Reinforcement Learning for Shoulder Control of the NAO Robot Using Q-Learning: Tests in Real and Simulated Environments

Vitor Amadeu Souza[1], **Hebert Azevedo Sá**[1]

[1]LIARC – Instituto Militar de Engenharia (IME)
22.290-270 – Rio de Janeiro – RJ – Brazil

vitor.souza@ime.eb.br, azevedo@ime.eb.br

Abstract. *This paper presents a Q-Learning approach for controlling the NAO humanoid robot's ShoulderPitch joint, motivated by the limitations of traditional PID controllers in handling nonlinearities and inter-joint coupling. The methodology involves training in CoppeliaSim simulation before deployment on the physical robot, with a reward function based on absolute angular error relative to a 45° target and discretized state space. Results demonstrate policy convergence in simulation and real-world implementation, with the performance gap providing insights into sim-to-real transfer challenges and establishing a foundation for future multi-joint applications.*

1. Introduction

Humanoid robotics has experienced unprecedented growth in recent years, with applications ranging from service robotics to human-robot interaction and industrial automation [Kober et al. 2013, Gouaillier et al. 2009]. The NAO humanoid robot, developed by SoftBank Robotics, has emerged as a prominent platform for research and education due to its accessibility, robust design, and comprehensive sensor suite [Shamsuddin et al. 2011]. However, achieving precise and adaptive control of humanoid robots remains a significant challenge, particularly when transitioning from simulation environments to real-world applications.

Traditional control approaches for humanoid robots typically rely on classical methods such as PID controllers, which, while effective for certain applications, often lack the adaptability required for complex, dynamic environments [González-Fierro et al. 2014]. The inherent nonlinearities, uncertainties, and disturbances present in real-world scenarios necessitate more sophisticated control strategies that can learn and adapt to changing conditions. This has led to increased interest in machine learning approaches, particularly reinforcement learning (RL), for robotic control applications.

Reinforcement learning has demonstrated remarkable success in various robotic applications, from bipedal locomotion [Kuindersma et al. 2016, Tedrake et al. 2004] to manipulation tasks [Levine et al. 2016]. Recent advances in deep reinforcement learning have enabled humanoid robots to learn complex behaviors directly from experience, showing promising results in both simulated and real-world environments [Mnih et al. 2015]. However, the application of RL to specific joint control problems in humanoid robots, particularly with emphasis on simulation-to-reality transfer, remains an active area of research.

The simulation-to-reality gap represents one of the most significant challenges in robotic learning. Policies learned in simulation often fail to perform adequately when deployed on real robots due to differences in dynamics, sensor noise, actuator limitations, and environmental conditions. This challenge is particularly pronounced in humanoid robotics, where the complexity of the system and the need for real-time control create additional constraints.

Q-Learning, as a model-free reinforcement learning algorithm, offers several advantages for robotic control applications. Its ability to learn optimal policies without requiring explicit knowledge of the system dynamics makes it particularly suitable for complex robotic systems where accurate modeling is challenging [Watkins and Dayan 1992, Sutton and Barto 2018]. Furthermore, Q-Learning's convergence properties and relative simplicity make it an attractive choice for real-time control applications.

This work addresses the specific problem of precise shoulder joint control in the NAO robot using Q-Learning, with particular emphasis on validating the approach across both simulated and real environments. The primary contributions of this research include: (1) development of a Q-Learning-based control strategy specifically tailored for NAO robot shoulder positioning, (2) comprehensive validation in CoppeliaSim simulation environment with systematic parameter tuning, (3) successful implementation and evaluation on the physical NAO robot, and (4) detailed comparison between simulation and real-world performance, providing insights into the simulation-to-reality transfer process.

2. NAO Robot

The NAO is a humanoid robot widely used in research and education [Gouaillier et al. 2009]. Standing at 58 cm tall and weighing approximately 5.48 kg, the NAO has 25 degrees of freedom (DOF), including 2 DOF in the neck, 5 DOF in each arm, 1 DOF in each hand, 5 DOF in each leg, and 1 DOF in the pelvis [Shamsuddin et al. 2011], as shown in Figure 1.

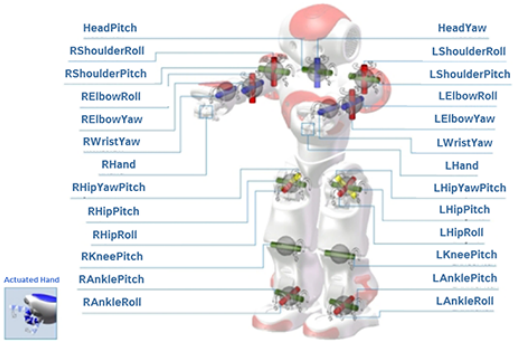

Figure 1. NAO Robot with 25 DOF.
[Aldebaran n.d.a]

The NAO's shoulder joints are particularly important for object manipulation and body language expression. Each shoulder has 2 degrees of freedom (DOF): pitch (forward and backward movement) and roll (lateral movement). The internal and external rotation of the arm, known as yaw, is performed by the ElbowYaw joint located at the elbow

[Shamsuddin et al. 2011]. These joints are driven by DC electric motors equipped with encoders for position feedback [Gouaillier et al. 2009].

The NAO's architecture consists of a distributed control system, where each joint is managed by a dedicated microcontroller that communicates with the main processor via a data bus [SoftBank Robotics 2014]. This processor runs the NAOqi operating system, which provides a development platform for controlling the robot at various levels of abstraction [Pot et al. 2009], as illustrated in Figure 2.

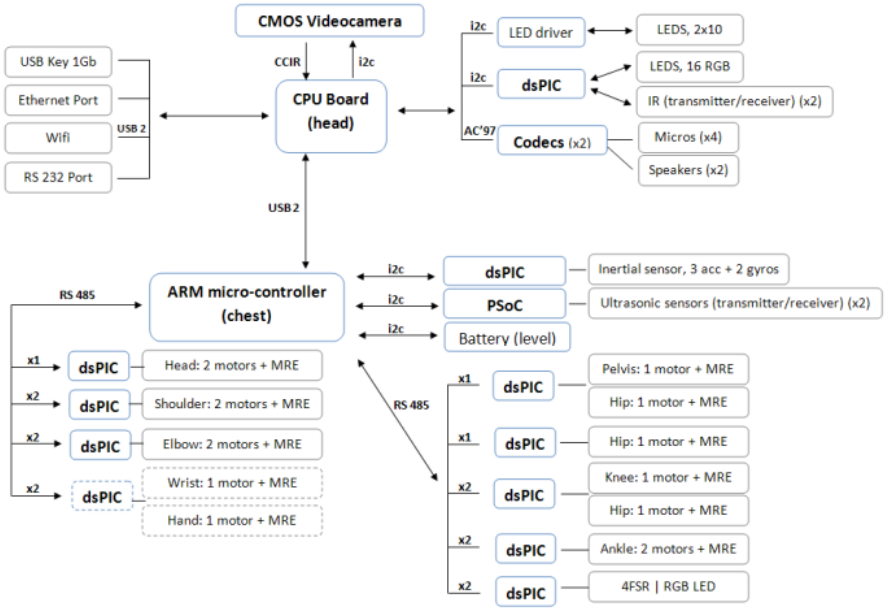

Figure 2. Internal Architecture of the NAO Robot
[Aldebaran n.d.a]

The dsPIC modules control the motors and sensors distributed across the robot's body, including the head, shoulders, elbows, hips, knees, and ankles. This hybrid architecture enables the NAO robot to perform a wide range of movements and interactions with its environment.

3. CoppeliaSim

CoppeliaSim is a robotic simulation environment that enables the development, testing, and validation of control algorithms before their implementation on physical robots [Rohmer et al. 2013]. The simulator provides a flexible platform with a realistic physics engine, allowing the simulation of dynamic properties of robotic joints, such as inertia, friction, and torque limits.

CoppeliaSim supports multiple programming APIs, including Python, which facilitates the implementation of reinforcement learning algorithms [Rohmer et al. 2013]. Additionally, the simulator allows the creation of accurate models of existing robots, such as the NAO, with faithful representations of their physical and kinematic characteristics. Figure 3 shows the simulator's interface with the NAO robot integrated into it.

The use of simulation before implementation on physical robots is a common practice in robotics, as it allows for the safe and efficient development and testing of

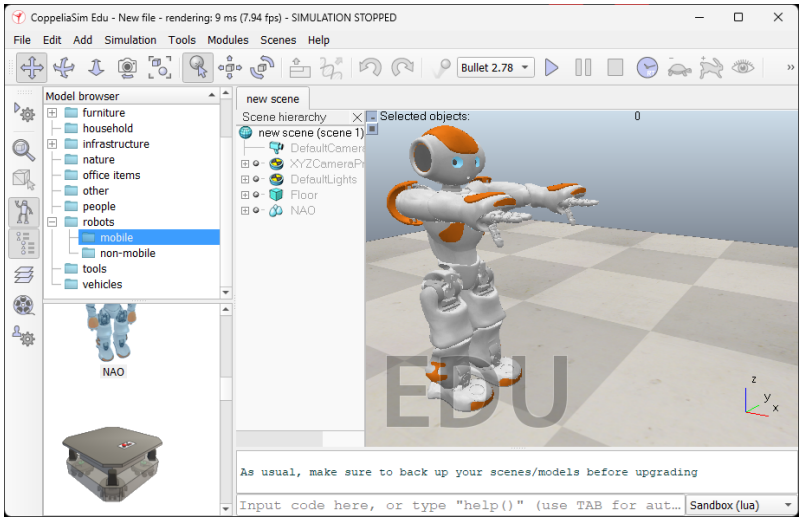

Figure 3. CoppeliaSim

algorithms [Kober et al. 2013]. In the context of reinforcement learning, simulation is particularly valuable, as it enables the collection of large amounts of training data without causing wear on physical hardware.

4. Reinforcement Learning

Reinforcement Learning (RL) is a machine learning paradigm where an agent learns to make decisions through interaction with an environment [Sutton and Barto 2018]. The agent's goal is to maximize the cumulative reward over time by learning through trial and error which actions lead to higher rewards in different states of the environment.

The interaction between the agent and the environment in reinforcement learning follows a continuous cycle, as illustrated in Figure 4. In this cycle, the agent observes the current state S_t of the environment and chooses an action A_t based on its policy. The environment, in turn, responds by providing a reward R_{t+1} and transitioning to a new state S_{t+1}. This process repeats, allowing the agent to learn the optimal policy through trial and error [Sutton and Barto 2018]. In the context of Q-Learning, this cycle is fundamental, as the agent iteratively updates the function $Q(s, a)$ based on the received rewards and future states, as described in the next subsection.

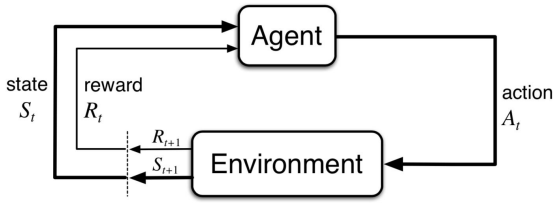

Figure 4. Reinforcement learning interaction cycle: agent observes state S_t, takes action A_t, and environment returns reward R_{t+1} and next state S_{t+1} [Sutton and Barto 2018].

Formally, an RL problem is modeled as a Markov Decision Process (MDP), defined by a tuple (S, A, P, R, γ), shown in Table 1. The agent's goal is to learn a policy

π: S $\rightarrow$ A that maximizes the expected accumulated reward. In robotic systems, RL has been successfully applied to learn control policies for complex tasks such as bipedal locomotion [Tedrake et al. 2004], object manipulation [Levine et al. 2016], and joint control [Kober et al. 2013].

Table 1. Components of the Markov Decision Process (MDP) used in Q-Learning.

Term	Description
S	Set of possible states
A	Set of possible actions
$P(s'\|s,a)$	State transition function defining the probability of moving to state s' given current state s and action a
$R(s,a,s')$	Reward function defining the reward received after transitioning from s to s' due to action a
$\gamma \in [0,1]$	Discount factor weighting future rewards

5. Methodology

For the simulation of the NAO robot, we utilized the model available in the CoppeliaSim library, which includes the robot's geometry, kinematics, and dynamics. The model was configured to focus solely on the left shoulder pitch joint, keeping all other joints in fixed positions. The physical parameters of the joint were adjusted to reflect the characteristics of the real robot. The simulator was set to run with a time step of 10 ms, providing a good balance between simulation accuracy and execution speed.

The implementation of the Q-Learning algorithm for controlling the NAO robot's shoulder joint was carried out using Python in conjunction with CoppeliaSim's ZMQ Remote API. The developed code is structured around a main class named `NAORobotController`, responsible for establishing communication with the simulator and implementing the reinforcement learning algorithm.

The controller initialization process begins by establishing a connection with CoppeliaSim and obtaining the handles for the NAO robot and its joints. The Q-table is initialized with zero values for all state-action pairs, structured to store values for 10 discrete states and 3 possible actions. For state space discretization, the code implements a `get_state_index()` function that converts the continuous joint position into a discrete index, normalizing the angle within the range defined by `ANGLE_LIMIT` (90 degrees).

The Q-Learning algorithm is implemented in the `q_learning()` function, which executes the learning process for a predefined number of episodes (100). In each episode, the controller retrieves the current position of the shoulder joint (ShoulderPitch, index 2), selects an action using the ϵ-greedy policy, and executes the action by modifying the joint angle in increments or decrements of 0.1 radians. The new position is then applied to the robot via the `set_joint_positions()` function.

The reward is calculated as the negative of the absolute difference between the current position and the desired position ($r = -|\theta_{desired} - \theta_{current}|$), encouraging the

algorithm to minimize positioning error. The Q-table update follows the standard Q-Learning equation, as shown in Equation 1.

$$Q(s,a) \leftarrow (1-\alpha) \cdot Q(s,a) + \alpha \cdot \left(r + \gamma \cdot \max_{a'} Q(s',a') \right) \tag{1}$$

where α is the learning rate and γ is the discount factor. To balance exploration and exploitation, the code implements a gradual decay of the ϵ value by a factor of 0.995 per episode.

The action space was simplified to five possibilities: decrease the angle by -0.2 rad or -0.1 rad, maintain the current angle (0), or increase the angle by 0.1 rad or 0.2 rad. This simplification enables faster algorithm convergence without significantly compromising control accuracy. The physical limits of the joint are respected using the `np.clip()` function, which constrains values to the range $[-ANGLE\_LIMIT, ANGLE\_LIMIT]$.

In each episode, the code records the total accumulated reward and the final position achieved, allowing for the evaluation of learning progress. At the end of the simulation, the connection with CoppeliaSim is properly closed to release system resources. Figure 3 shows the simulation environment with the NAO robot, where the arm is controlled by RL.

The complete source code, implemented in Python for the simulator with API integration and on the physical robot, is available for consultation at [GitHub]. Additionally, several videos showcasing the control of the NAO robot's arm using the Q-Learning algorithm can be accessed via [YouTube], providing a practical visualization of the obtained results.

6. Algorithm Design

The Q-Learning algorithm for NAO shoulder control, presented in Algorithm 1, implements a tabular reinforcement learning approach with discrete state-action space representation. The algorithm begins by initializing a Q-table with dimensions 10×3, corresponding to 10 discretized angular positions and 3 possible actions. State discretization maps continuous joint angles from the range $[-90, 90]$ to discrete indices, enabling efficient lookup and update operations during learning.

The core learning loop executes episodic training where each episode starts from the current shoulder position and iteratively selects actions using an ϵ-greedy policy. Actions modify the joint angle by discrete increments of ± 0.1 radians or maintain the current position, with resulting angles clamped to respect physical joint limits. The reward signal, computed as the negative absolute angular error from the target position, provides immediate feedback that drives policy improvement through temporal difference updates of the Q-table using learning rate $\alpha = 0.1$ and discount factor $\gamma = 0.9$.

Convergence is achieved through progressive exploration reduction, where ϵ decays from 0.1 to 0.01 over training episodes, shifting the policy from random exploration to greedy exploitation of learned values. Episodes terminate when the angular error falls below 0.5 or after a maximum number of steps, and the final Q-table is saved for deployment on either the simulated or physical robot platform.

Algorithm 1 Q-Learning for NAO Shoulder Control

Require: $\alpha = 0.1$, $\gamma = 0.9$, $\epsilon_{init} = 0.1$, θ_{target}, $N_{episodes}$
Ensure: Converged Q-table for shoulder positioning

$Q \leftarrow$ Initialize(10, 3) with zeros
$\epsilon \leftarrow \epsilon_{init}$
Connect to NAO (CoppeliaSim or physical robot)
$handles \leftarrow$ GetJointHandles()

for $episode = 1$ to $N_{episodes}$ **do**
  $\theta_{current} \leftarrow$ GetPosition(ShoulderPitch)
  $s \leftarrow$ Discretize($\theta_{current}$)

  while not converged **do**
    // ϵ-greedy action selection
    if Random() $< \epsilon$ **then**
      $a \leftarrow$ RandomAction($\{0, 1, 2\}$)
    else
      $a \leftarrow \arg\max_{a'} Q[s, a']$
    end if

    // Execute action and observe outcome
    $\Delta\theta \leftarrow (a - 1) \times 0.1$
    $\theta_{new} \leftarrow$ Clip($\theta_{current} + \Delta\theta$, -1.57, 1.57)
    SetPosition(ShoulderPitch, θ_{new})

    // Compute reward and next state
    $r \leftarrow -|\theta_{target} - \theta_{new}|$
    $s' \leftarrow$ Discretize(θ_{new})

    // Q-value update
    $Q[s, a] \leftarrow (1 - \alpha)Q[s, a] + \alpha(r + \gamma \max_{a'} Q[s', a'])$

    $s \leftarrow s'$
    $\theta_{current} \leftarrow \theta_{new}$

    if $|\theta_{target} - \theta_{new}| < 0.009$ **then**
      break *// Target reached*
    end if
  end while

  $\epsilon \leftarrow \max(0.01, 0.995 \times \epsilon)$ *// Decay exploration*
end for

SaveQTable(Q)
return Q

7. Results Analysis

The simulation data, conducted in the CoppeliaSim environment, encompass ten runs with a varying number of episodes, ranging from 5 to 27, totaling 122 episodes. Each episode records the total reward, normalized position, angular position (in radians and degrees), action performed, and time (in milliseconds). The total reward is a key performance indicator, reflecting the quality of the Q-Learning agent's actions in controlling the NAO robot's shoulder. Additionally, a file containing the Q-table, which stores the learned Q-values for each state-action pair, was generated, enabling a detailed analysis of the developed policy. The following presents the statistical analysis of the rewards, including mean, median, standard deviation, minimum, and maximum, followed by a graphical analysis of the angular position versus time for Simulation 5, which has 11 episodes, the closest to the average of 12.2 episodes. The discussion interprets these results, contextualizing the agent's performance, the role of the Q-table, and suggesting future directions.

Table 2 summarizes the total reward statistics for each simulation and in aggregate. The global mean of -0.23, with a median of -0.21 and a standard deviation of 0.19, indicates that the agent frequently received negative feedback, typical of the initial phases of reinforcement learning where exploration predominates. Rewards range from 0.00 (optimal actions) to -0.79 (significant challenges), reflecting variability in performance. Simulation 5 stands out with the lowest negative reward mean (-0.16), suggesting more stable learning, possibly due to a more refined Q-table, as recorded in the generated file.

Table 2. Summary of total reward statistics for the simulations.

Run	File	Episodes	Mean	Median	Standard Deviation	Minimum	Maximum
1	SIM1	7	-0.23	-0.19	0.20	-0.49	0.00
2	SIM2	17	-0.21	-0.13	0.18	-0.59	0.00
3	SIM3	27	-0.19	-0.20	0.15	-0.53	0.00
4	SIM4	5	-0.25	-0.26	0.18	-0.48	0.00
5	SIM5	11	-0.16	-0.15	0.17	-0.48	0.00
6	SIM6	8	-0.30	-0.27	0.23	-0.69	0.00
7	SIM7	5	-0.25	-0.26	0.18	-0.48	0.00
8	SIM8	21	-0.24	-0.20	0.24	-0.79	0.00
9	SIM9	16	-0.20	-0.21	0.18	-0.68	0.00
10	SIM10	5	-0.25	-0.26	0.18	-0.48	0.00
Global (All Runs)			-0.23	-0.21	0.19	-0.79	0.00

To assess the evolution of shoulder control, Figure 5 plots the angular position (in degrees, `deg_position`) as a function of time (in milliseconds, `timestamp_ms`) for Simulation 5. The graph shows a downward trend, starting at 76.29 and reaching a minimum of 36.85 at 379ms, with fluctuations that reflect the agent's exploration. The stabilization around 44 to 47 in the final episodes suggests the beginning of convergence, possibly reflected in the generated Q-table values.

The negative mean reward (-0.23) and standard deviation (0.19) indicate that the Q-Learning agent is in an initial learning phase, exploring the state and action space, as evidenced by the Q-table values recorded in the generated file. The downward trend in angular position, observed in Figure 5, suggests that the agent is learning to adjust the shoulder towards a target position. Fluctuations, such as the increase from 36.85 to 47.56, reflect the exploration necessary to refine the Q-table, while stabilization in the final episodes indicates progress in the agent's policy, corroborated by the Q-table's structure. The results align with the theoretical expectations of Q-Learning, where the agent updates

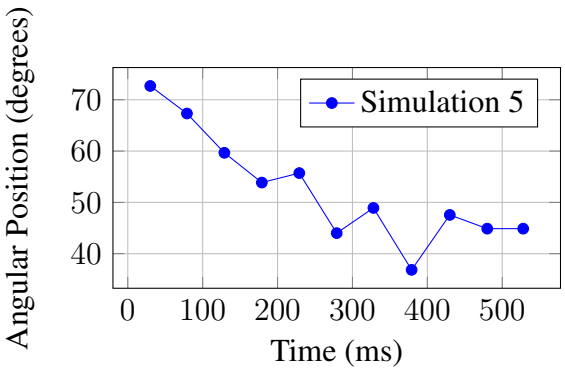

Figure 5. Angular position (degrees) as a function of time (ms) for Simulation 5, with 11 episodes.

its policy based on reward feedback, storing learned information in the Q-table. The Q-table generated in the simulator (`qtablenao.npy`) was used to test the physical NAO robot.

Data collected from the NAO robot, in experiments conducted in a real environment, span ten runs with a variable number of episodes, from 5 to 11, totaling 75 episodes. The same statistical analysis was performed for these results, where Simulation 1, which has 7 episodes, was the closest to the average of 7.3 episodes. Table 3 summarizes the total reward statistics for each run and in aggregate. The global mean of -0.32, with a median of -0.32 and a standard deviation of 0.19, indicates that the agent received predominantly negative feedback, characteristic of the initial phases of reinforcement learning, where exploration is predominant. Rewards vary from -0.01 (actions close to optimal) to -0.74 (significant challenges), reflecting greater variability compared to the simulated results. Simulation 1, with a mean of -0.35, is aligned with the overall performance, while Simulation 9, with a mean of -0.23, suggests more stable learning, possibly reflected in more refined Q-values in the generated file.

Table 3. Statistical summary of total reward in trials with the NAO robot.

Trial	File	Episodes	Mean	Median	Standard Deviation	Minimum	Maximum
1	NAO1	7	-0.35	-0.39	0.19	-0.54	-0.04
2	NAO2	7	-0.34	-0.26	0.22	-0.72	-0.05
3	NAO3	9	-0.38	-0.34	0.19	-0.70	-0.08
4	NAO4	6	-0.33	-0.32	0.24	-0.69	-0.02
5	NAO5	7	-0.24	-0.26	0.15	-0.55	-0.06
6	NAO6	9	-0.44	-0.48	0.23	-0.74	-0.03
7	NAO7	11	-0.36	-0.39	0.16	-0.58	-0.05
8	NAO8	5	-0.25	-0.25	0.17	-0.54	-0.08
9	NAO9	5	-0.23	-0.26	0.18	-0.55	-0.01
10	NAO10	7	-0.28	-0.29	0.16	-0.54	-0.02
Global (All Trials)			-0.32	-0.32	0.19	-0.74	-0.01

To evaluate the evolution of shoulder control, Figure 6 plots the angular position (in degrees, `deg_position`) as a function of time (in milliseconds, `timestamp_ms`) for Simulation 1. The graph shows a downward trend, starting at 76.29 and reaching 44.55 at 8186ms, with fluctuations that reflect the agent's exploration. The progressive reduction suggests learning; however, the lack of clear stabilization indicates challenges in the real-world environment.

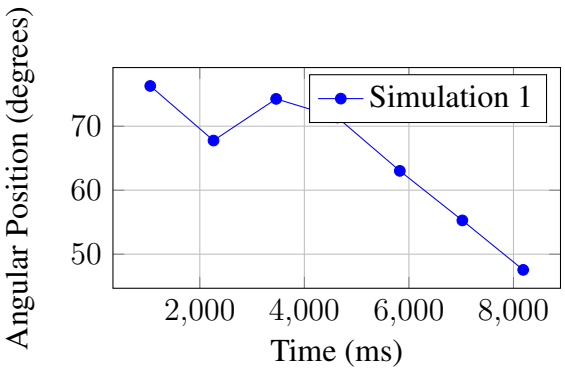

Figure 6. Angular position (degrees) as a function of time (ms) for Simulation 1, with 7 episodes.

The results obtained with the NAO robot reveal a more challenging performance compared to the simulations in CoppeliaSim, where the average reward was -0.23, the median -0.21, and the standard deviation 0.19. In the real environment, the average reward of -0.32 and the same standard deviation of 0.19 indicate greater difficulty for the Q-Learning agent, likely due to sensor noise, delays in action execution, and hardware limitations that introduce uncertainties absent in the simulated environment. The generated Q-table records the learned Q-values, but its convergence is less pronounced in the real-world setup, as suggested by the consistently negative rewards.

The chart for Simulation 1 (Figure6) shows a downward trend in angular position, similar to that observed in Simulation 5 from CoppeliaSim. Fluctuations, such as the increase from 67.76 to 74.27, reflect the agent's intensive exploration, exacerbated by uncertainties in the real environment. The CoppeliaSim simulated environment, free from noise and physical limitations, enabled more consistent learning, with less negative rewards and greater stabilization of angular position. In the real-world setup, the agent faces additional challenges such as actuator communication latency and sensor inaccuracies, which impact the quality of the Q-table.

The results are aligned with the theoretical expectations of Q-Learning, where the agent updates its policy based on reward feedback, stored in the Q-table. The downward trend in angular position is promising, but the negative rewards and lack of stabilization in the real environment highlight the need for further optimizations. Adjustments in hyperparameters, such as learning rate or discount factor, or a more robust reward function, could improve convergence. Approaches such as Deep Q-Networks (DQN) or transfer learning techniques between simulation and the real environment could help mitigate the observed discrepancies. The generated Q-table provides a foundation for future analyses, allowing correlations between Q-values and agent performance. These findings reinforce the complexity of reinforcement learning in real-world environments and contribute to the development of more robust robotic control strategies.

8. Conclusion

This study demonstrated the application of the Q-Learning algorithm for controlling the shoulder joint of the NAO robot, integrating both simulated experiments in CoppeliaSim and real-world trials with the physical robot. The statistical and graphical analyses reveal

that the agent effectively learns through reinforcement, as evidenced by the predominance of negative rewards typical of the initial exploration phase and the gradual trend toward stabilization in angular positions during simulation.

The Q-table generated throughout training played an essential role in policy development, enabling insights into the agent's learning dynamics and convergence behavior in simulation. It allowed for the identification of state-action pairs that were most frequently explored and provided a compact representation of the learned behavior. However, the real-world experiments revealed increased variability and more pronounced challenges, such as sensor noise, actuator latency, physical constraints, and environmental uncertainties, all of which impacted the agent's performance and slowed convergence. These observations underscore the well-known sim-to-real gap, a critical limitation in the deployment of reinforcement learning algorithms on physical robotic platforms.

Despite these challenges, the outcomes validate the potential of Q-Learning for controlling simple robotic joints under uncertain conditions. The results show that even a tabular, model-free reinforcement learning algorithm can achieve a degree of reliable performance, laying the groundwork for more scalable solutions. Furthermore, the work highlights the importance of well-structured state representations and the effect of parameter tuning such as learning rate, discount factor, and exploration rate on training efficiency and policy performance.

Future research should explore advanced reinforcement learning techniques such as Deep Q-Networks (DQN), Double DQN, or other deep reinforcement learning architectures capable of handling continuous and higher-dimensional state and action spaces. Moreover, hybrid approaches that combine model-free and model-based methods could be investigated to improve sample efficiency and decision accuracy.

Incorporating more sophisticated exploration strategies (e.g., entropy-based methods), reward shaping, curriculum learning, and transfer learning methods could further enhance policy robustness and accelerate convergence in real-world settings. The integration of adaptive mechanisms that adjust learning parameters online in response to agent performance may also offer practical benefits in dynamic environments.

Additionally, extending control beyond a single joint to coordinated multi-joint movements introduces new challenges in terms of state space complexity, joint coupling, and synchronization, but is essential for enabling more natural and functional humanoid motions. Conducting long-term evaluations under varying physical conditions, as well as integrating vision and proprioceptive feedback, would provide a more comprehensive validation of the learned policies.

Finally, this work contributes to the growing body of literature on reinforcement learning in humanoid robotics by bridging the gap between theoretical models and practical implementations. It provides a concrete case study of the challenges and successes in applying autonomous learning to embodied agents, offering a foundation for the development of more adaptive, robust, and generalizable robotic control systems in the future.

References

J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

D. Gouaillier et al., "Mechatronic design of NAO humanoid," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2009, pp. 769–774. doi: 10.1109/ROBOT.2009.5152516.

S. Shamsuddin et al., "Humanoid robot NAO: Review of control and motion exploration," in *IEEE International Conference on Control System, Computing and Engineering*, 2011, pp. 511–516.

M. González-Fierro et al., "Full-body postural control of a humanoid robot with both imitation learning and skill innovation," *International Journal of Humanoid Robotics*, vol. 11, no. 2, p. 1450012, 2014.

R. Tedrake, T. W. Zhang, and H. S. Seung, "Stochastic policy gradient reinforcement learning on a simple 3D biped," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2004, vol. 3, pp. 2849–2854.

S. Kuindersma et al., "Optimization-based locomotion planning, estimation, and control design for Atlas," *Autonomous Robots*, vol. 40, no. 3, pp. 429–455, 2016.

S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1334–1373, 2016.

V. Mnih et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3–4, pp. 279–292, 1992.

R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

Aldebaran, "Electronics architecture." [Online]. Available: `http://doc.aldebaran.com/1-14/naoqi/sensors/dcm/low_level_architecture.html`.

SoftBank Robotics, *NAO Technical Documentation*, 2014. [Online]. Available: `http://doc.aldebaran.com/2-1/home_nao.html`.

E. Pot, J. Monceaux, R. Gelin, and B. Maisonnier, "Choregraphe: A graphical tool for humanoid robot programming," in *IEEE International Symposium on Robot and Human Interactive Communication*, 2009, pp. 46–51.

E. Rohmer, S. P. N. Singh, and M. Freese, "V-REP: A versatile and scalable robot simulation framework," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2013, pp. 1321–1326.

Souza and Sá, "Source code." [Online]. Available: `https://github.com/vitor-souza-ime/qlearning_sim_nao`.

Souza and Sá, "Videos." [Online]. Available: `https://www.youtube.com/watch?v=N5S1xrPWhlA&list=PLEqLg1mJU_X3GkVBhJfR67C3XYgVcdefh`.