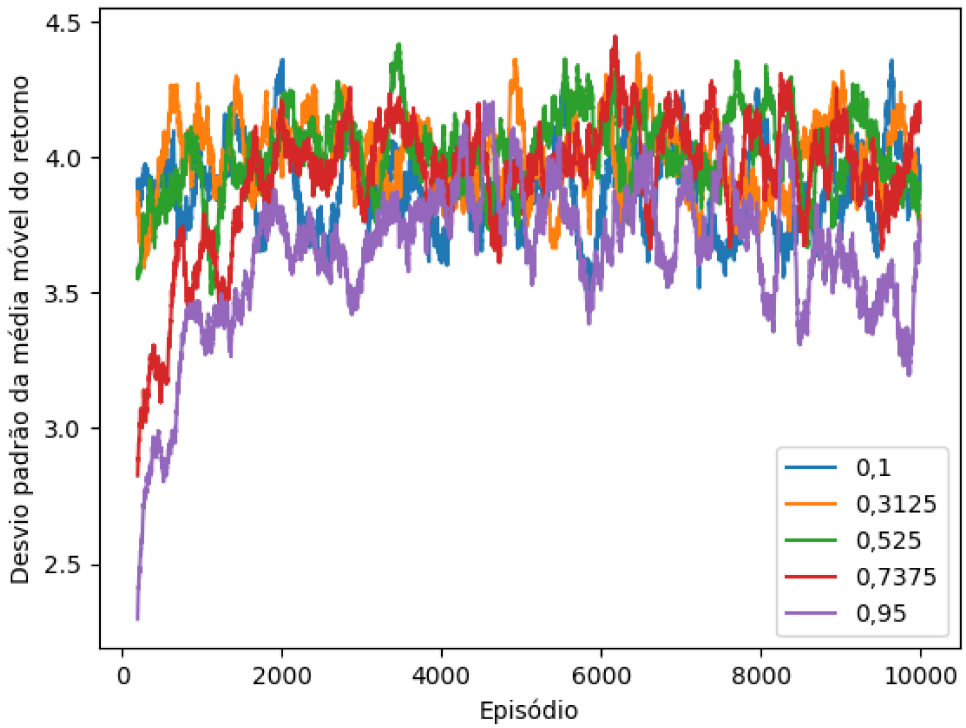

Figura 15 – Evolução dos desvios padrão entre diversos valores de ϵ.

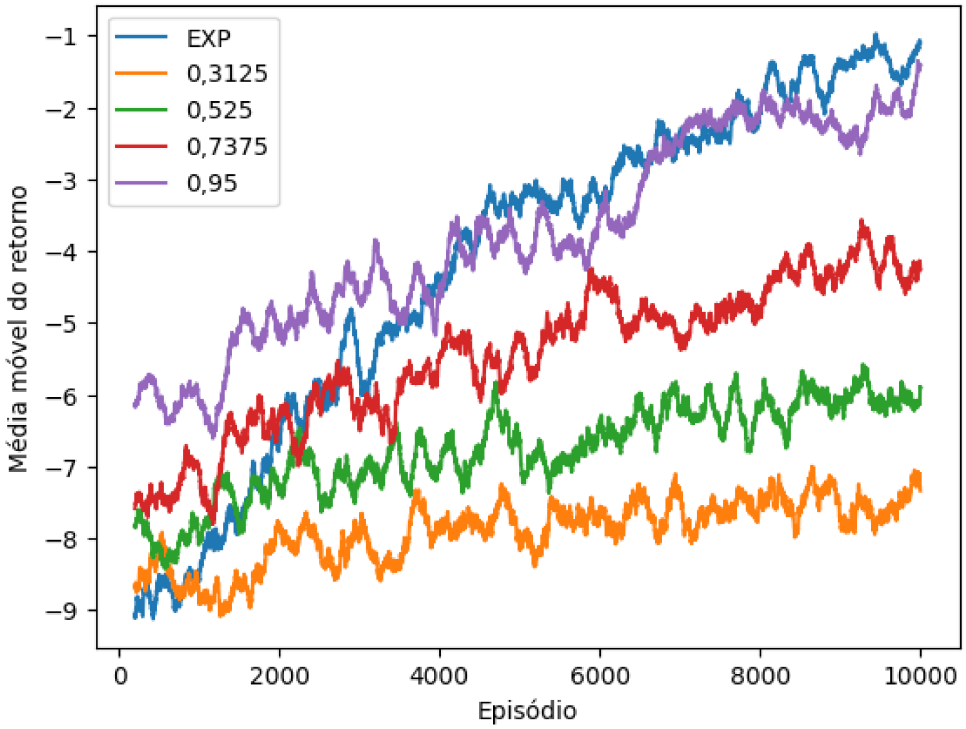

Figura 16 – Comparação do retorno do episódio com ϵ variável.

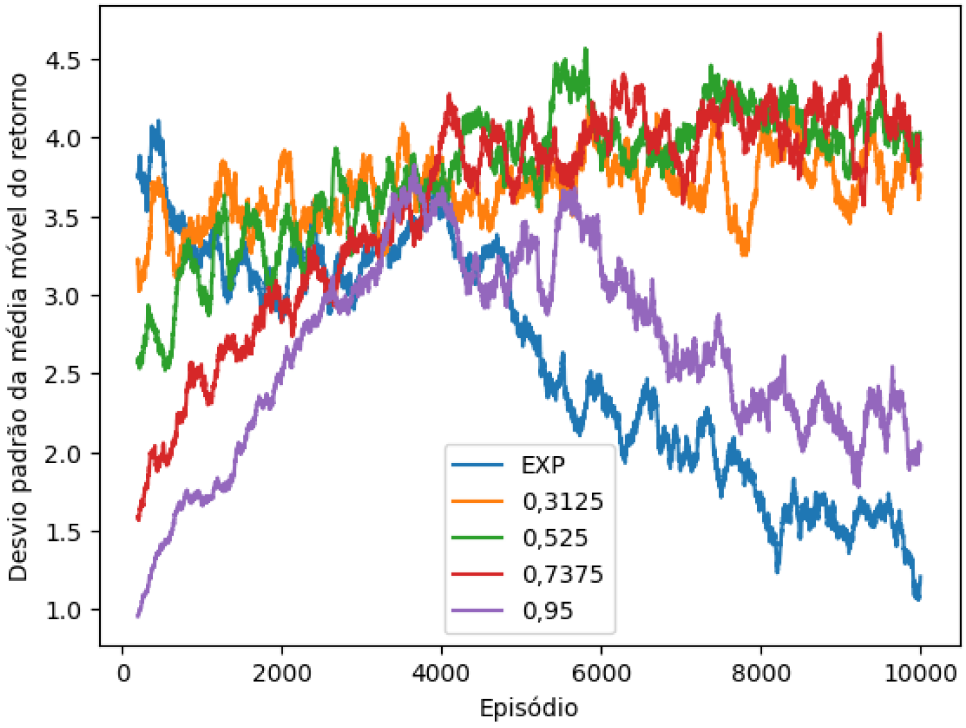

Figura 17 – Comparação do desvio padrão do retorno do episódio com ϵ variável.

Percebe-se que o mesmo padrão que apresentado no SARSA ocorreu no *Q-Learning*. Ao adotar ϵ fixo, obteve-se um maior retorno para valores maiores como mostrado com

$\epsilon = 0{,}95$. Mesmo assim, com ϵ variável obteve-se vantagem quando comparado com todos os outros valores. Neste caso, como apresentado na Figura 18, com $\beta = -0{,}0008$ teve-se o melhor aprendizado.

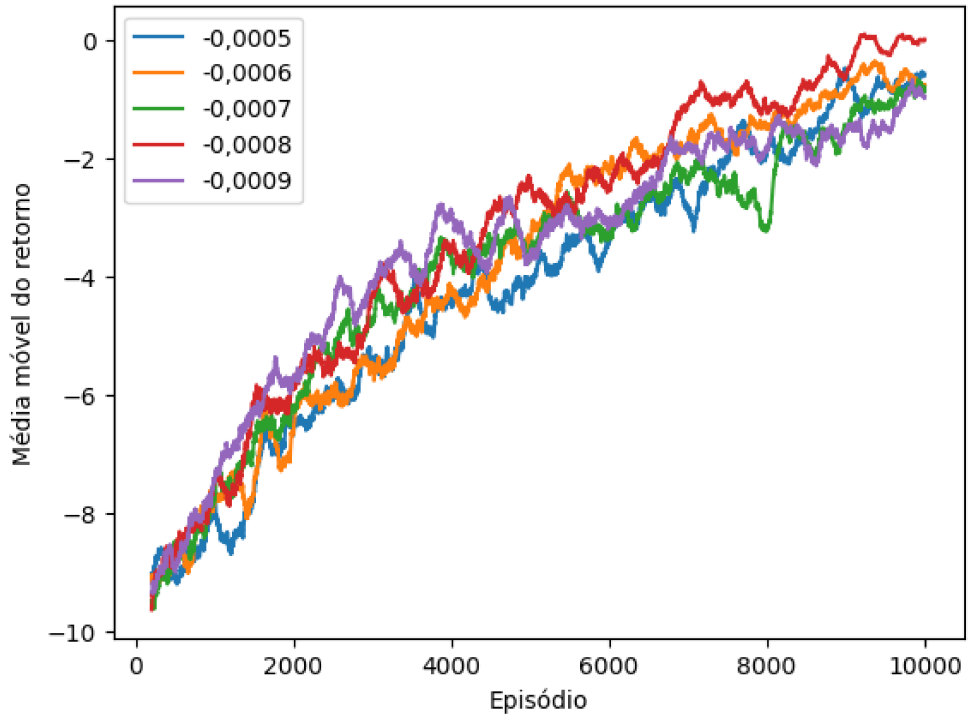

Figura 18 – Comparação do desempenho do sistema para β variável.

4.2.2 Fator de aprendizagem α

Os resultados obtidos no SARSA variando-se o fator α se mostraram similares no *Q-learning*. Como mostrado nas Figuras 19 e 20, para $\alpha \approx 0{,}5$ obtiveram-se os melhores resultados tanto em média quanto em desvio padrão. Da mesma forma, percebe-se a perda de performa advinda da diminuição excessiva de α

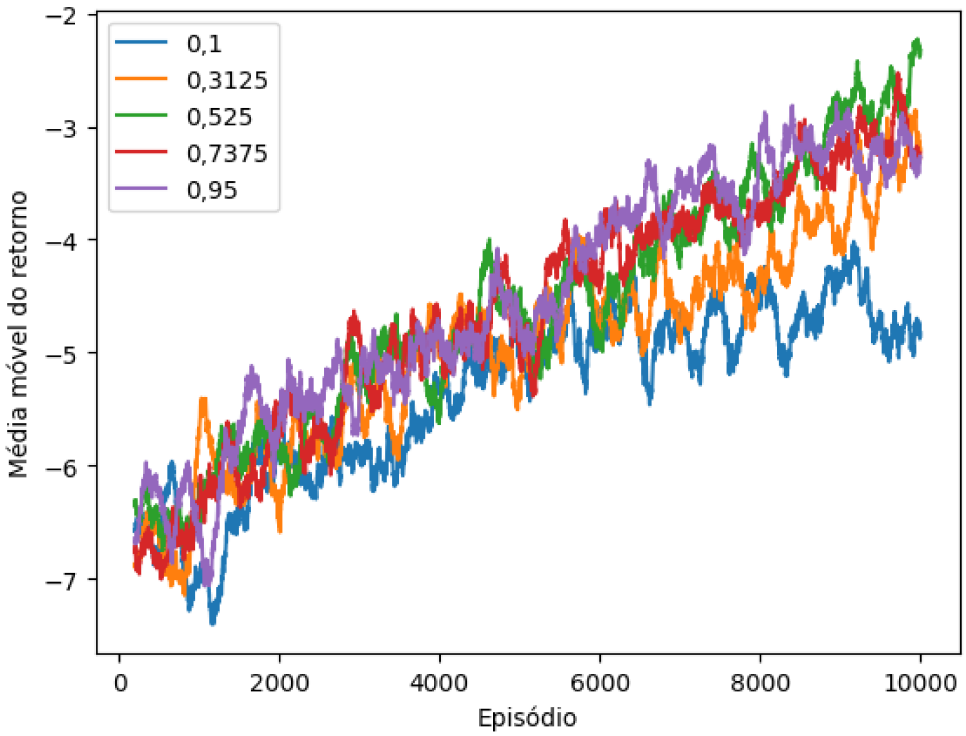

Figura 19 – Comparativo da média do retorno por episódio para diferentes valores de α.

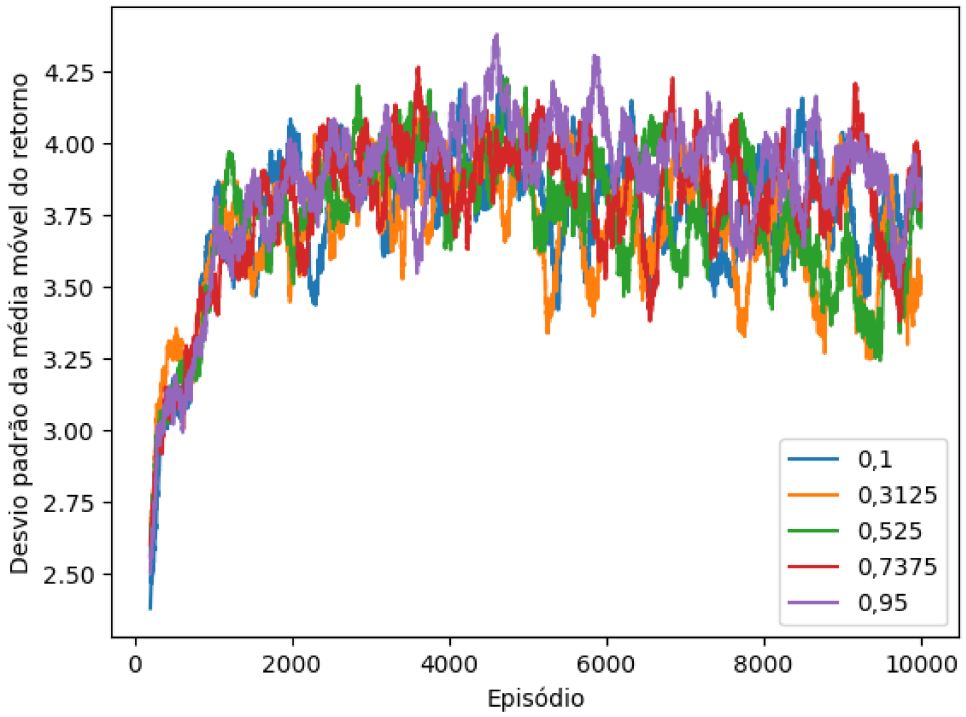

Figura 20 – Comparativo do desvios padrão do retorno por episódio para diferentes valores de α.

4.2.3 Fator de desconto γ

Para o fator γ o, os resultados do aprendizado foram diferentes daqueles obtidos com o SARSA. Para todos os valores escolhidos 0,1, 0,525, 0,95 o retorno por episódio não mudou de forma significante. Isso pode ser verificado nas Figuras 21 e 22.

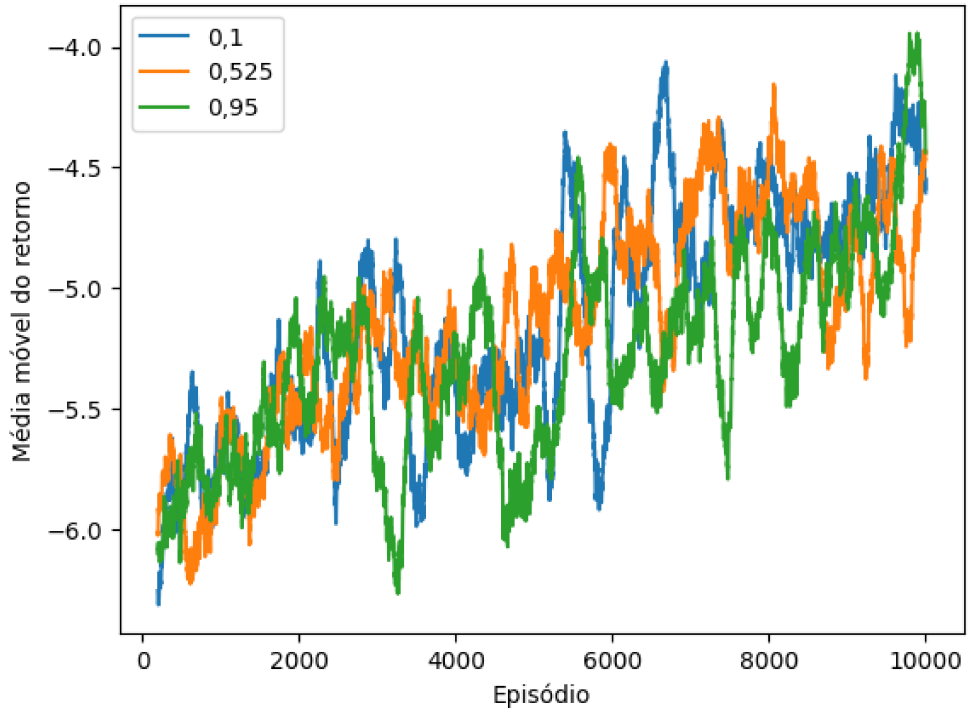

Figura 21 – Comparativo da média do retorno por episódio para diferentes valores de γ.

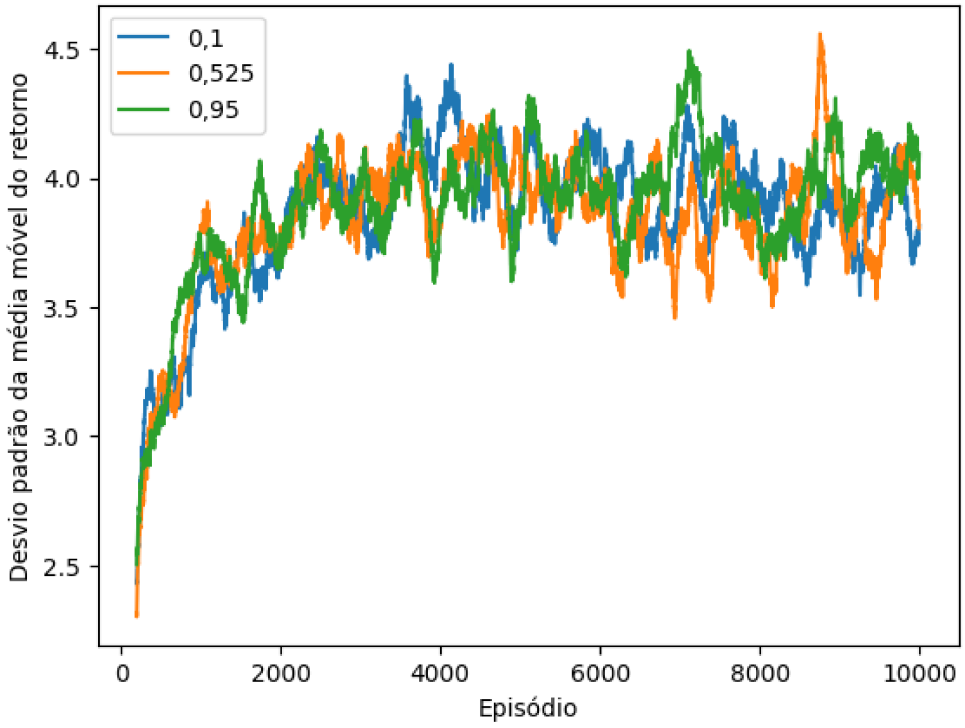

Figura 22 – Comparativo do desvios padrão do retorno por episódio para diferentes valores de γ.

Nas Figuras 23 e 24, compara-se a performance dos dois métodos com os parâmetros otimizados, ou seja, para *Q-Learning*, ϵ obedecendo a equação (4.1) como $\beta = -0{,}0008$ e $\gamma = 0{,}9$ e $\alpha = 0{,}525$ e para SARSA $\beta = -0{,}0009$ e $\gamma = 0{,}1$ e $\alpha = 0{,}525$. Percebe-se tanto em média quanto em desvio uma vantagem para o *Q-Learning*. Isso pode ser justificado pela forma que é escolhida as ações nesses métodos: no SARSA ações aleatórias são consideradas no retorno esperado, enquanto no *Q-Learning*, considera-se a ação que resulta no maior retorno a longo prazo. Mesmo com a explicação dada, vale observar que os dois algoritmos apresentaram média do retorno similares no fim do experimento.

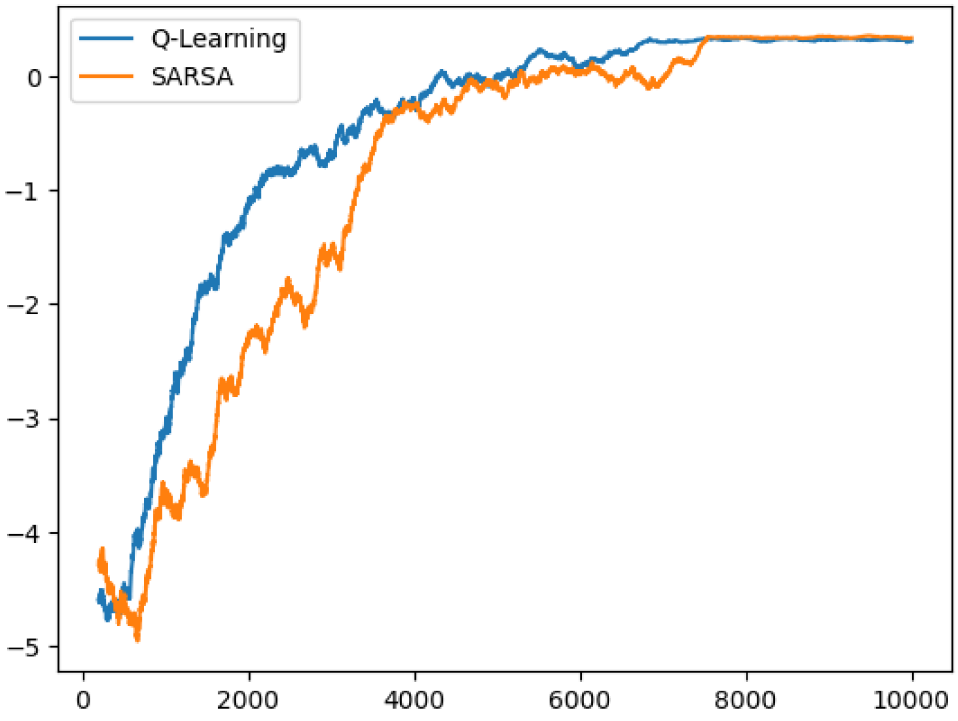

Figura 23 – Comparativo do retorno por episódio entre os algoritmos.

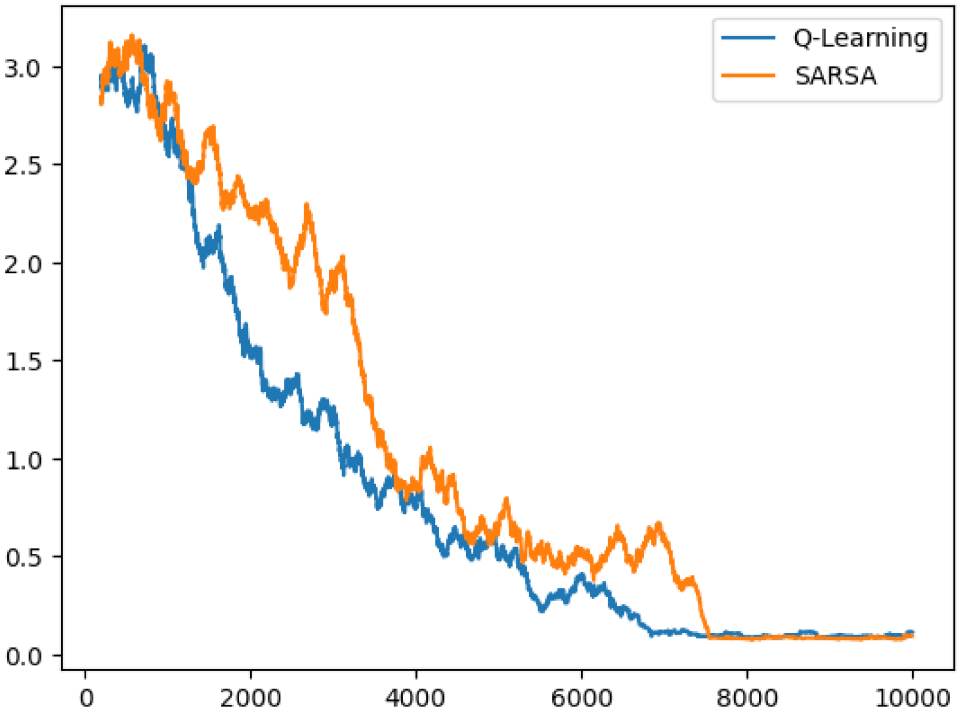

Figura 24 – Comparativo do desvio padrão do retorno por episódio entre os algoritmos.

Por fim, na Figura 26, pode-se observar o agente equilibrando o pêndulo. Nessa figura o agente mantém o pêndulo equilibrado pelo maior tempo possível até que x fique

maior que 0,5 m. Nesse caso, o agente passa a movimentar o carro visando a recuperação da condição de recompensa positiva, porém a simulação é interrompida devido a θ ter assumido o valor $\theta < -30°$. É importante ressaltar que, enquanto $|x| < 0{,}5$ m e $|\theta| < 2°$, a recompensa do agente é positiva e, por isso nenhuma ação é tomada.

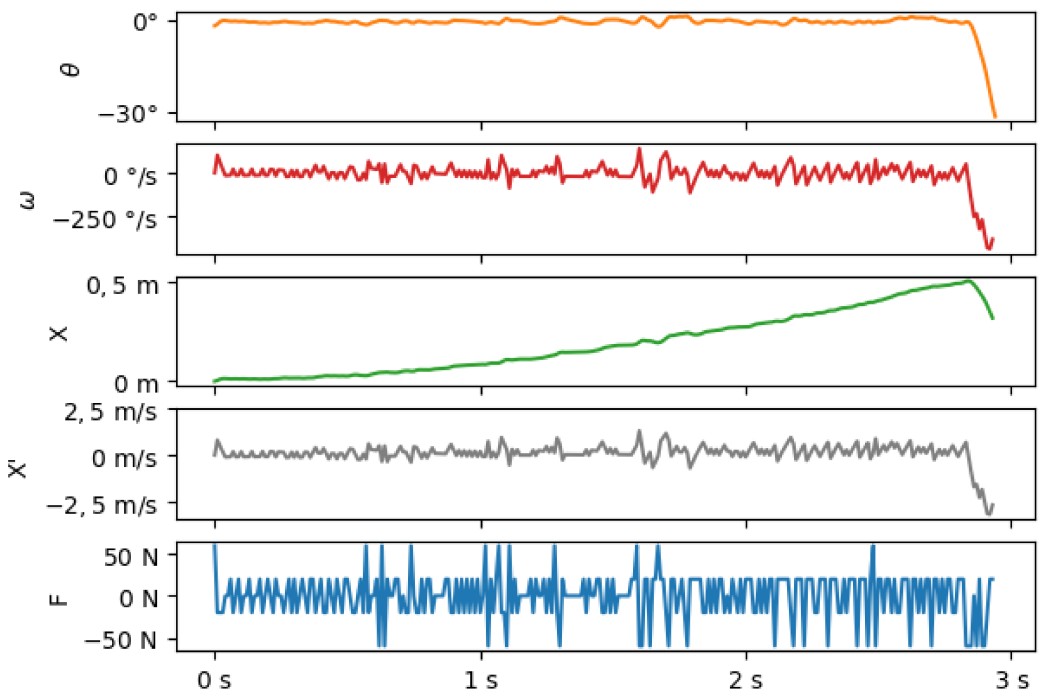

Figura 25 – Evolução dos estados (posições e velocidades angulares e lineares) e da ação de controle (força) em um teste após o aprendizado utilizando *Q-Learning*.

A definição da função de recompensas é um dos passos mais importantes no projeto de controladores baseados em RL. Através de uma definição de um sistema de recompensas inadequado, o comportamento do sistema pode apresentar um desempenho insatisfatório, como visto anteriormente. Por outro lado, com um sistema de recompensas adequado, pode-se adotar tarefas mais complexas, por exemplo, pode-se treinar o agente para retirar a haste da posição de repouso para baixo, $\theta = 180°$, e equilibrá-la na posição invertida. Esse tipo de resultado é mostrado na Figura 26. Nesse cenário, adotou-se $R = 1 - |\theta|$. Pode-se verificar que o agente aprendeu a retirar a haste da posição de repouso para baixo e levou a haste para a posição invertida.

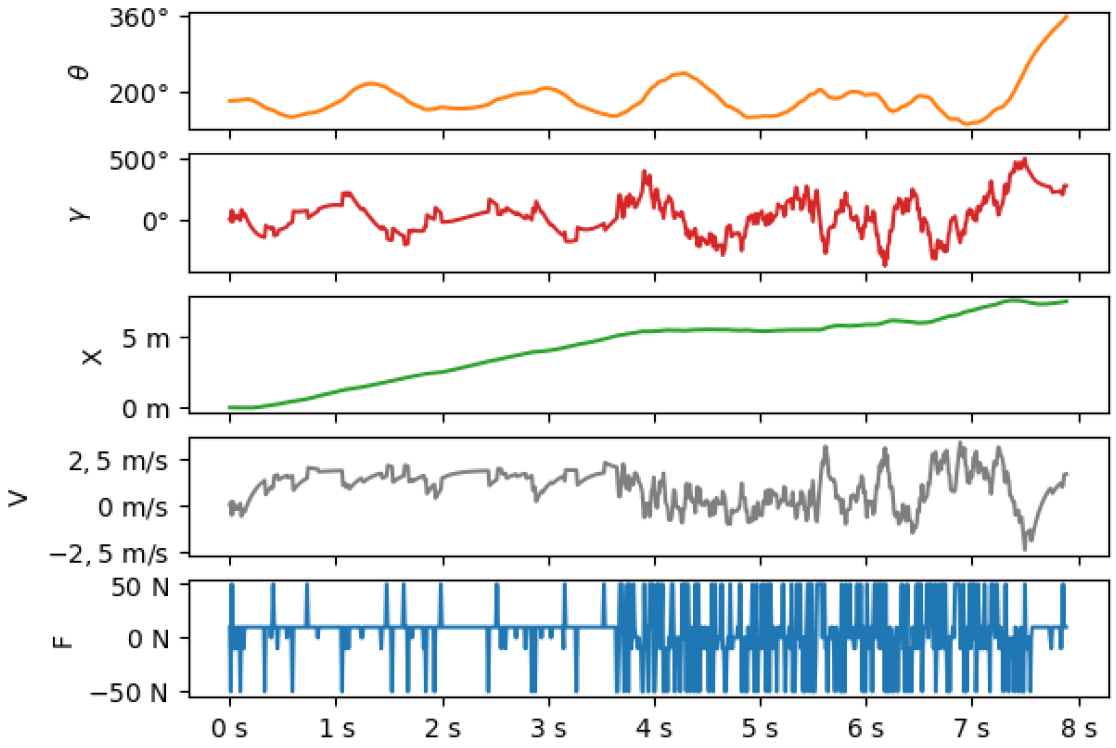

Figura 26 – Agente treinado por *Q-learning* equilibrando a haste a partir da posição de repouso "para baixo" (i.e. $\theta = 180°$).

4.3 *Deep Q-Learning*

Devido às operações envolvidas no treinamento e no cálculo de saída de uma rede neural, as abordagens de DQN e double DQN têm uma carga computacional significativamente maior do que as analisadas até aqui.

Dito isso, os resultados a serem apresentados foram obtidos de forma similar aos métodos anteriores no que se refere aos fatores α, γ e ϵ. Contudo, irá se treinar o agente por apenas 1000 episódios, reduzindo assim o tempo computacional para geração dos resultados.

A RNA treinada terá um modelo linear sequencial, com 2 camadas intermediárias que utilizam a função de ativação *Rectified Linear Unit* (RELU). Nessas camadas teremos 24 nós, na camada mais próxima a camada de entrada e 12 nós na camada mais próxima a camada de saída. A camada de saída utiliza a função de ativação linear e possui 5 neurônios referentes as 5 ações possíveis.

4.3.1 Fator de Ganância ϵ

Os valores de avaliação de ϵ foram os mesmos utilizados nos métodos anteriores. O resultado do treinamento pode ser visto nas Figuras 27 e 28. Os demais valores utilizados

foram $\alpha = 0{,}9$ e $\gamma = 0{,}9$. Foi realizado também um teste variando ϵ conforme a equação (4.1) e definindo $\beta = -0{,}005$, denotado como "EXP". Percebe-se que fatores de ganância mais altos estão, novamente, ligados a resultados melhores, porém, dessa vez, ao utilizar um ϵ variável, a curva não foi melhor do que para $\epsilon = 0{,}95$ e existiu uma fase do treinamento que essa variação de causou uma perda significativa de desempenho. É importante ressaltar, porém, que com ϵ variável a variância dos resultados foi menor, indicando uma maior consistência do desempenho do agente.

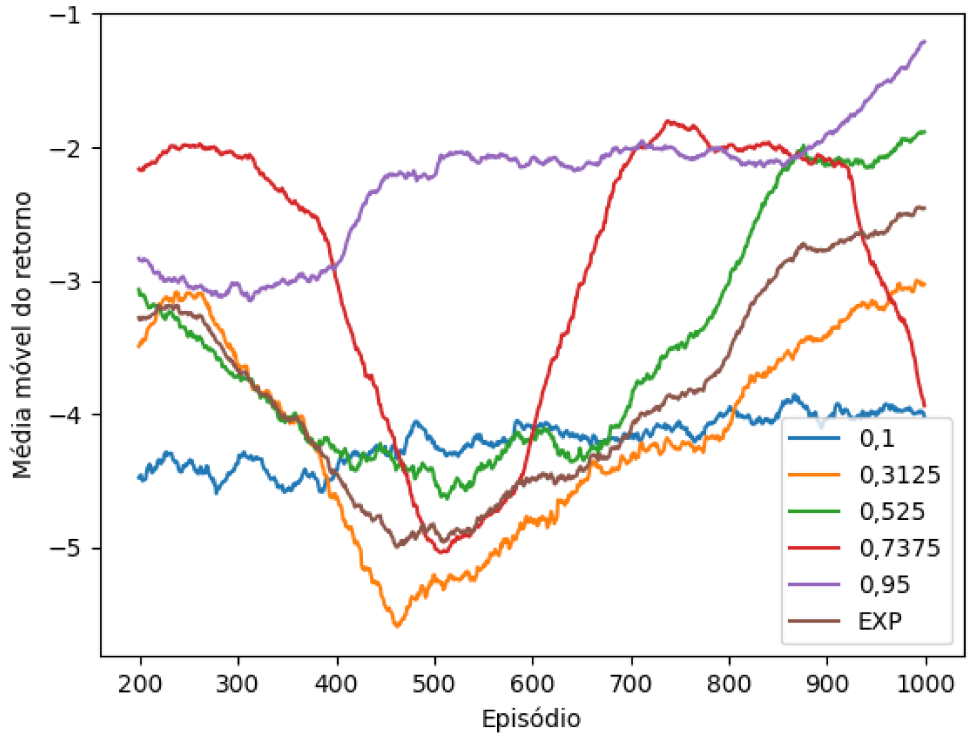

Figura 27 – Comparativo da média do retorno por episódio para diferentes valores de ϵ.

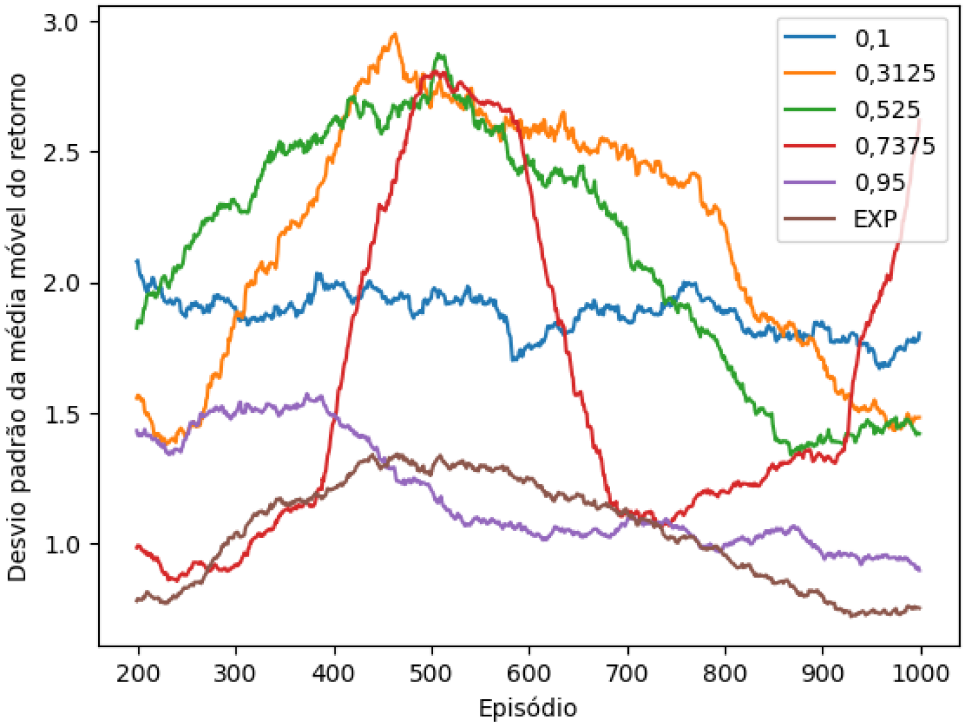

Figura 28 – Comparativo do desvio padrão da média do retorno por episódio entre valores de ϵ.

4.3.2 Fator de aprendizagem α

Também variou-se α nos mesmos valores considerados anteriormente. Os resultados para essa variação são apresentados nas Figuras 29 e 30. Os demais valores utilizados foram $\gamma = 0{,}9$ e $\epsilon = 0{,}85$. Para esses valores percebe-se a influência de α na velocidade do aprendizado do agente, isto é, para $\alpha = 0{,}525$ o aprendizado ocorre mais cedo. Especificamente por volta do episódio 500 a curva já demonstra vantagem quando comparada aos outros valores, e essa vantagem permanece até o fim. Vantagem essa que também é observada se tratando de desvio padrão.

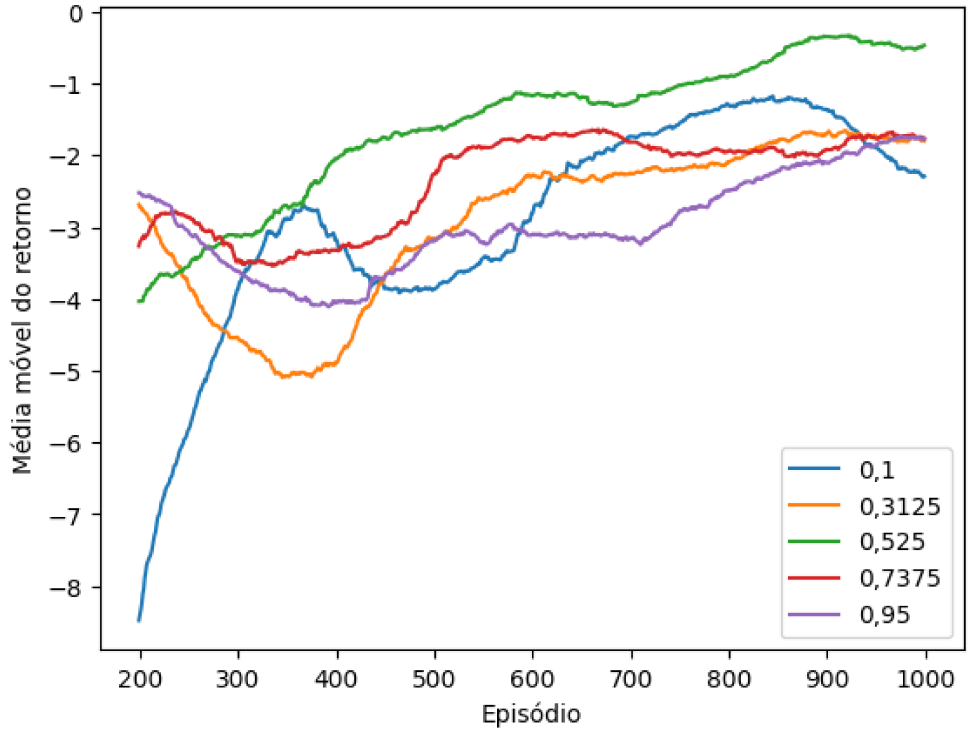

Figura 29 – Comparativo da média do retorno por episódio para diferentes valores de α.

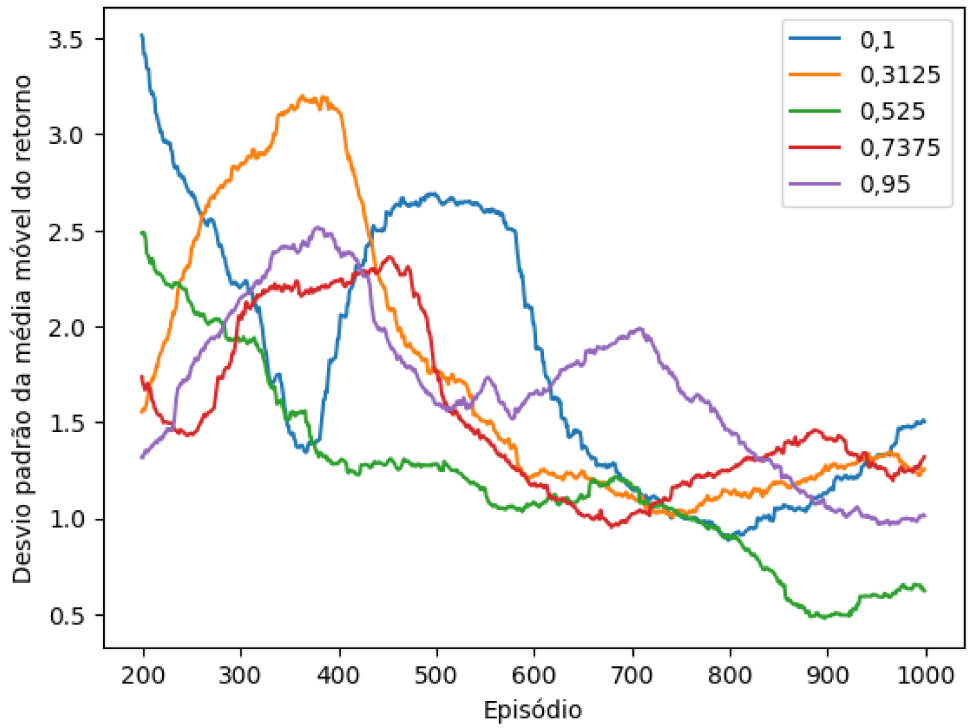

Figura 30 – Comparativo do desvio padrão da média do retorno por episódio entre valores de α.

4.3.3 Fator de desconto γ

Ao variar-se γ observou-se um maior retorno por episódio com $\gamma = 0{,}525$, nos demais parâmetros adotou-se $\epsilon = 0{,}85$ e $\alpha = 0{,}9$. Analisando o desvio padrão, observou-se que com exceção a $\gamma = 0{,}95$, $\gamma = 0{,}525$ obteve o menor desvio no fim do treinamento. Vendo o resultado mostrado nas Figuras 31 e 32 não é possível inferir uma relação direta entre o valor de γ e o retorno, uma vez que é apresentado um comportamento similar nos extremos 0,1 e 0,95

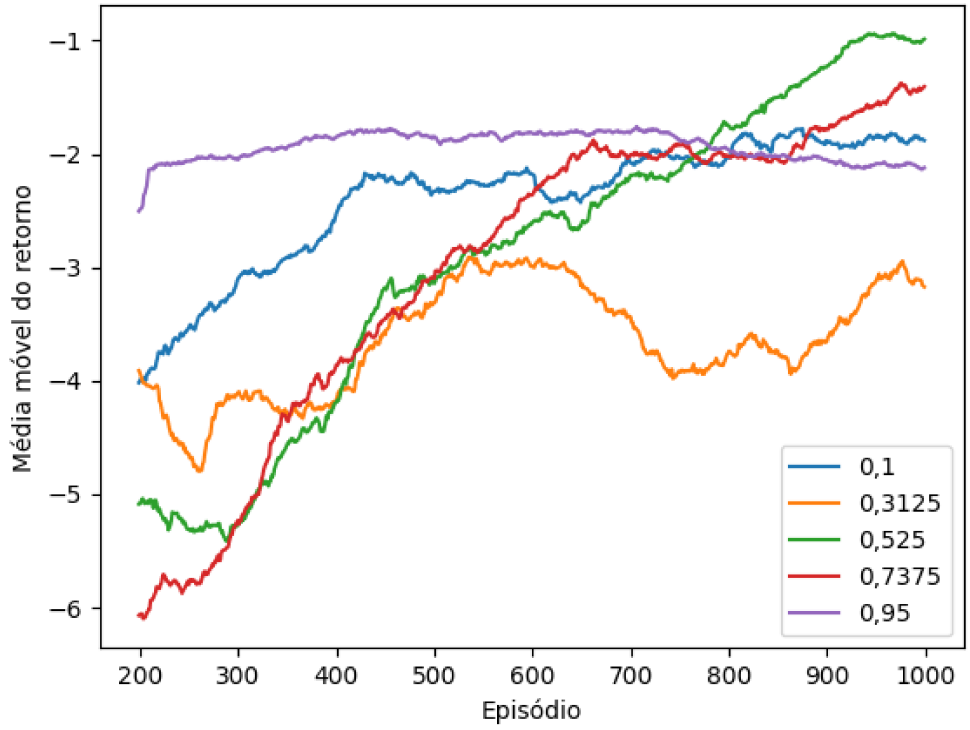

Figura 31 – Comparativo da média do retorno por episódio para diferentes valores de γ.

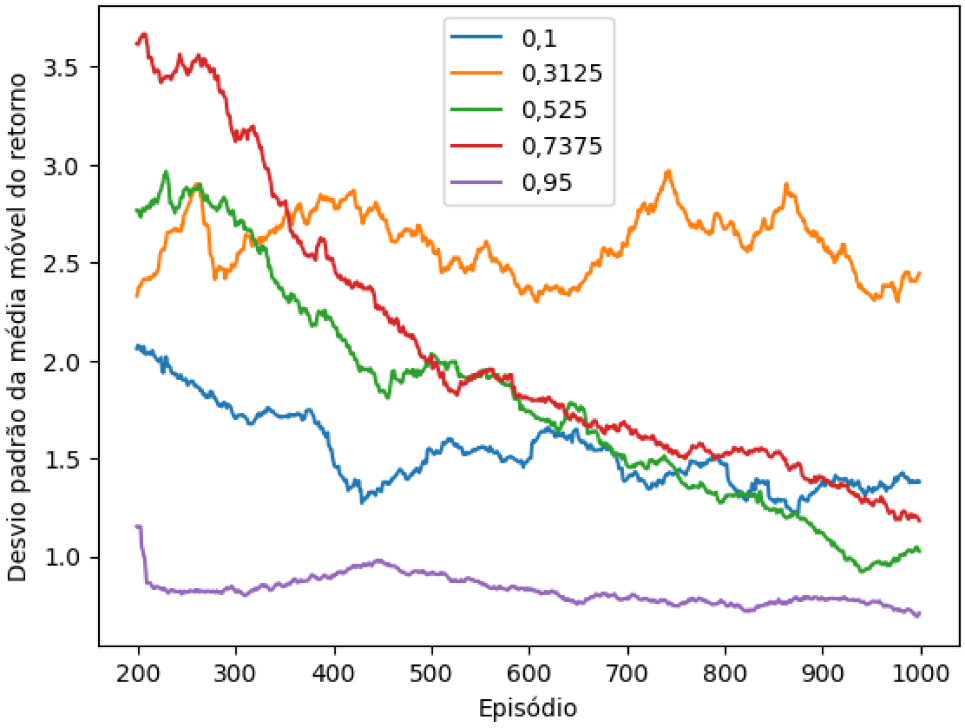

Figura 32 – Comparativo do desvio padrão da média do retorno por episódio entre valores de γ.

4.3.4 Frequência de treinamento

Como explicado anteriormente, a quantidade de episódios tem uma função diferente no *Deep Q-Learning*. Enquanto no SARSA e no *Q-Learning* o número de episódios (ou número de iterações) era diretamente ligado à quantidade de atualizações na tabela de valores Q, no *Deep Q-learning* as transições servem apenas como experiências, podendo assim a atualização da rede (treinamento) ser realizada em frequência independente. Em específico, o número de episódios entre cada treinamento também é parâmetro de ajuste. As Figuras 33 e 34 mostram a evolução da média e desvio padrão do retorno treinando a rede neural a cada 5, 20 e 100 episódios. Os demais parâmetros foram ajustados iguais a $\alpha = 0{,}9$, $\gamma = 0{,}9$ e $\epsilon = 0{,}85$. Pode-se constatar que treinar a rede neural com maior frequência acelera o aprendizado do agente. Contudo, isso está associado a um custo computacional maior. Esse comportamento foi observado tanto no *Deep Q-learning* como no *Double deep Q-learning*.

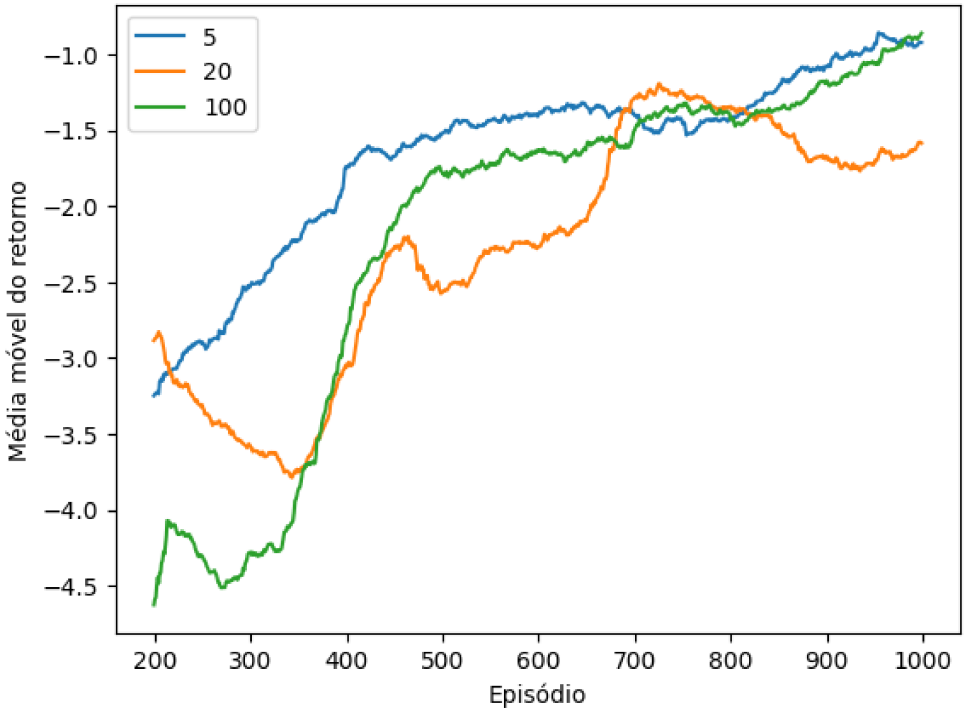

Figura 33 – Comparativo da média do retorno para diversas frequências de treinamento.

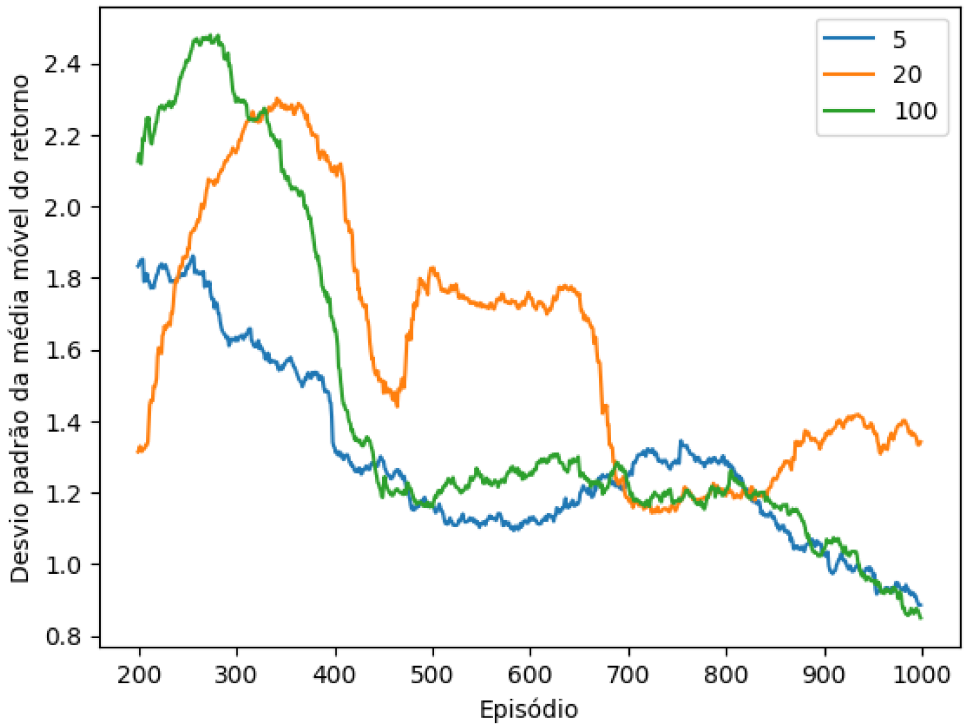

Figura 34 – Comparativo do desvio pradrão da média do retorno para diversas frequências de treinamento.

Por fim, na Figura 35 observa-se o agente estabilizando o agente até o fim do

experimento. Esse por sua vez chegou ao fim pelo carro ter ultrapassado o limite de $x < -1$.

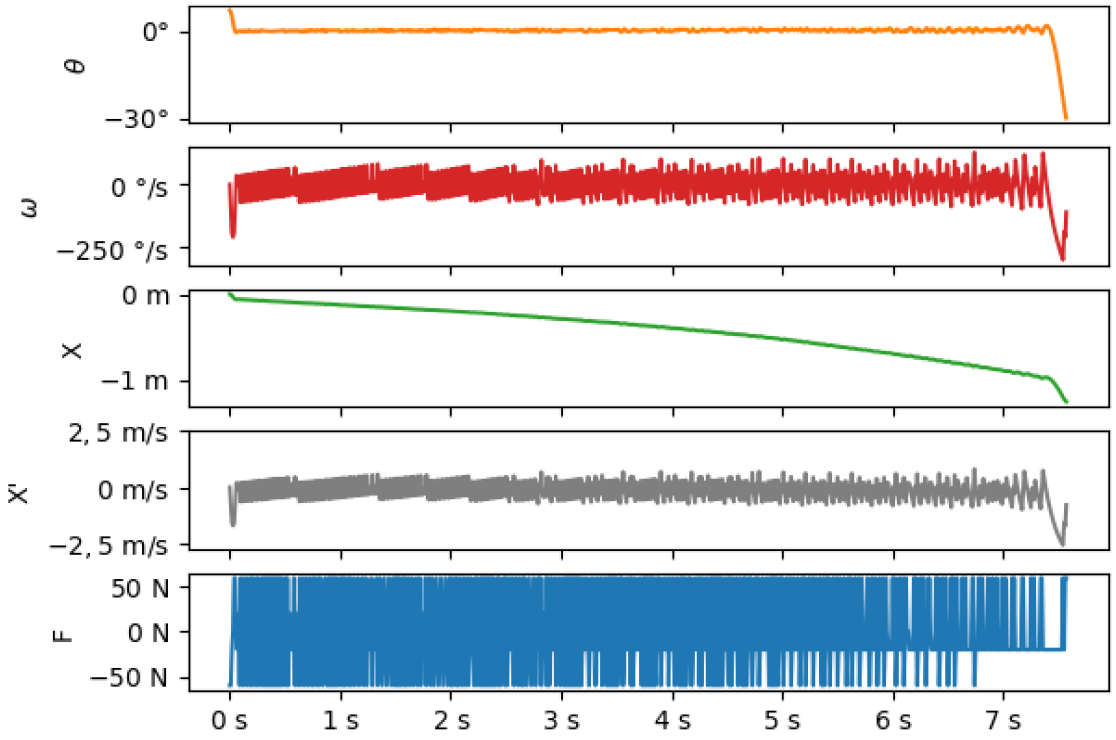

Figura 35 – Evolução dos estados (posições e velocidades angulares e lineares) e da ação de controle (força) em um teste após o aprendizado utilizando *Deep Q-Learnging*.

4.4 *Double Deep Q-Learning*

Para o algoritmo de *Double deep Q-learning* os testes foram realizados da mesma forma já citada, e assim obtiveram-se resultados descritos a seguir.

4.4.1 Fator de Ganância ϵ

Fatores de ganância ϵ maiores se mostraram melhores tanto na velocidade de aprendizado, quanto no retorno ao final de 1000 episódios (vide Figuras 36 e 37). Em particular $\epsilon = 0{,}95$ esteve associado com um aprendizado mais rápido e uma maior média móvel do retorno. Nos métodos que utilizam redes neurais para a previsão do retorno, explorar novas ações antes de seguir uma política *greedy* não se mostrou tão eficiente quanto nos outros métodos, esse fato reforça a diferença do modo de treinamento entre os dois modos mostrando que enquanto no SARSA e *Q-learning* é necessário testar a melhor ação em cada estado, métodos de que utilizam de redes neurais precisam de uma base de dados e frequência de treinamento para atingirem a estabilidade.

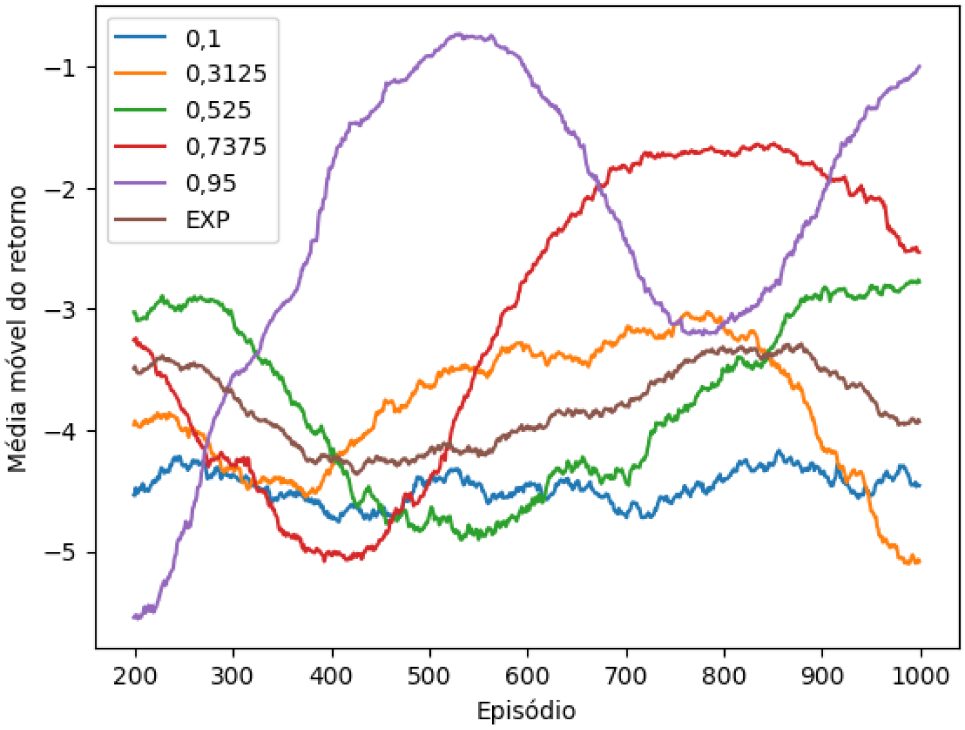

Figura 36 – Comparativo da média do retorno por episódio para diferentes valores de ϵ.

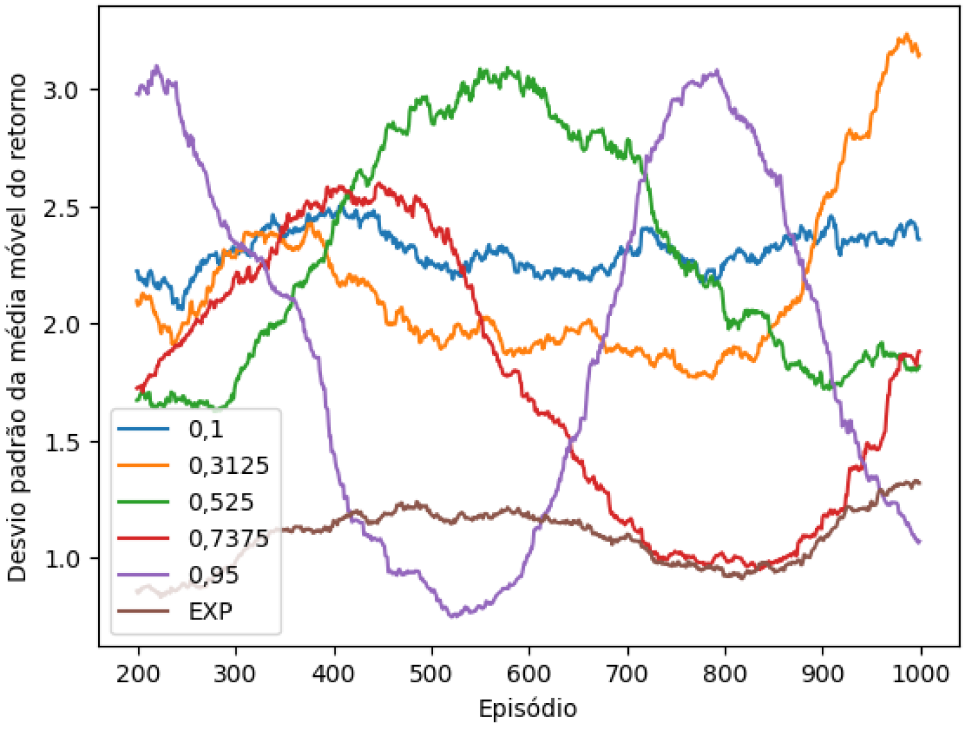

Figura 37 – Comparativo da média do retorno por episódio para diferentes valores de ϵ.

4.4.2 Fator de aprendizagem α

Os resultados observados no método de *Deep Q-Learning* ocorreram de forma similar no *Double deep Q-Learning*. Nos resultados nas Figuras 38 e 39, percebe-se que o agente aprende mais cedo para maiores valores medianos de α. Esse comportamento se demonstrou uma constante em todos os métodos apresentados nesse trabalho. Para os demais valores utilizou-se $\epsilon = 0{,}85$ e $\gamma = 0{,}9$.

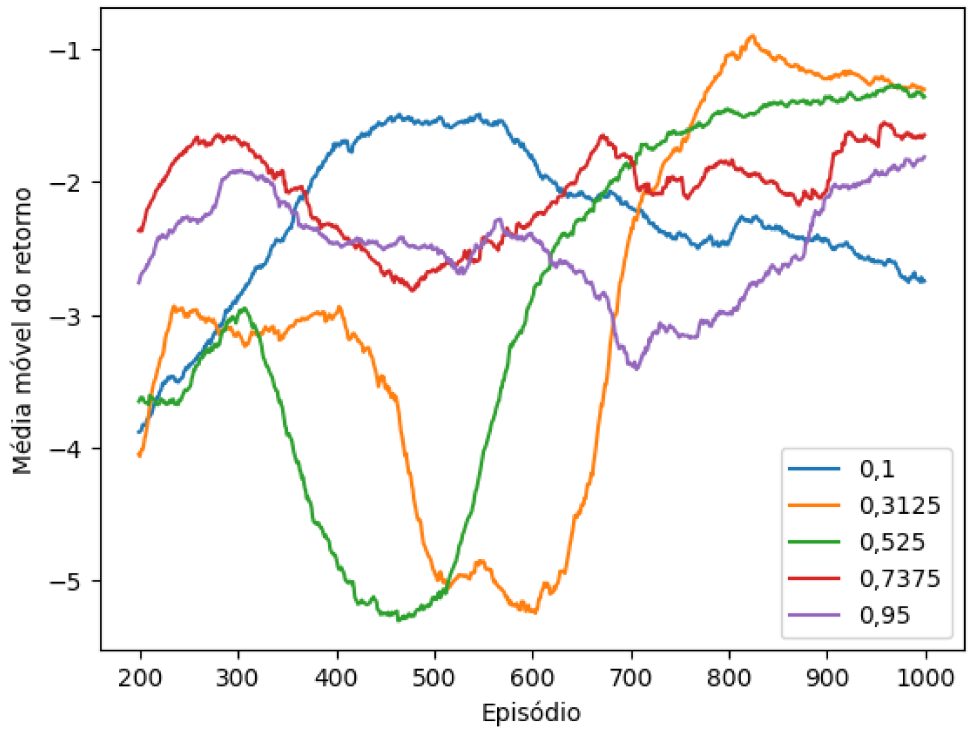

Figura 38 – Comparativo da média do retorno por episódio para diferentes valores de α.

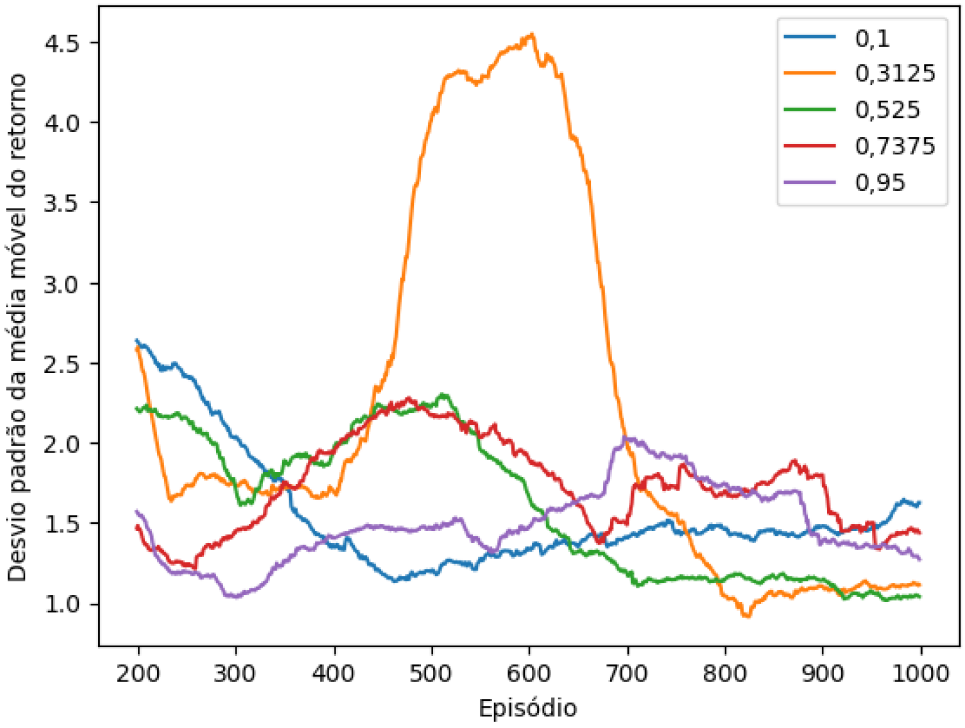

Figura 39 – Comparativo da média do retorno por episódio para diferentes valores de α.

4.4.3 Fator de desconto γ

Finalmente, foram testados os valores de γ conforme apresentado nas Figuras 40 e 41 e os melhores resultados foram alcançados com $\gamma = 0{,}7375$. Com esse ajuste, o agente leva em média 80 ms para atingir seu objetivo dado a posição inicial descrita nesse capítulo. Os demais valores utilizados foram $\epsilon = 0{,}85$ e $\alpha = 0{,}9$.

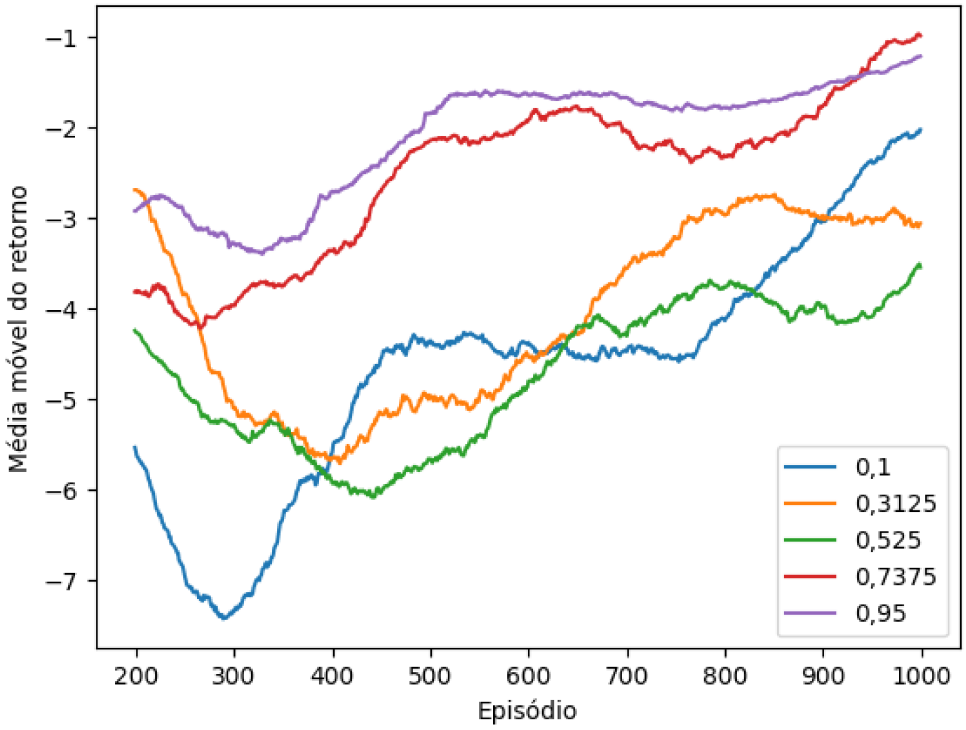

Figura 40 – Comparativo da média do retorno por episódio para diferentes valores de γ.

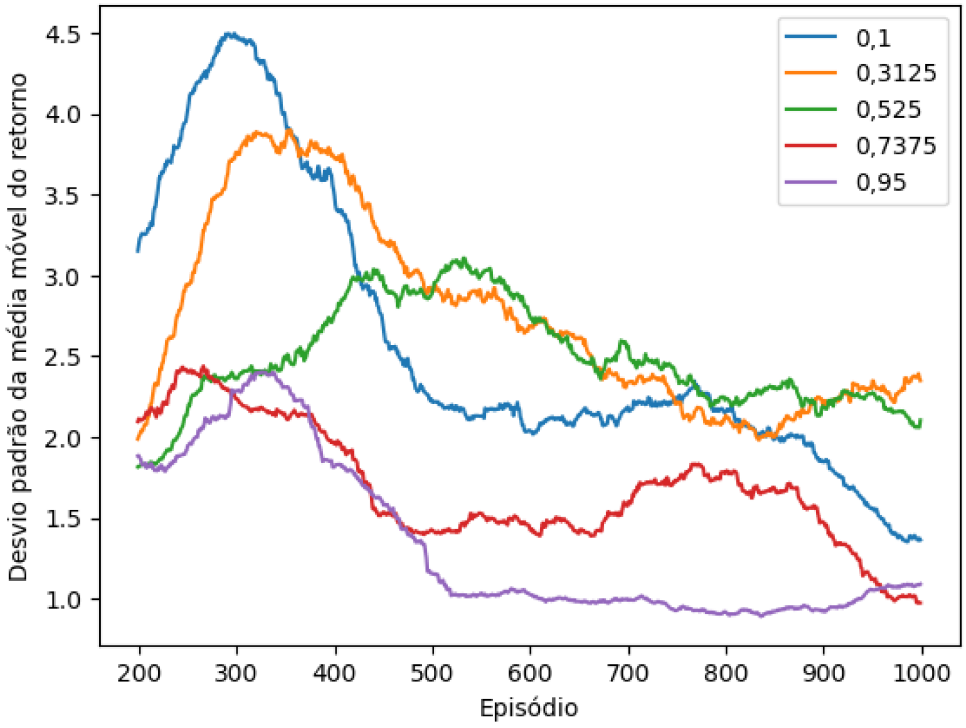

Figura 41 – Comparativo da média do retorno por episódio para diferentes valores de γ.

Na Figura 42 pode-se observar o comportamento da planta controlada por um agente treinado. Observa-se que o agente conseguiu equilibrar o pêndulo na sua posição

invertida até o fim da simulação, resultado esse não obtido em nenhum dos métodos anteriores.

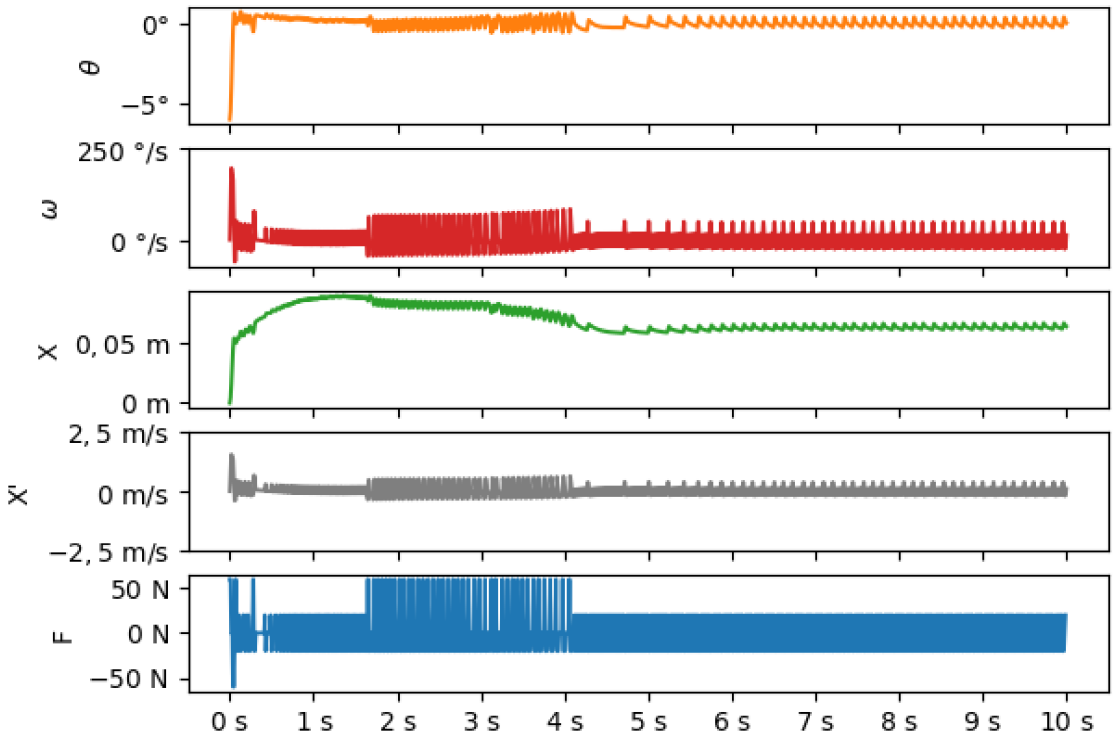

Figura 42 – Evolução dos estados (posições e velocidades angulares e lineares) e da ação de controle (força) em um teste após o aprendizado utilizando *Double Deep Q-Learning*.

Por fim, apresenta-se nas Figuras 43 e 44 (média e desvio padrão) a comparação entre todos os métodos de aprendizado por reforço testados no presente trabalho. Os parâmetros adotas para realização dessa comparação são mostrados na Tabela 2.

Tabela 2 – Parâmetros otimizados utilizados.

Algoritmo	ϵ	α	γ
SARSA	Variável com $\beta = -0{,}009$	0,525	0,1
Q-Learning	Variável com $\beta = -0{,}009$	0,525	0,95
Deep Q-Learning	0,95	0,525	0,525
Double Deep Q-Learning	0,95	0,525	0,7375

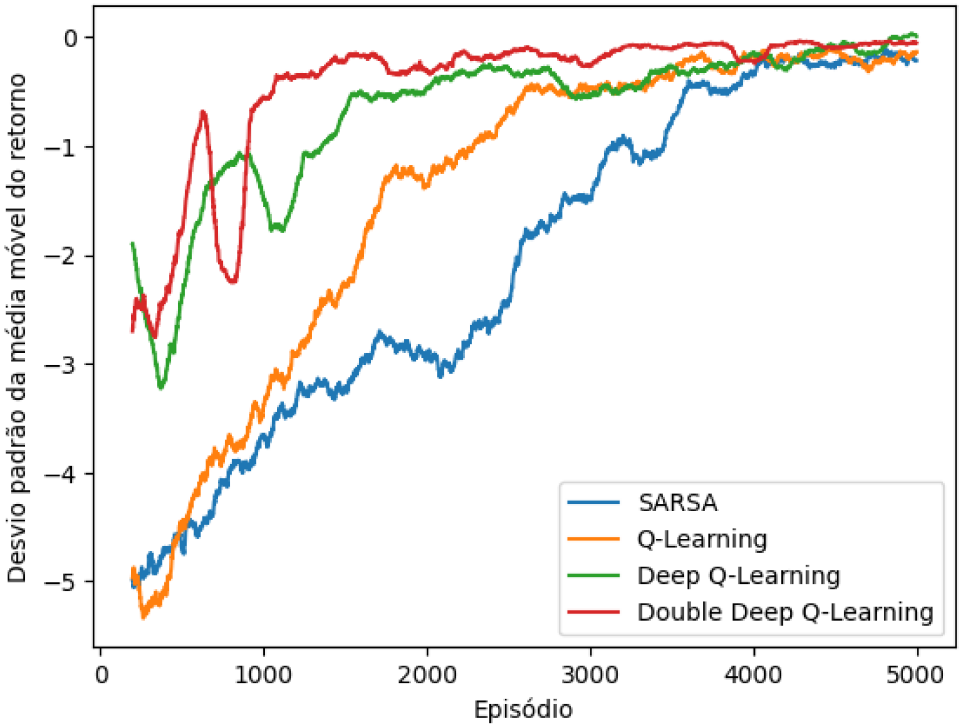

Figura 43 – Comparação da Evolução da média do retorno por episódio dentre os algoritmos.

Percebe-se um melhor resultado tanto em média quanto em desvio do algoritmo de *Double Deep Q-Learning*. Isso foi notado ao longo da maior parte do decorrer dos 3000 episódios de treinamento. Ademais, percebe-se também uma vantagem, principalmente nos episódios iniciais, de métodos que utilizam redes neurais, *Deep Q-Learning* e *Double Deep Q-Learning*, sobre métodos que não, tornando-os, mesmo com as desvantagens citadas nesse capítulo, como métodos mais interessantes para o aprendizado do controle de sistemas. Entretanto, percebe-se também a convergência de todos os métodos para o mesmo ponto ao final do treinamento, não divergindo de forma significante o resultado dos métodos ao fim.

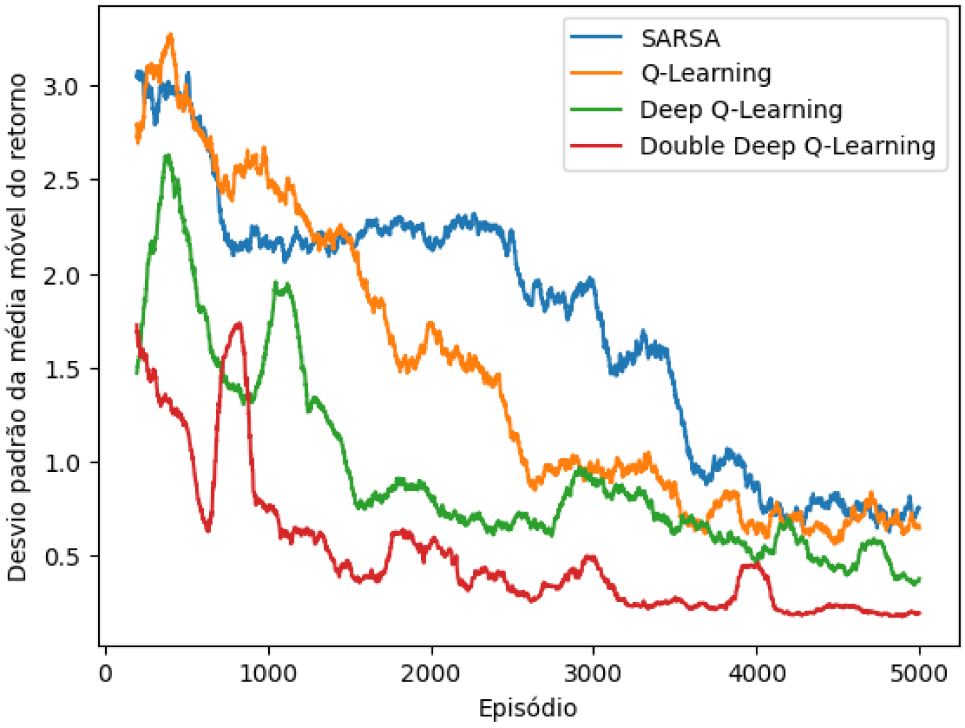

Figura 44 – Comparação da Evolução do desvio padrão da média do retorno por episódio dentre os algoritmos.

5 Conclusão

No aprendizado baseado em reforço (*Reinforcement Learning* – RL), pretende-se determinar uma sequência de ações para que o agente atinja um objetivo. Tipicamente supõe-se que o agente encontra-se em um ambiente desconhecido no qual em cada instante de tempo, aplica-se uma ação, de acordo com uma política π e os estados, recebendo do ambiente a recompensa e os novos estados. Os algoritmos de RL visam modificar essa política de decisões do agente de modo a maximizar o somatório das recompensas ao longo do tempo, o que é denominado retorno.

No presente trabalho foram avaliados quatro RL para o controle de um carro pêndulo. Em particular foram implementados SARSA, *Q-Learning*, *Deep Q-Learning* e *Double Deep Q-Learning*. O objetivo dos métodos consistia em treinar um agente para manter uma haste equilibrada e o carro dentro de um certo intervalo em torno da origem. Com esse propósito, definiu-se uma função que retorna recompensas positivas, caso esses objetivos fossem alcançados, e recompensas negativas, caso contrário. Essa mesma função foi adotada em todos os métodos. Os métodos e a simulação do sistema foram implementados em Python considerando diferentes condições iniciais.

Resultados de simulação no modelo não linear do sistema demostraram a capacidade do aprendizado do agente, sendo este capaz de realizar a tarefa proposta a todos os métodos implementados. Além disso, foi verificado o efeito de variações nos parâmetros de ajuste de cada método no aprendizado do agente. Esses últimos resultados podem orientar outros projetistas na implementação dos métodos de RL aqui considerados. Por meio dos resultados, pôde-se constatar que o método *Double Deep Q-Learning* proporcionou um aprendizado mais rápido. Portanto, pode-se considerar o *Double Deep Q-Learning* como o melhor método testado.

Trabalhos futuros podem incrementar a comparação aqui realizada incluindo métodos mais recentes de RL. Por exemplo, sugere-se verificar o desempenho do *Twin Delayed Deep Deterministic Policy Gradient*, que hodiernamente pode ser entendido como o estado da arte em RL, no controle do carro-pêndulo.

Referências

ACHIAM, J. *Part 1: Key Concepts in RL*. 2018. Disponível em: <https://spinningup.openai.com/en/latest/spinningup/rl_intro.html>. Citado na página 13.

FUKUSHIMA, W. *Aprendizado por Reforço 2 Processo de Decisão de Markov — Parte 1*. 2020. Disponível em: <https://medium.com/turing-talks/aprendizado-por-reforço-2-processo-de-decisão-de-markov-mdp-parte-1-84e69e05f007>. Citado na página 15.

HAFNER, R.; RIEDMILLER, M. Reinforcement learning in feedback control: Challenges and benchmarks from technical process control. *Machine learning*, Springer, v. 84, p. 137–169, 2011. Citado na página 13.

HASSELT, H. van; GUEZ, A.; SILVER, D. *Deep Reinforcement Learning with Double Q-learning*. 2015. Citado na página 23.

KAELBLING, L. P.; LITTMAN, M. L.; MOORE, A. W. Reinforcement learning: A survey. *Journal of artificial intelligence research*, v. 4, p. 237–285, 1996. Citado 2 vezes nas páginas 15 e 18.

MARTINS, L.; VIEIRA, R. Controle por modos deslizantes aplicado a um pêndulo invertido. In: . [S.l.: s.n.], 2015. Citado na página 14.

MNIH, V. et al. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. Disponível em: <http://arxiv.org/abs/1312.5602>. Citado 2 vezes nas páginas 13 e 22.

NETTO, J. L. L. *Relatório Final*. Tese (Doutorado) — Universidade Estadual de Campinas, 2016. Citado 3 vezes nas páginas 25, 26 e 27.

PELLEGRINI, J.; WAINER, J. Processos de decisão de markov: um tutorial. *Revista de Informática Teórica e Aplicada*, v. 14, n. 2, p. 133–179, 2007. Citado 2 vezes nas páginas 16 e 19.

SCHULMAN, J. *Optimizing expectations: From deep reinforcement learning to stochastic computation graphs*. Tese (Doutorado) — UC Berkeley, 2016. Citado 2 vezes nas páginas 12 e 13.

SUTTON, R. S.; BARTO, A. G. *Reinforcement learning: An introduction*. 2. ed. Estados Unidos da América: MIT press, 2018. Citado 8 vezes nas páginas 12, 13, 15, 16, 17, 19, 20 e 21.