Universidade Federal de Minas Gerais
Escola de Engenharia
Curso de Graduação em Engenharia de Controle e Automação

Controle de um Manipulador Robótico via Aprendizado por Reforço e Cinemática Inversa para Desvio de Obstáculos

Izabela Borges Batalha

Orientador: Prof. Dr. Víctor Costa da Silva Campos
Coorientador: Prof. Dr. Vinicius Mariano Gonçalves

Belo Horizonte, Novembro de 2022

Monografia

Controle de um Manipulador Robótico via Aprendizado por Reforço e Cinemática Inversa para Desvio de Obstáculos

Monografia submetida à banca examinadora designada pelo Colegiado Didático do Curso de Graduação em Engenharia de Controle e Automação da Universidade Federal de Minas Gerais, como parte dos requisitos para aprovação na atividade Projeto Final de Curso II.

Belo Horizonte, Novembro de 2022

Resumo

A presente monografia aborda o problema de definir um caminho factível para um manipulador robótico. O objetivo desse trabalho é desenvolver um controle para que o efetuador do robô chegue na referência. Para tanto, o robô não deverá colidir com nenhum obstáculo, nem com ele mesmo e tampouco realizar movimentos que possam prejudicar sua estrutura mecânica.

Para tal fim, vale-se de uma das vertentes do aprendizado por reforço para a exploração do ambiente. Dessa forma, coletam-se informações a respeito do espaço no qual o manipulador está inserido, o que fará com que ele não colida com obstáculos.

Atrelado a isso, utiliza-se a cinemática inversa para melhorar a eficiência, haja vista que ela fornece um conhecimento prévio do ambiente. Dessa forma, evita-se que haja a exploração exacerbada pelo agente e garante-se que os limites físicos serão respeitados.

Abstract

This monograph addresses the problem of defining a feasible path for a robotic manipulator. The objective of this work is to develop a control so that the robot achieves the reference. Therefore, the robot must not collide with any obstacle, not even with itself or perform movements that may damage its mechanical structure.

For this purpose, one of the strands of reinforcement learning is used to explore the environment. Therefore information is collected about the space in which the manipulator is inserted, which will ensure that the robot does not collide with obstacles.

Linked to this, inverse kinematics is used to improve efficiency given that it provides prior knowledge of the environment. In this way, exacerbated exploration by the agent is avoided and guarantees that the physical limits requirements.

Agradecimentos

Em primeiro lugar, eu agradeço a Deus, pois sem o apoio e a sabedoria d'Ele, eu não teria chegado até aqui. Em segundo lugar, agradeço à minha mãe, Lilian, que é a minha base e meu maior exemplo de força e superação.

Ao meu irmão, Ivan Carvalho, que sempre me apoia e fornece alguma novidade sobre o Cruzeiro para que eu me distraia quando estou muito cansada. À Docinho (*in memorian*), que foi minha fiel companheira por mais de dez anos e sei que, onde quer que ela esteja, ela sempre estará comigo.

Ao meu amigo e colega de faculdade Pedro Henrique, que queimou alguns neurônios comigo durante essa jornada turbulenta da graduação. Ao meu colega e namorado Uiseman, que sempre acreditou em mim, mesmo quando nem eu acreditei. Ele sempre me incentivou e segurou minha mão quando eu precisei.

Ao meu colega e amigo Felipe Cazotti, por me aturar durante esses cinco anos de graduação e tentar me ajudar da melhor maneira possível. Ao meu colega e amigo Wallasce Leite, por me aconselhar quando precisei e me ensinar a programar.

À minha amiga e colega Bruna Ferreira, que além de me ajudar sempre que eu peço, tem o coração mais bondoso que eu já conheci. Ao meu colega Renato Junio, que foi minha companhia durante muito tempo na volta para casa.

Por último, mas não menos imporante, agradeço aos meus professores Dr. Víctor Costa da Silva Campos e Dr. Vinicius Mariano Gonçalves, que aceitaram o desafio de me orientar para fazer o Projeto Final de Curso em um tempo tão curto e por me auxiliarem da melhor maneira possível durante o percurso.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A robótica já está amplamente difundida no meio industrial. O uso de manipuladores e outros mecanismos autônomos são cruciais para os mais diversos sistemas produtivos. Dentre eles, pode-se citar o uso na área automobilística, na produção em massa de medicamentos, movimentações de cargas e inspeção de produtos. Esse conhecimento, atualmente, tem extrapolado a fronteira da indústria. Carros autônomos, robôs domésticos e drones são alguns exemplos disso. Entretanto, esse avanço significa que os robôs serão constantemente expostos a ambientes não controlados. Ou seja, é necessário que esses dispositivos tenham ciência do espaço no qual estão inseridos para não causar acidentes.

Para tanto, um dos mais importantes objetivos da robótica é que os robôs sejam dotados de habilidades motoras semelhantes às humanas. Isto é, que eles possam se locomover de forma suave e consciente do ambiente ao seu redor. Entretanto, sua movimentação, tal qual o corpo humano, deve ser condizente com os limites da sua estrutura. Isso significa que o robô não pode colidir consigo mesmo e nem forçar suas conexões mecânicas.

Uma maneira promissora de se conseguir alcançar tais objetivos é criando robôs que possam aprender novas habilidades sozinhos. Apesar disso, adquirir novas habilidades não é tão simples e envolve várias formas de aprendizado, tais como programação direta, aprendizado por reforço (mais usualmente conhecido como *Reinforcement Learning* ou RL) ou por imitação.

1.1 Motivação e Justificativa

A Indústria 4.0 ou Quarta Revolução Industrial foi responsável por aplicar as novas tecnologias do século XXI ao ambiente industrial. Com isto, tem-se a ampliação de fábricas inteligentes, em que se tem, por exemplo, a inteligência artificial e a robótica

incorporadas aos processos, que garatem a precisão, rapidez e qualidade.

Com a forte presença de manipuladores robóticos no chão de fábrica, é preciso que eles tenham ciência do espaço, de modo que não colidam com nenhum objeto ou pessoa. Isso porque, as colisões podem por em risco tanto a integridade física de pessoas quanto a do próprio manipulador. Com o objetivo de contornar a problemática de colidir com objetos parados, pode-se realizar o planejamento de movimento offline de sua trajetória, com a condição de que se tenha conhecimento prévio da localização desses obstáculos. Dessa forma, calcula-se a melhor rota antes de o robô executar qualquer movimento e assim, ele poderá executar tarefas sem colocar a si mesmo em risco e a outros objetos.

Entretanto, apesar de ser uma excelente forma de garantir a integridade do manipulador, a utilização de planejamento de movimento offline é computacionalmente cara e não garante que pessoas não sejam feridas em sua locomoção. Isto ocorre graças ao desconhecimento da posição dos operadores da fábrica. Dessa forma, é necessário que se utilize outra abordagem para que se tenha um comportamento mais reativo às mudanças do ambiente.

Para contornar esse problema, pode-se utilizar o aprendizado por reforço, sendo este modelado como um Processo de Decisão de Markov. Nessa modelagem, um agente reage a partir da observação do ambiente. Ou seja, ele aprende a realizar uma tarefa sem ser explicitamente programado e mede a qualidade de cada ação tomada através de recompensas. Por exemplo, pode-se aumentar a recompensa de um manipulador soldador, a medida que ele se aproxima do local onde se deve soldar. Em contrapartida, pode-se diminuir drasticamente a recompensa a medida que o robô se aproxima de um obstáculo ou pessoa. Dessa forma, o robô interage melhor com o ambiente ao seu redor, fazendo com que ele cumpra a tarefa sem por em risco sua própria estrutura, além de prevenir a ocorrência de acidentes ao seu redor.

1.2 Objetivos do Projeto

Tendo em vista que o UAIBot é um simulador disponível na web, o projeto pode ser desenvolvido de forma totalmente remota. Para tanto, o robô não deverá colidir com nenhum obstáculo, nem com ele mesmo e respeitar seus limites de atuação. Para tal fim, será utilizado uma das vertentes de RL, nomeada de Gradientes de Política Determinística Profundos (mais conhecida como *Deep Deterministic Policy Gradient* ou DDPG), aliada a cinemática inversa no controle do simulador, UAIBot. Ele é um simulador de baixo nível desenvolvido pelo professor Doutor Vinicius Mariano Gonçalves e feito na linguagem de programação Python com hospedagem web, conforme pode ser visto na figura 1.1.

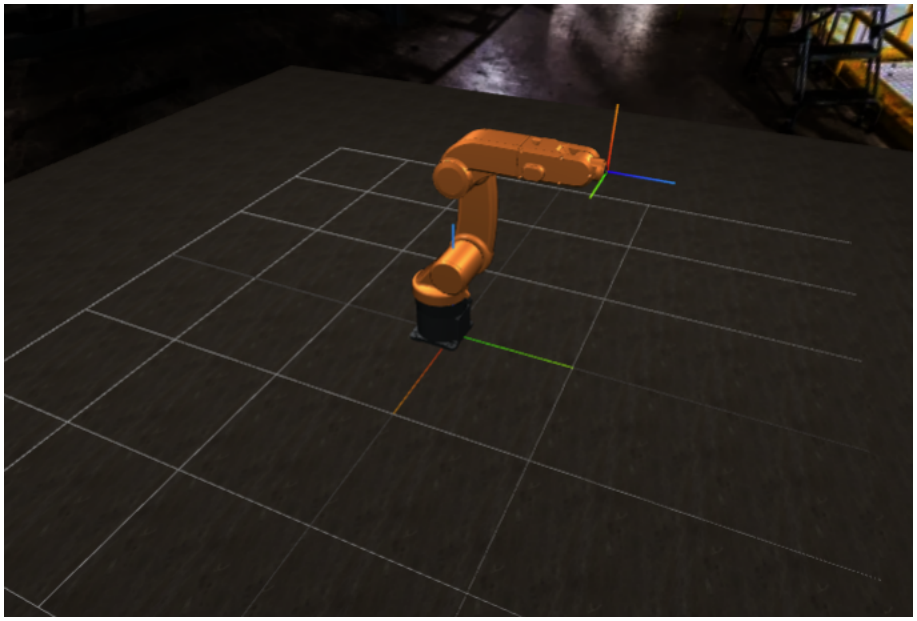

Figura 1.1: Manipulador Robótico, UAIBot, desenvolvido pelo professor Doutor Vinicius Mariano Gonçalves.

1.3 Local de Realização

Dada a natureza do projeto proposto, ele foi desenvolvido de forma totalmente remota. O orientador e coorientador deram suporte através de reuniões virtuais e troca de materiais, que permitiram o acompanhamento da evolução do trabalho.

1.4 Estrutura da Monografia

Este trabalho possui a seguinte estrutura:

- **Capítulo 2:** apresenta uma revisão bibliográfica sobre a problemática de se encontrar o melhor caminho para um manipulador robótico percorrer, sem que ele colida consigo mesmo ou com outros objetos.
- **Capítulo 3:** apresenta a modelagem do manipulador robótico e o funcionamento do aprendizado de máquina conhecido como *DDPG*.
- **Capítulo 4:** descreve a metodologia de desenvolvimento do projeto, expondo os resultados obtidos no planejamento do caminho para o simulador UAIBot por meio da utilização da cinemática inversa e *deep deterministic policy gradient*.
- **Capítulo 5:** conclusão do trabalho na qual se apresenta uma crítica da contribuição da monografia e se propõe trabalhos futuros.

Capítulo 2

Revisão da Literatura

2.1 Desafios do uso de aprendizado por reforço em manipuladores

Apesar do volume significativo de pesquisas sobre o uso de *reinforcement learning* (RL) em manipuladores, Nguyen and La [2019] afirma que existe uma série de desafios para a aplicação prática desse método. O principal está na amostragem ineficiente, podendo tornar a implementação inviável mesmo para os melhores algoritmos de RL. Uma forma de contornar esse problema consiste em utilizar um modelo do ambiente de interesse no treinamento da política. Entretanto, nem sempre essa representação do ambiente está disponível, ou ainda é muito complexa de ser obtida. Outro ponto relevante está na imprecisão do modelo, podendo resultar na menor performance do agente quando exposto ao ambiente real.

Assim como a imprecisão do modelo, Nguyen and La [2019] ressalta que variações no ambiente também prejudicam a performance do algoritmo. Como o intuito é que os robôs sejam expostos a ambientes complexos do mundo real, esse empecilho restringe significativamente a usabilidade dos algoritmos de *reinforcement learning*. Todavia, pode-se tornar a política de aprendizagem mais robusta às variações do ambiente. Assim como é feito com sistemas de controle, a performance é sacrificada em prol de maior tolerâncias às variabilidades.

Ainda segundo Nguyen and La [2019], por mais que a ideia da aprendizagem por reforço seja replicar a maneira que humanos aprendem novas habilidades, esses algoritmos ainda possuem uma diferença fundamental: as experiências passadas. Enquanto o algoritmo inicia a aprendizagem do zero, os seres humanos podem utilizar as experiências passadas para potencializar esse processo. Nesse sentido, o uso de modelos do ambiente pode auxiliar nessa questão, de modo que um mesmo ambiente possa ser

utilizado na aprendizagem de diferentes tarefas.

Outra questão, segundo Nguyen and La [2019], está na convergência dos algoritmos de aprendizagem. As ações do agente sempre deverão avaliar quando explorar de forma local ou de forma global. A primeira se refere à escolha da melhor ação dadas as informações disponíveis no instante. Já a exploração global busca maior compreensão do ambiente como um todo. As melhores soluções conseguem balancear esses dois fatores, sacrificando as recompensas no curto prazo para obter melhores resultados posteriormente. Isso evita tanto a convergência precoce para um mínimo local quanto a exploração excessiva e ineficiente.

Um algoritmo para o treinamento de políticas citado por Nguyen and La [2019] é o DDPG (*Deterministic Policy Gradient* ou Gradiente de Política Determinística). Nesse caso, o balanço entre os dois tipos de exploração mencionados é feito adicionando um ruído a ação durante o treinamento.

No trabalho desenvolvido por Zhong et al. [2022], pode-se notar o uso de algumas técnicas para contornar os problemas ressaltados por Nguyen and La [2019]. O algoritmo de *Deep-reinforcement learning* (aprendizagem por reforço profunda ou D-RL) foi utilizado no treinamento de um manipulador robótico com seis graus de liberdade. O intuito é que, ao final, o robô de solda seja capaz de chegar em uma posição alvo sem colidir com os elementos ao seu redor. Para tal, foi utilizada uma simulação de alta fidelidade do manipulador e do ambiente em volta.

Zhong et al. [2022] utilizou o DDPG para treinar o agente. Todavia, para maior velocidade de convergência, foi feita uma modificação, utilizando as informações da cinemática inversa do manipulador. Desse modo, o agente DDPG não precisa começar a aprendizagem do zero, haja vista que a cinemática inversa restringe o espaço de busca entre a posição de início e a objetivo. Sendo assim, o papel do algoritmo passa a ser ajustar a saída da cinemática inversa para evitar colisões.

Entretanto, Zhong et al. [2022] notou que a adição da cinemática inversa resultou na convergência prematura do algoritmo. Para sanar essa questão, adicionou-se um ganho variável à saída da cinemática inversa: $G(t) = ln(t + 1)$. Observa-se, portanto, que inicialmente a exploração global é favorecida e, no decorrer da aprendizagem, o peso da cinemática inversa se torna maior, acelerando a convergência. Os resultados da abordagem proposta foram comparados com outros algoritmos como *RRT-Connect*, *RRT*, *RRTStar*, e *BiTRRT*.

Notou-se que o DDPG modificado apresentou uma performance superior tanto em velocidade quanto em precisão da convergência. Em todas as tarefas, a solução de Zhong et al. [2022] conseguiu atingir o objetivo com o menor caminho percorrido dentre os algoritmos mencionados. Conforme mencionado pelo autor, o DDPG está limitado

pela baixa eficiência de amostragem. Apesar do trabalho desenvolvido ter mostrado melhorias nesse sentido, Zhong et al. [2022] aponta que trabalhos futuros são necessários para o desenvolvimento de soluções mais elegantes para esse problema.

2.2 Resumo do Capítulo

Nesse capítulo, foi feita uma revisão da literatura acerca das diferentes vertentes da aplicação de aprendizado por reforço em manipuladores. Dessa maneira, procurou-se agregar maior credibilidade e fundamentação ao método eleito para realização do projeto.

Capítulo 3

Modelagem

3.1 Introdução

O presente capítulo se dedica a apresentação de conceitos importantes relacionados a modelagem do robô, e técnica de aprendizado por reforço utilizados na monografia e a biblioteca para implementação da rede neural. O manipulador escolhido no projeto é o Kuka KR-5 R850, sendo ele composto por quatro elos conectados através de seis juntas rotativas. Ele está disponível no simulador robótico, UAIBot, do professor Doutor Vinicius Mariano Gonçalves e pode ser visto na figura 1.1.

Na extremidade de um manipulador é fixada uma ferramenta ou dispositivo, denominado efetuador, responsável por realizar uma tarefa. Esta pode ser, por exemplo, realizar a soldadem a arco ou a ponto, coletar ou dispensar um objeto em determinado lugar, dentre outras possibilidades. No UAIBot, tem-se um eixo que representa onde o efetuador estaria posicionado.

Dessa forma, como o objetivo do projeto é que o manipulador desvie de obstáculos sem forçar suas ligações mecânicas e sem colidir consigo mesmo, o efetuador do Kuka sairá de uma posição inicial até uma posição final respeitando esses critérios.

3.2 Cinemática Inversa

A convenção de Denavit-Hartenberg é amplamente utilizada nos cálculos da cinemática de manipuladores. Segundo Paredis and Khosla [1993], são necessários quatro parâmetros para descrever cada elo de um manipulador, sendo estes: comprimento, rotação, deslocamento do elo e ângulo das juntas. Valendo-se da convenção de Denavit-Hartenberg, pode-se encontrar a matriz de transformação Homogênea (MTH) do efetuador relativa à base. Para tanto, tem-se que:

$$MTH = T(q)_1^6 = \prod_1^6 T(q_i)_{i-1}^i, \tag{3.1}$$

em que $q = [q_1, q_2, q_3, q_4, q_5, q_6] \in \mathbb{R}^6$ é o ângulo da junta no espaço de configuração e $T(q_i)_{i-1}^i$ representa a matriz de transformação homogênea do i-ésimo relativo ao seu anterior. Para tanto:

$$T(q_i)_{i-1}^i = \begin{bmatrix} Q & p \\ 0_{1x3} & 1 \end{bmatrix}, \tag{3.2}$$

em que Q representa a matriz de rotação e p representa o deslocamento entre o i-ésimo link e seu anterior.

Por simplicidade, como o efetuador possui um eixo fixado em sua extremidade, a pose dele pode ser dada por $x_e = [p_e \ \phi_e] \in \mathbb{R}^6$, em que p_e é sua posição tridimensional e ϕ_e é a sua orientação em ângulos de Euler no espaço cartesiano. Assim, dada a velocidade $\dot{x}_e$ do efetuador, é possível encontrar a velocidade angular das juntas $\dot{q}$:

$$\dot{q} = J(q)^\dagger \dot{x}_e, \tag{3.3}$$

em que $J(q)^\dagger$ é a pseudo-inversa da Jacobiana. Como o manipulador escolhido possui 6 graus de liberdade, ela é uma matriz de ordem 6. Assim, pode-se resolver o problema da cinemática inversa, que é encontrar uma configuração q que faça o manipulador chegar na pose desejada, como:

$$q_{k+1} = q_k + \dot{q}\, dt, \tag{3.4}$$

em que dt representa o diferencial de tempo e os índices se relacionam aos passos do algortimo iterativo para calcular a configuração que resolve a cinemática inversa.

3.3 Aprendizado por Reforço

O aprendizado por reforço se caracteriza pelo agente interagir com o ambiente para aprender o que fazer, ou seja, como mapear situações em ações, para maximizar uma

recompensa, conforme explica Sutton and Barto [2018]. O agente não sabe quais ações tomar e assim, ele deve descobrir quais serão aquelas que retornarão as maiores recompensas.

3.3.1 Processos Finitos de Decisão de Markov

Processos Finitos de Decisão de Markov ou *Finite Markov Decision Processes* (MDP) representam uma formalização clássica de se realizar uma série de escolhas sequenciais. As ações tomadas não influenciam apenas recompensas imediatas, mas também futuras recompensas, estados e ações. No MDP, assim como em outras vertentes de RL, descreve-se o problema valendo-se das palavras-chave: agente, ambiente, recompensa, estado e ação.

O agente é aquele que toma as decisões e interage com o ambiente, sendo que este último compreende tudo o que não é o agente. Eles interagem continuamente, isso porque, o agente seleciona uma ação aleatória e o ambiente reage a ela com uma recompensa. Esta é um valor numérico que o agente busca maximizar com o tempo através de suas ações tomadas e é comumente representada por R.

A interação entre o agente e o ambiente ocorre através de uma sequência de passos em tempo discreto, $t = 0, 1, 2, 3, \ldots$. A cada passo t, tem-se o estado atual do ambiente, $S_t \in S$, sendo S o conjunto de todos os estados possíveis do ambiente. Dessa forma, a partir do estado atual, o agente elege uma ação, $A_t \in A$, em que A é o conjunto de todas as ações possíveis a serem realizadas.

Em decorrência da ação A_t, o estado atual passa a ser S_{t+1} e tem-se a recompensa $R_{t+1} \in R$, sendo R o conjunto de todas as recompensas possíveis. Assim, em um Processo de Decisão de Markov Finito, tem-se um número finito de estados, recompensas e ações que podem ser representados pela sequência

$$S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{N-1}, A_{N-1}, R_N, S_N, \tag{3.5}$$

em que N é o número total de passos. Para exemplificar, tem-se a figura 3.1 que foi traduzida de Sutton and Barto [2018] e representa a interação entre o ambiente e o agente.

3.3.2 Política e Funções de Valor

Funções de valor ou funções de estado ou ainda funções de pares de estado-ação estimam quão boa é a execução de uma ação em um determinado estado. Essa métrica é baseada

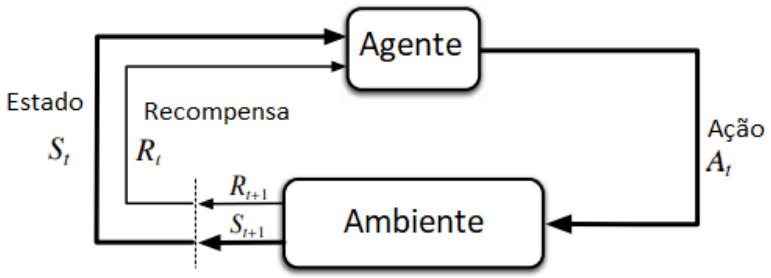

Figura 3.1: Diagrama que representa a interação entre o ambiente e o agente. Fonte: Sutton and Barto [2018]

na recompensa futura que se terá a partir de determinada ação. Assim, funções de valor são definidas através de políticas. Estas representam o mapeamento dos estados para a probabilidade de seleção de cada ação possível, conforme Sutton and Barto [2018].

Dessa forma, se o ator ou agente seguir uma política π no tempo t, a probabilidade que a ação $A_t = a$ ocorra no estado $S_t = s$ é $\pi(\text{a} \mid \text{s})$. O símbolo "|" define a probabilidadede $a \in A$ dado $s \in S$. G_t, por sua vez, representa a recompensa que se deseja maximizar com o tempo. Assim, tem-se que

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T, \tag{3.6}$$

onde T representa o passo final. Em aprendizado por reforço, entretanto, no geral, não se utiliza a recompensa como na equação 3.6. Isso porque, vale-se do conceito de recompensa com desconto, ou seja, o agente tenta selecionar ações que maximizarão a soma das recompensas descontadas que ele receberá no futuro. Assim, G_t é definido como

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \gamma^k R_{t+k+1}, \tag{3.7}$$

em que γ é um parâmetro nomeado de taxa de desconto e $0 \leq \gamma < 1$. A recompensa total de se selecionar a ação a em um estado s sob uma política π é definida como $q_\pi(s, a)$ e é chamada de função ação-valor. Ela é dada por:

$$q_\pi(s,a) \doteq \left[G_t \middle| S_t = s, A_t = a\right] = E_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a\right]. \quad (3.8)$$

A função de valor de um estado s sob a política π representa a recompensa esperada e é denotada por $v_\pi(s)$. Em Processos Finitos de Decisão de Markov, ela é dada por:

$$v_\pi(s) \doteq \left[G_t \middle| S_t = s\right] = E_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s\right], \text{para todo s} \in S, \quad (3.9)$$

em que $E_\pi[.]$ denota a recompensa esperada quando o agente segue a política π no tempo t.

Para Processos de Decisão de Markov Finitos pode-se encontrar uma política ótima. Isto é, pode-se definir que uma política π é melhor ou igual a π' se a recompensa esperada for maior para todos os estados. Em outras palavras, $\pi \geq \pi'$ se e somente se $v_\pi(s) \geq v_{\pi'}(s)$ para todo $s \in S$. Assim, a função valor ótima, denotada por v_*, é dada por:

$$v_*(s) \doteq \max_\pi v_\pi(s), \text{para todo s.} \in S \quad (3.10)$$

Políticas ótimas também possuem uma função ação-valor ótima, denotada por q_*, definida por:

$$q_*(s,a) \doteq \max_\pi q_\pi(s,a), \text{para todo s} \in S \text{ e a} \in A(s). \quad (3.11)$$

Pode-se, portanto, obter facilmente políticas ótimas uma vez que se tenha encontrado a função de valor ótima, o que satisfaz a equação ótima de Bellman:

$$v_*(s) = \max_a E \left[R_{t+1} + \gamma v_*(S_{t+1}) \middle| S_t = s, A_t = a\right]. \quad (3.12)$$

3.3.3 Métodos *on-policy* e *off-policy*

Existem duas formas de aprendizado por reforço que são os métodos *on-policy* e *off-policy*. O primeiro se vale de uma mesma política para selecionar e avaliar ações durante o aprendizado. O segundo elege diferentes políticas de seleção e avaliação das ações.

3.3.4 Método Ator-Crítico

O método Ator-Crítico, comumente conhecido na literatura como *Actor-Critic Methods*, é utilizado em algumas metodologias de aprendizado por reforço para aprender as funções ator e crítica. A função crítica estima a função de valor, enquanto a primeira é parametrizada pela política e usa a crítica para aprender seus parâmetros.

3.3.5 Gradientes de Política Determinística Profundos

Nessa seção não se aprofundará em Gradientes de Política Determinística Profundos, mais detalhes podem ser encontrados em Lillicrap et al. [2015]. DDPG representam uma das formas de aprendizado por reforço aprofundado. O DDPG usa o método *off-policy* e utiliza a função ação-valor ótima para encontrar a ação ótima, denominada $a_*(s)$. Ela é dada por:

$$a_*(s) \doteq \arg\max_a q_*(s,a), \text{para todo s} \in S \text{ e a} \in A(s). \tag{3.13}$$

DDPG é especificamente utilizado para ambientes que contenham espaço de ação contínuo. Isto porque, nessa condição não é possível avaliar infinitamente o espaço e aprender uma aproximação para q_* e em seguida, uma aproximação para a_*. Assim, como o espaço de ação é contínuo, pode-se assumir que q_* é diferenciável com relação à ação, conforme Silver et al. [2014]. Por simplificação de notação, será escrito q_* como Q nas próximas equações.

Assim, assume-se uma política ambiciosa, $\mu(s) = \max_a Q(s,a)$, que mapeia o estado atual em uma ação específica. Dessa forma, Lillicrap et al. [2015] utiliza uma aproximação para essa política, representada por θ^Q, através do treinamento de uma rede neural que busca a minimização da função perda, sendo esta dada por:

$$L(\theta^Q) = E[(Q(s_t, a_t|\theta^Q) - y_t)^2], \tag{3.14}$$

em que y_t é dado por:

$$y_t = R(s_t, a_t) + \gamma Q'(s_{t+1}, \mu'(s_{t+1}|\theta^{\mu'})|\theta^{Q'}), \tag{3.15}$$

em que se tem a rede de ação-valor alvo dada por $Q(s, a|\theta^{Q'})$ e a rede política alvo representada por $\mu(s|\theta^{\mu'})$. Elas são basicamente uma cópia das redes de ação-valor, representada por $Q(s, a|\theta^{Q})$, e política, $\mu(s|\theta^{\mu})$, respectivamente. A diferença entre as redes originais e as cópias é que as redes-alvo são atualizadas devagar pelo rastreamento das redes originais. Para tanto, existe um valor $\tau \ll 1$, que fará as seguintes atualizações:

$$\theta^{Q'} \longleftarrow \tau\theta^{Q} + (1 - \tau)\theta^{Q'}, \tag{3.16}$$

$$\theta^{\mu'} \longleftarrow \tau\mu^{Q} + (1 - \tau)\mu^{Q'}. \tag{3.17}$$

O objetivo da introdução de redes-alvo é fazer com que o processo de treinamento seja mais estável. Além das redes supracitadas, o algoritmo do DDPG ainda utiliza a metodologia de Ator-Crítico. Dessa forma, a rede de ação-valor é chamada de crítica enquanto a rede de política é chamada de ator. Assim, a performance da rede do ator é dada por:

$$\nabla_{\theta^{\mu}} J = E[\nabla_a Q(s, a|\theta^{Q})|s = s_t, a = \mu(s_t)\nabla\theta^{\mu}\mu(s|\theta^{\mu})|s_t]. \tag{3.18}$$

3.4 Keras

Keras, segundo Ketkar [2017], é uma API (*Application Programming Interface* ou Interface de Programação de Aplicativos) voltada a implementação de redes de aprendizagem profunda. As funcionalidades disponibilizadas são construídas através de Theano e *TensorFlow*. O primeiro é utilizado no cálculo do gradiente de funções perda. Já o segundo, é uma biblioteca de código aberto voltada à aprendizagem de máquinas, permitindo a criação e treinamento de redes neurais artificias. De forma resumida, o

Keras pode ser entendido como uma ferramenta para simplificar a implementação de redes neurais.

Desta forma, segundo Ketkar [2017], é possível ao usuário utilizar recursos como funções de ativação e otimizadores com baixo esforço de programação. Dentre os otimizadores disponibilizados, pode-se citar o RMSProp, AdaGrad, AdataDelta, Adam, Adamax, e Nadam. Esses recursos são fundamentais em processos como a atualização das redes crítica e do agente, permitindo que estes ocorram de forma mais rápida.

3.5 Resumo do Capítulo

Nesse capítulo foi apresentado todo o ferramental e conceitos importantes para o entendimento da metodologia de desenvolvimento do projeto presente no próximo capítulo. Para tanto, foram expostos os conceitos de cinemática direta e inversa, mostrou-se algumas definições de aprendizado por reforço para a introdução da DDPG e introduziu-se a API Keras.

Capítulo 4

Resultados

4.1 Introdução

Este capítulo é dedicado a explicação do desenvolvimento do projeto e a exposição dos resultados.

O objetivo do presente Projeto Final de Curso é fazer com que o efetuador do robô Kuka KR-5 850 que, a priori, está na quarta prateleira de uma estante vá para a prateleira imediatamente abaixo no ponto representado por um eixo, conforme pode ser visto na figura 4.1. Para o escopo do projeto, o manipulador não poderá colidir com nenhuma parte da estante ou consigo mesmo e nem poderá ultrapassar seus limites físicos. Para isso, ele deverá respeitar os limites presentes no manual GmbH, em que a numeração dos eixos é dada de acordo com a figura 4.2.

Eixos	Variação de Movimento (°)	Variação Máxima Velocidade Angular ($°/s$)
1	± 170	250
2	+45 a -190	250
3	+165 a -119	250
4	± 190	410
5	± 120	410
6	± 358	660

Tabela 4.1: Variações máximas da configuração de junta e velocidade angular do robô Kuka KR-5 850. Fonte: GmbH

4.2 Metodologia

Nessa seção, será detalhada a implementação para a escolha do caminho que o manipulador deverá seguir.

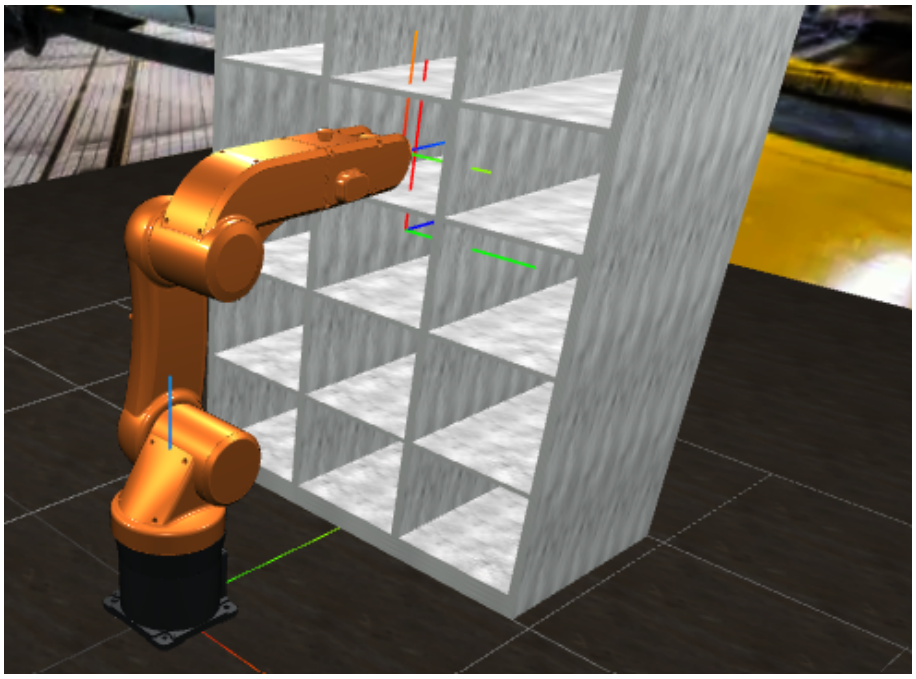

Figura 4.1: UAIBot na quarta prateleira da estante e posição desejada na prateleira imediatamente abaixo.

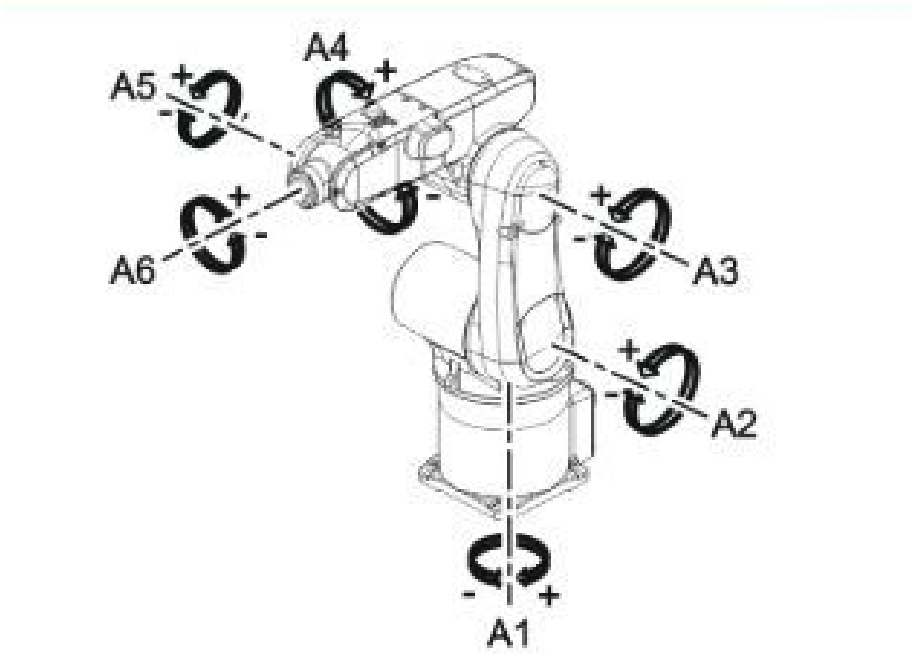

Figura 4.2: Numeração dos eixos do robô Kuka KR-5 850, segundo Ketkar [2017].

4.2.1 Espaço de estados

Como o espaço de estados do robô é infinito, definiu-se que ele seria dado por $s_t = [q, p_e, d, c] \in S$, tal qual Zhong et al. [2022]. Em que s_t representa o estado no tempo t, $q \in \mathbb{R}^6$ é a configuração de juntas do robô obtida através da função "robot.q", $p_e \in \mathbb{R}^3$ é a posição do efetuador do manipulador obtida através da função "robot.fkm()", d representa a distância euclidiana entre o efetuador e a posição desejada e c é um valor binário que indica se o robô colidiu com alguma das paredes do robô. Vale frisar que todas essas variáveis se referem ao robô no tempo t.

Assim, pode-se identificar um estado dentro do conjunto de estados de forma única e o agente poderá saber qual ação será realizada. Nota-se, portanto, que cada estado é representado por um vetor de 11 posições. Isso porque, q é um vetor com 6 elementos, p_e é um vetor de 3 elementos e os elementos d e c são escalares.

4.2.2 Função colisão

Cada prateleira e lateral da caixa foi construída através de um objeto "Box" presente no simulador UAIBot. Assim, a função colisão recebia como parâmetro o robô e a partir dele, usava as funções "robot.compute_dist(Box)" e "get_closest_item().distance" para verificar se o manipulador colidiu com determinada parede.

Dessa forma, verificou-se se em algum momento a distância entre qualquer parte do robô e as paredes ou prateleiras foi igual a 0,0000 m. Caso ele tenha colidido, retorna-se um binário com valor Verdadeiro e caso contrário, com valor falso.

4.2.3 Espaço de ação

O Kuka KR-5 850 é o ator e ele é caracterizado por possuir 6 graus de liberdade. Dessa forma, assim como em Zhong et al. [2022], o método de controle do agente será a velocidade angular ($\dot{q} \in \mathbb{R}^6$):

$$a_t = \dot{q} \in A. \tag{4.1}$$

Deve-se frisar que utilizou-se a função "numpy.clip()" para que os valores estivessem entre os limites físicos do robô presentes na tabela 4.1. O papel dessa função é saturar os valores nos limites máximos disponíveis de configuração de juntas e velocidade angular do robô, caso eles sejam ultrapassados.

4.2.4 Recompensa

Para a construção do sinal de recompensa, avaliou-se, a priori, o valor retornado pela função colisão. Isso porque, caso o robô tenha colidido, será retornada uma recompensa com o valor -10. Caso contrário, o valor retornado será proporcional à distância euclidiana elevada ao quadrado. Foi necessário que o valor estivesse ao quadrado para que a convergência ocorresse de forma mais rápida. Dessa forma, a função é dada como:

$$R = \begin{cases} -10 & \text{se colisão} \\ -d^2 & \text{caso contrário} \end{cases} \tag{4.2}$$

4.3 Treinamento

Utilizou-se o código apresentado em 1 retirado de Lillicrap et al. [2015] para realizar o treinamento. Para tanto, valeu-se da API Keras para a construção da rede neural DDPG e do otimizador Adam para o cálculo das funções ator e crítica. Além disso, utilizou-se um ruído branco denominado Ornstein-Uhlenbeck para realizar a exploração. Para tanto, escolheu-se o desvio padrão com valor de 0.1, $\theta = 0.15$ e dt $= 0.005$.

Porém, como o espaço de estados é muito grande, para que o algoritmo convergisse mais rapidamente para a posição desejada, utilizou-se a pseudo-inversa para obter a velocidade das juntas, $\dot{q}$. Para isso, adaptou-se o código de Zhong et al. [2022] que calculava $\dot{q}$ para se obter a cinemática inversa do manipulador. O pseudo-código adaptado pode ser visto em 2, em que x_e^* é a pose desejada.

É importante dizer que o cálculo de Δx_e no algoritmo 2 foi retirado de Zhong et al. [2022]. Ele é calculado através de:

$$\Delta x_e \longleftarrow x_e \ominus x_e^* \doteq \begin{bmatrix} p_e^* - p_e \\ vex(R_e^* R_e^T - I_{3x3}), \end{bmatrix} \tag{4.3}$$

em que p_e^* é a posição desejada, p_e é a posição atual do robô, $R_e^* \in \mathbb{R}^3$ é a matriz de rotação da posição desejada e $R_e \in \mathbb{R}^3$ é a matriz de rotação da posição atual do robô. O operador $vex(.)$, por sua vez, mapeia uma matriz antissimétrica em um vetor.

Assim, foi preciso adaptar o algoritmo 1 para que a posição do efetuador se tornasse

$$q_{t+1} = q_t + (G(t)\dot{q} + a_t) * dt, \tag{4.4}$$

Algorithm 1 Algoritmo DDPG

1: Inicializar aleatoriamente a rede crítica $Q(s, a|\theta^Q)$ e o ator $\mu(s|\theta^\mu)$ com pesos θ^Q e θ^μ
2: Inicializar rede objetivo Q' e μ' com pesos $\theta^{Q'} \leftarrow \theta^{Q'}$, $\theta^{\mu'} \leftarrow \theta^\mu$
3: Inicializar buffer de repetição R
4: **for** $episodio = 0, M-1$ **do**
5: Inicializar processo aleatório N para ação de exploração
6: Receber estado inicial de observação s_0
7: **for** $0, T-1$ **do**
8: Selecionar ação $a_t = \mu(s_t|\theta^\mu) + N$ de acordo com a política atual e o ruído de exploração
9: Executar ação a_t, observar a recompensa r_t e o novo estado s_{t+1}
10: Armazenar transição (s_i, a_i, r_i, s_{i+1}) em R
11: Fazer $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
12: Atualizar o crítico minimizando a perda: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
13: Atualizar a política do ator usando o gradiente de política amostrado
$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|s = s_i, a = \mu(s_i)\nabla\theta^\mu \mu(s|\theta^\mu)|s_i$$
14: Atualizar redes alvo
$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$
15: **end for**
16: **end for**

Algorithm 2 Algoritmo Pseudo Inversa em Zhong et al. [2022]

1: Calcula pose atual $x_e \longleftarrow robot.fkm()$
2: Computa pose diferencial $\Delta x_e \longleftarrow x_e \ominus x_e^*$
3: Computa velocidade de junta $\dot{q} \longleftarrow J(q)^\dagger \Delta x_e$

em que q_t é a configuração de junta do robô no tempo t, dada pela função "robot.q". Ademais, G(t) é um ganho dado por:

$$G(t) = ln(t + 10), \text{onde } t = 0, 1, \ldots, M-1. \tag{4.5}$$

Assim, o processo é mais explorativo no início, porque o valor do ganho é pequeno e com o passar do tempo, o manipulador tende a ir mais para a posição desejada. Com a introdução da cinemática inversa, apenas se modifica o passo 9 do algoritmo 1 para que o ambiente passe a seguir a equação 4.4. Dessa forma, o diagrama esquemático 4.3, baseado em Zhong et al. [2022], representa o processo de aprendizado do manipulador robótico.

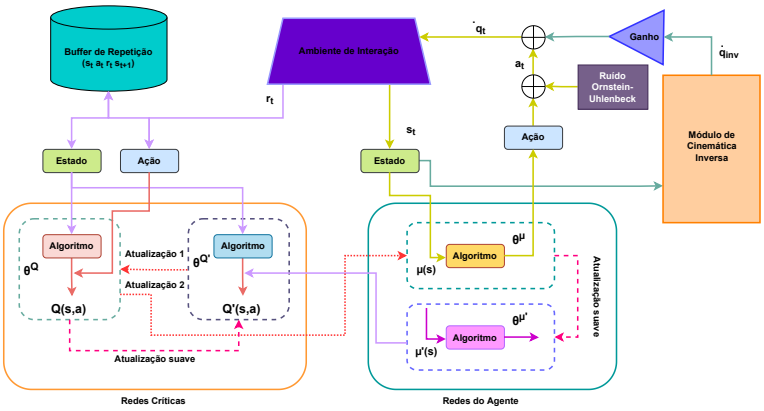

Figura 4.3: Diagrama esquemático do processo de aprendizado do manipulador robótico, baseado em Zhong et al. [2022].

4.4 Resultados

A partir de uma série de testes, chegou-se nos parâmetros mostrados na tabela 4.4 para o agente do DDPG.

Parâmetros	Valor
Número de episódios (N)	100
Fator de desconto (γ)	0,99
Fator de atualização suave (τ)	0.005
Tamanho de buffer	50.000

Tabela 4.2: Parâmetros obtidos através de uma série de testes no simulador UAIBot.

A partir dos algoritmos 1 e 2 e dos parâmetros mostrados na tabela 4.4, obteve-se a curva de recompensa por episódio, mostrada na figura 4.4. Nota-se que a medida que o número de episódios aumenta, o valor da recompensa aumenta drasticamente. Isso se deve ao fato de o robô explorar mais nos primeiros episódios para descobrir qual ação dará a maior recompensa. É importante ressaltar que a curva mostrada se relaciona a posição desejada mostrada na figura 4.1.

As imagens 4.5 e 4.6 mostram que o manipulador conseguiu alcançar a posição desejada com um erro inferior a 1 mm. A imagem 4.7 mostra o tempo necessário para que o manipulador alcançasse essa posição desejada no último episódio. Nesta figura também é interessante perceber que o robô demora muito mais para convergir

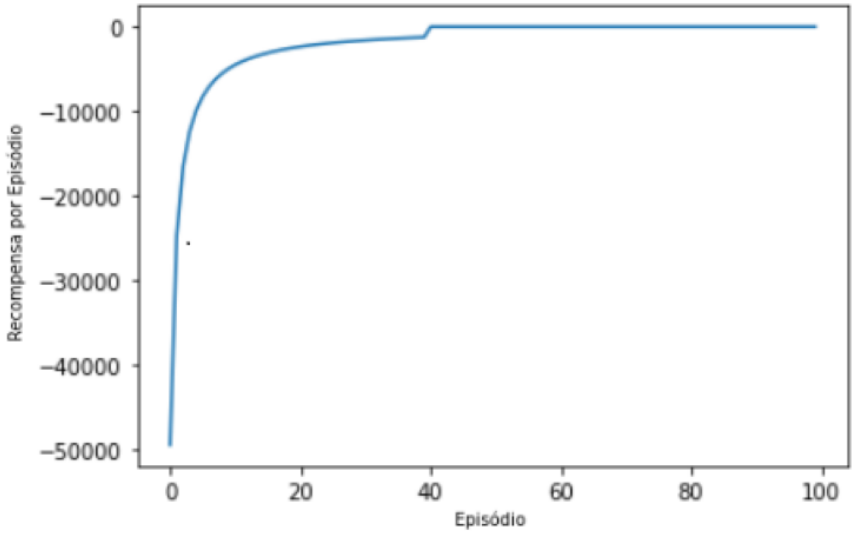

Figura 4.4: Curva de recompensa por episódio.

nos 2 cm finais. Isso porque, as variações de recompensa nos centímetros finais são muito pequenas, já que, elas se relacionam com a distância entre o efetuador e a posição desejada ao quadrado. Dessa forma, o manipulador tem mais dificuldade para encontrar a melhor ação para maximizar a recompensa.

Deve-se ressaltar que o otimizador Adam foi necessário durante a execução do projeto. Isso porque, a atualização das redes crítica e do agente estava sendo computacionalmente cara e com isso, o projeto com os mesmos parâmetros definidos na tabela 4.4 demorava cerca de 10 vezes mais para convergir. Vale destacar que como o projeto utiliza ruído branco, não é possível mensurar de forma precisa o tempo de execução com e sem otimização. Em outras palavras, a cada interação, o resultado e o tempo de simulação são diferentes. Entretanto, foi possível notar de forma prática os dados citados anteriormente.

Deve-se destacar também que a simulação foi realizada para diversas posições desejadas para validação do código. Para tanto, as imagens 4.9 e 4.11 representam duas diferentes posições desejadas e as imagens 4.8 e 4.10 representam a convergência do manipulador, respectivamente. Vale ressaltar também que a cinemática inversa garante que o robô convirja mais rapidamente, porém é o aprendizado por reforço que garante que o manipulador não colidirá com os obstáculos.

4.5 Resumo

Este capítulo mostrou a metodologia e o treinamento realizado no UAIBot para cumprir os objetivos. Pode-se perceber que o erro entre a posição desejada e a pose do efetuador

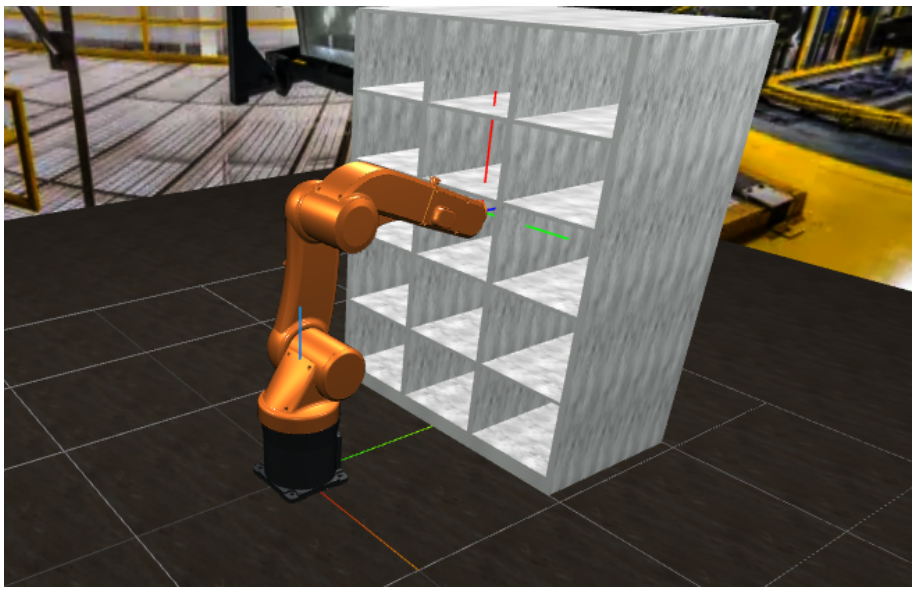

Figura 4.5: Efetuador do UAIBot convergindo para posição desejada 1.

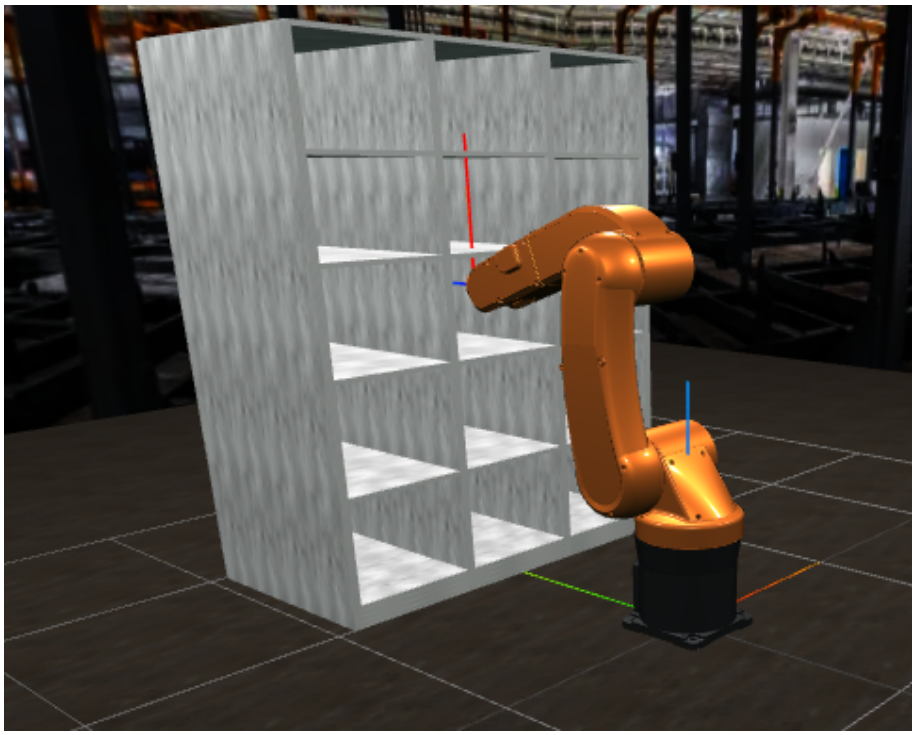

Figura 4.6: Efetuador do UAIBot convergindo para posição desejada 1.

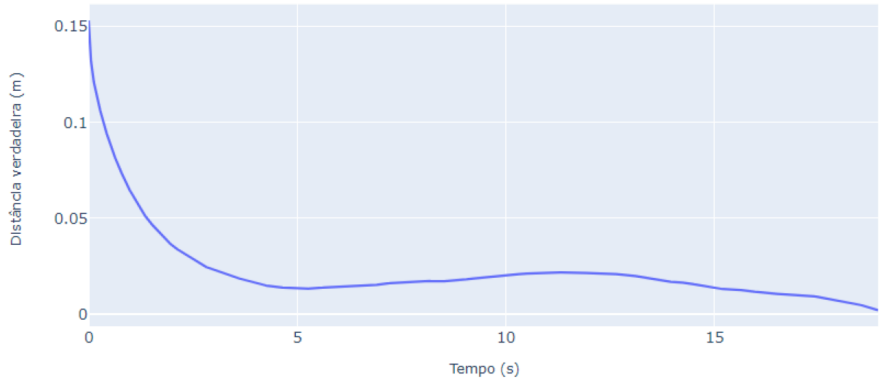

Figura 4.7: Gráfico que mostra o tempo para o robô convergir para a posição desejada no último episódio.

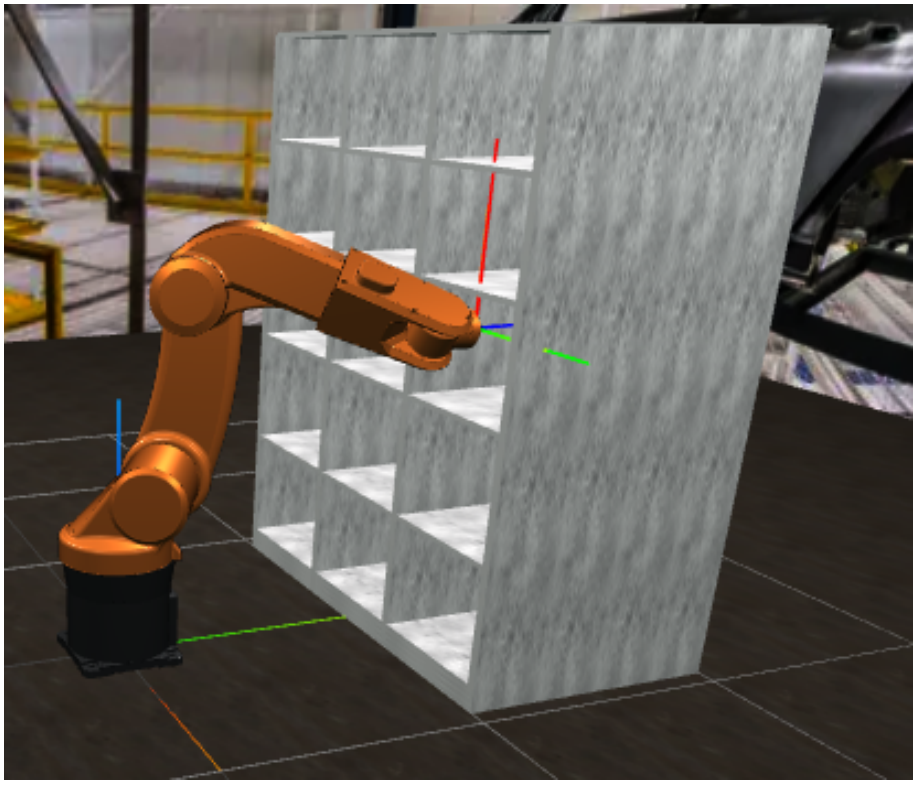

Figura 4.8: Efetuador do UAIBot convergindo para posição desejada 2.

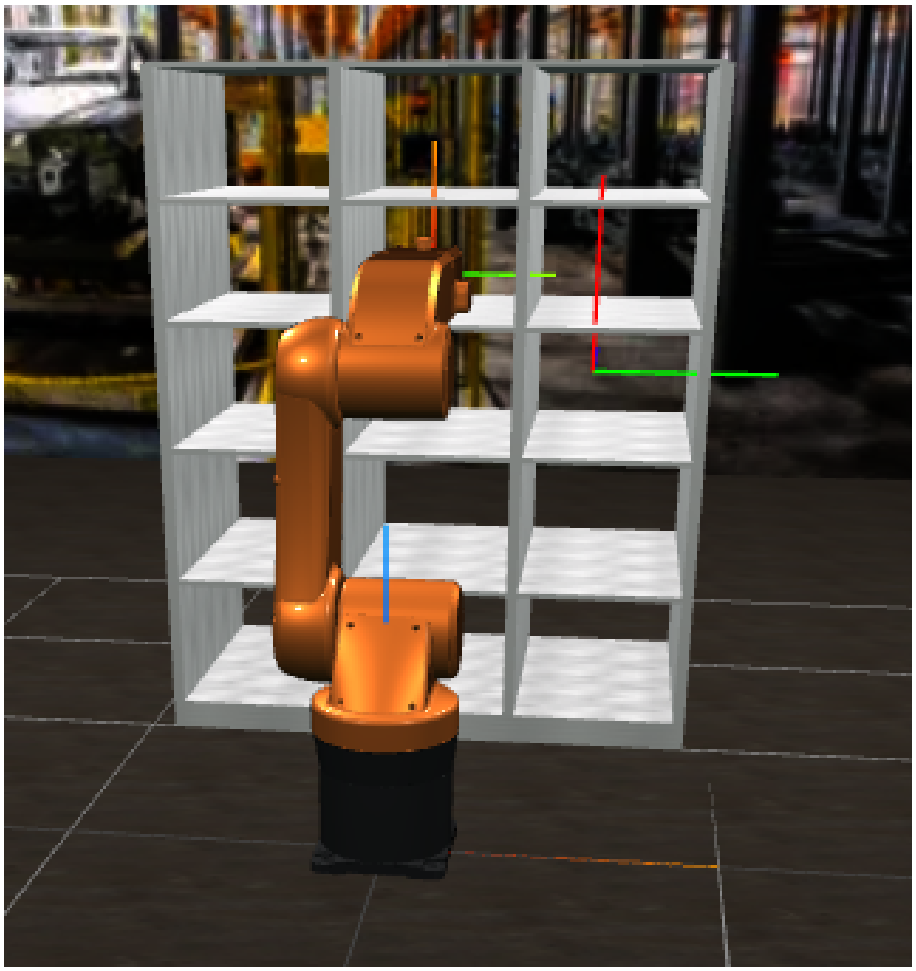

Figura 4.9: Posição inicial do manipulador e posição desejada 2.

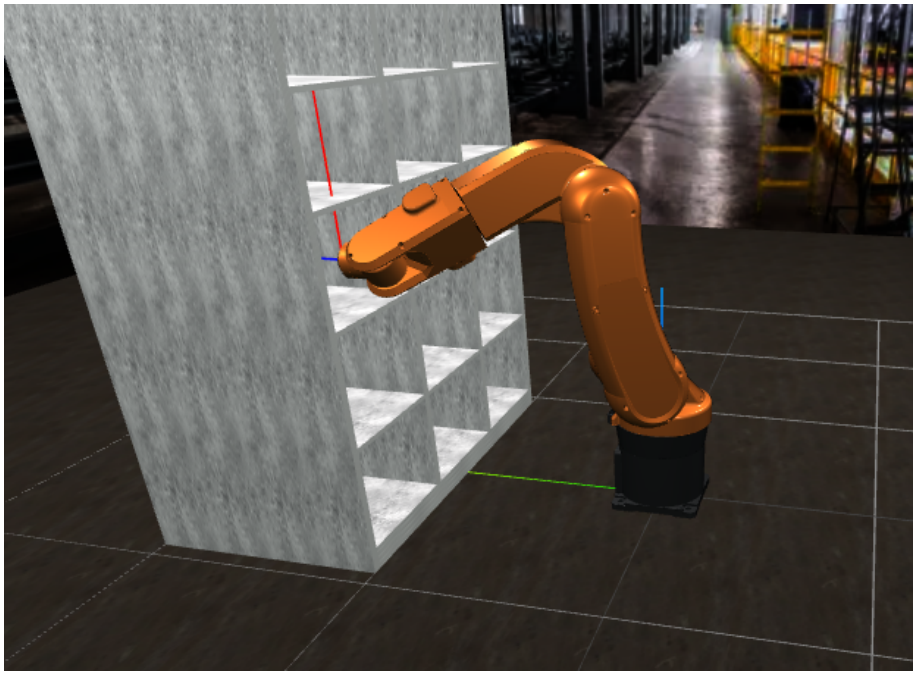

Figura 4.10: Efetuador do UAIBot convergindo para posição desejada 3.

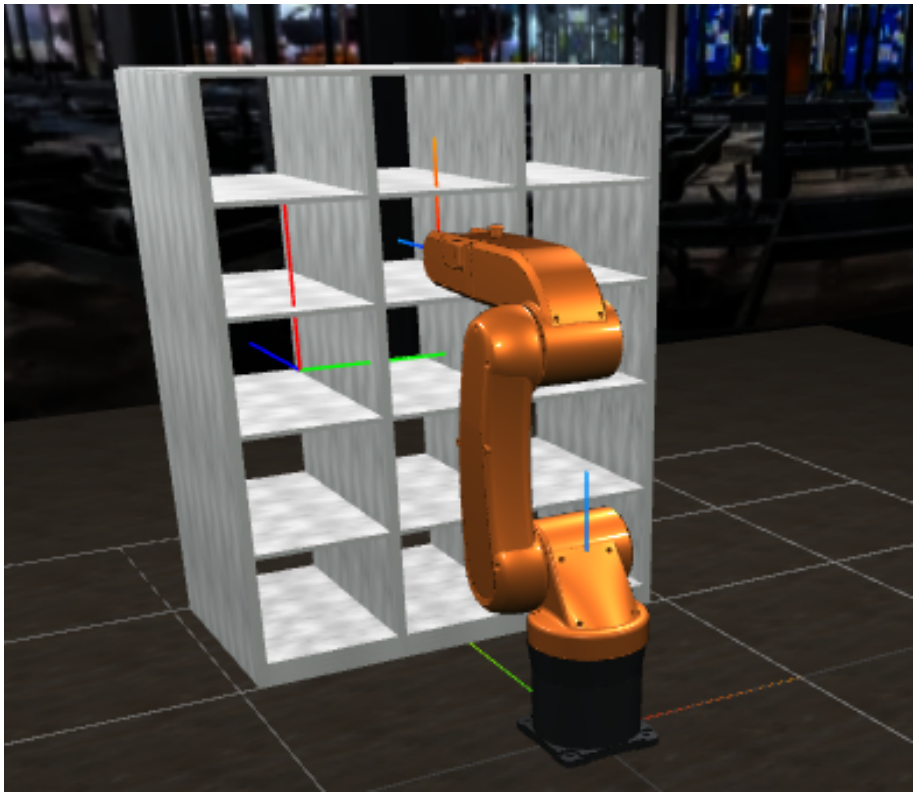

Figura 4.11: Posição inicial do manipulador e posição desejada 3.

do robô possuem um erro bem pequeno e portanto, pode-se considerar que se obteve êxito na realização do projeto.

Capítulo 5

Conclusões

5.1 Considerações Finais

Na presente monografia, utilizou-se de Gradientes de Política Determinística Profundos (DDPG) e cinemática inversa para planejar um caminho factível para o manipulador UAIBot. Para que o robô explorasse o ambiente sem que fosse extremamente custoso em termos computacionais, se utilizou da cinemática inversa para que o robô convergisse para a posição desejada mais rapidamente.

Pode-se notar que se obteve êxito na resolução da problemática, haja vista que o erro encontrado entre a posição desejada e a pose do efetuador foi inferior a 1 mm. Ademais, nota-se que, pela escolha da função recompensa, quanto menor for a distância entre o efetuador e a posição desejada, mais difícil é para o agente definir qual a melhor ação baseada na maior recompensa.

Percebe-se também, que houve a maximização da recompensa com o passar dos episódios, o que indica o funcionamento correto da rede neural. Sendo que, para o planejamento da trajetória preocupou-se em respeitar os limites de variação do movimento e da velocidade angular para que fossem cumpridos todos os objetivos propostos.

Por último, deve enfatizar que criou-se um algoritmo genérico. Isto é, pode-se definir diferentes posições desejadas dentre aquelas factíveis de serem realizadas pelo robô que só será preciso a mudança da respectiva variável.

5.2 Trabalhos Futuros

A partir do trabalho desenvolvido na monografia, pode-se propor os seguintes trabalhos futuros:

- Pode-se utilizar diferentes manipuladores robóticos presentes no UAIBot e veri-

ficar como eles reagem ao algoritmo implementado.

- Implementar o código de DDPG utilizando a API pytorch e comparar o desempenho entre ela e a utilizada na presente monografia.
- Pode-se testar diferentes funções recompensas para avaliar tempos de convergência.

Referências Bibliográficas

K. R. GmbH. *KR5 sixx R650, R850 Specification.* Thorlabs.

N. Ketkar. Introduction to keras. In *Deep learning with Python*, pages 97–111. Springer, 2017.

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

H. Nguyen and H. La. Review of deep reinforcement learning for robot manipulation. In *2019 Third IEEE International Conference on Robotic Computing (IRC)*, pages 590–595. IEEE, 2019.

C. J. Paredis and P. K. Khosla. Kinematic design of serial link manipulators from task specifications. *The international journal of robotics research*, 12(3):274–287, 1993.

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

R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction.* MIT press, 2018.

J. Zhong, T. Wang, and L. Cheng. Collision-free path planning for welding manipulator via hybrid algorithm of deep reinforcement learning and inverse kinematics. *Complex & Intelligent Systems*, 8(3):1899–1912, 2022.