UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO, DE CIÊNCIAS EXATAS E EDUCAÇÃO
DEPARTAMENTO DE ENG. DE CONTROLE, AUTOMAÇÃO E COMPUTAÇÃO
CURSO DE GRADUAÇÃO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Luiz Adelar Soldatelli Neto

Cinemática Inversa de Manipuladores Robóticos Baseada em Aprendizado por Reforço e Redes Neurais

Blumenau
2022

Luiz Adelar Soldatelli Neto

Cinemática Inversa de Manipuladores Robóticos Baseada em Aprendizado por Reforço e Redes Neurais

Trabalho de Conclusão de Curso de Graduação em Engenharia de Controle e Automação do Centro Tecnológico, de Ciências Exatas e Educação da Universidade Federal de Santa Catarina como requisito para a obtenção do título de Engenheiro de Controle e Automação.
Orientador: Prof. Dr. Maiquel de Brito

Blumenau
2022

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Neto, Luiz
Cinemática Inversa de Manipuladores Robóticos Baseada em Aprendizado por Reforço e Redes Neurais / Luiz Neto ; orientador, Maiquel de Britto, 2022.
70 p.

Tese (doutorado) - Universidade Federal de Santa Catarina, , Programa de Pós-Graduação em , Florianópolis, 2022.

Inclui referências.

1. . 2. Cinemática Inversa. 3. Inteligência Artificial. 4. Aprendizado por Reforço. 5. Redes Neurais. I. de Britto, Maiquel. II. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em . III. Título.

Luiz Adelar Soldatelli Neto

Cinemática Inversa de Manipuladores Robóticos Baseada em Aprendizado por Reforço e Redes Neurais

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de "Engenheiro de Controle e Automação" e aprovado em sua forma final pelo Curso de Graduação em Engenharia de Controle e Automação.

Blumenau, 20 de 07 de 2022.

Banca Examinadora:

Documento assinado digitalmente
Maiquel de Brito
Data: 03/08/2022 10:53:47-0300
CPF: 979.657.150-15
Verifique as assinaturas em https://v.ufsc.br

Prof. Dr. Maiquel de Brito
Universidade Federal de Santa Catarina

Documento assinado digitalmente
Leonardo Mejia Rincon
Data: 03/08/2022 10:00:52-0300
CPF: 011.678.849-65
Verifique as assinaturas em https://v.ufsc.br

Prof. Dr. Leonardo Mejia Rincon
Universidade Federal de Santa Catarina

Documento assinado digitalmente
Mauri Ferrandin
Data: 03/08/2022 10:40:05-0300
CPF: 018.580.869-73
Verifique as assinaturas em https://v.ufsc.br

Prof. Dr. Mauri Ferrandin
Universidade Federal de Santa Catarina

AGRADECIMENTOS

Agradeço, primeiramente, aos meus pais Itália e Luiz Tadeu (*in memorian*), por me concederem a base, estrutura e oportunidade para engajar em meus estudos.

Ao meu irmão e minha cunhada Bruno e Juliana, que juntos formam meu inesgotável poço de amor, me fornecendo combustível de sobra para sempre ir mais longe e seguir meus sonhos.

Aos meus amigos e colegas de faculdade Dartagnan e José Baretta, que compartilharam comigo essa jornada turbulenta mas satisfatória da graduação. Alguns neurônios foram queimados, algumas noites não foram dormidas, mas o sucesso junto a estes dois sempre foi garantido!

Ao meu ex colega de quarto, José Vitor, e ao meu querido amigo Gabriel Bollmann. Sempre que precisei desabafar, vocês estavam ali pra me dar todo o apoio. Vocês são a prova de que as boas amizades compõem a mais simples operação matemática: divisão das tristezas e multiplicação das alegrias. Amo vocês.

Aos meus professores, em especial ao Prof. Dr. Marcelo Esposito, que me incentivou na transferência para a UFSC Blumenau, e ao Prof. Dr. Maiquel de Brito, que apesar desta etapa do TCC ter tomado mais de um semestre, atuou em seu papel de orientador com verdadeira maestria. Eu não poderia ter escolhido um melhor corpo docente.

Agradeço à minha namorada Alana, que aliviou não apenas um, mas alguns de meus surtos. Enquanto eu pensava em desistir, lá vinha ela com seus chacoalhões. Ela me ensinou que, independente do que você esteja fazendo, quando você está com alguém que esteja na mesma sintonia, você vai muito mais longe.

Por último, mas não menos importante, agradeço ao meu primo Rissieri. Em uma curta porém inspiradora conversa, recordou-me de que a vida é feita de etapas, e que estas exigem foco. Era a força que eu precisa para dar fim à esta.

RESUMO

Quando um ser humano abre uma porta, o cérebro calcula as posições de cada uma das articulações do braço, para que a mão seja levada à maçaneta. Este processo chama-se *cinemática inversa*, e, quando trata-se de manipuladores robóticos, este é associado ao calculo das posições de cada uma das juntas atuadas do mecanismo para que o elo de interesse seja levado à uma posição desejada.

O cálculo das posições das juntas atuadas do mecanismo de um manipulador robótico, para satisfazer a cinemática inversa de forma analítica, pode facilmente tornar-se uma tarefa complexa: é comum deparar-se com equações não lineares, múltiplas soluções ou até casos em que as combinações das posiçoes de juntas encontradas são fisicamente impossíveis de serem replicadas pelo manipulador.

Este trabalho apresenta uma solução baseada em *Q-Learning* e redes neurais. A combinação de *Q-Learning*, técnica de inteligência artificial e aprendizado de máquina, com o apoio de redes neurais, dá origem ao algoritmo *Deep Q-Learning*, implementado neste trabalho.

Palavras-chave: Cinemátiva Inversa; Tabela-Q; Redes Neurais; Aprendizado de Máquina; Aprendizado por Reforço

ABSTRACT

When a human being opens a door, the brain calculates the positions of each one of its arm's articulations, so the hand of interest can be moved to the door handle. This process is called *inverse kinematic*, and, when it's associated with robotic manipulators, the process is related to the position calculation for each one of the actuated manipulator joints, so the link of interest is placed on the desired position.

The calculation of the manipulator's actuated joints positions, to satisfy the inverse kinematic analytically, can easily turn into a complex task: It usually takes non linear equations to solve, multiple solutions or even cases where the calculated positions are phisically impossible to be replicated by the manipulator.

This paper presents a solution based on *Q-Learning* and neural networks. The combination of *Q-Learning*, artificial intelligence and machine learning technique, powered by neural networks, gives origin to the *Deep Q-Learning* algorithm, implemented on this paper.

Keywords: Inverse Kinematics; Q-Table; Neural Networks; Machine Learning; Reinforcement Learning

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

É claramente notável o papel das máquinas no contexto da sociedade contemporânea. O ser humano, há algumas décadas, vem construindo autômatos semelhantes a si, robôs, que possam o substituir em tarefas repetitivas ou pesadas, como por exemplo, as tarefas industriais, abundantes desde a primeira revolução industrial. A ciência da robótica é um campo de estudo tão promissor que, inevitavelmente, torna-se alvo de interesse em diversos segmentos industriais, como a indústria farmacêutica (Figura 1), automotiva, alimentícia (Figura 2) e agroindustra. Até fora do espectro industrial, como a bio-medicina, é possível notar a busca por soluções inspiradas em robótica. A partir dos anos 2000, os robôs começaram a ser desenvolvidos com um número cada vez maior de tecnologias embarcadas, sendo então aptos a realizar processamentos computacionais avançados, estratégias complexas, comportamento colaborativo e *deep learning* (GASPARETTO; SCALERA, 2019).

Figura 1 – Um dos primeiros protótipos do robô SCARA.

Fonte: (GASPARETTO; SCALERA, 2019).

Robôs são máquinas que imitam o comportamento de corpos animados. Estes movimentos podem ser reproduzidos através de uma ou mais juntas atuadas, também chamados de manipuladores robóticos. Por definição, um mecanismo é construído pela conexão de partes rígidas, conhecidas como elos, por partes móveis denominadas juntas, resultando em um artefato que movimenta-se conforme esperado ou projetado (Figura 3). Denomina-se

Figura 2 – Robôs Delta na célula de manufatura da companhia DEMAUREX SA®, empacontando pretzels.

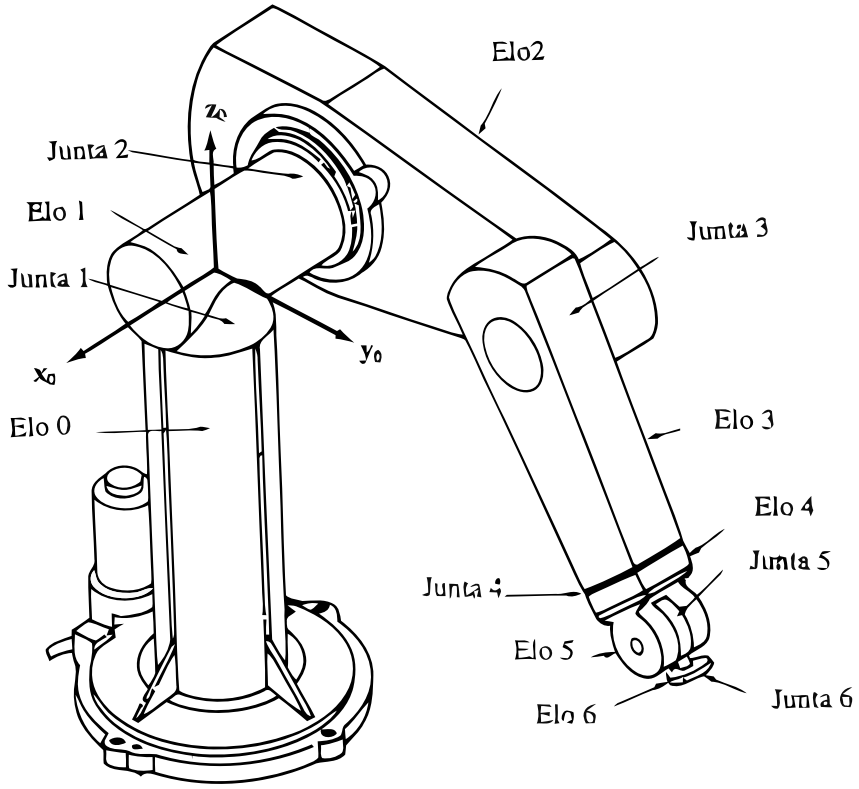
Fonte: (GASPARETTO; SCALERA, 2019).

manipulador robótico o mecanismo que possui como função a manipulação de materiais sem que haja a necessidade de contato físico pelo operador. Normalmente, são mecanismos que possuem juntas atuadas, controladas por um operador ou programadas para realizar tarefas especificas. Entende-se como junta atuada uma junta do mecanismo que é movimentada, tipicamente por motores elétricos de posicionamento (LYNCH; PARK, 2017). O controle sobre o posicionamento das juntas do mecanismo permite que o manipulador robótico seja movimentado de forma programada, obedecendo rotinas de tarefas sequenciais pré-programadas ou algum algoritmo que seja responsável por ditar o posicionamento de suas juntas (LYNCH; PARK, 2017).

Compreender as características físicas do manipulador, a partir de sua configuração de elos e juntas, permitem formular as equações matemáticas que regem o comportamento dinâmico e de posição do manipulador corretamente. Define-se como *espaço de juntas* as posições, em graus, de cada junta do manipulador, e *espaço Cartesiano* as coordenadas das posições no espaço tridimensional de cada uma das juntas. I.e., para cada posição da junta de interesse do manipulador no espaço Cartesiano há, ao menos, uma combinação de posições angulares em cada junta (espaço de juntas) para que a posição no espaço Cartesiano seja satisfeita. São duas formas diferentes de representar a atual posição do manipulador no espaço em um determinado instante, e estão diretamente relacionadas. *Cinemática direta* refere-se à transformação da representação no espaço de juntas para o espaço Cartesiano, enquanto o sentido inverso de transformação é denominado *cinemática inversa*, como demonstrado na Figura 4 (KUCUK, Serdar; BINGUL, Zafer, 2006).

A solução do problema da cinemática direta tende a ser pouco complexa, independente da quantidade de juntas e elos do manipulador. Sempre haverá uma solução da

Figura 3 – Robô manipulador PUMA mostrando juntas e elos.



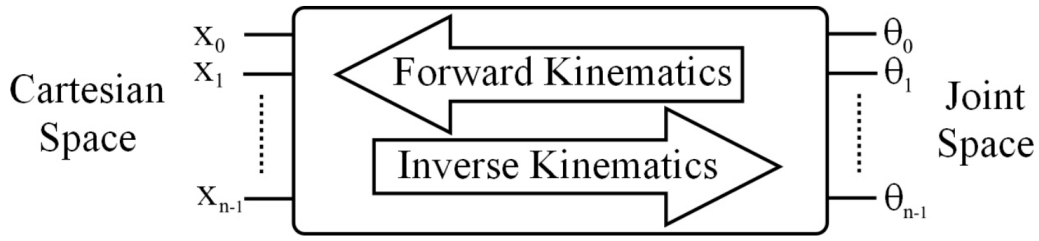
Fonte: (LOPES, 2001).

cinemática direta para qualquer manipulador robótico, seja por análise geométrica ou por metodologias mais poderosas. Por outro lado, calcular a cinemática inversa pode facilmente tornar-se uma tarefa árdua. Resolver o problema da cinemática inversa analiticamente, através da análise geométrica do manipulador, pode suceder em cálculos de equações não lineares, podendo haver mais de uma solução, ou até mesmo as combinações de ângulos que satisfazem a solução serão fisicamente incompatíveis com a configuração de juntas do manipulador. Ademais, as soluções, quando encontradas, irão atender somente os modelos em questão, sendo necessário formular uma nova solução para cada configuração diferente de manipulador (KUCUK, S.; BINGUL, Z., 2004).

Para solucionar o problema da cinemática inversa de forma geral, independente da configuração do manipulador, muitos algoritmos de minimização de erro foram propostos. Os algoritmos, em grande parte focados em redes neurais ou algoritmos genéticos, não garantem soluções exatas, que seria a combinação de ângulos das juntas do manipulador que levam o manipulador exatamente às posições tridimensionais desejadas. As soluções estão associadas a um erro, que depende de variáveis como o número de iterações de

treinamento e poder computacional (KÖKER, 2013).

Figura 4 – Representação de cinemáticas direta e inversa.



Fonte: Adaptado de (KENWRIGTH, 2013).

1.1 OBJETIVO GERAL

O objetivo deste trabalho é implementar um algoritmo para solucionar a cinemática inversa de manipuladores robóticos para diferentes posições desejadas. O algoritmo deverá, também, ser arquitetado visando apenas alterações de parâmetros no código fonte quando modificada a configuração do manipulador. I.e., O código fonte do algoritmo não deve ser modificado. Para buscar os resultados desejados, será utilizado *aprendizado por reforço*, técnica de aprendizado de máquina baseado na utilização de sistemas de recompensa e punição para ações tomadas durante a fase de aprendizado do algoritmo, e Redes Neurais.

1.2 OBJETIVOS ESPECÍFICOS

Desenvolvimento, implementação e avaliação dos resultados de algoritmo capaz de solucionar a cinemática inversa de manipuladores robóticos, sendo que:

- O algoritmo deve utilizar aprendizado por reforço e redes neurais;
- O algoritmo deve ser implementado para que mínimas alterações possíveis no código fonte sejam necessárias quando modificada a estrutura do manipulador robótico;
- O algoritmo deve atender qualquer manipulador robótico, independente de sua complexidade.

1.3 ESTRUTURA DO TRABALHO

Este documento é dividido em quatro seções principais, incluindo esta. Nesta seção, **Introdução**, é apresentada uma breve introdução dos conceitos abordados, assim como os objetivos do trabalho. Na seção **Embasamento Teórico**, é fundamentada a base teórica

utilizada no desenvolvimento do algoritmo. Na seção **Metodologia**, como descrito no título da seção, são demonstrados os métodos utilizados durante o desenvolvimento. Por último, na seção **Resultados e Conclusão**, é apresentado os resultados obtidos e as análises destes, concluindo o trabalho e sugerindo implementações futuras.

2 EMBASAMENTO TEÓRICO

Esta seção é composta pela fundamentação teórica e bibliográfica que são utilizadas com base para o desenvolvimento deste trabalho. É descrita a principal ferramenta de solução para cinemática direta de manipuladores robóticos, que será necessária para a obtenção da cinemática inversa pelo algoritmo. É também realizada uma revisão bibliográfica dos conceitos de Aprendizado de Máquina, esclarecendo onde as técnicas *Q-Learning* e *Deep Q-Learning* estão localizadas dentro no espectro da Inteligência Artificial.

2.1 CINEMÁTICA DE MANIPULADORES ROBÓTICOS

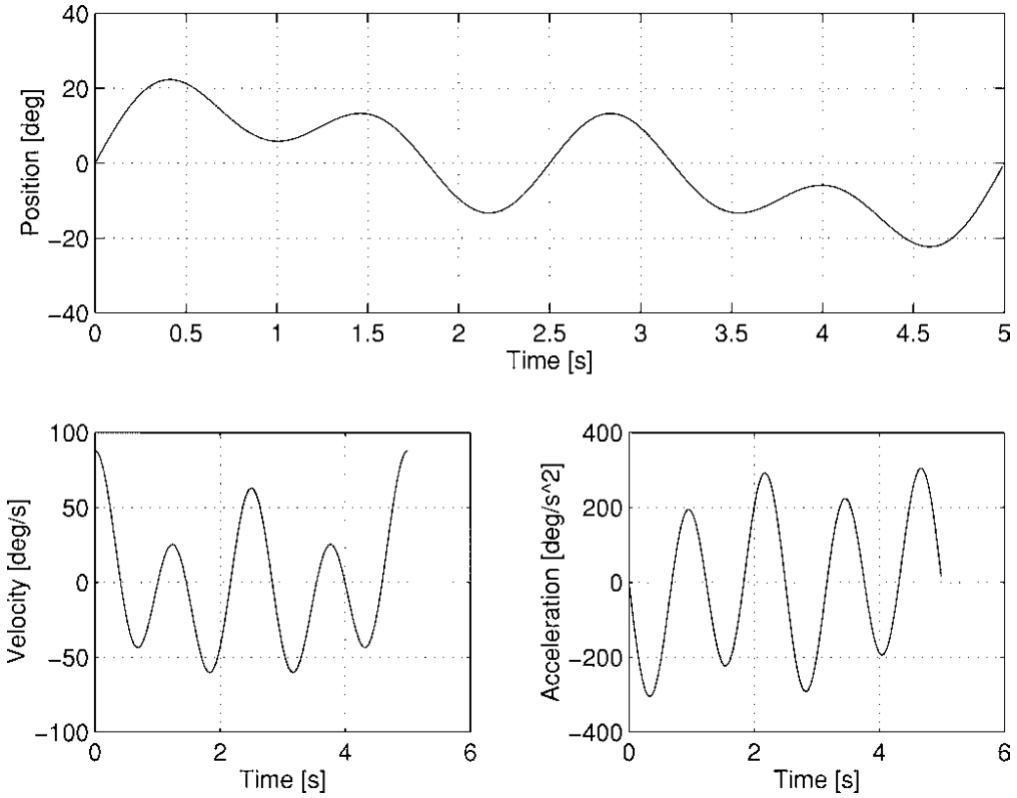
De forma geral, cinemática é considerada a ciência que estuda os movimentos de estruturas geométricas. Dento do escopo da robótica, trata-se da modelagem de sistemas e equações, a partir da álgebra de vetores e matrizes, que regem o movimento dos manipuladores robóticos em respeito a um referencial de coordenadas no espaço (JAZAR, 2010).

Considerando o escopo deste trabalho, somente será levado em consideração as posições estáticas dos manipuladores e suas geometrias. Entretanto, o estudo da cinemática de manipuladores é uma ciência que também abrange as propriedades baseadas no tempo (Figura 5), como a relação entre torques, forças, velocidades e acelerações nas juntas e elos que resultam no movimento dinâmico do manipulador robótico (CRAIG, 2005).

Mesmo que o objetivo do algoritmo a ser implementado neste trabalho seja a solução da cinemática inversa de manipuladores robóticos, a cinemática direta ainda deve ser modelada. Esta será utilizada com referência qualitativa para avaliar os resultados obtidos pelo algoritmo. Por exemplo, quando uma configuração de posições angulares dos atuadores for definida pelo algoritmo, precisamos aplicar estas posições nas entradas do modelo matemático da cinemática direta para, consequentemente, saber a posição tridimensional do elo de interesse no espaço e comparar com a posição desejada.

Definindo um manipulador robótico como um mecanismo de elos atuados por motores e ligados por juntas, a cinemática direta pode ser representada por um modelo de sistema que resulta na posição final tridimensional de um elo de interesse em função das posições angulares das juntas, caso haja somente juntas rotacionais no manipulador. Usualmente, o elo de interesse é o último elo do manipulador quando trata-se de manipuladores robóticos humanoides: é no último elo que são fixadas as garras, instrumentos de solda, furadores ou qualquer que seja o mecanismo utilizado pelo robô em sua empregabilidade. Na Figura 3, o elo de interesse deve ser referente à posição final do mecanismo acoplado à junta 6.

Figura 5 – Exemplo de posição, velocidade e aceleração em funções do tempo de junta de manipulador em movimento.



Fonte: (PARRA-VEGA *et al.*, 2003).

2.1.1 Cinemática Direta - Método de Denavit-Hartenberg

Existem algumas técnicas para modelar a cinemática direta: vetor de Gibbs, parâmetros de Cayley-Klein, matrizes de Pauli, *Hamilton's quaternions*, entre outros. Porém, o método mais frequentemente utilizado nas bibliografias de robótica é o método de Denavit-Hartenberg. O método é considerado, inclusive, um padrão para descrever cinemática de manipuladores robóticos. Uma das vantagens deste método é a padronização da metodologia. Independente da configuração do manipulador, os passos executados para o cálculo da cinemática direta através desta técnica são quase sempre idênticos.

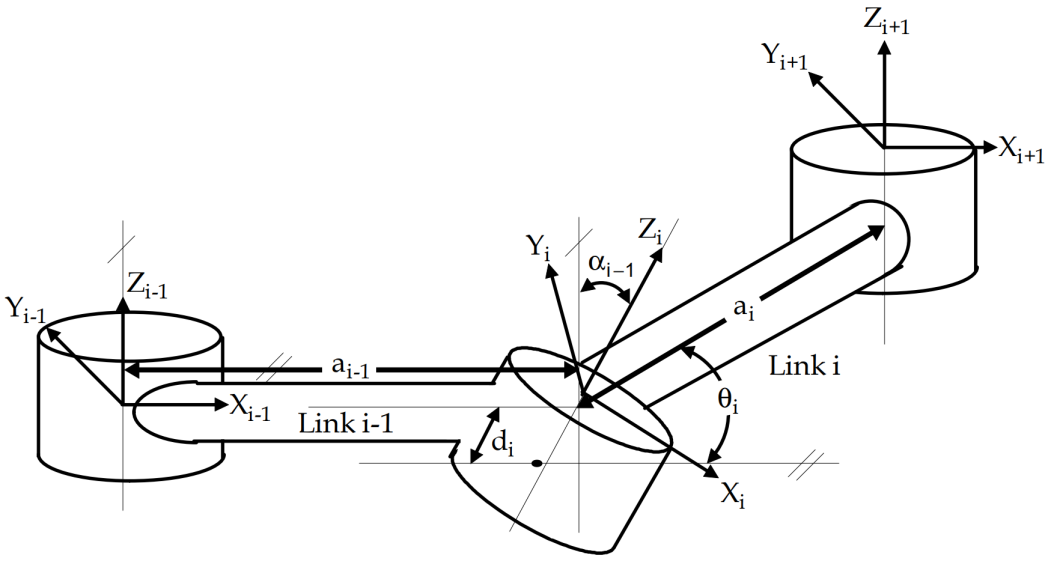
O método de Denavit-Hartenberg utiliza quatro parâmetros para descrever a cinemática direta: a_{i-1}, α_{i-1}, d_i e θ_i. Cada junta do manipulador pode ser associada com próprios sub-sistemas de eixos tridimensionais, como ilustra a Figura 6. Os parâmetros de Denavit-Hartenberg carregam as informações que associam estes sub-sistemas de eixos em cada junta, criando uma relação entre elas (KUCUK, Serdar; BINGUL, Zafer, 2006). Utilizando a Figura 6 como exemplo e tomando as juntas $i-1$ e i como, respectivamente, as juntas inicial e final, os parâmetros de Denavit-Hartenberg que definem suas relações

são dadas por:

- A distância entre os eixos Z_{i-1} e Z_i ao longo de X_{i-1} é dada pelo parâmetro a_{i-1};
- O deslocamento angular entre Z_{i-1} e Z_i é dado por α_{i-1};
- A distância entre X_{i-1} e Xi ao longo de Z_i é dado por d_i;
- O deslocamento angular entre X_{i-1} e X_i em Z_i é dado por θ_i.

A partir dos parâmetros acima, é possível construir a matriz de transformação homogênea, conforme descrito na Equação (1). Em robótica, as matrizes de transformação homogênea são utilizadas para descrever tanto a posição quanto a orientação de componentes do robô (FONSECA, 2017).

Figura 6 – Exemplo de eixos de coordenadas de cada junta em manipulador genérico, especificando os parâmetros de Denavit-Hartenberg.



Fonte: (KUCUK, Serdar; BINGUL, Zafer, 2006).

$$
{}^{i-1}T_i = \left[\begin{array}{ccc|c}
\cos\theta_i & -\sin\theta_i & 0 & a_{i-1} \\
\sin\theta_i\cos\alpha_{i-1} & \cos\theta_i\cos\alpha_{i-1} & -\sin\alpha_{i-1} & -\sin\alpha_{i-1}d_i \\
\sin\theta_i\sin\alpha_{i-1} & \cos\theta_i\sin\alpha_{i-1} & \cos\alpha_{i-1} & \cos\alpha_{i-1}d_i \\
\hline
0 & 0 & 0 & 1
\end{array}\right] \tag{1}
$$

A Equação (1) pode ser simplificada como Equação (2) ou Equação (3). Os elementos $r_{(k,j)}$ ou R descrevem as informações da orientação espacial da junta final, enquanto os elementos P_x, P_y e P_z, ou T, descrevem as informações de posição tridimensional no espaço tridimensional (KUCUK, Serdar; BINGUL, Zafer, 2006).

$$ {}^{i-1}T_i = \begin{bmatrix} \mathrm{r}_{11} & \mathrm{r}_{12} & \mathrm{r}_{13} & \mathrm{p}_\mathrm{x} \\ \mathrm{r}_{21} & \mathrm{r}_{22} & \mathrm{r}_{23} & \mathrm{p}_\mathrm{y} \\ \mathrm{r}_{31} & \mathrm{r}_{32} & \mathrm{r}_{33} & \mathrm{p}_\mathrm{z} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2} $$

$$ {}^{i-1}T_i = \left[\begin{array}{ccc|c} & R & & T \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{3} $$

Sendo um manipulador robótico a união de várias juntas por elos, e cada par de juntas em sequência pode ser acoplado pela matriz de transformação homogênea, caso o manipulador possua n juntas, basta calcular as matrizes de todos os pares unidos por elos. Obtendo todas as matrizes de transformação homogênea, basta multiplicar as matrizes sequencialmente, respeitando a ordem sequencial da junta da base até a última junta de interesse do manipulador, e será possível calcular a orientação e a posição tridimensional do último elo em função das posições angulares dos atuadores, como descrito na Equação (4) (KUCUK, Serdar; BINGUL, Zafer, 2006).

$$ {}^{0}T_6 = {}^{0}T_1\,(\mathrm{junta}_1) \cdot {}^{1}T_2\,(\mathrm{junta}_2) \cdot {}^{2}T_3\,(\mathrm{junta}_3) \cdot {}^{3}T_4\,(\mathrm{junta}_4) \cdot {}^{4}T_5\,(\mathrm{junta}_5) \cdot {}^{5}T_6\,(\mathrm{junta}_6) \tag{4} $$

2.1.2 Ângulos de Euler

A partir da matriz R, da matriz exemplificada na Equação (3), pode-se extrair os ângulos de Euler. Estes ângulos representam as rotações sequenciais realizadas no elo de interesse ao redor de cada um dos eixos X, Y e Z. Por exemplo, para descrever uma alteração na orientação de um objeto qualquer no espaço tridimensional a partir de sua orientação inicial, primeiro rotaciona-se o objeto ao redor do eixo X, depois ao redor do eixo Y e, por fim, ao redor do eixo Z. Os ângulos rotacionados ao redor de cada um dos eixos são chamados de ângulos de Euler, simbolizados sequencialmente como α, β, e γ. Se a matriz de rotação R for a Equação (5), os ângulos podem ser obitdos, respectivamente, pela Equação (6), Equação (7) e Equação (8) (HENDERSON, 1977).

$$ R = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix} \tag{5} $$

$$\alpha = \tan^{-1}\left(\frac{R_{32}}{R_{22}}\right) \tag{6}$$

$$\beta = \tan^{-1}\left(\frac{-R_{12}}{\sqrt{1-(R_{12})^2}}\right) \tag{7}$$

$$\gamma = \tan^{-1}\left(\frac{R_{13}}{R_{11}}\right) \tag{8}$$

2.2 APRENDIZADO DE MÁQUINA

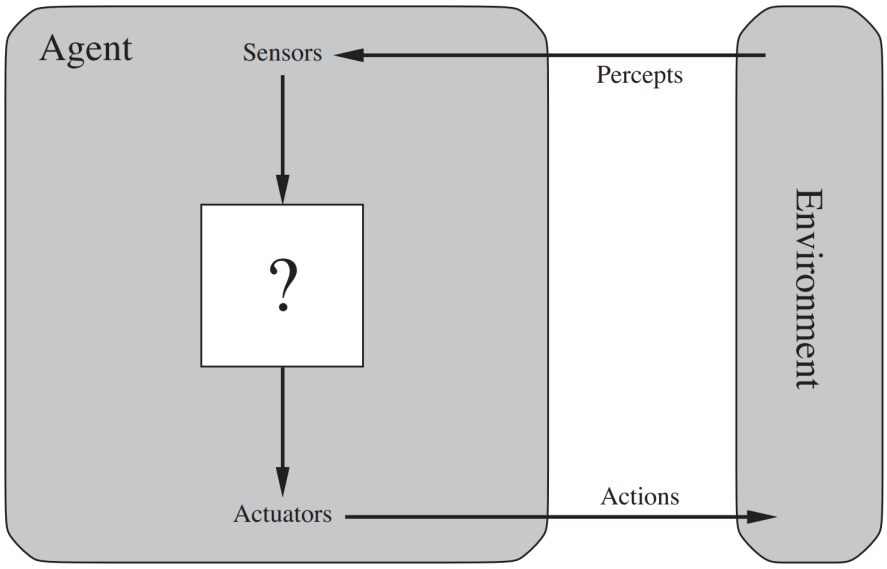
Para tratar sobre aprendizado de máquina, primeiramente, deve-se estabelecer o conceito de *agente* e como este é comparado com um mero programa de computador. Programas são instruções que devem ser executadas, sequencialmente, por um processador ou unidade de processamento. No escopo da inteligência artificial, *agente* é dito como o elemento que percebe o ambiente por sensores e pode agir neste através de atuadores, como descrito na Figura 7 (NORVIG; RUSSEL, 1970).

Neste trabalho, o agente pode ser considerado como não apenas o manipulador robótico ou mecanismo, mas como toda a arquitetura de sensores que definem o estado em que o manipulador se encontra e o controlador responsável pelo acionamento de seus atuadores, ou seja, o próprio robô. Também será considerado como pertencente ao agente o algoritmo responsável pela tomada de decisões em cada estado. Em outras palavras, se a tarefa do agente for alcançar, com seu "braço", algum ponto específico no ambiente (cinemática inversa), o próprio agente será responsável por ditar quais dos atuadores de seu braço devem ser acionados, e em qual sequência (estado), para que o objetivo seja alcançado. A sequência das ações tomadas será ditada pelos estados alcançados pelo agente, sendo que cada estado está associado à cada posição única tomada pelo agente (combinação de posições angulares das juntas).

Ao invés de desenvolver um programa que prescreva as ações tomadas em cada estado, de acordo com uma estratégia de tomadas de decisão pré-programada, neste trabalho será desenvolvido um agente que será responsável pela própria estratégia de tomada de decisões. O que será programado é, justamente, a capacidade de o agente definir sua própria estratégia de atuação no ambiente. O aprendizado de máquina é, por fim, a ciência que estuda a programação do próprio aperfeiçoamento de um agente, permitindo que este realize tomadas de decisão mais significantes para sua função ao decorrer do tempo (NORVIG; RUSSEL, 1970).

Aprendizado de máquina pode ser separado em alguns sub-campos, como aprendizado supervisionado, aprendizado não supervisionado e aprendizado por reforço, sendo este último utilizado como base para a implementação deste trabalho.

Figura 7 – Agentes, no escopo da inteligência artificial, podem observar e atuar sobre o ambiente através de sensores e atuadores.



Fonte: (NORVIG; RUSSEL, 1970).

Aprendizado supervisionado trata da formulação de uma função, pelo agente, que mapeia as entradas e saídas a partir de exemplos pré definidos. Pode-se utilizar, como exemplo, classificação de imagens: se o agente for treinado previamente por um banco de imagens, e cada imagem estiver corretamente rotulada com a informação de que é uma imagem representativa de um cachorro ou de um gato, por exemplo, o agente poderá discernir novas imagens de gatos e cachorros e rotulá-las corretamente (LAPAN, 2018).

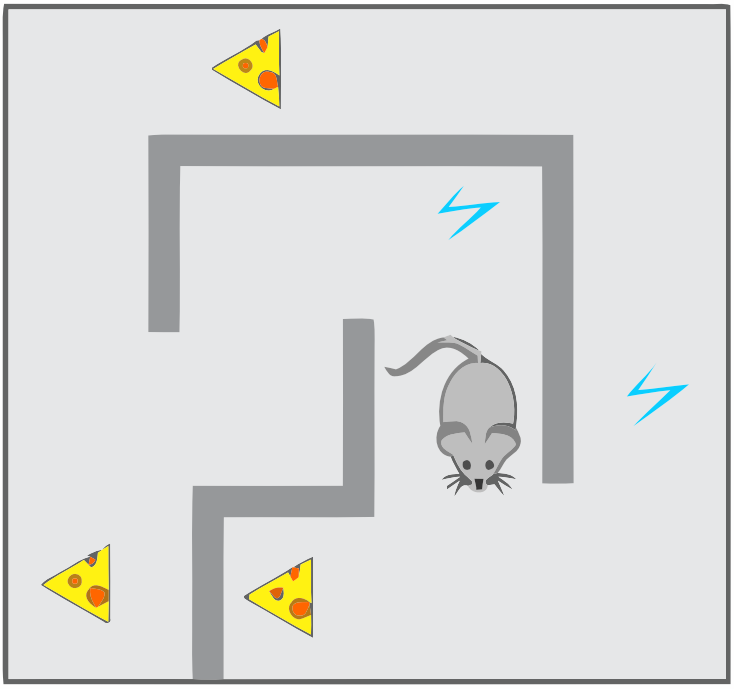
A principal diferença do aprendizado supervisionado para o aprendizado não supervisionado é que, neste último, não haverá um banco de informações para um aprendizado prévio. Na verdade, aprendizado não supervisionado também pode ser chamado de *clustering*, ou seja, agrupamento. O objetivo principal é o descobrimento e aprendizado de estruturas omitidas de relacionamentos em um acervo de dados e o agrupamento destes dados de acordo com a forma com que estes dados estão relacionados através destas estruturas (GENTLEMAN; CAREY, 2008).

2.2.1 Aprendizado por Reforço

Aprendizado por reforço pode ser considerado algo entre o aprendizado supervisionado e não supervisionado. Por exemplo, nota-se o cenário descrito na Figura 8: um rato, com fome, em um labirinto que possui comidas espalhadas e armadilhas de choques elétricos. Assim que o rato é posicionado dentro do labirinto, este não sabe nada sobre o cenário, logo não é possível relacionar o modelo diretamente com o aprendizado supervisionado, já que este exige um conhecimento prévio do modelo. Porém, conforme

o rato explora o cenário, este intuitivamente dará prioridade aos caminhos que o levarão à comida, e evitará os caminhos que resultem em choques elétricos. Este sistema de recompensa também impede de caracterizar o modelo de aprendizado como totalmente não supervisionado. Em outras palavras, o aprendizado por reforço é caracterizado pela falta de conhecimento prévio do modelo e pela existência de um sistema de recompensa que induz o agente a escolher "melhores"ações conforme o passar de certos períodos de tempo, denominados épocas de aprendizado (LAPAN, 2018).

Figura 8 – Exemplificação de aprendizado por reforço: cenário onde um rato, perdido em um labirinto, deve evitar choques elétricos e procurar comida.



Fonte: (LAPAN, 2018).

2.2.1.1 Processos de Decisão Markovianos

Aprendizado por reforço pode ser utilizado para resolver cenários enquadrados como *Markovian Decision Processes* (Processos de Decisão Markovianos). Para que um cenário seja classificado como um processo de decisão Markoviano, as seguintes condições devem ser satisfeitas:

- Deve ser possível separar o cenário em estados S, possuindo um estado inicial s_0. Para este trabalho, os estados serão as combinações das posições angulares do manipulador robótico;
- O agente deve possuir um conjunto específico de ações A possíveis de serem tomadas, e este grupo deve ser o mesmo para todos os estados do cenário

onde o agente está inserido. Em outras palavras, o agente deve poder realizar as mesmas ações independente de qual seja o atual estado do cenário. Para o agente robótico, as ações serão incrementos ou reduções de ângulos nas juntas atuadas. É o que levará o agente a outros estados. Um episódio pode ser considerado como um conjunto de ações que levam o agente de uma posição inicial s_0 até uma posição terminal, que indica o fim do episódio. Normalmente, a posição terminal é definida por estados que levam o episódio à falhas catastróficas ou ao estado final desejado. A sequência das ações tomadas em um episódio é denominado política de decisão, ou π. A política de decisão está diretamente relacionada com a estratégia de tomada de decisões do agente;

- O agente, no cenário, deve possuir um modelo de transição. Se s for o estado atual do cenário e a^s for a ação atual do agente no estado, o modelo de transição dirá com qual probabilidade um estado futuro s' será alcançado após a realização da ação: $P(s'|s,a)$. Esta noção de probabilidade é associada ao modelo transicional quando o cenário é estocástico, ou seja, uma ação tomada em um mesmo estado pode levar o agente a estados diferentes em função de uma certa probabilidade inerente ao cenário. O agente robótico conforme definido neste trabalho não possui probabilidades relacionadas ao seu modelo de transição, pois um acréscimo ou redução de ângulo em uma junta caracteriza um comportamento determinístico: a mesma ação tomada no mesmo estado atual sempre irá resultar no mesmo estado futuro. Logo, $P(s'|s,a)$ pode ser desconsiderado e tratado como uma constante $P(s'|s,a) = 1$. Porém, esta é uma característica do cenário proposto. Caso houvesse não-linearidades ou, por exemplo, obstáculos não determinados inseridos na trajetória do robô, o modelo de transição deve ser devidamente levantado. Considera-se o caso do modelo de transição determinista um caso especial de processo de decisão Markoviano;
- Cada estado $s \in S$ tem associado a si uma recompensa $R(s)$. Esta recompensa deve ser utilizada para verificar o quão desejado é cada estado para o objetivo do agente no cenário, de forma quantitativa. No caso do agente robótico, o estado desejado deve ser previamente definido. É a posição tridimensional desejada para o posicionamento do último elo do manipulador. A função de recompensa deve retornar valores inversamente proporcionais à distância entre a posição tomada pelo elo de referência do manipulador no estado e a posição tridimensional desejada.

A sequência de ações tomadas que levará o agente aos melhores estados é chamada de política ótima, ou π^*, e é chamada de política ótima. Entende-se, como melhores estados, os estados que maximizam a função de recompensa. A política ótima é a melhor estratégia de tomada de decisões possível de ser utilizada pelo agente no cenário. Considera-se o aprendizado bem sucedido quando a estratégia de tomadas de decisão do agente está

próxima ou é idêntica à estratégia ditada pela política ótima(NORVIG; RUSSEL, 1970).

Em conclusão, reconhece-se a cinemática inversa resolvida quando o agente, através do aprendizado de máquina, encontra a política ótima que leva o manipulador robótico ao estado que posiciona o elo de interesse sobre a posição tridimensional desejada.

2.2.1.2 Utilidade

Considerando que um episódio é uma sequência de estados visitados pelo agente, desde o estado inicial até o estado final, seguindo uma certa política de decisão π, a utilidade U é definida como acúmulo das recompensas $R(s)$ de acordo com os estados visitados pela dada utilidade, conforme mostra a Equação (9). O fator de desconto γ, número entre 0 e 1, descreve a preferência pelo agente aos estados mais próximos em relação aos futuros. Quanto mais próximo de 0 o valor de γ, menos será considerado os estados além dos próximos na utilidade. Caso $\gamma = 1$, a utilidade torna-se cumulativa, e não é interessante para cenários em que não haja estados terminais definidos pois poderá tender o valor da utilidade para o infinito. Se o agente partir de um estado inicial s_0, e há mais de uma política de decisão, é possível comparar, no estado, qual política terá a maior utilidade esperada, através da Equação (10). Como a política de decisão é o que define a tomada de uma decisão pelo agente em cada estado, pode-se considerar que, para cada ação possível em um estado, há uma ação a ser tomada por uma política diferente. A política ótima é a política que resultará na tomada de ação que retornará a máxima utilidade esperada, conforme descrito na Equação (11) (NORVIG; RUSSEL, 1970).

$$U^{\pi} = R\left(s_0\right) + \gamma R\left(s_1\right) + \gamma^2 R\left(s_2\right) + \cdots \tag{9}$$

$$U^{\pi}(s) = \left[\sum_{t=0}^{\infty} \gamma^t R\left(S_t\right)\right] \tag{10}$$

$$\pi^*(s) = \underset{a \in A(s)}{\operatorname{argmax}} \sum_{s'} P\left(s' \mid s, a\right) U\left(s'\right). \tag{11}$$

Em cenários determinísticos, como o cenário do manipulador robótico, a Equação (11) pode ser simplificada para a Equação (12).

$$\pi^*(s) = \underset{a \in A(s)}{\operatorname{argmax}} \sum_{s'} U\left(s'\right). \tag{12}$$

2.2.1.3 Equação de Bellman

A diferença entre a recompensa imediata de um estado $R(s)$ e a utilidade U^{π} é que esta última traz o conceito de recompensas acumuladas ao longo dos estados visitados pelo agente, quando este segue a estratégia de tomada de decisões ditada pela política π. Ou seja, a utilidade está diretamente relacionada à uma sequência de estados visitados de

forma sequencial. Esta relação é melhor definida pela equação de Bellman (Equação (13)). Se os valores de utilidade forem armazenados, a cada vez que um estado é visitado pelo agente este valor deve ser atualizado seguindo a Equação (14).

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P\left(s' \mid s, a\right) U\left(s'\right) \tag{13}$$

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P\left(s' \mid s, a\right) U_i\left(s'\right) \tag{14}$$

A Equação (14), que rege a atualização dos valores de utilidade de Bellman, pode ser modificada para que a utilidade do estado atual seja atualizada em função da diferença entre esta e a utilidade estimada do estado futuro regulada pela constante α, nominada de fator de aprendizado (Equação (15)). Escolher o valor de α entre 0 e 1 evita que os valores de utilidade sejam atualizados abruptamente quando transições raras acontecem, o que pode ocorrer em cenários estocásticos. Quanto mais próximo α for de 0, menos impactante serão as atualizações. A utilização da equação de Bellman neste formato é uma técnica chamada de *temporal-difference approach*, ou *TD-Learning*, e é interessante pois dispensa o conhecimento do modelo de transição $P\left(s' \mid s, a\right)$ (NORVIG; RUSSEL, 1970).

$$U^{\pi}(s) \leftarrow U^{\pi}(s) + \alpha\left(R(s) + \gamma U^{\pi}\left(s'\right) - U^{\pi}(s)\right) \tag{15}$$

Nota-se que a equação de Bellman e suas variantes sempre utilizam a informação da utilidade do estado atual s e o estado futuro s', relacionados pela política π. Caso, no episódio de aprendizado, o agente esteja visitando s' pela primeira vez, a utilidade neste estado será apenas a recompensa imediata $R(s\prime)$. É definida, então, a seguinte lógica:

$$U^{\pi}(s') \leftarrow \begin{cases} R(s') & \text{caso } s' \text{ não tenha sido visitado pelo agente} \\ U^{\pi}(s') + \alpha\left(R(s') + \gamma U^{\pi}\left(s''\right) - U^{\pi}(s')\right) & \text{caso contrário} \end{cases} \tag{16}$$

2.2.1.4 Q-Learning

O algoritmo *Q-Learning* utiliza a equação de Bellman no formato *TD-Learning* e o conceito de episódios de treinamento para criar um aprendizado que converge para a política ótima do cenário. Uma das características principais desta técnica é o formato tabular, conhecido como *Q-Table*. para formar a *Q-Table*, inicialmente, cria-se uma tabela Q de elementos $Q(s, a)$. Esta tabela possui número de linhas igual ao número de ações possíveis de serem tomadas pelo agente, enquanto que o número de colunas é igual ao número de estados possíveis de serem visitados no cenário pelo agente. Os elementos $Q(s, a)$ representam a utilidade de executar a ação a no estado s. Dado um estado s, $\max_a Q\left(s, a\right)$ retorna a utilidade que possui o maior valor dentre as utilidades de todas as ações possíveis de serem tomadas no estado em questão. A atualização das utilidades da *Q-Table* é dada pela Equação (17) (LAPAN, 2018).

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(R(s) + \gamma \max_{a'} Q\left(s', a'\right) - Q(s,a) \right) \tag{17}$$

Para o cenário da solução de cinemática inversa de manipuladores robóticos, o número de colunas da *Q-Table* é dado pelo número de combinações possíveis das posições angulares dos manipuladores. O número de linhas será equivalente ao número de ações diferentes que o manipulador poderá tomar em um determinado estado. Como exemplo, pode-se utilizar duas ações distintas por atuador no manipulador, que podem representar acréscimo e redução das unidades de posições angulares.

A escolha da ação, ou seja, a política utilizada durante a execução de um episódio deve estar bem definida. Estabelecendo um episódio de treino como a atuação do agente aprendiz no cenário a partir de um estado inicial até encontrar um estado terminal seguindo uma determinada política de decisão, nos primeiros episódios a tabela estará majoritariamente vazia. Para que a política ótima seja definida, a tabela deve estar preenchida com, no mínimo, as combinações de $Q(s,a)$ dos estados que devem ser visitados pelo agente entre o estado inicial e o estado que fornecerá a máxima recompensa imediada $R(s)$.

2.2.1.5 Política de Decisão: Exploração ou Explotação

Em um episódio de treino, o agente inicia localizando-se em um estado qualquer e deve tomar uma ação. O agente, através da *Q-Table*, terá estimativas de utilidade para cada uma das ações possíveis de serem tomadas no estado em questão. Caso a *Q-Table* esteja devidamente preenchida, isto é, possua uma relação de ações-utilidades por estado que sejam idênticas ao cenário real, basta escolher a ação que retorna o maior valor de utilidade possível. Em outras palavras, a política ótima será a estratégia de escolha de ações que sempre retorna o maior valor de utilidade e, consequentemente, o agente acabará sendo deslocado do estado qualquer citado ao estado desejado. Quando o agente escolhe uma ação de acordo com o maior valor de utilidade, denomina-se como uma estratégia de explotação.

Porém, no começo do aprendizado, enquanto a *Q-Table* não estiver devidamente preenchida e com seus valores suficientemente atualizados, não garante-se que os valores das utilidades relacionadas às possíveis ações sejam representativas do cenário real. É por este motivo que o agente deve explorar o cenário, e para isso, a Q-Table deve ser atualizada com os valores reais do cenário, mesmo que estes não possuam os máximos valores de utilidade. Quando o agente executa uma ação independente do valor de utilidade, com o intuito de preencher e atualizar a *Q-Table*, define-se esta ação como pertencente à uma estratégia de exploração. Uma forma de otimizar o processo de aprendizado é garantir que, nas primeiras épocas de aprendizado, o agente tenha preferência por ações de exploração e gradativa- mente mude a estratégia para escolher ações de explotação (ZAI; BROWN, 2020).

2.2.2 Redes Neurais

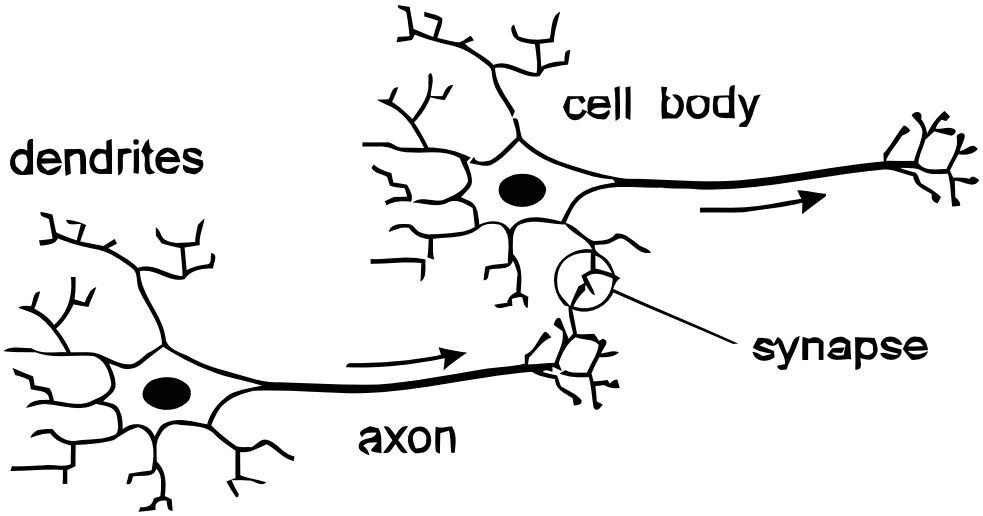
O algoritmo *Q-Learning* clássico, também conhecido como *Q-Learning* tabular, têm um custo computacional para ser executado que é diretamente proporcional à complexidade do cenário. Quanto maior o número de estados possíveis de serem visitados pelo agente, maior será este custo. Da mesma forma, quanto maior o número de ações possíveis de serem tomadas pelo agente, maior será o número destes estados. O motivo deste custo ser atrelado à complexidade do cenário é que, para que o agente consiga definir a política ótima, este deve possuir conhecimento prévio de, no mínimo, todos os estados que o devem ser visitados para deslocar-se entre o estado inicial e o estado desejado. Além disso, as utilidades para cada par de ações-estados nestes estados intermediários devem ser atualizadas suficientemente para que seus valores convirjam. Já que a *Q-Table* é inicializada vazia, para que a definição da política ótima seja possível, o agente deve utilizar políticas de decisão exploratórias até que este conheça suficientemente os conjuntos de ações-estados possíveis de serem tomados pelo agente no cenário, ou seja, uma grande parcela da *Q-Table* deve ser preenchida e devidamente armazenada em memória para ser utilizada como parâmetro de decisão da política. Caso o agente não explore o cenário suficientemente, a estratégia de tomadas de decisão não convergirá para a política ótima. Para garantir a característica exploratória, a política de decisão deve tender à escolher ações que levam o agente a estados ainda não visitados.

Se for considerado um manipulador robótico de apenas duas juntas e de precisão simplificada, permitindo somente números inteiros nas posições das juntas, e que cada junta possa tomar 360°posições diferentes, combinado com quatro ações possíveis por estado (uma ação de acréscimo e uma ação de redução de graus em cada uma das duas juntas) implica em 518.000 combinações possíveis de pares ação-estado. Caso houvessem três juntas de revolução, o número de possíveis combinações seria 279.936.000, e 134.369.280.000 para quatro juntas de revolução. A manipulação e o processamento de uma quantidade tão massiva de dados inviabiliza a utilização do algoritmo clássico de *Q-Learning* tabular. Nota-se que, em casos reais, a precisão de posições angulares pode tomar alguns decimais após a vírgula, aumentando ainda mais o número de pares ação-estado. Para viabilizar estes casos, uma das possíveis soluções pode ser encontrada na utilização de redes neurais.

Redes neurais são conhecidas como algoritmos computacionais que aprendem soluções de problemas através de exemplos, e a inspiração para a arquitetura do algoritmo vêm de estudos de processamentos biológicos no sistema nervoso humano. Na Figura 9, nota-se que os neurônios são formados por dendritos, que agem como entradas, disparando pulsos elétricos que percorrem pelo áxion até a próximo neurônio. A região de ligação entre um neurônio e outro é chamado de sinapse (BISHOP, 1994).

As redes neurais são compostas por entradas, saídas e, semelhante aos neurônios, possuem vários nós que são interligados por somatórios e multiplicação de pesos. Para dada combinação de valores das entradas, a saída dependerá do arranjo dos nós e dos

Figura 9 – Exemplo de processamento biológico no sistema nervoso.



Fonte: (BISHOP, 1994).

valores dos pesos. Com valores pré determinados de saídas e entradas, a Rede Neural pode ser treinada, isto é, ter seus pesos manipulados até que as relações entre entradas e saídas sejam satisfeitas para todas as combinações desejadas. A manipulação dos pesos para atender as combinações pré determinadas de entradas e saídas, ou pelo menos, gerar saídas com valores próximos dos esperados pelas combinações, significa treinar a Rede Neural. Nota-se, na Figura 10, a composição de um nó com duas entradas X_1 e X_d. O sinal a é composto por $X_1 * W_1 + X_d * W_d$. A saída z é dado por $g(a)$, onde g é uma função de ativação. A função de ativação é importante para agregar as características não lineares da rede. Sem ela, o treinamento da rede seria o mesmo que uma regressão linear: é graças às propriedades não lineares que as redes neurais são capazes de serem treinadas para relacionar conjuntos de entradas e saídas complexas (BISHOP, 1994).

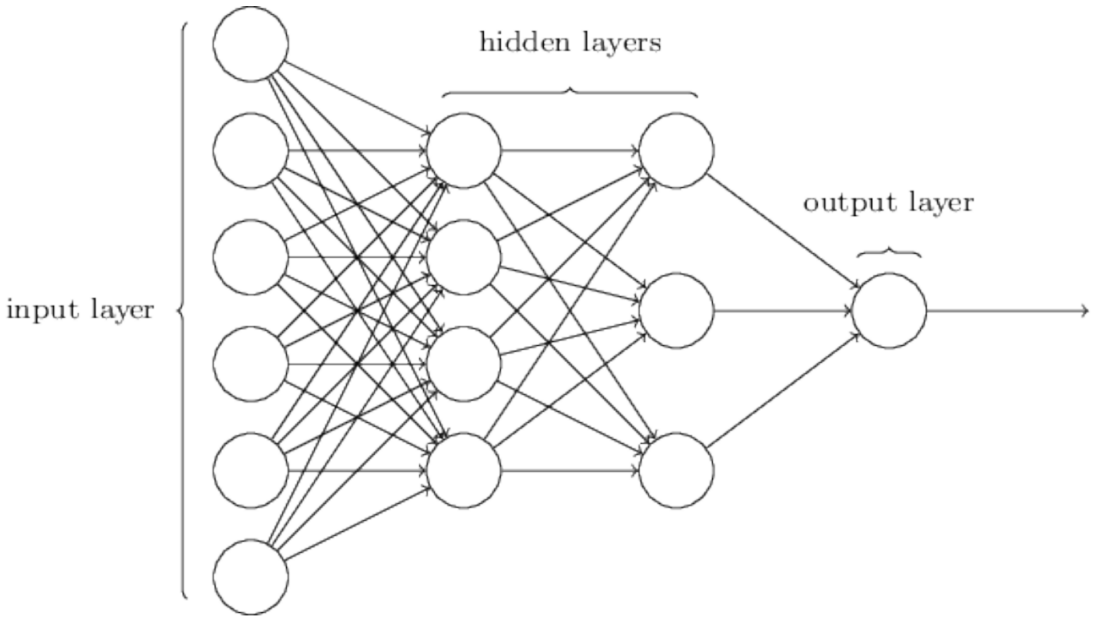
Conforme a Figura 11, algumas típicas funções não lineares são:

- (a): função sinal, ou degrau;
- (b): função ReLU, ou *rectified linear unit* (unidade linear retificada);
- (c): função sigmoide.

2.2.2.1 Deep Q-Learning

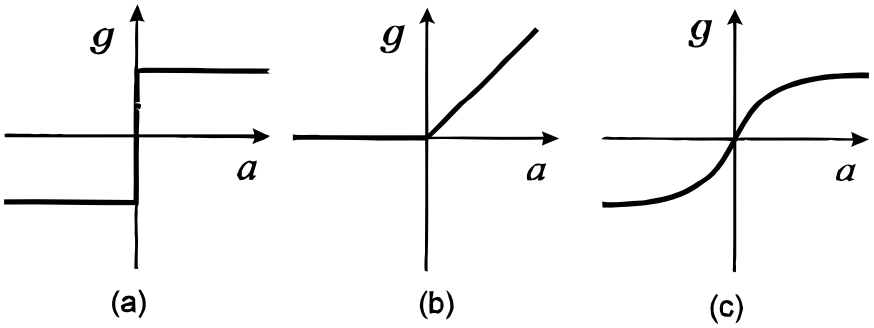
Deep Q-Learning é um algoritmo de aprendizado por reforço muito parecido com o clássico *Q-Learning* tabular. A diferença é que este utiliza uma arquitetura de rede neural para dar suporte à *Q-Table*. A ideia é exatamente a mesma, compor um agente

Figura 10 – Exemplo de arquitetura de rede neural.



Fonte: (NIELSEN, 2015).

Figura 11 – Exemplos de funções de ativação para redes neurais.

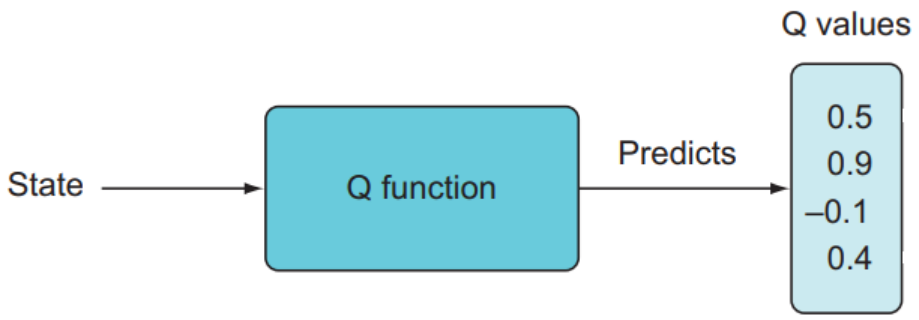


Fonte: (BISHOP, 1994).

que aprenda combinações de estado-ação suficientes do cenário para que a política ótima possa ser extraída, porém, ao invés de preencher as combinações em uma tabela que irá conter todos os estados como colunas e ações possíveis como linhas, iremos treinar uma rede neural com as amostras de utilidades $U(s, a)$ colhidas pelo agente em episódios de treinamento, chamadas de épocas de aprendizado. A rede neural será arquitetada para receber estados em suas entradas e as utilidades por cada ação possível de ser tomada em suas saídas (ZAI; BROWN, 2020).

Conforme mostra na Figura 12, para o caso do cenário da cinemática inversa do manipulador robótico, as entradas serão os estados, ou seja, a combinação das posições angulares dos atuadores. As saídas serão as estimativas de utilidades esperadas para cada ação tomada no estado utilizado na entrada. Caso o manipulador possua duas juntas rotacionais, por exemplo, e for utilizada quatro ações possíveis de serem tomadas (acréscimo e decréscimo de ângulos em cada junta), cada uma das quatro saídas representará a utilidade recebida caso a ação relacionada seja tomada pelo agente.

Figura 12 – Exemplo da representação da *Q-Table* como rede neural.



Fonte: (ZAI; BROWN, 2020).

Para treinar a rede neural, ainda será utilizado o conceito da *Q-Table* tabular, porém apenas de forma temporária. A cada época de aprendizado, o agente irá preencher uma *Q-Table* tabular, conforme ocorre no *Q-Learning* clássico. Ao fim da época, as informações colhidas na tabela serão utilizadas para treinar a rede neural, e no próximo episódio, a tabela pode ser limpa, reduzindo consideravelmente a carga computacional exigida pelo processo de aprendizado. Esta tabela, por ser apenas secundária, utilizada para reservar as informações colhidas pelo agente na época de aprendizado de forma temporária, pode ser chamada de *buffer* (ZAI; BROWN, 2020).

3 METODOLOGIA

Neste capítulo, é descrito a implementação do algoritmo para a solução da cinemática inversa de manipuladores robóticos, assim como todas as estruturas da implementação. O algoritmo será apresentado de forma genérica, sem especificação de linguagem de programação ou de configuração do manipulador robótico. Porém, o capítulo irá contar com exemplos de implementação em Matlab de um manipulador robótico cuja a configuração do mecanismo contemple duas juntas de rotação e dois elos de ligação. O capítulo é separado em quatro seções: uma visão geral e estrutural algoritmo (Seção 3.1), demonstração de funções específicas que dão suporte ao algoritmo (Seção 3.2), a inicialização do algoritmo (Seção 3.3) e, por fim, o processo de aprendizado (Seção 3.4).

3.1 VISÃO GERAL DO ALGORITMO

Uma estrutura básica do algoritmo é exemplificada no Pseudocódigo (1). O algoritmo é, basicamente, separado em duas partes estruturais: a inicialização e o laço de aprendizado:

- A inicialização é o trecho do algoritmo em que define-se os parâmetros que serão utilizados ao decorrer do aprendizado;
- O laço de aprendizado é um *loop* que têm, como objetivo principal, garantir que o agente conheça o cenário em que este se encontra e aprenda sobre ele, levantando a quantidade suficiente de informações sobre ele para que o agente consiga definir a melhor estratégia de atuação. Ao fim de cada ciclo de aprendizado, os resultados do aprendizado devem ser analisados para verificar se a cinemática inversa foi solucionada. Se sim, encerra-se o laço de aprendizado e, consequentemente, a execução do algoritmo.

```
Inicialização();
while true do
    Aprendizado();
    if Resultados() then
        break;
    end
end
```

Pseudocódigo 1: Estrutura do algoritmo implementado.

O ambiente de desenvolvimento escolhido é o Matlab, plataforma de programação muito utilizada por engenheiros e cientistas de dados. Os motivos que levaram à escolha do Matlab como plataforma de desenvolvimento foram o suporte à redes neurais e a facilidade para modelar sistemas algébricos.

3.2 FUNÇÕES DE SUPORTE

As funções apresentadas nesta seção são chamadas pelo algoritmo uma ou mais vezes durante sua execução. Separar algumas funcionalidades específicas do algoritmo em funções de suporte é considerada uma boa prática pois ajuda a reduzir linhas de código duplicadas, além de contribuir com a organização geral do código fonte.

3.2.1 *cinematica_direta()*

Esta função é responsável pela cinemática direta do manipulador robótico. A implementação recria os passos especificados para obtenção da cinemática direta pelo método de Denabit-Hartenberg, especificado na Seção 2.1.1. A função recebe, como parâmetro, um vetor de n elementos (para $n > 0$) referentes às posições angulares das juntas, e retorna dois vetores, com três números reais cada, notados genericamente neste documento como: $[\alpha, \beta, \gamma]$ e $[X, Y, Z]$. Dos vetores retornados pela função, o primeiro contém os ângulos de Euler da rotação do elo de interesse (Seção 2.1.2), enquanto o segundo vetor demonstra a posição do elo de interesse no espaço tridimensional.

Por exemplo, se as posições angulares das juntas rotacionais do manipulador forem 45° e 90°, e *cinematica_direta*$([45, 90])$ = ([10, 90, 20], [20, 40, 30]), significa que a junta de interesse rotacionou 10° ao redor do eixo X, 90° ao redor do eixo Y e 20° ao redor do eixo Z, e está localizada na posição tridimensional ($X = 20u.d.$, $Y = 40u.d.$ e $Z = 30u.d.$), sendo *u.d.* uma abreviação para *unidades de distância*.

Um exemplo de implementação em Matlab encontra-se no Apêndice A.1

3.2.2 *interpretador_acao()*

Esta função traduz uma ação qualquer tomada pelo agente em uma determinado estado, retornando o estado em que o agente será posicionado caso a ação seja tomada. A saída da rede neural do *Deep Q-Learning*, como mostra a Figura 12, retorna um vetor de utilidades que possui o número de linhas igual ao número de ações possíveis de serem escolhidas pelo agente. Logo, deve-se associar cada ação a um número, que é associada a esta linha. É nessa função que o número e a ação são associados: a função recebe, como parâmetro, um estado e um número inteiro que representa uma ação, e retorna outro estado. O estado fornecido como parâmetro é o estado atual do agente, e o estado retornado pela função é o estado que o agente irá deslocar-se após a atuação referente.

A função recebe como parâmetro um vetor de n elementos (para $n > 0$) referente às posições angulares das juntas e um número inteiro, referente à ação tomada. A função retorna um vetor também de n elementos, sendo que o vetor entregue como parâmetro é associado ao estado atual, e o vetor de retorno da função é associado ao estado que será visitado, em sequência, pelo agente após a tomada de decisão.

Por exemplo, se a lógica da Tabela 1 for implementada no algoritmo, então *interpretador_acao([30, 40], 1)* = [31, 40]. Neste exemplo, caso o estado atual do agente seja [30, 40], e a ação 1 ("Somar posição angular na primeira junta.") for escolhida, o próximo estado do agente será o estado [31, 40]. Em outras palavras, caso as posições angulares dos atuadores do manipulador sejam 30° e 40°, e a ação de número 1 referente à Tabela 1 for escolhida pelo agente, o atuador da primeira junta do manipulador irá incrementar-se em 1°.

Tabela 1 – Exemplo de lógica utilizada pela função que interpreta as ações tomadas pelo agente.

Ação Tomada	**Consequência da Ação Tomada no Estado do Agente**
0	Somar posição angular em ambas juntas.
1	Somar posição angular na primeira junta.
2	Somar posição angular na segunda junta.
3	Somar posição angular na primeira junta e subtrair na segunda junta.
4	Subtrair posição angular na primeira junta e somar na segunda junta.
5	Subtrair posição angular na primeira junta.
6	Subtrair posição angular na segunda junta.
7	Subtrair posição angular em ambas as juntas.

Fonte: Autor.

É importante, nesta função, implementar os limites que garantem que as posições angulares estejam sempre entre -180° e +180°. Este limite é coerente com a modelagem do cenário e evita posições redundantes das juntas rotacionais do manipulador.

Um exemplo da implementação em Matlab encontra-se no Apêndice A.2.

3.2.3 *R()*

Uma das necessidades para implementação do *Q-Learning* é a formulação de uma função de recompensa, que irá ser utilizada pelo algoritmo como um parâmetro quantitativo para avaliar a utilidade dos estados em que o agente se encontra.

Pode-se comparar dois espaços tridimensionais pelo valor MSE (*Mean Squared Error*), conforme a Equação (18). A equação MSE é interessante pois sempre irá retornar um valor real positivo diretamente proporcional à distância entre os pontos comparados. Porém, para preservar o sentido de **recompensa**, basta inverter a equação, como exemplificado na Equação (19). Soma-se o denominador em uma unidade para que, caso $MSE = 0$, a recompensa não tenda ao infinito, limitando seu valor entre 0 e 1.

$$Erro = \sqrt{(X_{atual} - X_{desejada})^2 + (Y_{atual} - Y_{desejada})^2 + (Z_{atual} - Z_{desejada})^2} \quad (18)$$

$$R = \frac{1}{Erro + 1} \quad (19)$$

A função recebe, como parâmetro, uma posição tridimensional qualquer e a posição desejada, esta última sendo um dos parâmetros de inicialização do algoritmo. A função irá comparar a posição fornecida com a desejada, e irá retornar um valor proporcional ao inverso da distância entre as duas. Ou seja, se a posição tridimensional for a posição atual do elo de interesse do manipulador, quanto mais perto o elo estiver da posição tridimensional desejada, maior será a recompensa. Se a posição tridimensional fornecida for $(X_{atual}, Y_{atual}, Z_{atual})$, e a posição desejada for $(X_{desejada}, Y_{desejada}, Z_{desejada})$, a função deve implementar a Equação (19).

Um exemplo da implementação em Matlab encontra-se no Apêndice A.3.

3.2.4 *erro_com_penalidade()*

Esta função estende a função R(). Com um parâmetro a mais referente ao estado passado, a função verifica se, em relação ao estado desejado, o agente está visitando um estado que se afasta ou se aproxima do estado desejado. Se o agente está aproximando-se do estado desejado, a função retorna o valor de R(). Caso contrário, a função irá retornar um valor de recompensa negativo, como, por exemplo, -1. Desta forma, é possível penalizar o agente caso a estratégia de atuação no cenário esteja afastando o agente do estado desejado. A função comporta-se como Equação (20).

$$Retorna \longrightarrow \begin{cases} R() & \text{caso o agente esteja aproximando-se da posição desejada} \\ -1 & \text{caso contrário} \end{cases} \tag{20}$$

Um exemplo da implementação em Matlab encontra-se no Apêndice A.4.

3.3 INICIALIZAÇÃO

Esta seção descreve o segmento do algoritmo responsável por inicializar os parâmetros utilizados no laço de aprendizado, assim como a arquitetura da rede neural.

3.3.1 Arquitetura da Rede Neural

Nesta etapa, deve-se definir, em nível de código, a estrutura da rede neural. Caso o programador implemente a rede neural sem o auxílio de alguma biblioteca especializada ou recurso externo, deve-se definir, além do número de nós e camadas, as entradas, saídas e a metodologia de ajuste dos pesos. Durante o laço de aprendizado (Seção 3.4), a rede será alimentada com novas combinações de entradas e saídas armazenadas no *buffer*, ocorrendo de forma cíclica a cada época de aprendizado.

Para criar a arquitetura da rede neural no ambiente Matlab, utiliza-se a função *fitnet()*, que recebe como parâmetro um vetor com valores inteiros. Este vetor é utilizado para definição da arquitetura da rede, sendo que o tamanho do vetor será igual ao número

de camadas e os valores irão definir o número de nós da rede neural, de acordo com a sua posição no vetor. Por exemplo, para construir uma rede neural com três camadas, sendo que a primeira contenha 30 nós e as duas últimas, 15 nós, basta fornecer à função o seguinte parâmetro: [30 15 15]. A função retorna um objeto que representa a rede neural. Em sequência, define-se a função de ativação do tipo *poslin*, que é uma implementação em Matlab da função $ReLU$, representada pelo gráfico (b) na Figura 11. Conforme novas combinações de entradas e saídas sejam encontradas, a função *train()* tem como objetivo atualizar os pesos da rede para que esta reconheça os padrões das novas combinações de entradas e saídas. Como parâmetros, a função recebe o objeto da rede neural, uma matriz de entradas e uma de saídas. O número de linhas da matriz de entrada deve ser igual ao número de entradas da estrutura da rede neural, assim como para as saídas. A ligação entre as entradas e as saídas dos vetores se dá pelos índices das colunas, logo, os índices das colunas devem ser os mesmos.

Como exemplo em Matlab, no Apêndice B.1, cria-se um objeto do tipo rede neural com duas camadas e trinta nós em cada, e atribui-se a função de ativação $ReLU$. As tratativas de treinamento de redes neurais do Matlab utilizam uma parcela dos dados fornecidos para validações e testes, além dos dados que são utilizados para corrigir os parâmetros da rede. As funcionalidades de validação e testes estarão implementadas implicitamente no algoritmo, e podem ser descartadas conforme o Apêndice B.2.

3.3.2 Parâmetros

Deve-se definir alguns parâmetros que serão utilizados no laço de aprendizado:

- *pos_desejada*: Vetor que representa uma posição no espaço tridimensional. A cinemática inversa considera-se resolvida para dada posição quando o algoritmo for capaz de definir as posições angulares de cada junta do manipulador robótico para que o elo de interesse seja posicionado sobre este ponto no espaço;
- *gamma*: fator de desconto da equação de Bellman;
- *alpha*: fator de aprendizado da equação de Bellman na configuração da diferença temporal;
- *tamanho_buffer*: este parâmetro é utilizado para definir o tamanho do buffer de armazenamento de utilidades. No algoritmo, o buffer será utilizado como um armazenador temporário das informações de utilidade por estado colhidas pelo agente, e será referência de entradas e saídas da rede neural para o ajuste de parâmetros desta. O buffer será uma referência à uma Q-Table, e quando um estado que ainda não foi descoberto for visitado pelo agente, este será introduzido no buffer como uma nova coluna. Quando o número de colunas atingir o tamanho do buffer, o algoritmo irá começar a sobre-escrever os valores mais antigos (método *FIFO*, ou *First In, First Out*);

- *tamanho_episodio*: definição do número de ações executadas pelo agente em um episódio de treino. Como o cenário não considera nenhum estado específico como um estado terminal, este parâmetro é importante para definir um final ao episódio de treino. Quando o agente, partindo do estado inicial, toma um número de ações igual ao valor definido neste parâmetro, considera-se o episódio de treino encerrado, e o agente é re-posicionado para sua orientação inicial. A cada ação tomada em um estado específico, o agente armazena as informações de utilidade no buffer;
- *tamanho_epoca*: este parâmetro define o número de ações tomadas pelo agente em uma época de aprendizado. Uma época de aprendizado é considerada encerrada quando o agente toma um número de ações igual ao valor definido neste parâmetro, independentemente dos ciclos de episódios de treino. Ao fim de uma época de aprendizado, os pesos da rede neural serão atualizados utilizando as informações do cenário colhidas durante os episódios de treino e presentes no *buffer*.

Um exemplo da implementação dos parâmetros em Matlab encontra-se no Apêndice B.3.

3.3.3 Inicialização do Buffer e da Rede Neural

Após a definição da arquitetura da rede neural e dos parâmetros, a rede neural deve ser minimamente treinada para que o objeto da rede tenha suas entradas e saídas definidas. Desta forma, o agente torna-se capacitado para atuar no cenário no primeiro episódio de treino, mesmo que de forma inicialmente desorientada. Para treinar a rede neural, inicializa-se as variáveis do buffer com valores de utilidade reais de ações em alguns estados, preferencialmente o estado inicial do agente e os estados vizinhos, e em seguida atualiza-se os pesos da rede neural com os conjuntos inseridos no *buffer*. O funcionamento do *buffer* é explicado na Seção 3.4.1.

Como exemplo em Matlab, no Apêndice B.4, a função *inicializar_input_target()* é encarregada pelo comportamento de inicialização do buffer. Para o exemplo, preenche-se as matrizes *buffer_input* e *buffer_target* com os conjuntos de utilidade de oito ações possíveis de serem tomadas pelo agente nos estados $[0;0]$, $[0;1]$, $[1;0]$ e $[1;1]$. Utiliza-se, então, a função *train()* recebendo como parâmetro o objeto da rede neural, a matriz de entradas e de saídas do buffer (*buffer_input* e *buffer_target*, respectivamente), como exemplificado no Apêndice B.5.

3.4 LAÇO DE APRENDIZADO

Os trechos de código mostrados até então pertencem à etapa de na inicialização do algoritmo, e serão processados apenas uma vez. Após a atribuição dos parâmetros, a

construção do objeto da rede neural e a inicialização desta, a função que trata o treinamento do agente deve ser invocada e executada em *loop*.

O *loop* de aprendizado pode ser segmentado em duas etapas distintas: o episódio de treino e a época de aprendizado. Conforme explicado na seção Seção 3.3.2, o episódio de treino é definido como a exploração do agente no cenário, permitindo que este atue sobre e preencha o *buffer* com as informações de utilidade colhidas até que o número de ações definido pelo parâmetro *tamanho_buffer* seja alcançado. Ao fim do episódio de treino, o agente tem seu estado atual atualizado para o estado inicial. A época de aprendizado também possui seu período definido por um número de ações tomados pelo agente, limitado pelo parâmetro *tamanho_epoca*. Quando este número de ações é realizado pelo agente, é executada uma atualização dos pesos da rede neural utilizando as informações presentes no *buffer*.

Os episódios de treino e as épocas de aprendizado são independentes, ou seja, podem ocorrer várias épocas de aprendizado durante o período de um episódio de treino, ou vice versa: o período das épocas de aprendizado e dos episódios de treino são determinados apenas pelos parâmetros de inicialização do algoritmo.

3.4.1 Buffer

O buffer é composto por duas matrizes, *buffer_input*, *buffer_target* e uma variável de valor inteiro *buffer_indice*:

- *buffer_input* : Cada combinação de linhas em uma coluna da matriz *buffer_input* representa um estado do cenário.
- buffer_target : Cada combinação de linhas em uma coluna da matriz buffer_target representa as utilidades para cada uma das ações possíveis de serem atuadas pelo agente em cada um dos estados da matriz *buffer_input*. O número de colunas das duas matrizes devem ser sempre idênticos, e os índices das colunas também devem ser coerentes: os valores de utilidade das ações da matriz *buffer_target* estão diretamente relacionadas ao estado da matriz *buffer_input* no mesmo índice de colunas. A combinação das duas matrizes é utilizada como conjuntos de entradas e saídas (Figura 12).
- *buffer_indice* : O índice é utilizado para garantir que o tamanho de colunas das matrizes não seja maior do que o configurado pelo usuário na etapa de inicialização (parâmetro *tamanho_buffer*, exemplificado na Seção 3.3.2). Este parâmetro deve ser atualizado sempre que um conjunto novo de entradas e saídas for adicionado às matrizes do *buffer*.

Ao fim de uma época de aprendizado, o agente irá treinar a rede neural com as informações de entradas e saídas colhidas e armazenadas no *buffer*. No episódio de treino, a cada ação tomada pelo agente, o valor de utilidade é calculado e armazenado no *buffer*. Um

exemplo de implementação da função responsável por este comportamento, em Matlab, é exemplificado no Apêndice C.1.

3.4.2 Exploração vs Exploitação: O Algoritmo *Epsilon Greedy*

Conforme explicado na Seção 2.2.1.5, para que a rede neural forneça um modelo fiel das utilidades relacionadas às possíveis ações no cenário real, esta deve ser suficientemente treinada, isto é, ser alimentada suficientemente por combinações de entradas e saídas coletadas no cenário pelo agente. Uma forma de otimizar o processo de aprendizado é garantir que, nas primeiras épocas de aprendizado, o agente tenha preferência por ações de exploração e gradativamente mude a estratégia para escolher ações de exploitação. Este comportamento pode ser definido pela estratégia *epsilon-greedy*. A estratégia consiste em inicializar o algoritmo com uma variável $\epsilon = 0$, e este valor deve ser atualizado gradativemente, sempre ao fim de cada época de aprendizado, até atingir $\epsilon = 1$. Quando $\epsilon = 0$, o agente irá sempre escolher ações de exploração, e quando $\epsilon = 1$, o agente irá sempre optar por ações de exploitação. Quando $0 < \epsilon < 1$, o agente terá uma probabilidade de $(1 - \epsilon)$ em escolher ações de exploração (ZAI; BROWN, 2020).

O comportamento da estratégia *epsilon-greedy* pode ser transcrito em uma função que recebe, como parâmetros, a rede neural, o estado atual do agente e o número de épocas de aprendizado já executadas. O valor de ϵ deve estar em função das épocas de aprendizado, e é utilizado para calcular se o agente deve escolher uma ação de exploração ou de exploitação. Caso seja de exploração, o agente irá escolher, aleatoriamente, uma das ações possíveis de serem tomadas no estado em que se encontra. Caso a ação tomada seja de exploitação, deve-se recuperar os valores de utilidade estimadas por ação no estado e a ação que está associada ao maior valor de utilidade deve ser escolhida.

Um exemplo da implementação da função *epsilon-greedy* em Matlab encontra-se no Apêndice C.2.

3.4.3 Avaliação do Aprendizado

Ao fim de cada episódio de treino, é realizada uma varredura em todos os estados que foram percorridos pelo agente. Já que cada estado é uma combinação de posições angulares das juntas do manipulador robótico, cada um dos estados é analisado pela cinemática direta para obtenção das posições pelo elo de interesse no espaço tridimensional. Estas posições são comparadas com a posição desejada pelo *root mean square* da diferença das distâncias em relação aos eixos X, Y e Z. (Equação (18)). Caso, em alguma das posições comparadas, o erro quadrático for um valor igual ou que tende à zero, o estado relacionado são as posições angulares do manipulador que posiciona o elo de interesse sobre a posição desejada, e pode-se considerar a cinemática inversa solucionada.

3.4.4 Estrutura do Laço de Aprendizado

Definido-se:

- Os valores dos parâmetros *tamanho_episodio* e *tamanho_epoca* como o número de ações tomadas pelo agente que limitam, respectivamente, um episódio de treino e uma época de aprendizado;
- O estado inicial do agente *estado_inicial*;

O laço de aprendizado pode ser implementado conforme a estrutura do Pseudocódigo (2).

```
while true do
    numero_acoes_tomadas_episodio = 0 ;
    numero_acoes_tomadas_epoca = 0 ;
    estado_atual = estado_inicial ;
    while numero_acoes_tomadas_episodio <= tamanho_episodio do
        estado_atual = politica();
        atualiza_buffer();
        numero_acoes_tomadas_episodio++;
        if numero_acoes_tomadas_epoca >= tamanho_epoca then
            train(rede_neural);
            numero_acoes_tomadas_epoca = 0;
        else
            numero_acoes_tomadas_epoca++;
        end
    end
    if Resultado() then
        break;
    end
end
```

Pseudocódigo 2: Estrutura do Laço de Aprendizado.

Uma implementação em Matlab do laço de aprendizado é demonstrada no Apêndice C.3.

4 RESULTADOS E CONCLUSÕES

Este capítulo descreve os resultados obtidos ao final deste trabalho. Ao decorrer do desenvolvimento deste trabalho, a implementação do algoritmo passou por diversos aprimoramentos, resultando em algumas versões distintas. A princípio, seria implementado apenas *Q-Learning*. Porém, a decisão de implementar *Deep Q-Learning* resultou na versão final do algoritmo. Para analisar os resultados, será comprovado o ganho de performance quanto ao tempo de execução, número de episódios de treino e o erro em relação à posição desejada em relação às duas versões da implementação do algoritmo citadas. Para isso, a cada experimento, será utilizada a mesma configuração de manipulador robótico e a mesma posição tridimensional desejada para que a cinemática inversa seja resolvida, ou seja, para que a combinação de posições angulares das juntas atuadas do manipulador, que faça com que o elo de interesse seja posicionado sobre a posição desejada, seja encontrada.

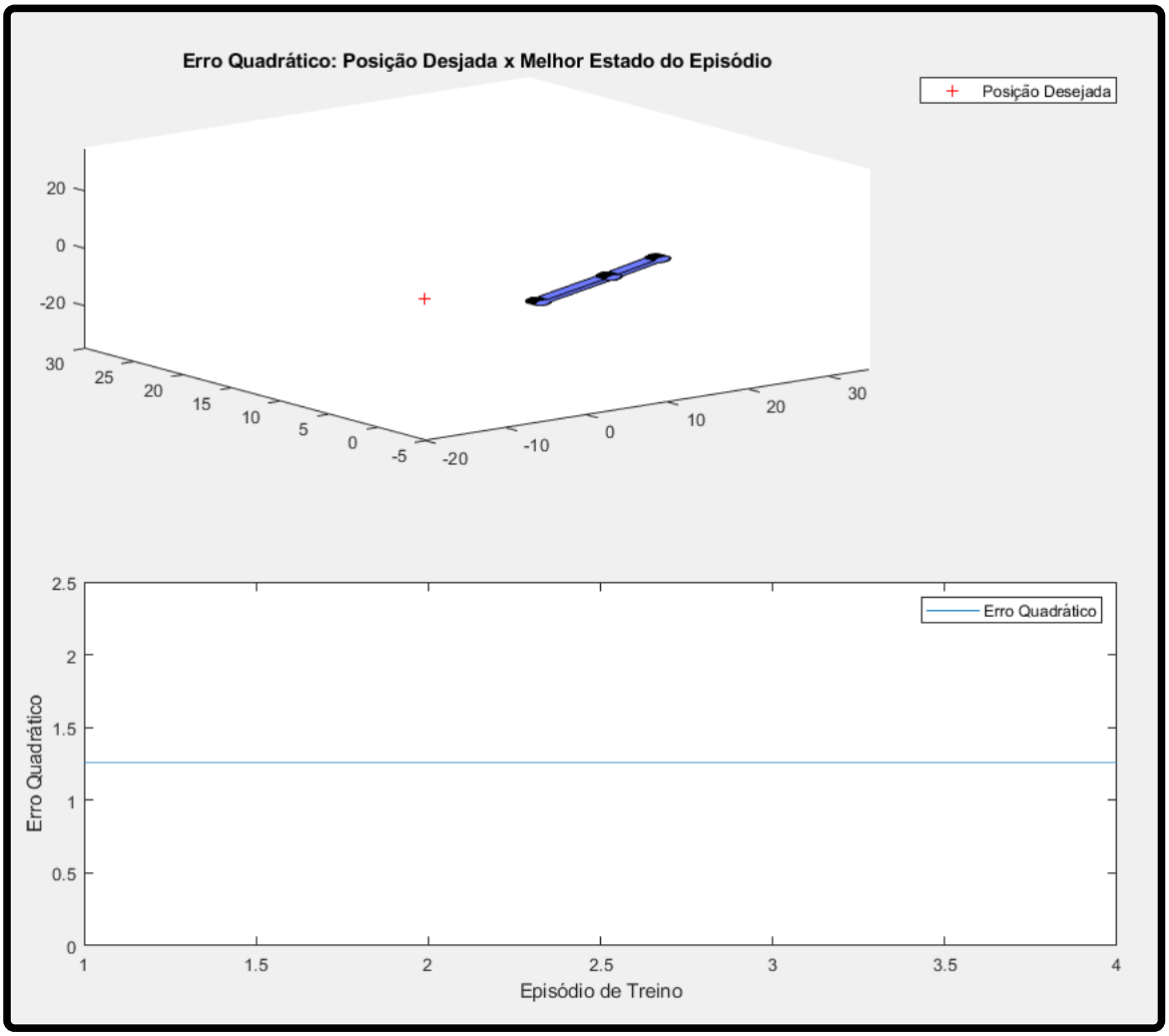
Para ilustrar a execução do algoritmo e avaliar os resultados obtidos, será utilizado um manipulador robótico de duas juntas de revolução. As implementações, tanto da versão *Q-Learning* quanto a *Deep Q-Learnig*, permitem que o usuário acompanhe a execução do algoritmo em tempo real através de gráficos e um modelo tridimensional do manipulador, conforme demonstrado na Figura 13 e na Figura 14. As figuras contém um modelo do manipulador e um gráfico que representa o erro quadrático das distâncias entre a posição desejada e a posição do melhor estado encontrado por época de aprendizado, conforme a Equação (18).

4.1 RESULTADOS

Para manter a integridade entre os testes, nesta seção as duas implementações *Q-Learning* e *Deep Q-Learning* serão testadas no aprendizado da cinemática inversa de quatro posições tridimensionais diferentes. Caso o agente não obtenha êxito em aprender exatamente a posição desejada, será considerada a posição obtida pelo estado final convergente do aprendizado. Ou seja, será considerada a "melhor"posição visitada pelo agente através de seu aprendizado, mesmo que esta seja diferente da posição desejada definida pelo usuário. A comparação será baseada nos seguintes itens:

- Tempo para resolução da cinemática inversa, ou seja, será medido e comparado o tempo que levará para cada um dos algoritmos encontrar o conjunto de posições angulares das juntas que leva o elo de interesse do manipulador à posição desejada;
- Erro quadrático medido pela Equação (18), comparando a posição final convergente do aprendizado e a posição desejada definida pelo usuário;
- Número de episódios de treino necessários para que o estado ótimo definido pelo aprendizado do agente convirja.

Figura 13 – Demonstração da execução do algoritmo em tempo real, no início de um novo processamento.

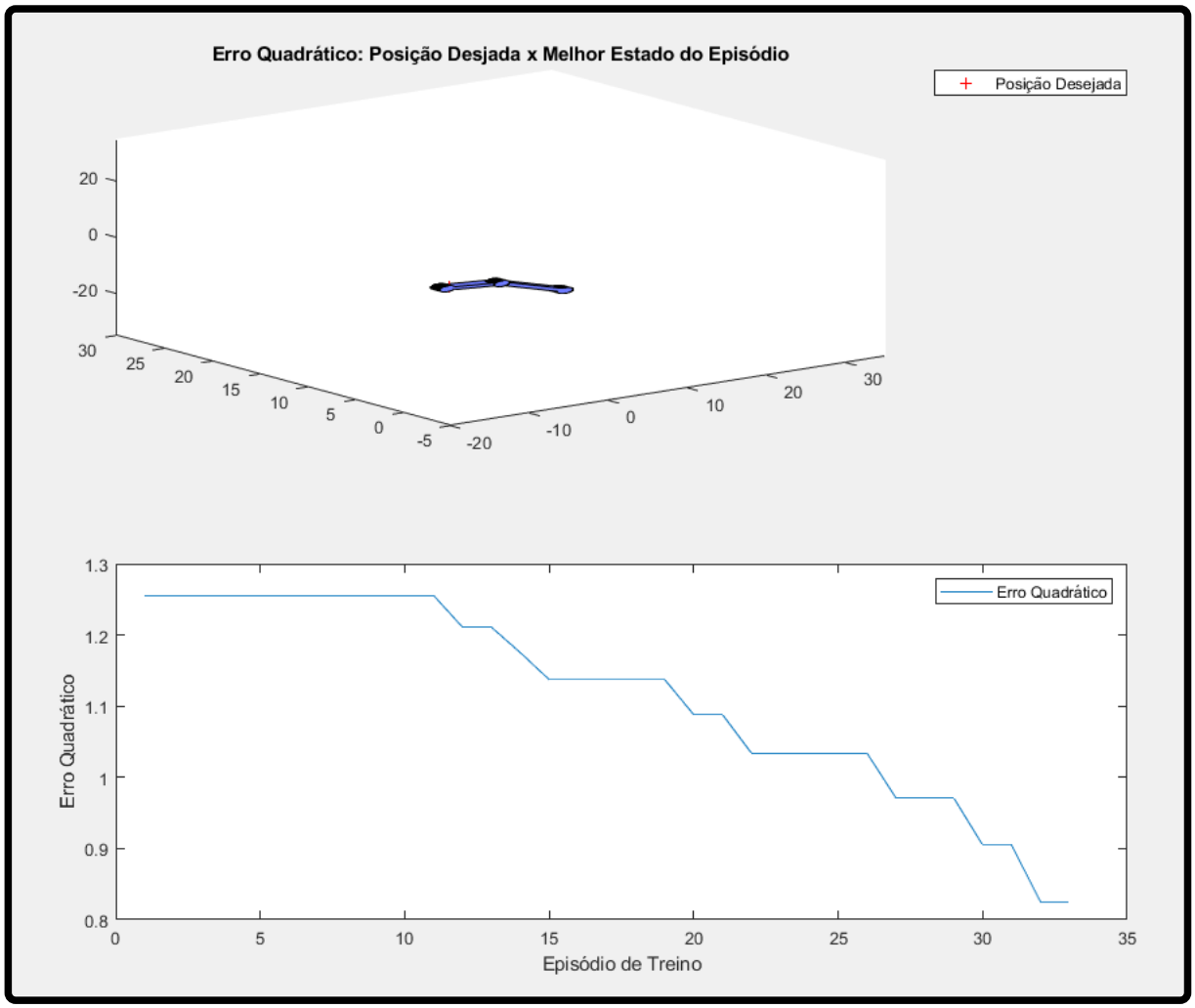


Fonte: Autor.

A Figura 15 representa o manipulador utilizado como referência, sendo que os comprimentos dos elos *l1* e *l2* possuem, respectivamente, $10u.c.$ e $7u.c.$ como medidas de comprimento dos elos. Os resultados serão demonstrados em 10 execuções por algoritmo, levando em consideração os itens citados acima.

Para ambos os algoritmos, foram mantidos os seguintes parâmetros de aprendizado: $\alpha = 0.85$ e $\gamma = 0.95$, referente à Equação (17). Especificamente para o algoritmo *Deep Q-Learning*, manteve-se o tamanho do buffer em 8000 registros, épocas de episódio a cada 500 ações tomadas pelo agente e as épocas de aprendizado a cada 100 ações tomadas pelo agente. A arquitetura da rede também foi mantida para todas as execuções do algortimo *Deep Q-Learning*, sendo 30 nós por camada e duas camadas de profundidade. Não foi realizada nenhuma otimização destes parâmetros.

Figura 14 – Demonstração da execução do algoritmo em tempo real, durante seu processamento.



Fonte: Autor.

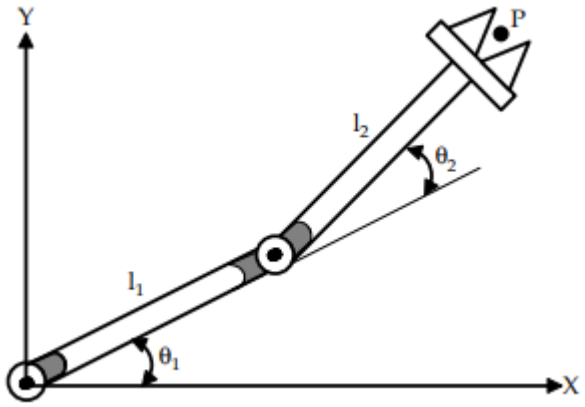
4.1.1 Posição Tridimensional (X = -14.0711, Y = 0, Z = 7.0711)

Para a posição tridimensional desejada (X = -14.0711, Y = 0, Z = 7.0711), a Tabela 2 e a Tabela 3 trazem as informações de 10 execuções das implementações do algoritmo de cinemática inversa com *Q-Learning* e *Deep Q-Learning*, respectivamente. A Figura 16[1] demonstra a primeira execução da Tabela 2, e a Figura 17[2] demonstra a primeira da Tabela 3.

[1] Link da demonstração da execução do algoritmo *Q-Learning*, para a posição (X = -14.0711, Y = 0, Z = 7.0711), no Youtube: https://www.youtube.com/watch?v=PKqMp1XHW6E&ab_channel=LuizSoldatelli

[2] Link da demonstração da execução do algoritmo *Deep Q-Learning*, para a posição (X = -14.0711, Y = 0, Z = 7.0711), no Youtube: https://www.youtube.com/watch?v=_KSHNnezjLk&ab_channel=LuizSoldatelli

Figura 15 – Manipulador robótico utilizado como exemplo para validação da implementação dos algoritmos *Q-Learning* e *Deep Q-Learning*.



Fonte: (KUCUK, Serdar; BINGUL, Zafer, 2006).

Tabela 2 – Execuções do algoritmo *Q-Learning* para a posição tridimensional (X = -14.0711, Y = 0, Z = 7.0711).

Execução	**Melhor estado**	**Erro quadrático**	**Episódios de treino**	**Tempo de execução em segundos**
1	135°, 45°	0	55	58.093221
2	135°, 45°	0	55	56.150280
3	135°, 45°	0	55	55.244339
4	135°, 45°	0	55	55.629668
5	135°, 45°	0	55	55.627668
6	135°, 45°	0	55	62.566350
7	135°, 45°	0	55	54.819110
8	135°, 45°	0	55	65.352871
9	135°, 45°	0	55	66.019053
10	135°, 45°	0	55	55.261446

Fonte: Autor.

4.1.2 Posição Tridimensional (X = 14.9497, Y = 0, Z = -4.9497)

Para a posição tridimensional desejada (X = 14.9497, Y = 0, Z = -4.9497), a Tabela 4 e a Tabela 5 trazem as informações de 10 execuções das implementações do algoritmo de cinemática inversa com *Q-Learning* e *Deep Q-Learning*, respectivamente. A Figura 18[3] demonstra a primeira execução da Tabela 4, e a Figura 19[4] demonstra a primeira da Tabela 5.

[3] Link da demonstração da execução do algoritmo *Q-Learning*, para a posição (X = 14.9497, Y = 0, Z = -4.9497), no Youtube: https://www.youtube.com/watch?v=_vvfvv6tx0M&ab_channel=LuizSoldatelli

[4] Link da demonstração da execução do algoritmo *Deep Q-Learning*, para a posição (X = 14.9497, Y = 0, Z = -4.9497), no Youtube: https://www.youtube.com/watch?v=C9fhOh36us0&ab_channel=LuizSoldatelli

Tabela 3 – Execuções do algoritmo *Depp Q-Learning* para a posição tridimensional (X = -14.0711, Y = 0, Z = 7.0711).

Execução	**Melhor estado**	**Erro quadrático**	**Episódios de treino**	**Tempo de execução em segundos**
1	$128°, 74°$	2.3057	13	283.0218
2	$-180°, -60°$	1.1593	7	75.1018
3	$180°, -89°$	3.9496	8	144.7609
4	$108°, 109°$	5.6747	8	126.3111
5	$141°, 30°$	0.6912	5	107.4815
6	$114°, 101°$	4.6942	10	215.7148
7	$143°, 25°$	0.8620	10	178.0897
8	$137°, 39°$	0.3272	7	34.46061
9	$170°, -42°$	0.2011	4	60.6809
10	$-180°, 23°$	10.0891	55	105.0706

Fonte: Autor.

Tabela 4 – Execuções do algoritmo *Q-Learning* para a posição tridimensional (X = 14.9497, Y = 0, Z = -4.9497).

Execução	**Melhor estado**	**Erro quadrático**	**Episódios de treino**	**Tempo de execução em segundos**
1	$0°, -45°$	0	67	75.266425
2	$0°, -45°$	0	67	67.681758
3	$0°, -45°$	0	67	76.828725
4	$0°, -45°$	0	67	71.087715
5	$0°, -45°$	0	67	76.416319
6	$0°, -45°$	0	67	74.663307
7	$0°, -45°$	0	67	67.973740
8	$0°, -45°$	0	67	74.653419
9	$0°, -45°$	0	67	78.106255
10	$0°, -45°$	0	67	79.260059

Fonte: Autor.

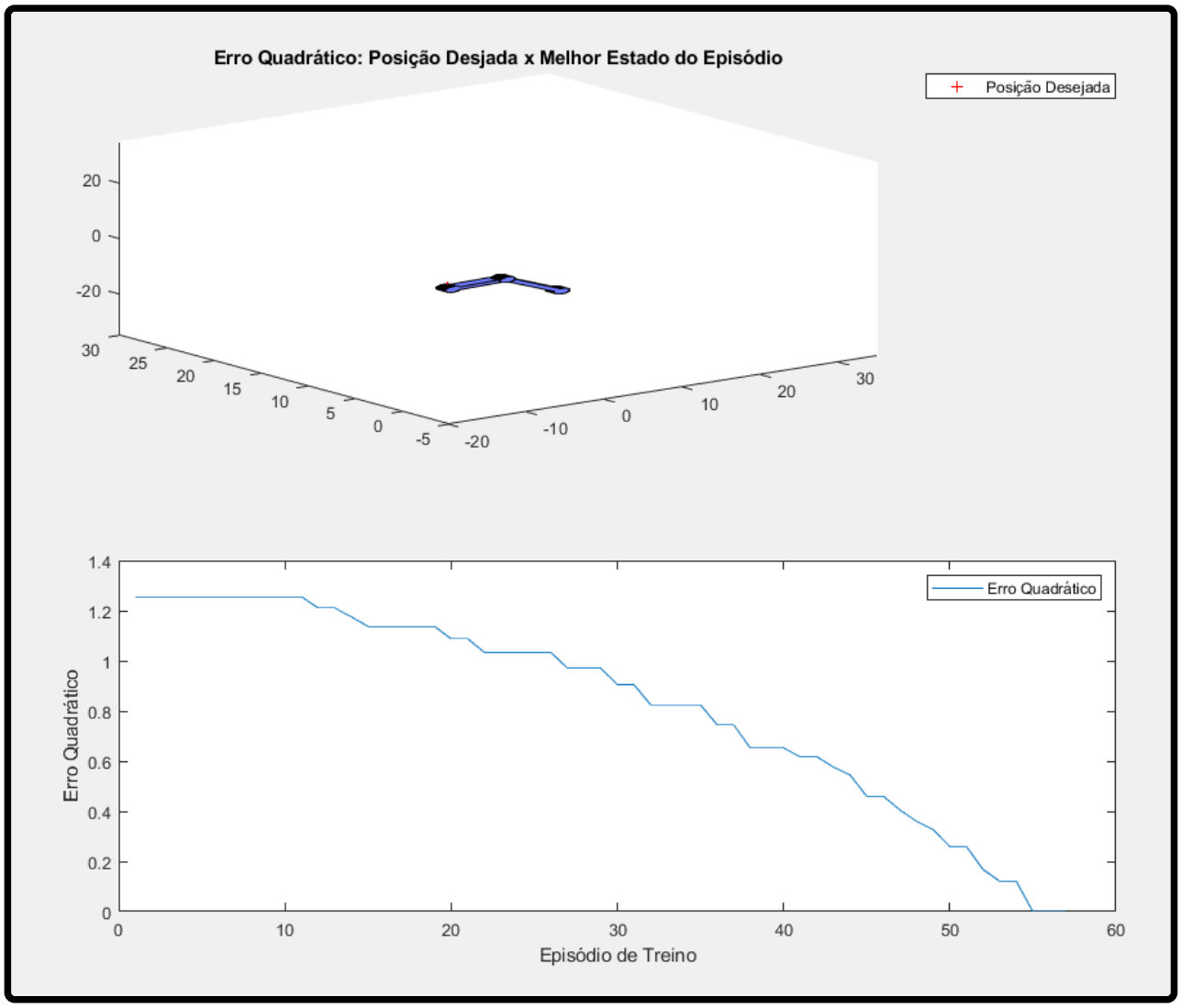
4.1.3 Posição Tridimensional (X = 0, Y = 0, Z = 15)

Para a posição tridimensional desejada (X = 0, Y = 0, Z = 15), a Tabela 6 e a Tabela 7 trazem as informações de 10 execuções das implementações do algoritmo de cinemática inversa com *Q-Learning* e *Deep Q-Learning*, respectivamente. A Figura 20[5] demonstra a primeira execução da Tabela 6, e a Figura 21[6] demonstra a primeira da Tabela 7.

[5] Link da demonstração da execução do algoritmo *Q-Learning*, para a posição (X = 0, Y = 0, Z = 15), no Youtube: https://www.youtube.com/watch?v=Ef6cJC2WWrM&ab_channel=LuizSoldatelli

[6] Link da demonstração da execução do algoritmo *Deep Q-Learning*, para a posição (X = 0, Y = 0, Z = 15), no Youtube: https://www.youtube.com/watch?v=N8QTjC84okI&ab_channel=LuizSoldatelli

Figura 16 – Primeira execução do algoritmo *Q-Learning* da Tabela 2, para a posição tridimensional (X = -14.0711, Y = 0, Z = 7.0711).



Fonte: Autor.

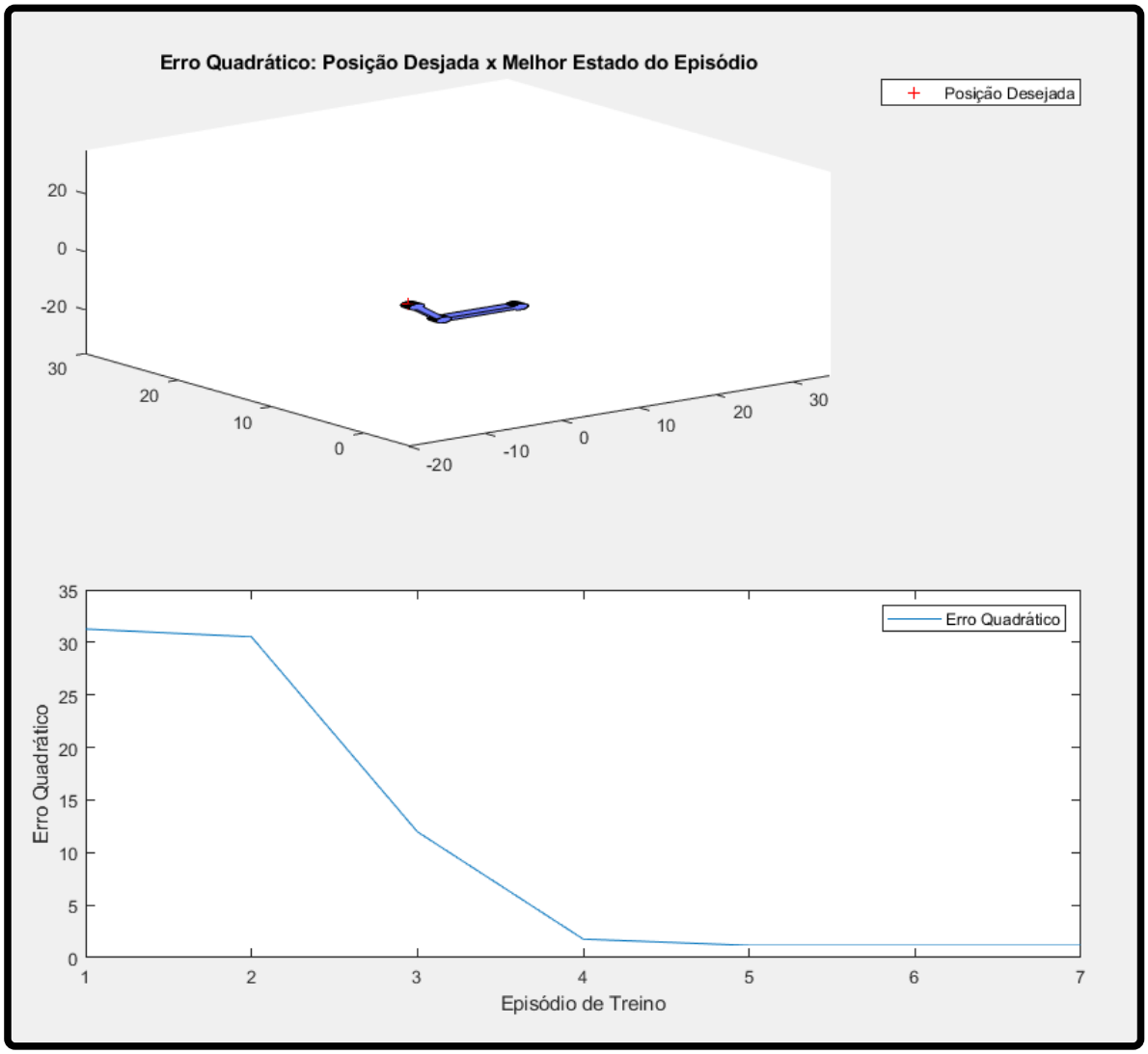
4.1.4 Posição Tridimensional (X = 5, Y = 0, Z = 0)

Para a posição tridimensional desejada (X = 5, Y = 0, Z = 0), a Tabela 8 e a Tabela 9 trazem as informações de 10 execuções das implementações do algoritmo de cinemática inversa com *Q-Learning* e *Deep Q-Learning*, respectivamente. A Figura 22[7] demonstra a primeira execução da Tabela 8, e a Figura 23[8] demonstra a primeira da Tabela 9.

[7] Link da demonstração da execução do algoritmo *Q-Learning*, para a posição (X = 5, Y = 0, Z = 0), no Youtube: https://www.youtube.com/watch?v=L8PHTo5j3iU&ab_channel=LuizSoldatelli

[8] Link da demonstração da execução do algoritmo *Deep Q-Learning*, para a posição (X = 5, Y = 0, Z = 0), no Youtube: https://www.youtube.com/watch?v=V7tG6G_3K0Y&ab_channel=LuizSoldatelli

Figura 17 – Primeira execução do algoritmo *Deep Q-Learning* da Tabela 3, para a posição tridimensional (X = -14.0711, Y = 0, Z = 7.0711).



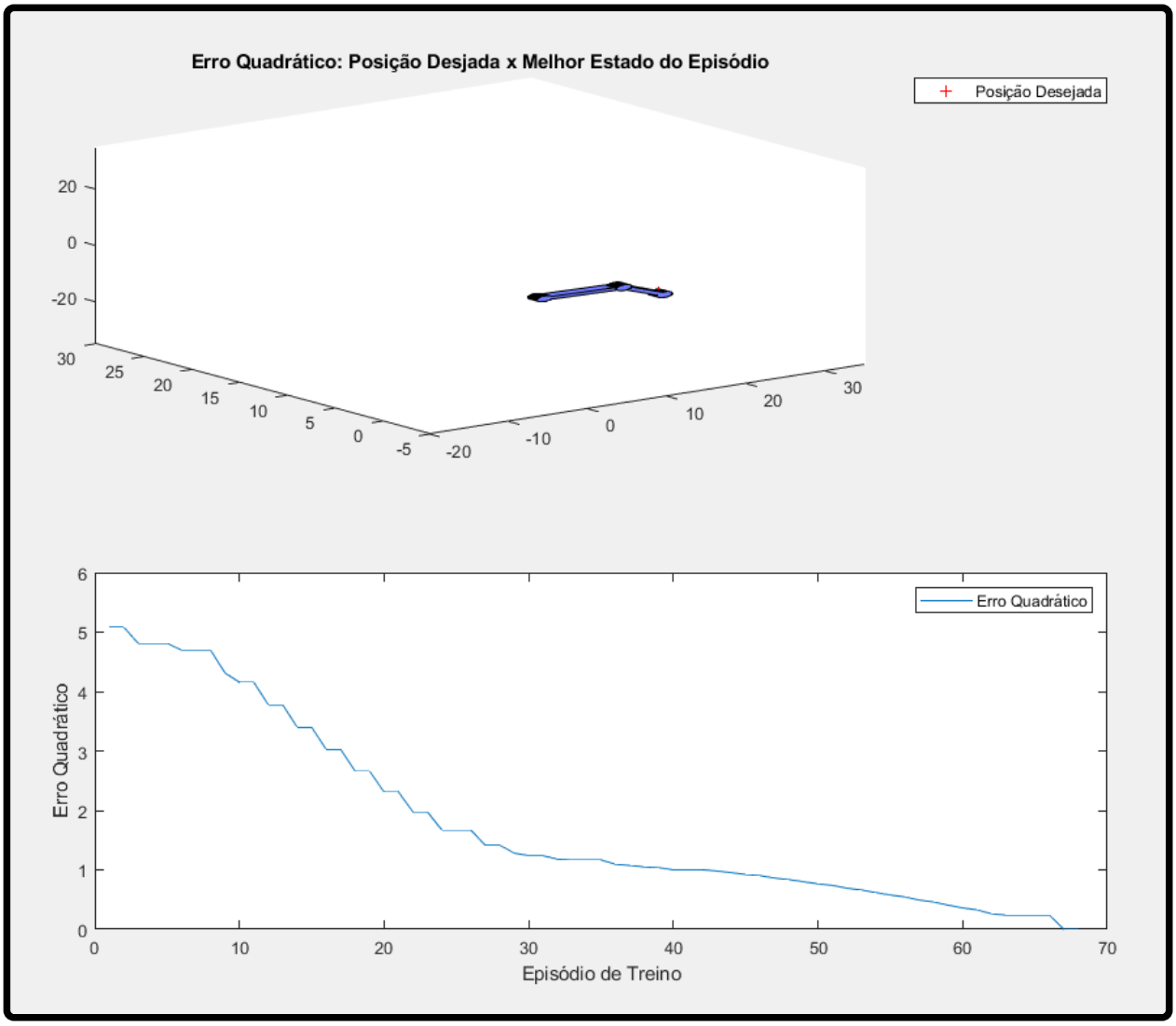
Fonte: Autor.

4.2 CONCLUSÕES FINAIS E SUGESTÕES FUTURAS

Através da analise dos resultados obtidos pela comparação entre o algoritmo que resolve a cinemática direta implementando *Q-Learning* e *Deep Q-Learning*, confirma-se que o algoritmo que implementa *Q-Learning* é consideravelmente mais estável, ao menos para a complexidade do manipulador testado. Isto é, para cada execução diferente do algoritmo, espera-se que o algoritmo encontre sempre o mesmo estado como melhor estado visitado pelo agente, tomando um semelhante intervalo de tempo de execução.

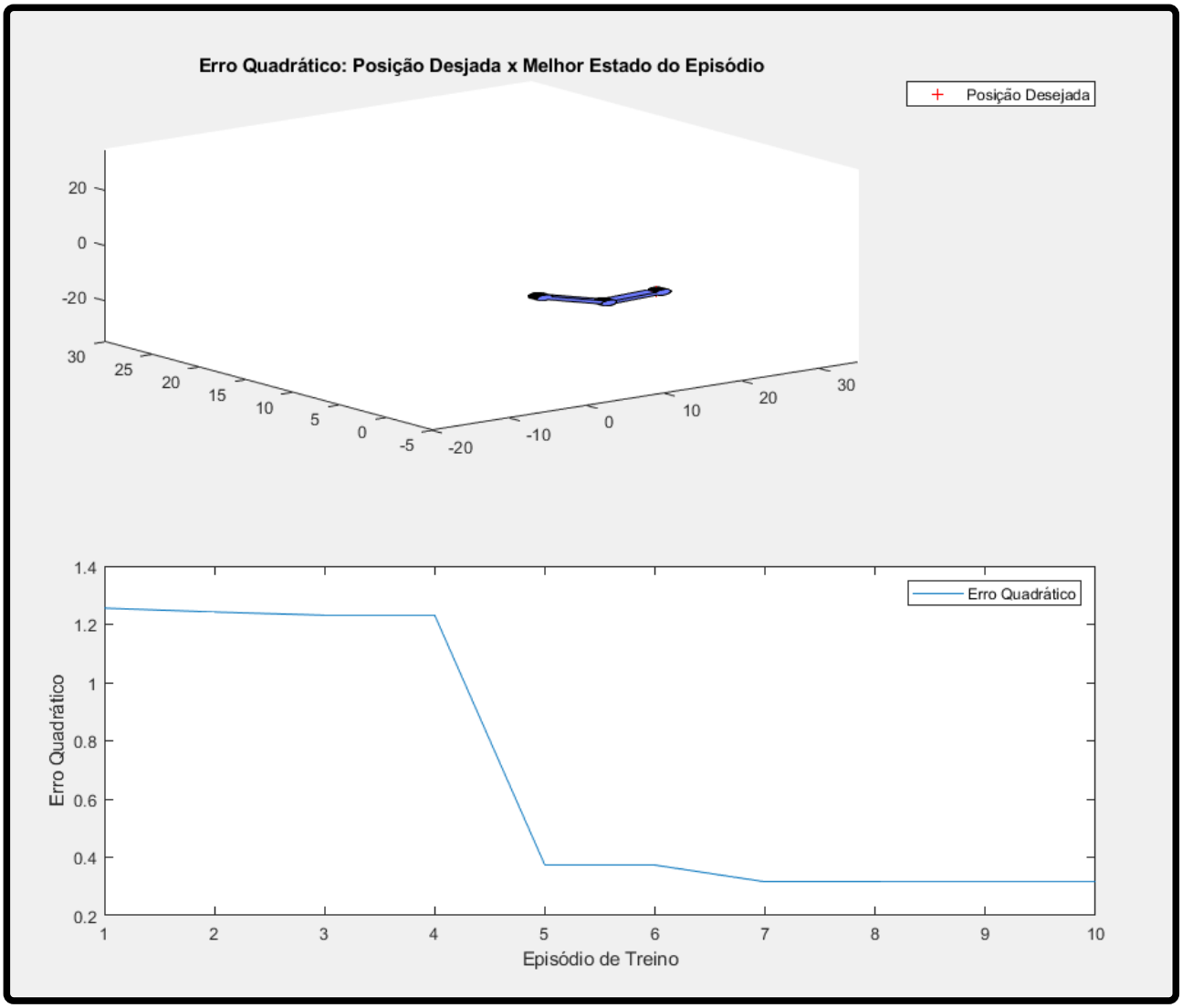
Porém, nota-se que, ao implementar a *Deep Q-Learning*, muitas variáveis que interferem a obtenção de resultados são adicionadas ao algoritmo. Alterações no tamanho do *buffer*, a arquitetura da rede neural e a definição das características exploratórias em

Figura 18 – Primeira execução do algoritmo *Q-Learning* da Tabela 4, para a posição tridimensional (X = 14.9497, Y = 0, Z = -4.9497).



Fonte: Autor.

Figura 19 – Primeira execução do algoritmo *Deep Q-Learning* da Tabela 5, para a posição tridimensional (X = 14.9497, Y = 0, Z = -4.9497).



Fonte: Autor.

Tabela 5 – Execuções do algoritmo *Depp Q-Learning* para a posição tridimensional (X = 14.9497, Y = 0, Z = -4.9497).

Execução	**Melhor estado**	**Erro quadrático**	**Episódios de treino**	**Tempo de execução em segundos**
1	$-35°, 39°$	0.3731	5	122.0928
2	$-70°, -27°$	0.8003	5	126.4248
3	$-10°, -21°$	0.9804	7	150.7009
4	$-4°, -35°$	0.4905	4	69.4526
5	$-38°, 48°$	0.1761	1	19.1590
6	$-37°, 46°$	0.0562	2	42.9830
7	$-13°, 14°$	1.1364	4	80.8419
8	$-2°, -39°$	0.3273	4	84.6840
9	$-26°, 19°$	1.0270	7	143.4882
10	$-32°, 28\circ$	0.9828	7	143.6079

Fonte: Autor.

Tabela 6 – Execuções do algoritmo *Q-Learning* para a posição tridimensional (X = 0, Y = 0, Z = 15).

Execução	**Melhor estado**	**Erro quadrático**	**Episódios de treino**	**Tempo de execução em segundos**
1	$67°, 57°$	0.0109	67	70.891899
2	$67°, 57°$	0.0109	67	75.883428
3	$67°, 57°$	0.0109	67	77.848916
4	$67°, 57°$	0.0109	67	81.797102
5	$67°, 57°$	0.0109	67	82.186608
6	$67°, 57°$	0.0109	67	83.684737
7	$67°, 57°$	0.0109	67	89.779219
8	$67°, 57°$	0.0109	67	80.093835
9	$67°, 57°$	0.0109	67	74.433406
10	$67°, 57°$	0.0109	67	74.014113

Fonte: Autor.

relação às ações de explotação podem influenciar diretamente na precisão e tempo de execução dos resultados obtidos.

Se uma esfera de raio igual a $1u.a.$ for traçada ao redor das quatro posições desejadas utilizadas nos testes, e caso considerarmos válidas as execuções que levam o elo de interesse do manipulador à uma posição dentro desta esfera, todas as execuções do algoritmo *Q-Learning* mostram-se capazes para tal. Porém, das 40 execuções do algoritmo *Deep Q-Learning*, somente 23 podem ser consideradas válidas, o que equivale à uma porcentagem de 57,5%. Caso o raio da esfera for aumentado para $2u.a.$, o número de posições válidas aumenta para 30, o que equivale à uma porcentagem de 75%.

Como sugestão futura, pode-se realizar os mesmos experimentos realizados neste trabalho com diferentes complexidades de manipuladores robóticos, assim como diferentes parâmetros de aprendizado e arquiteturas da rede neural.

Tabela 7 – Execuções do algoritmo *Depp Q-Learning* para a posição tridimensional (X = 0, Y = 0, Z = 15).

Execução	**Melhor estado**	**Erro quadrático**	**Episódios de treino**	**Tempo de execução em segundos**
1	$58°, 77°$	1.5470	4	78.3692
2	$70°, -6°$	6.5252	10	170.4413
3	$66°, 59°$	0.1406	3	65.4194
4	$95\circ, -11°$	1.9287	3	63.4438
5	$80°, 27°$	1.5731	5	96.1839
6	$84°, 15°$	1.8597	5	146.1621
7	$48°, 100°$	3.7264	8	152.2852
8	$58°, 81°$	1.9272	6	116.94150
9	$70\circ, 50°$	0.4660	3	61.3553
10	$61°, 71\circ$	1.0645	4	77.1133

Fonte: Autor.

Tabela 8 – Execuções do algoritmo *Q-Learning* para a posição tridimensional (X = 5, Y = 0, Z = 0).

Execução	**Melhor estado**	**Erro quadrático**	**Episódios de treino**	**Tempo de execução em segundos**
1	$-41°, 152°$	0.0462	161	183.929889
2	$-41°, 152°$	0.0462	161	177.256353
3	$-41°, 152°$	0.0462	161	188.923747
4	$-41°, 152°$	0.0462	161	183.062091
5	$-41°, 152°$	0.0462	161	178.903815
6	$-41°, 152°$	0.0462	161	182.173785
7	$-41°, 152°$	0.0462	161	178.457205
8	$-41°, 152°$	0.0462	161	206.269282
9	$-41°, 152°$	0.0462	161	205.375096
10	$-41°, 152°$	0.0462	161	215.112618

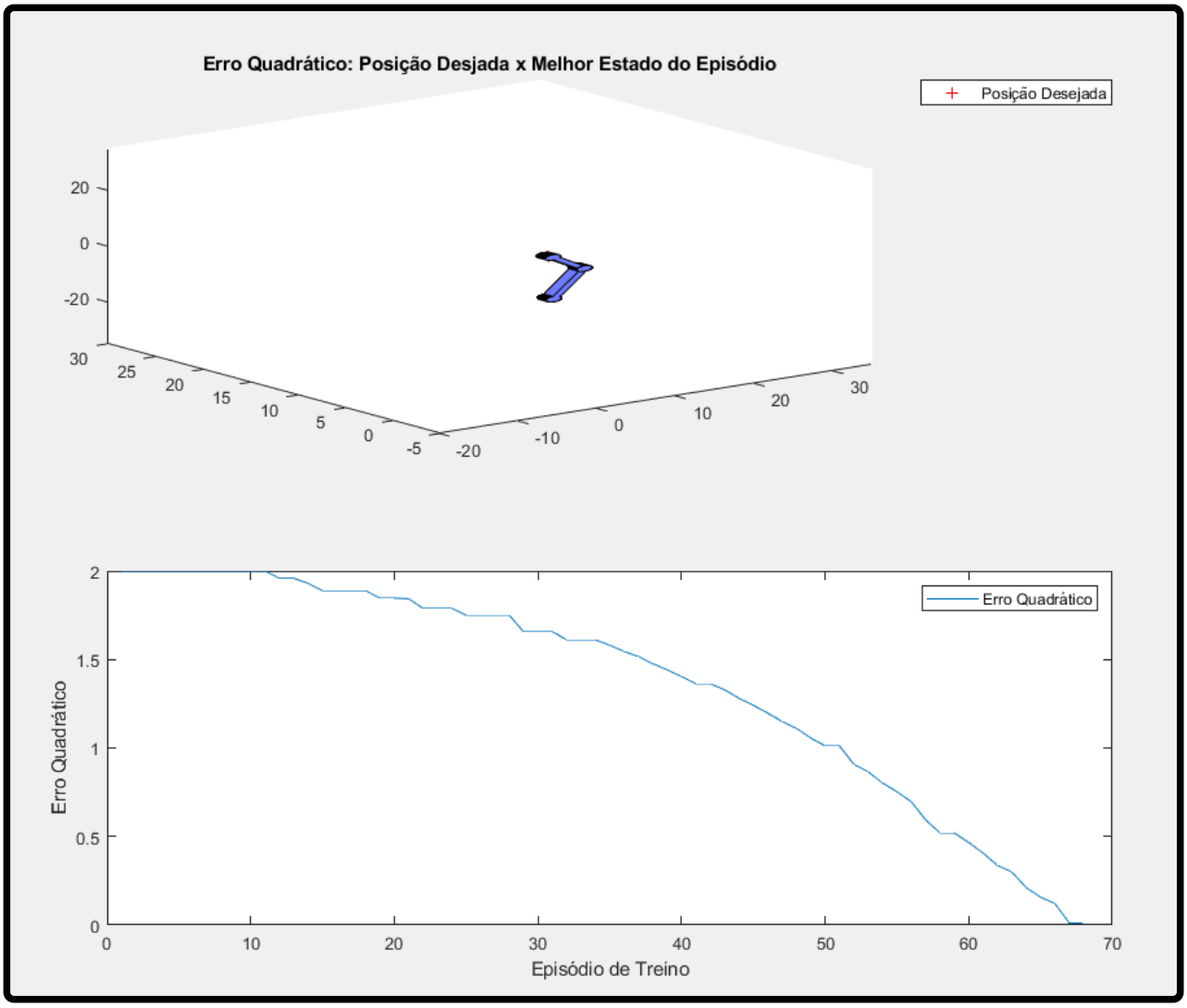
Fonte: Autor.

Tabela 9 – Execuções do algoritmo *Deep Q-Learning* para a posição tridimensional (X = 5, Y = 0, Z = 0).

Execução	**Melhor estado**	**Erro quadrático**	**Episódios de treino**	**Tempo de execução em segundos**
1	$41°, -152°$	0.0462	4	69.5464
2	$28°, -159°$	0.9634	6	130.5669
3	$83°, -137°$	2.8443	12	255.0411
4	$44°, -153°$	0.3389	5	90.6264
5	$0°, 180°$	2.0000	11	230.0968
6	$-39°, 148°$	0.5903	4	84.8133
7	$2°, -179°$	1.9966	7	104.0213
8	$49°, -169°$	2.4426	12	182.9914
9	$-38°, 153°$	0.2032	7	134.0550
10	$82°, -133°$	4.5333	7	137.2921

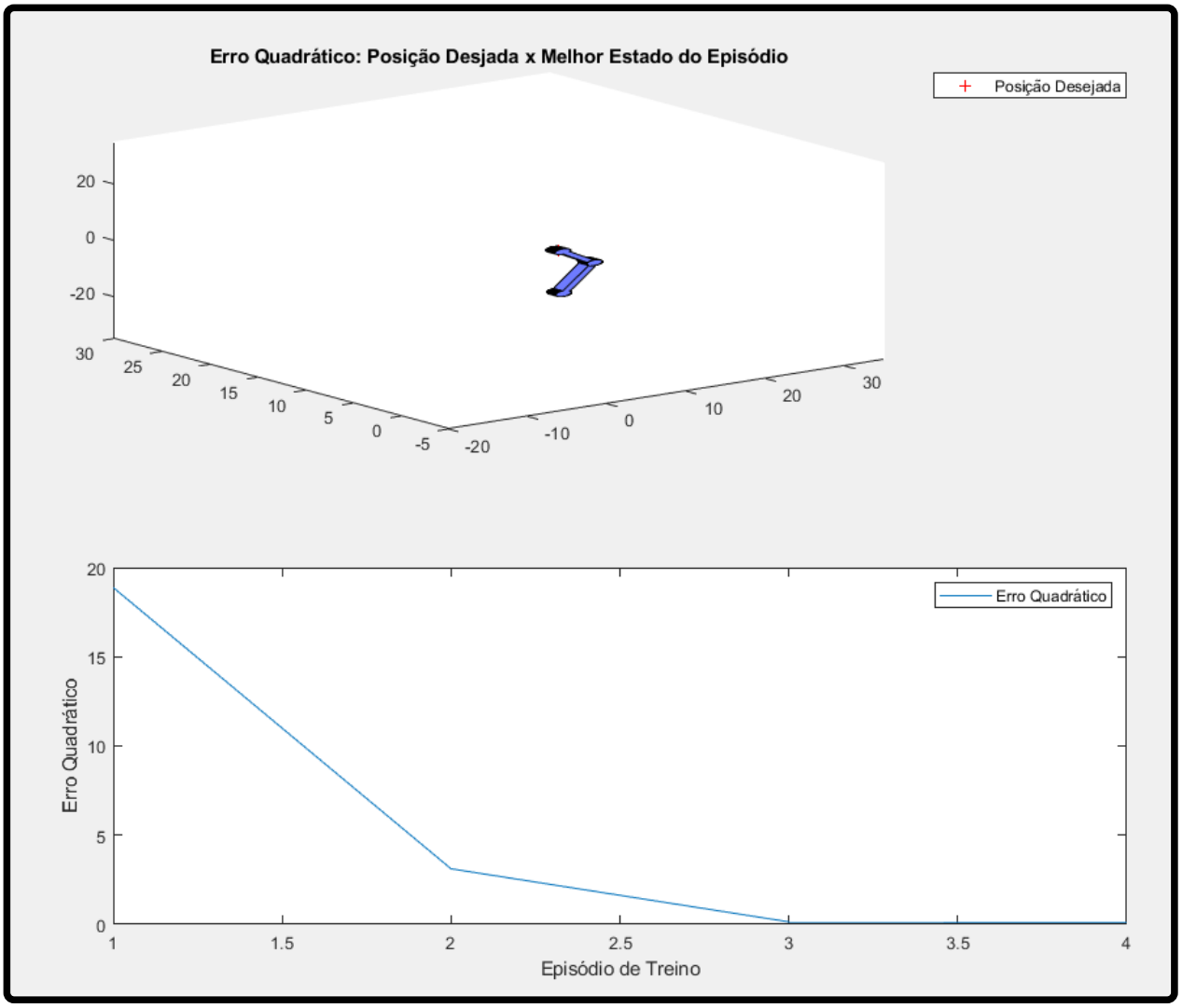
Fonte: Autor.

Figura 20 – Primeira execução do algoritmo *Q-Learning* da Tabela 6, para a posição tridimensional (X = 0, Y = 0, Z = 15).



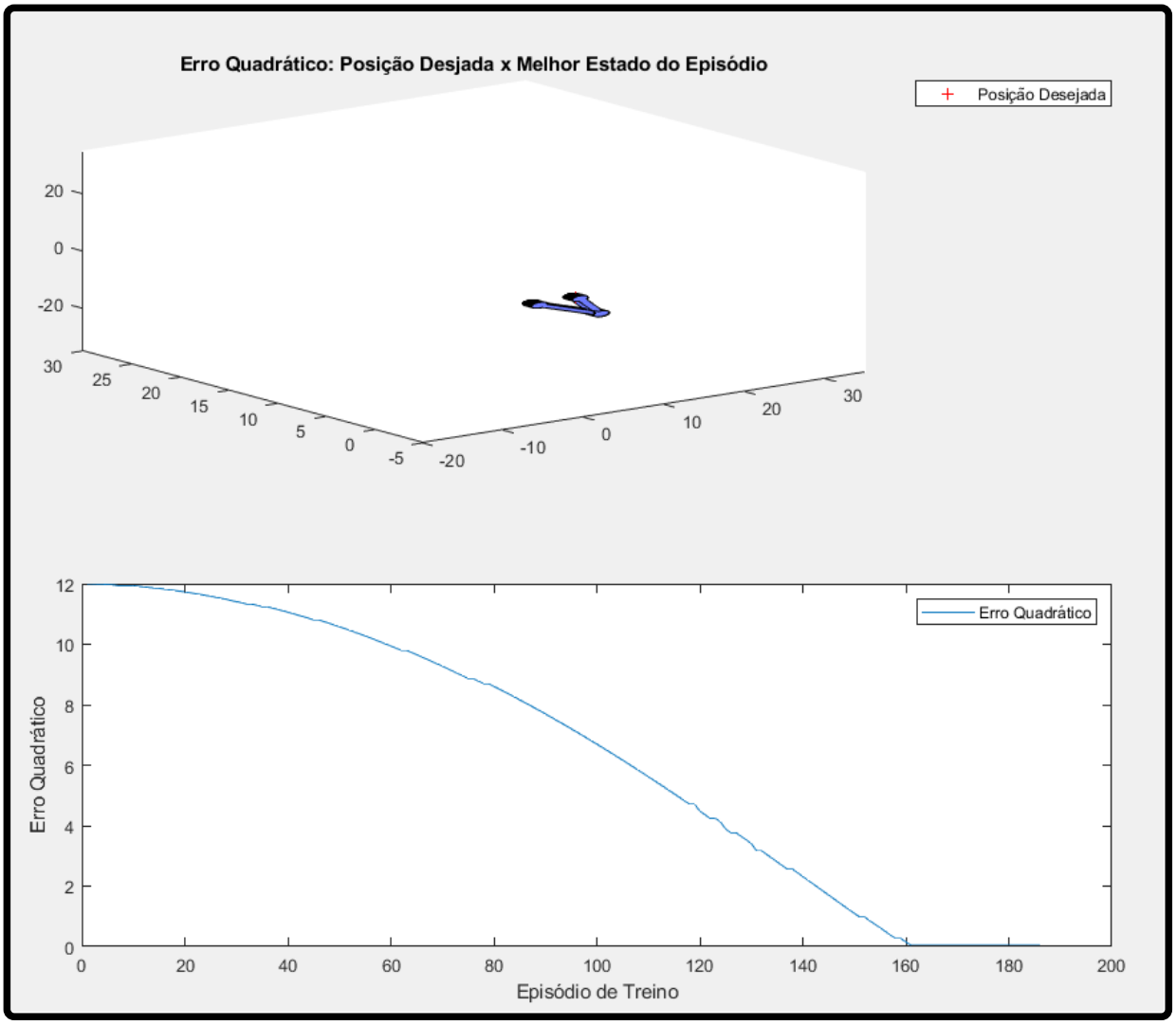
Fonte: Autor.

Figura 21 – Primeira execução do algoritmo *Deep Q-Learning* da Tabela 7, para a posição tridimensional (X = 0, Y = 0, Z = 15).



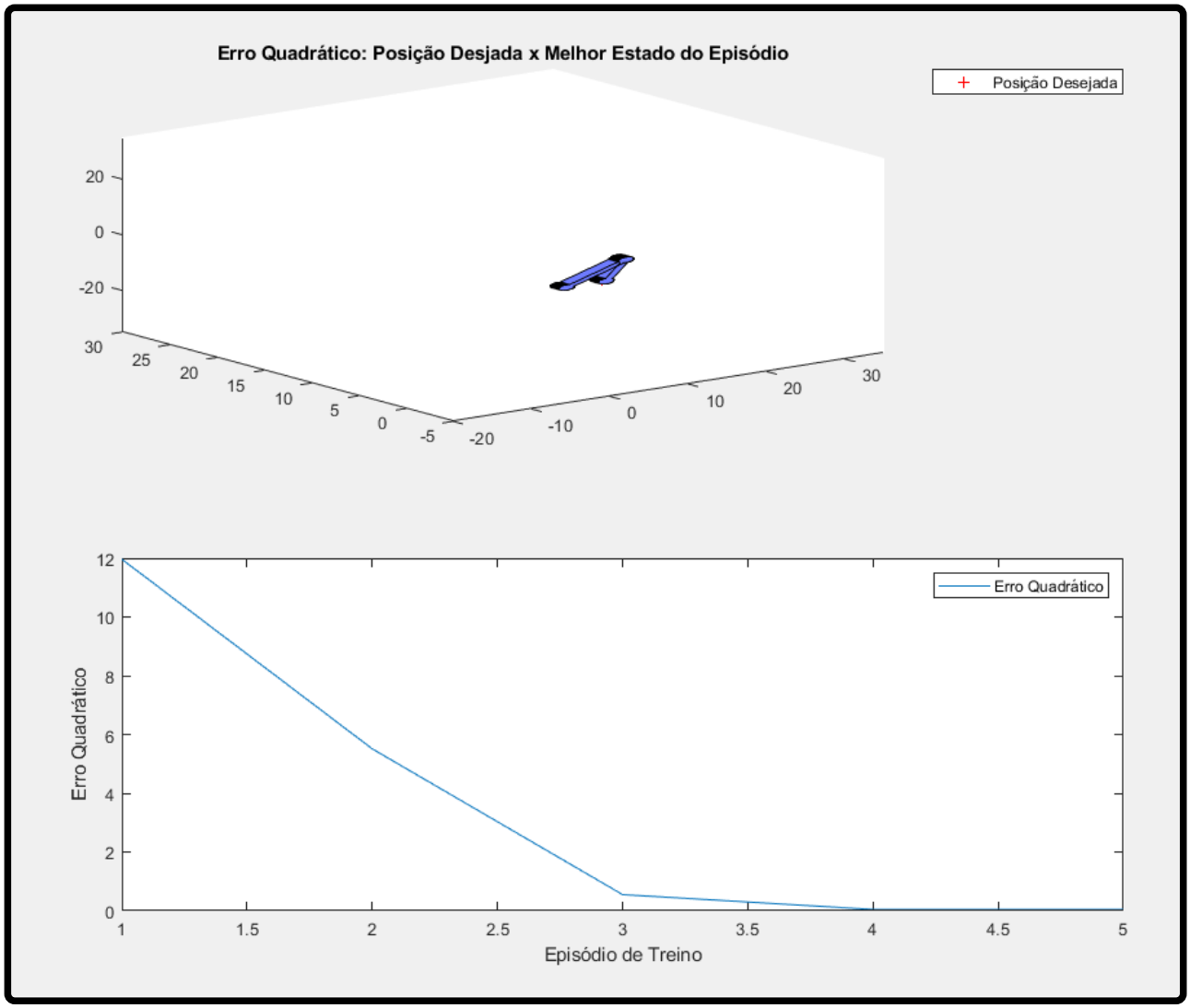
Fonte: Autor.

Figura 22 – Primeira execução do algoritmo *Q-Learning* da Tabela 8, para a posição tridimensional (X = 5, Y = 0, Z = 0).



Fonte: Autor.

Figura 23 – Primeira execução do algoritmo *Deep Q-Learning* da Tabela 9, para a posição tridimensional (X = 5, Y = 0, Z = 0).



Fonte: Autor.

REFERÊNCIAS

BISHOP, Chris M. Neural networks and their applications. **Review of scientific instruments**, American Institute of Physics, v. 65, n. 6, p. 1803–1832, 1994.

CRAIG, John J. **Introduction to robotics: mechanics and control**. [*S.l.*]: Pearson Educacion, 2005.

FONSECA, Pedro. **Application of Homogeneous Transformation Matrices to the Simulation of VLP Systems**. [*S.l.*: *s.n.*], nov. 2017.

GASPARETTO, A; SCALERA, L. A brief history of industrial robotics in the 20th century. **Advances in Historical Studies**, Scientific Research Publishing, v. 8, n. 1, p. 24–35, 2019.

GENTLEMAN, Robert; CAREY, Vincent J. Unsupervised machine learning. *In*: BIOCONDUCTOR case studies. [*S.l.*]: Springer, 2008. P. 137–157.

HENDERSON, David M. **Euler angles, quaternions, and transformation matrices for space shuttle analysis**. [*S.l.*], 1977.

JAZAR, Reza N. **Theory of applied robotics**. [*S.l.*]: Springer, 2010.

KENWRIGTH, B. Inverse kinematics with dual-quaternions, exponential-maps, and joint limits. **International Journal on Advances in Intelligent Systems**, Citeseer, v. 6, n. 1, 2013.

KÖKER, Raşlt. A genetic algorithm approach to a neural-network-based inverse kinematics solution of robotic manipulators based on error minimization. **Information Sciences**, Elsevier, v. 222, p. 528–543, 2013.

KUCUK, S.; BINGUL, Z. The inverse kinematics solutions of industrial robot manipulators. *In*: PROCEEDINGS of the IEEE International Conference on Mechatronics, 2004. ICM '04. [*S.l.*: *s.n.*], 2004. P. 274–279.

KUCUK, Serdar; BINGUL, Zafer. **Robot kinematics: Forward and inverse kinematics**. [*S.l.*]: INTECH Open Access Publisher, 2006.

LAPAN, Maxim. **Deep Reinforcement Learning Hands-On: Apply modern RL methods, with deep Q-networks, value iteration, policy gradients, TRPO, AlphaGo Zero and more**. [*S.l.*]: Packt Publishing Ltd, 2018.

LOPES, AM. Modelação cinemática e dinâmica de manipuladores de estrutura em série. **Porto: FEUP**, 2001.

LYNCH, Kevin M; PARK, Frank C. **Modern robotics**. [*S.l.*]: Cambridge University Press, 2017.

NIELSEN, Michael A. **Neural networks and deep learning**. [*S.l.*]: Determination press San Francisco, CA, USA, 2015. v. 25.

NORVIG, Peter; RUSSEL, Stuart. Artificial Intelligence. **A Modern Approach Prentice Hall Series in Artificial Intelligence**, v. 2, 1970.

PARRA-VEGA, Vicente; ARIMOTO, Suguru; LIU, Yun-Hui; HIRZINGER, Gerd; AKELLA, Prasad. Dynamic sliding PID control for tracking of robot manipulators: Theory and experiments. **IEEE Transactions on Robotics and Automation**, IEEE, v. 19, n. 6, p. 967–976, 2003.

ZAI, Alexander; BROWN, Bran. **Deep reinforcement learning in action**. [*S.l.*]: Manning Publications, 2020.

APÊNDICE A – Implementações em Matlab das funções que dão suporte ao algoritmo.

Apêndice A.1 – Implementação de função para obtenção da cinemática direta, utilizando manipulador com 2 juntas rotacionais e dois elos.

```
1  function [angles_output, pos] = cinematica_direta(angles_input,
       imprime)
2
3  % Comprimento dos Elos
4  L1 = 10;
5  L2 = 7;
6
7  % Angles
8  theta1 = angles_input(1);
9  theta2 = angles_input(2);
10
11 % Matriz H01
12 theta = 0;
13 d = 0;
14 alfa = 90;
15 a = 0;
16 H01 = DH(theta, d, alfa, a);
17
18 % Matriz H12
19 theta = theta1 ;
20 d = 0;
21 alfa = 0;
22 a = L1;
23 H12 = DH(theta, d, alfa, a);
24
25 % Matriz H23
26 theta = theta2;
27 d = 0;
28 alfa = 0;
29 a = L2;
30 H23 = DH(theta, d, alfa, a);
31
32 % Matriz Final
33 H03  = H01*H12*H23;
```

```
34
35 % Angulos de Euler
36 angles_output(1) = atan2d(H03(3,2), H03(3,3));
37 angles_output(2) = atan2d(-H03(3,1),sqrt((H03(3,2)^2) + (H03
   (3,3)^2)));
38 angles_output(3) = atan2d(H03(2,1), H03(1,1));
39
40 % Posicao Final
41 pos = H03(1:3,4);
42
43 end
```

Apêndice A.2 – Implementação da função que interpreta as ações escolhidas pelo agente, e atualiza o estado de acordo com a ação escolhida.

```
1  function [angulos_futuros] = interpretador_acao(acao,
   angulos_atuais)
2
3      switch acao
4          case 0
5              angulos_atuais(1) = angulos_atuais(1) + 1;
6              angulos_atuais(2) = angulos_atuais(2) + 1;
7          case 1
8              angulos_atuais(1) = angulos_atuais(1) + 1;
9          case 2
10             angulos_atuais(2) = angulos_atuais(2) + 1;
11         case 3
12             angulos_atuais(1) = angulos_atuais(1) + 1;
13             angulos_atuais(2) = angulos_atuais(2) - 1;
14         case 4
15             angulos_atuais(1) = angulos_atuais(1) - 1;
16             angulos_atuais(2) = angulos_atuais(2) + 1;
17         case 5
18             angulos_atuais(1) = angulos_atuais(1) - 1;
19         case 6
20             angulos_atuais(2) = angulos_atuais(2) - 1;
21         case 7
22             angulos_atuais(1) = angulos_atuais(1) - 1;
23             angulos_atuais(2) = angulos_atuais(2) - 1;
24     end
```

```
25
26 % Nao permite que os angulos sejam maiores ou menores que,
   respectivamente, 180 e -180.
27         if angulos_atuais(1) > 180
28             angulos_atuais(1) = 180;
29         end
30         if angulos_atuais(1) < -180
31             angulos_atuais(1) = -180;
32         end
33         if angulos_atuais(2) > 180
34             angulos_atuais(2) = 180;
35         end
36         if angulos_atuais(2) < -180
37             angulos_atuais(2) = -180;
38         end
39         angulos_futuros = angulos_atuais;
40 end
```

Apêndice A.3 – ImplementaçãO da função de recompensa.

```
1 function [Reward] = R(pos, pos_desejada)
2 Reward = (1/(sqrt((pos_desejada(1) - pos(1))^2+(pos_desejada(2)
   - pos(2))^2+(pos_desejada(3)-pos(3))^2)+1))^3;
3 end
```

Apêndice A.4 – Implementação da função de recompensa com penalidade, caso o agente escolha uma ação que leva à um estado que afaste o elo de interesse da posição desejada.

```
1 function [Reward] = R_com_penalidade(pos_atual, pos_passada,
   pos_desejada)
2
3 R_pos_atual    = R(pos_atual, pos_desejada);
4 R_pos_passada  = R(pos_passada, pos_desejada);
5
6 if R_pos_atual <= R_pos_passada
7     Reward = -1;
8 else
9     Reward = R(pos_atual, pos_desejada);
10 end
```

APÊNDICE B – Implementação em Matlab das funções responsáveis pela etapa de inicialização do algoritmo.

Apêndice B.1 – Implementação da criação do objeto que representa a rede neural, definindo-se sua arquitetura estrutura, ou seja, a quantidade de camadas e de nós.

```
nodes = 30;
hidden_layers = 2;
for i = 1:hidden_layers
    neural_structure(i) = nodes;
end
neural_network = fitnet([neural_structure]);
neural_network.layers{1}.transferFcn = 'poslin';
```

Apêndice B.2 – Implementação de tratativa para que a rede neural não separe parcelas dos dados de treino fornecidos para validação e testes.

```
neural_network.divideParam.trainRatio = 1;
neural_network.divideParam.valRation = .0;
neural_network.divideParam.testRatio = .0;
```

Apêndice B.3 – Implementação da atribuição dos parâmetros utilizados ao longo da execução do algoritmo.

```
pos_desejada = [-4.9497; 0; 14.9497];
gamma = 0.95;
alpha = 0.85;
tamanho_buffer = 8000;
tamanho_episodio = 500;
tamanho_epoca = 100;
```

Apêndice B.4 – Implementação da inicialização do *buffer*.

```
function [input, target, buffer_indice] =
   inicializar_input_target(pos_desejada)

buffer_indice = 4;

    input = [0;0];
    [angle_original, pos_original] = cinematica_direta([0;0],
       false);
    [angle_acao_0, pos_acao_0] = cinematica_direta(
       interpretador_acao(0, [0;0]), false);
```

```
8     [angle_acao_1, pos_acao_1] = cinematica_direta(
          interpretador_acao(1, [0;0]), false);
9     [angle_acao_2, pos_acao_2] = cinematica_direta(
          interpretador_acao(2, [0;0]), false);
10    [angle_acao_3, pos_acao_3] = cinematica_direta(
          interpretador_acao(3, [0;0]), false);
11    [angle_acao_4, pos_acao_4] = cinematica_direta(
          interpretador_acao(4, [0;0]), false);
12    [angle_acao_5, pos_acao_5] = cinematica_direta(
          interpretador_acao(5, [0;0]), false);
13    [angle_acao_6, pos_acao_6] = cinematica_direta(
          interpretador_acao(6, [0;0]), false);
14    [angle_acao_7, pos_acao_7] = cinematica_direta(
          interpretador_acao(7, [0;0]), false);
15
16    target = [R_com_penalidade(pos_acao_0, pos_original,
          pos_desejada);
17              R_com_penalidade(pos_acao_1, pos_original,
                    pos_desejada);
18              R_com_penalidade(pos_acao_2, pos_original,
                    pos_desejada);
19              R_com_penalidade(pos_acao_3, pos_original,
                    pos_desejada)
20              R_com_penalidade(pos_acao_4, pos_original,
                    pos_desejada);
21              R_com_penalidade(pos_acao_5, pos_original,
                    pos_desejada);
22              R_com_penalidade(pos_acao_6, pos_original,
                    pos_desejada);
23              R_com_penalidade(pos_acao_7, pos_original,
                    pos_desejada)];
24
25    input(:, 2) = [1;0];
26    [angle_acao_0, pos_acao_0] = cinematica_direta(
          interpretador_acao(0, [1;0]), false);
27    [angle_acao_1, pos_acao_1] = cinematica_direta(
          interpretador_acao(1, [1;0]), false);
28    [angle_acao_2, pos_acao_2] = cinematica_direta(
          interpretador_acao(2, [1;0]), false);
```

```
29      [angle_acao_3, pos_acao_3] = cinematica_direta(
            interpretador_acao(3, [1;0]), false);
30      [angle_acao_4, pos_acao_4] = cinematica_direta(
            interpretador_acao(4, [1;0]), false);
31      [angle_acao_5, pos_acao_5] = cinematica_direta(
            interpretador_acao(5, [1;0]), false);
32      [angle_acao_6, pos_acao_6] = cinematica_direta(
            interpretador_acao(6, [1;0]), false);
33      [angle_acao_7, pos_acao_7] = cinematica_direta(
            interpretador_acao(7, [1;0]), false);
34      target(:, 2) = [R_com_penalidade(pos_acao_0, pos_original,
            pos_desejada);
35                      R_com_penalidade(pos_acao_1, pos_original,
                            pos_desejada);
36                      R_com_penalidade(pos_acao_2, pos_original,
                            pos_desejada);
37                      R_com_penalidade(pos_acao_3, pos_original,
                            pos_desejada)
38                      R_com_penalidade(pos_acao_4, pos_original,
                            pos_desejada);
39                      R_com_penalidade(pos_acao_5, pos_original,
                            pos_desejada);
40                      R_com_penalidade(pos_acao_6, pos_original,
                            pos_desejada);
41                      R_com_penalidade(pos_acao_7, pos_original,
                            pos_desejada)];
42
43      input (:, 3) = [0;1];
44      [angle_acao_0, pos_acao_0] = cinematica_direta(
            interpretador_acao(0, [0;1]), false);
45      [angle_acao_1, pos_acao_1] = cinematica_direta(
            interpretador_acao(1, [0;1]), false);
46      [angle_acao_2, pos_acao_2] = cinematica_direta(
            interpretador_acao(2, [0;1]), false);
47      [angle_acao_3, pos_acao_3] = cinematica_direta(
            interpretador_acao(3, [0;1]), false);
48      [angle_acao_4, pos_acao_4] = cinematica_direta(
            interpretador_acao(0, [0;1]), false);
49      [angle_acao_5, pos_acao_5] = cinematica_direta(
```

```
        interpretador_acao(1, [0;1]), false);
50      [angle_acao_6, pos_acao_6] = cinematica_direta(
        interpretador_acao(2, [0;1]), false);
51      [angle_acao_7, pos_acao_7] = cinematica_direta(
        interpretador_acao(3, [0;1]), false);
52      target(:, 3) = [R_com_penalidade(pos_acao_0, pos_original,
        pos_desejada);
53                      R_com_penalidade(pos_acao_1, pos_original,
                            pos_desejada);
54                      R_com_penalidade(pos_acao_2, pos_original,
                            pos_desejada);
55                      R_com_penalidade(pos_acao_3, pos_original,
                            pos_desejada)
56                      R_com_penalidade(pos_acao_4, pos_original,
                            pos_desejada);
57                      R_com_penalidade(pos_acao_5, pos_original,
                            pos_desejada);
58                      R_com_penalidade(pos_acao_6, pos_original,
                            pos_desejada);
59                      R_com_penalidade(pos_acao_7, pos_original,
                            pos_desejada)];
60      input (:, 4) = [1;1];
61      [angle_acao_0, pos_acao_0] = cinematica_direta(
        interpretador_acao(0, [1;1]), false);
62      [angle_acao_1, pos_acao_1] = cinematica_direta(
        interpretador_acao(1, [1;1]), false);
63      [angle_acao_2, pos_acao_2] = cinematica_direta(
        interpretador_acao(2, [1;1]), false);
64      [angle_acao_3, pos_acao_3] = cinematica_direta(
        interpretador_acao(3, [1;1]), false);
65      [angle_acao_4, pos_acao_4] = cinematica_direta(
        interpretador_acao(0, [1;1]), false);
66      [angle_acao_5, pos_acao_5] = cinematica_direta(
        interpretador_acao(1, [1;1]), false);
67      [angle_acao_6, pos_acao_6] = cinematica_direta(
        interpretador_acao(2, [1;1]), false);
68      [angle_acao_7, pos_acao_7] = cinematica_direta(
        interpretador_acao(3, [1;1]), false);
69      target(:, 4) = [R_com_penalidade(pos_acao_0, pos_original,
```

```
        pos_desejada);
70                      R_com_penalidade(pos_acao_1, pos_original,
                            pos_desejada);
71                      R_com_penalidade(pos_acao_2, pos_original,
                            pos_desejada);
72                      R_com_penalidade(pos_acao_3, pos_original,
                            pos_desejada)
73                      R_com_penalidade(pos_acao_4, pos_original,
                            pos_desejada);
74                      R_com_penalidade(pos_acao_5, pos_original,
                            pos_desejada);
75                      R_com_penalidade(pos_acao_6, pos_original,
                            pos_desejada);
76                      R_com_penalidade(pos_acao_7, pos_original,
                            pos_desejada)];
77  end
```

Apêndice B.5 – Implementação da inicialização rede neural.

```
1  % Inicializar buffer de input e target:
2  [buffer_input, buffer_target, buffer_indice] =
       inicializar_input_target(pos_desejada);
3
4  % Inicializar rede neural com os valores m nimos de input/
       target:
5  [neural_network, neural_network_train_info] = train(
       neural_network, buffer_input, buffer_target);
```

APÊNDICE C – Exemplos de código fonte das funções do laço de aprendizado.

Apêndice C.1 – Implementação da função responsável por preencher o *buffer* com as informações do cenário colhidas pelo agente durante cada ação tomada no episódio de treino.

```
1  function [buffer_input, buffer_target, buffer_indice] =
       atualizar_buffer(neural_network, tamanho_buffer, buffer_input
       , buffer_target, buffer_indice, gamma, alpha, pos_desejada,
       estado_passado, acao_passada, estado_atual)
2
3      % Recupera o valor da utilidade.
4      [angle_passado, pos_passado] = cinematica_direta(
           estado_passado, false);
5      [angle_atual, pos_atual] = cinematica_direta(estado_atual,
           false);
6      Q_s_passado = neural_network(estado_passado);
7      Q_s_a_passado = Q_s_passado(acao_passada+1);
8      Q_s_atual = neural_network(estado_atual);
9
10     % Atualizacao da nova utilidade
11     Q_s_a_passado = Q_s_a_passado +  alpha*(R_com_penalidade(
           pos_atual, pos_passado, pos_desejada) + gamma * Q_s_atual
           (find(max(Q_s_atual))) - Q_s_a_passado);
12
13     % Descobrir se estado ja existe no buffer.
14     [estado_existe_buffer, buffer_indice_estado_passado] =
           ismember(estado_passado', buffer_input', 'rows');
15
16     % Atualiza buffer
17     if estado_existe_buffer
18         % Caso estado ja exista no buffer, atualiza os valores
               de utilidade.
19         buffer_target(acao_passada+1,
               buffer_indice_estado_passado) = Q_s_a_passado;
20     else
21         % Caso estado nao exista no buffer, adicionar o novo
               estado na matriz buffer_input, adicionar as
               utilidades na matriz buffer_target e incrementar a
```

```
            variavel de indice do buffer.
22          buffer_indice = buffer_indice + 1;
23          if buffer_indice > tamanho_buffer
24              buffer_indice = 1;
25          end
26          buffer_input(:, buffer_indice) = estado_passado;
27          buffer_target(:, buffer_indice) = neural_network(
               estado_passado);
28          buffer_target(acao_passada+1, buffer_indice) =
               Q_s_a_passado;
29      end
30  end
```

Apêndice C.2 – Implementação da estratégia *epsilon-greedy*.

```
1   function [acao] = politica(neural_network, estado_passado, epoca
       )
2
3       utilidades_estado = neural_network(estado_passado);
4       numero_aleatorio_1_100 = randi([1 100]);
5
6       threshold = ((-100)/25)*(epoca)+100;
7
8       if numero_aleatorio_1_100 < threshold
9           % Retorna acao aleatoria.
10          acao = randi([0 7]);
11      else
12          % Retorna a acao que possui a maior utilidade para o
               estado.
13          acao = find(utilidades_estado == max(utilidades_estado))
               -1;
14      end
15  end
```

Apêndice C.3 – Implementação do laço de aprendizado.

```
1   function laco_aprendizado(neural_network, pos_desejada,
       tamanho_buffer, gamma, alpha, tamanho_episodio, tamanho_epoca
       )
2   epoca = 0;
3   estado_inicial = [0;0];
4   numero_acoes_tomadas_epoca = 0;
```

```
5     while true
6         epoca = epoca+1;
7
8         % Inicializacao de um novo episodio de treino.
9         numero_acoes_tomadas_episodio = 0;
10        loop_episodio = true;
11        estado_passado = estado_inicial;
12        estados_episodio = estado_passado;
13        acao_passada = politica(neural_network, estado_passado,
              epoca);
14
15        %% Laco de Aprendizado.
16        while loop_episodio
17
18            % Encontra o novo estado baseado no estado e acao
                  passados.
19            estado_atual = interpretador_acao(acao_passada,
                  estado_passado);
20            acao_atual = politica(neural_network, estado_atual,
                  epoca);
21            estado_passado = estado_atual;
22            acao_passada = acao_atual;
23
24            % Atualiza o Buffer com a combinacao acao-estado
                  passados.
25            [buffer_input, buffer_target, buffer_indice] =
                  atualizar_buffer(neural_network, tamanho_buffer,
                  buffer_input, buffer_target, buffer_indice, gamma
                  , alpha, pos_desejada, estado_passado,
                  acao_passada, estado_atual);
26            numero_acoes_tomadas_epoca =
                  numero_acoes_tomadas_epoca+1;
27
28            % Verifica encerramento da epoca de aprendizado.
29            if numero_acoes_tomadas_epoca >= tamanho_epoca
30                numero_acoes_tomadas_epoca = 0;
31                [neural_network, neural_network_train_info] =
                      train(neural_network, buffer_input,
                      buffer_target);
```

```
32              else
33                  numero_acoes_tomadas_epoca =
                        numero_acoes_tomadas_epoca+1;
34              end
35
36              % Verifica encerramento do episodio de treino.
37              acoes_tomadas = acoes_tomadas+1;
38              estados_episodio(:, numero_acoes_tomadas_episodio) =
                    estado_atual;
39              if numero_acoes_tomadas_episodio > tamanho_episodio
40                  loop_episodio = false;
41              end
42          end
43
44          %% Tratativas de Resultados.
45
46          % Armazena os estados visitados pelo agente no episodio
                de treino.
47          [linhas_estados_episodio, colunas_estados_episodio] =
                size(estados_episodio);
48          for i = 1:colunas_estados_episodio
49              [angulo_estados_episodio, posicao_estados_episodio]
                    = arm_2_dof(estados_episodio(:, i), false);
50              erro_real_estados_episodio(i) = sqrt((pos_desejada
                    (1) - posicao_estados_episodio(1))^2+(
                    pos_desejada(3)-posicao_estados_episodio(3))^2);
51          end
52
53          % Armazena o menor erro encontrado.
54          menor_erro(epoca) =  min(erro_real_estados_episodio)
55
56          % Armazena o estado do menor erro encontrado.
57          ocorrencias_menor_erro = find(erro_real_estados_episodio
                == min(erro_real_estados_episodio));
58          estado_menor_erro(:, epoca) = estados_episodio(:,
                ocorrencias_menor_erro(1))
59
60          % Atualiza o estado inicial.
61          index_menor_estado = find(menor_erro == min(menor_erro))
```

```
            ;
62          estado_inicial = estado_menor_erro(:, index_menor_estado
                (1));
63      end
64  end
```