engineering proceedings

MDPI

Proceeding Paper

A Deep Reinforcement Learning Algorithm for Robotic Manipulation Tasks in Simulated Environments †

Carlos Calderon-Cordova * and Roger Sarango

Department of Computer Science and Electronics, Universidad Técnica Particular de Loja, Loja 1101608, Ecuador; rasarango1@utpl.edu.ec

* Correspondence: cacalderon@utpl.edu.ec or cacalderon@ieee.org

† Presented at the XXXI Conference on Electrical and Electronic Engineering, Quito, Ecuador, 29 November–1 December 2023.

Abstract: Industrial robots are used in a variety of industrial process tasks, and due to the complexity of the environment in which these systems are deployed, more robust and accurate control methods are required. Deep reinforcement learning emerges as a comprehensive approach that directly allows for the mapping of sensor data and the setting of motion actions to the robot. In this work, we propose a robotic system implemented in a semi-photorealistic simulator whose motion control is based on the A2C algorithm in a DRL agent; the task to be performed is to reach a goal within a work area. The evaluation is executed in a simulation scenario where a fixed position of a target is maintained while the agent (robotic manipulator) tries to reach it with the end-effector from an initial position. Finally, the trained agent fulfills the established task; this is demonstrated by the results obtained in the training and evaluation processes, and the reward value increases when the measured distance decreases between the end-effector and the target.

Keywords: robotics manipulation; DRL algorithms; deep learning; reinforcement learning; robotic simulator; CoppeliaSim

Citation: Calderon-Cordova, C.; Sarango, R. A Deep Reinforcement Learning Algorithm for Robotic Manipulation Tasks in Simulated Environments. *Eng. Proc.* **2023**, *47*, 12. https://doi.org/10.3390/engproc2023047012

Academic Editor: Jackeline Abad

Published: 4 December 2023

1. Introduction

In recent decades, Industry 4.0 (I4.0) has promoted the upgrade of automation and the digitization of industrial processes, using emerging technologies such as cloud computing, artificial intelligence (AI), internet of things (IoT), cyber-physical systems, etc. [1]. Some of the industrial processes such as welding, manufacturing, and object manipulation require robotic systems that enable decision making to execute specific tasks within production lines [2].

The integration of artificial intelligence improves the efficiency of factory automation; in particular, machine learning (ML) is implemented, which gives systems the ability to learn to execute a task and has been demonstrated to improve control in many applications [3]. The learning process is limited by several factors: the volume of data required for the system, the elements of scenarios in industrial areas, deployment time, the cost of equipment, and others. Currently, there are several simulation platforms [4] that allow for the simulation of various scenarios, generating synthetic data and robotic components that allow for the developing of an end-to-end system based on artificial intelligence.

In machine learning, Deep Reinforcement Learning (DRL) has been of great interest to researchers with academic and industrial approaches for several application areas [5]. In the field of robotic manipulation, there are several frameworks and simulation platforms that allow for the implementation and evaluation of various control algorithms for the application of RL in robotics [6]. Despite the success of DRL, there are also methods that allow for the optimization of DRL algorithms by performing CPU and GPU combinations, thus allowing larger data batches to be used without affecting the final performance [7].

In the literature, there is a large number of papers presenting reviews of DRL algorithms for robotic manipulation; however, in these papers, they do not provide details of the implementation process for a specific problem. The contribution of this work is to implement a robotic system in a simulation platform; the end-to-end control approach is based on DRL, and its performance in the classical task of reaching a goal using a robotic manipulator is evaluated.

The structure of the paper is described as follows: Section 2 describes the main fundamentals and structure involved in deep reinforcement learning applied to robotic manipulation. Section 3 describes the components and architecture of the robotic system implemented in simulation. Section 4 provides the details of the training and evaluation processes of the robotic system whose control is based on DRL. Section 5 presents some results obtained with the simulation implementation of the DRL algorithms. Finally, some conclusions obtained from the complete development process of this work are presented.

2. Deep Reinforcement Learning

Deep reinforcement learning algorithms are based on the iterative trial-and-error learning process. The interaction between the agent and the environment in which the task is performed is modeled after a Markov Decision Process (MDP). With this method, the interaction is reduced to three signals: the current state of the environment (observations), a decision made by the agent based on the state (action), and feedback with positive or negative value depending on the action performed by the agent (reward) [8]. The mathematical foundations of DRL approaches are based on the Markov Decision Process (MDP) [9], which consists of five elements:

$$MDP = (S, A, P, R, \gamma), \tag{1}$$

where S is the set of states of the agent and the environment, A is the set of actions executed by the agent, P is the model of the system—in other words, it is the transition probability of a state—R is the reward function, and γ is a discount factor [10]. The DRL objective function has two forms: the first is a value function that defines the expectation of the accumulated reward. $V^{\pi}(s)$ represents the estimates of the state-value function in the policy for the MDP; given a policy π, we have the expected return:

$$V^{\pi}(s) = E_{\pi}\left[r_t + \gamma r_{t+1} + \gamma r_{t+2} + \cdots \middle| s_t = s\right], \tag{2}$$

The second is the action-value function, which is known as the Q function; this function indicates that after performing an action a, based on the policy π on the state s, an accumulative reward (r) is generated. $Q^{\pi}(s, a)$ represents the estimates of the action-value function in the policy for the MDP; given a policy π, we have the following expected return:

$$Q^{\pi}(s, \alpha) = E_{\pi}\left[r_t + \gamma r_{t+1} + \gamma r_{t+2} + \cdots \middle| s_t = s, a_t = a\right], \tag{3}$$

The Advantage Actor-Critic (A2C) algorithm combines two elements of reinforcement learning: a policy gradient (actor) and a learned value function (critic). The Actor network learns a parameterized policy, and the Critic network learns the value function that evaluates pairwise state–action signals. The Critic network provides a reinforcement signal to the Actor network. The Advantage function of this algorithm is that the Actor network chooses an action at each time step, and the Critic network evaluates that action based on the Q value of an input state. As the Critical network learns which states are better, the Actor network uses that information to teach the agent to search for better states.

$$A_t = R_t + V\gamma(r_{t+1}) - V\gamma(r_t), \tag{4}$$

where A_t is the action of the state, R_t is the reward obtained in a given a state S_t, $R_t = R(S_t)$, and R is the reward function. To have a correct advantage function, the TD error is used as a better estimator of the decision-making process when choosing an action in a given

state [9]. Agents can be trained using A2C to estimate the advantage function by combining the estimated reward value and the observed reward value.

3. Proposed System

The robotic system proposed in this work was developed using free software tools for robotics research, the agent to be evaluated is a robotic manipulator of 7 degrees of freedom, and the task to be performed is to reach a target within its working area in a semi-photorealistic scene. The training data are obtained directly from the simulator, and by means of the API integrated in this platform, the data flow control and control of the robot is performed.

3.1. System Architecture

The robotic manipulator control approach is based on the general DRL scheme; the software system architecture is a client–server type. The client contains the components to train and evaluate the DRL agent located on the server side, which consists of the simulation platform. Since this is an end-to-end control approach, the input data to the system consists of the position of the robot joints, the position (x, y, z) of a tip located at the end of the end-effector, and the measured distance between the end-effector and the target sphere. The control actions are angles for each of the joints of the 7-DoF robot.

This configuration is shown in Figure 1 and describes the process of learning an agent using DRL for robotic manipulator in CoppeliaSim.

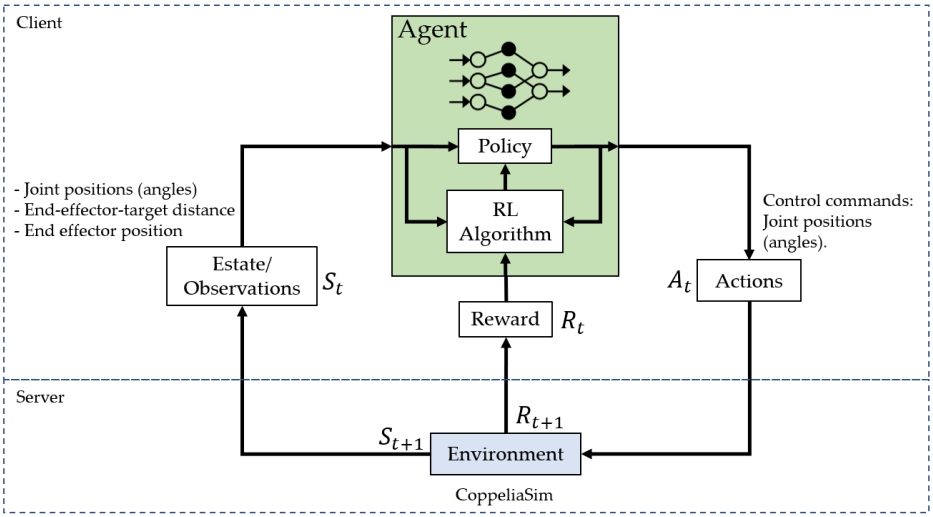


Figure 1. DRL process architecture for the robotic manipulator.

When using an A2C algorithm with an Actor-Critic policy, the model structure consists of two networks: (1) the Actor network recommends the action to be taken by the agent and gives the probability distribution of the state, and (2) the Critical network will give an estimate of the reward value for the executed actions. It yields the estimated total rewards in the given state in the future. The neural network architecture consists of the structure shown in Figure 2, which consists of the two networks Actor and Critic; each network layer has a size of 256, 512, and 256, respectively, with an ReLU activation function.

3.2. Reward Function

The reward function should make the robotic manipulator reach the target position, whose end-effector should approach a sphere located in the working area. After executing an action, an observation is obtained, and the total reward is calculated, which is achieved with the following expression:

$$R_t = r_{dist} + r_{ctrl} + r_{pa} + r_l + r_c, \quad (5)$$

where r_{dist} represents the distance reward, which quantifies the Euclidean distance between the end of the robotic manipulator and the target sphere. This reward has a more negative value when the end of the manipulator is farther away from the target. On the other hand, r_{ctrl} corresponds to the control reward, which is calculated as the squared Euclidean norm of the action taken by the agent. This reward penalizes the agent when it makes excessively large movements. In addition, the reward r_{pa} is obtained by comparing the previous distance with the current distance after performing an action. If the previous distance is less than the current distance, the agent receives a negative reward, as this indicates that the end-effector of the manipulator is moving away from the target position. The difference reward is calculated by the following function:

$$r_{pa} = d_p - d_a, \tag{6}$$

where d_p is the previous distance between the end-effector and the target to be reached, this distance value is calculated before executing an action of the robot, and d_a, indicates the current distance between the end-effector position and the sphere position. The arrival reward r_l of the manipulator is defined as a positive value that is within a threshold distance between the end-effector and the target. The collision reward r_c is a positive value given to the agent when the effector touches the target object. The total reward is composed of the five elements already mentioned and is defined by the following function:

$$R_t = \begin{cases} r_{dist} : -\sqrt{\sum_{i=1}^{n}(y_i - x_i)^2} \\ r_{ctrl} : -\sum_{i=1}^{n}(Action)^2 \\ r_{pa} : d_p - d_a \\ r_l : if|d_a| < d_u \\ r_c : if\ collision = True \end{cases} \tag{7}$$

where *Action* is a vector composed of the angular positions of the seven joints of the robot. The term d_u is the threshold distance that determines that the end-effector is approaching the target, and the term "collision" has a logical value that represents the collision between these two objects in the simulation; once this condition is fulfilled, the agent is given a positive value for reaching the target. These rewards are essential components in guiding the agent's on-task behavior, encouraging precise movements toward the target sphere and discouraging abrupt or distant actions.

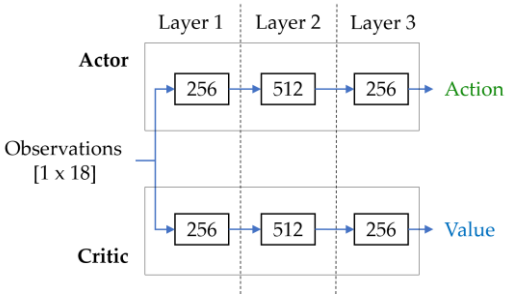


Figure 2. Neural network structure for A2C.

3.3. Action and Observation Space

The robotic manipulator consists of seven joints (7 DoF); as a result, we have the action space of a vector containing the angles of each joint of the robot. The observation space is a vector of 1×18 elements containing the position of the end-effector, the angles of each joint, and the distance between the end-effector and the target sphere.

3.4. Simulation

The implementation of the robotic manipulator occurs in the CoppeliaSim simulator [11]; this platform provides an API in Python that allows for communication with

external software, data acquisition, and control functions. The base architecture for this system consists of two elements: the first is the server in which the robotic manipulator (Franka Emika Panda) and a static element (red sphere) that will be the target to reach are deployed. The second is the client, which consists of Python scripts for the creation of a customized environment with Gymnasium [12] and the training and evaluation of the agent with Stable Baselines 3 [13].

The simulation environment is presented in Figure 3; the objective of the agent is to move the end-effector to a target position, which is marked by a red sphere. At the end of the end-effector, a mark is placed in order to set a parameter for the observation space; the working area is inside the white colored area.

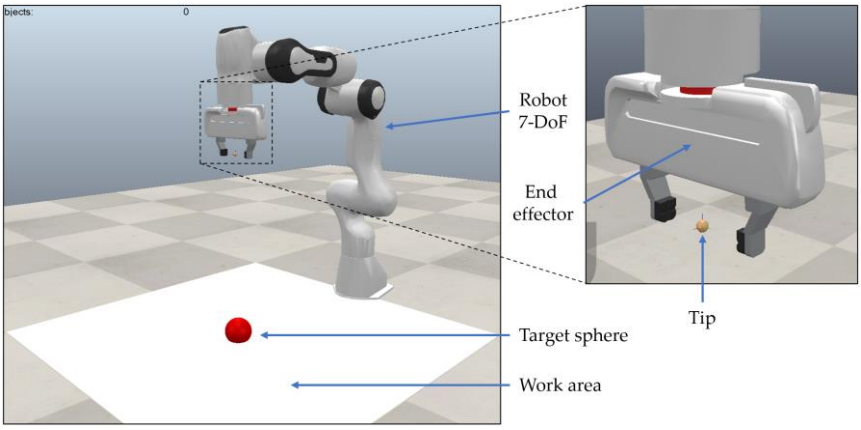


Figure 3. Simulation scene for the robotic manipulator.

4. Training and Evaluation Details

The robotic system was trained and evaluated in a simulation environment, and the following software components were used: Python as the base programming language, CoppeliaSim as the simulation platform, Stable Baselines 3 as the DRL framework that allowed for the integration of a deep neural network (DNN), and the A2C algorithm in the test agent. The hardware used for the implementation of this system has the following features: Ryzen 7 CPU, NVIDIA GeForce RTX 3070 8 GB GPU, and 16 GB RAM.

During the training phase, the robotic manipulator searches to approximate the position of the target located at a point in the scene; the coordinates of the robot joints towards the target are given randomly and updated after several episodes, making the robot perform its task better. This makes the DRL training efficient and robust.

Figure 4 presents the main elements used for this robotic system in software. In (a), the custom environment for CoppeliaSim is presented, and the class is created; Gymnasium methods for RL and control functions and the data acquisition of the simulator are established. (b) is part of the client in which the agent training is performed, and the model is saved. (c) is also part of the client, and the trained agent is evaluated, which involves loading the model and running it on the simulation platform to verify that the assigned task is being performed.

The selected algorithm is the Actor-Critical Advantage (A2C) [14], because it meets the requirements of the action and observation spaces used for Gymnasium that are supported by the DRL framework, and it is one of the algorithms used for robotic manipulation [15]. The input state of the deep neural network consists of a vector with these elements: the position of the tip point of the end-effector, the angles of the joints, and the distance from the end-effector to the target. The action performed by the system is a vector containing seven elements corresponding to each of the robot's joints.

For additional details on the development and implementation of the robotic system in the simulation platform, a link of the repository is attached here: https://github.com/RogerSgo/DRL-Sim-Reach-Target (accessed on 2 October 2023).

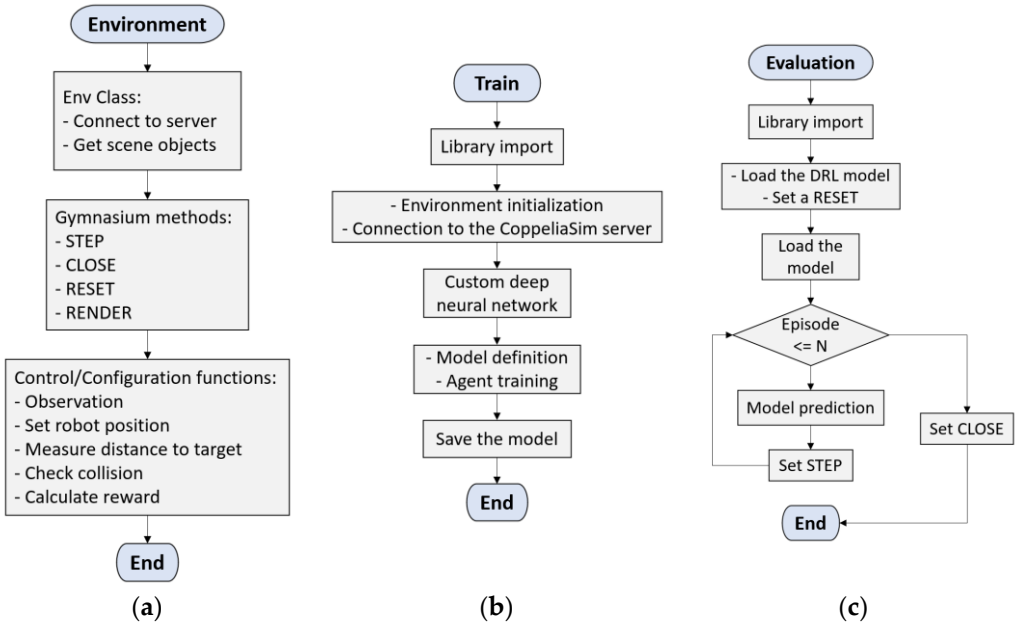


Figure 4. The flowcharts of the software system: (**a**) Customized environment for CoppeliaSim; (**b**) Flowchart for client training; (**c**) Flowchart for the evaluation of the trained model.

5. Results

This section presents the main results obtained from the experimentation carried out in the CoppeliaSim EDU 4.5 simulation environment. The model was trained with 1000 episodes and with a standard Actor-Critic neural network architecture of 3 layers of the sizes 256, 512, and 256 in each network. The resulting test model took about 1 h to train.

Table 1 shows the reward values obtained during the training of the agent with the A2C algorithm during 1000 episodes; it can be seen that the training worked according to the criteria established for the assigned task. In this case, it is observed that the as the distance (d_a) between the end-effector and the target decreases, the reward value (R_t) increases, which indicates the agent learning to perform this movement action.

Table 1. Reward values.

Episode	d_p	d_a	r_{dist}	r_{ctrl}	r_{pa}	r_l	r_c	R_t
1	0.467	0.465	−0.465	−0.350	0.002	0.000	0.000	−0.813
500	0.158	0.145	−0.145	0.552	0.013	0.145	0.00	−0.538
1000	0.092	0.090	−0.090	−0.0333	0.001	0.090	0.00	−0.332

The learning curve of the A2C agent is based on the training reward. To obtain the learning curve, the sum of the rewards that the agent obtains during the set of episodes determined for training is measured. A high cumulative reward indicates that the agent is successful in the task. The learning curve is presented in Figure 5.

To verify the performance of the reach-to-target task using the A2C algorithm, the trained model is evaluated by performing multiple simulation experiments in CoppeliaSim. In Figure 6, which robotic manipulator can execute the task of reaching a target using the DRL control method is demonstrated. The robot starts from an initial position, and with the use of the trained model, it performs motion predictions until it reaches the target located at a position in the working area of the environment.

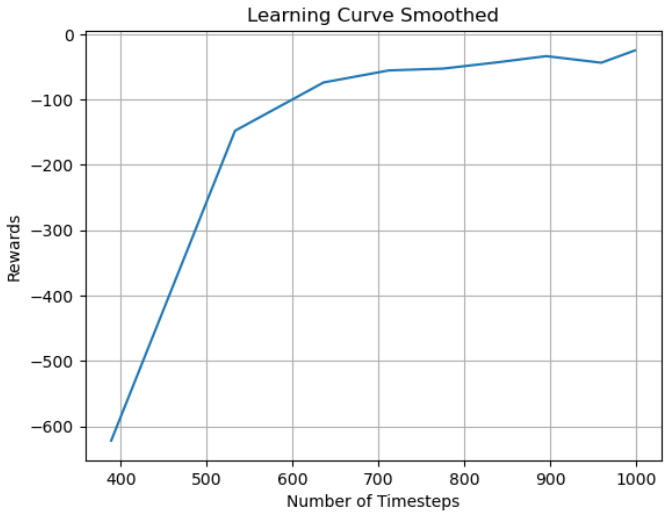
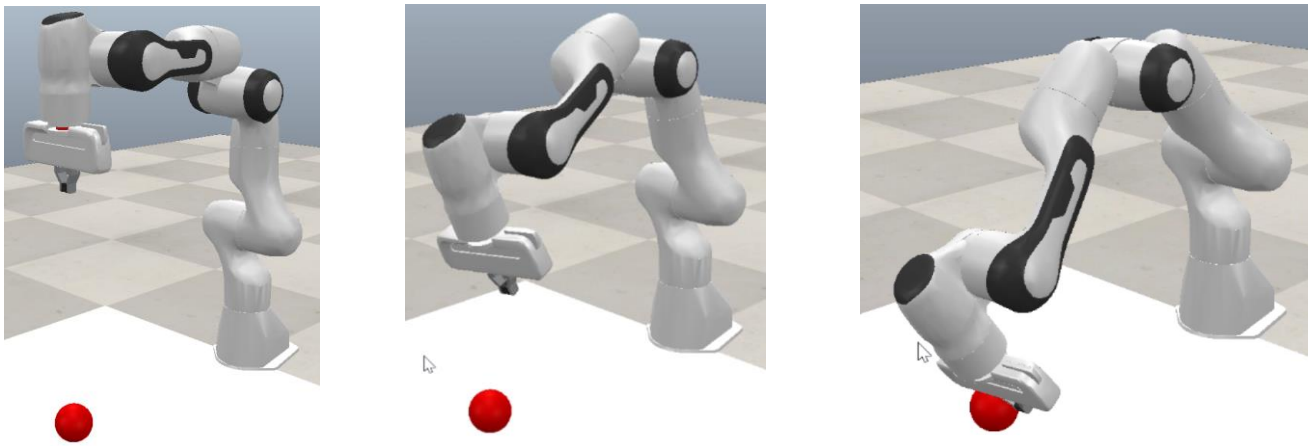


Figure 5. Agent learning curve.



(**a**) (**b**) (**c**)

Figure 6. The sequence of manipulator movement to reach the target: (**a**) The initial position of the robotic manipulator; (**b**) End-effector moves about half the distance to the target; (**c**) End-effector arrives at the target goal.

The performance evaluation of the A2C method for this robotics application is based on the average reward; this parameter is a common metric for the evaluation of RL algorithms. The task to be performed in this work was the reaching of a target by the end-effector of a robotic manipulator, and according to the reward function designed in this system, the reward value should increase as the measured distance between end-effector and target decreases. This shows that the performance is satisfactory.

The evaluation of the trained DRL model was performed by running it with 100 episodes. Two important parameters are presented to measure the agent's performance based on A2C as shown in Figure 7; the curve in red is the reward value accumulated during that number of episodes, and the curve in blue represents the measured distance of the end-effector tip when approaching the target.

With the results obtained from the training process in Table 1 and the evaluation process in Figure 7, it is shown that the reward value increases when the measured distance between the tip positioned at the extreme of the end-effector and the target sphere decreases, with each action predicted by the DRL model. This indicates that the trained agent satisfies the execution of the task during the evaluation of the system in simulation.

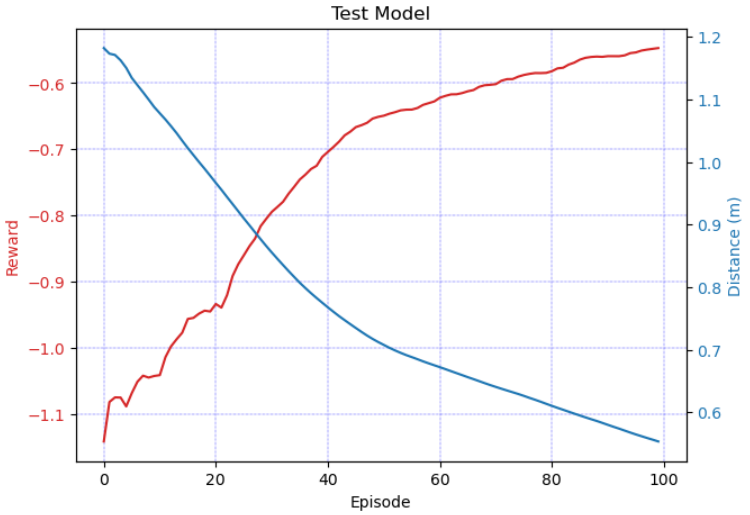


Figure 7. The evaluation parameters of the DRL Agent. The red curve is the reward accumulated when the end-effector approaches the target sphere position. The blue curve is the distance between the end-effector and the target sphere.

In addition to the metrics (reward) and measured distance (m) of the robotic system training, the use of the PC hardware resources of some software components was checked. Table 2 shows the hardware resources used by the software components of the robotic manipulator system during the training process and verifies that the highest consumption occurs by the following: the simulation platform consumes 35% of the total GPU, the Python programming language consumes 39% of the RAM memory, and finally, when using Anaconda to implement the different scripts in Firefox, it uses 9% of the RAM.

Table 2. Hardware performance.

Software	PC Component				
	CPU (%)	GPU (%)	VRAM (%)	RAM (%)	FPS
CoppeliaSim 4.5.1	2	35	37	1	59
Python 3.11.4	1	0.4	-	39	-
Firefox 118.0.2	0.2	0.1	-	9	-

6. Conclusions

In this paper, the A2C-based deep reinforcement learning method to solve the task of reaching a goal for a robotic manipulator is presented; simulation results in CoppeliaSim validate the performance of the proposed system. In addition to the training and evaluation processes of the DRL model, the impact of this DRL control method on the performance of a PC hardware is verified.

Reward calculation is mainly based on the measured distance between the end-effector position and the target to be reached, as well as a penalty to the agent when making movements that are too large. During the training and evaluation processes of the agent, each parameter of the reward function is monitored in order to determine if the task to be executed is being fulfilled; these data are evidenced in Table 1 and Figure 6. Therefore, as the end-effector-target distance decreases, the reward has a higher value.

The agent is composed of artificial neural networks that are integrated in a critical actor architecture and learned based on sensory information of the end-effector's state with respect to the target. Its policy based on direct feedback from the environment with data is evaluated with the following parameters: position, distance, and motion cost.

In Section 5, some quantitative results were presented such as the agent's learning curve, rewards obtained during a certain number of episodes, and the distance measured

between the end-effector and the target. These values were obtained from the training and evaluation of the system and show that the agent was learning to perform the task of reaching a target within the work zone in the CoppeliaSim simulation scene.

The simulation experiments performed showed that the training process was performed in a short time due to GPU acceleration. Current implementation used data from various sources such as joint positions given in angles, data related to the distance from the end-effector to the target, and the distance between the end-effector mark and the target sphere. The effect of the processing performance can be varied by scaling the system via increase of the number of sensing sensors and/or complexity of the deep neural network.

The performance of the DRL agent depends on many factors, and we can emphasize these: the number of episodes for training the model, the framework selected to implement the DRL algorithm, the design of the neural network architecture, and process logic for the actions at each step to be taken by the agent.

Author Contributions: Conceptualization, methodology, C.C.-C. and R.S.; software, R.S.; validation, R.S.; writing—original draft preparation, C.C.-C. and R.S.; writing—review and editing, R.S.; supervision, C.C.-C.; project administration and funding acquisition, C.C.-C. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Universidad Tecnica Particular de Loja, grant number PROY_ARTIC_CE_2022_3667.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. del Real Torres, A.; Andreiana, D.S.; Ojeda Roldan, A.; Hernandez Bustos, A.; Acevedo Galicia, L.E. A Review of Deep Reinforcement Learning Approaches for Smart Manufacturing in Industry 4.0 and 5.0 Framework. *Appl. Sci.* **2022**, *12*, 12377. [CrossRef]
2. Bhuiyan, T.; Kästner, L.; Hu, Y.; Kutschank, B.; Lambrecht, J. Deep-Reinforcement-Learning-based Path Planning for Industrial Robots using Distance Sensors as Observation. In Proceedings of the 2023 8th International Conference on Control and Robotics Engineering (ICCRE), Niigata, Japan, 21–23 April 2023.
3. Jiang, R.; Wang, Z.; He, B.; Di, Z. Vision-Based Deep Reinforcement Learning For UR5 Robot Motion Control. In Proceedings of the 2021 IEEE International Conference on Consumer Electronics and Computer Engineering (ICCECE), Guangzhou, China, 15–17 January 2021; pp. 246–250.
4. Collins, J.; Chand, S.; Vanderkop, A.; Howard, D. A review of physics simulators for robotic applications. *IEEE Access* **2021**, *9*, 51416–51431. [CrossRef]
5. Gupta, S.; Singal, G.; Garg, D. Deep reinforcement learning techniques in diversified domains: A survey. *Arch. Comput. Methods Eng.* **2021**, *28*, 4715–4754. [CrossRef]
6. Nguyen, H.; La, H. Review of deep reinforcement learning for robot manipulation. In Proceedings of the 2019 Third IEEE International Conference on Robotic Computing (IRC), Naples, Italy, 25–27 February 2019; pp. 590–595.
7. Stooke, A.; Abbeel, P. Accelerated methods for deep reinforcement learning. *arXiv* **2018**, arXiv:1803.02811.
8. Gym, O.; Sanghi, N. *Deep Reinforcement Learning with Python*; Springer: Berlin/Heidelberg, Germany, 2021.
9. Dong, H.; Ding, Z.; Zhang, S. *Deep Reinforcement Learning—Fundamentals, Research and Applications*; Springer: Berlin/Heidelberg, Germany, 2020.
10. Liu, L.-L.; Chen, E.-L.; Gao, Z.-G.; Wang, Y. Research on motion planning of seven degree of freedom manipulator based on DDPG. In Proceedings of the Advanced Manufacturing and Automation VIII 8, Changzhou, China, 20–21 September 2018; pp. 356–367.
11. Robotics, C. Robotics Simulator CoppeliaSim. Available online: https://www.coppeliarobotics.com/ (accessed on 1 July 2023).
12. Towers, M.; Terry, J.K.; Kwiatkowski, A.; Balis, J.U.; Cola, G.d.; Deleu, T.; Goulão, M.; Kallinteris, A.; Arjun, K.G.; Krimmel, M.; et al. Gymnasium. Available online: https://github.com/Farama-Foundation/Gymnasium (accessed on 1 July 2023).
13. Raffin, A.; Hill, A.; Gleave, A.; Kanervisto, A.; Ernestus, M.; Dormann, N. Stable-baselines3: Reliable reinforcement learning implementations. *J. Mach. Learn. Res.* **2021**, *22*, 12348–12355.

14. Mnih, V.; Badia, A.P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In Proceedings of the International Conference on Machine Learning, New York, NY, USA, 20–22 June 2016; pp. 1928–1937.
15. Han, D.; Mulyana, B.; Stankovic, V.; Cheng, S. A Survey on Deep Reinforcement Learning Algorithms for Robotic Manipulation. *Sensors* **2023**, *23*, 3762. [CrossRef] [PubMed]