# Task-motion Planning via Tree-based Q-learning Approach for Robotic Object Displacement in Cluttered Spaces

Giacomo Golluccio, Daniele Di Vito, Alessandro Marino, Alessandro Bria and Gianluca Antonelli*
*Department of Electrical and Information Engineering,*
*University of Cassino and Southern Lazio, via Di Biasio 43, 03043 Cassino (FR), Italy*

Keywords: Motion Planning, Task Planning, Reinforcement Learning.

Abstract: In this paper, a Reinforcement Learning approach to the problem of grasping a target object from clutter by a robotic arm is addressed. A layered architecture is devised to the scope. The bottom layer is in charge of planning robot motion in order to relocate objects while taking into account robot constraints, whereas the top layer takes decision about which obstacles to relocate. In order to generate an optimal sequence of obstacles according to some metrics, a tree is dynamically built where nodes represent sequences of relocated objects and edge weights are updated according to a Q-learning-inspired algorithm. Four different exploration strategies of the solution tree are considered, ranging from a random strategy to a ε-Greedy learning-based exploration. The four strategies are compared based on some predefined metrics and in scenarios with different complexity. The learning-based approaches are able to provide optimal relocation sequences despite the high dimensional search space, with the ε-Greedy strategy showing better performance, especially in complex scenarios.

## 1 INTRODUCTION

Nowadays, the problem of retrieving objects from clutter is of particular interest in fields like industrial and assistive robotics. This involves the robot planning over a task-motion space, and combining discrete decisions about objects and actions with continuous decisions about collision-free paths (Dantam et al., 2016). A vast portion of industrial processes are characterized by a static environment with known object positions, and thus can be easily automated exploiting the advances in robot technologies such as mechanism design and control (Lee et al., 2019).

On the other hand, grasping a target from clutter in assistive scenarios remains challenging due to a highly unstructured environment populated by a variety of objects arranged irregularly and accessible only after relocating other objects (Cheong et al., 2020). Since isolated motion planning typically assumes a fixed configuration space, integrated Task-Motion Planning (TMP) that orchestrates high level task planning and low-level motion planning is required in this scenario. Moreover, the problem of planning for object rearrangement has been shown to be $\mathcal{NP}$-hard (Cheong et al., 2020).

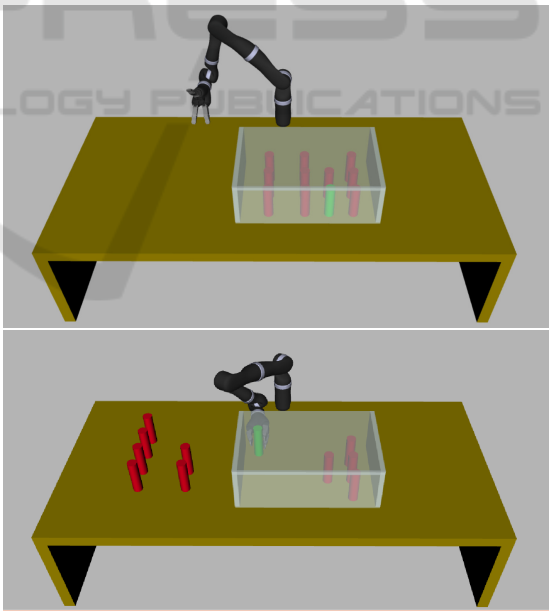

Figure 1: An example of cluttered environment. The target is represented by the cylinder in green, while obstacles are in red. Starting from the scenario on the top, the manipulator is required to relocate some obstacles in order to grasp the target (bottom scenario).

*This work was supported by the Dipartimento di Eccellenza through the Department of Electrical and Information Engineering (DIEI), University of Cassino and Southern Lazio



Golluccio, G., Di Vito, D., Marino, A., Bria, A. and Antonelli, G.
Task-motion Planning via Tree-based Q-learning Approach for Robotic Object Displacement in Cluttered Spaces.
DOI: 10.5220/0010542601300137
In *Proceedings of the 18th International Conference on Informatics in Control, Automation and Robotics (ICINCO 2021)*, pages 130-137
ISBN: 978-989-758-522-7

A number of approaches have been proposed in literature to overcome the abovementioned issues. In (Dogar et al., 2014) it is reported an algorithm that computes a sequence of objects to be removed while minimizing the expected time to find a hidden target. This presents difficulties linked to running time, consequently in a very cluttered environment it might be inoperable. A similar problem is solved in (Srivastava et al., 2014), but the minimization of the number of actions is neglected, while in (Lagriffoul et al., 2014), a combined task and motion planning with the aim to reduce the dimensionality of search space through a constraint-based approach is proposed. In (Nam et al., 2020), the Authors compute the object to relocate by building a traversability graph in a static and known environment with a heuristics in the order to grasp the objects based on their volumes.

An interesting area of machine learning is Reinforcement Learning (RL) that concerns an agent proceeding by trial-and-error in a complex environment in order to maximize a cumulative reward that ultimately leads to the goal achievement. Recently, RL techniques have received a growing share of research in the field of combined TMP (Eppe et al., 2019) including object manipulation and grasping in a cluttered environment (Mohammed et al., 2020). In this context, a widely used approach is to train an end-to-end *deep*-RL model that takes a visual input (e.g. single- or multi-view RGB or RGB-D image) and maps it to feasible actions that lead to the goal (Deng et al., 2019; Yuan et al., 2019). This is possible by leveraging the recent advances in deep convolutional neural networks (LeCun et al., 2015) and value/policy-based RL (Mnih et al., 2013). Despite its merits, deep-RL-based robotic grasping might have some limitations such as the difficulty to interpret the deep neural network model, the high computational complexity and the requirement of a significant amount of attempts, before to reach convergence.

In this paper, we propose a method to solve pick-and-place problems in a cluttered environment integrating TMP and Q-value-based RL. The idea is to find the optimal sequence of obstacles to relocate using Q-learning (Watkins and Dayan, 1992; Jang et al., 2019) in order to reach the target taking into account kinematic constraints. The innovative contribution is twofold: firstly, we design an ordered-tree structure (*Q-Tree*) that replaces the traditional value-matrix in Q-learning, and adapt the learning algorithm accordingly; secondly, we integrate in the Q-Tree learning the motion planner based on the inverse kinematics framework proposed in (Di Vito et al., 2020).

# 2 PROBLEM FORMULATION AND MATHEMATICAL BACKGROUND

## 2.1 Problem Formulation

In this paper, we consider the problem of moving a target object $T$ from its initial location $p_{t,0} \in \mathbb{R}^3$ to a final one $p_{t,f} \in \mathbb{R}^3$ from clutter; therefore, in order to achieve this objective, it might be required to relocate some of or all the $N_o$ obstacles $O = \{O_1, \ldots, O_{No}\}$ present in the scene having position $p_{o,i} \in \mathbb{R}^3$ ($i = 1, 2, \cdots, N_o$). Furthermore, target grasping and object relocation are performed by a robot manipulator; an illustrative scenario is reported in Figure 1, where the initial scene is shown in the top image, while the scene after obstacles relocation is shown in the bottom.

An object, either a target or an obstacle, is occluded if no trajectory for the robot exists to grasp and relocate it, that means that other objects prevent the robot end effector from reaching the object.

Let us denote with $H$ the initial configuration of the scenario, where the robot is in its home position and neither obstacles nor the target were relocated, with $\mathcal{S}_k = \{O_{i_1}, \cdots, O_{i_k}\}$ any sequence of cardinality $|\mathcal{S}_k| = k$ and with $\mathcal{S}_k^T$ the sequence $\{\mathcal{S}_{k-1}, T\}$, i.e. the sequence obtained from $\mathcal{S}_{k-1}$ by adding $T$ as last element. In particular, the generic object $O_{i_j}$ ($j = 1, \ldots, k$) inside a sequence $\mathcal{S}_k$ is defined relocatable if all precedent ones $\{O_{i_1}, \cdots, O_{i_{j-1}}\}$ can be relocated. Thus, a sequence $\mathcal{S}_k$ (or $\mathcal{S}_k^T$) is considered feasible if all its objects are relocatable. Since multiple feasible sequences generally exist, an optimal (or sub-optimal) sequence should be searched for. In this case, for example, the cardinality of the sequence is considered. The following assumptions are made.

**Assumption 1.** *The scene configuration in terms of target location and obstacles positions is known beforehand.*

**Assumption 2.** *Objects are relocated without affecting the remaining ones.*

**Assumption 3.** *For both obstacles and target, only side grasp is allowed.*

In this paper, Reinforcement Learning is adopted as a tool to learn an *optimal* (or sub-optimal) feasible sequence on a tree data structure, for which basics are reported in the following.

## 2.2 Reinforcement Learning

The Reinforcement Learning approach is, in general, a trial-and-error based technique, i.e., an agent iteratively makes observations and takes actions within an environment while receiving feedback referred to as reward. In detail, at each time step $k$ and given a state $s_k$, the agent selects an action $a_k$ and receives a scalar reward $r_k$, trying to maximise the expected reward over time. More in detail, the decision-making problem is modelled as a *Markov decision process* (MDP), that is defined as a tuple $(\mathcal{S}, \mathcal{A}, p, r, \gamma)$ where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $p = Pr(s_{k+1} = s' | s_k = s, a_k = a)$ is the transition probability from the current state to the next one, $r$ is the reward function and $\gamma \in [0,1]$ is the discount factor tuning the rewards weight on the agent action selection. factor, $\gamma \in [0,1]$

In order to decide what action to take at a particular time step, the agent has to know how good a particular state is. A way of measuring the state goodness is the *action-value function* also known as the *Q-function*, defined as the expected sum of rewards taking action $a_k$ in the state $s_k$ following a policy $\pi$:

$$Q_\pi(s,a) = \mathbb{E}_\pi(\sum_{l=0}^{\infty} \gamma^l r_{k+l+1} | s_k = s, a_k = a) \, . \quad (1)$$

Since the agent aim is to maximise the expected cumulative reward, the optimal action-value function is given by

$$Q_*(s,a) = \max_\pi Q_\pi(s,a), \forall s \in S, a \in A \, , \quad (2)$$

which combined with the Eq. (1) leads to the *Bellman optimality equation*

$$Q_*(s,a) = \sum_{s'} p(s'|s,a)[r_{k+1} + \gamma \max_{a'} Q_*(s',a')] \, , \quad (3)$$

where $s_k = s, s_{k+1} = s', a_k = a, a_{k+1} = a'$, respectively. Equation (3) takes into account the state transition probability $p(s'|s,a)$ which assumes the knowledge of the environment model, i.e., a *model-based* problem. However, the same expression can be adopted for *model-free* approaches as well. In the latter case, the state transition probability is set to one resulting in the following:

$$Q_*(s,a) = r_{k+1} + \gamma \max_{a'} Q_*(s',a') \, . \quad (4)$$

How it is noticeable, the expression in Eq. (4) is a recursive nonlinear function with no closed form solution. For this reason, several iterative approaches have been proposed in literature like *Q-learning* in (Watkins and Dayan, 1992). According to this approach, the following iterative update law is adopted:

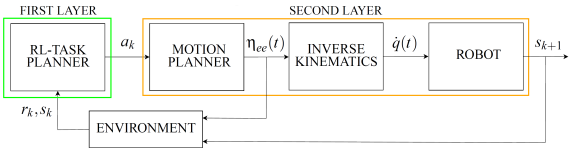


Figure 2: The proposed architecture. The RL-Task Planner choices the action $a_k$ with an appropriate policy, while the Motion Planner provides the feasibility of request. The Inverse Kinematics block sends the reference signal to Robot that acts the motion. The environment elaborates feedback signals from Motion Planner and Robot updating the agent.

$$Q_{k+1}(s_k,a_k) = Q_k(s_k,a_k) + \alpha(r_{k+1} + \gamma \max_{a \in \mathcal{A}} Q_k(s_{k+1},a) - Q_k(s_k,a_k)) \, . \quad (5)$$

where $\alpha \in [0,1]$ is a hyper-parameter known as *learning rate*.

# 3 PROPOSED SOLUTION

Aimed at solving the problem described in Sect. 2.1, we propose the architecture in Figure 2 that is a two-layer architecture, where the high level (the *RL-Task Planner* block) is in charge of *learning* the *optimal* (according to a predefined metrics) feasible sequence of obstacles to relocate in order to reach and grasp the target, while the low level (the *Motion planner, Inverse kinematics (IK)* and *Robot* blocks) is in charge of providing information to the former about feasility of robot movement.

It is worth noticing that the word *Task* used within the low level context does not refer to the high level discrete planning, unlike all the other sections of this work. Indeed, regarding the constrained planning described in next Section, it concerns analytical functions representing control objectives inside the inverse kinematics framework.

## 3.1 Constrained Planning

As mentioned before, the motion planner is in charge of planning feasible trajectories $\eta_{ee}(t)$ for the robotic manipulator end-effector to grasp and relocate objects (either the target or an obstacle) in the scene as selected by the task planning layer. In particular, it is designed as two processes that are the sampling-based algorithm and the Set-based Task-Priority Inverse Kinematics (STPIK) check, respectively (Di Vito et al., 2020). The sampling-based algorithm is performed for planning a feasible trajectory only in the Cartesian space. Thus, the sampling-related computational load is kept down. All the constraints in the system joint space are then verified in

the second process, i.e., the STPIK-check. The latter process consists of verifying the output trajectory $\eta_{ee}(t)$ feasibility through the Set-based Task-Priority Inverse Kinematics framework. This framework is used for the local controller as well. Thus, in case of static environment the trajectory tracking by the robotic system is guaranteed.

Given a robotic manipulator system with $n$ DOFs and, therefore, $q = [q_1 \dots q_n]^{\mathrm{T}}$ its joint space, the STPIK framework allows to performs several priority ordered tasks simultaneously in addition to the end-effector pose $\eta_{ee}$, such as, e.g., obstacle avoidance, virtual walls and joint limits. In particular, the $i$-th task $\sigma_i$ can be set in a priority hierarchy $\mathcal{H}$ such that the velocity components of the lower priority task are projected into the null space of the higher priority one. In this way, the highest priority task is always guaranteed. Thus, given $h$ tasks, the joint velocities $\dot{q}$ are computed as (Siciliano and Slotine, 1991)

$$\dot{q}_h = \sum_{i=1}^{h} (J_i N_{i-1}^A)^{\dagger} (\dot{\sigma}_{i,d} + K_i \tilde{\sigma}_i - J_i \dot{q}_{i-1}) , \quad (6)$$

where $N_i^A$ is the null space of the augmented Jacobian matrix $J_i^A$ stacking the task Jacobian matrices from task 1 to $i$, $\dot{\sigma}_d$ is the time derivative of the task desired value $\sigma_d$, $K$ is a positive-definite gain matrix and $\tilde{\sigma} = \sigma_d - \sigma$ is the task error.

Equation 6 allows to implement the Task-Priority Inverse Kinematics (TPIK) for running multiple tasks simultaneously. The latter are equality-based tasks, i.e., they present a specific desired value as control objective. However, tasks such as joint limits and obstacle avoidance, present a set of allowed values and, therefore, they are defined as set-based tasks. Within the aim to manage this kind of tasks as well, the Set-based Task-Priority Inverse Kinematics (STPIK) algorithm, first proposed in (Di Lillo et al., 2020) and (Di Lillo et al., 2021), is here exploited. The key idea of this method consists of considering the generic set-based task $\sigma$ as disabled, i.e., not included in the hierarchy, until it is in the valid range of values $\sigma_{a,l} < \sigma < \sigma_{a,u}$, and enabled, i.e., included in the hierarchy as a high priority equality task, when it exceeds a predefined lower $\sigma_{a,l}$ or higher $\sigma_{a,u}$ activation threshold. In this way both equality and set-based tasks are properly considered.

## 3.2 RL-Task Planner

Given $N_o$ obstacles, the number of possible sequences $\mathcal{S}$ made of $k \leq N_o$ obstacles with the target $T$ as last element of the sequence is

$$\xi_t = \sum_{i=1}^{No} i! \binom{N_o}{i} , \quad (7)$$

which is obtained by considering all permutations without repetition of $k$ obstacles and where the order matters, plus the target as last element. Obviously, not all these sequences are feasible in the sense defined in Sect. 2.1 and, as already mentioned, we are interested in *efficiently* exploring the search space so as to find, among the feasible solutions, optimal ones according to a given metrics. The proposed method, named Q-Tree significantly limits the computational burden.

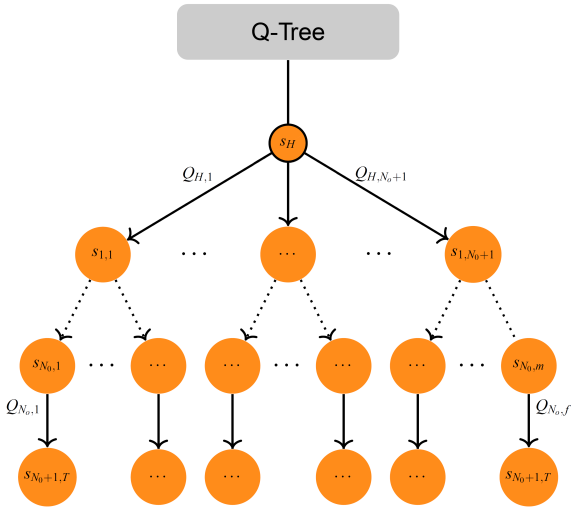


Figure 3: An example of complete Q-Tree. The node root $s_H$ represents the initial configuration where all objects are in the initial location. The last nodes $s_{N_0+1,T}$ represent the states that contain the target.

According to the RL formulation (see Sect. 2.2), actions and reward function need to be defined for the problem at hand. The set of actions $\mathcal{A} = \{a_1, \cdots, a_{N_o}, a_T\}$ is such that $a_i$ $(i = 1, 2, \cdots, N_o)$ and $a_T$ represent the request to the motion planner layer (see Figure 2) to relocate obstacle $O_i$ and target $T$, respectively.

Therefore, given an action in the set $\mathcal{A}$, the motion planner layer provides a binary feedback to the task planning layer regarding the possibility to relocate an object, while meeting all kinematic constraints. In case an object is relocatable.

With regards to the reward function $r$ introduced in Sect. 2.2, the following expression is adopted

$$r(s,a) = \begin{cases} 0, & \text{if grasp}_O \\ -1, & \text{if } \overline{\text{grasp}} \\ 100, & \text{if grasp}_T \end{cases} , \quad (8)$$

meaning that, given a state $s \in \mathcal{S}$ and an action $a \in \mathcal{A}$, the reward is:

- 0 if the obstacle $i$ can be relocated according to the feedback provided by the motion planner;
- $-1$ if either an obstacle or the target cannot be relocated because of occlusions and robot constraints;
- 100 if $a = a_T$ and the target can be grasped and relocated.

Based on this information, the weighted directed tree structure is dynamically updated. In details, a sequence $\mathcal{S}_{s_k}$ is associated to each node $s_k$ of the tree and corresponds to the objects that have been relocated so far (the sequence might be either feasible or unfeasible); the sequence $\mathcal{S}_{s_H}$ associated to the root node, $s_H$, corresponds to the state in which no object was relocated and is the empty sequence (i.e., $\mathcal{S}_{s_H} = \{\}$). Starting from a given node $s_k$, the next action from the set $\mathcal{A}$ corresponding to those objects that have not yet been relocated, and that the motion planner has not already tried to relocate without success is selected; then, two cases are possible: (i) if an action is selected for the very first time from the current node, a new child node $s_{k+1}$ is generated with associated sequence $\mathcal{S}_{k+1} = \{\mathcal{S}_k, O_i\}$ or $\mathcal{S}_k^T = \{\mathcal{S}_k, T\}$ depending on selected action ($a_i$ or $a_T$, respectively). In this case, the weight associated to the new edge connecting $s_k$ and $s_{k+1}$ is initialised to the reward in (8); (ii) if the selected action is relative to a previously explored child node, the edge weight connecting the parent and child nodes is updated according to the Bellman equation in (5).

It is worth noticing that a given node is always a terminal node when the associated sequence is of the type $\{\mathcal{S}_k, O_i\}$, $\forall \mathcal{S}_k$, and is unfeasible, or it is like $\{S_k, T\}$ (i.e., the last object is the target $T$) and it is either feasible or unfeasible.

#### 3.2.1 Tree Policy Exploration

Four exploration strategies, which differ on how the next node to explore are proposed. In details:

- *Random Exploration*-(RND). This case is a no learning approach that represents a random exploration, where the action choice is purely random and the agent does not exploit the information of past choices
- *Learning Random Exploration*-(LRND). According to this policy, given a node $s_{k,l}$ with associated sequence $S_k$, the tree is explored by randomly choosing the next action $a$ in the set $\mathcal{A}$. If the corresponding object cannot be relocated (case $\overline{\text{grasp}}$ in Eq. (8) the current episode is aborted and a new one is started.
- *Random Exploration with Heuristics*-(H-LRND). Differently from the previous case, in a given node, action $a_T$ is selected first if not selected in previous episodes. It is straightforward that this strategy increases the probability to find feasible solution to the proposed problem. In case in the same node the target was already selected in previous episodes, the H-LRND algorithm randomly selects an action not selected in previous episodes or already selected but corresponding to a positive reward (see Eq. (8))
- $\varepsilon$-*Greedy Exploration with Heuristics*-(H-$\varepsilon$G). Like the classical $\varepsilon$-Greedy algorithm, this strategy aims at balancing exploration and exploitation by choosing between them randomly.

  The advantage of this policy with respect to H-LRND is that the latter will spend more time exploring the *promising* parts of the tree by exploiting what learned so far (Géron, 2019). Finally, this attitude towards the exploitation is made increasing over episodes by letting $\varepsilon$ have at each episode $k$ the following expression

$$\varepsilon(k) = \varepsilon_0 \left( \frac{\varepsilon_{\min}}{\varepsilon_0} \right)^{\frac{k-1}{Ep_{\max}-1}}, \tag{9}$$

  where $Ep_{\max}$ is the maximum number of training episodes, $\varepsilon_{\min}$ is the minimum value of $\varepsilon$ to be reached at the end of the training and $\varepsilon_0$ is the initial $\varepsilon$.

The three learning algorithms are summarised in Algorithm 1. The *Q-Tree* algorithm takes as input the clutter scenario, the robotic arm and the learning policy and as output provides optimal or sub-optimal feasible sequence.

In details, at the lines $1, 2$ the tree is initialized and the training starts. At the beginning of each episode, the current node is the root node $s_H$ (line 3) and the next action is chosen according to the predefined policy (line 5). Based on the chosen action, the motion planner is asked to plan a trajectory to relocate the corresponding object (line 6) and a reward is obtained at line 7 as in Eq.(8). If the explored sequence was not explored in previous episodes, a new node is added to the tree (line 9), and edge weights are updated at line 10. The algorithms ends either when the maximum number of episodes $Ep_{\max}$ has been reached or when the tree reaches a steady state condition (no nodes are added and no edge weights are significantly updated). To the aim, we define the quantity $\overline{Q}(k)$ as

$$\overline{Q}(k) = \sum_j |Q_j(k)|, \tag{10}$$

which is the sum of the absolute values of all edge weights at a given episode $k$. We assume the steady

Algorithm 1: Q-Tree.

**Data:**
Obstacles, target, manipulator;
Policy; // LRND,H-LRND,H-εG
**Result:**
(sub)-Optimal feasible sequences
// Create the root node of Tree
CreateRootNode();
// Start Training Agent
**while** $i \leftarrow 1 < Ep_{max}$ & *not(SteadyState)* **do**
  // Reset the scene
  CurrentState = $s_H$ ;
  **for** $j \leftarrow 1$ **to** $It_{max}$ **do**
    // Choose next action
    NextAction = ChooseAction(Policy);
    // Feedback Motion Planner
    MotionPlanner(CurrentState,NextAction);
    r = getReward();
    // Check on the tree
    **if** *not (IsInTree(CurrentState))* **then**
      AddNode(CurrentState);
    UpdateTree();

state condition verified when

$$|\overline{Q}(k) - \overline{Q}(k-1)| \leq \beta \,, \tag{11}$$

where $\beta > 0$ is an appropriate threshold.

#### 3.2.2 Tree Analysis for Optimal Solution Retrieval

After the training phase, a decision tree is available from which the (sub)-optimal sequence can be extracted. In detail, starting from the root node $s_H$, the agent selects the action $a$ in the set $\mathcal{A}$ corresponding to the tree branch with the highest weight, i.e., the maximum $Q$ value. Indeed, the latter represents the optimal choice leading to the target through the shortest tree nodes sequence. Within the aim to prove its effectiveness, let us take into consideration eq. (5). It can be rewritten as:

$$\begin{aligned} Q_{k+1}(s_k,a_k) =& (1-\alpha)Q_k(s_k,a_k) + \\ & \alpha r_{k+1} + \alpha\gamma \max_{a\in\mathcal{A}} Q_k(s_{k+1},a_{k+1}) \,. \end{aligned} \tag{12}$$

From the above equation, it can be possible to notice that it owns the following structure

$$y(k+1) = (1-\alpha)y(k) + \alpha\gamma u(k) \,, \tag{13}$$

where $(1-\alpha)$ is the eigenvalue responsible of the convergence time and $y(k)$, $u(k)$ are the output, input of discrete system, respectively. The corresponding system *transfer function* $G(z)$ is defined in the $\mathcal{Z}$ domain as

$$G(z) = \frac{Y(z)}{U(z)} = \frac{\alpha\gamma}{z-(1-\alpha)} \,, \tag{14}$$

from which it is possible to obtain the static gain $g$:

$$g = \lim_{z\to 1} G(z) = \gamma \,. \tag{15}$$

Therefore, when the tree reaches the steady state, the edge local value is given by

$$y(k) = \gamma u(k) \,, \tag{16}$$

with $y(k)$, $u(k)$ the output and input of the discrete system represented by the single tree edge, respectively. Equation (16) is repeated for each edge inside any feasible tree nodes sequence, i.e., linking the root node $s_H$ to the target node $T$. Then, this affects the reward propagation from $T$ to $s_H$ meaning that a longer sequence implies a larger damping of the tree weights, i.e., in proximity of the root node the edges belonging to longest sequences present smaller weight values. Considering an optimal solution, e.g., at tree level $m$, it is possible to compute (at steady state) the value of edge weights starting from node $s_H$, $\overline{Q}_k(s_H,a_k)$ as

$$\overline{Q}_k(s_H,a_k) = \gamma^m r_T \,, \tag{17}$$

where $r_T$ is the reward on the target $T$ object. Therefore, in presence of an optimal solution at level $m$ and a sub-optimal solution at level $b$, with $m < b$, it is straightforward that

$$\overline{Q}_m(s_H,a_m) > \overline{Q}_b(s_H,a_b) \,, \tag{18}$$

with actions $a_m, a_b$ corresponding to the optimal and sub-optimal branches, respectively. In virtue of the above consideration, at the end of the training, the optimal sequence of actions can be iteratively retrieved from the tree starting from the root node, $s_H$, as

$$a_k^\star = \max_{a\in\mathcal{A}} \overline{Q}_k(s_k,a) \,, \tag{19}$$

and with $s_{k+1} = Q_k(s_k,a_k^\star)$.

Concerning the computational complexity, the motion planner is activated only when a new node is added to the tree, since the corresponding trajectory can be stored and exploited in next episodes. The number of calls to the motion planner is equal to the number of edges at the end of the learning phase. It is also possible to elaborate the *difficulty* to find the optimal solution or a sub-optimal one. Equation (7) gives us the total number of potential paths to be checked. A subset $\xi$ of them brings the robot to the target and subset of the latter $\xi^*$ exhibits the lowest metrics.By defining as $\xi_t$ the total number of paths it is possible to introduce two metrics of difficulty in finding the optimum $\xi^*/\xi_t$ and a sub-optimum $\xi/\xi_t$. Intuitively, with high metrics, easy problem, all the algorithms should work well while with a more tough problem the algorithm H-εG will overcome the others.

Table 1: Scenario 1.

| Algorithm | Episodes | $MP_{call}$ | $1^{st}*$ |
|---|---|---|---|
| LRND | 144 | 42 | 20 |
| H-LRND | 119 | 37 | 18 |
| H-εG | 115 | 34 | 16 |

Table 2: Scenario 2.

| Algorithm | Episodes | $MP_{call}$ | $1^{st}*$ |
|---|---|---|---|
| LRND | 1079 | 236 | 96 |
| H-LRND | 1007 | 218 | 77 |
| H-εG | 717 | 188 | 74 |

## 4 NUMERICAL VALIDATION

Numerical simulations have been run in order to validate the proposed approach. The robot at hand is a 7-DoF Jaco$^2$ manufactured by Kinova available in the robotics lab of the University of Cassino. The implemented motion planning algorithm introduced in Sect. 3.1 is fully described in (Di Vito et al., 2020). In detail, the following tasks have been implemented: 1) Mechanical joint limits avoidance; 2) Self-hit avoidance; 3) Obstacle avoidance with respect to the $N_o$ objects and the static ones; 4) Position and orientation of the arm's end-effector.

Three different scenarios have been considered characterised by an increasing number of obstacles:

- **Scenario 1.** $N_o = 5$, $\xi_p = 10$. There are $\xi_p = 10$ possible sequence from the home to the target nodes. In particular, since one solution is at level 4 and the other at level $> 4$, the percentage of optimal solution is 0.3% while the percentage of sub-optimal solutions is $\approx 3\%$.
- **Scenario 2.** $N_o = 10$, $\xi_p = 20$. One solution is at level 5 and the other at level $> 5$, the percentage of optimal solution is thus $\approx 10^{-5}\%$ while the percentage of sub-optimal solutions is $\approx 10^{-4}\%$.
- **Scenario 3.** $N_o = 15$, $\xi_p = 20$. One solution is at level 6 and the other at level $> 6$, the percentage of optimal solution is $\approx 10^{-11}\%$ while the percentage of sub-optimal solutions is $\approx 10^{-10}\%$.

For all the learning algorithms (see Sect. 3.2.1), it is set $\alpha = 0.5$ and $\gamma = 0.9$ and the training is repeated 50 times from scratch. In the case of H-εG algorithm, it is $\varepsilon_0 = 1$ and $\varepsilon_{min} = 10^{-4}$ in (9). The training ends whenever condition eq. (11) is satisfied or the maximum number of episodes $Ep_{max} = 5000$ is reached. As reported in Figure 4, H-εG is the best algorithm when it comes to reach for the first time the optimal solution and the interrogation at motion planner system. In addition, in Figure 5 is possible to see that

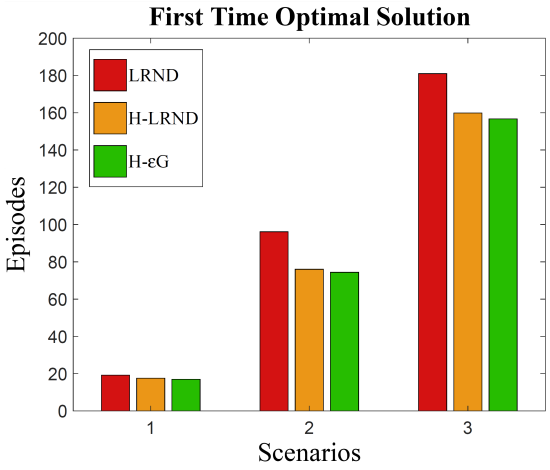


Figure 4: Comparison of three learning policies defined for each scenario. The average episodes is calculated on 50 training and represent the first time that the target $T$ is reached through optimal sequence $\xi^*$.

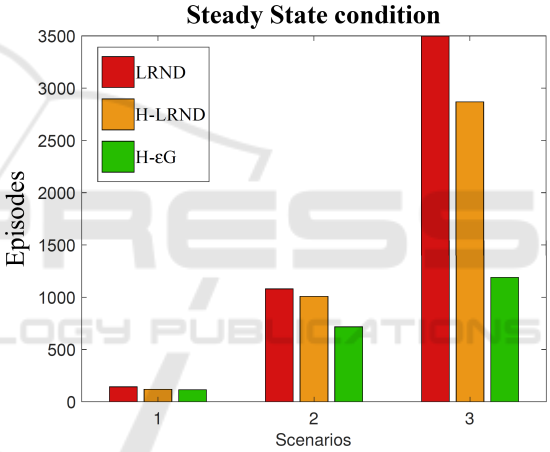


Figure 5: Comparison of convergence between learning algorithms defined above for each scenario. The average episodes number is calculated on 50 training.

the algorithm H-εG reach the steady state condition faster than other algorithms in every scenario for the ε value chosen. This latter depends on whether that the ε introduces variation on the learning dynamic.

An additional comparison with the no learning technique (RND) introduced in Sect. 3.2.1 is here considered. The implemented algorithm *simply* explores the graph resorting to a *Breadth First Search* policy by randomly choosing actions until the target is reached. This is clearly a fast method for a very

Table 3: Scenario 3.

| Algorithm | Episodes | $MP_{call}$ | $1^{st}*$ |
|---|---|---|---|
| LRND | 3496 | 665 | 181 |
| H-LRND | 2868 | 588 | 160 |
| H-εG | 1189 | 331 | 156 |

small number of objects, but that becomes quickly intractable. In fact, scenarios 2 and 3 described above could not be solved in a *reasonable* amount of time. Concerning the scenario 1 introduced above, 50 trials were executed and an optimal solution is found in $\approx$ 60% of cases (100% in the case of learning techniques). In the remaining 40% of trials, all obstacles are relocated leading to a sub-optimal solutions.

# 5 CONCLUSION

In this paper, a Reinforcement Learning approach aimed at performing robotic target relocation in cluttered environments is presented. In detail, the proposed method exploits, at a high level, a Q-learning approach on a dynamic tree structure in order to chose optimal sequences of obstacles to relocate while, at a low level, a constraint motion planning is adopted to plan feasible trajectories for object relocation. Several exploration strategies of the solution tree, based on a Breadth-First-Search technique, are presented and compared, showing that an ε-Greedy approach with heuristics outperforms other baseline methods and efficiently solve the problem.

Concerning future work, a pragmatic comprise between elaboration time and optimality will be sought by investigating also a Depth-First-Search paradigm, as well as different and more elaborated reward functions, so as to take into account energetic issues or other properties of the objects. Similarly, a comparison with a Deep Q-Network approach will be done.

# REFERENCES


Cheong, S. H., Cho, B. Y., Lee, J., Kim, C., and Nam, C. (2020). Where to relocate?: Object rearrangement inside cluttered and confined environments for robotic manipulation. *arXiv preprint arXiv:2003.10863*.

Dantam, N. T., Kingston, Z. K., Chaudhuri, S., and Kavraki, L. E. (2016). Incremental task and motion planning: A constraint-based approach. In *Robotics: Science and systems*, volume 12. Ann Arbor, MI, USA.

Deng, Y., Guo, X., Wei, Y., Lu, K., Fang, B., Guo, D., Liu, H., and Sun, F. (2019). Deep reinforcement learning for robotic pushing and picking in cluttered environment. In *2019 IEEE/RSJ International Conf. on Intelligent Robots and Systems (IROS)*, pages 619–626.

Di Lillo, P., Arrichiello, F., Di Vito, D., and Antonelli, G. (2020). BCI-controlled assistive manipulator: developed architecture and experimental results. *IEEE Trans. on Cognitive and Developmental Systems*.

Di Lillo, P., Simetti, E., Wanderlingh, F., Casalino, G., and Antonelli, G. (2021). Underwater intervention with remote supervision via satellite communication: Developed control architecture and experimental results within the dexrov project. *IEEE Trans. on Control Systems Technology*, 29(1):108–123.

Di Vito, D., Bergeron, M., Meger, D., Dudek, G., and Antonelli, G. (2020). Dynamic planning of redundant robots within a set-based task-priority inverse kinematics framework. In *2020 IEEE Conf. on Control Technology and Applications (CCTA)*, pages 549–554.

Dogar, M. R., Koval, M. C., Tallavajhula, A., and Srinivasa, S. S. (2014). Object search by manipulation. *Autonomous Robots*, 36(1-2):153–167.

Eppe, M., Nguyen, P. D., and Wermter, S. (2019). From semantics to execution: Integrating action planning with reinforcement learning for robotic causal problem-solving. *Frontiers in Robotics and AI*, 6:123.

Géron, A. (2019). *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media.

Jang, B., Kim, M., Harerimana, G., and Kim, J. W. (2019). Q-learning algorithms: A comprehensive classification and applications. *IEEE Access*, 7:133653–133667.

Lagriffoul, F., Dimitrov, D., Bidot, J., Saffiotti, A., and Karlsson, L. (2014). Efficiently combining task and motion planning using geometric constraints. *The International Journal of Robotics Research*, 33(14):1726–1747.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, 521(7553):436–444.

Lee, J., Cho, Y., Nam, C., Park, J., and Kim, C. (2019). Efficient obstacle rearrangement for object manipulation tasks in cluttered environments. In *2019 International Conf. on Robotics and Automation (ICRA)*, pages 183–189.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing ATARI with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Mohammed, M. Q., Chung, K. L., and Chyi, C. S. (2020). Review of deep reinforcement learning-based object grasping: Techniques, open challenges and recommendations. *IEEE Access*.

Nam, C., Lee, J., Cheong, S. H., Cho, B. Y., and Kim, C. (2020). Fast and resilient manipulation planning for target retrieval in clutter. *arXiv preprint arXiv:2003.11420*.

Siciliano, B. and Slotine, J.-J. E. (1991). A general framework for managing multiple tasks in highly redundant robotic systems. In *Proc. Fifth International Conf. on Advanced Robotics (ICAR)*, pages 1211–1216.

Srivastava, S., Fang, E., Riano, L., Chitnis, R., Russell, S., and Abbeel, P. (2014). Combined task and motion planning through an extensible planner-independent interface layer. In *2014 IEEE International Conf. on Robotics and Automation (ICRA)*, pages 639–646.

Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8(3-4):279–292.

Yuan, W., Hang, K., Kragic, D., Wang, M. Y., and Stork, J. A. (2019). End-to-end nonprehensile rearrangement with deep reinforcement learning and simulation-to-reality transfer. *Robotics and Autonomous Systems*, 119:119–134.