REVISTA Mundi
Engenharia, Tecnologia e Gestão
INSTITUTO FEDERAL Paraná

GERAÇÃO DE TRAJETÓRIAS DE MANIPULADORES ROBÓTICOS USANDO *DEEP REINFORCEMENT LEARNING*

TRAJECTORY GENERATION OF ROBOTIC MANIPULATORS USING DEEP REINFORCEMENT LEARNING

Jaqueline S. Machida[1] Fabian A. Lara-Molina [2] Edson H. Koroishi [3]

Resumo: Manipuladores robóticos são utilizados para automatizar processos industriais poupando mão de obra humana em ambientes de trabalho insalubres e em tarefas repetitivas. Abordagens tradicionais para a geração de trajetórias são usadas em ambientes estruturados que limitam-se principalmente a aplicações indústrias. No entanto, estas abordagens não permitem que o manipulador opere em ambientes dinâmicos nem interação humano-robô. Este trabalho propõe uma metodologia para a geração de trajetórias do manipulador robótico *OpenMANIPULATOR-X* em uma aplicação *reaching* mediante técnicas de inteligência artificial. Esta metodologia é baseada em *Deep Reinforcement Learning*, sendo o trabalho dividido em duas etapas: implementação do algoritmo *Soft Actor-Critic* combinado com *Hindsight Experience Replay* e a aplicação deste algoritmo ao *OpenMANIPULATOR-X* usando *Robot Operating System*. Os resultados demonstraram que a metodologia proposta é promissora para o controle de manipuladores robóticos em aplicações mais complexas.

***Palavras-chave*:** Robótica. Aprendizado por reforço. Manipuladores. Inteligencia Artificial. *Robot Operating System*

Abstract: Robotic manipulators are applied in automatizing industrial processes; they minimize the human working in hazardous and dirty environments or repetitive tasks. Classical approaches for trajectory generation are used in structured environments limited to industrial applications. Nevertheless, these approaches do not permit the operation of dynamic enviroments and human-robot cooperation. The present research study proposes a methodology for the trajectory generation of the robotic manipulator OpenMANIPULATOR-X in a reaching application using Robot Operating System (ROS). This methodology is based on Deep Reinforcement Learning (DRL); the manuscript is split into two stages: first, the implementation of the Soft Actor-Critic algorithm combined with the Hindsight Experience Replay and the application of this algorithm to the OpenMANIPULATOR-X using Robot Operating System. The results show that the proposed methodology has potential advantages in controlling robotic manipulators applied to complex tasks.

***Keywords*:** Robotics. Deep Reinforcement Learning. Manipulator. Artifitial Inteligence. Robot Operating System.

[1]Eng., Universidade Tecnológica Federal do Paraná, jaquelinemachida@alunos.utfpr.edu.br
[2]Dr., Universidade Federal do Triângulo Mineiro, fabian.molina@uftm.edu.br
[3]Dr., Universidade Tecnológica Federal do Paraná, edsonh@utfpr.edu.br

1 Introdução

Manipuladores robóticos são amplamente empregados na indústria para automatizar processos, poupando humanos de trabalhos repetitivos, árduos e perigosos. Na abordagem tradicional de controle desses robôs (COSTA et al., 2018; LARA-MOLINA; DUMUR, 2021), um bloco gerador de trajetória fornece as trajetórias de referência das juntas. As tarefas a serem executadas são especificadas mediante trajetórias que são movimentos que cada uma das juntas devem realizar de forma coordenada.

A abordagem clássica para a geração de trajetórias interpola uma sequência de posições no espaço de trabalho especificados arbitrariamente para realizar uma determinada tarefa; este procedimento não é automatizado e deve ser realizado para cada tarefa (LARA-MOLINA et al., 2014). Face a esta limitação, a abordagem que usa inteligência artificial consegue gerar uma trajetória para que o manipulador desempenhe um determinado movimento sem intervenção de um operador humano. A vantagem potencial da utilização de técnicas de inteligência artificial no planejamento de trajetórias de manipuladores consiste em operar o manipulador em tempo real, o que é ideal para o ambiente dinâmico, onde o manipulador pode ter de lidar com diferentes objetos e dividir espaço com outro manipulador ou operar com a presença de pessoas. Dessa forma, o emprego de inteligência artificial no controle de manipuladores robóticos representa um avanço na automatização de processos industriais. Adicionalmente, está abordagem permite a utilização de sensores mais baratos, uma vez que a estratégia foca em desenvolver algoritmos robustos (LIU et al., 2021).

Diversos trabalhos têm abordado o planejamento dos movimentos dos manipuladores usando inteligência artificial. Em Staley, Katyal e Burlina (2018), usou o agente *Deep Q-Network* para controlar 4 juntas do manipulador Jaco com uma câmera Intel Realsense e *ROS* para a comunicação entre o agente e o Gazebo. A função recompensa norteia o treinamento dos agentes. Xie et al. (2019) realizam experimentos com diferentes funções de recompensa, modeladas para o problema de planejamento de trajetória de um manipulador robótico em um ambiente com obstáculos entre o robô e o objeto alvo. Em Breyer et al. (2018), várias técnicas foram combinadas e aplicadas no robô YuMi da ABB. Nesta abordagem, uma política que tem como saída a posição do efetuador final mediante do algoritmo *Trust Region Policy Optimization*. Outro trabalho que fez uso do *HER* é Prianto et al. (2020). Foi comparada a performance de *SAC*, *TD3* e *Probabilistic Road Map* (um método baseado no algoritmo Dijkistra), todos combinados com *HER* (*Hindsight Experience Replay*). Os autores treinaram os agentes para controlar dois manipuladores *OMX* simultaneamente, evitando que eles colidissem entre si e com um obstáculo, enquanto alcançavam posições no espaço. *Soft Actor-Critic* superou os demais métodos planejando trajetórias mais curtas e suaves do que seus

concorrentes.

O *Robot Operating System (ROS)* tem sido usado na implementação experimental do planejamento de trajetórias usando *DRL*. Gomes et al. (2021) fez uso do *ROS* para conectar manipulador, câmera, agente e ambiente de simulação. Nesse trabalho, o agente foi treinado para agarrar objetos (*grasping*) de diferentes formatos e cores sobre uma mesa cuja cor também foi variada. Outros exemplos de trabalhos que fizeram uso do *ROS* são Zhang et al. (2015), Staley, Katyal e Burlina (2018), Vermedal (2019) e Liu et al. (2020).

O *Soft Actor-Critic (SAC)* é um algoritmo de *DRL* que busca maximizar tanto a recompensa a longo prazo quanto a entropia, uma medida de quão aleatória é a ação do agente (HAARNOJA et al., 2018a). Trata-se de um método *model-free*, ou seja, ele busca aproximar a função valor, que recebe como entrada um estado e retorna a recompensa total esperada quando o agente parte desse estado. Uma vez otimizada, essa função permite obter as ações ótimas para um determinado estado e, portanto, a política ótima. Métodos *model-free* possuem duas vantagens: são fáceis de implementar e de otimizar seus hiperparâmetros. Comparado com outros algoritmos, o *SAC* é mais eficiente em termos de dados, mais estável e confiável. *Soft Actor-Critic* também apresenta vantagens na exploração adequada do ambiente: é possível balancear *exploration* e *exploitation* através do parâmetro temperatura. Em outras palavras, é possível regular a proporção entre comportamento estocástico e determinístico (NGUYEN; LA, 2019).

Os trabalhos apresentados anteriormente, mostraram a implementação de diversas técnicas de *DRL* em manipuladores. No entanto, estes trabalhos não mostram aspectos relacionados à implementação do *SAC* usando *Robot Operating System* em manipuladores robóticos e a integração do agente treinado com o manipulador. Este trabalho tem como objetivo propor uma metodologia baseada em *Deep Reinforcement Learning* para controlar um manipulador robótico. Para esse fim, escolheu-se trabalhar com o manipulador *OpenMANIPULATOR-X*. A metodologia, baseada nos algoritmos *Soft-Actor Critic* e *Hindsight Experience Replay*, foi desenvolvida para controlar o robô em uma aplicação *reaching*. Finalmente, avaliou-se o comportamento do manipulador no *Robot Operating System* com a metodologia proposta.

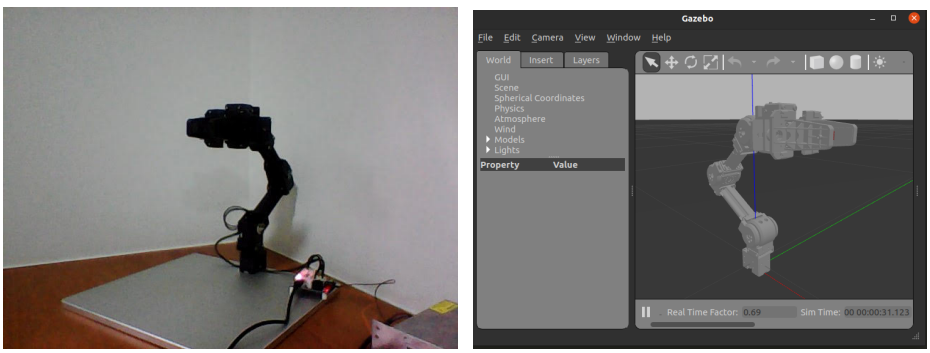
O artigo é divido em quatro seções. Seção 2 apresenta a metodologia baseada no DRL para o planejamento da trajetória. A seção 3 mostra os resultados obtidos decorrentes da aplicação do DRL no planejamento da trajetória do *OpenMANIPULATOR-X*. Finalmente as conclusões obtidas e propostas para dar continuidade aos trabalhos são mostradas na seção 4.

2 Metologia

Neste capítulo são detalhados os procedimentos e recursos usados no trabalho, os quais foram divididos em três etapas: *OpenMANIPULATOR-X* no *Robot Operating System*; implementação e treinamento do agente de *Deep Reinforcement Learning*; e integração do agente treinado com o manipulador no *ROS*.

2.1 *OpenMANIPULATOR-X*

O manipulador escolhido por este trabalho é o *OpenMANIPULATOR-X* (*chain*) desenvolvido pela ROBOTIS (2022). Trata-se de um braço articulado com 4 graus de liberdade, ou 5 incluindo o efetuador final. O manipulador possui carga útil de 500 g, repetibilidade menor que 0,2 mm, pesa 700 g e possui um alcance de 380 mm. A Fig. 1 mostra o *OpenMANIPULATOR-X*: especificamente o protótipo (na Fig. 1a) e a simulação no *Gazebo* (na Fig. 1b) que pode ser realizada mediante o *ROS*.



(a) Protótipo. (b) Simulação no *Gazebo*.

Figura 1: *OpenMANIPULATOR-X*: Protótipo e simulação no *Gazebo*.

O *software* também é aberto e baseado no *Robot Operating System* (*ROS*). Ele abrange pacotes de controle, de operação e de simulação. Como citado anteriormente, o *OpenMANIPULATOR-X* é baseado em *ROS*. O *Robot Operating System* é um meta-sistema operacional: ele precisa ser executado sobre um sistema operacional e permite a comunicação entre diferentes computadores e dispositivos. O *framework* conta com bibliotecas e ferramentas criadas pela comunidade e facilitam o desenvolvimento de *softwares* para robôs. Há ainda programas para simulação como *Gazebo* e *MoveIt*, e para visualização de dados, como Rviz (ROS, 2022). A comunicação no *ROS* ocorre da seguinte maneira: cada programa executado em paralelo constitui um nó e esses nós trocam informações. As informações trocadas são chamadas de mensagens e podem ser do tipo tópico ou ação.

2.2 Implementação e treinamento do agente de *Deep Reinforcement Learning*

A inteligência artificial atua como controlador e é encarregada de gerar a trajetória do manipulador para alcançar alvos posicionados no espaço de trabalho. Note que o manipulador escolhido não possui graus de liberdade suficientes para orientar completamente o efetuador final, por isso apenas a posição do alvo foi considerada.

Tendo isso em vista, são aplicados os conceitos de *Deep Reinforcement Learning* (*DRL*) (HAARNOJA et al., 2018a) e, mais especificamente, dos algoritmos *Soft Actor-Critic* e *Hindsight Experience Replay* (ANDRYCHOWICZ et al., 2017). Estes algoritmos permitem implementar o agente e todo o ambiente de treinamento, *Python* e não envolveram o *ROS* conforme os métodos apresentados na seção 2.2.3. Optamos por treinar o agente com o manipulador simulado para assegurar a integridade física desse. A programação do ambiente fez uso do equacionamento da cinemática direta do *OpenMANIPULATOR-X*. A próxima e última tarefa foi realizar o treinamento do agente no ambiente desenvolvido.

2.2.1 *Soft Actor-Critic*

O problema tratado envolve espaço de ação contínuo, porque as ações representam valores contínuos dos ângulo das juntas. De acordo com Haarnoja et al. (2018b), no *SAC* a política do agente é estocástica e otimizada para maximizar tanto a recompensa acumulada, recebida pelo ambiente, quanto a entropia H. Quanto maior a entropia, mais aleatórias são as ações do agente. O parâmetro que pondera recompensa acumulada e entropia é a temperatura α e o termo *soft* refere-se a essa regularização. Dessa forma, é possível equilibrar a exploração em profundidade e em largura do agente, fundamental para que esse encontre soluções eficientes.

A função de recompensa da Eq. (1) deve ser maximizada. Ela representa o acúmulo do valor esperado de recompensa e de entropia, para todos os passos. A Eq. (2) traz a definição de entropia (PRIANTO et al., 2020). Note que maximizar a entropia incentiva a escolha de ações menos prováveis graças ao logaritmo na expressão.

$$J(\pi) = \sum_{t=0}^{T} E_{s_t,a_t}[r(s_t, a_t) + \alpha H(\pi(a_t|s_t)] \tag{1}$$

$$H(\pi(a_t|s_t)) = -\sum_{a_t} \pi(a_t|s_t) log \pi(a_t|s_t) \tag{2}$$

O termo *actor-critic* refere-se ao emprego de redes neurais para aproximarem a política (*actor*) e as funções valor (*critic*) (HAARNOJA et al., 2018b). Essas funções valor buscam prever a recompensa que será recebida do ambiente (LAVET et al., 2018). Dessa forma, elas servem para avaliar o estado ou o par estado e ação.

O agente em *SAC* possui cinco redes neurais: a rede *value* aproxima a função *soft value* (Eq. (3)); a segunda rede é usada como rede *target* da primeira; a terceira e a quarta aproximam duas funções *soft Q-value* (Eq. (4)) e são chamadas de redes *critics*, por fim; a quinta rede, *actor*, aproxima a política. Note que a quantidade de neurônios de entrada das redes *value* e *actor* é igual ao tamanho do vetor estado, enquanto que para a rede *critic* essa quantidade é igual à soma do tamanho do vetor estado e do número de ações. Na Eq. (4), γ representa o fator de desconto. Esse fator serve para garantir que a soma das recompensas e entropias esperadas seja um valor finito (PRIANTO et al., 2020).

$$V(s_t) = E_{a_t \sim \pi}[Q(s_t, a_t) - \alpha log \pi(a_t|s_t)] \tag{3}$$

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma E_{s_{t+1} \sim P}[V(s_{t+1})] \tag{4}$$

A política é representada pela rede *actor* e toma como entrada o vetor estado do ambiente e retorna dois valores: a média μ_ϕ e o desvio padrão θ_ϕ de uma distribuição normal de probabilidades das ações $N(\mu_\phi, \theta_\phi)$. As ações a_t são obtidas sorteando ações a_{t_N} de acordo com as probabilidades e redimensionando-as com a função tangente hiperbólica: $a_t = tanh(a_{t_N})$, onde $a_{t_N} \sim N(\mu_\phi, \theta_\phi)$.

Os pesos das redes são ajustados para minimizar as funções das Eqs. (5), (6) e (7) e a taxa de aprendizado é o hiperparâmetro responsável por definir a velocidade com que esses pesos são ajustados.

$$J_V(\psi) = E_{s_t}[\frac{1}{2}(V_\psi(s_t) - E_{a_t}[min_{k=1,2} Q_{\theta_k}(s_t, a_t) - \alpha log \pi(a_t|s_t)])^2] \tag{5}$$

$$J_Q(\theta_{k=1,2}) = E_{s_t, a_t}[\frac{1}{2}(Q_{\theta_k}(s_t, a_t) - (r(s_t, a_t) + V_{\bar{\psi}}(s_{t+1})))^2] \tag{6}$$

$$J_\pi(\phi) = E_{s_t, a_t}[log \pi_\phi(a_t|s_t) - min_{k=1,2} Q_{\theta_k}(s_t, a_t)] \tag{7}$$

sendo que ψ, θ e ϕ são os pesos das redes que representam as funções *soft value*, *soft Q-value* e a política, respectivamente.

Finalmente, a rede *target* tem seus pesos aproximados para os pesos da rede *online* com um retardo definido pelo parâmetro τ, assim $\bar{\psi} = \tau\psi + (1 - \tau)\bar{\psi}$. Ao decorrer do treinamento, várias políticas são aproximadas conforme os pesos são ajustados incrementalmente. *SAC* é considerado *off-policy* pois dados obtidos com políticas passadas podem ser usados para o treinamento da política atual. Essa característica melhora a eficiência do uso de dados (HAARNOJA et al., 2018b). O tamanho do conjunto de transições sorteado para ser utilizado

no processo de ajuste dos pesos das redes neurais (*RNA*s) é chamado de *Batch size*.

2.2.2 *Hindsight Experience Replay*

Hindsight Experience Replay é uma estratégia de aumento de dados que pode ser combinada com algoritmos de *DRL* como o *SAC*. Ela funciona da seguinte maneira, segundo Andrychowicz et al. (2017), ao final de cada episódio, para cada passo ocorrido, são sorteados estados pelos quais o ambiente passou até então durante o episódio. Para cada um desses estados selecionados, a transição do passo é modificada como se o estado sorteado fosse o alvo do episódio. Essa modificação inclui o cálculo da nova recompensa com base no novo alvo. Essas transições sintéticas são armazenadas para serem usadas no treinamento. Assim, o *HER* permite que o agente aprenda mesmo se o episódio for mal sucedido. Para a aplicação desse trabalho, mesmo que o efetuador final do manipulador não tenha atingido a posição alvo, o agente ainda assim terá aprendido a levar o robô para as posições alcançadas durante a tentativa.

2.2.3 Ambiente e simulação do *OpenMANIPULATOR-X* com *Python*

Neste trabalho, o ambiente foi criado para a aplicação *reaching* que consiste em que o efetuador final deverá alcançar uma determinada posição. O estado adotado é composto pela posição do efetuador final e pela posição do alvo no espaço de trabalho. O estado do ambiente se altera com as ações do agente, que são os valores de incremento/decremento dos ângulos das juntas do manipulador, em radianos. As ações são contidas em um intervalo de valores com o objetivo de limitar a exploração do espaço de ações.

O ambiente foi criado em *Python* e não foi utilizado o *ROS* nesta etapa, a fim de poupar o esforço computacional de integrar *Python* e *ROS*. Para isto, foi criada uma classe em *Python* que simula o manipulador e seu modelo cinemático, usando o método de Denavit-Hartenberg (TING et al., 2021). As equações foram então incorporadas à classe do manipulador.

A recompensa r_t retornada para o agente é em termos da distância euclidiana d entre a posição do efetuador final p_g e a posição do alvo p_a, dada por $d = \sqrt{(x_{p_g} - x_{p_a})^2 + (y_{p_g} - y_{p_a})^2 + (z_{p_g} - z_{p_a})^2}$. A escolha dessa função de recompensa foi baseada em Xie et al. (2019): $r_t = -d^2$, para maximizar a recompensa r_t. A Fig. 2 ilustra a interação entre agente e ambiente no contexto desse trabalho.

Tanto o agente quanto o ambiente e o programa de treinamento foram implementados totalmente em *Python*. A linguagem de programação e suas bibliotecas foram instaladas mediante o Anaconda, uma ferramenta de gerenciamento de bibliotecas (ANACONDA, 2022). As bibliotecas *Tensorflow*, *Numpy*, *Matplotlib* e bibliotecas nativas foram utilizadas neste procedimento.

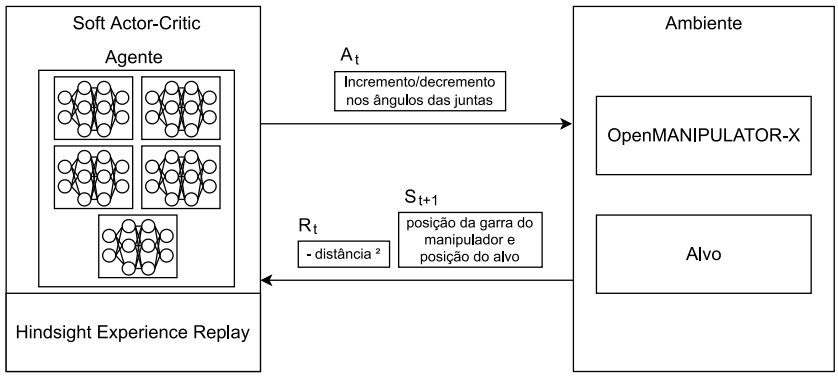


Figura 2: Interação entre agente e ambiente durante o treinamento.

3 Resultados

Os resultados apresentam os dados obtidos do treinamento do manipulador para o planejamento das trajetórias usando o *Soft Actor-Critic*. Adicionalmente, são apresentados resultados da implementação e integração do *OpenMANIPULATOR-X* no *ROS* e sua simulação no *Gazebo*.

3.1 Implementação e Treinamento do Manipulador usando o *Soft Actor-Critic*

Com relação à implementação do *Soft Actor-Critic*, tomou-se como base o código de Tabor (2021). Essa implementação segue o algoritmo apresentado em Haarnoja et al. (2018a), onde é adotado o hiperparâmetro escala de recompensa ao invés da temperatura. Trata-se de uma equivalência, exceto que a relação com a aleatoriedade das ações do agente é invertida: quanto maior a escala, mais peso se atribui ao termo de recompensa (vide Eq. (1)) e mais determinístico se torna o agente (HAARNOJA et al., 2018a).

Outra modificação introduzida na implementação deste trabalho foi a adoção de passos iniciais aleatórios: durante uma determinada quantidade de passos no começo do treinamento, as ações do agente serão ignoradas e em seu lugar serão tomadas ações aleatórias. Essa é uma estratégia adotada em OpenAI (2022) para melhorar a exploração do espaço de ações.

Além disso, foram feitas alterações para incluir o *Hindsight Experience Replay* baseado em Andrychowicz et al. (2017) e integrar o ambiente desenvolvido. Feito isso, iniciou-se o treinamento. Os hiperparâmetros adotados estão contidos na Tabela 1.

Ao decorrer do treinamento foram registrados os dados a seguir: a recompensa total de cada episódio, a quantidade de passos e a taxa de sucesso do agente. Essas métricas podem ser observadas na Fig. 3. O desempenho do algoritmo pôde ser monitorado pela otimização da recompensa total de cada episódio (Fig. 3a), pela redução na quantidade de passos necessários

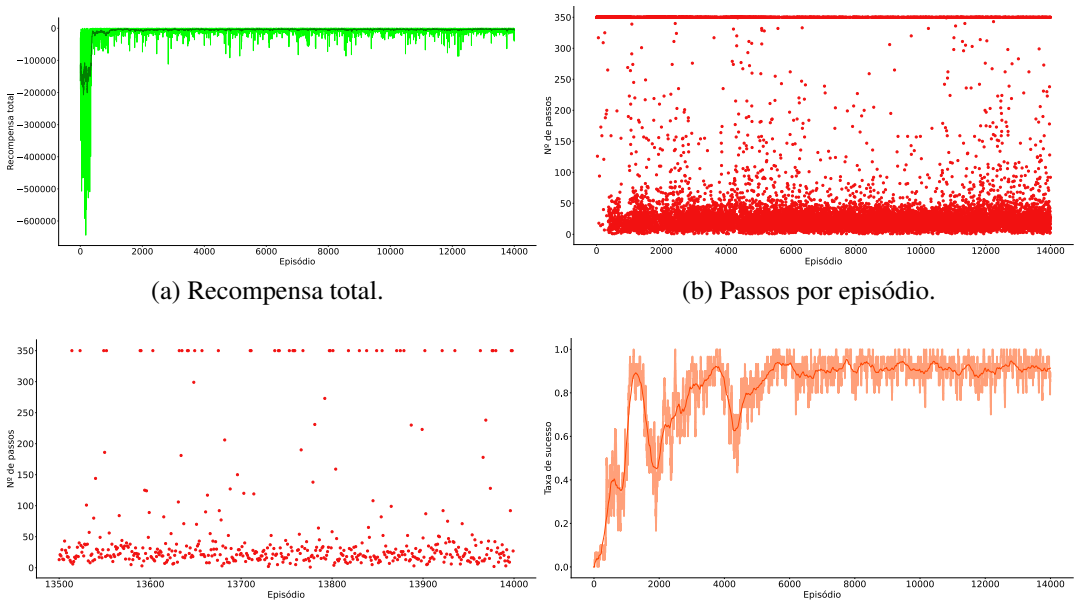
Tabela 1: Hiperparâmetros do treinamento

Hiperparâmetro	**Valor**
Nº de episódios N_E	14.000
Nº de passos por episódio N	350
Nº de passos iniciais aleatórios	350.000
Tamanho do *buffer*	1.000.000
Nº de estados	6
Nº de ações	4
Intervalo de ação	$[-2°, 2°]$
Intervalo de θ_1	$[-45°, 45°]$
Intervalo de θ_2	$[-45°, 0°]$
Intervalo de θ_3	$[-45°, 45°]$
Intervalo de θ_4	$[-45°, 45°]$
Dimensões da rede *critic*	10 x 256 x 256 x 1
Dimensões da rede *value*	6 x 256 x 256 x 1
Dimensões da rede *actor*	6 x 256 x 256 x 4
Margem de erro	1 cm
Batch size	512
Taxa de aprendizado	0,0001
Fator de desconto γ	0,98
Escala de recompensa	0,65
Coeficiente de atualização da rede *target* τ	0,005

(Fig. 3b e Fig. 3c) e pelo crescimento da taxa de sucesso (Fig. 3d). Na recompensa total da Fig. 3a, fica evidente o momento que o agente deixou de agir aleatoriamente a partir do episódio 1000, de acordo com o parâmetro de número de passos iniciais aleatórios (Fig. 1). Após esses episódios iniciais, nota-se que o agente rapidamente otimizou a recompensa acumulada até aproximadamente o episódio 1000. Por outro lado, na quantidade de passos, é possível observar que o agente tomava os 350 passos e não alcançava o objetivo do ambiente no início do treinamento. Na Fig. 3c, os últimos 500 episódios de treinamento são considerados, evidenciando que o efetuador final do manipulador alcançou o alvo para a maioria dos episódios ao final do treino, geralmente levando menos de 50 passos para isso. Por fim, o avanço da taxa de sucesso representa a curva de aprendizado do agente, que convergiu em torno do episódio 6000.

A Fig. 4 mostra a trajetória do efetuador final nos episódios 1, 7000 e 14000, respectivamente. O ponto verde representa o ponto inicial, o ponto vermelho representa o alvo e o ponto preto representa o ponto final da trajetória.

A trajetória do episódio 1 de treinamento (Fig. 4a) ilustra o comportamento de uma política aleatória, enquanto que as demais trajetórias, 7000 episódios (Fig. 4b) e 14000



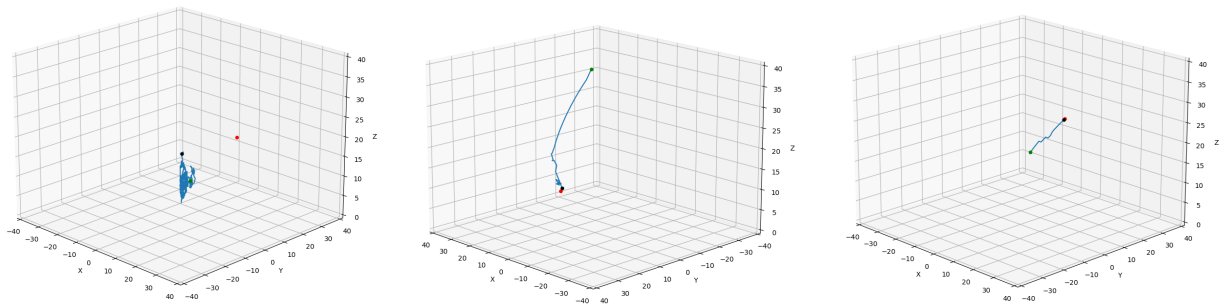
(a) Recompensa total.

(b) Passos por episódio.

(c) Passos dos últimos 500 episódios.

(d) Taxa de sucesso.

Figura 3: Resultados de cada episódio durante treinamento do agente.



(a) episódio 1.

(b) episódio 7000.

(c) episódio 14000.

Figura 4: Trajetórias ao longo dos Episódios.

DOI: 10.21575/25254782rmetg2022vol7n52243

episódios (Fig. 4c) revelam que as trajetórias geradas pelo agente ao longo do treinamento permitem que o manipulador alcance o alvo. Observa-se que no episódio 7000, o agente é capaz de gerar trajetórias objetivas. Um pequeno ruído foi observado apenas na trajetória do episódio 7000, pouco antes do efetuador final atingir o alvo. No entanto, isto foi corrigido para trajetória do episodio 14000 (Fig. 4c).

Este processo de aprendizagem demonstrou que a estratégia foi corretamente implementada; No entanto, a redefinição dos parâmetros de treinamento pode explorar melhor o espaço de ações e consequentemente melhorar o desempenho do algoritmo. Assim, foi treinado um agente para gerar a trajetórias em tempo real para aplicação *reaching* no ambiente programado em *Python*, demonstrando ser passível de integração com o *OpenMANIPULATOR-X* no *ROS* na terceira e última etapa.

3.2 Implementação e Integração com o *OpenMANIPULATOR-X*

Nesta etapa, a integração entre as redes neurais dos agentes treinados e o manipulador foi realizada usando o *ROS*. Para isto, três nós foram desenvolvidos em *Python* para trocaram informações entre si em *ROS*. O nó do agente imediatamente passou a gerar a trajetória em tempo real do manipulador robótico com base no estado observado até o efetuador final alcançar o alvo, evidenciando que a integração foi um sucesso como apresentado na Fig. 5.

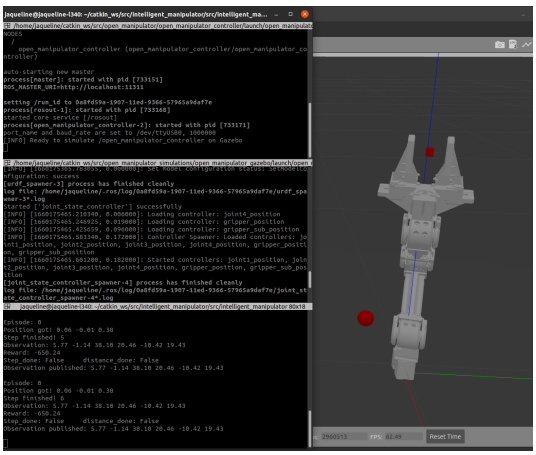


Figura 5: Integração do agente com *OpenMANIPULATOR-X* mediante *ROS* no *Gazebo*.

A comparação entre os resultados de desempenho nos dois ambientes - ambiente de treinamento e ambiente *Gazebo* no *ROS* - (Tabela 2) devido a que antes não havia simulação gráfica apenas um modelo cinemático, diferente de agora que toda a dinâmica e renderização do modelo é incluída na simulação do Gazebo, tanto em termos de número médio de passos quanto de taxa de sucesso, devido a transferência da inteligencia artificial (*IA*) para um ambiente diferente.

Tabela 2: Desempenho do agente nos diferentes ambientes

	Ambiente	
	Python	ROS
Número de passos	60	72
Taxa de sucesso	87,5%	86,7%

Todos os códigos desenvolvidos nesse trabalho, desde aqueles utilizados no treinamento do agente com ambiente em *Python* até a integração do agente com o manipulador no *ROS*, foram disponibilizados no seguinte site <https://github.com/JaquelineSayuri/intelligent_manipulator>. Também é possível encontrar os pesos das redes neurais treinadas.

4 Conclusão

Nesse trabalho foi proposta uma metodologia de controle de manipulador robótico baseada em *Deep Reinforcement Learning* aplicada no manipulador *OpenMANIPULATOR-X* para executar a aplicação *reaching* onde deve alcançar alvos definidos arbitrariamente.

Para cumprir esse objetivo, o trabalho foi dividido em duas etapas. A primeira etapa aborda o *Deep Reinforcement Learning* usando o *Soft Actor-Critic* e *Hindsight Experience Replay* em Python; esta etapa contemplou o treinamento do agente. A segunda e última etapa considerou integrar o agente treinado com o *OpenMANIPULATOR-X* no *Robot Operating System*.

A avaliação da metodologia indicou que a política é capaz de encontrar soluções de trajetória eficientes dentro do ambiente, com possibilidade de melhoria mediante o reajuste dos hiperparâmetros de treinamento do algoritmo. As limitações identificadas dessa abordagem foram o elevado número de hiperparâmetros, a escolha não trivial dos valores desses hiperparâmetros e o longo tempo de treinamento. Logo, foi demonstrado o potencial dessa abordagem no controle de manipuladores robóticos para aplicações mais desafiadoras do que a aplicação *reaching* e que justificam seu uso. Por fim, a metodologia proposta pode ser facilmente adaptada para outros manipuladores. Os conceitos aprendidos com o *OpenMANIPULATOR-X* podem ser estendidos para manipuladores industriais com mais graus de liberdade.

Em trabalhos futuros, pretende-se treinar um agente de *DRL* para gerar trajetórias de um manipulador para aplicações complexas, substituindo o papel do especialista humano. O emprego de uma câmera acoplada ao efetuador final do robô para captar imagens que irão alimentar a *IA* permitirá que esta opere o manipulador em tempo real em ambientes dinâmicos.

Referências

ANACONDA. *Anaconda Distribution*. 2022. Disponível em: <https://www.anaconda.com/products/distribution>. Acesso em: 28 mai. 2022. 7

ANDRYCHOWICZ, M. et al. Hindsight experience replay. *CoRR*, abs/1707.01495, 2017. Disponível em: <http://arxiv.org/abs/1707.01495>. 5, 7, 8

BREYER, M. et al. Flexible robotic grasping with sim-to-real transfer based reinforcement learning. *CoRR*, abs/1803.04996, 2018. Disponível em: <http://arxiv.org/abs/1803.04996>. 2

COSTA, T. L. et al. Robust H_∞ computed torque control for manipulators. *IEEE Latin America Transactions*, IEEE, v. 16, n. 2, p. 398–407, 2018. 2

GOMES, N. M. et al. Deep reinforcement learning applied to a robotic pick-and-place application. In: PEREIRA, A. I. et al. (Ed.). *Optimization, Learning Algorithms and Applications*. Cham: Springer International Publishing, 2021. p. 251–265. ISBN 978-3-030-91885-9. 3

HAARNOJA, T. et al. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *CoRR*, abs/1801.01290, 2018. Disponível em: <http://arxiv.org/abs/1801.01290>. 3, 5, 8

HAARNOJA, T. et al. Soft actor-critic algorithms and applications. *CoRR*, abs/1812.05905, 2018. Disponível em: <http://arxiv.org/abs/1812.05905>. 5, 6

LARA-MOLINA, F. A.; DUMUR, D. A fuzzy approach for the kinematic reliability assessment of robotic manipulators. *Robotica*, Cambridge University Press, v. 39, n. 12, p. 2095–2109, 2021. 2

LARA-MOLINA, F. A. et al. Robust generalized predictive control of the orthoglide robot. *Industrial Robot: An International Journal*, Emerald Group Publishing Limited, 2014. 2

LAVET, V. F. et al. *An Introduction to Deep Reinforcement Learning*. 11. ed. Boston: Foundations and Trends in Machine Learning, 2018. 5

LIU, N. et al. Real-world robot reaching skill learning based on deep reinforcement learning. In: *2020 Chinese Control And Decision Conference (CCDC)*. [S.l.: s.n.], 2020. p. 4780–4784. 3

LIU, R. et al. Deep reinforcement learning for the control of robotic manipulation: A focussed mini-review. *CoRR*, abs/2102.04148, 2021. Disponível em: <https://arxiv.org/abs/2102.04148>. 2

NGUYEN, H.; LA, H. Review of deep reinforcement learning for robot manipulation. In: *2019 Third IEEE International Conference on Robotic Computing (IRC)*. [S.l.: s.n.], 2019. p. 590–595. 3

OPENAI. 2022. Disponível em: <https://spinningup.openai.com/en/latest/algorithms/sac.html>. Acesso em: 7 ago. 2022. 8

PRIANTO, E. et al. Path planning for multi-arm manipulators using deep reinforcement learning: Soft actor–critic with hindsight experience replay. *Sensors*, v. 20, n. 20, 2020. ISSN 1424-8220. Disponível em: <https://www.mdpi.com/1424-8220/20/20/5911>. 2, 5, 6

ROBOTIS. *OpenMANIPULATOR-X*. 2022. Disponível em: <https://emanual.robotis.com/docs/en/platform/openmanipulator_x/overview/>. Acesso em: 23 mai. 2022. 4

ROS. 2022. Disponível em: <http://wiki.ros.org/>. Acesso em: 24 mai. 2022. 4

STALEY, E. W.; KATYAL, K. D.; BURLINA, P. Drl based intelligent joint manipulator and viewing camera control for reaching tasks and environments with obstacles and occluders. In: *2018 International Joint Conference on Neural Networks (IJCNN)*. [S.l.: s.n.], 2018. p. 1–7. 2, 3

TABOR, P. *Machine Learning with Phil*. 2021. Disponível em: <https://www.youtube.com/watch?v=YKhkTOU0l20>. Acesso em: 3 ago. 2022. 8

TING, H. Z. et al. Kinematic analysis for trajectory planning of open-source 4-dof robot arm. *Int. J. Adv. Comput. Sci. Appl.*, v. 12, n. 6, p. 769–777, 2021. 7

VERMEDAL, A. *Deep Reinforcement Learning for Valve Manipulation*. Junho 2019. 69 f. Dissertação (Mestrado em Tecnologia – Cibernética e Robótica) — Norwegian University of Science and Technology, Noruega, 2019. 3

XIE, J. et al. Deep reinforcement learning with optimized reward functions for robotic trajectory planning. *IEEE Access*, v. 7, p. 105669–105679, 2019. 2, 7

ZHANG, F. et al. Towards vision-based deep reinforcement learning for robotic motion control. *CoRR*, abs/1511.03791, 2015. Disponível em: <http://arxiv.org/abs/1511.03791>. 3

Enviado em: 06 out. 2022.

Aceito em: 26 dez. 2022.

Editor responsável: Mateus das Neves Gomes