UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

MARCOS LEVI SANTIAGO PAIVA

UM ESTUDO DA APLICAÇÃO DE APRENDIZAGEM POR REFORÇO NO CONTROLE DE SISTEMAS NÃO LINEARES

FORTALEZA

2023

MARCOS LEVI SANTIAGO PAIVA

UM ESTUDO DA APLICAÇÃO DE APRENDIZAGEM POR REFORÇO NO CONTROLE DE SISTEMAS NÃO LINEARES

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia Elétrica do Centro de Tecnologia da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia Elétrica.

Orientador: Prof. Dr. Arthur Plínio de Souza Braga.

FORTALEZA

2023

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

P169e Paiva, Marcos Levi Santiago.
Um estudo da aplicação de aprendizagem por reforço no controle de sistemas não lineares / Marcos Levi Santiago Paiva. – 2023.
78 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Centro de Tecnologia, Curso de Engenharia Elétrica, Fortaleza, 2023.
Orientação: Prof. Dr. Arthur Plínio de Souza Braga.

1. Aprendizagem por reforço. 2. Pêndulo invertido. 3. Teoria de controle. I. Título.
CDD 621.3

MARCOS LEVI SANTIAGO PAIVA

UM ESTUDO DA APLICAÇÃO DE APRENDIZAGEM POR REFORÇO NO CONTROLE DE SISTEMAS NÃO LINEARES

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia Elétrica do Centro de Tecnologia da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia Elétrica.

Aprovada em:

BANCA EXAMINADORA

Prof. Dr. Arthur Plínio de Souza Braga (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Márcio André Baima Amora
Universidade Federal do Ceará (UFC)

Prof. Dr. Magno Prudêncio de Almeida Filho
Universidade Federal do Ceará (UFC)

MSc. Eng. Kaio Martins Ramos
Universidade Federal do Ceará (UFC)

A todos os cérebros que se revoltam contra a estagnação. Benditos sejam os que não param diante das dúvidas, questionamentos e convicções.

AGRADECIMENTOS

À minha mãe, por sempre me incentivar a estudar.

À minha família, por sempre estar ao meu lado quando preciso.

Ao Prof. Dr. Arthur Plínio de Souza Braga, por me apresentar ao tema sobre aprendizagem por reforço e por me orientar neste trabalho.

Aos meus amigos que me acompanharam durante estes 5 anos de graduação.

Aos professores Nildo Loiola e Fernando Antunes, pelas oportunidades de bolsa.

Aos professores Márcio André, Magno Prudêncio e Kaio Martins por participarem da banca examinadora e pelas valiosas colaborações e sugestões.

À Comerc Energia, pela oportunidade de estágio.

"Eu vos digo: é preciso ter ainda caos dentro de si, para poder dar à luz uma estrela dançante. Eu vos digo: tendes ainda caos dentro de vós."
(Friedrich Nietzsche)

RESUMO

O pêndulo invertido em cima de um carrinho é um dos problemas mais conhecidos da teoria de controle. Existem diversas formas de controlá-lo, desde técnicas mais clássicas, como linearizar o problema, a técnicas mais elaboradas, como utilizar inteligência computacional. Este trabalho, portanto, apresentará duas formas para resolver este problema: a implementação de um controle LQR e a aplicação de Aprendizagem por Reforço. Devido ao aumento no interesse em *Machine Learning* e em novas técnicas de controle, o objetivo será fazer a comparação entre um método mais difundido na literatura, linearização do sistema utilizando variáveis de espaço de estados, com um método mais recente, mostrando a possibilidade de um sistema aprender a lei de controle apenas interagindo com a planta. Será feito uma revisão bibliográfica dos assuntos mais pertinentes para o entendimento do trabalho, em seguida será explicado a metodologia aplicada, montagem do problema utilizando agentes de aprendizagem por reforço e controlador LQR, por fim os resultados serão apresentados e comparados. Todo o trabalho será realizado por meio de simulações utilizando o *software* Matlab.

Palavras-chave: aprendizagem por reforço; pêndulo invertido; teoria de controle.

ABSTRACT

The inverted pendulum on top of a cart (cartpole problem) is one of the best-known problems in control theory. There are several ways to control it, from more classic techniques, such as linearizing the problem, to more elaborate techniques, such as using computational intelligence. This work, therefore, will present two ways to solve this problem: the implementation of an LQR control and the application of Reinforcement Learning. Due to the increase in interest in Machine Learning and new control techniques, the objective will be to compare a method more widespread in the literature, linearization of the system using state space variables, with a more recent method, showing the possibility of a system learning the control law just by interacting with the plant. A bibliographic review will be carried out on the most relevant subjects for understanding the work, then the methodology applied will be explained, the problem set up using reinforcement learning agents and LQR controller, and finally the results will be presented and compared. All work will be carried out through simulations using Matlab software.

Keywords: control theory; inverted pendulum; reinforcement learning

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AR	Aprendizagem por Reforço
DL	*Deep Learning*
DQN	*Deep Q-Network*
ICA	Inteligência Computacional Aplicada
LQR	Regulador Quadrático Linear
LTI	*Linear time-invariant*
MDP	*Markov Decision Processes*
MIMO	*multiple-input multiple-output*
ML	*Machine Learning*
MLP	Multilayer Perceptron
ReLU	*Rectified Linear Unit*
RNA's	Redes Neurais Artificiais
SISO	*single-input single-output*

LISTA DE SÍMBOLOS

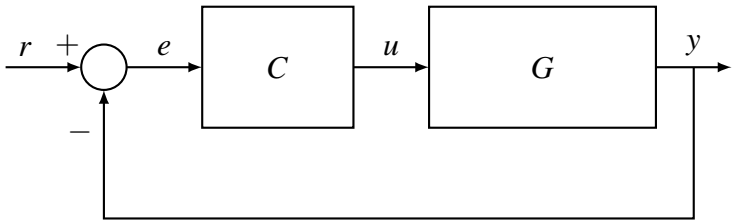
G	Função de transferência da planta
C	Função de transferência do controlador
r	Sinal da referência de entrada
e	Sinal de erro atuante
u	Sinal de controle
y	Sinal da saída
x_i	I-ésimo sinal de entrada
w_i	I-ésimo peso sináptico
η	Fator de aprendizagem rede neural
θ	Bias rede neural
J	Função custo
S	Conjunto discreto de estados
A	Conjunto discreto de ações
R	Conjunto discreto de recompensas
G_t	Retorno após t intervalos
γ	Desconto
$v(s)$	Função valor de estado
$q(s,a)$	Função valor de estado e ação

SUMÁRIO

1 INTRODUÇÃO

Controlar sistemas sempre foi uma tarefa presente na história da humanidade, seja controlando a quantidade de madeira para manter um abrigo numa temperatura agradável, seja controlando a angulação de um foguete que está viajando em direção à Lua. Em inúmeras tarefas, das mais simples as mais complexas, está presente a teoria básica de controle. Um sistema de controle pode ser definido como: um sistema (planta) sujeito a um sinal de controle, no qual se deseja uma determinada saída a partir de uma entrada especificada (NISE, 2012). A Figura 1 representa um sistema de controle em malha fechada, conhecidos também como sistemas de controle com realimentação, no qual existe uma comparação entre a saída e a entrada de referência, a fim de reduzir o erro, por meio da diferença entre estes dois sinais, e obter a saída desejada (OGATA, 2010).

Figura 1 – Sistema de controle em malha fachada



Fonte: elaborada pelo autor.

Sendo:

- C, o controlador.
- G, o modelo da planta.
- (r, e, u, y), os sinais de referência, erro, controle e saída, respectivamente.

Para controlar um sistema, geralmente é necessário modelá-lo matematicamente ou aplicar algum método de identificação de sistemas, mas atualmente estão sendo desenvolvidas novas técnicas de controle, em que não é necessário modelar o sistema para controlá-lo, por exemplo a técnica utilizando *Data Driven*, na qual é possível controlar o sistema apenas com dados de entrada e saída, e técnicas utilizando Inteligência Computacional Aplicada (ICA), por exemplo a técnica apresentada neste trabalho (CAMPI *et al.*, 2002; BOEIRA; ECKHARD, 2019).

Os sistemas podem ser classificados de duas maneiras: não lineares e lineares invariantes no tempo. Se o sistema satisfizer o princípio da superposição e a propriedade da homogeneidade, ele é classificado como *Linear time-invariant* (LTI), caso contrário, não linear

(DORF; BISHOP, 2008).

As técnicas de controle para sistemas LTI são difundidas na literatura clássica, por esse motivo em diversos casos torna-se mais simples linearizar um sistema e aplicar uma técnica para sistemas LTI. Porém, nem sempre esse artifício pode ser aplicável, visto que ele é apenas eficiente operando em torno de um ponto de operação (DORF; BISHOP, 2008). Para evitar a linearização do sistema, pode ser utilizada uma técnica mais sofisticada para sistemas não lineares, por exemplo a teoria de Lyapunov, ou até mesmo um controle adaptativo.

1.1 Motivação

A dinâmica de um pêndulo invertido está presente diariamente na vida todas as pessoas. Desde algo simples como, por exemplo, andar, tem relação com o pêndulo invertido. Quando se constrói arranha-céus a física por trás desse problema é utilizada. A dinâmica não linear associada às plataformas *offshore* também está relacionada com o tema. Novas tecnologias com drones são feitas a partir de estudos do pêndulo invertido. Portanto, mesmo sendo um problema clássico de controle, ainda é bastante importante estudá-lo (ANH *et al.*, 2007; KAREEM, 1983; HEHN; D'ANDREA, 2011).

Dentro da área de *Machine Learning* (ML), Aprendizagem por Reforço (AR) se mostrou uma excelente ferramenta para diversos problemas utilizando ICA, como projetar robôs automáticos e mapear tráfegos de carro. Porém, para compreender melhor os principais conceitos do assunto, é importante começar com algo chamado "desafio de brinquedo". Com ele é possível aplicar as principais ideias de AR a um problema simples antes de ir para um projeto de alto nível. O pêndulo invertido em cima de um carrinho, por exemplo, funciona muito bem como um desafio de brinquedo

1.2 Objetivos

Em suma, o objetivo desse trabalho é aplicar uma técnica de inteligência computacional para controlar um sistema não linear, a técnica escolhida foi AR, devido a possibilidade de controlar o sistema sem precisar modelá-lo, e o sistema um pêndulo invertido em cima de um carrinho. Além disso, comparações com outro método mais convencional na literatura serão realizadas como, por exemplo, o dimensionamento de um controlador Regulador Quadrático Linear (LQR), do inglês, *linear quadratic regulator*.

Ao decorrer do trabalho, serão apresentados os principais conceitos sobre AR, AR profundo, Redes Neurais Artificiais (RNA's), a teoria de controle e o problema do pêndulo invertido. Todas as simulações e comparações serão feitas utilizando o *software* Matlab.

1.3 Organização do Trabalho

Este trabalho está organizado da seguinte forma:

- Capítulo 1: introduz um breve resumo sobre o tema, as motivações, objetivos e organização do trabalho.
- Capítulo 2: formaliza o problema abordado no trabalho.
- Capítulo 3: apresenta a fundamentação teórica para um entendimento melhor do assunto.
- Capítulo 4: apresenta a metodologia utilizada.
- Capítulo 5: discute os resultados obtidos com as simulações.
- Capítulo 6: aborda as conclusões com a realização dos objetivos e sugere possibilidades para trabalhos futuros.

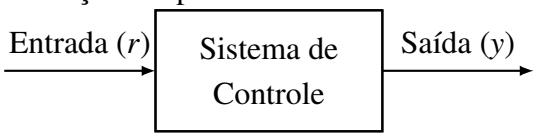
2 FORMALIZAÇÃO DO PROBLEMA

O problema neste trabalho consiste em controlar um sistema não linear. Por esse motivo, esta seção é voltada para formalizar o problema. Inicialmente aborda o tema sobre controle de sistemas dinâmicos e, posteriormente, elucida o pêndulo invertido.

2.1 Controle de Sistemas Dinâmicos

Pode-se definir um sistema de controle como um processo no qual, dada uma certa referência ou entrada, é possível obter uma saída com o desempenho e especificações desejadas. A Figura 2 representa um sistema de controle que recebe uma entrada especificada para alcançar a resposta desejada. Dentro desse sistema de controle, estão elementos relevantes sobre o assunto, o controlador e a planta, ambos explanados na Figura 1 (NISE, 2012).

Figura 2 – Descrição simplificada de um sistema de controle



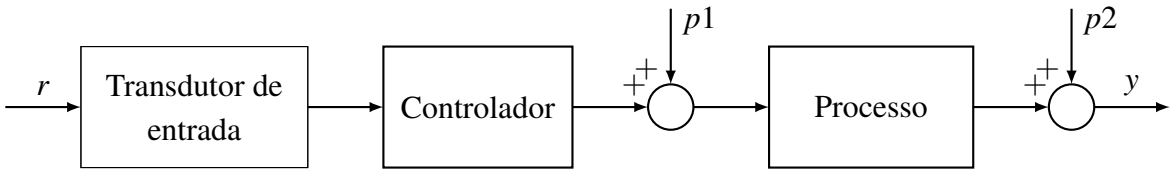
Fonte: elaborada pelo autor.

Sistemas de controle podem ser divididos em duas categorias: malha aberta e malha fechada. A maioria dos exemplos e projetos são modelados em malha fechada, devido a possibilidade de apresentar um desempenho satisfatório. Porém, em alguns casos, é mais vantajoso usar a configuração de malha aberta, principalmente porque são mais simples de serem montados e menos dispendiosos (OGATA, 2010).

A primeira e mais simples categoria, malha aberta, pode ser representada pela Figura 3. Um sistema em malha aberta é formado, basicamente, por três subsistemas: transdutor de entrada, controlador e processo (ou planta). O primeiro deles, transdutor de entrada, converte a forma da entrada, conhecida também como referência, para aquela utilizada pelo controlador. Este, por sua vez, aciona o processo ou planta do sistema. Com isso, é gerado o sinal de saída, ou variável controlada, e o sistema está controlado. Porém, não é possível compensar alguns sinais de perturbação, como os sinais $p1$ e $p2$ representados na Figura 3. Um exemplo de sistema em malha aberta bastante comum no dia a dia é o da torradeira, na qual a variável controlada é a torrada (NISE, 2012).

Embora os sistemas em malha aberta sejam simples de implementar e menos custo-

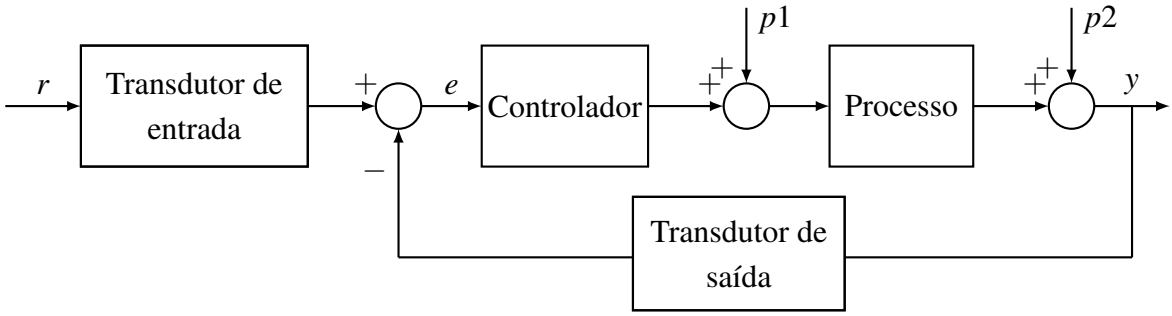
Figura 3 – Diagrama de blocos: sistema em malha aberta



Fonte: elaborada pelo autor.

sos, pois não há realimentação do sinal de saída, eles são insuficientes em relação a compensar e corrigir perturbações. Muitos desses problemas podem ser corrigidos pelo sistema na configuração em malha fechada, cujo modelo genérico é representado pela Figura 4. Comparando as duas configurações, nota-se que o sistema em malha fechado é fundamentalmente um sistema em malha aberta com um transdutor de saída e um sinal de realimentação. O transdutor de saída funcionará da mesma forma que o transdutor de entrada, convertendo o sinal de entrada para a forma utilizada pelo controlador. Caso o ganho dos dois transdutores seja unitário, haverá o sinal de erro (e), representado pela diferença entre o sinal de referência e o sinal de saída. Com essa estrutura o sistema consegue compensar os efeitos dos sinais de perturbação por meio da realimentação e do sinal de erro. Se o erro for zero, o processo fica no estado desejado (NISE, 2012; OGATA, 2010).

Figura 4 – Diagrama de blocos: sistema em malha fechada



Fonte: elaborada pelo autor.

2.1.1 Modelagem Matemática de Sistemas de Controle

Para realizar uma modelagem matemática de um sistema de controle, geralmente são utilizados dois métodos para um sistema LTI. O primeiro destes métodos é utilizando funções de transferência no domínio da frequência e o segundo método é utilizando equações de estado no domínio do tempo, ambos serão apresentados neste trabalho (NISE, 2012).

Em ambos os casos, o primeiro passo é conhecer leis básicas da física que são aplicados no sistema. Por exemplo, em problemas de circuitos elétricos são utilizadas as leis

de Kirchhoff e a lei de Ohm, por outro lado, em problemas mecânicos já são utilizadas as leis de Newton. Além disso, as equações diferenciais lineares invariantes no tempo são ferramentas essenciais para descrever a entrada e a saída do sistema. (NISE, 2012; OGATA, 2010).

Uma opção para modelar o sistema, caso não seja possível aplicar as leis da física, é utilizar métodos de identificação. Por exemplo métodos baseados no branqueamento do erro de predição, como o método dos mínimos quadrados recursivos, ou métodos baseados na não correlação do vetor de observação e no erro de predição, como o método da variável instrumental com modelo auxiliar (LANDAU; ZITO, 2006).

2.1.1.1 Funções de Transferência

O método mais clássico para modelar sistemas de controle é a técnica no domínio da frequência. Essa abordagem converte a equação diferencial que representa o sistema em uma função de transferência, dessa maneira é obtido um modelo que relaciona algebricamente a entrada com a saída (NISE, 2012).

Infelizmente, uma grande desvantagem desse método é a aplicação limitada: ela pode ser aplicada apenas em sistemas LTI, ou sistemas que sejam linearizados ao redor de um ponto de operação. Por outro lado, uma grande vantagem é a obtenção rápidas de algumas informações. Com a função de transferência do sistema, adquire-se dados sobre a estabilidade e resposta transitória (NISE, 2012).

A função de transferência de um sistema LTI pode ser definida como a relação entre a transformada de Laplace da saída e a transformada de Laplace da entrada. Considerando a Equação 2.1 e admitindo todas as condições iniciais nulas, é possível obter a função de transferência do sistema, sendo x é a entrada e y a saída (OGATA, 2010).

$$a_0 \overset{(n)}{y} + a_1 \overset{(n-1)}{y} + \ldots + a_{n-1}\dot{y} + a_n y = b_0 \overset{(m)}{x} + b_1 \overset{(m-1)}{x} + \ldots + b_{m-1}\dot{x} + b_m x \qquad (n \leq m) \tag{2.1}$$

Dessa forma, a função de transferência $G(s)$ pode ser representada pela Equação 2.2.

$$G(s) = \frac{\mathscr{L}[saida]}{\mathscr{L}[entrada]} = \frac{Y(s)}{X(s)} = \frac{b_0 s^m + b_1 s^{m-1} + \ldots + b_{m-1}s + b_m}{a_0 s^n + a_1 s^{n-1} + \ldots + a_{n-1}s + a_n} \tag{2.2}$$

O sistema é denominado de ordem n se a maior potência de s no denominador for n.

2.1.1.2 Espaço de Estados

Com o avanço da tecnologia, os sistemas de controle se tornaram cada vez maiores e mais complexos, por esse motivo abordagens mais clássicas, como a modelagem utilizando funções de transferência, se tornaram inviáveis para alguns problemas. Por meio do método de espaço de estados é possível analisar e projetar uma grande variedade de sistemas, desde sistemas não lineares que possuem saturação e zona morta, até mesmo sistemas com condições iniciais não nulas. Sistemas com múltiplas entradas e saídas, *multiple-input multiple-output* (MIMO), podem ser representados utilizando variáveis de espaço de estado, assim como sistemas com uma entrada e saída, *single-input single-output* (SISO) (NISE, 2012).

Embora seja utilizado em sistemas mais complexos, o método de espaço de estados pode ser utilizado para modelar e controlar sistemas LTI, mesmo que essa abordagem não seja tão intuitiva quanto a abordagem utilizando funções de transferência. Atualmente são disponibilizados diversos pacotes de programas que trabalham com espaço de estado. Dessa forma, muitos projetista optam por essa abordagem, mesmo para um problema mais simples (NISE, 2012).

Para utilizar essa técnica, é importante definir alguns conceitos:

- *Variável do sistema:* qualquer variável que responda a uma entrada do sistema;
- *Variáveis de estado:* conjunto com o menor número de variáveis que determinam o estado do sistema (uma vez dada a entrada para $t \geq t_0$ e o estado inicial $t = t_0$ é conhecido, todos os estados futuros do sistema são conhecidos);
- *Vetor de estado:* vetor no qual os elementos são as variáveis de estado;
- *Espaço de estados:* espaço n dimensional cujos eixos são as variáveis de estado;
- *Equações de estado:* equações diferenciais de primeira ordem com n variáveis, em que as n variáveis são as variáveis de estado;
- *Equação de saída:* equação algébrica que representa as variáveis de saída como uma combinação linear das variáveis de entrada e dos estados;

Com essas definições, fica mais intuitivo compreender as Equações 2.3 e 2.4, que modelam um sistema na representação de espaço de estados.

$$\dot{x} = Ax + Bu \tag{2.3}$$

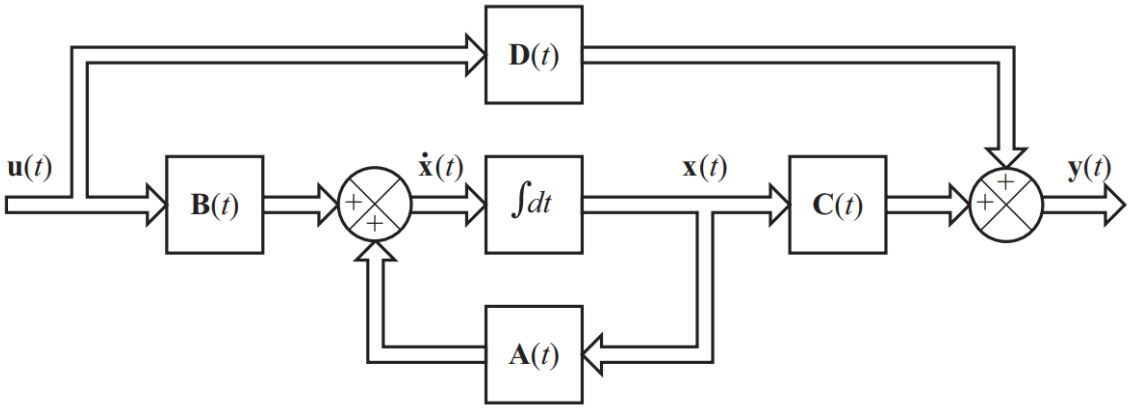
$$y = Cx + Du \tag{2.4}$$

Sendo:

- x = vetor de estado.
- $\dot{x}$ = derivada do vetor de estado em relação ao tempo.
- y = vetor de saída.
- u = vetor de controle.
- A = matriz do sistema.
- B = matriz de entrada.
- C = matriz de saída.
- D = matriz de transmissão direta.

Outra maneira de visualizar um sistema representado por espaço de estados é por meio de diagramas de blocos. A Figura 5, por exemplo, demostra um diagrama de blocos de um sistema LTI representado no espaço de estados (OGATA, 2010).

Figura 5 – Diagrama de blocos de um sistema representado no espaço de estados



Fonte: (OGATA, 2010).

2.2 O Problema do Pêndulo Invertido

Um dos problemas mais clássicos da teoria de controle é o pêndulo invertido. Ele pode até ser considerado uma porta de entrada para outros problemas mais complexos. Por exemplo, a movimentação de um robô humanoide pode ser modelada como algo parecido com o pêndulo invertido. Ele a todo momento começa a andar (cair para frente) e logo em seguida volta para posição vertical, embora instável, exatamente como a movimentação humana e um pêndulo invertido em cima de uma plataforma móvel. A ideia de uma estrutura se manter verticalmente num ponto de equilíbrio instável, sobre outra estrutura em movimento, é a ideia principal do pêndulo invertido em cima de um carrinho.

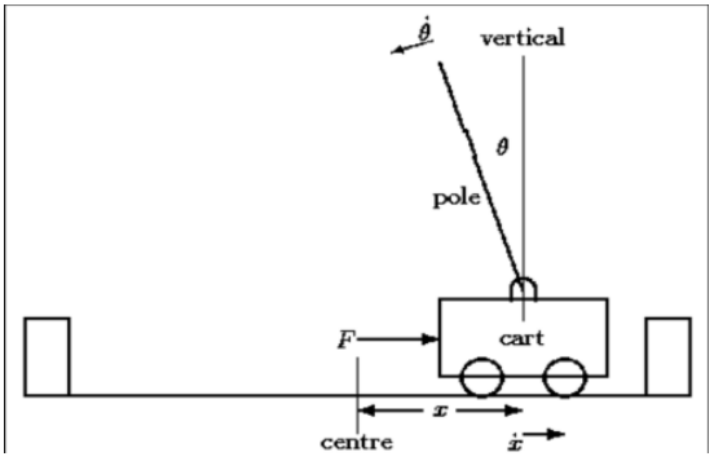
Na indústria *offshore*, por exemplo, estruturas quando há deslocamentos laterais significativos em resposta aos ventos e ondas, têm a comportamento dinâmico básico muito semelhante a um pêndulo invertido. Alguns exemplos dessas estruturas são: torres estaiadas, torres articuladas, plataformas de pernas atirantadas etc. Um motivo simples para essas estruturas ditas como complacentes apresentarem um comportamento similar a um pêndulo invertido é que elas são sistemas vibratórios flexíveis no plano horizontal e rígidos verticalmente (KAREEM, 1983).

Outra aplicação bastante conhecida utilizando a dinâmica de um pêndulo invertido é a construção de edifícios. Desde pequenos prédios de 4 ou 5 andares, até arranha-céus são modelados como pêndulos invertidos. Outro exemplo são os drones que transportam objetos, tanto em ambientes domésticos, quanto em ambientes industriais. Neste caso, o drone seria a plataforma se locomovendo na horizontal e o objeto uma estrutura na vertical(ANH *et al.*, 2007; HEHN; D'ANDREA, 2011).

2.2.1 Modelagem do Pêndulo Invertido

Existem incontáveis problemas que se resumem ao problema do pêndulo invertido. Para estudá-los, é importante conhecer o clássico problema proposto na Seção 2.2. A Figura 6 representa o pêndulo invertido em cima de um carrinho, ou em inglês como é conhecido, *cartpole problem*.

Figura 6 – Sistema de um pêndulo invertido em cima de um carrinho



Fonte: (NAGENDRA *et al.*, 2017).

Não existe uma única maneira para montar equações que modelem o sistema do

pêndulo invertido. Uma delas utiliza-se *Lagrange Equations*, ou seja, uma equação que relaciona a diferença entre a energia cinética e a energia potencial do sistema. A demonstração matemática não é trivial, mas o resultado obtido pode ser encontrado nas Equações 2.5 e 2.6 (KAFETZIS; MOYSIS, 2017).

$$D\ddot{\theta} + C\dot{\theta} + G = Hu \tag{2.5}$$

$$\dot{x} = \begin{bmatrix} 0 & I_2 \\ 0 & -D^{-1}C \end{bmatrix} x + \begin{bmatrix} 0 \\ -D^{-1}CG \end{bmatrix} + \begin{bmatrix} 0 \\ D^{-1}H \end{bmatrix} u \tag{2.6}$$

Sendo:

- θ = vetor com posição e ângulo.
- $\dot{\theta}$ = derivada do vetor com posição e ângulo.
- x = vetor de estado.
- $\dot{x}$ = derivada do vetor de estado em relação ao tempo.
- u = força aplicada no carrinho.
- I = momento de inércia do sistema.
- D = matriz que depende da posição e do ângulo.
- C = matriz que depende da posição e do ângulo.
- G = matriz que depende da posição e do ângulo.
- H = matriz constante.

Linearizando e fazendo algumas aproximações para pequenos valores de θ, como: $sen(\theta) = \theta$, $cos(\theta) = 1$ e $\dot{\theta}^2 = 0$, a Equação 2.5 se torna a Equação 2.7.

$$\begin{bmatrix} m_0 + m_1 & m_1\frac{L}{2} \\ m_1\frac{L}{2} & m_1\left(\frac{L}{2}\right)^2 + I \end{bmatrix} \ddot{\theta} + \begin{bmatrix} 0 & 0 \\ 0 & m_1 g\frac{L}{2} \end{bmatrix} \theta = Hu \tag{2.7}$$

Ou então na forma de espaço de estados:

$$\dot{x} = Ax + Bu \tag{2.8}$$

Sendo:

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \dfrac{gL^2m_1^2}{L^2m_0m_1 + 4I(m_0+m_1)} & 0 & 0 \\ 0 & -\dfrac{2gLm_1(m_0+m_1)}{L^2m_0m_1 + 4I(m_0+m_1)} & 0 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 \\ 0 \\ \dfrac{4I + L^2m_1}{L^2m_0m_1 + 4I(m_0+m_1)} \\ -\dfrac{2Lm_1}{L^2m_0m_1 + 4I(m_0+m_1)} \end{bmatrix}$$

$$x = \begin{bmatrix} \theta_0 & \theta_1 & \dot{\theta}_0 & \dot{\theta}_1 \end{bmatrix}^T$$

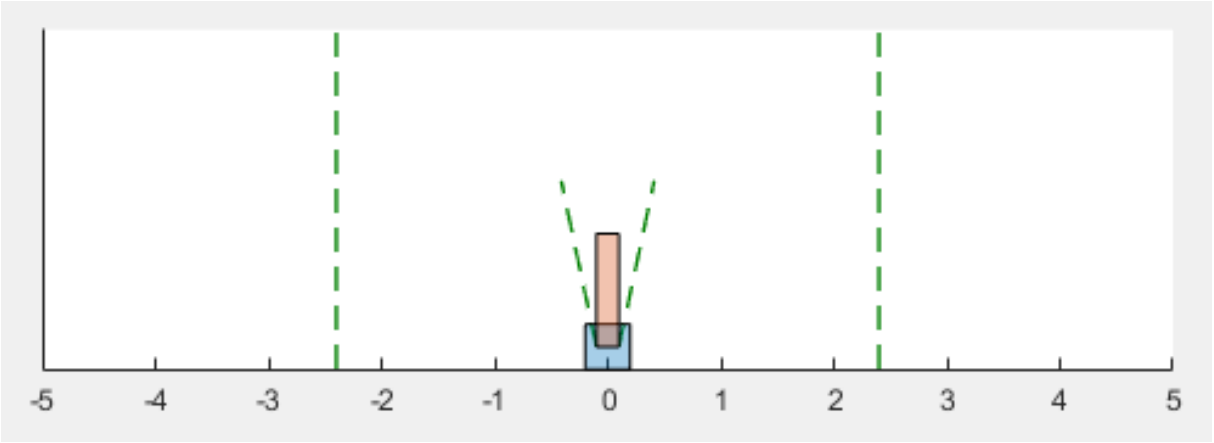
- θ_0 = distância do centro do carro a uma dada referência.
- θ_1 = ângulo de desvio da haste (pêndulo) na vertical.
- $\dot{\theta}_0$ = derivada de θ_0.
- $\dot{\theta}_1$ = derivada de θ_1.
- m_0 = massa do carro.
- m_1 = massa da haste.
- L = comprimento da haste.
- I = momento de inércia da haste.
- g = gravidade.

2.2.2 Simulação do Pêndulo Invertido

Todas as simulações do problema apresentado são feitas no *software* Matlab. O ambiente no qual o problema está inserido é criado por meio de um *script* e da *toolbox* de AR disponibilizada, representado pela Figura 7.

Dentre as duas modelagens para os sistemas de controle apresentadas na Seção 2.1.1, optou-se pela modelagem em variáveis de espaço de estados devido a possibilidade de utilizar o controlador LQR posteriormente.

Figura 7 – Ambiente criado pela *toolbox*



Fonte: elaborada pelo autor.

3 FUNDAMENTAÇÃO TEÓRICA

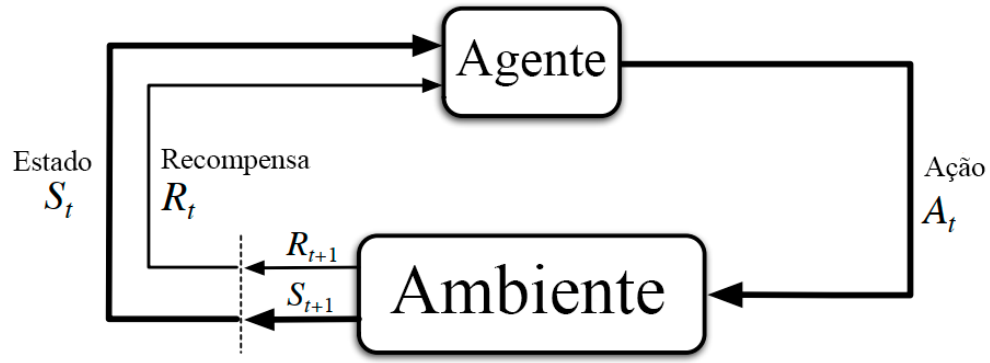
Para ter um entendimento melhor do trabalho realizado, esta seção é integralmente voltada para fundamentações técnicas sobre os assuntos tratados. Primeiramente é feito uma apresentação sobre AR, as RNA's e aprendizagem profundo. Em seguida, como esses assuntos se entrelaçam, além de apresentar o algoritmo *Deep Q-Network* (DQN). Por fim, será mostrado como AR pode ser usada para controlar sistemas dinâmicos.

3.1 Aprendizagem por Reforço

É uma das três grandes categorias de ML. Ela é uma forma de programar agentes, por meio de recompensas e punições, sem a necessidade de indicar como a tarefa deve ser realizada. Então o agente deve aprender a se comportar, utilizando o método de tentativa e erro, diante de um ambiente dinâmico (KAELBLING *et al.*, 1996).

Um modelo clássico de AR é constituído por um agente que está conectado ao ambiente por meio de percepção e ação, conforme representado pela Figura 8. A cada interação, o agente recebe alguma informação indicando como está o ambiente ou qual o estado do ambiente, a partir dessa informação, ele toma a decisão de realizar alguma ação. Esta ação, por sua vez, age sobre o ambiente alterando o estado dele. Então, o agente dessa vez recebe duas informações do ambiente: um sinal de percepção ou estado, como está o ambiente, e um sinal de reforço (ou recompensa), informando se a ação dele foi boa ou ruim. A partir desse ponto o ciclo de aprendizagem se inicia e o agente vai agir com o objetivo de aumentar esse sinal de reforço a longo prazo (KAELBLING *et al.*, 1996).

Figura 8 – Resumo de aprendizagem por reforço



Fonte: adaptado de (SUTTON; BARTO, 2018).

Resumidamente, o modelo consiste em:

1. Um conjunto discreto de estados, S;
2. Um conjunto discreto de ações, A;
3. Um conjunto discreto de sinais de recompensa, R;

A Tabela 1 respresenta um resumo dos conjuntos discretos aplicados ao problema do pêndulo invertido em cima do carrinho. A coluna da esquerda representa o conjunto discreto e a coluna da direita os respectivos elementos do conjunto.

Tabela 1 – Conjuntos discretos do pêndulo invertido em cima do carrinho

Conjunto discreto	**Elementos do conjunto**
Estados S	Posição, ângulo, derivada da posição e derivada do ângulo
Ações A	Locomoção para a esquerda e locomoção para direita
Recompensas R	1 e -5

Fonte: elaborada pelo autor.

Quando o agente recebe um sinal informando o estado S_t e a recompensa R_t, ele não obrigatoriamente vai escolher uma ação predeterminada, na verdade há uma probabilidade referente a cada ação A_t possível de realizar para o estado S_t. O mapeamento com as probabilidades de cada ação para um determinado estado é chamado de política, representada pela letra π. Dessa maneira, $\pi_t(a|s)$ representa a probabilidade do agente tomar a ação $A_t = a$ dado um estado $S_t = s$ (SUTTON; BARTO, 2018).

Durante o passar do tempo o agente vai construindo a própria política, ou seja, como ele busca as recompensas, assim surgem dois conceitos importantíssimos: *exploration* e *exploitation*. Existe um *trade-off* entre esses dois conceitos. Quando o agente dá prioridade para *exploration* significa que as recompensas que ele ganhou no passado não são muito relevantes para as possíveis ações que ele pode realizar a partir de um determinado estado, o objetivo dele agora é simplesmente explorar e conhecer novas ações que poderão dar recompensas maiores no futuro, dessa maneira ele poderá encontrar um máximo global. Por outro lado, caso o agente dê prioridade para *exploitation*, ele levará em consideração as ações tomadas no passado e suas respectivas recompensas, com o objetivo de obter recompensas maiores, porém essa recompensa não necessariamente é um máximo global, mas sim um máximo local. Por isso é importante equilibrar esses dois conceitos.

3.1.1 Recompensas

O objetivo do agente em AR não é obter a máxima recompensa na atual interação com o ambiente, mas sim maximizar o acumulo dessas recompensas durante todo o processo de aprendizagem. Esta ideia, na prática, provou-se ser bastante aplicável. Por exemplo, para ensinar um robô a coletar latas de refrigerantes vazias, pode-se atribuir 1 para cada tentativa de sucesso e 0 para cada tentativa de fracasso. Depois de várias tentativas, é possível somar os sinais de recompensa e obter uma recompensa acumulada. É importante indicar para o agente o que ele realmente deve fazer, não adianta um agente jogador de xadrez, por exemplo, receber diversas recompensas por capturar as peças do adversário e perder a partida no final. Então, nesse caso, a recompensa por ganhar a partida deve ser bem maior que capturar peças (SUTTON; BARTO, 2018).

A recompensa cumulativa que o agente recebe pode ser definida como retorno. Se um agente recebe recompensas R_{t+i} após um intervalo de tempo t $(R_{t+1}, R_{t+2}, R_{t+3}...)$, então o retorno será a soma dessas recompensas. A Equação 3.1 exemplifica este exemplo, em que G_t representa o retorno após t intervalos de tempo e R_T a recompensa final (SUTTON; BARTO, 2018).

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + ... + R_T \tag{3.1}$$

Quando existe a noção de um passo de tempo final entre a interação do agente e do ambiente, ou seja, quando o problema tem um fim, como no jogo de xadrez ou no jogo da velha, é possível definir todo esse conjunto de interações como um episódio. Ao final de cada episódio, o ambiente se encontra num estado específico e um novo episódio começa independente do anterior. Porém, existem casos em que não é possível dividir em episódios identificáveis, pois continua sem limites. Estas tarefas são chamadas de contínuas. Como T tende ao infinito, isso seria um problema, pois G_t também tenderia ao infinito. Para contornar este problema, foi criado um parâmetro chamado de desconto. Ele determina se as recompensas futuras são significativas ou não para o agente. A Equação 3.2 acrescenta o novo parâmetro na Equação 3.1, em que γ representa o desconto que varia de 0 a 1 (SUTTON; BARTO, 2018).

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k1} \tag{3.2}$$

Quanto maior o valor de γ, mais significativas serão as recompensas futuras para o agente, caso contrário para quanto menor for o valor de γ. Se γ tende a 1, a recompensa instantânea não tem muito impacto sobre a recompensa futura. Se γ tende a 0, a recompensa instantânea tem bastante impacto sobre a recompensa futura (SUTTON; BARTO, 2018; PESSOTTI, 2022).

3.1.2 A Propriedade de Markov

Em AR, uma das características principais é a propriedade de Markov. Ela afirma que o estado futuro, assim como a recompensa, depende apenas do estado e da ação atual. É chamado *Markov Decision Processes* (MDP), ou processo de decisão de Markov, todo processo em que a propriedade de Markov está presente. A Equação 3.3 representa a dinâmica de um MDP (SUTTON; BARTO, 2018).

$$p(s',r|s,a) \doteq Pr\{S_t = s', R_t = r | S_{t-1} = s, A_{t-1} = a\} \tag{3.3}$$

Dado um estado s e uma ação a, pertencentes a um conjunto discreto $\mathscr{S}$ e $\mathscr{A}$, respectivamente, a probabilidade de se obter um estado s' e uma recompensa r no próximo intervalo de tempo é dada pela Equação 3.3. A variável r também pertence a um conjunto discreto, $\mathscr{R}$. Por esse motivo, como p especifica a probabilidade para cada escolha de a e s, então o somatório de todas essas probabilidades é igual a 1, conforme representado na Equação 3.4 (SUTTON; BARTO, 2018).

$$\sum_{s' \in \mathscr{S}} \sum_{r \in \mathscr{R}} p(s',r|s,a) = 1 \tag{3.4}$$

Além disso, com a propriedade de Markov é possível modelar a recompensa para um par de estado-ação, conforme representado na Equação 3.5, em que $\mathbb{E}$ representa o valor esperado de uma variável aleatória (SUTTON; BARTO, 2018).

$$r(s,a) \doteq \mathbb{E}[R_t | S_{t-1} = s, A_{t-1} = a] = \sum_{r \in \mathscr{R}} r \sum_{s' \in \mathscr{S}} p(s',r|s,a) \tag{3.5}$$

3.1.3 Função Valor

Para estimar o quão bom é para o agente estar num determinado espaço, geralmente os algoritmos de AR utilizam funções de valor. Elas estimam recompensas futuras quem podem

ser esperadas, para ser mais preciso, o retorno. As recompensas esperadas dependem das ações que o agente vai tomar, ou seja, da política (SUTTON; BARTO, 2018).

A política pode ser definida como o mapeamento das probabilidades de um agente escolher uma certa ação a a partir de um estado s. Se um agente segue uma política π no instante t, então $\pi(a|s)$ é a probabilidade, tal que $A_t = a$ e $S_t = s$ (SUTTON; BARTO, 2018).

Uma função valor de um estado s sob uma política π, $v_\pi(s)$, pode ser representado pela Equação 3.6.

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s\right] \tag{3.6}$$

De maneira similar, tomando uma ação a sob uma política π no estado s, pode-se definir $q_\pi(s,a)$ como o retorno esperado, conforme a Equação 3.7.

$$q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a\right] \tag{3.7}$$

É possível utilizar experimentos para estimar os valores de $v_\pi(s)$ e $q_\pi(s,a)$. Se um agente segue uma política π e mantém uma média de retorno para cada estado s encontrado, então a média converge para o valor do estado, $v_\pi(s)$, como o número de vezes esse estado encontrado se aproxima do infinito. Se médias separadas forem mantidas levando em consideração cada ação a tomada pelo agente em cada estado, então essas médias convergirão de forma semelhante para os valores da ação, $q_\pi(s,a)$ (SUTTON; BARTO, 2018).

Diante de incontáveis possíveis ações e estados, torna-se praticamente inviável representar as funções de valor na forma de uma tabela, como no algoritmo *Q-learning*. Para contornar esse problema, foi desenvolvido um algoritmo utilizando RNA's que estimam os valores das funções, o DQN, discutido melhor na Seção 3.3.

3.2 Redes Neurais Artificiais e Aprendizagem Profunda

As RNA's são modelos computacionais baseados numa das principais células do sistema nervoso, o neurônio. Podem ser classificadas como um conjunto de processamento de informações determinado por neurônios artificiais interligados por diversas interconexões chamadas de *sinapses artificiais*, representadas matematicamente por matrizes de pesos sinápticos (SILVA *et al.*, 2010). Elas se assemelham demasiadamente ao cérebro, visto que não só o

conhecimento é adquirido por meio do ambiente e de um processo de aprendizagem, mas também os pesos sinápticos são utilizados como unidade de armazenamento para o conhecimento adquirido (HAYKIN, 1999).

A primeira publicação relacionada ao tema foi feita por McCulloch e Pitts na década de 40, na qual os autores fizeram a modelagem matemática de um neurônio biológico, resultando na concepção de um neurônio artificial (MCCULLOCH; PITTS, 1943). Anos depois o neurofisiológico Hebb desenvolveu o primeiro método de treinamento para as RNA's (HEBB, 1949). A partir desses dois eventos, se originaram inúmeros estudos sobre neurônios artificiais e RNA's.

Dentre as diversas características das RNA's, as principais são: adaptação por experiência, capacidade de aprendizagem, habilidade de generalização, organização de dados e tolerância a falhas. Dessa forma, as RNA's têm seus pesos sinápticos adaptados quando aprensentadas a exemplos relacionados ao processo, além de extrair o relacionamento entre as diversas variáveis. Quando treinada, a rede tem a capacidade de estimar soluções antes desconhecidas. Como os neurônios artificiais estão excessivamente interligados, a robustez é garantida para o modelo (SILVA *et al.*, 2010).

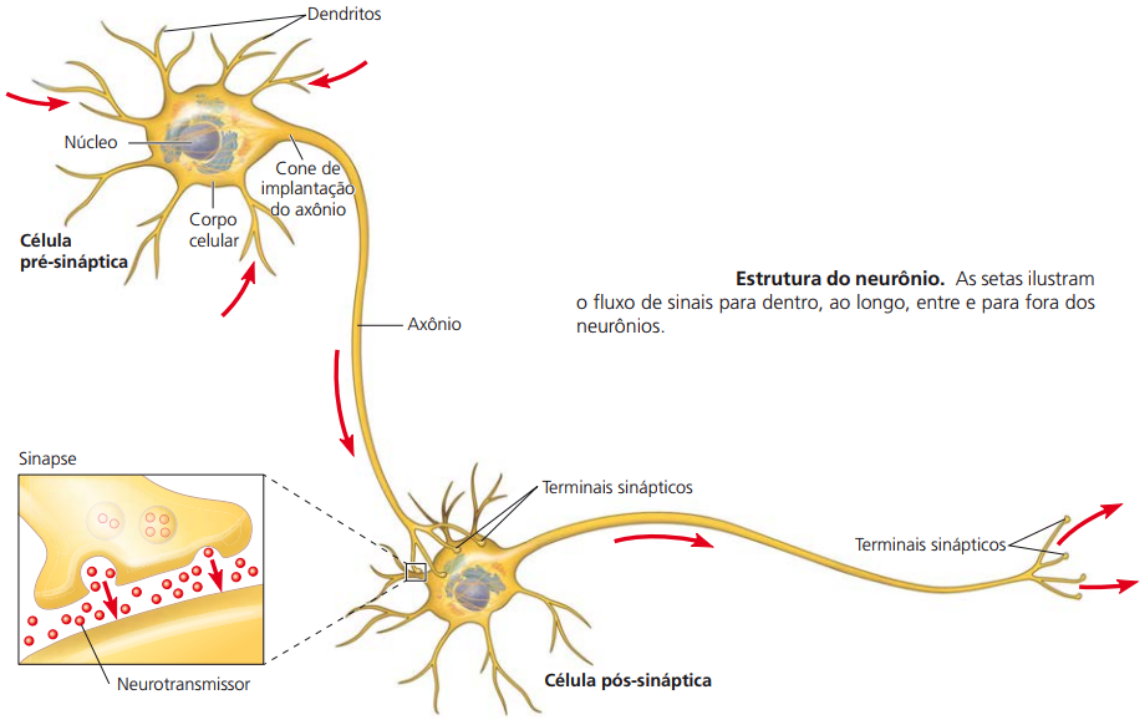
Há numerosas áreas dentro das engenharias nas quais RNA's podem ser aplicadas, como, por exemplo, reconhecimento/classificação de padrões, agrupamento de dados (clusterização), sistemas de previsão, etc. Porém a aplicação mais útil das RNA's para este trabalho é o controle de processos. Assim, a rede busca o controle seguindo os requisitos de qualidade, eficiência e robustez. Alguns sistemas em que elas são empregadas são controles em robótica, aeronaves, elevadores, eletrodomésticos, etc (SILVA *et al.*, 2010). Neste trabalho em específico, será usada para controlar um sistema com pêndulo invertido.

3.2.1 O Neurônio Biológico

Dentre os diversos sistemas presentes no organismo humano, um dos que mais se destaca é o sistema nervoso. Os neurônios são, definitivamente, as células mais importantes nesse sistema, permitindo que os organismos que os possuem sejam capazes de sentir o ambiente e responder rapidamente. A Figura 9 representa um neurônio biológico e suas principais partes: corpo celular, dendritos e axônio (REECE *et al.*, 2015).

Dentro do corpo celular estão as principais organelas do neurônio, incluindo o núcleo, processando todas as informações que chegam até ele. Os dendritos, finos prolongamentos que forma uma espécie de "árvore", estão juntos ao corpo celular recebendo sinais de outros neurônios.

Figura 9 – Representação de um neurônio biológico



Fonte: (REECE *et al.*, 2015).

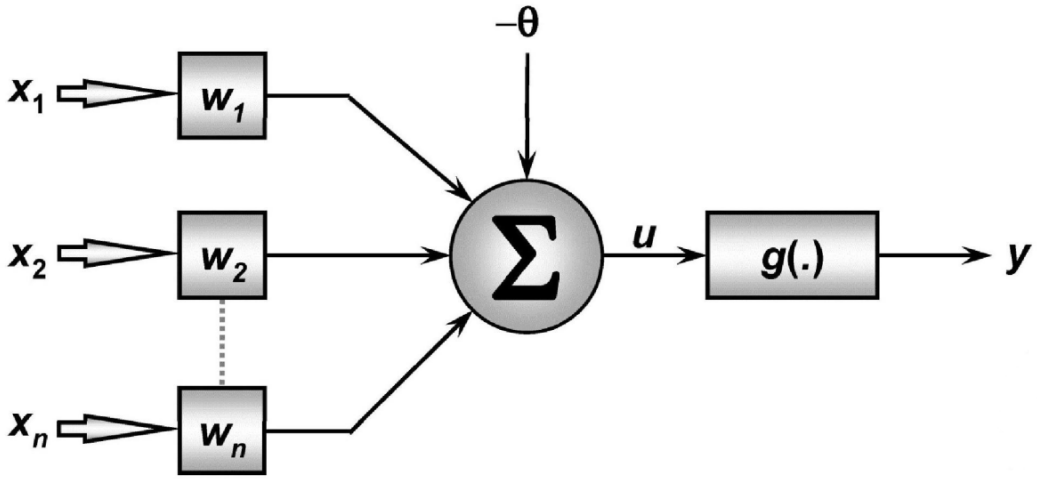
O axônio é uma única extensão do neurônio que transmite sinais para as outras células. A região na qual o axônio transmite informação para outra célula é chamada de sinapse, ou seja, onde os neurotransmissores passam informação do neurônio para a célula receptora. Com essa estrutura, é formada uma grande rede neural com 100 bilhões de neurônios. Além disso, deve-se mencionar que não há contato físico entre os neurônios na sinapse, apenas liberação de neurotransmissores (REECE *et al.*, 2015; HODGKIN; HUXLEY, 1952).

Nos neurônios, assim como em outras células, os íons são distribuídos de maneira desigual entre o interior e exterior da célula, sendo o interior carregado negativamente em ralação ao exterior. Essa diferença de cargas é chamada de potencial de membrana. Em neurônios em repouso, por exemplo, aqueles que não estão enviando nenhum sinal, o potencial de repouso está entre -60 e -80 milivolts. Perturbações ou estímulos de outros neurônios causam mudança no potencial de membrana que agem como sinais, transmitindo a informação. Os principais íons que desempenham o papel essencial na formação do potencial de membrana são o K^+ e o Na^+ (REECE *et al.*, 2015).

3.2.2 O Neurônio Artificial

Inspirado na biologia dos neurônios do sistema nervoso, foram iniciadas pesquisas com o objetivo de desenvolver um modelo matemático que resolvesse diversos tipos de problemas, assim foi concebido o neurônio artificial representado pela Figura 10.

Figura 10 – Representação de um neurônio artificial



Fonte: (SILVA *et al.*, 2010).

Os sinais de entrada são representados pelo conjunto $\{x_1, x_2, ..., x_n\}$, fazendo uma analogia aos impulsos elétricos recebidos pelos dendritos. O conjunto $\{w_1, w_2, ..., w_n\}$ representa os pesos sinápticos, ou no neurônio biológico, representariam as junções sinápticas. Os demais elementos básicos de um neurônio artificial representado pela Figura 10 são:

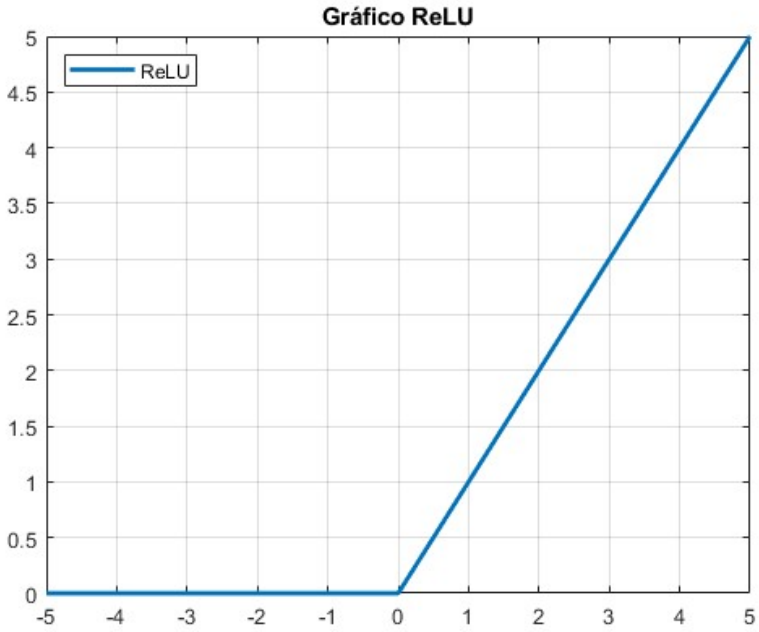
1. Combinador linear (Σ): soma todos os sinais de entrada que foram ponderados pelos pesos sinápticos.
2. Liminar de ativação ou bias (θ): serve como um parâmetro para que o resultado do combinador linear possa gerar um valor de disparo em direção à saída do neurônio.
3. Potencial de ativação (u): resultado da diferença entre a saída do combinador linear e o limiar de ativação. Caso $u \geq 0$, então o neurônio produz um potencial estimulante, caso contrário, um potencial inibitório.
4. Função de ativação (g): limita o sinal de saída dentro de um intervalo desejado.
5. Sinal de saída (y): sinal final produzido pelo neurônio artifical, podendo ser utilizado por outros neurônios em sequência.

As funções de ativação são normalmente divididas em funções parcialmente diferenciáveis ou funções totalmente diferenciáveis. Alguns exemplos de funções parcialmente

diferenciáveis que podem ser utilizadas como funções de ativação são: função degrau, função degrau bipolar ou função sinal e função rampa simétrica. Por outro lado, exemplos de função totalmente diferenciáveis que frequentemente são utilizadas como funções de ativação são: função logística, função tangente hiperbólica e função gaussiana (SILVA *et al.*, 2010).

Neste trabalho, em específico, será utilizada a função de ativação *Rectified Linear Unit* (ReLU), devido a facilidade de implementação e presença em outros problemas envolvendo AR, representada pela Figura 11. Ela é definida como $f(x) = max(x, 0)$, ou seja, retorna o valor de x para $x \geq 0$ e 0 para $x < 0$ (RAMACHANDRAN *et al.*, 2017; PESSOTTI, 2022).

Figura 11 – Representação função *rectified linear unit*



Fonte: elaborada pelo autor.

Em suma, o neurônio artificial atua da seguinte forma: é apresentado um conjunto de valores que representam os sinais de entrada; estes, por sua vez, serão multiplicados pelos respectivos pesos sinápticos; dessa forma, é obtido um potencial de ativação que será subtraído pelo limiar de ativação; depois é aplicada uma função de ativação apropriada, a fim de limitar a saída do neurônio; por fim, têm-se o sinal de saída do neurônio (SILVA *et al.*, 2010).

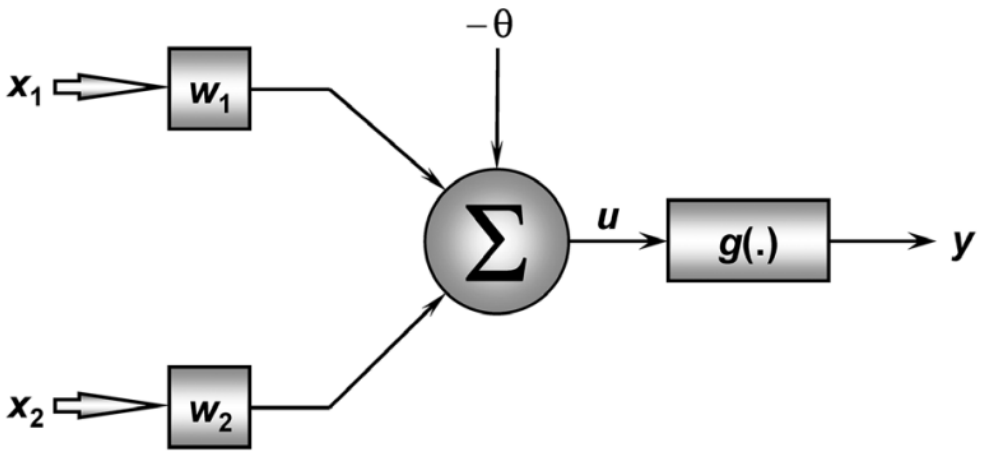
3.2.3 Perceptrons de Camada Única

A concepção mais simples dentre as RNA's é a rede perceptron. Ela é constituída por apenas um elemento, o perceptron, e sua aplicação inicial consistia em identificar padrões

geométricos. A Figura 10, por exemplo, pode representar uma rede perceptron com n entradas e apenas uma saída. Além disso, nela percebe-se também que esse tipo de rede tem uma arquitetura *feedforward* de camada única, ou seja, o fluxo de informação sempre reside no sentido da da camada de entrada em direção à camada de saída, sem nenhum tipo de realimentação (ROSENBLATT, 1958; SILVA *et al.*, 2010).

O princípio de funcionamento da rede perceptron é basicamente o que já foi apresentado na Seção 3.2.2, porém com apenas um neurônio na rede. Para exemplificar, consideremos a rede perceptron da Figura 12 com apenas duas entradas, além da função de ativação do tipo sinal, ou seja, a saída só pode ter o valor 1 ou -1.

Figura 12 – Exemplo de uma rede perceptron



Fonte: (SILVA *et al.*, 2010).

Fazendo uma análise matemática, é possível chegar na equação que representa a saída da rede:

$$y = \begin{cases} 1, & \text{se } w_1 \cdot x_1 + w_2 \cdot x_2 - \theta \geq 0 \\ -1, & \text{se } w_1 \cdot x_1 + w_2 \cdot x_2 - \theta < 0 \end{cases} \tag{3.8}$$

As desigualdades na Equação 3.8 representam expressões lineares que podem formar a fronteira de decisão, caracterizada por uma reta, para um perceptron com duas entradas. Portanto, a rede perceptron com apenas um único neurônio age como um classificador de padrões, cujo objetivo é dividir classes que sejam linearmente separáveis (SILVA *et al.*, 2010).

Se a saída do neurônio não for o valor desejado, então será necessário modificar os pesos sinápticos utilizando algum método de aprendizagem, por exemplo a regra de aprendizagem

de Hebb (HEBB, 1949). Então, os novos valores para os pesos sinápticos podem ser obtidos pela expressão representada na Equação 3.9.

$$w \leftarrow w + \eta \cdot (d^k - y) \cdot x^k \tag{3.9}$$

Sendo w é o vetor com o bias e os pesos, x^k é o vetor com a *k-ésima* amostra de treinamento, d^k é o valor desejado para a *k-ésima* amostra de treinamento, y é a saída do neurônio e η um fator de aprendizagem que está no intervalo em $0 < \eta < 1$. Assim, basta utilizar a Equação 3.9, dentro de um loop no algoritmo para atualizar os pesos sinápticos, até obter um conjunto de valores desejados (SILVA *et al.*, 2010).

3.2.4 Perceptrons de Multicamadas

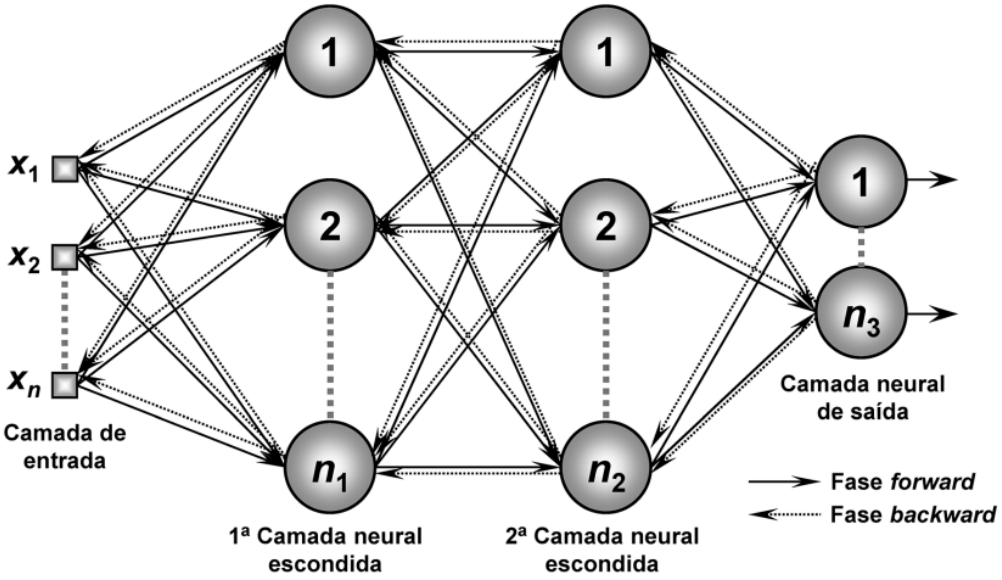
Normalmente as redes de perceptrons com multicamadas são chamadas de redes Multilayer Perceptron (MLP), as quais podem ser consideradas como um caso generalizado do perceptron de camada única. O treinamento da rede é de forma supervisionada, ou seja, é dado um conjunto com entradas e saídas desejadas para ajustar os pesos sinápticos, junto com o algoritmo de retropropagação de erro (RUMELHART; MCCLELLAND, 1987; HAYKIN, 1999).

Considerando o exemplo da Figura 13, tem-se uma rede MLP com x_n sinais de entrada, n_1 neurônios na primeira camada neural escondida, n_2 neurônios na segunda camada neural escondida e n_3 neurônios na camada neural de saída. O algoritmo de treinamento, *back-propagation*, será aplicado neste exemplo para facilitar o entendimento, as setas representam os dois sentidos do algoritmo, *forward* e *backward*.

A primeira parte do algoritmo é aplicada, chamada de propagação adiante ou *forward*. Nela é separado, a partir de uma amostra para treinamento, um conjunto de sinais de entrada $\{x_1, x_2, ..., x_n\}$ que são inseridos nas entradas da rede MLP, sendo propagados até as saídas da rede. O objetivo da aplicação dessa etapa visa, portanto, apenas obter um conjunto de dados de saída, sem ajustar nenhum valor dos pesos sinápticos ou modificar os limiares dos neurônios.

Posteriormente, com o conjunto de dados de saída obtido, há uma comparação com um conjunto de saídas desejada para o conjunto de sinais de entrada aplicado na primeira parte do algoritmo, vale ressaltar que esse artifício só é possível pois trata-se de um processo supervisionado. Supondo uma rede MLP como a da Figura 13, com n_3 neurônios de saída, o algoritmo calcula todos os n_3 erros (desvios) da rede. Dessa forma, os pesos sinápticos e

Figura 13 – Exemplo de uma rede *multilayer perceptron*



Fonte: (SILVA *et al.*, 2010).

bias podem ser ajustados, começando, portanto, a segunda parte de algoritmo, chamada de propagação reversa (*backward*). Diferentemente da primeira parte, todas as alterações desses elementos (pesos sinápticos e bias) são feitas, porém não no sentido convencional da rede neural, mas sim no sentido oposto, da camada de neurônios de saída, para a 1ª camada de neurônios escondida.

Resumidamente, aplicações de sucessivas das duas etapas do algoritmo, *forward* e *backward*, farão o ajuste automaticamente dos pesos sinápticos e limiares dos neurônios a cada iteração. Com isso, o somatório dos erros diminuirá e a rede MLP estará treinada.

3.2.5 Método de Otimização

Após a aplicação do algoritmo *forward* e obtenção dos erros, é possível calcular o erro quadrático médio. Com isso, para realizar o algoritmo *backward* e diminuir os erros, geralmente se utiliza uma função custo, por exemplo representada pela Equação 3.10. Dessa maneira, o aprendizado é obtido por meio de algum algoritmo de otimização que processe os dados desejados e obtido (BASHEER; HAJMEER, 2000; KRUL, 2021).

$$J = \frac{1}{N} \sum_{N=1}^{N} E(N) \tag{3.10}$$

Em que N representa o número de amostras e $E(N)$ o erro quadrático médio.

O algoritmo de otimização utilizado neste trabalho será o Adam. Ele é um método de otimização estocástica que requer gradientes de primeira ordem. A partir do primeiro e segundo momento dos gradientes, o método calcula os coeficientes de aprendizagem. O momento de primeira ordem representa a média do gradiente, Equação 3.11, e o segundo momento representa a variância não centrada do gradiente, Equação 3.12 (KINGMA; BA, 2015).

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \tag{3.11}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \tag{3.12}$$

Em que β representa a taxa de decaimento exponencial para o momento, geralmente valores entre 0 e 1 e o parâmetro g representa o gradiente da função custo.

Após obter os valores para os dois momentos, é necessário fazer uma pequena correção: multiplicar por um fator de correção. Este fator e dado pela Equação 3.13, em que β_i representa a respectiva taxa de decaimento.

$$\frac{1}{1-\beta_i} \tag{3.13}$$

Dessa maneira, o primeiro momento corrigido pode ser representado pela Equação 3.14 e o segundo momento corrigido pela Equação 3.15.

$$\hat{m}_t = \frac{m_t}{1-\beta_1} \tag{3.14}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2} \tag{3.15}$$

Por fim, é possível obter os novos valores para os pesos por meio da Equação 3.16. Em que α representa um parâmetro para convergência e ε um fator que tende a zero.

$$w_{t+1} = w_t - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon} \tag{3.16}$$

3.3 Algoritmo *Deep Q-Network*

Existem diversos algoritmos para resolver problemas envolvendo AR. Como, por exemplo, o *Hill Climbing*, que busca uma política ótima para o agente. O *REINFORCE*, que tem o objetivo de maximizar uma função de recompensa que envolva o estado presente, a ação presente e o próximo estado. O clássico *Q-Learning*, que procura a melhor ação possível a ser tomada, dado o estado atual. O algoritmo utilizado nesse trabalho será o DQN, uma abordagem utilizando *Deep Learning* (DL) que toma elementos do método *Q-Learning* e acrescenta RNA's (KRUL, 2021; MNIH *et al.*, 2015).

Infelizmente AR pode apresentar vários desafios do ponto de vista do DL. Por exemplo, diversas aplicações bem-sucedidas utilizando DL exigem grandes quantidades de dados de treinamento rotulados manualmente. Em AR, por outro lado, o agente precisa aprender utilizando apenas sinais escalares de reforço ou recompensa. Além disso, algoritmos que utilizam DL muitas vezes partem do princípio que as amostras de dados são independentes, em AR os dados normalmente são sequências de estados correlacionados (MNIH *et al.*, 2015).

Porém, uma maneira de contornar esses problemas é utilizando RNA's para aprender políticas de controle bem-sucedidas. A rede é treinada utilizando uma variação do algoritmo *Q-Learning*, com gradiente descendente estocástico para atualizar os pesos sinápticos (MNIH *et al.*, 2015).

Inicialmente, é definida uma função valor ótima que busca obter um retorno máximo após ver alguns estados s e realizar algumas ações a, conforme a Equação 3.17, em que π representa a política de mapeamento de sequências para ações. A função valor ótima segue a identidade conhecida como Equação de Bellman. Ela é baseada na seguinte intuição: se o valor ótimo $Q^*(s',a')$ da sequência s' no próximo passo de tempo for conhecido por todas as possíveis ações a', então a estratégia é selecionar a ação a' que maximize o valor esperado de $r+\gamma Q^*(s',a')$, conforme a Equação 3.18. Sendo ε o ambiente, γ o fator de desconto e R_t o retorno futuro descontado (MNIH *et al.*, 2015).

$$Q^*(s',a') = max_\pi \mathbb{E}[R_t|s_t = s, a_t = a, \pi] \tag{3.17}$$

$$Q^*(s',a') = \mathbb{E}_{s'\sim\varepsilon}[r+\gamma max Q^*(s',a')|s,a] \tag{3.18}$$

A ideia é estimar uma função ação-valor usando a Equação de Bellman com um

incremento iterativo, conforme a Equação 3.19. Tal que, o valor iterativo convirja para um valor ótimo, à medida que as iterações tendam em direção ao infinito. Para que isso seja possível, é comum usar um aproximador de função para estimar a função valor-ação, ou seja, $Q(s,a;\theta) \approx Q^*(s,a)$ (MNIH *et al.*, 2015).

$$Q_{i+1}(s,a) = \mathbb{E}[r + \gamma max_{a'} Q_i(s',a')|s,a] \tag{3.19}$$

Este aproximador pode ser, por exemplo, uma rede neural. Pode-se referir uma rede neural com pesos sinápticos θ, quando utilizada como aproximador de função, como uma *Q-network*. Uma *Q-network* pode ser treinada minimizando uma sequência de funções de perda que muda a cada iteração, conforme representada pela Equação 3.20 (MNIH *et al.*, 2015).

$$L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(.)}\left[(y_i - Q(s,a;\theta_i))^2\right] \tag{3.20}$$

Sendo y_i o valor objetivo para a iteração i e ρ é uma distribuição de probabilidade sobre sequências s e ações a chamadas de distribuição de comportamento. Para minimizar o valor da Equação 3.20, é computacionalmente conveniente utilizar o método de gradiente descendente estocástico (MNIH *et al.*, 2015).

Para utilizar o agente DQN, é necessário seguir passo a passo:

1. Criar o ambiente.
2. Configurar o ambiente.
3. Criar a rede neural.
4. Criar o agente.
5. Configurar o agente.
6. Configurar as opções de treinamento.
7. Treinar o agente.
8. Simular o agente.

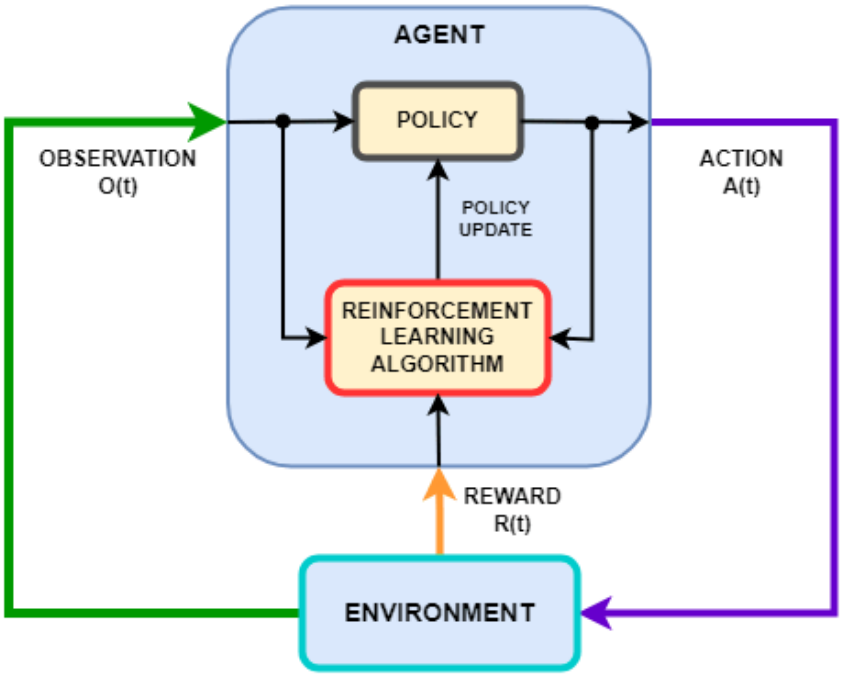
Exemplos de códigos com agentes do tipo DQN podem ser encontrados nos Apêndices B e C.

3.4 Controle Utilizando Aprendizagem por Reforço

A técnica de inteligência computacional AR é capaz de resolver diversos tipos de problemas. Por exemplo: supondo um jogo de xadrez, o jogador 1 faz sua jogada e o tabuleiro fica com uma configuração específica, agora é a vez do jogador 2. Dependendo das situações conhecidas previamente por ele, o jogador 2 estará mais ou menos preparado para fazer uma boa jogada. Utilizando AR, seria possível conhecer diversos desfechos para a jogada da jogador número 2 (SUTTON; BARTO, 2018).

Dentre as diversas possibilidades de utilização para AR, uma dela é projetar um controle adaptativo. Cada conceito fundamental sobre o tema AR pode ser "traduzido"para a linguagem de controle e, dessa forma, controlar um sistema dinâmico. A Figura 14 mostra resumidamente como funciona a técnica de AR e a Figura 15 "traduz"esses conceitos num sistema de controle (SUTTON; BARTO, 2018).

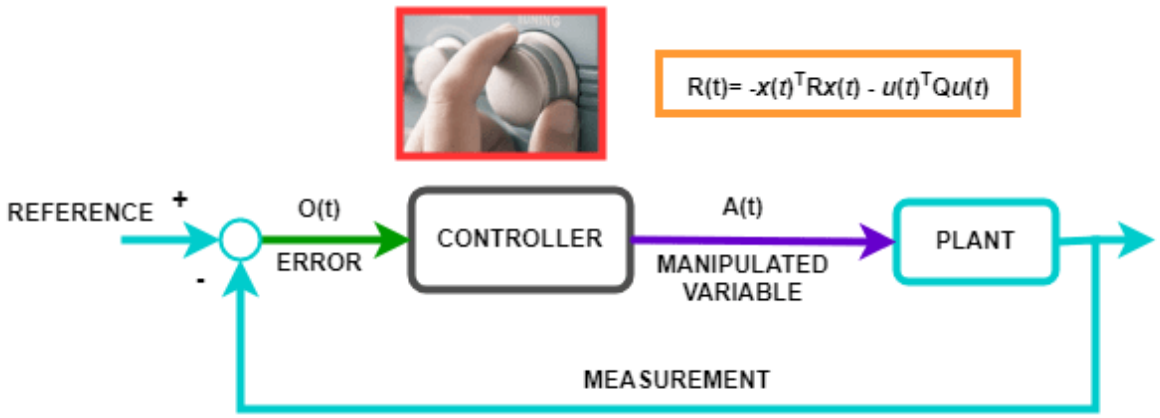
Figura 14 – Representação simplificada da técnica de aprendizagem por reforço



Fonte: MathWorks.

A Tabela 2 mostra um resumo de como os conceitos de AR podem ser aplicados em sistemas de controle.

Figura 15 – Controle utilizando aprendizagem por reforço



Fonte: MathWorks.

Tabela 2 – Aprendizagem por reforço em sistemas de controle

Aprendizagem por Reforço	Sistema de Controle
Política (Policy)	Controller (Controle)
Ambiente (Environment)	Quase tudo que não é o controle. Planta, medição, referência etc.
Estado (Observation)	Qualquer medida que pode ser vista pelo agente, por exemplo, sinal de erro.
Ação (Action)	Sinal de saída do controlador
Recompensa (Reward)	Alguma métrica de performance, por exemplo, sinal de erro.
Algoritmo de Aprendizagem (Learning Algorithm)	Mecanismo de adaptação.

Fonte: elaborada pelo autor.

4 METODOLOGIA

O controle de um pêndulo invertido é um ótimo exemplo para testar a estratégia de AR, pois ajuda a se familiarizar com os principais conceitos sobre o assunto e permite a desenvolver projetos futuros bem mais complexos. Atualmente existem diversas maneiras de elaborar e reproduzir testes de AR no computador, por exemplo Python e Matlab. Neste trabalho será utilizado como ferramenta computacional o Software Matlab, pois a Universidade Federal do Ceará disponibiliza o acesso livre para todos os alunos.

Inicialmente será elaborado um controle convencional para controlar o sistema do pêndulo invertido, neste caso foi escolhido um controlador LQR, devido a possibilidade de ser desenvolvido a partir de um modelo espaço de estados. O objetivo deste controlador é apenas para fins de comparação com o controle utilizando AR. Posteriormente serão desenvolvidos outros dois controles utilizando AR.

4.1 Controle LQR

Dado um sistema representado na forma de variáveis de espaço de estados, o método de controle quadrático ótimo, LQR, consiste em minimizar uma função do tipo custo representada pela Equação 4.1, com o objetivo de determinar a matriz K do controle ótimo. Dessa maneira, a lei de controle pode ser encontrada pela Equação 4.2.

$$J = \int_0^\infty (xQx^T + u^T Ru)\, dt \tag{4.1}$$

$$u = -Kx \tag{4.2}$$

Em que Q e R são matrizes do tipo simétrica, definida positiva e semidefinida positiva, que determinam a importância relativa do erro e o consumo dessa energia. No exemplo do pêndulo invertido, existem limites para o deslocamento no eixo horizontal, 2,4 metros para a esquerda e para a direita, e no ângulo com a vertical, $0,2094 rad$, por esse motivo serão escolhidos valores maiores para a matriz Q. Em relação aos outros dois estados, velocidade do carro e derivada do ângulo, não existe nenhuma limitação particular nesse exemplo, por isso as entradas correspondentes na matriz Q serão zero. Colocando um valor maior para o segundo estado,

ângulo com a vertical, permitirá obter ângulos menores e retornar o sistema para o equilíbrio mais rápido.

Para iniciar a construção do controle, é necessário conhecer as características e o ambiente de simulação. A Tabela 3 faz um breve resumo do ambiente utilizado, conhecido como *"CartPole-Discrete"* no Matlab.

Tabela 3 – Ambiente de simulação

Parâmetro	**Valor**
Gravidade	9,8 m/s^2
Massa do carro	1 kg
Massa do pêndulo	0,1 kg
Comprimento da haste	0,5 m
Ângulo limite	0,2094 rad
Deslocamento limite	2,4 m

Fonte: elaborada pelo autor.

Com os dados da Tabela 3, é possível calcular o momento de inércia I, assim como a matriz A do sistema e a matriz B de entrada. Além disso, a matriz de saída C será a matriz identidade e a matriz de transmissão direta D será nula. Dessa forma, as matrizes que representam o sistema ficam da seguinte maneira:

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0,7171 & 0 & 0 \\ 0 & -31,5512 & 0 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 \\ 0 \\ 0,9756 \\ -2,9268 \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$D = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Com as matrizes do sistema definidas, basta definir as matizes Q e R conforme seja necessário para o problema, depois utilizar a função *"lqr()"* no Matlab para encontrar o valor de K do controlador. Assim, é possível analisar para quais valores de Q o sistema estabiliza melhor, o sinal de controle, o valor de cada estado em função do tempo etc.

4.2 Agente DQN Utilizando um Crítico de Múltiplas Saídas

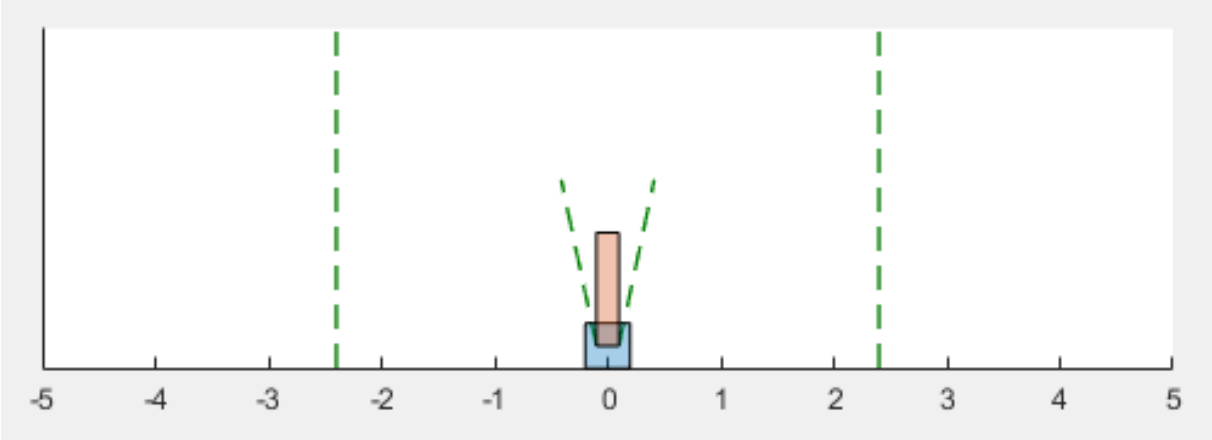
O primeiro controle utilizando a técnica AR será um agente do tipo DQN utilizando um crítico de múltiplas saídas. Por meio de uma função crítica de valor Q, o agente DQN aproxima a recompensa a longo prazo.

Este tipo de agente tem um espaço de ação discreto, possibilitando a criação de uma função de valor Q vetorial, ou seja, com múltiplas saídas, que geralmente é mais eficiente que uma função com apenas uma saída. Uma função dessa natureza, valor Q vetorial, pode ser vista como um mapeamento da observação de um ambiente. Cada elemento do vetor representa a recompensa cumulativa de longo prazo, o desconto, quando o agente parte de um determinado estado correspondente à dada observação, além de executar a ação correspondente ao número do elemento. Posteriormente, o agente segue uma determinada política.

O primeiro passo é criar o ambiente de simulação. Neste trabalho, o ambiente de simulação é o *CartPole-Discrete* disponibilizado pelo Matlab e representado pela Figura 16. Em seguida, para criar a função de valor Q, utiliza-se uma rede neural profunda, em inglês, *deep neural network*. A rede deve possuir uma camada de entrada, na qual recebe o conteúdo do canal de observação e as informações de ambiente, camadas intermediárias ou escondidas, cuja função é processar a informação, e uma camada de saída que retorna um vetor com os valores de todas as ações possíveis, conforme representada pela Figura 17. Neste primeiro agente, a rede neural tem 4 entradas, duas camadas intermediárias com 24 neurônios cada, e uma saída com dois sinais.

Com a rede criada e as informações do ambiente obtidas, agora é possível criar a função de valor Q para o agente. Para verificar o resultado da função, basta aplicar valores de

Figura 16 – Ambiente utilizado para simulação



Fonte: elaborada pelo autor.

Figura 17 – Rede neural para o agente utilizando um crítico de múltiplas saídas

Fonte: elaborada pelo autor.

entrada para retornar uma observação aleatória. A partir de ponto, é possível criar, configurar e verificar o agente.

O próximo passo é treinar o agente criado anteriormente. Para isso, é necessário configurar as opções de treinamento, como número máximo de episódios e critérios de parada, e depois simular. Também é possível adicionar critérios para salvar vários agentes durante o

treinamento, possibilitando compará-los durante as simulações. Para visualizar os resultados obtidos, uma possibilidade é utilizar a toolbox de AR do Matlab.

Existem inúmeras configurações tanto para o agente, quanto para o treinamento do agente. A Tabela 4, por exemplo, apresenta algumas configurações utilizadas para criar os agentes e a Tabela 5 apresenta as configurações para o treinamento. Cada projetista pode modificar do jeito que achar melhor.

Tabela 4 – Configurações para agente DQN utilizando um crítico de múltiplas saídas

Parâmetro	**Valor / Configuração**
Epsilon decay	0.0050
Epsilon	1
Epsilon mínimo	0,01
Taxa de aprendizagem	0,01
Otimizador	Adam
Fator de desconto	0,99

Fonte: elaborada pelo autor.

Tabela 5 – Configurações para o treinamento do agente DQN utilizando um crítico de múltiplas saídas

Parâmetro	**Valor / Configuração**
Número máximo de episódios	500
Número máximo de passos por episódio	500
Critério de parada	Média contínua de recompensa por episódio
Valor para o critério de parada	500
Critério de salvar	Média contínua de recompensa por episódio
Valor para o critério de salvar	500

Fonte: elaborada pelo autor.

Os parâmetros para o ambiente de simulação são bem parecidos com os parâmetros para o controlador LQR, porém, para o controlador utilizando AR há valores para recompensa e penalidade. A Tabela 5 apresenta estes valores.

4.3 Agente DQN com Rede Neural Recorrente

O processo de criação de um agente do tipo DQN com rede neural recorrente é bastante similar ao que foi visto na Seção 4.2. Primeiramente, no processo de criação da rede neural, é necessário criar uma camada de entrada do tipo sequencial, ou seja, uma camada capaz de inserir dados de sequência em uma rede neural e aplicar a normalização de dados, conforme exemplificado na Figura 18. No Matlab é possível utilizar a função *sequenceInputLayer* para

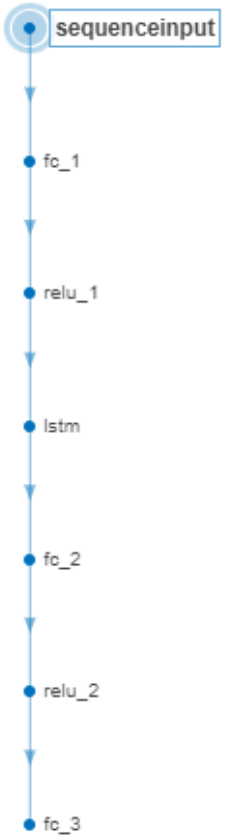
Tabela 6 – Ambiente de simulação do agente DQN utilizando um crítico de múltiplas saídas

Parâmetro	Valor
Gravidade	9,8 m/s^2
Massa do carro	1 kg
Massa do pêndulo	0,1 kg
Comprimento da haste	0,5 m
Ângulo limite	0,2094 rad
Deslocamento limite	2,4 m
Recompensa por não cair	1
Punição por cair	-5

Fonte: elaborada pelo autor.

criar a camada sequencial de entrada. Além disso, é necessário criar pelo menos uma camada de memória de longo prazo, em inglês, *long short-term memory* (LSTM). No Matlab existe a função *lstmLayer* que realiza esta operação. Neste segundo agente, a rede neural tem 4 entradas, 3 camadas intermédiarias, sendo uma do tipo LSTM com 20 neurônios e as demais com 24, e, assim como no primeiro agente, uma saída com dois sinais.

Figura 18 – Rede neural para o agente utilizando rede neural recorrente



Fonte: elaborada pelo autor.

Com a rede neural devidamente criada, é possível criar o agente DQN da mesma forma que foi criado na Seção 4.2. Porém, existe um parâmetro durante o processo de criação que precisa ser modificado: o comprimento da sequência. Este parâmetro define o comprimento máximo da trajetória para o treinamento de uma rede neural recorrente. Obrigatoriamente o valor tem que ser maior que 1.

Os demais parâmetros de configuração do agente se encontram na Tabela 4, assim como os parâmetros do treinamento se encontram na Tabela 5. Para efeito de comparação, as simulações serão feitas no mesmo ambiente e os parâmetros podem ser vistos na Tabela 6.

5 DISCUSSÃO DOS RESULTADOS

Este capítulo está dividido em três partes: apresentação dos resultados do controlador LQR, do controlador utilizando um agente DQN com função crítica de múltiplas saídas e do controlador utilizando um agente DQN com rede neural recorrente.

Na primeira parte serão apresentados três testes utilizando o controlador LQR, cada teste apresentará um valor diferente para a matriz Q. A variação dessa matriz representa o quanto cada estado é significativo, quando maior o valor, mais significativo o estado. Na segunda e terceira parte serão feitos as simulações com os dois tipos de agentes treinados, como o algoritmo salva vários possíveis agentes, os dois agentes escolhidos serão os últimos que foram armazenados pelo Matlab. Dentre as simulações, serão analisadas as três melhores para cada agente, ou seja, aquelas que obtiveram máximas recompensas, isso significa que os agentes conseguiram estabilizar o sistema dentro dos limites estabelecidos.

5.1 Controlador LQR

As simulações do controlador LQR foram feitas utilizando o sistema em variáveis de espaço de estado conforme a Seção 4.1. Para efeito de comparação, variou-se a matriz Q, representada pela Equação 5.1, para encontrar diferentes valores de K.

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & a & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{5.1}$$

Os valores na diagonal da matriz Q, Equação 5.1, representam o peso que cada estado tem, ou seja, quanto maior o valor, maior a influência no controle. Por meio dela é possível afirmar que há influência em apenas dois estados: posição e ângulo. Durante as simulações, apenas o segundo estado variou (ângulo), representado pela letra a na matriz Q.

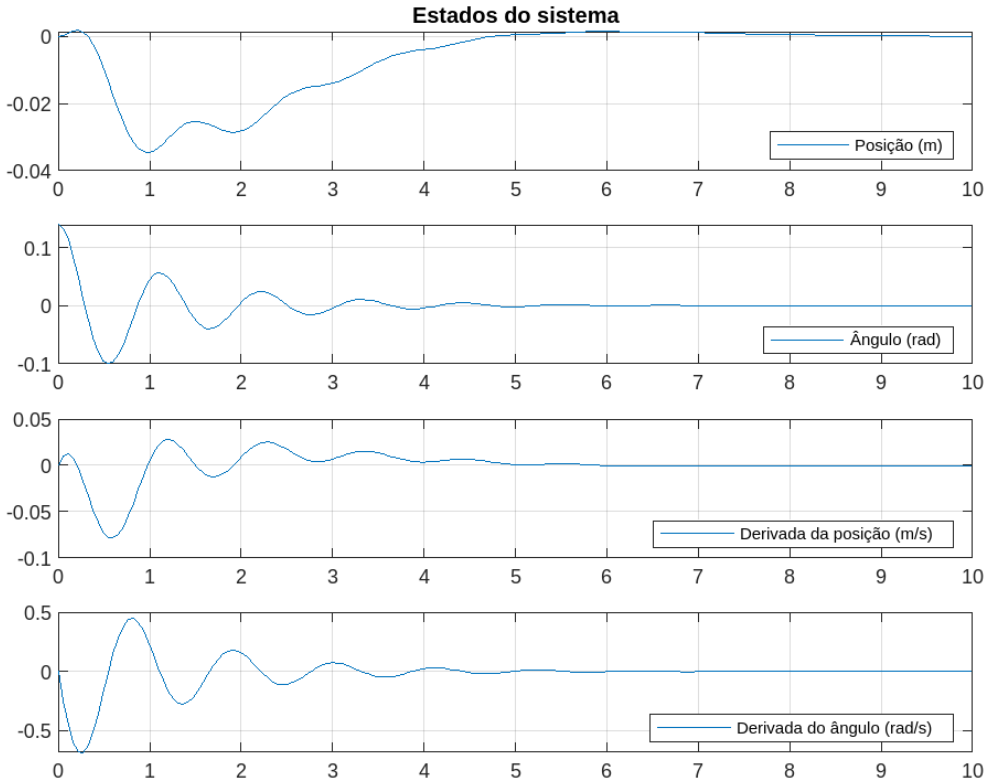
5.1.1 Resultados para a = 10

Foi encontrado o seguinte valor para K:

$$K = \begin{bmatrix} 1,0000 & -1,1519 & 1,5605 & -0,4873 \end{bmatrix}$$

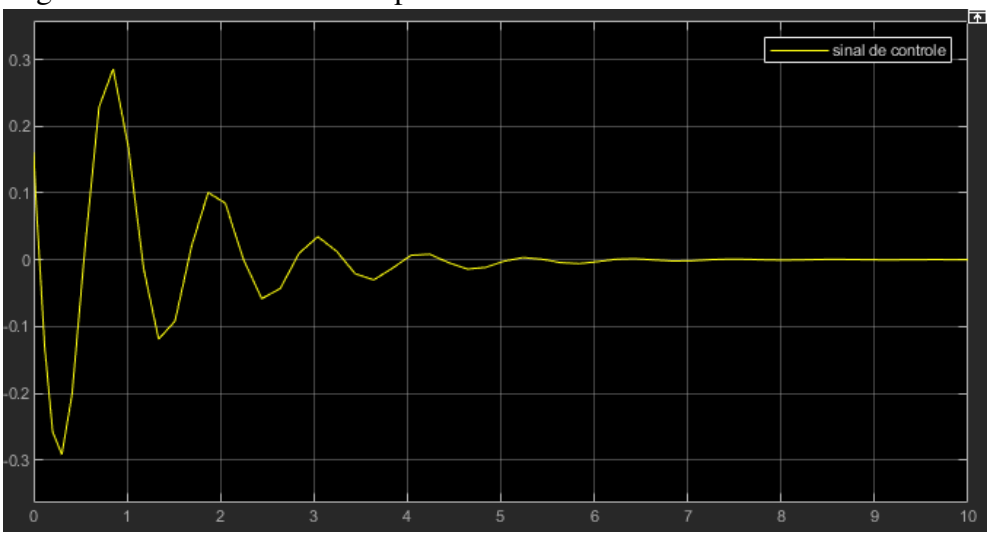
A evolução de cada estado utilizando o controlador LQR está representada pela Figura 19, assim como o sinal de controle pela Figura 20.

Figura 19 – Estados para a = 10



Fonte: elaborada pelo autor.

Figura 20 – Sinal de controle para a = 10



Fonte: elaborada pelo autor.

O ângulo se estabilizou em aproximadamente 4 segundos, sem ultrapassar os limites de posição e ângulo, 2,4 metros e 12 graus, respectivamente. O valor máximo do sinal de controle

foi 0,29.

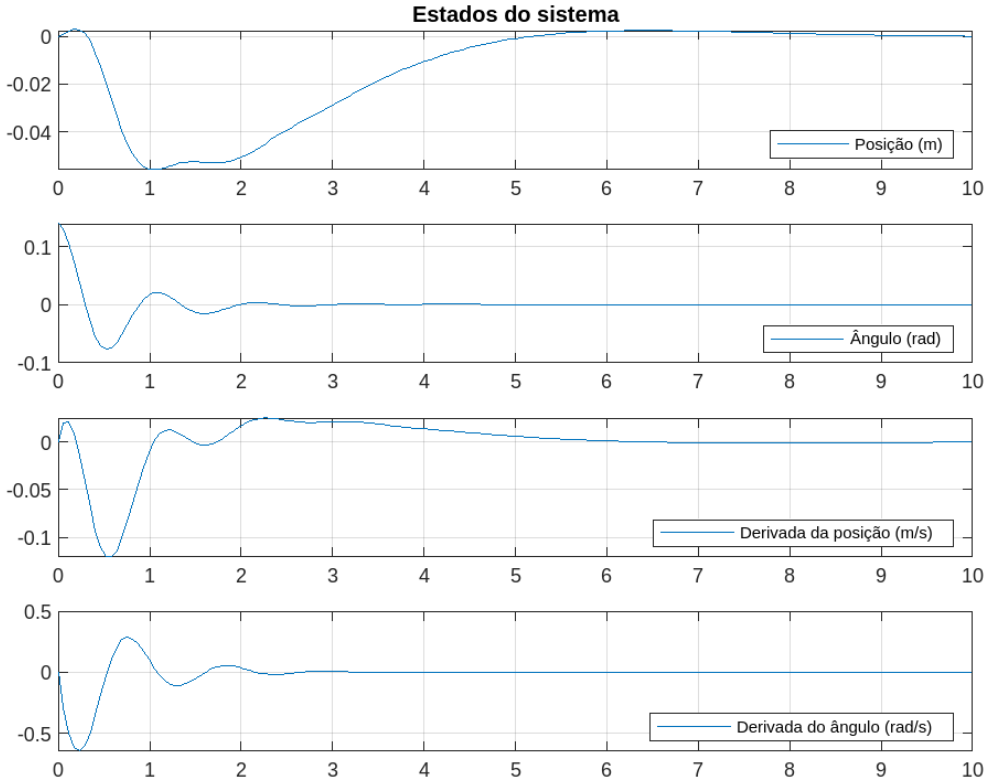
5.1.2 Resultados para a = 40

Foi encontrado o seguinte valor para K:

$$K = \begin{bmatrix} 1,0000 & -3,0023 & 1,6771 & -0,9507 \end{bmatrix}$$

A evolução de cada estado utilizando o controlador LQR está representada pela Figura 21, assim como o sinal de controle pela Figura 22.

Figura 21 – Estados para a = 40



Fonte: elaborada pelo autor.

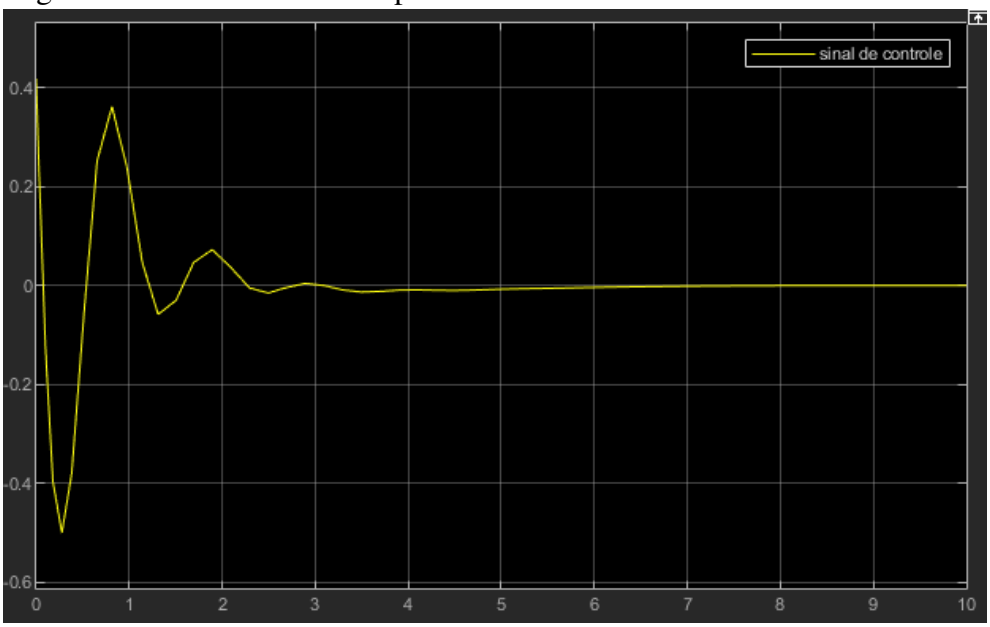
O sistema se estabilizou em aproximadamente 3 segundos, sem ultrapassar os limites de posição e ângulo, 2,4 metros e 12 graus, respectivamente. O valor máximo do sinal de controle foi 0,42.

5.1.3 Resultados para a = 80

Foi encontrado o seguinte valor para K:

$$K = \begin{bmatrix} 1,0000 & -4,8788 & 1,7876 & -1,2913 \end{bmatrix}$$

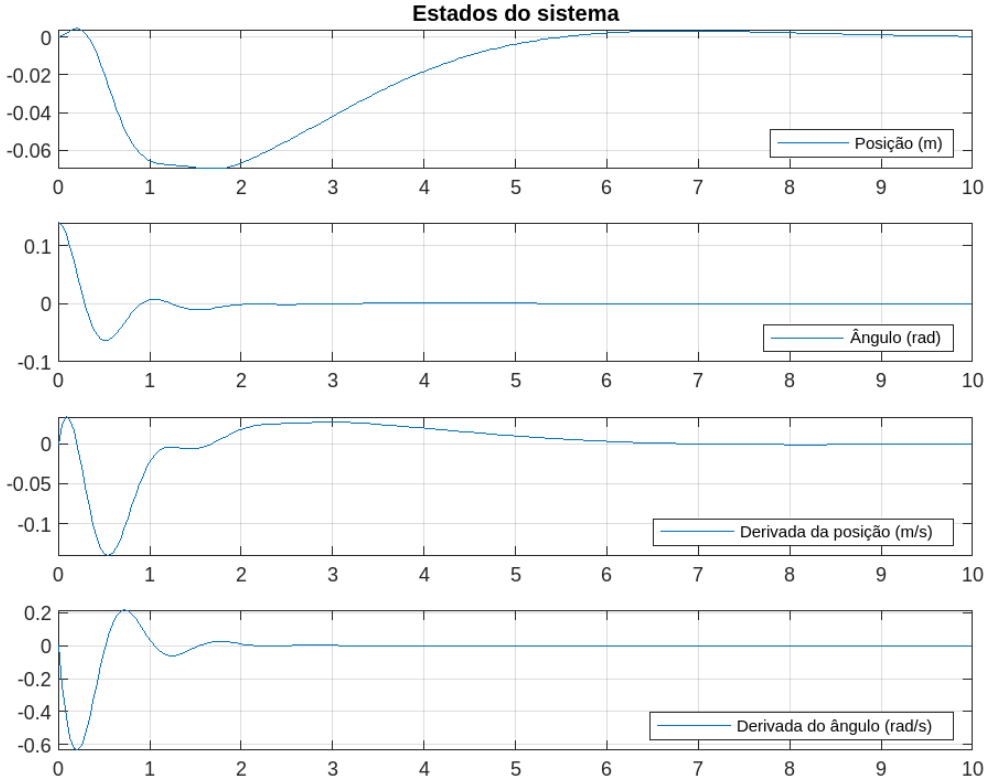
Figura 22 – Sinal de controle para a = 40



Fonte: elaborada pelo autor.

A evolução de cada estado utilizando o controlador LQR está representada pela Figura 23, assim como o sinal de controle pela Figura 24.

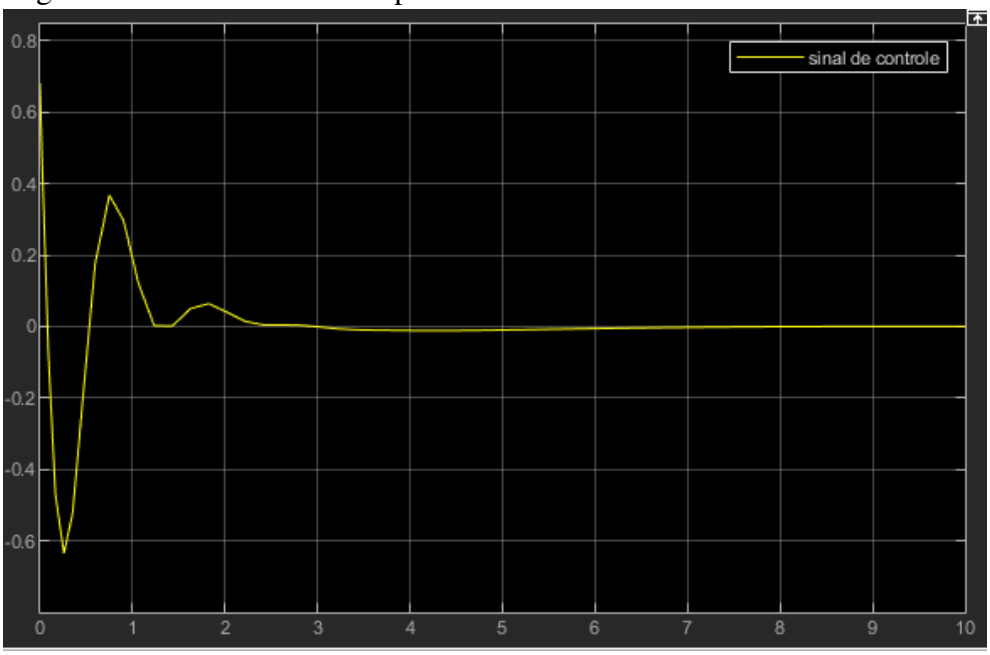
Figura 23 – Estados para a = 80



Fonte: elaborada pelo autor.

O ângulo se estabilizou em aproximadamente 2 segundos, sem ultrapassar os limites

Figura 24 – Sinal de controle para a = 80



Fonte: elaborada pelo autor.

de posição e ângulo, 2,4 metros e 12 graus, respectivamente. O valor máximo do sinal de controle foi 0,68.

Por meio dos resultados obtidos via simulações no Matlab, é possível afirmar que, de fato, variar o parâmetro *a* da matriz Q, Equação 5.1, modifica consideravelmente o segundo estado do sistema, o ângulo na vertical. Quanto maior o valor de *a*, mais rápido o sistema estabiliza. Porém, foi necessária mais energia para poder fazer isso, conforme visto nos sinais de controle. O código utilizado para simulação está disponível no Apêndice A.

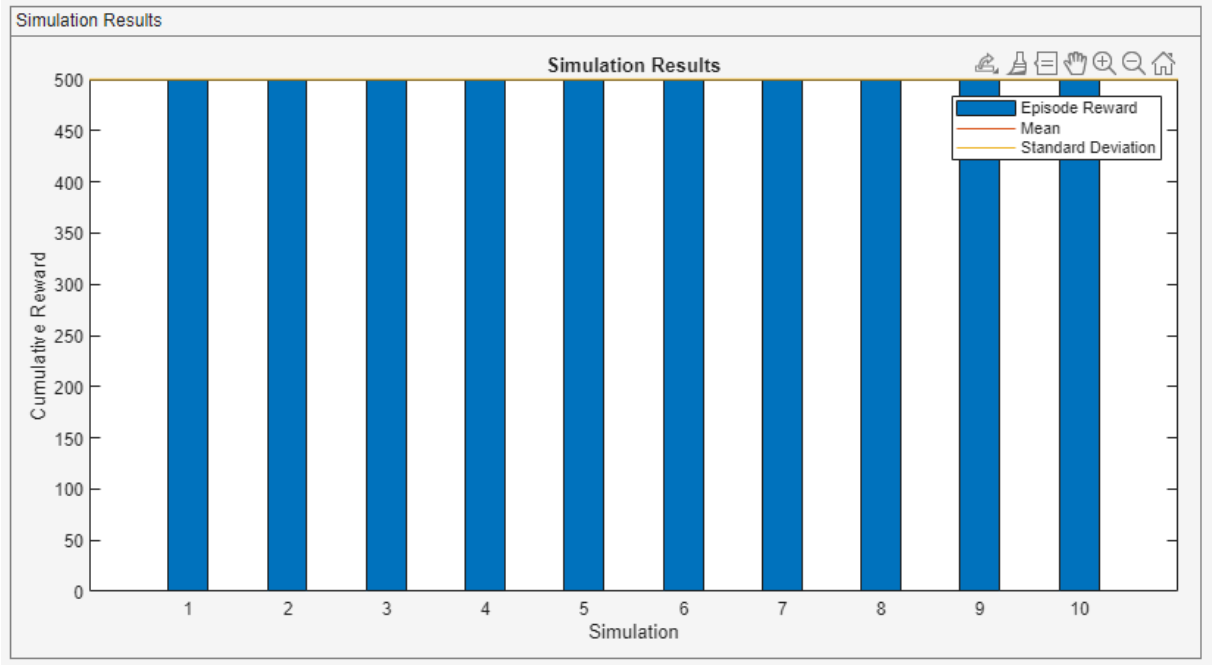
5.2 Primeiro Agente DQN

O primeiro agente é o controle apresentado na Seção 4.2, Agente DQN utilizando um crítico de múltiplas saídas.

O treinamento de todos os agentes levou aproximadamente cinco minutos. O Matlab salvou um total de 39 agentes, ou seja, estes 39 agentes conseguiram a recompensa máxima acumulada, 500, durante os episódios da fase de treinamento. O agente escolhido para as simulações foi o agente 130 (agente salvo durante o episódio 130), ele obteve a recompensa máxima em todas as 10 simulações, como pode ser visto na Figura 25, que representa as simulações realizadas e a respectiva recompensa acumulada pelo agente.

As simulações 1, 2 e 3 foram escolhidas para comparação. A visualização dos estados é diferente na toolbox de AR no Matlab em relação aos estados apresentado na Seção

Figura 25 – Resultados do primeiro agente DQN



Fonte: elaborada pelo autor.

2.2.1. O primeiro estado se refere a posição, o segundo estado se refere à derivada da posição, o terceiro estado se refere ao ângulo e o quarto estado se refere à derivada do ângulo. Todos os resultados mostrados na toolbox do Matlab seguirão essa ordem.

5.2.1 Simulação 1 do Primeiro Agente DQN

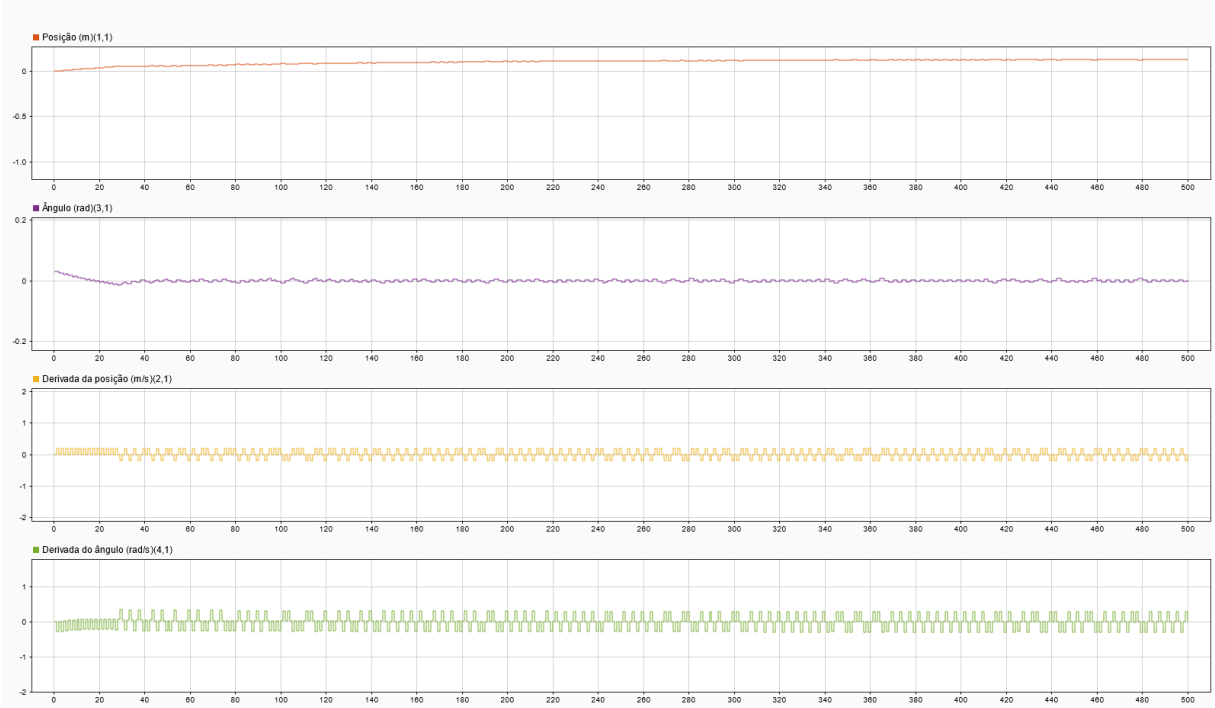
Os resultados da simulação 1 podem ser visualizados na Figura 26. Cada estado está representado por uma cor diferente. Embora o eixo das abscissas represente a interação entre o ambiente e agente, é possível descobrir quanto tempo leva para a simulação atingir um dos limites estabelecidos previamente. Para isso, basta utilizar as funções *tic* e *toc* no Matlab. Nas simulações da Seção 5.2, por exemplo, cada uma levou aproximadamente 4,5 segundos.

O carrinho saiu da origem e ficou em $0,13m$, aproximadamente. A velocidade ficou atingindo dois valores, $0,20m/s$ e $-0,20m/s$. O ângulo começou em $0,031rad$ e terminou oscilando em zero. Por fim, a derivada do ângulo ficou variando em dois valores, $0,32rad/s$ e $-0,26rad/s$.

5.2.2 Simulação 2 do Primeiro Agente DQN

Os resultados da simulação 2 podem ser visualizados na Figura 27. Assim como na simulação anterior, cada estado está representado por uma cor diferente.

Figura 26 – Resultados da simulação 1 do primeiro agente DQN



Fonte: elaborada pelo autor.

O carrinho partiu da origem, teve um pico em $0,17m$ e depois estabilizou em $0,13m$, aproximadamente. A velocidade teve um pico de $0,38m/s$ e depois ficou atingindo dois valores, $0,20m/s$ e $-0,19m/s$. O ângulo começou em $0,041rad$, depois foi para esquerda até atingir uma angulação de $-0,029rad$ e terminou oscilando em zero. Por fim, a derivada do ângulo ficou variando em dois valores depois de um momento de transição, $0,29rad/s$ e $-0,30rad/s$.

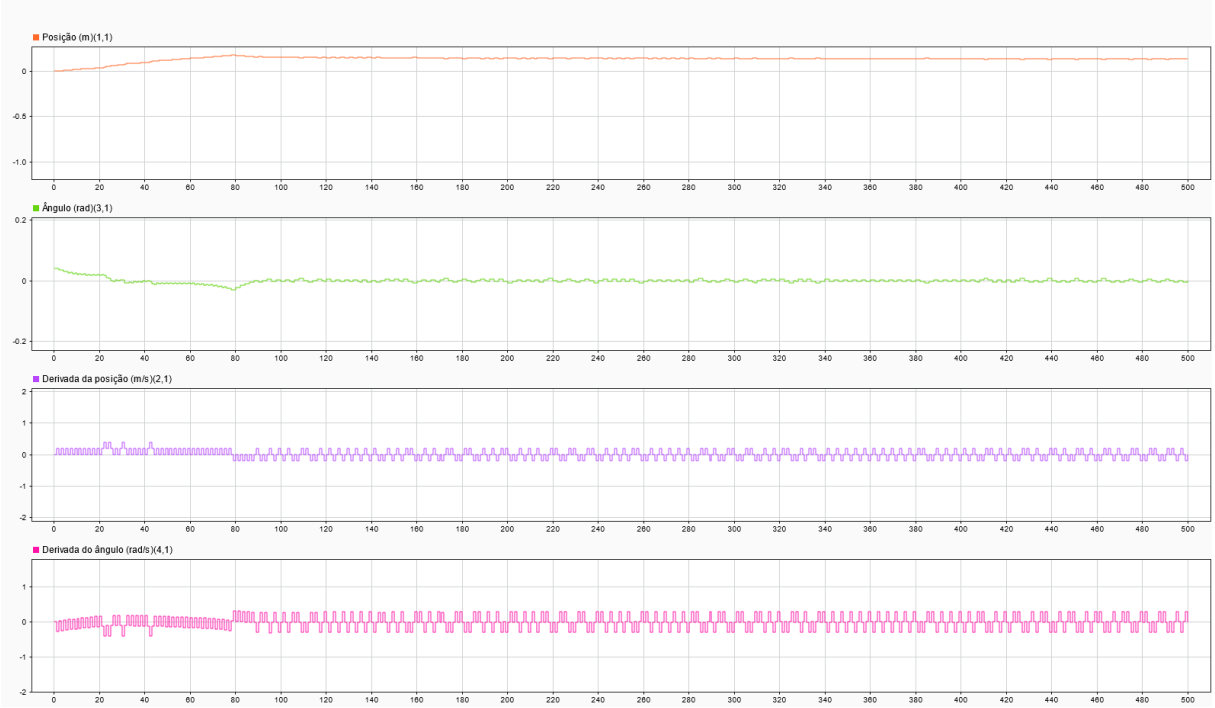
5.2.3 Simulação 3 do Primeiro Agente DQN

Os resultados da simulação 3 podem ser visualizados na Figura 28. Assim como na simulação anterior, cada estado está representado por uma cor diferente.

O carrinho partiu da origem, se deslocou para a esquerda $0,08$, depois foi para a direita e teve um pico em $0,20m$, por fim estabilizou em $0,14m$. A velocidade teve um pico de $-0,39m/s$ para a esquerda e $0,38m/s$ para a direita, e depois ficou atingindo dois valores, $0,20m/s$ e $-0,19m/s$. O ângulo começou em $-0,037rad$, depois foi para direita até atingir uma angulação de $0,039rad$, em seguida voltou para a esquerda e atingiu uma angulação de $-0,035rad$, em seguida terminou oscilando em zero. Por fim, a derivada do ângulo ficou variando em dois valores depois de um momento de transição, $0,28rad/s$ e $-0,30rad/s$.

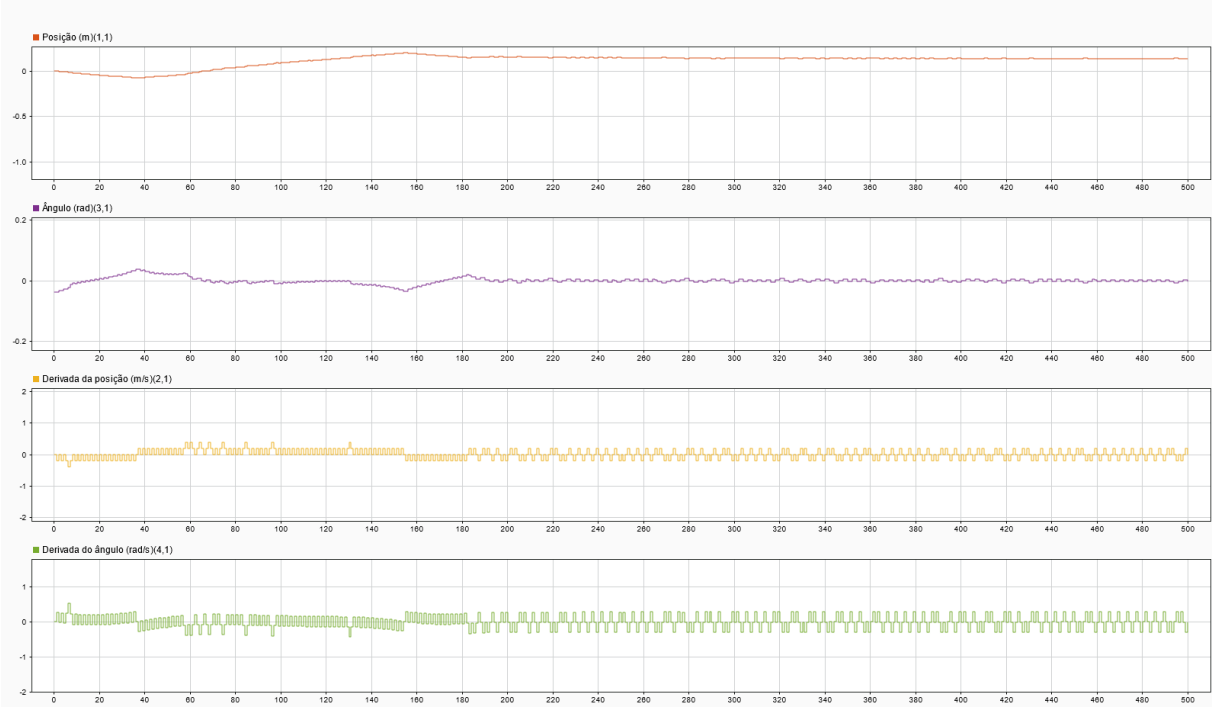
As três simulações apresentaram resultados extremamente satisfatórios, visto que o agente conseguiu equilibrar o pêndulo em todas as simulações e no final houve pouca variação

Figura 27 – Resultados da simulação 2 do primeiro agente DQN



Fonte: elaborada pelo autor.

Figura 28 – Resultados da simulação 3 do primeiro agente DQN



Fonte: elaborada pelo autor.

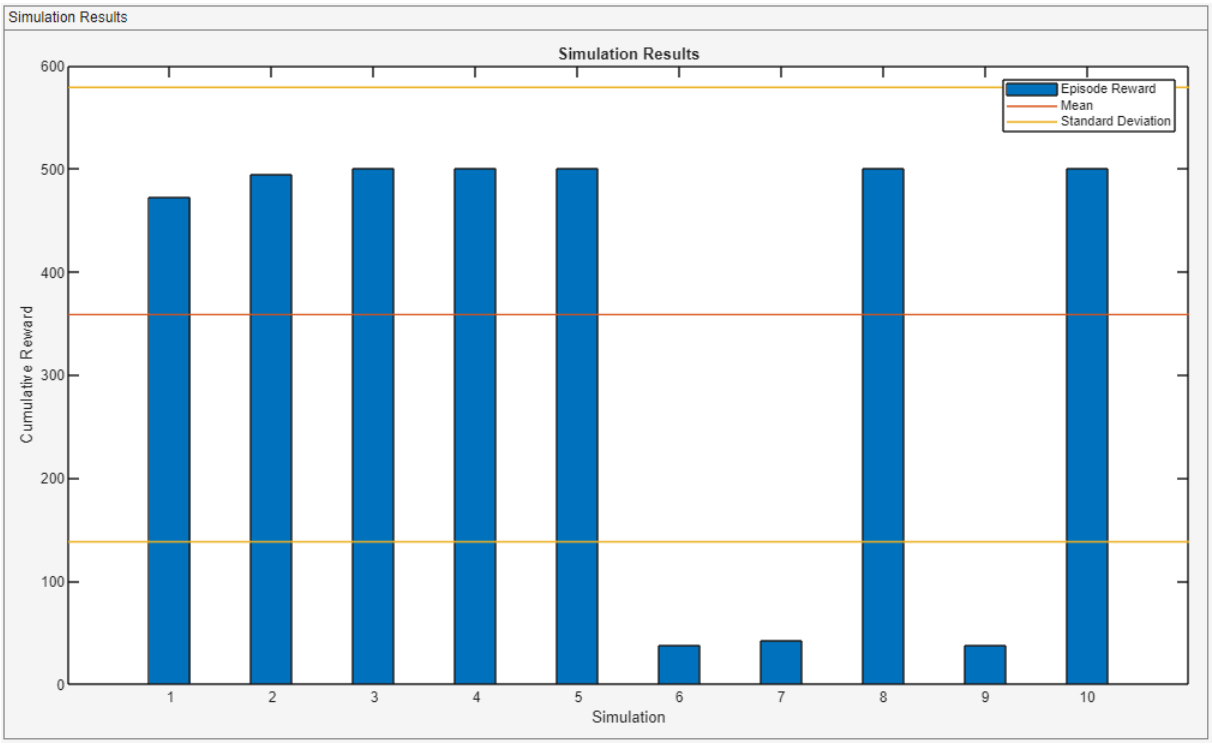
do ângulo. No geral o carrinho tentava terminar um pouco mais a direita, tendendo a voltar para a origem, mas isso não aconteceu. Após um determinado momento de transição, os outros estados também tenderam a atuar num ponto em comum. O código para criar, configurar, treinar e simular o agente se encontra no Apêndice B.

5.3 Segundo Agente DQN

O segundo agente é o controle apresentado na Seção 4.3, Agente DQN com Rede Neural Recorrente.

O treinamento de todos os agentes levou aproximadamente duas horas. O Matlab salvou um total de 9 agentes, ou seja, estes 9 agentes conseguiram a recompensa máxima acumulada, 500. O agente escolhido para as simulações foi o agente 410, ele obteve a recompensa máxima em 5 simulações, como pode ser visto na Figura 29.

Figura 29 – Resultados do segundo agente DQN



Fonte: elaborada pelo autor.

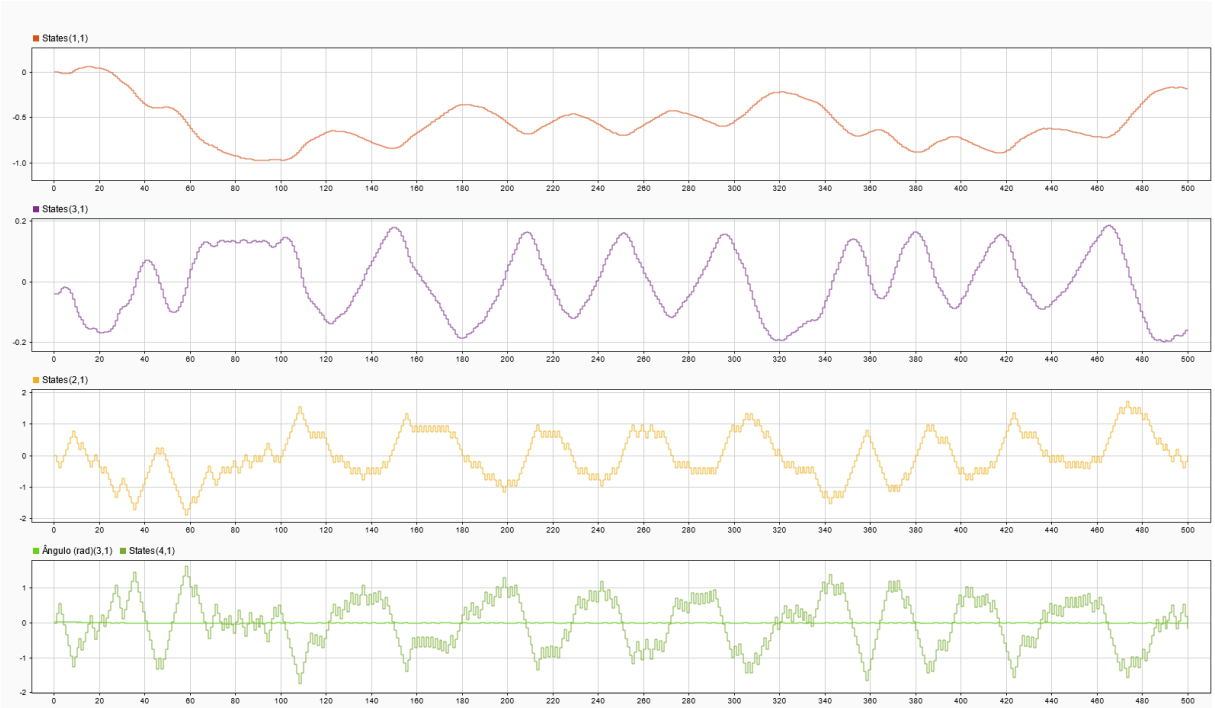
As simulações escolhidas para comparação foram as três últimas que obtiveram recompensa máxima: 5, 8 e 10. Como os resultados são visualizados na toolbox do Matlab, então a maneira como os estados são distribuídos é equivalente aos apresentados na Seção 5.2.

5.3.1 Simulação 5 do Segundo Agente DQN

Os resultados da simulação 5 podem ser visualizados na Figura 30. Cada estado está representado por uma cor diferente. Assim como na Seção 5.2, foram utilizadas as funções *tic* e *toc* no Matlab para calcular o tempo de simulação do agente na Seção 5.3. Cada simulação durou, aproximadamente, 7 segundos.

O carrinho partiu da origem, foi para $-0,97$, oscilou por um tempo em $-0,5m$ e depois ficou em $-0.18m$. A velocidade ficou oscilando entre $-0,8m/s$ e $0,9m/s$ durante a maior parte da simulação. O ângulo começou em aproximadamente zero, mas logo em seguida foi para $-0,17rad$ e depois $0,18rad$. Por fim, a derivada do ângulo ficou variando entre $-1,1rad/s$ e $0,8rad/s$, mas também atingiu picos de $-1,6rad/s$ e $1,6rad/s$.

Figura 30 – Resultados da simulação 5 do segundo agente DQN



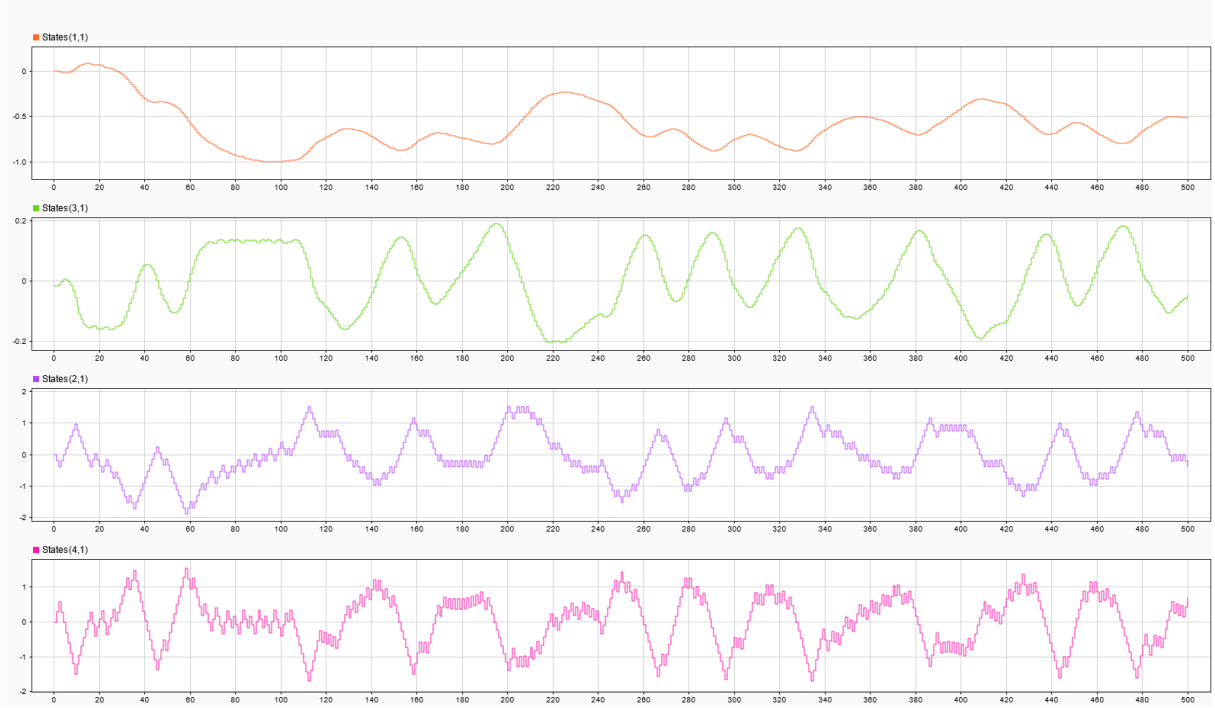
Fonte: elaborada pelo autor.

5.3.2 Simulação 8 do Segundo Agente DQN

Os resultados da simulação 8 podem ser visualizados na Figura 31. Cada estado está representado por uma cor diferente.

O carrinho partiu da origem, se deslocou para $-1m$, depois teve um pico em $-0,23m$ e terminou em $-0,52m$. A velocidade ficou oscilando entre $-1m/s$ e $1m/s$ durante a maior parte da simulação, porém chegou a atingir $-1,7m/s$ e $1,5m/s$ em alguns casos. O ângulo começou em $-0,02rad$, foi para $-0,16rad$, mas logo em seguida ficou oscilando entre $-0,07rad$ e $0,15rad$ durante alguns momentos, porém em dois casos atingiu $-0,19rad$. Por fim, a derivada do ângulo ficou variando entre $-1,6rad/s$ e $1rad/s$.

Figura 31 – Resultados da simulação 8 do segundo agente DQN



Fonte: elaborada pelo autor.

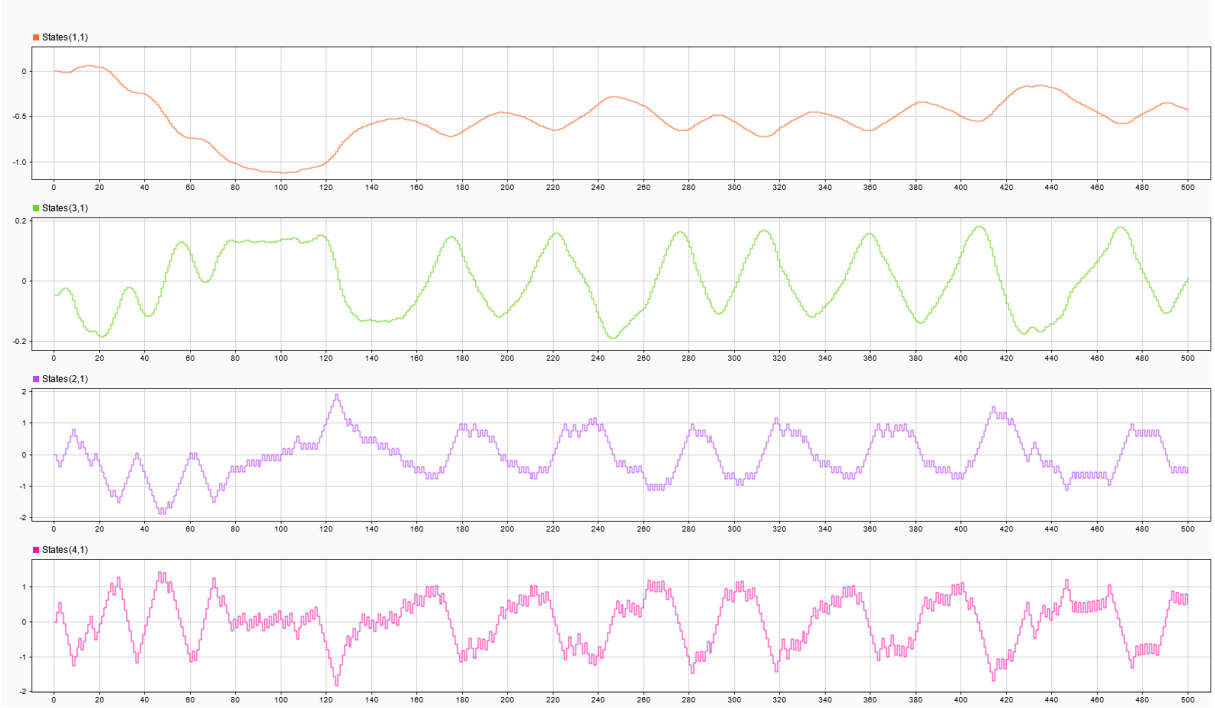
5.3.3 Simulação 10 do Segundo Agente DQN

Os resultados da simulação 10 podem ser visualizados na Figura 32. Cada estado está representado por uma cor diferente.

O carrinho mais uma vez partiu da origem, se deslocou para $-1,12m$, depois seguiu para a direita e terminou em $-0,43m$. A velocidade começou oscilando pouco entre $-1,5m/s$ e $0,1m/s$, depois teve um pico em $1,9m/s$, em seguida ficou oscilando entre valores próximos de $-0,8m/s$ e $1m/s$. O ângulo começou em $-0,05rad$, foi para $-0,19rad$, em seguida foi para a direita com ângulos próximos de $0,13rad$, porém em seguida ficou oscilando entre $0,16rad$ e $-0,12rad$. Por fim, a derivada do ângulo ficou variando entre $-1,2rad/s$ e $1rad/s$, atingindo alguns picos de $-1,7rad/s$.

Em todas as três simulações o carrinho conseguiu equilibrar o pêndulo. O limite do ângulo quase foi atingido algumas vezes, porém nunca foi ultrapassado, assim como o limite para o deslocamento. No geral houve muita oscilação nos dois principais estados, posição e ângulo, por conseguinte, nas respectivas derivadas também. O código para criar, configurar, treinar e simular o agente se encontra no Apêndice C.

Figura 32 – Resultados da simulação 10 do segundo agente DQN



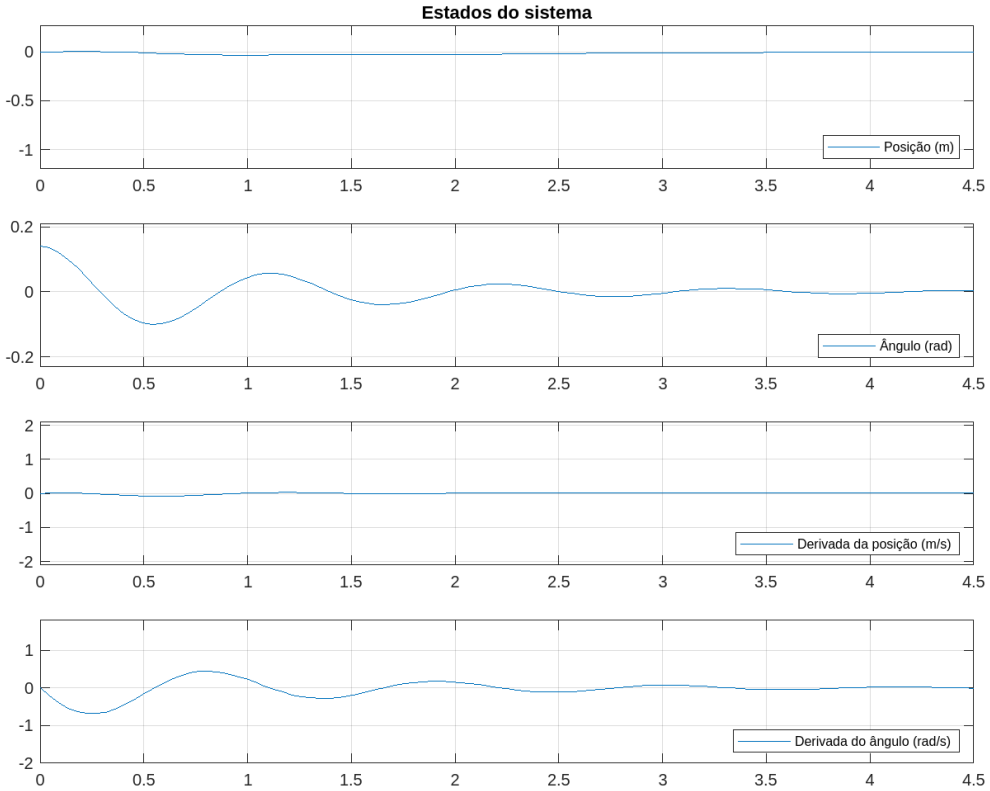
Fonte: elaborada pelo autor.

5.4 Comparação Entre os 3 Controladores

As 3 abordagens propostas neste trabalho conseguiram atingir o objetivo: controlar o pêndulo invertido em cima de um carrinho para as mesmas condições. Porém, como esperado, a maneira como os métodos realizaram o controle não foi a mesma. O primeiro método, utilizando o controlador LQR, controlou muito bem o sistema nos primeiros 4,5 segundos. A Figura 33 mostra que em 4 segundos o ângulo já está equilibrado com pouca variação no eixo vertical. A Figura 34, por outro lado, mostra que em 2,5 segundos o ângulo, assim como no primeiro caso, já está equilibrado devido a mudança de valor do parâmetro "a". Por fim, a Figura 35, com um valor para "a"ainda maior, mostra que o sistema entrou em equilíbrio em 2 segundos. Aumentando o valor do parâmetro "a", faz o sistema atingir o equilíbrio mais rapidamente. Porém, a consequência é também aumentar o sinal de controle como foi visto na Seção 5.1.

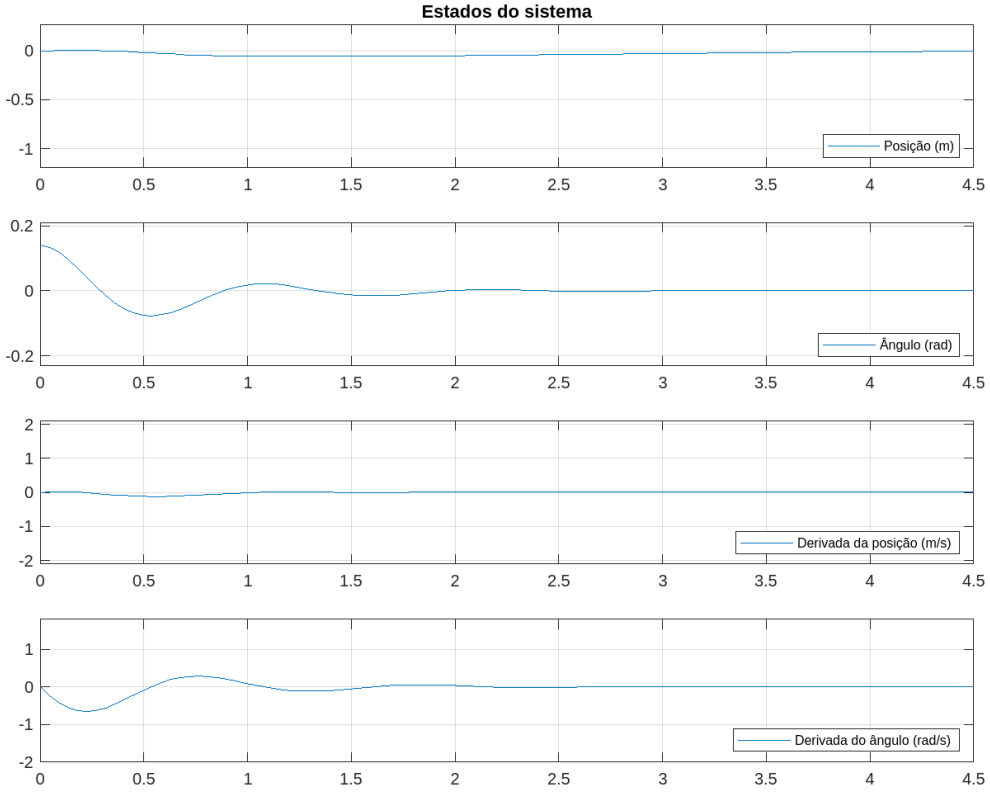
O segundo controlador utilizado, o primeiro agente DQN, conseguiu manter pequenas oscilações ao redor do ângulo de equilíbrio. Na Figura 36, representando a primeira simulação, o sistema começou a oscilar perto do ângulo de equilíbrio em aproximadamente 0,36 segundos, mas nunca se mantendo em zero. Por outro lado, na segunda simulação, conforme a Figura 37, o agente precisou de 0,9 segundos para começar a oscilar ao redor de zero, enquanto na terceira simulação, conforme a Figura 38, o agente precisou de 1,8 segundos.

Figura 33 – Controlador LQR para a = 10 nos 4,5 primeiros segundos



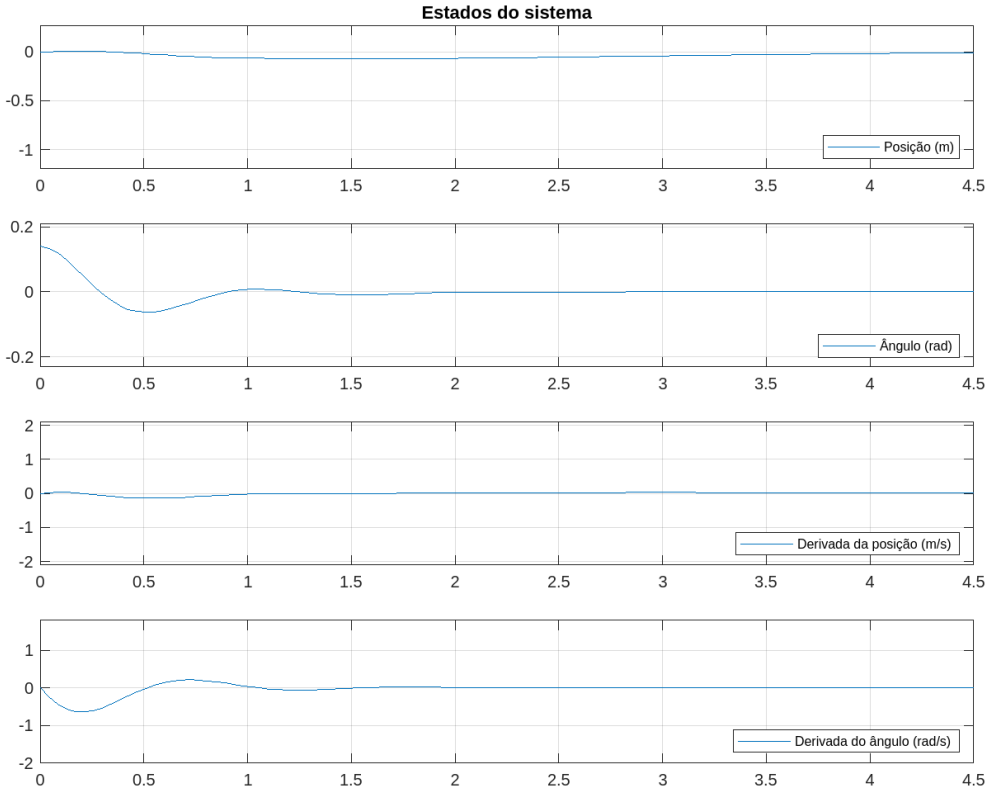
Fonte: elaborada pelo autor.

Figura 34 – Controlador LQR para a = 40 nos 4,5 primeiros segundos



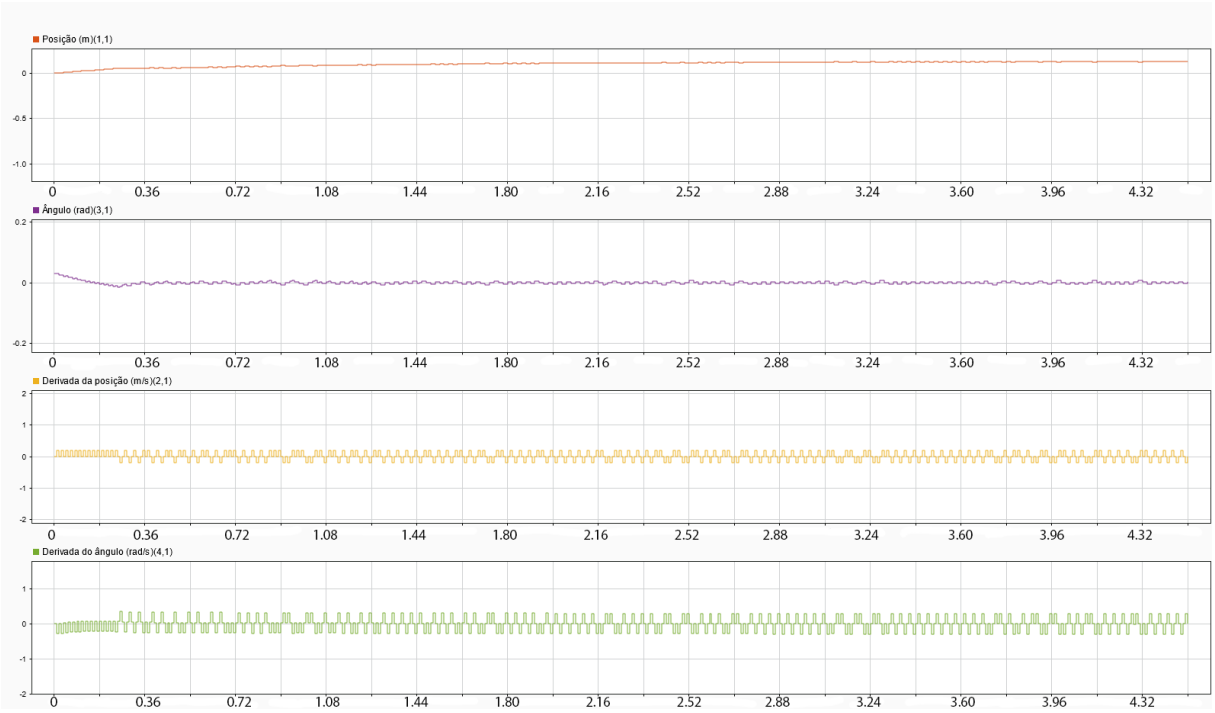
Fonte: elaborada pelo autor.

Figura 35 – Controlador LQR para a = 80 nos 4,5 primeiros segundos



Fonte: elaborada pelo autor.

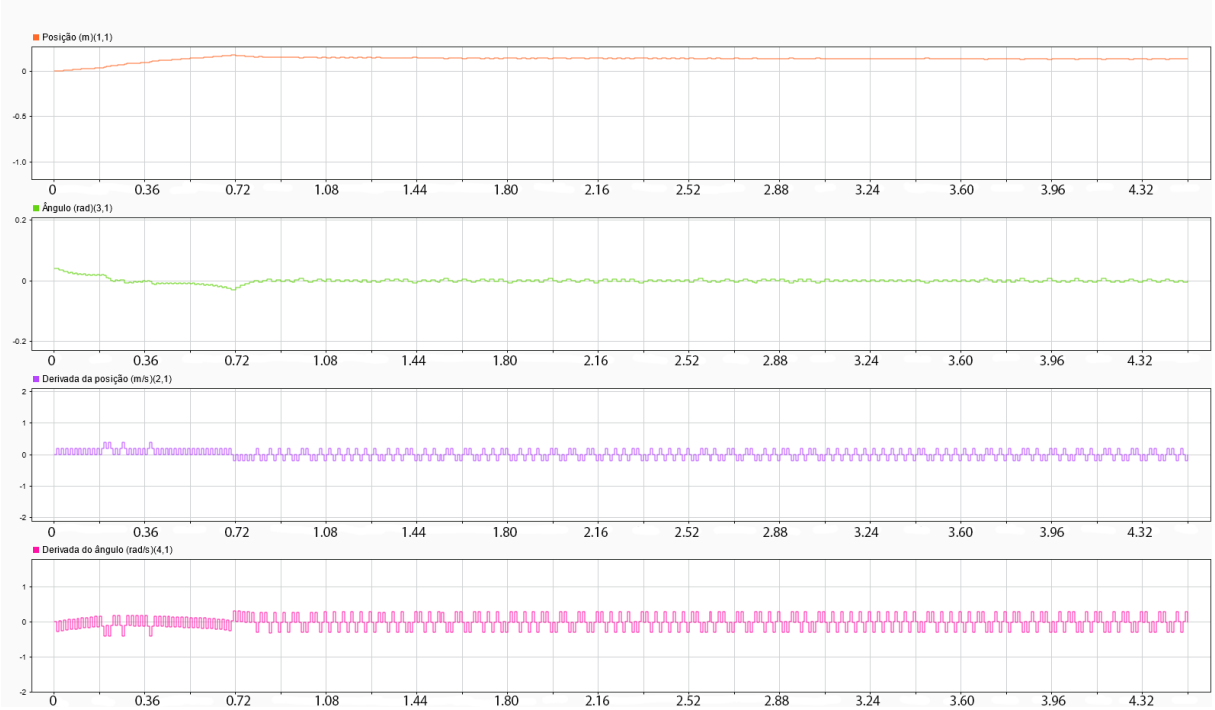
Figura 36 – Primeiro Agente DQN nos 4,5 primeiros segundos (simulação 1)



Fonte: elaborada pelo autor.

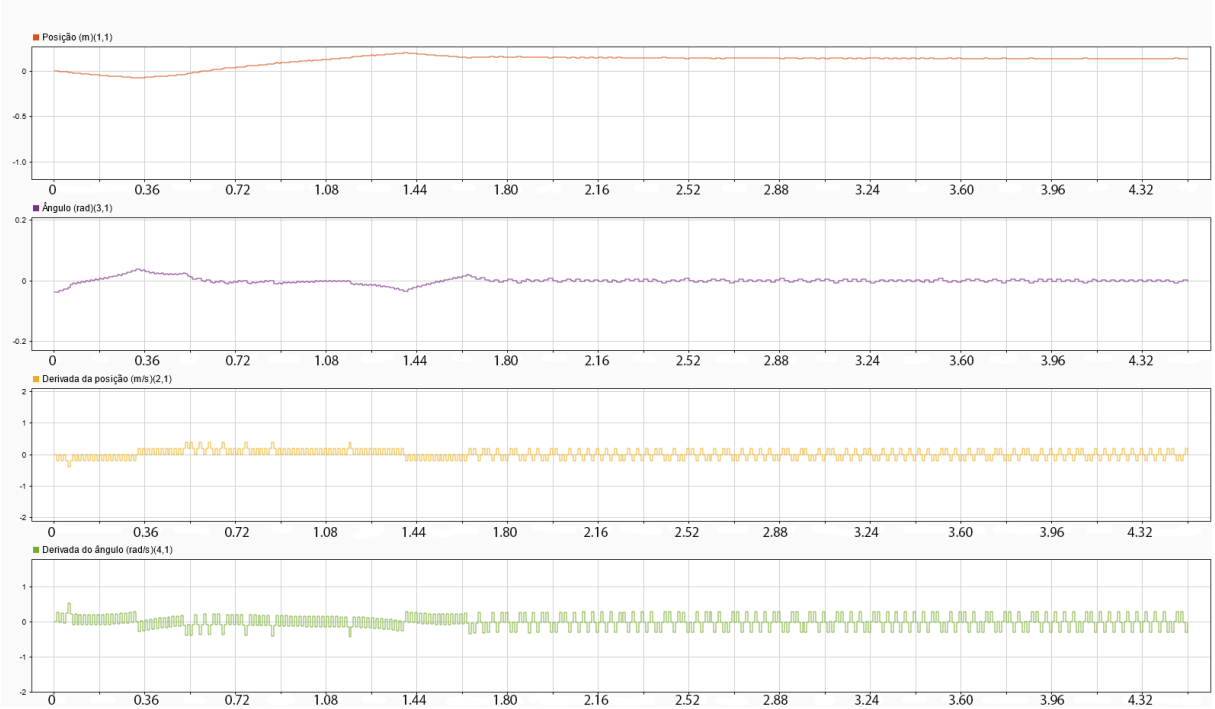
Por fim, o último controlador utilizado foi o segundo agente DQN. Ele conseguiu manter o pêndulo dentro dos limites estabelecidos e no mesmo intervalo de tempo que os demais controladores. Porém, ele não conseguiu diminuir consideravelmente as oscilações, possivel-

Figura 37 – Primeiro Agente DQN nos 4,5 primeiros segundos (simulação 2)



Fonte: elaborada pelo autor.

Figura 38 – Primeiro Agente DQN nos 4,5 primeiros segundos (simulação 3)

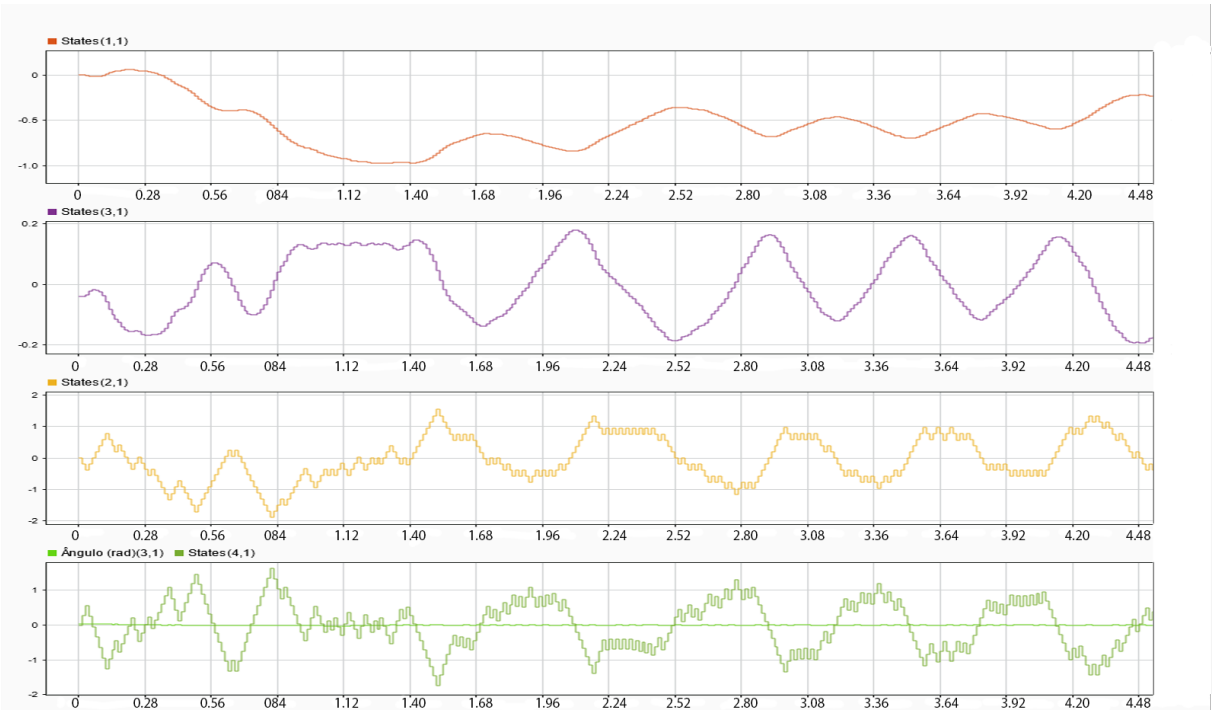


Fonte: elaborada pelo autor.

mente devido as configurações da rede neural. Na simulação 5, conforme a Figura 39, percebe-se que ele quase não diminui a amplitude durante o intervalo de tempo analisado. Nas demais simulações, assim como na simulação 5, o agente não conseguiu diminuir consideravelmente a amplitude ao redor do ângulo. Mas, conforme a Figura 40, pode-se observar que em alguns

momentos o agente conseguiu diminuir um pouco a amplitude na simulação 8. Por último, a simulação 10 foi bastante parecida com a 8, a Figura 41 mostra que o agente não foi capaz de fazer o pêndulo oscilar bem próximo de zero, mas, em alguns momentos, a amplitude também diminuiu um pouco.

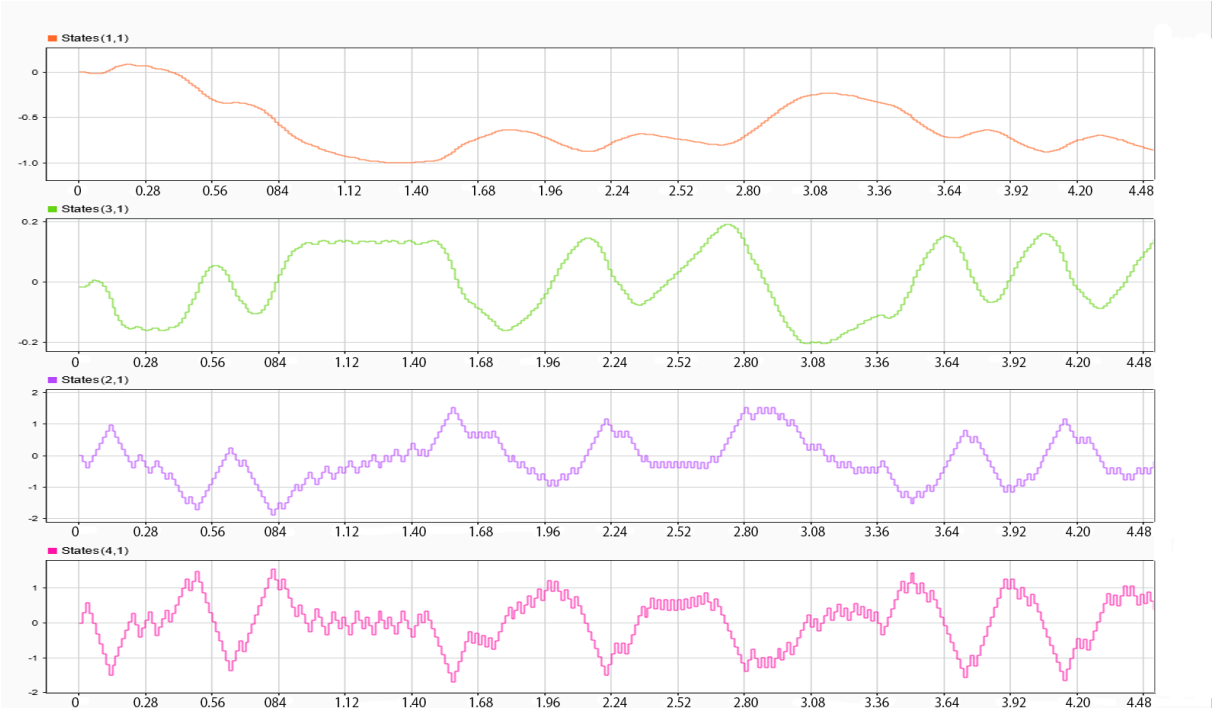
Figura 39 – Segundo Agente DQN nos 4,5 primeiros segundos (simulação 5)



Fonte: elaborada pelo autor.

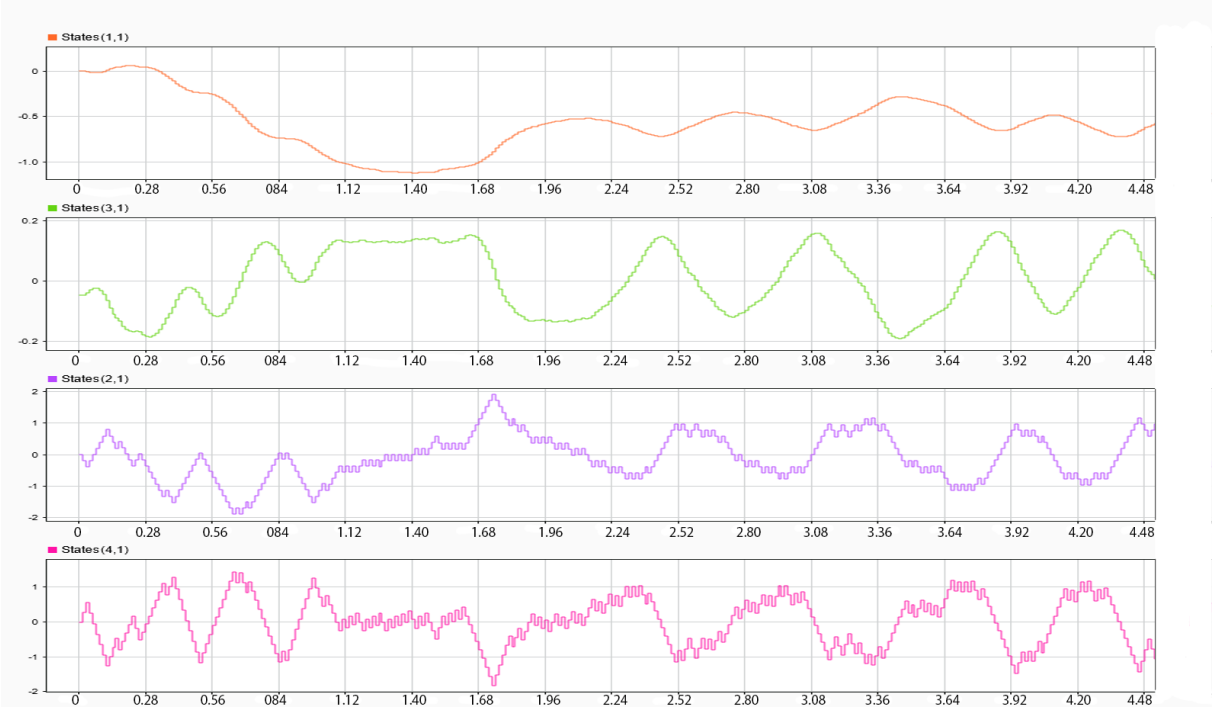
Com todas essas comparações, nas mesmas condições e no mesmo intervalo de tempo, demostra-se que os três controladores foram capazes de solucionar o problema. O primeiro praticamente elimina as oscilações ao redor do ponto de equilíbrio, o segundo mantem pequenas oscilações e o terceiro não diminui consideravelmente as oscilações. Embora cada um entregue resultados diferentes, todos conseguiram cumprir o objetivo: equilibrar o pêndulo em cima do carrinho, sem ultrapassar os limites. A Tabela 7 faz de forma resumida a consolidação dos resultados obtidos. Sendo o tempo para estabilizar, o tempo que o controlador leva para que o ângulo fique oscilando próximo de zero e a distância máxima obtida, a posição obtida mais distante da origem.

Figura 40 – Segundo Agente DQN nos 4,5 primeiros segundos (simulação 8)



Fonte: elaborada pelo autor.

Figura 41 – Segundo Agente DQN nos 4,5 primeiros segundos (simulação 10)



Fonte: elaborada pelo autor.

Tabela 7 – Resumo dos resultados obtidos

Controlador	**Tempo para estabilizar (s)**	**Distância máxima obtida (m)**
LQR a = 10	4,0	-0,0346
LQR a = 40	3,0	-0,0559
LQR a = 80	2,0	-0,0697
DQN 1 exp. 1	0,27	0,125
DQN 1 exp. 2	0,9	0,174
DQN 1 exp. 3	1,8	0,2
DQN 2 exp. 5	-	-0,98
DQN 2 exp. 8	-	-1,0
DQN 2 exp. 10	-	-1,12

Fonte: elaborada pelo autor.

6 CONCLUSÕES

Este trabalho consegue apresentar três maneiras para resolver um dos problemas mais clássicos de controle: o pêndulo invertido. O primeiro método foi utilizando variáveis de espaço de estado para criar um controle LQR, uma abordagem mais clássica que serviu como comparação para as demais técnicas apresentadas. O segundo e terceiro método foi utilizando a técnica de inteligência computacional aplicada conhecida como aprendizagem por reforço. Todas as simulações, visualizações e comparações foram feitas utilizando o *software* Matlab.

O controle LQR, apresentado na Seção 4.1, exibiu um excelente resultado. Foi possível avaliar como se comportou o sinal de controle variando determinados parâmetros, assim como o tempo que levou para o sistema ficar estável com essas modificações. Para um modelo linearizado, serviu como um bom exemplo de comparação para os outros dois controles.

O primeiro controle utilizando AR, apresentado na Seção 4.2, também apresentou um excelente resultado. Todo o processo de treinamento dos agentes levou aproximadamente cinco minutos para ser realizado, além de ter selecionado um total de 39 agentes que obtiveram a recompensa máxima. Durante as simulações utilizando um desses agentes, ele foi capaz de atingir a recompensa máxima 100% dos testes realizados, conforme mostra a Figura 25.

Por outro lado, o segundo controle utilizando AR, apresentado na Seção 4.3, não apresentou resultados tão agradáveis quanto aos demais controladores. O processo de treinamento dos agentes levou aproximadamente duas horas, além de ter selecionado apenas nove agentes. Durante as simulações de um desses agentes, ele atingiu a recompensa máxima em 50% dos testes realizados, conforme mostra a Figura 29.

Dentre os três controladores apresentados, o mais fácil de implementação é o primeiro, podendo ser facilmente configurado e modificado pelo usuário, porém ele tem a limitação de ser um modelo linearizado. No entanto, o segundo controlador demostrou ser uma opção mais segura para controlar o sistema se comparado com os outros dois, pois apresentou excelentes resultados em todas as simulações, diferentemente do terceiro controlador.

Com este trabalho, fica evidente, portanto, que o pêndulo invertido é um ótimo exemplo para conhecer os principais conceitos de AR e comparar com outras soluções mais difundidas na bibliografia. Ele pode ser considerado um "desafio de brinquedo"e uma porta de entrada para estudos mais complexos na área de AR.

6.1 Possibilidade de Trabalhos Futuros

Para trabalhos futuros as possibilidades são inúmeras. Uma delas é aplicar toda a teoria vista nesse trabalho e construir um controlador para fazer testes reais e sair do campo das simulações. Modificar alguns parâmetros do segundo agente DQN, por exemplo as funções de ativação da rede neural ou o número de episódios da fase de treinamento, e comparar as mudanças. Atualmente outra ferramenta importantíssima utilizada em diversos projetos de AR é a linguagem Python, então uma sugestão de pesquisa seria substituir a abordagem do Matlab, vista nesse trabalho, por uma abordagem em Python. Além de tão utilizada quando, seria uma opção gratuita para quem não tem acesso ao Matlab. Por fim, é possível utilizar este trabalho como base para projetos bem mais complexos de AR.

REFERÊNCIAS

ANH, N.; MATSUHISA, H.; VIET, L.; YASUDA, M. Vibration control of an inverted pendulum type structure by passive mass–spring-pendulum dynamic vibration absorber. **Journal of Sound and Vibration**, v. 307, p. 187–201, 2007.

BASHEER, I. A.; HAJMEER, M. N. Artificial neural networks: fundamentals, computing, design, and application. **Journal of microbiological methods**, Elsevier, v. 43, p. 3–31, 2000.

BOEIRA, E.; ECKHARD, D. Pyvrft: A python package for the virtual reference feedback tuning, a direct data-driven control method. **Elsevier**, v. 11, n. 100383, 2019.

CAMPI, M.; LECCHINI, A.; SAVARESIC, S. Virtual reference feedback tuning: a direct method for the design of feedback controllers. **Elsevier**, n. 1, p. 1337 – 1346, 2002.

DORF, R. C.; BISHOP, R. H. **Modern Control Systems**. [*S. l.*]: Pearson, 2008. ISBN 978-0-13-206710-2.

HAYKIN, S. **Redes Neurais**: Princípios e prática. 2. ed. [*S. l.*]: Prentice Hall, 1999. ISBN 0-13-273350-1.

HEBB, D. O. **The organization of behavior**. [*S. l.*]: Wiley, 1949.

HEHN, M.; D'ANDREA, R. A flying inverted pendulum. **IEEE**, p. 763–770, 2011.

HODGKIN, A. L.; HUXLEY, A. F. A quantitative description of membrane current and its application to conduction and excitation in nerve. **Journal of Physiology**, v. 117, p. 500–544, 1952.

KAELBLING, L. P.; LITTMAN, M. L.; MOORE, A. W. Reinforcement learning: A survey. **JAIR**, Journal of Articial Intelligence Research, v. 4, p. 237–285, 1996.

KAFETZIS, I.; MOYSIS, L. **Inverted Pendulum: A system with innumerable applications**. [*S. l.*]: 9th International Week Dedicated to Maths, 2017.

KAREEM, A. Nonlinear dynamic analysis of compliant offshore platforms subjected to fluctuating wind. **Journal of Wind Engineering and Industrial Aerodynamics**, v. 14, p. 345–356, 1983.

KINGMA, D. P.; BA, J. L. Adam: A method for stochastic optimization. **ICLR**, arXiv:1412.6980, 2015.

KRUL, A. M. **APRENDIZAGEM POR REFORÇO COMO TÉCNICA DE CONTROLE PARA O PROBLEMA DO PÊNDULO INVERTIDO**. 2021.

LANDAU, I. D.; ZITO, G. **Digital Control Systems**: Design, identification and implementation. [*S. l.*]: Springer, 2006. ISBN 978-1-84628-055-9.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. **Bulletin of Mathematical Biophysics**, v. 5, p. 115–133, 1943.

MNIH, V.; KAVUKCUOGLU, K.; SILVER, D.; RUSU, A. A.; VENESS, J. Human-level control through deep reinforcement learning. **Nature**, v. 518, p. 529–533, 2015.

NAGENDRA, S.; PODILA, N.; UGARAKHOD, R.; GEORGE, K. Comparison of reinforcement learning algorithms applied to the cart-pole problem. **IEEE**, p. 26–32, 2017.

NISE, N. S. **Engenharia de Sistemas de Controle**. [*S. l.*]: LTC, 2012. ISBN 978-85-216-2136-2.

OGATA, K. **Engenharia de Controle Moderno**. [*S. l.*]: Pearson, 2010. ISBN 978-85-4301-375-6.

PESSOTTI, L. R. **UMA AVALIAÇÃO DA UTILIZAÇÃO DE APRENDIZADO POR REFORÇO PARA O CONTROLE DE SISTEMAS DE TEMPO CONTÍNUO NÃO LINEARES**. 2022.

RAMACHANDRAN, P.; ZOPH, B.; LE, Q. V. Searching for activation functions. arXiv:1710.05941v2, 2017.

REECE, J. B.; URRY, L. A.; CAIN, M. L.; WASSERMAN, S. A.; MINORSKY, P. V.; JACKSON, R. B. **Biologia de Campbell**. 10. ed. [*S. l.*]: Artmed, 2015. ISBN 978-85-8271-230-6.

ROSENBLATT, F. The perceptron: A probabilistic model for information storage and organization in the brain. **Psychological Review**, v. 65, p. 386–408, 1958.

RUMELHART, D. E.; MCCLELLAND, J. L. **Parallel Distributed Processing**: Explorations in the microstructure of cognition: Foundations. 1. ed. Cambridge, Massachusetts, EUA: MIT Press, 1987. ISBN 978-0262680530.

SILVA, I. N. da; SPATTI, D. H.; FLAUZINO, R. A. **Redes Neurais Artificiais**: Para engenharia e ciÊncias aplicadas. 2. ed. São Paulo: Artliber, 2010. ISBN 978-8588098879.

SUTTON, R. S.; BARTO, A. G. **Reinforcement Learning**: An introduction. 2. ed. [*S. l.*]: The MIT Press, 2018. ISBN 978-0-262-19398-6.

APÊNDICE A – CÓDIGO PRIMEIRO CONTROLE

```
%%
clc, clear, close all

%% Parametros
g = 9.8;    % gravidade
m0 = 1;  % massa do carro
m1 = 0.1;  % massa do pendulo
L = 0.5;         % compimento da haste
I = (m1*L^2)/12; %momento de inercia

%% Matrizes
%
% x_dot = Ax + Bu
% y     = Cx + Du
%
A = [0,                                          0,    1,    0;
     0,                                          0,    0,    1;
     0,         g*L^2*m1^2/(L^2*m0*m1+4*I*(m0+m1)),    0,    0;
     0, -2*g*L*m1*(m0+m1)/(L^2*m0*m1+4*I*(m0+m1)),    0,    0];

B = [0; 0; (4*I+L^2*m1)/(L^2*m0*m1+4*I*(m0+m1)); -(2*L*m1)/(L^2*m0*m1
    +4*I*(m0+m1))];

C = eye(4);

D = [0; 0; 0; 0];

%% Sistema
sys = ss(A,B,C,D)

%% Controle
Q = diag([1,10,0,0]);
R = 1;
K_lqr = lqr(sys,Q,R)

%% Sistema Controlado
sys_new = ss(A-B*K_lqr,B,C,D)
```

```
37
38 %% Plot
39 x0 = [0;deg2rad(8);0;deg2rad(0)]; %estados iniciais
40 [y,t,x]= initial(sys_new,x0,10);
41 y(:,2)=rad2deg(y(:,2)); %transform from rad to degrees.
42 y(:,4)=rad2deg(y(:,4));
43
44 subplot(4,1,1)
45 plot(t,y(:,1))
46 title('Estados do sistema')
47 legend('Posicao')
48 grid on
49
50 subplot(4,1,2)
51 plot(t,y(:,2))
52 legend('Angulo')
53 grid on
54
55 subplot(4,1,3)
56 plot(t,y(:,3))
57 legend('Derivada da posicao')
58 grid on
59
60 subplot(4,1,4)
61 plot(t,y(:,4))
62 legend('Derivada do angulo')
63 grid on
```

APÊNDICE B – CÓDIGO SEGUNDO CONTROLE

```
clc, clear, close all

env = rlPredefinedEnv("CartPole-Discrete");

%% Criando o agente
obsInfo = getObservationInfo(env);
actInfo = getActionInfo(env);

dnn = [
    featureInputLayer(prod(obsInfo.Dimension))
    fullyConnectedLayer(24)
    reluLayer
    fullyConnectedLayer(24)
    reluLayer
    fullyConnectedLayer(numel(actInfo.Elements))
    ];

dnn = dlnetwork(dnn);
summary(dnn)

critic = rlVectorQValueFunction(dnn,obsInfo,actInfo);

getValue(critic,{rand(obsInfo.Dimension)})

agent = rlDQNAgent(critic)

agent.AgentOptions.UseDoubleDQN=false;
agent.AgentOptions.TargetUpdateMethod="periodic";
agent.AgentOptions.TargetUpdateFrequency=4;
agent.AgentOptions.ExperienceBufferLength=100000;
agent.AgentOptions.DiscountFactor=0.99;
agent.AgentOptions.MiniBatchSize=256;
agent.AgentOptions.CriticOptimizerOptions.LearnRate=1e-2;
agent.AgentOptions.CriticOptimizerOptions.GradientThreshold=1;

getAction(agent,{rand(obsInfo.Dimension)})

```

```
%% Treinando o agente
trainOpts = rlTrainingOptions;

trainOpts.MaxEpisodes = 500;
trainOpts.MaxStepsPerEpisode = 500;
trainOpts.StopTrainingCriteria = "AverageReward";
trainOpts.StopTrainingValue = 500;
trainOpts.ScoreAveragingWindowLength = 5;

trainOpts.SaveAgentCriteria = "EpisodeReward";
trainOpts.SaveAgentValue = 500;
trainOpts.SaveAgentDirectory = "savedAgents";

trainOpts.Verbose = false;
trainOpts.Plots = "training-progress";

plot(env)

trainingInfo = train(agent,env,trainOpts);

%% Simulacao
simOptions = rlSimulationOptions(MaxSteps=500);
experience = sim(env,agent,simOptions);

totalReward = sum(experience.Reward)
```

APÊNDICE C – CÓDIGO TERCEIRO CONTROLE

```
clc, clear, close all

env = rlPredefinedEnv("CartPole-Discrete");

%% Criando o agente
obsInfo = getObservationInfo(env);
actInfo = getActionInfo(env);

net = [
    sequenceInputLayer(prod(obsInfo.Dimension))
    fullyConnectedLayer(24)
    reluLayer
    lstmLayer(20,OutputMode="sequence");
    fullyConnectedLayer(24)
    reluLayer
    fullyConnectedLayer(numel(actInfo.Elements))
    ];

net = dlnetwork(net);
summary(net);

critic = rlVectorQValueFunction(net,obsInfo,actInfo);

getValue(critic,{rand(obsInfo.Dimension)})

criticOptions = rlOptimizerOptions( ...
    LearnRate=1e-2, ...
    GradientThreshold=1);

agentOptions = rlDQNAgentOptions(...
    UseDoubleDQN=false, ...
    TargetSmoothFactor=1e-3, ...
    ExperienceBufferLength=1e4, ...
    SequenceLength=32, ...
    CriticOptimizerOptions=criticOptions);

agentOptions.EpsilonGreedyExploration.EpsilonDecay = 0.0050;
```

```
38
39 agent = rlDQNAgent(critic,agentOptions)
40
41 getAction(agent,rand(obsInfo.Dimension))
42
43 %% Treinando o agente
44 trainOpts = rlTrainingOptions;
45
46 trainOpts.MaxEpisodes = 500;
47 trainOpts.MaxStepsPerEpisode = 500;
48 trainOpts.StopTrainingCriteria = "AverageReward";
49 trainOpts.StopTrainingValue = 500;
50 trainOpts.ScoreAveragingWindowLength = 5;
51
52 trainOpts.SaveAgentCriteria = "EpisodeReward";
53 trainOpts.SaveAgentValue = 500;
54 trainOpts.SaveAgentDirectory = "savedAgents";
55
56 trainOpts.Verbose = false;
57 trainOpts.Plots = "training-progress";
58
59 plot(env)
60
61 trainingInfo = train(agent,env,trainOpts);
62
63 %% Simulacao
64 simOptions = rlSimulationOptions(MaxSteps=500);
65 experience = sim(env,agent,simOptions);
66
67 totalReward = sum(experience.Reward)
```