

Kinematic Control of Manipulators using Multi Deep Q-Learning

Estanislau Filho, Mario F. M. Campos and Armando A. Neto

Abstract—In this paper, we investigate the problem of controlling the position and orientation of a manipulator end-effector, in tasks such as pick-and-place or trajectory tracking, using deep reinforcement learning. Unlike other approaches in the literature that control the robot joints using just one learning agent, enabling it to perform the tasks, here we propose to separate each degree of freedom of the manipulator into a different agent, learning distinct policies for each joint. As a consequence, when using discrete action-based RL algorithms, such as DQN, instead of having a large set of possible actions as the output of a single neural network given by the combination of all individual actions in the joints, we have multiple parallel neural networks with a small number of actions for each joint. In other words, we reduce the problem's action space by learning it as a multiagent system. As simulated and real-world experiments demonstrate, our multi-deep reinforcement learning strategy enables faster policy learning than conventional approaches.

I. INTRODUCTION

Over the past decade, the field of robotics has made significant advancements, especially with the spread of Reinforcement Learning (RL) techniques. Tasks that were once complex can now be learned by robots through interaction with the environment, using trial-and-error approaches aimed at maximizing a gain function, known as a *reward*. In this context, Deep Reinforcement Learning (DRL) methods have emerged, leveraging neural networks to approximate policies in complex decision-making processes.

Among these challenging scenarios is the control of robotic manipulators, widely used in various sectors to automate repetitive and monotonous tasks, especially those involving object handling. Although there is extensive literature on the subject, most approaches require precise modeling of the robot and its environment, which can be costly, inefficient, and prone to errors in practice.

DRL methods can address these challenges, but due to the large number of joints and the coupling between them, the state and action spaces grow significantly, making the learning process slow and ineffective.

In this paper we present an application of DRL algorithms to perform joint level control of a manipulator in manipulation tasks. More specifically, we propose the application of a multiagent learning approach, where each agent independently

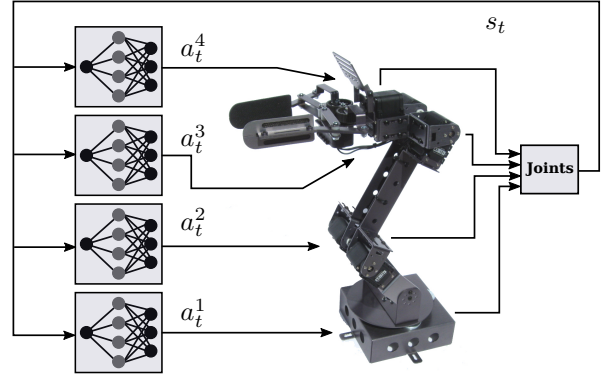


Fig. 1. Multi-DQN: each robot joint is controlled by a different learning agent based on the entire state space, and these policies are trained with a variant of the DQN algorithm.

controls a single joint of the robot, as illustrated in Fig. 1. These agents are trained using the Multi Deep Q-Learning, a variant of DQN algorithm where a single neural network that models the actions for n outputs can be replaced by n smaller neural networks, each handling the actions for a single output. In our case, each of these networks receives the entire state of the manipulator as its input but produces the output for a single joint. In this way, the relationships between the joints will be learned as the robot interacts with the environment.

When compared to other works in the literature that use DRL, our method proved to be faster in terms of the number of episodes required for training and more effective in maximizing the reward function, with greater accuracy at task completion. We also demonstrate that it is possible to apply the model trained in simulation to a real robot in a zero-shot approach, that is, without requiring additional training. This indicates that our technique can be just as generalizable as others in the literature. Furthermore, since our method is based on a discrete-action algorithm, it tends to be simpler than other approaches such as DDPG, SAC, TD3, and others [1].

II. RELATED WORK

The application of DRL methods in robotic manipulators has gained prominence in the literature due to their ability to solve complex tasks and generalize in dynamic scenarios [2], where agents learn to make sequential decisions that maximize rewards through interactions with the environment. These decisions may focus on numerous possibilities such as end-effector trajectory and approach angles, force and torque, or the type of grasping. The action space may be

E. Filho and M. Campos are with the Computer Vision and Robotics Laboratory (VeRLab), Dep. of Computer Science, Universidade Federal de Minas Gerais, Belo Horizonte, Brazil, CEP 31270-901. E-mail: estanislau.filho@dcc.ufmg.br, mario@dcc.ufmg.br.

A. Neto is with the Dep. of Electronics Engineering, Universidade Federal de Minas Gerais, Belo Horizonte, Brazil, CEP 31270-901. E-mails: aaneto@cpdee.ufmg.br

979-8-3315-5288-6/25/\$31.00©2025 IEEE

either discrete or continuous depending on the requirements to be met for the task at hand.

Methods such as Deep Deterministic Policy Gradient (DDPG) [3] and Proximal Policy Optimization (PPO) [4] have proven effective in controlling within continuous action spaces for tasks such as locomotion and object manipulation in three-dimensional space [5], [6], [7], [8]. However, methods that use continuous action spaces often exhibit instabilities and high sensitivity to hyperparameter configurations [9].

Approaches based on the Q -learning algorithm, which utilize discrete action spaces, have been applied to learn control strategies for soft robots, allowing movements to be guided precisely by the distance between the end-effector and the target, even in unstructured environments [10]. The results demonstrate the efficacy and robustness of this method, highlighting its portability and adaptability to different scenarios.

The deep reinforcement learning algorithm Deep Q -Network with Normalized Advantage Functions (DQN-NAF), compared to other algorithms such as Trust Region Policy Optimization (TRPO) and DDPG, has proven to be efficient in learning manipulation tasks [11]. Similarly, the Deep Q -Networks (DQN) algorithm was applied to the Sawyer manipulator with 7 degrees of freedom, using the kinematic model "Product of Exponentials" (PoE) to simplify training [12]. Both algorithms demonstrated robustness and precision in solving manipulation problems in complex scenarios.

The DQN and REINFORCE algorithms were used to control a KUKA KR16 manipulator, positioning the end-effector and avoiding collisions with promising results. DQN stood out for its efficiency and faster training, showing potential to replace manual trajectory programming [13]. Similarly, DQN was applied to manipulators with 7 degrees of freedom, using environmental images to locate and move objects [14]. In both cases, the experiments were conducted in simulation, and the results indicated that the learned policies could be transferred to real robots, demonstrating their effectiveness and scalability for manipulation tasks.

While Deep Q -Learning (DQN) focuses on single-agent learning to estimate the Q -function and determine optimal actions based on the current state, Multi Deep Q -Learning further extends this concept to multi-agent systems, where each agent can have its Q -network. In this setting, agents learn cooperatively or competitively, facing additional challenges due to the inter-dependencies among their actions and the more complex dynamics of the environment, making the learning process significantly more challenging [15].

A multi-agent reinforcement learning (MARL) method was developed to control redundant robots in task space [16]. In this method, each robot joint acts as a cooperative agent, facilitating adaptation to the environment. Tests with simulated planar manipulators with 6 degrees of freedom demonstrated the effectiveness of cooperative agents in inverse kinematics problems, ensuring safe and precise movements in complex tasks [17].

A multiagent reinforcement learning approach based on Deep Q -Network (MADQN) was implemented to control

continuous manipulators with 2 degrees of freedom in minimally invasive medical scenarios [18]. The method includes a shielding strategy that allows for continuous adjustments to action limits, ensuring greater stability and precision when dealing with external interactions such as loads and obstacles. The results demonstrated that the controller achieves sub-millimeter trajectory tracking accuracy, which is an efficient solution for controlling flexible manipulators in dynamic environments.

III. METHODOLOGY

In this section we present our approach to perform the kinematic control of manipulators using a variant of the DQN with multiple neural networks

A. Variant of DQN with Multiple Neural Networks

The Deep Q -Network (DQN) is a well-known extension of the classical Q -Learning algorithm that uses neural networks to address common challenges in problems with large or continuous state spaces. Its goal is to find an "optimal" policy that maximizes the expected value of the accumulated rewards in all states and actions. The DQN introduces two main components to stabilize training: the *Replay Buffer* and the *Target Network*. The Replay Buffer stores the transitions (s_t, a_t, r_t, s_{t+1}) collected during the interaction of the agent with the environment. Random samples are drawn from the buffer during training to compute the loss gradient. This breaks the sequential correlation between the samples and improves the training stability. The Target Network $Q(s_t, a_t, \bar{\theta})$ is a copy of the main network $Q(s_t, a_t, \theta)$, but its parameters are periodically updated to decouple the computation of the main network, thereby reducing training instability.

Based on this, the agent in a state $s_t \in S$ at time t selects and executes an action $a_t \in A$ according to the policy $\pi(s_t)$. After executing the action, the agent receives a reward r_t , and the state transitions to s_{t+1} . The Q -value is updated using the following equation:

$$Q(s_t, a_t, \theta) = r_t + \gamma \max_{a'} Q(s_{t+1}, a', \bar{\theta}), \quad (1)$$

where θ and $\bar{\theta}$ are the parameters of the main network and the target network respectively, and $\gamma \in [0, 1]$ is the discount factor. The DQN loss function is defined as the mean squared error between the Q -value, estimated by the main network, and the target value calculated using the target network:

$$L(\theta) = \mathbb{E} \left[\left(r_t + \gamma \max_{a'} Q(s_{t+1}, a', \bar{\theta}) - Q(s_t, a_t, \theta) \right)^2 \right]. \quad (2)$$

The algorithm also uses an ϵ -greedy policy to balance exploration and exploitation. With probability ϵ , the agent selects a random action; with probability $1 - \epsilon$, it gets the action that maximizes the Q -value estimated by the main network. The objective of the DQN is to learn an approximation of the optimal Q -function, defined as:

$$Q^*(s, a, \bar{\theta}) = \max_{\pi} Q_{\pi}(s, a, \bar{\theta}). \quad (3)$$

To achieve this goal, on the one hand the agent adjusts the parameters θ of the main network to minimize the difference between the estimated Q -values and the target Q -values, trying to lead the learning to converge to the optimal policy π^* . On the other hand, in the Multi Deep Q -Network algorithm, each of the n neural networks receives the same input s_t but produces different outputs corresponding to various control components. Parameter updates are performed independently for each network, allowing them to learn different policies while sharing the same observation space. The algorithm below shows the pseudo-code for a variant of the Deep Q -Network (DQN) algorithm that uses n independent neural networks, each with the same input but different outputs.

Algorithm 1 Multi DQN

Require: n main $Q(s_t, a_t, \theta)$ and target $Q(s_t, a_t, \bar{\theta})$ networks, n memory replays $\mathcal{D}$, $\alpha > 0$, $\varepsilon > 0$, and $\tau \ll 1$

- 1: initialize $\mathcal{D}^i$ for all $i = 1, \dots, n$
- 2: initialize θ^i and $\bar{\theta}^i$ arbitrarily for all $i = 1, \dots, n$
- 3: **for** each episode **do**
- 4: initialize s_0
- 5: **repeat**
- 6: **for** each network θ^i **do**
- 7: select a_t^i from s_t (ε -greedy)
- 8: **end for**
- 9: execute $a_t = [a_t^1, \dots, a_t^n]$, receive s_{t+1}, r_t
- 10: **for** each replay buffer $\mathcal{D}^i$ **do**
- 11: store $\langle s_t, a_t^i, s_{t+1}, r_t \rangle$ in $\mathcal{D}^i$
- 12: **end for**
- 13: **for** each network θ^i **do**
- 14: randomly select a mini-batch $\mathcal{D}^j \in \mathcal{D}^i$
- 15: $y_j \leftarrow \begin{cases} r_t^j & \text{if } s_{t+1}^j \text{ is terminal,} \\ r_t^j + \gamma \max_a Q(s_{t+1}^j, a_t^j, \bar{\theta}^i) & \text{otherwise} \end{cases}$
- 16: $\theta^i \leftarrow \theta^i + \alpha [y_j - Q(s_t^j, a_t^j, \theta^i)]^2 \nabla Q(s_t^j, a_t^j, \theta^i)$
- 17: $\bar{\theta}^i \leftarrow (1 - \tau) \bar{\theta}^i + \tau \theta^i$
- 18: **end for**
- 19: $s_t \leftarrow s_{t+1}$
- 20: **until** $s_t = \text{terminal}$
- 21: **end for**

Here we have n main neural networks, one for each degree of freedom we want to control and their respective target networks. Furthermore, since each network learns independently, we also need n replay buffers $\mathcal{D}$ to store the experiences of each agent. Also, as in the classic DQN, α represents the learning rate, ε is the exploration parameter, and τ is the soft-update constant.

After the appropriate initializations, we use neural networks to select the action a_t^i of each agent i , concerning a ε -greedy exploratory policy. Then, all actions of each degree of freedom are composed into an action vector a_t , which is applied to the environment. This, in turn, returns the new state of the system s_{t+1} and the reward r_t obtained.

In our case, the state space is shared with both single-agent and multi-agent approaches, and was defined to

capture essential information for the efficient control of the manipulator. It consists of the joint angles of the manipulator (in one of our cases there were four joints), the position of the end-effector and the target, the orientation of the end-effector and the target, the difference between the position of the end-effector and the target, as well as the difference between the orientation of the end-effector and the target. This comprehensive representation allows both single-agent and multi-agent approaches to have a detailed view of the manipulator's current configuration relative to the objective, enabling effective decision-making to align the position and orientation of the end-effector with the target in a precise and efficient manner.

In the single-agent approach, the action space was defined based on incremental changes in the manipulator joint angles. For each joint, the possible actions consist of increasing the angle, decreasing the angle, or keeping it unchanged. The increment used for these changes was fixed at 0.035 radians, uniformly applicable to all joints. Considering that the manipulator used in this example has four joints, there are 81 possible combinations of actions as output from the neural network.

In the multi-agent approach, using the Multi Deep Q -Learning strategy, each manipulator joint is treated as an independent agent, enabling more detailed and distributed control. Unlike the single-agent approach, the increments in joint angles are adjusted individually for each joint, reflecting the specific characteristics of each one. The increment defined for joint 0 was 0.035 radians, while joints 1, 2, and 3 had increments of 0.025, 0.026, and 0.027 radians, respectively. Each agent has three possible actions: increase the angle, decrease the angle, or keep it unchanged.

B. Reward Function

To evaluate the accuracy of position and orientation control of the manipulator, a total reward metric was used, which combines the position and orientation errors of the end-effector relative to the target. Initially, the position (p_{error}) and orientation errors (R_{error}) are obtained from the pose difference calculated via forward kinematics. The orientation error is derived from the rotation matrix R_{error} , converted into minimal representation angles in Roll-Pitch-Yaw. The position error is measured by the Euclidean norm of the translation vector p_{error} :

$$\|p_{\text{error}}\|_2 = \sqrt{p_x^2 + p_y^2 + p_z^2} \quad (4)$$

The position and orientation errors are then weighted by specific values, w_{position} and $w_{\text{orientation}}$, which control the relative importance of accuracy in position and orientation, respectively. Thus, the total reward metric is calculated as:

$$r_{\text{total}} = (w_{\text{position}} \cdot r_{\text{position}}) + (w_{\text{orientation}} \cdot r_{\text{orientation}}) \quad (5)$$

In the experiment, the weights were defined as $w_{\text{position}} = 0.90$ and $w_{\text{orientation}} = 0.10$, prioritizing the accuracy of the end-effector's position over its orientation. This approach

allows for flexible control adjustments, focusing on the most critical aspects of the task that the manipulator must perform.

The variables r_{i-1} and r_i represent the total rewards of the previous and current iterations, respectively. These values are fundamental for computing the reward difference between two consecutive iterations, which reflects the agent's progress toward the desired objective. This difference is used as the basis for determining the agent's final reward.

When the agent selects a good action, it leads to an improvement in the reward, as the reward difference between iterations results in a positive value. Conversely, a deterioration leads to a penalty, encouraging the agent to adopt actions that maximize convergence to the objective. This mechanism promotes precise and consistent adjustments in the agent's behavior throughout the training process.

The final reward is calculated using a sigmoidal function adapted to the interval $[-1, 1]$, allowing for the smoothing and the limiting the reward values. This approach ensures a gradual transition between gains and penalties, guaranteeing stability of the learning process. The final reward function is defined by

$$R = \left(2 \cdot \frac{1}{1 + e^{-x}} - 1 \right), \quad (6)$$

where the value of x is given by:

$$x = -\frac{(r_{i-1} - r_i)}{r_g} \cdot k.$$

In this case, x represents the difference between the rewards from the previous iteration (r_{i-1}) and the current iteration (r_i), normalized by the desired objective (r_g) and adjusted by a scaling factor k , which controls the slope of the sigmoid curve. This formulation ensures that reward improvements result in positive values, while degradations produce negative values, encouraging the agent to consistently maximize its performance over iterations.

IV. EXPERIMENTS AND RESULTS

Experiments using the Deep Q-Network and Multi Deep Q-Network algorithms were performed both in simulation and implemented on a real manipulator to compare the performances of single-agent and multi-agent approaches. For the real robot we used the CrustCrawler AX-12, a manipulator with four degrees of freedom (DOF), depicted in Fig. 1. It receives commands via a serial protocol which encodes position, velocity, and torque control that are parsed and executed by microcontroller.

The manipulator's kinematic model was developed based on the Denavit-Hartenberg notation [19] using a coordinate system associated with the manipulator's joints. The D-H parameters are shown in Tab. I. With this information, it is possible to solve the inverse kinematics and calculate the joint angles required to achieve the desired pose of the end-effector.

The position and orientation of the end-effector, like the manipulator's links, are described using homogeneous transformations that define the relationship of the i -th link relative to the $i - 1$ -th link in three-dimensional space. The final homogeneous transformation matrix describes the

TABLE I
CRUSTCRAWLER AX-12 DENAVIT-HARTENBERG PARAMETERS.

Link	l_i (cm)	λ_i (cm)	α_i (deg)	Θ_i (deg)
1 st	5.51	9.21	90	$0 \div 360$
2 nd	17.3	0	0	$0 \div 90$
3 rd	2.0	0	90	$0 \div 180$
4 th	0	6.8	0	$-180 \div 180$

position and orientation of the end-effector frame relative to the base frame. The pose error between the end-effector and the target is expressed by

$$T_{\text{diff}} = T_e^{-1} \cdot T_{\text{target}}, \quad (7)$$

where T_e^{-1} represents the inverse matrix that transforms the end-effector coordinates to the global reference frame, while T_{target} is the homogeneous transformation matrix associated with the target. The resulting homogeneous transformation matrix T_{diff} contains both the translational error p_{error} and the angular error R_{error} .

Since the CrustCrawler AX-12+ has only four degrees of freedom, complete orientation control is not possible. Therefore, only the rotation around the z -axis is considered to correct the orientation, and the angular error is given by:

$$\psi = \arctan 2(r_{21}, r_{11}), \quad (8)$$

which converts the matrix

$$R_{\text{error}} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

into minimal representation angles in the ZYX convention, where ψ (yaw) represents the only corrected rotation around the z -axis.

To identify the most efficient neural network architecture and adjust the corresponding hyper-parameters, 135 distinct neural network configurations were initially evaluated. The tests considered both single-agent and multi-agent approaches, aiming at identifying the most suitable combination for controlling the pose of the robotic manipulator toward Target 1, as defined in Table II.

TABLE II
DEFINITION OF ALL TARGET LOCATIONS.

Target	x (cm)	y (cm)	z (cm)	ϕ (rad)	θ (rad)	ψ (rad)
1	+20.0	0.0	+2.0	-0.86π	0.0	0.0
2	+15.0	-8.0	+10.0	$+1.14\pi$	0.0	0.0
3	+17.0	+5.0	+16.0	$+\pi$	-0.10π	$+0.35\pi$
4	+10.0	-12.0	+2.0	$+1.14\pi$	0.0	0.0
5	+17.0	+6.0	+2.0	$+\pi$	0.0	0.0

Each training session consisted of 1000 episodes and 200 steps per episode. The architecture and hyper-parameters were adjusted distinctly for the single-agent and multi-agent approaches, aiming at optimizing performance in each scenario. In the single-agent approach, the best configuration consisted of a network with 256 neurons in each of its three hidden layers, whereas in the multi-agent approach, each agent used a network with 64 neurons in the same two hidden layers.

For both approaches, the hyper-parameters were defined as $\gamma = 0.90$, a batch size of 128, and $\tau = 0.005$. Also, we set $\alpha = 0.00025$ for the single-agent approach and $\alpha = 0.0025$ for the multi-agent approach.

Figure 2 presents the accumulated average reward graph over episodes for two approaches: single-agent (blue line) and multi-agent (red line) during training. In the initial episodes, both scenarios show a sharp increase in average reward, indicating that the agents are quickly learning to improve their performance. However, the multi-agent scenario demonstrates faster convergence and achieves a higher average reward plateau, stabilizing around 5000, while the single-agent scenario stabilizes at approximately 4000. This difference suggests that the multi-agent approach enables greater learning efficiency due to collaboration among agents.

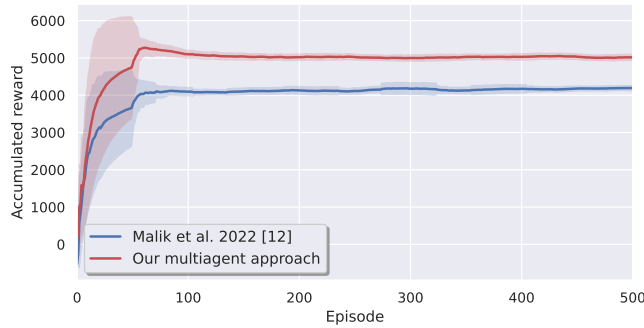


Fig. 2. graph of average accumulated rewards for multi-agent vs single-agent approach

After defining the best neural network architecture and its respective hyper-parameters based on Target 1 (as shown in Table II), new training sessions were conducted considering the remaining four targets. These targets were positioned at different locations within the manipulator's workspace to evaluate the network's ability to generalize pose control across various scenarios. This approach allowed testing the robustness of the network in varied situations, ensuring its applicability under different operating conditions.

The best neural network architecture and its respective hyper-parameters, previously determined, were also used to evaluate the network's performance in a more complex task: the "pick and place" task. In this experiment, 1000 episodes were conducted, with 200 steps per episode. The initial position of the target object and its destination were kept fixed, corresponding to the coordinates of Target 4 and Target 5, as specified in Table II. This scenario was designed to test the network's ability to handle tasks that require greater precision and planning.

Figure 3 shows the accumulated average reward during training for two approaches: single-agent (blue line) and multi-agent (red line) in the "pick and place" task. The multi-agent approach consistently outperforms the single-agent approach, achieving higher rewards and demonstrating superior efficiency and adaptability.

To validate the trained models in each of the previous tasks,

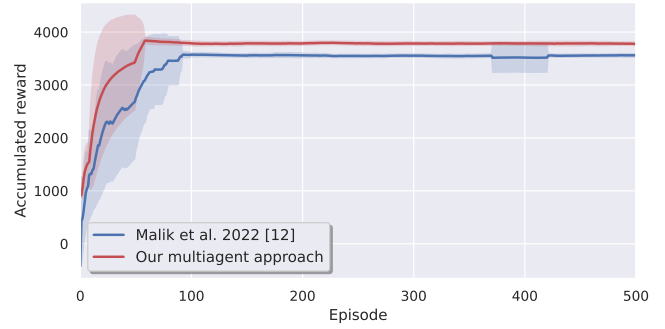


Fig. 3. graph of average accumulated rewards for multi-agent vs single-agent approach

the neural networks were loaded and tested in simulations with specific scenarios. For each task, 100 test runs were performed, with each run consisting of 100 episodes with 200 steps. These tests were conducted to evaluate the model's performance in terms of consistency and accuracy, analyzing its ability to generalize the knowledge acquired during training to new scenarios presented in the testing phase.

Table III presents the results of the tests conducted to compare the performance of single-agent and multi-agent approaches for pose control across five different target positions within the manipulator's workspace. The indicators include accuracy (Acc) (%), position error (cm), and orientation error (rad). All values in the table are placeholders set to zero, which can be replaced with actual results obtained from the experiments.

TABLE III
VALIDATION OF PERFORMANCE METRICS FOR SINGLE-AGENT AND MULTI-AGENT APPROACHES ACROSS TARGETS IN SIMULATION.

Targets	Agent	Acc (%)	P_{error} (cm)	O_{error} (rad)
1	[12]	100.0	0.805	2.96×10^{-2}
	Ours	100.0	0.432	4.13×10^{-2}
2	[12]	98.71	0.608	2.92×10^{-2}
	Ours	100.0	0.584	3.79×10^{-2}
3	[12]	66.2	0.436	5.28×10^{-2}
	Ours	99.99	0.921	7.30×10^{-3}
4	[12]	73.94	0.898	2.33×10^{-3}
	Ours	99.93	0.306	4.52×10^{-2}
5	[12]	100.0	0.904	3.02×10^{-3}
	Ours	100.0	0.933	9.06×10^{-3}

Table IV presents the results of the tests conducted with models trained to perform the "pick and place" task using both single-agent and multi-agent approaches. These tests evaluated the performance of the models by applying the knowledge acquired during the training phase to a specific scenario of the pick and place task. The metrics include accuracy (%), average position error (cm), and average orientation error (rad).

After training and validating the models in simulation, experiments were conducted using the real manipulator to evaluate the direct transfer of learning obtained with the virtual environment. This approach, known as "zero-

TABLE IV

VALIDATION RESULTS COMPARING SINGLE-AGENT AND MULTI-AGENT APPROACHES FOR THE "PICK AND PLACE" TASK.

Approach	Acc (%)	P_{error} (cm)	O_{error} (rad)
[12]	98.67	0.6272	3.75×10^{-3}
Ours	99.78	0.7202	7.71×10^{-3}

shot," involves implementing a policy trained exclusively in simulation directly on the real manipulator, without additional adjustments. The objective of these experiments was to evaluate the model's ability to generalize pose control across five different target positions and to perform the "pick and place" task in the real world.

Table V presents the results obtained for pose control, considering the same five target positions defined in Table II during simulation tests.

TABLE V

PERFORMANCE METRICS FOR SINGLE-AGENT AND MULTI-AGENT APPROACHES ACROSS TARGETS ON THE REAL MANIPULATOR.

Targets	Agent	Acc (%)	P_{error} (cm)	O_{error} (rad)
1	[12]	100.0	0.436	8.52×10^{-3}
	Ours	100.0	0.525	4.46×10^{-3}
2	[12]	90.0	0.745	3.23×10^{-2}
	Ours	100.0	0.515	3.46×10^{-2}
3	[12]	70.0	0.872	4.85×10^{-2}
	Ours	90.0	0.686	4.18×10^{-3}
4	[12]	80.0	0.628	4.26×10^{-2}
	Ours	90.0	0.425	4.76×10^{-2}
5	[12]	100.0	0.874	4.56×10^{-2}
	Ours	100.0	0.633	4.80×10^{-2}

These experiments demonstrate the efficacy of the policy trained in simulation when applied directly to the real manipulator, validating the potential of the "zero-shot" approach for robotic systems in real-world scenarios.

All experiments were conducted on a workstation equipped with an Intel XEON E5-2630 2.4GHz processor with 32 threads, 256 GB of RAM, and a GeForce GTX Titan X 12GB graphics card, which hosted the simulation of the Multi Deep Q-Learning algorithm developed in Python.

The videos of the tests with the real manipulator can be accessed at: this link.

V. CONCLUSION AND FUTURE WORK

This work presented a novel approach to controlling robotic manipulators using deep reinforcement learning in a multi-agent manner. Separating the control of each joint into independent agents reduces the action space, facilitating the learning of effective policies and enabling the application of algorithms based on discrete actions, such as DQN. The results obtained, both in simulations and in the real world, demonstrate that the proposed multi-agent strategy not only accelerates the learning process but also offers robust and efficient performance in complex tasks, such as pick-and-place and trajectory tracking.

This approach highlights the potential of multi-agent reinforcement learning to advance the autonomous control of

robotic manipulators, providing a solid foundation for future research and applications. For example, it is possible to extend the method to learn more complex tasks, such as multiple robots coordination in cooperative manipulation. We can also include aspects of system dynamics to perform torque control on joints, improving the accuracy of the results.

REFERENCES

- [1] D. Han, B. Mulyana, V. Stankovic, and S. Cheng, "A survey on deep reinforcement learning algorithms for robotic manipulation," *Sensors*, vol. 23, no. 7, p. 3762, Apr. 2023.
- [2] S. Amariyoti, "Deep reinforcement learning for robotic manipulation-the state of the art," 2017.
- [3] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.
- [4] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.
- [5] F. R. Campos, A. X. Fidencio, J. Domingues, G. Pessin, and G. Freitas, "Application of reinforcement learning to the orientation and position control of a 6 degrees of freedom robotic manipulator," in *2022 Latin American Robotics Symposium (LARS), 2022 Brazilian Symposium on Robotics (SBR), and 2022 Workshop on Robotics in Education (WRE)*. IEEE, Oct. 2022.
- [6] Z. Guo, J. Huang, W. Ren, and C. Wang, "A reinforcement learning approach for inverse kinematics of arm robot," in *Int. Conf. on Robotics, Control and Automation*, ser. ICRA 2019. ACM, July 2019, p. 95–99.
- [7] R. Zeng, M. Liu, J. Zhang, X. Li, Q. Zhou, and Y. Jiang, "Manipulator control method based on deep reinforcement learning," in *2020 Chinese Control And Decision Conf. (CCDC)*. IEEE, Aug. 2020, p. 415–420.
- [8] C. Zhao, Y. Wei, J. Xiao, Y. Sun, D. Zhang, Q. Guo, and J. Yang, "Inverse kinematics solution and control method of 6-degree-of-freedom manipulator based on deep reinforcement learning," *Scientific Reports*, vol. 14, no. 1, May 2024.
- [9] E. Nikishin, P. Izmailov, B. Athiwaratkun, D. Podoprikin, T. Garipov, P. Shvechikov, D. Vetrov, and A. G. Wilson, "Improving stability in deep reinforcement learning with weight averaging," in *Uncertainty in artificial intelligence workshop on uncertainty in Deep learning*, 2018.
- [10] X. You, Y. Zhang, X. Chen, X. Liu, Z. Wang, H. Jiang, and X. Chen, "Model-free control for soft manipulators based on reinforcement learning," in *2017 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*. IEEE, Sept. 2017.
- [11] A. Franceschetti, E. Tosello, N. Castaman, and S. Ghidoni, *Robotic Arm Control and Task Training Through Deep Reinforcement Learning*. Springer Int. Publishing, 2022, p. 532–550.
- [12] A. Malik, Y. Lischuk, T. Henderson, and R. Prazenica, "A deep reinforcement-learning approach for inverse kinematics solution of a high degree of freedom robotic manipulator," *Robotics*, vol. 11, no. 2, p. 44, Apr. 2022.
- [13] L. P. Cotrim, M. M. José, and E. L. L. Cabral, "Reinforcement learning control of robot manipulator," *Revista Brasileira de Computação Aplicada*, vol. 13, no. 3, p. 42–53, Oct. 2021.
- [14] S. James and E. Johns, "3d simulation for robot arm control with deep q-learning," 2016.
- [15] L. Buşoni, R. Babuška, and B. De Schutter, *Multi-agent Reinforcement Learning: An Overview*. Springer Berlin Heidelberg, 2010, p. 183–221.
- [16] A. Perrusquia, W. Yu, and X. Li, "Multi-agent reinforcement learning for redundant robot control in task-space," *Int. Journal of Machine Learning and Cybernetics*, vol. 12, no. 1, p. 231–241, July 2020.
- [17] Y. Ansari, E. Falotico, Y. Mollard, B. Busch, M. Cianchetti, and C. Laschi, "A multiagent reinforcement learning approach for inverse kinematics of high dimensional manipulators with precision positioning," in *2016 6th IEEE Int. Conf. on Biomedical Robotics and Biomechanics (BioRob)*. IEEE, June 2016, p. 457–463.
- [18] G. Ji, J. Yan, J. Du, W. Yan, J. Chen, Y. Lu, J. Rojas, and S. S. Cheng, "Towards safe control of continuum manipulator using shielded multiagent reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, p. 7461–7468, Oct. 2021.
- [19] J. Denavit and R. S. Hartenberg, "A kinematic notation for lower-pair mechanisms based on matrices," *Journal of Applied Mechanics*, vol. 22, no. 2, pp. 215–221, 1955.