Revista Brasileira de Computação Aplicada, November, 2021

DOI: 10.5335/rbca.v13i3.12091
Vol. 13, Nº 3, pp. 42–53
Homepage: seer.upf.br/index.php/rbca/index

ISSN 2176-6649

ORIGINAL PAPER

Reinforcement learning control of robot manipulator

Lucas Pereira Cotrim [1], Marcos Menon José [2], Eduardo Lobo Lustosa Cabral[3]

[1,2]Escola Politécnica da Universidade de São Paulo (POLI-USP), [3]Instituto de Pesquisas Nucleares (IPEN)

*lucas.cotrim@usp.br; †marcos.jose@usp.br; ‡elcabral@ipen.br

Received: 2020-12-12. **Revised:** 2021-09-14. **Accepted:** 2021-10-31.

Abstract

Since the establishment of robotics in industrial applications, industrial robot programming involves the repetitive and time-consuming process of manually specifying a fixed trajectory, resulting in machine idle time in production and the necessity of completely reprogramming the robot for different tasks. The increasing number of robotics applications in unstructured environments requires not only intelligent but also reactive controllers due to the unpredictability of the environment and safety measures, respectively. This paper presents a comparative analysis of two classes of Reinforcement Learning algorithms, value iteration (Q-Learning/DQN) and policy iteration (REINFORCE), applied to the discretized task of positioning a robotic manipulator in an obstacle-filled simulated environment, with no previous knowledge of the obstacles' positions or of the robot arm dynamics. The agent's performance and algorithm convergence are analyzed under different reward functions and on four increasingly complex test projects: 1-Degree of Freedom (DOF) robot, 2-DOF robot, Kuka KR16 Industrial robot, Kuka KR16 Industrial robot with random setpoint/obstacle placement. The DQN algorithm presented significantly better performance and reduced training time across all test projects, and the third reward function generated better agents for both algorithms.

Keywords: Artificial Intelligence; Deep Neural Networks; Reinforcement Learning; Robotics.

Resumo

Desde o estabelecimento da robótica em aplicações industriais, a programação de robôs manipuladores envolve o processo repetitivo e demorado de especificação manual de uma trajetória fixa, o que resulta em tempo ocioso de produção e na necessidade de reprogramação completa para diferentes tarefas a serem executadas pelo robô. A tendência de aumento das aplicações da robótica em ambientes não estruturados requer controladores inteligentes e reativos, devido respectivamente à imprevisibilidade do ambiente e a medidas de segurança. Este artigo apresenta uma análise comparativa de duas classes de algoritmos de Aprendizagem por Reforço, iteração de valor (Q-Learning / DQN) e iteração de política (REINFORCE), aplicada à tarefa discretizada de posicionar um manipulador robótico em um ambiente simulado repleto de obstáculos, sem conhecimento prévio das posições dos obstáculos ou da dinâmica do braço do robô. O desempenho do agente e a convergência do algoritmo são analisados sob diferentes funções de recompensa e em quatro projetos de teste cada vez mais complexos: robô 1-DOF, robô 2-DOF, robô industrial Kuka KR16, robô industrial Kuka KR16 com *setpoint* em posição aleatória. O algoritmo DQN apresentou desempenho significativamente melhor e tempo de treinamento reduzido em todos os projetos de teste e a terceira função de recompensa gerou melhores agentes para ambos os algoritmos.

Palavras-Chave: Aprendizado por Reforço; Inteligência Artificial; Redes Neurais Profundas; Robótica.

1 Introduction

The diversity of modern industrial robotics applications requires the emergence of robots with different degrees of autonomy, appropriate for executing different tasks, such as welding, machining, assembly and cargo handling. The development of more sophisticated sensors, along with the increasing computational capacity of controllers and advances in the fields of computational vision and artificial intelligence has shifted the field of robotic manipulators:

Repetitive and fixed pre-programmed routines have given way to flexible and more reactive controllers, capable of dynamically identifying the orientation of workpieces or learning optimal routines directly from data (Rosen, 1999).

This tendency is not limited to robotics. Recent developments in Artificial Intelligence, namely Reinforcement Learning, have been dedicated to training robust models for a wide variety of applications, from economics and finance (Charpentier et al., 2021) to healthcare systems (Coronato et al., 2020). Reinforcement Learning is an increasingly popular field in AI in which an intelligent agent is trained to perform a specific task while maximizing a reward signal (Sutton and Barto, 2018).

This work aims to obtain the optimal reward function formulation and algorithm choice for the task of positioning a simulated KUKA-KR16 industrial robot while avoiding both known and unknown obstacles. The agents are trained through two different reinforcement learning algorithms over successive interactions with an obstacle-filled simulated environment. For training, it is only necessary to provide the initial specification of a reward function, which represents the quality of actions taken by the agent and guides its exploration. After training, the agent is capable of positioning the robot's end effector in generic positions while avoiding obstacle collision-based solely on sensor data from its current pose.

The main contributions of this work are the development of a Reinforcement Learning (RL) framework for robotics applications in MATLAB, including training and visualization modules, and a comparative analysis of standard RL algorithms: episodic REINFORCE and DQN. Different reward functions are tested and the agent's performance is evaluated. The entire project is open source, and all codes can be found in Github Repository

2 State of the Art

The recent development of Reinforcement Learning means that its practical applications are currently mostly restricted to simulation environments for testing and performance validation, such as OpenAI gym (Brockman et al., 2016). There are several Robotic related tasks in OpenAI gym which utilize a physics engine for simulation and collision detection known as MuJoCo (Todorov et al., 2012). The concept of state and action space exploring inherently requires large amounts of data to be processed and training directly in the real world may lead to accidents. Simulation-based training solves both issues by providing a risk-free environment in which the control agent can acquire faster experience.

Several authors have tried to train RL agents in simulated environments and transfer the resulting model directly to real-world applications. James and Johns (2016) were partially successful in the simulation-based training and subsequent model transferring of a DQN agent for controlling a seven DOF robot in a cube locating and lifting task. The work environment was structured in a way that maximizes the similarity with the simulation environment in order to enable model transfer. The resulting RL agent was able to correctly locate the cube when applied directly to the real-world robot, but subtle differences in the environment prevented it from grabbing and lifting it.

One of the biggest challenges associated with implementing Reinforcement Learning in industrial robotics is Low Sampling Efficiency. Most RL algorithms typically require a large volume of training data before optimal policies can be learned, and the generation of data in real-world settings is often impractical, as it requires a long idle time. To work around this problem, hand-crafted specific initial policies that capture the desired behavior are often used. However, this approach conflicts with the main advantage of RL, i.e., the autonomous learning of various behaviors with minimal human intervention. Gu et al. (2017) present an innovative architecture of the DDPG (*Deep Deterministic Policy Gradient*) and NAF (*Normalized Advantage Function*) algorithms in which multiple robots interact with the environment, gain experience according to its current action politics and send data asynchronously to a server that samples transitions and trains a DQN network. This architecture allows the robot to continue interacting with the environment and collecting state transitions while the DQN parameters are updated, promoting scalability for the inclusion of new robots. The authors validated the proposed architecture in learning the task of opening a door by manipulating robots with seven degrees of freedom, and the action policy was obtained without previous demonstrations.

Chen et al. (2019) used a combination of the Distributed Proximal Policy Optimization (DPPO) and DQN algorithms to solve the similar task of positioning a simulated 2-dimensional robot manipulator while avoiding multiple obstacles. The authors showed that the two-step solution of using DPPO to perform obstacle avoidance while a DQN agent performs navigation resulted in better performance than either algorithm individually.

A major issue in path planning tasks for robotic manipulators in unstructured obstacle-filled environments is the blindness of exploration. Common sparse functions that reward an agent's action only when the proposed task is successfully completed, and provide zero information otherwise, can lead to a highly inefficient learning process. In order to solve this, Xie et al. (2019) have developed an azimuth dense reward function that provides feedback to the agent regularly, reducing the number of training epochs and improving learning efficiency.

3 Problem Definition

A typical Reinforcement Learning framework consists of three interacting modules: environment, interpreter and agent. The environment's current condition is captured by an interpreter, which encodes it at time t as a state $\mathbf{s_t}$ and assigns a reward value $\mathbf{r_t}$. The agent, based on the state and reward received by the interpreter, takes an action $\mathbf{a_t}$, which leads to a state transition according to the system dynamics.

The application of this framework to the control of a

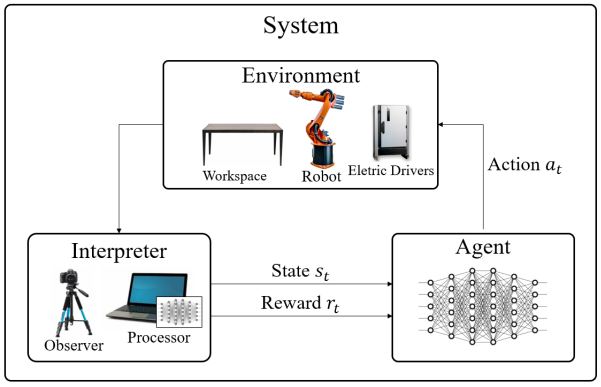


Figure 1: Simplified diagram of a real system environment trained by reinforcement learning.

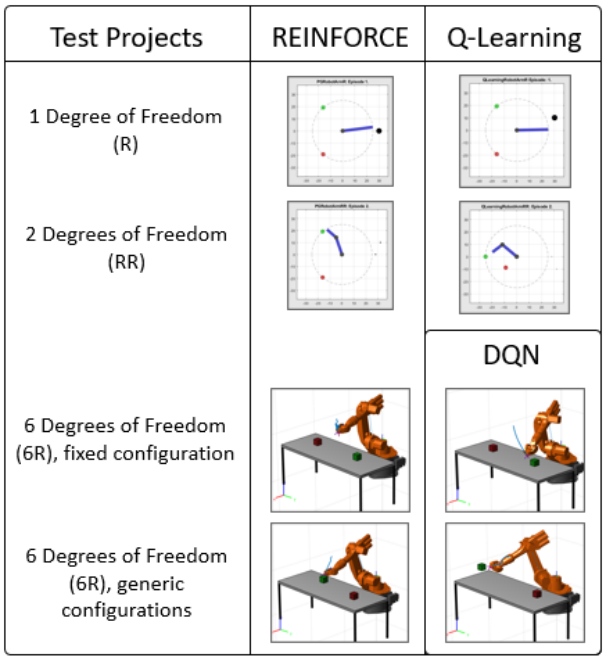


Figure 2: Test projects developed for comparative analysis of algorithms.

robot manipulator is exemplified in the diagram of Fig. 1.

In order to determine the best algorithm, reward function and hyperparameters, simplified versions of the problem were studied under two main classes of algorithms: Iteration over policy function π_θ and iteration over value function $\mathbf{Q}_\theta$. Fig. 2 shows the four test projects considered and the two algorithms implemented for each: Episodic REINFORCE and Q-Learning/DQN.

In the first two simplified projects (1 and 2 DOF robots), the reduced dimension of the state $\mathcal{S}$ and action $\mathcal{A}$ spaces allowed the use of a Q-Learning algorithm known as Q-Tables, in which every possible state-action combination is directly mapped to $\mathbf{Q}(\mathbf{s}, \mathbf{a})$, which is a function of state $\mathbf{s}$ and action $\mathbf{a}$, given by a table. However, due to the increased dimensions of the last two projects, the more sophisticated algorithm DQN was implemented, in which a Feedforward Neural Network approximates the state-action value function $\mathbf{Q}(\mathbf{s}, \mathbf{a})$.

The software chosen was MATLAB because of the support libraries and functions for robotics simulations. This saves time programming both the visualization and the dynamics computation, besides having all the necessary tools of reinforcement learning and neural network implementations. While other programming languages such as Python offer significantly more support for machine learning applications in libraries like Tensorflow and Keras, the available Python libraries for robot kinematics present limited functionality compared to MATLAB.

4 Implemented Algorithms

Reinforcement Learning algorithms can be divided in two major classes: Policy-based and Value-based. The former represents the agent's policy directly and performs updates on it according to the reward obtained by taking different actions in different states. The latter learns a state-action value function $\mathbf{Q}(\mathbf{s}, \mathbf{a})$ instead from which the agent's actions are derived.

In this work, a baseline Deep Reinforcement Learning algorithm of each class was implemented: The Policy-based REINFORCE and the Value-based DQN. However, the developed framework allows for other RL algorithms compatible with discrete action spaces to be implemented, such as A2C (Mnih et al., 2016) and PPO (Schulman et al., 2017).

4.1 First Algorithm: Episodic REINFORCE

The first implemented algorithm is the classic action policy iteration algorithm, episodic REINFORCE proposed by Williams (1992), adapted to the manipulation robot positioning problem according to the pseudocode below. REINFORCE consists in the parameterization of the action policy function $\pi(\mathbf{s})$ as $\pi_\theta(\mathbf{s})$ using any function approximation method, such as neural networks or high-degree polynomials, and training by successive updates of the parameters θ in order to maximize the Performance function $J(\pi_\theta)$, which represents the quality of policy π_θ. Let $\tau = \{s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_{T-1}, a_{T-1}, r_T\}$ be a trajectory generated by a generic policy π_θ, the performance function $J(\pi_\theta)$ can be defined as the expected value of discounted rewards over the trajectory (Eq. (1)).

$$J(\theta) = \mathbf{E}_{\tau\sim\pi(\tau)}[r(\tau)] \tag{1}$$

where $r(\tau) = \sum_{t=1}^{T} \gamma^{t-1} r_t = V_\pi(\mathbf{s_0})$ is equivalent to the value of the initial state $V_\pi(\mathbf{s_0})$ according to policy π_θ. Since knowledge of the environment and reward is gathered through environmental interaction, the gradient of the performance function $\nabla J(\theta)$ must be approximated by a

sufficient number of trajectories N (Eq. (2)).

$$\nabla J(\theta) \approx \frac{1}{N} \sum_{n=1}^{N} \sum_{t=0}^{T-1} G_t \frac{\nabla \pi_\theta(\mathbf{s_t}, \mathbf{a_t})}{\pi_\theta(\mathbf{s_t}, \mathbf{a_t})} \tag{2}$$

where γ is known as the discount factor and the returns $G_t = \sum_{k=0}^{T-t-1} \gamma^k r_{t+k+1} = r_{t+1} + \gamma r_{t+2} + \ldots + \gamma^{T-t-1} r_T$ are defined as the sum of discounted rewards from instant t onward. Historically used to bound the sum of expected rewards of infinite horizon models, the discount factor γ can be interpreted as an interest rate which prioritizes actions with higher immediate rewards while also taking into account future rewards (Kaelbling et al., 1996).

Algorithm 1: Episodic REINFORCE

- Initialize Robot, setpoint, obstacle, initial state $\mathbf{s_0}$ and action space $\mathcal{A}$;
- Initialize Hyperparameters (bonus and penalties, network size, number of timesteps, trajectories and epochs, discount factor γ and learning rate α);
- Initialize data structure to store epochs;
- Initialize parameterized action policy π_{θ_0} randomly;

Generate N trajectories $\{\tau_n\}_{n=1}^{N}$ from action policy π_{θ_0}, where $\tau_\mathbf{n} = \mathbf{S_0^{(n)}}, \mathbf{A_0^{(n)}}, R_0^{(n)}, \ldots, \mathbf{S_{T-1}^{(n)}}, \mathbf{A_{T-1}^{(n)}}, R_T^{(n)}$;
Determine returns $\{G_t\}_{t=0}^{T-1}$, where $G_i = \sum_{k=0}^{T} \gamma^k r_{i+k}$;
Store $\{\tau\}_{n=1}^{N}$ in EpochBuffer(1);
for $ep \leftarrow 2$ **to** *MaxEpoch* **do**
  Apply Gradient Ascent Method on $J(\theta)$ to get $\pi_{\mathbf{ep}}$: $\theta_{ep} \leftarrow \theta_{ep-1} + \alpha \frac{1}{N} \sum_{n=1}^{N} \sum_{t=0}^{T-1} G_t \frac{\nabla \pi_\theta(\mathbf{s_t}, \mathbf{a_t})}{\pi_\theta(\mathbf{s_t}, \mathbf{a_t})}$;
  Generate N trajectories $\{\tau_n\}_{n=1}^{N}$ from action policy $\pi_{\theta_{\mathbf{ep}}}$, where $\tau_\mathbf{n} = \mathbf{S_0}, \mathbf{A_0}, R_0, \ldots, \mathbf{S_{T-1}}, \mathbf{A_{T-1}}, R_T$;
  Determine returns $\{G_t\}_{t=0}^{T-1}$, where $G_i = \sum_{k=0}^{T} \gamma^k r_{i+k}$;
  Store $\{\tau\}_{n=1}^{N}$ in EpochBuffer(ep);
end

4.2 Second Algorithm: DQN

DQN (Deep Q Network) can be seen as a generalization of the simple Q-Learning algorithm known as Q-tables. Rather than directly mapping each state-action pair to a value $Q(\mathbf{s}, \mathbf{a})$ and performing successive iterations on the resulting table, DQN performs the parameterization of the state-action value function $Q_\theta(\mathbf{s}, \mathbf{a})$ as a weighted neural network (Mnih et al., 2013). The network is initialized arbitrarily with random weights θ, which are updated successively as state transitions $(\mathbf{s}, \mathbf{a}, r, \mathbf{s}')$ are observed by the agent. The DQN network is trained to satisfy the Bellman Equation (Eq. (3) – (Bellman, 1967)):

$$Q_\theta(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \max_{\mathbf{a}' \in \mathcal{A}} Q_\theta(\mathbf{s_{t+1}}, \mathbf{a}') \tag{3}$$

Algorithm 2: DQN

- Initialize the robot, *setpoint*, *obstacle*, initial state $\mathbf{s_0}$ and action space $\mathcal{A}$;
- Initialize Hyperparameters (bonuses and penalties, network, number of *timesteps*, epochs and transitions at *Buffer*, discount factor γ, learning rateα) and ϵ;
- Initialize epoch storage structure;
- Initialize parameterized DQN network Q_{θ_0} randomly;

for $ep \leftarrow 1$ **to** *MaxEpoch* **do**
  Initialize state: $\mathbf{s} \leftarrow \mathbf{s_0}$;
  Fill *Experience Buffer* with N transitions given current network $Q_{\theta_{\mathbf{ep}}}$ and ϵ-*Greedy* method ;
  Sample random *mini-batch* of size N_{batch} from *Experience Buffer*;
  for $i \leftarrow 1$ **to** N_{batch} **do**
    Read i-th transition: $(\mathbf{s}, \mathbf{a}, r, \mathbf{s}', bool_{term})$;
    if $bool_{term} == true$ **then**
      $y = r$;
    else
      $y = r + \gamma \max_{\mathbf{a}' \in \mathcal{A}} Q_{\theta_{\mathbf{ep}}}(\mathbf{s}', \mathbf{a}')$;
    end
    store expected output $q = Q(\mathbf{s}, \mathbf{a})$ and target y ;
  end
  Applies Gradient Descent to minimize cost function given by $\mathcal{L}(\theta) = \frac{1}{2}(Q_\theta(\mathbf{s}, \mathbf{a}) - y)^2$, that is: $\theta_{\mathbf{ep+1}} \leftarrow \theta_{\mathbf{ep}} - \alpha \frac{1}{N_{batch}} \sum_{i=1}^{N_{batch}} \nabla_\theta \frac{1}{2}(Q_\theta(\mathbf{s}, \mathbf{a}) - y)^2$;
  Clear *Experience Buffer*;
end

which associates the value of a state-action pair to the maximum value of subsequent state-action pairs. DQN can be seen as a Supervised Learning Algorithm in which the *target* is non-stationary, as it depends on the $\mathbf{Q}_\theta(\mathbf{s}, \mathbf{a})$ function itself, except for terminal states, in which the target is simply the reward $r(\mathbf{s_t}, \mathbf{a_t})$. This is one of the major difficulties associated with this method and causes its convergence to depend on sufficient exploration of different actions across the entire state space. However, given sufficient exploration, the algorithm is proved to converge to the optimal state-action value function $\mathbf{Q}^*(\mathbf{s}, \mathbf{a})$.

In order to improve numerical conditioning and allow for faster convergence, a technique known as Prioritized Experience Replay (Schaul et al., 2016) is implemented, where state transitions are stored in a experience buffer and sampled randomly, while terminal transitions are always sampled. State transition tuples are defined as $(\mathbf{s}, \mathbf{a}, r, \mathbf{s}', bool_{term})$ where $\mathbf{s}$ is the system's current state, $\mathbf{a}$ is the action taken by the agent, r the reward obtained and $\mathbf{s}'$ is the subsequent state, in addition, a Boolean variable $bool_{term}$ indicates if the state is terminal.

5 Reward Functions

Reward function engineering is critical in reinforcement learning applications. The reward function determines the quality of actions taken by the agent and influences not only the policies it is capable of learning but also the algorithm's convergence. As a result, there is an increasing effort in recent research to optimize reward functions for different tasks.

Over the test projects, three different reward functions are implemented. In the first two test projects, agents trained with one reward function showed significantly better performance than the others. As a result, the Kuka projects focused on the implementation of this reward function. The following sections detail their mathematical implementations, key insights and intuition.

5.1 First Reward Function: Absolute Distances

The first reward function (Eq. (4)) considered is inspired in the potential field method for path planning of mobile robots. The reward function depends on the euclidean distances between the end effector, the obstacle and the goal, similarly to the reward function used in Sangiovanni et al. (2018), which also applies a distance based reward function to the task of training a robotic manipulator for positioning while avoiding obstacles.

$$\begin{aligned} r_1(\mathbf{s},\mathbf{a}) = {} & k(r_{setpoint}(\mathbf{s},\mathbf{a}) + r_{obstacle}(\mathbf{s},\mathbf{a}) + c) \\ & + B_{goal}(\mathbf{s},\mathbf{a}) \\ & + P_{joint\ boundary}(\mathbf{s},\mathbf{a}) \\ & + P_{collision}(\mathbf{s},\mathbf{a}) \end{aligned} \tag{4}$$

where

$$\begin{cases} r_{setpoint} = -k_1 \left\| \mathbf{p}_{sp} - \mathbf{p}'_{ef} \right\|_2 \\ r_{obstacle} = k_2 \left\| \mathbf{p}_{obs} - \mathbf{p}'_{ef} \right\|_2 \end{cases}$$

$\mathbf{p}'_{ef}$ is the end effector's future position after action $\mathbf{a}$ is taken and $\mathbf{p}_{sp}$ and $\mathbf{p}_{obs}$ are the setpoint and obstacle positions respectively. The remaining terms represent bonuses or penalties given to the agent based on the desired behavior: B_{goal} is a bonus given when the desired object is reached, $P_{collision}$ is a penalty given when either the table or the red obstacle is hit and $P_{joint\ boundary}$ is a penalty given when one of the robot's joint's limit is reached.

5.2 Second Reward Function: Discrete under Approximation or Distancing

In order to correct problems observed in the first function, such as high magnitude and non-zero average value over all the possible actions at a given state, a second function is tested. The second reward function (Eq. (5)) is dependent on the relative approximation or distancing between the end effector, the obstacle and the goal.

$$\begin{aligned} r_2(\mathbf{s},\mathbf{a}) = {} & (k_s r_{setpoint}(\mathbf{s},\mathbf{a}) + k_o r_{obstacle}(\mathbf{s},\mathbf{a})) \\ & + B_{goal}(\mathbf{s},\mathbf{a}) \\ & + P_{joint\ boundary}(\mathbf{s},\mathbf{a}) \\ & + P_{collision}(\mathbf{s},\mathbf{a}) \end{aligned} \tag{5}$$

where

$$r_{setpoint} = \begin{cases} -1, & if \left\| \mathbf{p}_{sp} - \mathbf{p}'_{ef} \right\|_2 > \left\| \mathbf{p}_{sp} - \mathbf{p}_{ef} \right\|_2 \\ 0, & if \left\| \mathbf{p}_{sp} - \mathbf{p}'_{ef} \right\|_2 = \left\| \mathbf{p}_{sp} - \mathbf{p}_{ef} \right\|_2 \\ 1, & if \left\| \mathbf{p}_{sp} - \mathbf{p}'_{ef} \right\|_2 < \left\| \mathbf{p}_{sp} - \mathbf{p}_{ef} \right\|_2 \end{cases}$$

$$r_{obstacle} = \begin{cases} 0, \ if \left\| \mathbf{p}_{obs} - \mathbf{p}'_{ef} \right\|_2 > r_{infl} \\ \begin{cases} -1, & if \left\| \mathbf{p}_{obs} - \mathbf{p}'_{ef} \right\|_2 < \left\| \mathbf{p}_{obs} - \mathbf{p}_{ef} \right\|_2 \\ 0, & if \left\| \mathbf{p}_{obs} - \mathbf{p}'_{ef} \right\|_2 = \left\| \mathbf{p}_{obs} - \mathbf{p}_{ef} \right\|_2 \\ 1, & if \left\| \mathbf{p}_{obs} - \mathbf{p}'_{ef} \right\|_2 > \left\| \mathbf{p}_{obs} - \mathbf{p}_{ef} \right\|_2 \end{cases} \end{cases}$$

The discrete penalties and rewards given in case of collision are the same as defined in $\mathbf{r_1}(\mathbf{s},\mathbf{a})$, given in Eq. (4). Fig. 3 illustrates the normalized average reward obtained by REINFORCE and Q-Learning agents for a 1-DOF robot (Test Project 1).

5.3 Third Reward Function: Projection of Displacement Vector

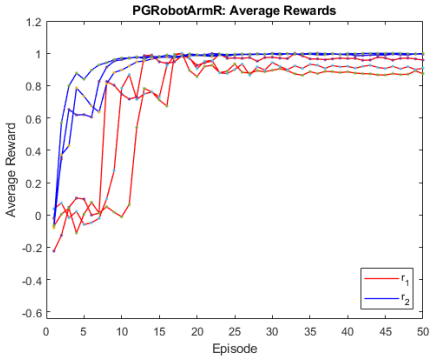
Finally, the third reward function (Eq. (6)) is similar to the second, but the terms $r_{setpoint}$ and $r_{obstacle}$ are no longer limited to −1, 0 and 1, but are given by the projection of the displacement vector $\mathbf{n}_{ef->ef'}$ in the directions that point to the goal $\mathbf{n}_{ef->setpoint}$ and to the obstacle $\mathbf{n}_{ef->obstacle}$.

$$\begin{aligned} r_3(\mathbf{s},\mathbf{a}) = {} & (k_s r_{setpoint}(\mathbf{s},\mathbf{a}) + k_o r_{obstacle}(\mathbf{s},\mathbf{a})) \\ & + B_{goal}(\mathbf{s},\mathbf{a}) \\ & + P_{joint\ boundary}(\mathbf{s},\mathbf{a}) \\ & + P_{collision}(\mathbf{s},\mathbf{a}) \end{aligned} \tag{6}$$

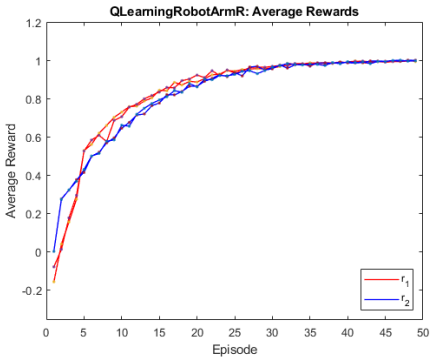
where

$$\begin{cases} r_{setpoint}(\mathbf{s},\mathbf{a}) = (\mathbf{n}_{ef->ef'} \bullet \mathbf{n}_{ef->setpoint}) \\ r_{obstacle}(\mathbf{s},\mathbf{a}) = \begin{cases} 0, \text{ if } \left\| \mathbf{p}_{obstacle} - \mathbf{p}'_{ef} \right\|_2 > r_{infl} \\ -(\mathbf{n}_{ef->ef'} \bullet \mathbf{n}_{ef->obstacle}), \text{ otherwise} \end{cases} \end{cases}$$

and $B_{goal}(\mathbf{s},\mathbf{a})$, $P_{joint\ boundary}(\mathbf{s},\mathbf{a})$ and $P_{collision}(\mathbf{s},\mathbf{a})$ are defined the same way as in previous reward functions. Fig. 4 illustrates a diagram of the third reward function for two different actions taken in the initial state $\mathbf{s_0}$. $\mathbf{s'_A}$ and $\mathbf{s'_B}$ are the system's states after the agent has taken actions $\mathbf{a_A}$ and $\mathbf{a_B}$ respectively. The dotted lines represent the vectors that point to the desired position and to the



(a) Reward function comparison for REINFORCE agent.



(b) Reward function comparison for Q-Learning agent.

Figure 3: Normalized average reward per epoch obtained by (a) REINFORCE and (b) Q-Learning agents for reward functions r_1 (red) and r_2 (blue) on test project 1.

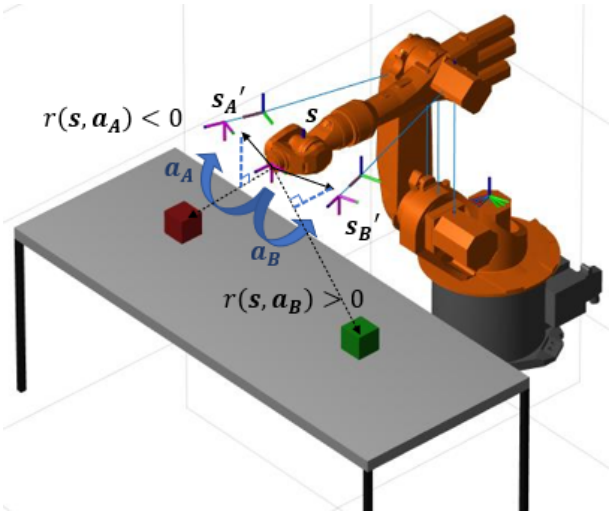
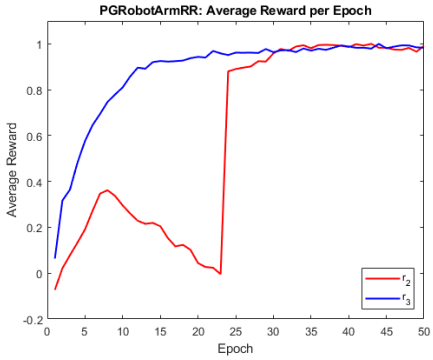


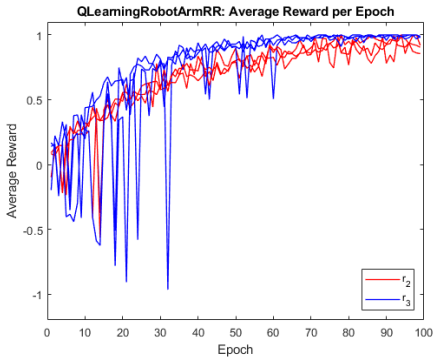
Figure 4: Diagram of the third reward function.

obstacle. Finally, the reward is given by the corresponding projections.

Overall, agents trained with the reward function r_3 showed better performance in comparison to those



(a) Reward function comparison for REINFORCE agent.



(b) Reward function comparison for Q-Learning agent.

Figure 5: Normalized average reward per epoch obtained by REINFORCE (a) and Q-Learning (b) agents for reward functions r_2 (red) and r_3 (blue) on test project 2.

trained with r_2. Fig. 5 shows the average reward per epoch obtained during training of a 2-DOF robot. The REINFORCE agent trained with reward function r_3 presented a significantly better performance in terms of convergence time and stability (Fig. 5a) while the Q-Learning agent showed more frequent drops in its learning curve during epochs in which a collision with the obstacle occurred (Fig. 5b), which is possibly due to r_3's priority to direct paths to the goal combined with a goal-obstacle configuration in which a direct path is obstructed.

6 Results

From the partial results obtained during training of a 1 and 2-DOF robot, agents trained with the third reward function presented significantly better performance. As a result, only r_3 is implemented in the last two projects, which focus on a comparative analysis between both classes of algorithms. In this section, the training frameworks and detailed results obtained from applying both classes of algorithms to the Kuka test projects are presented. The results are followed by a brief comparative analysis and summarized at the end of the section.

A side-by-side comparison of both algorithms in increasingly more sophisticated applications is valuable

Table 1: Kuka KR16's joint limits and implemented angular limits.

Joint	Angular Limits	$[\theta_{i_{inf}}, \theta_{i_{sup}}]$
A1	$[-185^o, 185^o]$	$[-30^o, 40^o]$
A2	$[-65^o, 125^o]$	$[-20^o, 40^o]$
A3	$[-220^o, 64^o]$	$[-30^o, 50^o]$
A4	$[-350^o, 350^o]$	$[-40^o, 40^o]$
A5	$[-130^o, 130^o]$	$[-40^o, 40^o]$
A6	$[-350^o, 350^o]$	not controlled

as it allows us to focus on the algorithms' foundations, eliminating sources of instability or non-convergence and comparing both in identical settings. Another advantage is the possibility to exploit modular, object-oriented program design, since most functions are shared by the various test projects and can be easily adapted to other applications. As shown in Fig. 2, the test projects are characterized by the simplifications considered: The first two implement 1 and 2 DOF robots, while the last two implement the 6-DOF KUKA KR16 robot, but with initially fixed and then generic configurations of goal and obstacle.

6.1 Test Project 3: KUKA KR16 - Fixed Configuration

After comparing both the reward function and the algorithm on robots with a reduced number of degrees of freedom, a 3-dimensional simulation and visualization environment for the KUKA-KR16 robot was implemented on Matlab by loading *RigidBodyTree* object representation of the robot based on its urdf file and stl meshes.

In this project, the agent's task is to control the robot's first five degrees of freedom to position its end effector on the fixed goal position (green) while avoiding collision with a known obstacle (red) and the table, which is unknown and is only detectable through interaction. Due to overall better performance observed on agents trained with the third reward function (Eq. (6)) on both algorithms, the following projects implement only r_3 as the reward function and focus on a comparative analysis between algorithms as well as on techniques to overcome the dimensionality issue on real robotics applications.

The State Space is now given by Eq. (7):

$$\mathcal{S} = \left(\prod_{i=1}^{5} \mathcal{S}_i\right) \text{x } \mathcal{R}^3 \text{ x } \mathcal{R}^3, \quad \text{where } \mathcal{S}_i = \{\theta_{i_{inf}} + i\Delta\theta | i = 1, ..., \frac{\theta_{i_{sup}}}{\Delta\theta}\} \tag{7}$$

Similarly to previous test projects, the State Space is the combination of possible angular positions for each controllable rotating joint $\mathcal{S}_i$ and all possible Cartesian positions for the goal and the obstacle in three-dimensional space $\mathcal{R}^3$. Table 1 indicates Kuka KR16's joint limits and the implemented limits give the table workspace.

The Action Space is: $\mathcal{A} = \prod_{i=1}^{5} \mathcal{A}_i$, where $\mathcal{A}_i = \{-1, 0, 1\}$,

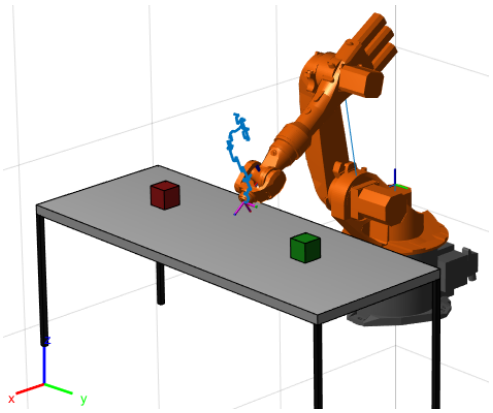


Figure 6: Simulation Environment developed for test projects 3 and 4. Goal and obstacle are represented by green and red cubes and a random trajectory is shown in blue.

which corresponds to all possible positive, neutral and negative increments for all joint actuators. As a result, the number of possible actions is $3^5 = 243$ (Eq. (8)). Fig. 6 illustrates the simulation environment.

$$\mathcal{A} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & -1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & -1 \\ 1 & 1 & 1 & -1 & 1 \\ 1 & 1 & 1 & -1 & 0 \\ 1 & 1 & 1 & -1 & -1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ -1 & -1 & -1 & -1 & 1 \\ -1 & -1 & -1 & -1 & 0 \\ -1 & -1 & -1 & -1 & -1 \end{bmatrix} \tag{8}$$

6.1.1 *Episodic REINFORCE*

The algorithm's generic formulation allowed for relatively simple adaptation to the new project. The policy function $\pi_\theta(\mathbf{s}, \mathbf{a})$ neural network complexity was increased in order to allow for the abstraction of more complex policies. A three-layer feedforward network with 104000 trainable parameters was implemented. Table 2 summarizes the project's hyperparameters.

Similarly to previous test projects, the direct approximation and training of the policy function $\pi_\theta(\mathbf{a}|\mathbf{s})$ yielded a smooth, monotonically increasing average reward curve (Fig. 7). As opposed to value iteration algorithms, which search for the optimal state-action value function $Q_\theta(s, a)$ and derive the optimal policy by taking the action of most value at each state.

In order to study the agent's increasing preference for optimal actions during training, the probability

Table 2: Variables and Hyperparameters of Episodic REINFORCE implementation of Test Project 3

Parameter	Description	Value
$\mathbf{p_{setpoint}}$	Goal Position (m)	(1.05, 0.45, 0.75)
$\mathbf{p_{obstacle}}$	Obstacle Position (m)	(1.05, −0.55, 0.75)
$\Delta\theta$	Minimum Joint Angle Step	1^o
α	Learning Rate	0.0005
B_{goal}	Goal Bonus	400
$P_{collision}$	Collision Penalty	-400
$P_{joint\ boundary}$	Joint Limit Penalty	-400
r_{infl}	Obstacle Influence Radius (m)	0.50
$MaxEpoch$	Maximum Number of Training Epochs	50
N	Number of Trajectories per Epoch	30
T	Maximum Number of Actions per Trajectory	70
γ	Discount Factor	0.3
$dim(s)$	State **s** Dimension	11
$dim(a)$	Action **a** Dimension	5
$size(\mathcal{A})$	Action Space $\mathcal{A}$ Size	243
$(n_{in}, n_{h1}, n_{h2}, n_{out})$	$\pi_\theta(\mathbf{a}\|\mathbf{s})$ Network Neurons per Layer	(11, 100, 300, 243)
k_s	Goal Multiplicative Factor	100
k_o	Obstacle Multiplicative Factor	70

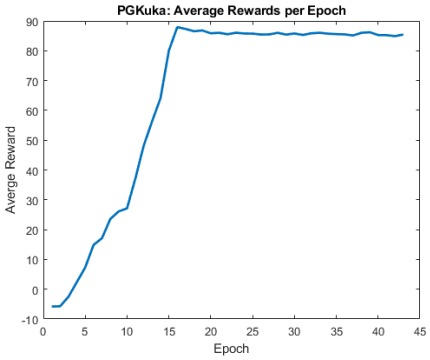
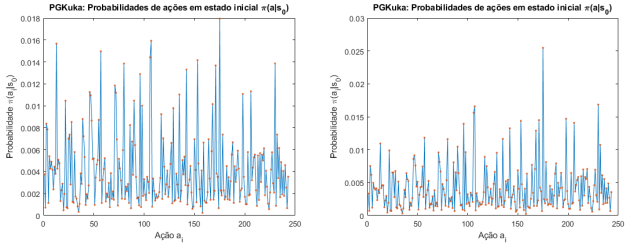


Figure 7: Normalized average reward per epoch obtained by REINFORCE agent trained with reward function r_3.

distribution of actions in $\mathcal{A}$ given the initial state was plotted for epochs 1, 4, 8 and 12 (Fig. 8).



(a) $\pi_\theta(\mathbf{a}|\mathbf{s_0})$ in epoch 1 **(b)** $\pi_\theta(\mathbf{a}|\mathbf{s_0})$ in epoch 4

(c) $\pi_\theta(\mathbf{a}|\mathbf{s_0})$ in epoch 8 **(d)** $\pi_\theta(\mathbf{a}|\mathbf{s_0})$ in epoch 12

Figure 8: Probability Distribution $\pi_\theta(\mathbf{a}|\mathbf{s_0})$ over action space $\mathcal{A}$ for initial state $\mathbf{s_0}$ during training in epochs 1, 4, 8 and 12.

6.1.2 DQN

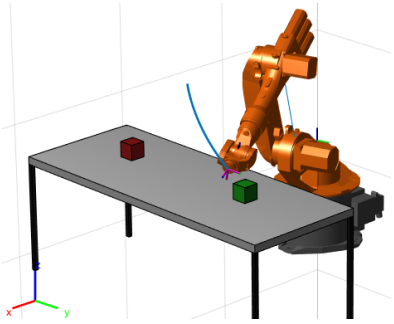
Due to exponentially increasing state-action space dimension as the number of degrees of freedom increases, a Q-table algorithm is impracticable as a result of memory and computation limitations. In order to overcome the dimensionality issue, the state-action value function $\mathbf{Q}_\theta(\mathbf{s}, \mathbf{a})$ was represented as a Multi-layer Perceptron (MLP). The algorithm's formulation, detailed in Section 4.2, consists in applying gradient descent in order to minimize the mean squared error between the network's current output $\mathbf{Q}_\theta(\mathbf{s}, \mathbf{a})$ and the target $r(\mathbf{s}, \mathbf{a}) + \gamma \max_{\mathbf{a}' \in \mathcal{A}} Q_{\theta_{\mathbf{ep}}}(\mathbf{s}', \mathbf{a}')$, where $\mathbf{s}'$ denotes the state reached after action $\mathbf{a}$ is executed in state $\mathbf{s}$. Table 3 summarizes the algorithm-specific hyperparameters implemented.

Fig. 9 illustrates the trajectory taken by the DQN agent after convergence and the average reward performance curve during training.

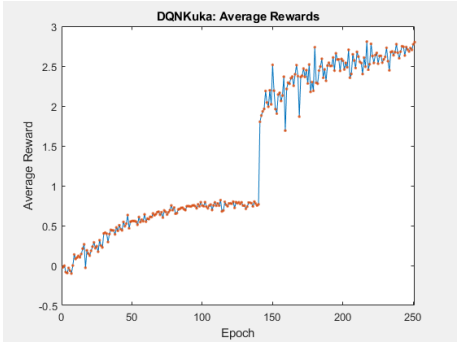
In order to analyze the agent's behavior in cases where a direct path to the goal is blocked by an unknown object, a wall was placed between the effector's initial position and the goal. Similarly to table collision, wall collision is incorporated into the state transition function and terminates a trajectory if the robot's end effector is sufficiently close to the wall. A negative reward of $P_{collision}$ is given during collision and the wall's position can only be learned through environmental interaction. Fig. 10 illustrates the optimal trajectory found by the agent and the corresponding learning curve during training. As expected, an increased number of epochs was necessary for the abstraction of more complex behavior, but the DQN agent was able to dodge the wall correctly with no algorithmic changes.

Table 3: Variables and Hyperparameters of DQN implementation on Test Project 2

Parameter	Description	Value
α	Learning Rate	0.002
MaxEpoch	Maximum Number o Training Epochs	250
Ntrajs	Number of Trajectories per Epoch	10
MiniBatchSize	Number of Sampled Transitions for Training	200
T	Maximum Number of Actions per Trajectory	70
γ	Discount Factor	0.3
$dim(s)$	State **s** Dimension	11
$dim(a)$	Action **a** Dimension	5
$size(\mathcal{A})$	Action Space $\mathcal{A}$ Size	243
$(n_{in}, n_{h1}, n_{h2}, n_{out})$	$\pi_\theta(\mathbf{a}\|\mathbf{s})$ Network Neurons per Layer	$(11, 100, 300, 243)$

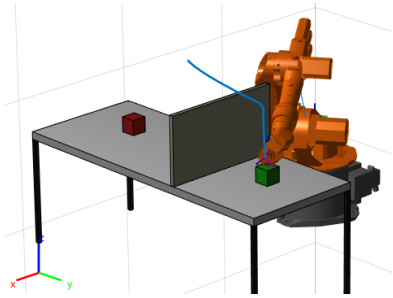


(a) Trajectory taken by DQN agent after training

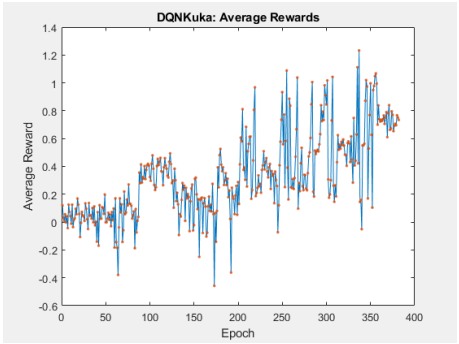


(b) DQN agent performance curve

Figure 9: Optimal trajectory taken by agent (a) and average rewards obtained per epoch during training (b) in DQN implementation on Test Project 3.



(a) Optimal Trajectory found by DQN agent in environment with unknown blocking obstacle



(b) Average rewards per epoch obtained during training

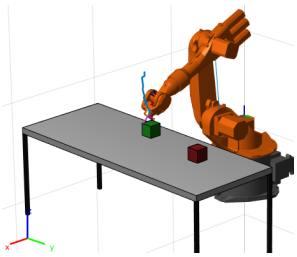
Figure 10: Optimal Trajectory (a) and learning curve during training (b) obtained by DQN agent implementation on variation of Test Project 3 with unknown obstacle.

6.2 Test Project 4: KUKA KR16 - Generic Configurations

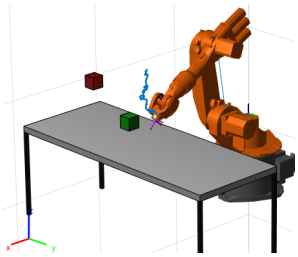
The main advantage of adaptive learning applied to the control of industrial robots is the flexibility in unexpected scenarios, the scalability provided by training over time and the abstraction of complex and often nonintuitive policies with minimal human intervention. In order to investigate both agents' capability of learning an efficient positioning task for objects randomly located on the workspace, both the goal's (green) and the known obstacle's (red) positions were changed randomly during training. A subspace $\mathcal{W} \subset \mathcal{R}^3$ of the robot's work volume, defined as $\mathcal{W} = \{(x, y, z) \in \mathcal{R}^3 | 0.80 < x < 1.4, -0.90 < y < 0.90, 0.80 < z < 1.00\}$, was chosen for the possible goal and obstacle positions. During testing, planar goal-obstacle configurations often did not require the agent to avoid the obstacle, as a direct path to the goal was frequently present. In order to test the RL agent on more challenging scenarios, a three-dimensional volume of possible goal obstacle configurations was chosen, giving the impression that some objects are floating.

6.2.1 *Episodic REINFORCE*

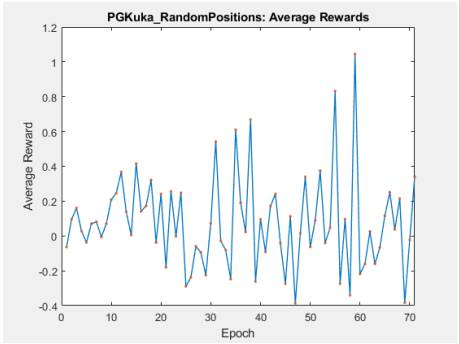
A REINFORCE agent with $\pi_\theta(\mathbf{a}|\mathbf{s})$ policy network architecture equal to Test Project 3 was implemented and trained. However, there was no noticeable increase in performance or convergence to the optimal policy, as



(a) Correct positioning trajectory performed by the agent.

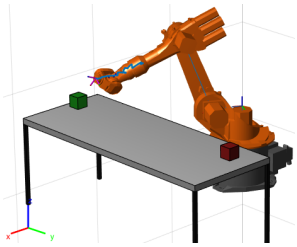


(b) Table collision trajectory performed by the agent.

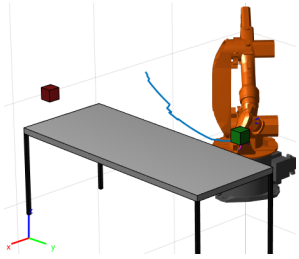


(c) Average rewards per epoch during training

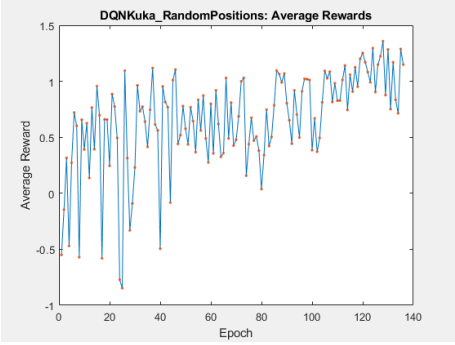
Figure 11: Trajectories generated by the REINFORCE agent after training for different goal-obstacle configurations (a),(b) and learning curve (c) during training.



(a) Trajectory performed by DQN agent in epoch 132 of training.



(b) Trajectory performed by trained agent on random goal-obstacle configuration.



(c) Average rewards per epoch during training

Figure 12: Trajectories performed by DQN agent during (a) and after training (b) and associated learning curve (c) during training.

shown in Fig. 11c. The agent's performance presented an undesired high sensitivity to goal-obstacle configuration, performing the positioning task correctly on specific configurations (Fig. 11a) and incorrectly on others (Fig. 11b). Moreover, proximity between the goal and obstacle resulted in poor performance and an increased chance of table collision.

6.2.2 *DQN*

6.3 Comparative Analysis of Algorithms

In order to compare both classes of algorithms, three main criteria were analyzed: convergence rate over the test projects, execution time and smooth increase of average reward. The policy-iteration algorithm REINFORCE outperformed DQN in the latter, while the value-iteration algorithm showed better execution time and convergence results. Table 4 illustrates the execution time until convergence for both algorithms on the four test projects. DQN showed better scalability with increasing application complexity, while REINFORCE agents presented longer training times.

Table 4: Average training times associated with both agents on each test project.

Project	REINFORCE	Q-Learning
1 Degrees of Freedom (R)	6,8min	6,9 min
2 Degrees of Freedom (RR)	11.7min	7.4min
6 Degrees of Freedom (6R), fixed configuration	30h	16h
6 Degrees of Freedom (6R), random configuration	No convergence	25h

7 Conclusion

The application of newly developed methods, especially in consolidated industries where sensitive operations require that safety conditions must be met, is subject to an extensive research in simulated settings and controlled environments. Reinforcement Learning is a relatively new field with promising results in control and game theory.

The main contributions of this work are the evaluation of two classes of RL algorithms applied to a typical industrial robotics task and the development of a modular simulation architecture that allows for simplicity in further investigation of similar problems. We also present a new reward function formulation based on the projection of the end effector's displacement, which significantly improved the agent's performance on both algorithms.

A comparative analysis of both classes of algorithms on increasingly complex environments also highlighted

their main limitations and points to improve future research: sensitivity to reward function, state-action space exponential increase in dimensions, low sample efficiency and consequently high training time. Reward function engineering is where human expert analysis is fundamental, and the dimensionality issue is often overcome by algorithmic changes, such as the replacement of Q-tables with DQN or by modeling the action space $\mathcal{A}$ as continuous and having the policy network $\pi_\theta(\mathbf{a}|\mathbf{s})$ output allow for the mapping onto continuous actions, commonly done in algorithms such as REINFORCE, DDPG and Actor-Critic (Sutton and Barto, 2018).

The non-convergent behavior obtained by the REINFORCE agent on the last project can be explained by common limitations associated with policy iteration algorithms in general, such as high sensitivity to learning rate and exploratory variance (Kormushev et al., 2013). DQN's overall better performance in shorter training periods is possibly due to higher frequency network updates and the implementation of an experience replay from which state transitions are randomly sampled (Lin, 1993).

Acknowledgments

This project was developed in the Mechatronics Department of Escola Politécnica da USP (PMR Poli-USP), which provided access to the Kuka-KR16 robot.

The first author has the support of TPN-Poli (Tanque de Provas Numérico da Escola Politécnica da USP).

The second author is supported by Itaú Unibanco S.A through the scholarship program of Programa de Bolsas Itaú (PBI), linked to the Centro de Ciência de Dados (C2D), Escola Politécnica da Universidade de São Paulo.

References

Bellman, R. (1967). *Introduction to the Mathematical Theory of Control Processes*, Mathematics in science and engineering, v. 40-1, Academic Press, New York. Available at https://www.sciencedirect.com/bookseries/mathematics-in-science-and-engineering/vol/40/part/P1.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J. and Zaremba, W. (2016). Openai gym. Available at https://arxiv.org/abs/1606.01540.

Charpentier, A., Élie, R. and Remlinger, C. (2021). *Reinforcement Learning in Economics and Finance*, Vol. 4. https://doi.org/10.1007/s10614-021-10119-4.

Chen, S., Yan, D., Zhang, Y., Tan, Y. and Wang, W. (2019). Live working manipulator control model based on dppo-dqn combined algorithm, *2019 IEEE 4th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, Vol. 1, pp. 2620–2624. https://doi.org/10.1109/IAEAC47372.2019.8997839.

Coronato, A., Naeem, M., De Pietro, G. and Paragliola, G. (2020). Reinforcement learning for intelligent healthcare applications: A survey, *Artificial Intelligence in Medicine* **109**: 101964. https://doi.org/10.1016/j.artmed.2020.101964.

Gu, S., Holly, E., Lillicrap, T. and Levine, S. (2017). Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates, *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3389–3396. https://doi.org/10.1109/ICRA.2017.7989385.

James, S. and Johns, E. (2016). 3d simulation for robot arm control with deep q-learning. Available at https://arxiv.org/abs/1609.03759.

Kaelbling, L. P., Littman, M. L. and Moore, A. W. (1996). Reinforcement learning: A survey, *J. Artif. Int. Res.* **4**(1): 237–285. https://doi.org/10.1613/jair.301.

Kormushev, P., Calinon, S. and Caldwell, D. (2013). Reinforcement learning in robotics: Applications and real-world challenges, *Robotics* **2**: 122–148. https://doi.org/10.3390/robotics2030122.

Lin, L.-J. (1993). *Reinforcement Learning for Robots Using Neural Networks*, PhD thesis, USA. Available at https://isl.anthropomatik.kit.edu/pdf/Lin1993.pdf.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D. and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning, *in* M. F. Balcan and K. Q. Weinberger (eds), *Proceedings of The 33rd International Conference on Machine Learning*, Vol. 48 of *Proceedings of Machine Learning Research*, PMLR, New York, New York, USA, pp. 1928–1937. Available at https://proceedings.mlr.press/v48/mniha16.html.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. Available at https://arxiv.org/abs/1312.5602.

Rosen, C. A. (1999). Chapter 3 - robots and machine intelligence, *Handbook of Industrial Robotics*, John Wiley & Sons, New York,United States of America, pp. 19–30. https://doi.org/10.1002/9780470172506.ch3.

Sangiovanni, B., Rendiniello, A., Incremona, G. P., Ferrara, A. and Piastra, M. (2018). Deep reinforcement learning for collision avoidance of robotic manipulators, *2018 European Control Conference (ECC)*, pp. 2063–2068. https://doi.org/10.23919/ECC.2018.8550363.

Schaul, T., Quan, J., Antonoglou, I. and Silver, D. (2016). Prioritized experience replay. Available at https://arxiv.org/abs/1511.05952.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A. and Klimov, O. (2017). Proximal policy optimization algorithms. Available at https://arxiv.org/abs/1707.06347.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*, A Bradford Book, Cambridge, MA, USA. Available at http://incompleteideas.net/book/the-book.html.

Todorov, E., Erez, T. and Tassa, Y. (2012). Mujoco: A physics engine for model-based control, *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. https://doi.org/10.1109/IROS.2012.6386109.

Xie, J., Shao, Z., Li, Y., Guan, Y. and Tan, J. (2019). Deep reinforcement learning with optimized reward functions for robotic trajectory planning, *IEEE Access* **7**: 105669–105679. https://doi.org/10.1109/ACCESS.2019.2932257.