CONTROL OF INDUSTRIAL MANIPULATORS THROUGH REINFORCEMENT LEARNING: A STUDY OF THE PANDA MANIPULATOR IN A SIMULATED ENVIRONMENT

Marcelo Albergaria Paulino Fernandes Ferreira[1],Taniel Silva Franklin [2],Oberdan Rocha Pinheiro [2]
[1] Bolsista Big Data - IA; Centro Universitário SENAI CIMATEC; marcelo.ferreira@fbter.org.br;
[2] Centro Universitário SENAI CIMATEC; Salvador-BA; {taniel.franklin, oberdan.pinheiro}@fieb.org.br;

Abstract: The wide variety of scenarios in industrial environments requires intelligent robotics capable of directly interacting with the environment for problem-solving. Through reinforcement learning, robots can quickly adapt to new situations and learn from direct interaction with the environment. This work proposes a simulation environment based on Robotics Toolbox for Python to solve a classic problem of the inverse kinematics of manipulators, ensuring that the robot reaches the desired position without colliding with the obstacles present in the scene. The potential of this reinforcement learning method is illustrated through simulation using the Franka-Emika Panda manipulator trained by the Deep Deterministic Policy Gradient algorithm.

Keywords: deep reinforcement learning; robot; manipulator; machine learning; artificial intelligence.

CONTROLE DE MANIPULADORES INDUSTRIAIS ATRAVES DO APRENDIZADO POR REFORÇO: ESTUDO DO MANIPULADOR PANDA EM AMBIENTE SIMULADO

Resumo: A ampla variedade de cenários em ambientes industriais requer uma robótica inteligente capaz de interagir diretamente com o ambiente para soluções de problemas. Através do aprendizado por reforço, os robôs podem adaptar-se rapidamente a novas situações e aprender com a interação direta com o ambiente. Este trabalho propõe um ambiente de simulação baseado em *Robotics Toolbox for Python* com o objetivo de resolver um problema clássico da cinemática inversa de manipuladores, garantindo que o robô alcance a posição desejada sem colidir com os obstáculos presentes na cena. O potencial desse método de aprendizado por reforço é ilustrado através de simulação utilizando o manipulador Franka-Emika Panda treinado pelo algoritmo *Deep Deterministic Policy Gradient.*

Palavras-chave: Aprendizado Profundo; Robótica; Aprendizado por Reforço; Manipulador;

1. INTRODUCTION

One of the main challenges in robotics is to create agents able to directly interact with the world around them and achieve their goals. The wide variety of deployment scenarios and environmental variations suggest that an effective manipulator robot must be able to move in environments that neither it nor its designers anticipated or encountered before [1].

Deep Reinforcement Learning (DRL) is the combination of Reinforcement Learning (RL) and Deep Learning (DL). Reinforcement Learning is a machine learning method, where the agent learns the ideal behavior in an environment through trial and error interactions to obtain the maximum reward, the agent acts complete complex tasks, interacting with the environment that responds with observations or states, and rewards or costs [2]. These two components continuously interact so that the agent tries to influence the Environment, at each interaction step the agent receives an observation of the state of the world, and then decides which action to take, for each action it receives a reward signal from the environment. This reward needs to inform how successful the action was on the way to achieving the goal, so maximizing the total reward will lead the agent to solve the problem by taking the best actions.

RL has emerged as a promising approach in several domains, including recommendation systems, robotics, and video games. In the field of robotic manipulation control, RL has shown remarkable progress in recent years. [2] provides an overview of recent advances made in deep reinforcement learning algorithms specifically tailored for robotic manipulation control tasks and discusses challenges that still need to be addressed for real-world applications.

With the advent of powerful computers and an abundance of data, it is possible to train models to generalize based on raw inputs such as images, text, and voice, similar to the way that humans learn [3]. Deep Learning is a subfield of machine learning that deals with algorithms inspired by the human brain to process a large amount of data using layers of stacked neurons. In this work, Deep Neural Networks will be used to model the decision-making mechanism of agents.

Commercial and industrial robots nowadays have a wide range of applications. They often help humans in dangerous environments or repetitive and exhaustive tasks. Many extreme environments are challenging to access or require expensive logistics to transport specialist personnel. The oil and gas industries need autonomous robots to complete manipulation tasks on their assets.

Training with a real robot is expensive because machine learning requires a considerable amount of data that represents the robot's experience in the environment. Researchers have sought ways to reduce this dependency through training approaches simulation-inspired due to the low risk to equipment parts and the availability of synthetic data acquired using many simulations. However, this approach needs to include strategies to minimize the reality gap. In addition, Furthermore, the quality of the generated data depends on the quality of the simulator and its speed.

This project aims to solve the problem of inverse kinematics using a simplified environment built with Robotics Toolbox for Python [4]. This Framework provides specific robotics functionalities to represent the kinematics and dynamics of rigid and

mobile manipulators, making it possible to import a URDF file or use more than 30 models provided, in addition to enabling the creation of obstacles.

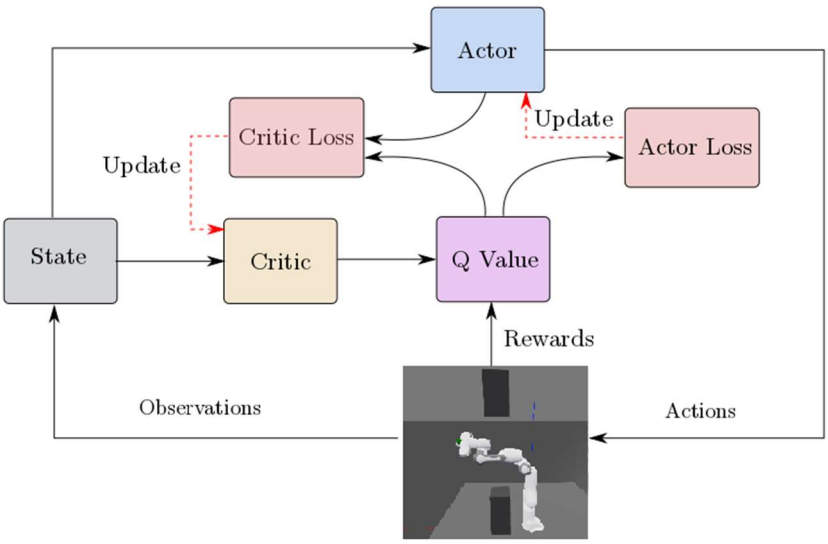
In this context, it is useful to present a reinforcement learning environment for training robot manipulators in Python. The simulator is based on Robotics Toolbox for Python. The environment provides training and visualization modules to compare standard DRL algorithms such as the Deep Deterministic Policy Gradient (DDPG) algorithm. The project is open source and all the code can be found on the GitHub repository.

1.1. Reinforcement Learning Tasks

The essential elements of this technique are the agent, the environment, the action, and the reward. The agent aims to perform a task, executing actions. The environment can be a virtual world where the current state *(s)* and the agent's action *(a)* are observed, returning the agent's reward *(r)* as output [1]. The action is the set of all actions that can be performed by the agent, and the reward is the feedback by which the success or failure of an agent's actions is measured in a given state.

This article is based on the DDPG (Deep Deterministic Policy Gradient) algorithm, a reinforcement learning approach that uses direct agent modeling to estimate the reward function through real interactions with the environment. This allows the agent to learn the optimal policy without relying on an explicit model of the environment. The schema of the algorithm is illustrated in Figure 1.

Figure 1. DDPG schema



The DDPG algorithm uses deep neural networks to learn deterministic policies, the agent consists of two neural networks, actor and critic. The actor tries to approximate the best policy that maps a state to the optimal action, while the critical network estimates the Q-values of the state-action pairs. There are also Target networks, which are copies of each of the networks and aim to provide more stable and fixed targets for updating the main networks. These networks are updated using a

technique called soft update, which uses the interpolation parameter (tau) to combine the weights of the main and target networks, ensuring that the updates are not sudden and help to control the training convergence [5].

During training the agent interacts with the environment using the current policy. At each step, the agent performs an action chosen in the environment and observes the reward received and the next resulting state, storing these pairs in a replay buffer. This technique aims to break the temporal correlation between the samples, improving the efficiency of the data in the process of learning.

The DDPG agent is trained using out-of-policy learning with replay buffer samples. The critic network is updated using the mean squared error (MSE) loss between the predicted Q values and the target Q values derived from the target critic network. The actor is updated by maximizing the expected Q value for the actions performed by the actor in the current states, promoting actions that lead to higher Q values. For agent exploration, noise is added to the actions selected by the network of actors. However, as training progresses, the noise is reduced for easy exploration, allowing the agent to exploit its learned policy effectively.

2. METHODOLOGY

The Robotics Toolbox for Python library was used to create the simulation environment and represent the manipulator kinematics. In this step, the floor, obstacles, and the Panda manipulator with its random initial position, target, and coordinates in the plane were added, as shown in Figures 3 and 4.

The Franka Emika company manufactures the Panda manipulator to achieve high performance and affordability, combining human-centered design. The responsive arm features 7 degrees of freedom with torque sensors at each joint, allowing for an adjustable fit and advanced torque control. It has a payload of 3kg and a reach of 855mm.

For the experiment, the Franka-Emika Panda collaborative robot was included to reach a final position, starting from a random initial position, without colliding with the obstacles inserted in the scene.

In this project, the environment was simplified, and the increments and decrements were performed in three joints of the manipulator, aiming to make the task more didactic and with less computational effort. Our main assumptions are that the movement is planar, with random initial states and continuous actions. In addition, the environment includes the possibility of collisions with blocks, floors and boundaries. joints, as shown in Figure 4.

For training, the Deep Deterministic Policy Gradient (DDPG) algorithm was implemented, a variation of the popular actor-critic that is suitable for environments with continuous action spaces. DDPG is based on deep neural networks to estimate actor policies and critic values. The objective of this algorithm is to learn a deterministic policy that maximizes expected returns in a continuous environment.

The method collects experiences, selecting actions with the actor's current policy and storing rewards and states in the replay buffer. The agent samples a batch

of experiences, calculates discounted future rewards using the target crit network, and updates the crit weights. The actor's network is updated using an upward gradient to maximize the Q value estimated by the critic. Training continues iteratively, updating networks to improve policy and value estimation. The DDPG seeks to find an optimal policy to maximize expected returns, allowing precise and effective actions in continuous environments.

The main class is responsible for initializing the Neural Networks and applying the actor-network policy in each of the steps of each episode. During this execution, an exploration noise, known as Ornstein-Uhlenbeck, is added to the action taken by the agent, to improve the exploration of the environment. The agent acts as the environment and, in response, receives the next state, the reward obtained and a flag indicating whether the episode was completed. At each step, the experience tuple (state, action, reward, next state, conclusion) is stored in the agent's replay memory buffer. When the amount of experiences stored exceeds the batch size, the agent starts updating its actor and critic networks using samples from the replay buffer. The actor's network is updated to improve the policy, using the policy gradient method. Meanwhile, the critic network is updated to minimize the mean squared error (MSE) between estimated Q values and target Q values.

In addition, at the end of each evaluation stage, the average reward of the last 10 episodes is compared with the best performance recorded so far. If the current average reward is greater, the actor and critic models are saved as the best models.

The agent aims to learn an optimal policy, that is, a strategy that maximizes the expected long-term rewards. To achieve this goal, the agent uses learning-by-assurance algorithms to explore and learn from continuous interaction with the environment and value estimates. Over time, the agent adjusts its actions based on past experiences and learns to make the best decisions, maximizing rewards over time, as shown in Figure 2.

The reward function described in equation (1) takes into account the position error, attitude error, and respective penalties, allowing the algorithm to learn to track both the manipulator's Cartesian trajectory and its orientation. The fitness $\mathcal{F}$ *represents* how close the configuration of the end effector is to the desired configuration in the trajectory. The equation applies the previous iteration's fitness value, the current iteration's fitness value, the fitness value of the objective, an amplification factor m, and an additional penalty for each step, to optimize the agent [6].

$$R = \begin{cases} r_{col}\,, & if\ collided \\ r_{suc}\,, & if\ sucess \\ arctan\left[\left(\mathcal{F}_{i-1}^{j} - \mathcal{F}_{i}^{j}\right)\frac{\pi}{2} \times \frac{1}{\mathcal{F}_g}\right] m + step_{penalty}, & otherwise \end{cases} \tag{1}$$

Training the DDPG agent in the learning environment is an iterative and adaptive process that aims to enable the agent to perform a specific task. For this, parameters and configurations are defined that guide the agent's learning and exploration in the simulated environment.Each joint of the robot is controlled incrementally in 0.02 steps (Delta) to improve the precision of movements.

The task is completed when the agent presents a fitness smaller than 0.002. To discourage unwanted actions, penalties are applied. For example, the agent receives a -30 penalty for every step taken during the episode, encouraging the agent to complete the task efficiently. Also, collisions with obstacles result in a more significant reduction of -800 to avoid unsafe behavior and the episode ends. On the other hand, the agent is rewarded with 1000 when he reaches the goal. Such rewards and compensations are amplified by a factor of 100 to accelerate learning.

Figure 2. Reinforcement Learning diagram

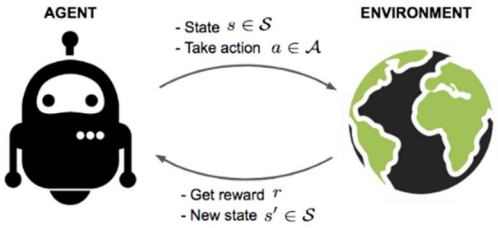


Figure 3. Robot in starting position

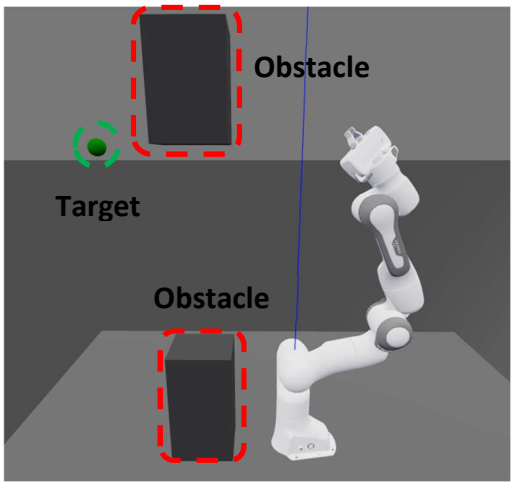
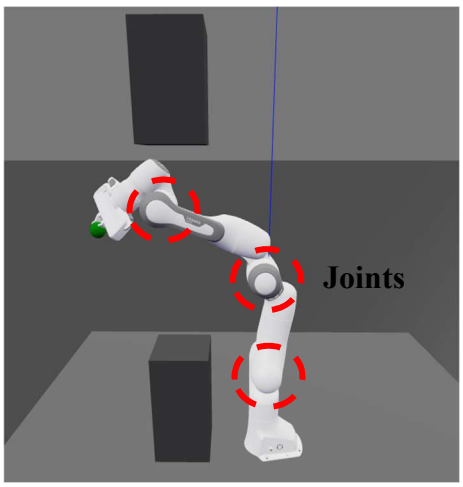


Figure 4. Robot in final position



3. RESULTS AND DISCUSSION

The training rounds were performed using the Optuna optimizer to find better hyperparameters. The result of this study provided the parameters for the best performance model, aiming to reach the final position of the manipulator without any collision with obstacles located on the floor and ceiling of the environment. Figure 3 represents a random initial position of the robot and fixed obstacles after the manipulator reset. Figure 4 represents the goal to be achieved after training with their respective active joints .

The training is conducted in several steps defined by the total number of steps. Each episode starts with the agent in an initial state of the environment. The agent interacts with the environment using learned policy to determine its actions, which are affected by noise. This noise allows for a more diversified exploration of possibilities for action, which can lead to a more efficient learning process.At each step taken by the agent during an episode, the experiences are stored in a replay buffer, which is used to periodically update the actor and critic models. This update is done by adjusting

the weights of the agent's neural networks to improve both the policy and the value function. This iterative and feedback process allows the agent to improve its strategy and obtain better performances throughout the training.

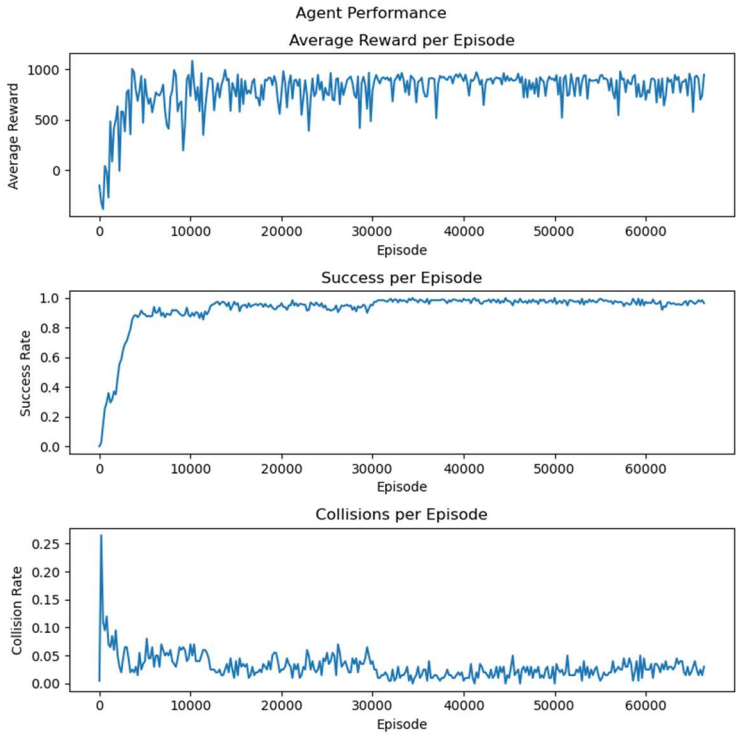
The manipulator was trained with the parameters described in Table 1, and its average reward converged around 946. In the simulation phase, he obtained a total reward of 925 and completed the task in 4 steps. After optimization on the hyperparameters, the agent was trained with the reward function described by Eq. (1) adapted from, converging on a successful positioning trajectory.

Table 1. Training Parameters

Parameter	Value
Learning Rate	0.001
Total Number of Steps	66574
Batch Size	256
Magnification Factor m	100
fitness	0.002
Step Penalty	-30
Collision Penalty	-800
Success Reward	1000
Delta	0.02

Figure 4 shows the success rates, collision rates, and average reward value for the best model. It can be seen that the average reward stabilizes around 1000, and the graphs related to success and collisions behave as expected for policy convergence, showing that the agent learns quickly to avoid obstacles.

Figure 4. Success Rate, Collision Rate and Average Reward.



4. CONCLUSION

This work demonstrated the applicability and usability of the DDPG algorithm and the Robotics Toolbox for Python environment for the classic inverse kinematics problem. It successfully reached the final pose from a random starting point while avoiding obstacles. The simulation environment allows the development of the necessary skills in a safe, controlled and easy-to-install interface without relying on a real robot, saving time and costs. In some situations, these learnings can be transferred and tested in real manipulators after achieving good performance. For future studies and improvements, the movement in three dimensions will be considered in an environment with physics and will test the performance of algorithms such as Monte Carlo Policy Gradient and Proximal Policy Optimization, both methods involving policy iteration.

Acknowledgments

This research was executed in partnership between SENAI CIMATEC and Shell Brasil. The authors would like to acknowledge Shell Brasil Petróleo LTDA, the Brazilian Company for Industrial Research and Innovation (EMBRAPII), and Brazilian National Agency for Petroleum, Natural Gas and Biofuels (ANP) for the support and investments in RD&I.

5. REFERENCES

[1] KROEMER, Oliver; NIEKUM, Scott; KONIDARIS, George. A review of robot learning for manipulation: Challenges, representations, and algorithms. **The Journal of Machine Learning Research**, 2021.

[2] Liu, R., Nageotte, F., Zanne, P., de Mathelin, M., Dresp-Langley, B., 2021. **Deep reinforcement learning for the control of robotic manipulation: A focussed mini-review.** Robotics 10, 22. doi:10.3390/robotics10010022.

[3]Data Science Academy. **Deep Learning Book, 2022.** Disponivel em: < https://www.deeplearningbook.com.br/?s=65 > Acesso em: 5 abril. 2023.

[4] P.Corke And J.Haviland. **Not your grandmother's toolbox- The Robotic's Toolbox reinvented for Python.** Proc. ICRA,2021.

[5] Timothy P. Lillicrap and Jonathan J. Hunt and Alexander Pritzel and Nicolas Heess and Tom Erez and Yuval Tassa and David Silver and Daan Wierstra. **Continuous control with deep reinforcement learning.** Machine Learning. https://doi.org/10.48550/arXiv.1509.02971

[6] Malik, A.; Lischuk, Y.; Henderson, T.; Prazenica, R. **A Deep Reinforcement-Learning Approach for Inverse Kinematics Solution of a High Degree of Freedom Robotic Manipulator**. *Robotics* 2022, *11*, 44. https://doi.org/10.3390/robotics11020044