Panda Manipulator Control with Obstacle Avoidance Through Reinforcement Learning in a Simulated Environment

Marcelo Albergaria Paulino Fernandes Ferreira[1*], Taniel Silva Franklin[1], Oberdan Rocha Pinheiro[1]
[1]SENAI CIMATEC University; Salvador, Bahia, Brazil

Robotic systems with learning capabilities have many powerful applications in unstructured environments. Through reinforcement learning, robots can quickly adapt to new situations and learn from direct environmental interaction. 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: Reinforcement Learning. Machine Learning. Robotics.

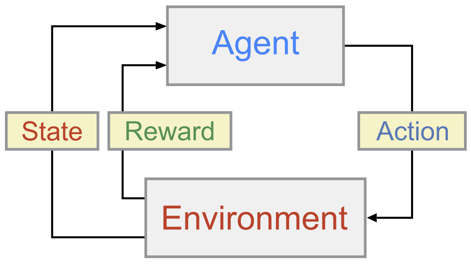
Commercial and industrial robots nowadays have a wide range of applications. They often assist humans in dangerous, repetitive, and exhausting tasks. Many extreme environments are challenging to access or require expensive logistics to transport specialized personnel. One of intelligent robotics's main challenges is creating robots capable of interacting directly with the world around them to achieve their goals [1]. The wide variety of usage scenarios and environmental variations suggests that an effective manipulator must be able to cope with environments that neither it nor its designers have foreseen or encountered before. The growing availability of computational resources has boosted the development of machine learning, enabling the emergence of promising technologies such as recommendation systems, autonomous vehicles, video games, energy management, and robotics, among others [2].

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. RL, the Agent has the function of taking actions to resolve complex problems, interacting with the environment that responds with observations or states, and rewards or costs [3].

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, he receives a reward signal from the environment (Figure 1). 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.

Training with a real robot is expensive because machine learning requires considerable data representing the robot's experience in the environment. Researchers have sought ways

Figure 1. Reinforcement learning scheme.



Received on 27 September 2024; revised 28 December 2024.
Address for correspondence: Marcelo Albergaria Paulino Fernandes Ferreira. Av. Orlando Gomes, 1845, Piatã. Zipcode: 41650-010. Salvador, Bahia, Brazil. E-mail: marcelo_albergaria@hotmail.com.

J Bioeng. Tech. Health 2025;8(1):68-73

to reduce this dependency through training approaches by simulation 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.

Inverse kinematics is a classic problem in robotics. The aim is to determine the angles of a manipulator's joints to achieve a specific pose in space. This problem can be addressed in two main ways: analytical methods, which provide direct mathematical solutions, or numerical methods, which involve iterative algorithms such as optimization or solving systems of non-linear equations [4].

Reinforcement learning (RL) presents a model-free alternative to robot control. In this strategy, the robot learns a control policy by interacting with its environment and learning to make decisions that result in desired behaviors, such as reaching a pose or manipulating objects. This strategy is particularly effective when there is no perfect model of the system or when the environment is complex and non-deterministic, allowing the robot to adapt and improve its performance based on accumulated experience.

This project aims to solve the problem of inverse kinematics using a simplified environment built with Robotics Toolbox for Python [5]. 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 and create obstacles.

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 in the scene.

In this context, it is useful to present a reinforcement learning environment for training robot manipulators in Python. The simulator is based on the 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.

Materials and Methods

The Robotics Toolbox for Python library was used to create the simulation environment and represent the kinematics of the manipulator. The floor, the table (obstacle), and the Panda manipulator were added in this step. A random initial position for the robot and target was used, as shown in Figures 2 and 3.

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. The joints of the Panda robot have specific range limits to ensure safe and effective operation: Joint 0, joint 2, joint 4, and joint 6 have a range of -166° to 166°, joint 1 ranges from -101° to 101°, joint 3 ranges from -176° to 4° and joint 5 ranges from -1° to 215°.

Our main assumptions are that motion occurs in 3D, initial states are random (Figure 2), and actions are continuous with 7 active rotational joints, allowing for collisions with the table, floor, and joint boundaries (Figure 3).

The Deep Deterministic Policy Gradient (DDPG) algorithm was used for training. This algorithm is a variation of actor-critic suitable for environments with continuous action spaces. DDPG is based on using deep neural networks to estimate actor policies and critic values. This off-policy algorithm aims to learn a deterministic policy that maximizes expected returns in a continuous environment.

The method collects experiences, selects actions according to the actor's current policy, and stores 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 critic weights. The actor's

Figure 2. Robot in starting position.

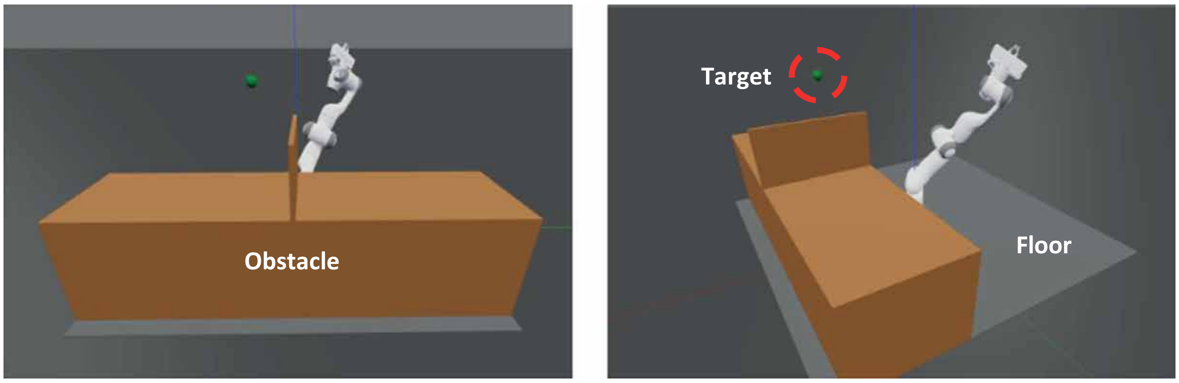
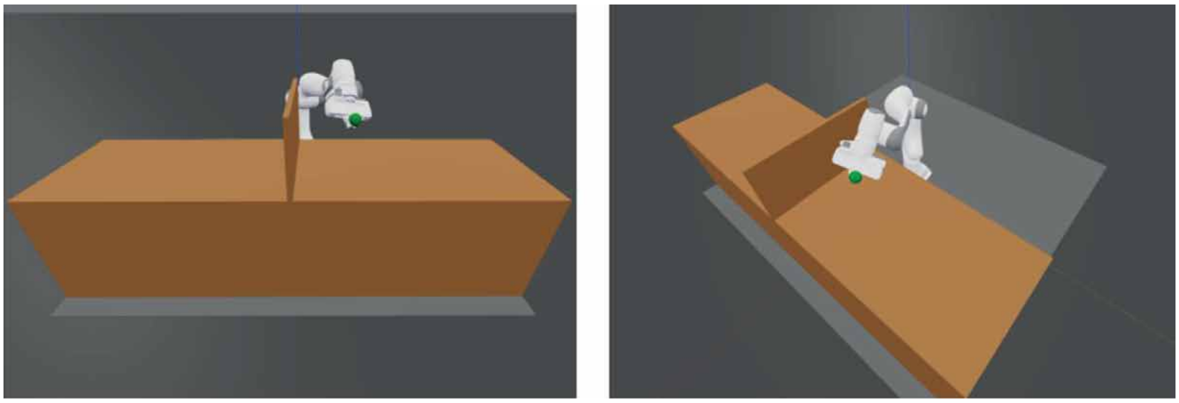


Figure 3. Robot in final position.



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 neural networks are initialized during training, and the actor-network policy is applied to each episode. An exploration noise, known as Ornstein-Uhlenbeck, is added to the action taken by the Agent to ensure further 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 has been completed. When the amount of stored experiences exceeds the batch size, the Agent updates its actor and critic networks using samples from the replay buffer. The actor-network is updated to improve the policy using the policy gradient method. Meanwhile, the critic network is updated to minimize the mean square error (MSE) between the estimated and desired Q-values.

The reward function used is a scalar reward function (1), which aims to balance two crucial aspects of the manipulator's behavior: the robot's proximity to the target and the magnitude of the action performed. The total reward is a weighted sum of two components and their coefficients.

$$R = \begin{cases} r_{col}, & \textit{if collided} \\ r_{suc}, & \textit{if sucess} \\ r(s,a) = \left(c_1 \times r_{tg}(s,a) + c_2 \times r_{action}(s,a)\right), & \textit{otherwise} \end{cases} \quad (1)$$

$C1$, weights the importance of proximity to the target, indicating a significant penalty for considerable distances to the target, and $C2$weights the magnitude of the action, encouraging more minor actions to promote efficient movements.

The object proximity reward $r$tg (2) encourages the manipulator to approach the goal precisely, where d is the manipulator's current distance from the goal and δ is a smoothing parameter. The action magnitude reward $r$action penalizes large action magnitudes, encouraging smooth and efficient movements, a is the applied action vector (3).

$$r_{tg}(d) = \begin{cases} \frac{1}{2} d^2, & if\ d < \delta \\ \delta(d - 0.5\delta), & if\ d \geq \delta \end{cases} \tag{2}$$

$$r_{action}(a) = -\|a\| \tag{3}$$

Collision and step penalties ensure that the Agent minimizes excessive movements and avoids collisions. For example, the Agent receives a -0.5 penalty for each step taken during the episode, encouraging it to complete the task efficiently. Collisions with obstacles result in a significant penalty of -500 to avoid unsafe behavior. On the other hand, the Agent is rewarded with 500 when it reaches the goal. During agent training in the simulated Panda robot environment, fitness is calculated at each step to provide a metric for the Agent to adjust its actions and improve its performance. This metric continuously evaluates the position and orientation of the robot's end-effector relative to the desired target. Training involves applying the Agent's action, calculating the new state, evaluating the fitness, and deciding on the reward. The task is completed when the fitness value is smaller than the target of 0.004.

Results and Discussion

The training rounds were carried out using the Optuna optimizer [6] to automate adjusting the hyperparameters, saving time and computational resources. The result of this study provided the parameters for the best-performing model, aiming to achieve the manipulator's final position without any collision with obstacles (Table, Floor, Joint Boundaries) in the environment. Figure 2 represents the random initial position of the robot and the obstacles after resetting the manipulator, and Figure 3 represents the final state of the robot, with the objective achieved. The model's architecture includes neural networks for the actor and the critic. Both networks are configured with multiple densely connected layers, using Rectified Linear Unit (ReLU) activation functions in the hidden layers and a Tangent Hyperbolic (Tanh) activation function in the output layer of the actor network. This architecture allows the model to learn and generalize complex control patterns, optimizing the Agent's performance in the simulation environment.

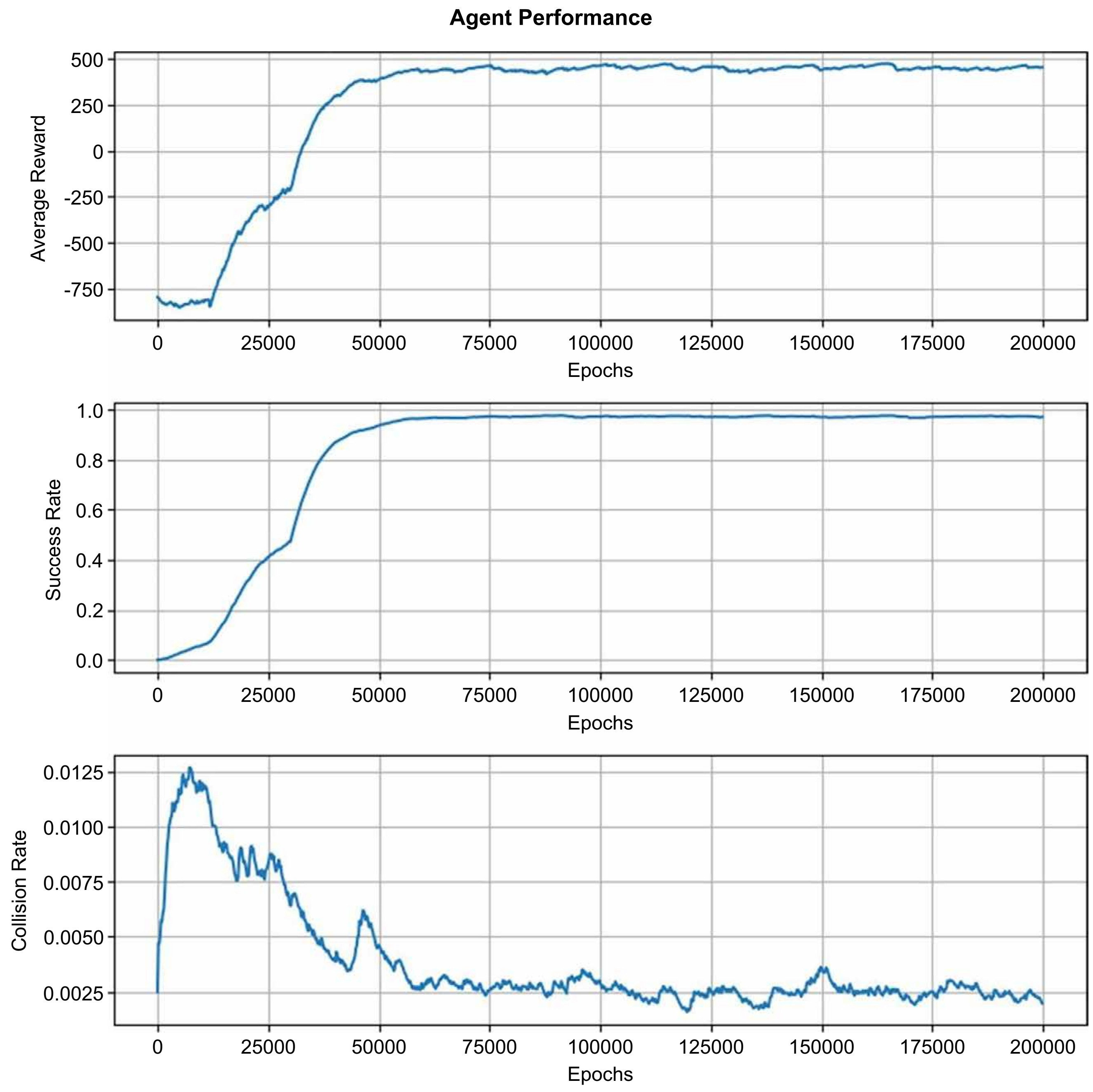
The manipulator was trained with the parameters described in Table 1, and its average reward converged around 480. In the simulation phase, he obtained a total reward of 500 and completed the task in 1 step. After optimization on the hyperparameters, the Agent was trained with the reward function described by Eq. (1), converging on a successful positioning trajectory.

Figure 4 presents the best model's success rates, collision rates, and average reward values. The average reward stabilizes around 500, indicating policy convergence. The success and collision graphs demonstrate that the Agent quickly learns to avoid obstacles and reach the target efficiently.

Table 1. Training parameters.

Parameter	Value
Learning Rate	0.001
Total Number of Steps	200000
Batch Size	200
Fitness	0.004
Step Penalty	-0.5
Collision Penalty	-500
Sucess Reward	500
Gamma	0.99
Memory Size	50000
First Hidden Layer	1200
Second Hidden Layer	1800

Figure 4. Success rate, collision rate, and average reward.



To illustrate the results, we have included a file (https://1drv.ms/f/c/f6931d804d905e8a/EhicS8adWDFIsr0h8sRQ0fgBCT1u13Vxfcqy29-j0Vwpnw?e=KI1WcZ) containing simulations that characterize successful tasks, failed tasks and random start positions of the manipulator based on the trained model.

Conclusion

This work demonstrated the applicability and use of the DDPG algorithm and the Robotics Toolbox for the Python environment for the classic inverse kinematics problem. The results indicate that the trained Agent could learn to interact with the environment appropriately, reaching the goal in a minimum number of steps and avoiding collisions. The designed reward function, which balances the target's proximity and the actions' magnitude, proved effective in learning. The simulation environment allows the necessary skills to be developed in a safe, controlled, and easy-to-install interface without relying on a real robot, saving time and costs. These learnings can be transferred and tested on real manipulators after performing well in some situations. For future studies and improvements, implementing the task in a simulation environment with more realistic physics, such as Pybullet, would help to reduce the gap between simulation and reality. In addition, tests with reinforcement learning algorithms already used in other robotic tasks, such as Proximal Policy Optimization (PPO), could provide a solid comparative basis.

Acknowledgments

This research was carried out in partnership between SENAI CIMATEC and Shell Brasil. The authors would like to thank Shell Brasil Petróleo LTDA, Empresa Brasileira de Pesquisa e Inovação Industrial (EMBRAPII) and Agência Nacional do Petróleo, Gás Natural e Biocombustíveis (ANP) for their support and investments in RD&I. investments in RD&I.

References

1. Kroemer O, Niekum S, Konidaris G. A review of robot learning for manipulation: Challenges, representations, and algorithms. The Journal of Machine Learning Research 2021.
2. Elguea-Aguinaco I, Serrano-Muñoz A, Chrysostomou D, Inziarte-Hidalgo I, Bogh S, Arana-Arexolaleiba, N. A review on reinforcement learning for contact- rich robotic manipulation tasks. Robotics and Computer-Integrated Manufacturing 2023;81:102517,2023.
3. Liu R, Nageotte F, Zanne P, de Mathelin M, Dresp-Langley B. Deep reinforcement learning for the control of robotic manipulation: A focussed mini-review. Robotics 2021;10:22. doi:10.3390/robotics10010022.
4. CraigJJ.Introduction to Robotics: Mechanics and Control. Pearson/Prentice Hall, 2005.
5. Corke P, Haviland J. Not your grandmother's toolbox-The Robotic's Toolbox reinvented for Python. Proc. ICRA 2021.
6. Akiba T, Sano S, Yanase T, Ohta T, Masanori K. Optuna: A Next-generation Hyperparameter Optimization Framework. In KDD 2019.