Noname manuscript No.
(will be inserted by the editor)

Objects Relocation in Clutter with Robot Manipulators via Tree-based Q-Learning Algorithm: Analysis and Experiments

**Giacomo Golluccio, Paolo Di Lillo,
Daniele Di Vito, Alessandro Marino,
Gianluca Antonelli**

the date of receipt and acceptance should be inserted later

Abstract This work addresses the problem of retrieving a target object from cluttered environment using a robot manipulator. In the details, the proposed solution relies on a Task and Motion Planning approach based on a two-level architecture: the high-level is a Task Planner aimed at finding the optimal objects sequence to relocate, according to a metric based on the objects weight; the low-level is a Motion Planner in charge of planning the end-effector path for reaching the specific objects taking into account the robot physical constraints. The high-level task planner is a Reinforcement Learning agent, trained using the information coming from the low-level Motion Planner. In this work we consider the *Q-Tree* algorithm, which is based on a dynamic tree structure inspired by the Q-learning technique. Three different RL-policies with two kinds of tree exploration techniques (Breadth and Depth) are compared in simulation scenarios with different complexity. Moreover, the proposed learning methods are experimentally validated in a real scenario by adopting a KINOVA Jaco^2 7-DoFs robot manipulator.

Keywords Motion Planning · Task Planning · Reinforcement Learning

1 Introduction

Robotic systems are deployed in industrial environment since decades for performing programmed routines in highly-structured environments, but recent changes in production processes require more flexible robots, capable of adapting to dynamic and unstructured environments. In particular, robotic manipulation of objects in clutter is a problem of growing interest, due the high number of application contexts, e.g. in manufacturing industry driven by Industry 4.0 requirements, warehousing logistics and in domestic environments

Authors are with the Department of Electrical and Information Engineering (DIEI), University of Cassino and Southern Lazio, {giacomo.golluccio, pa.dilillo, d.divito, al.marino, antonelli}@unicas.it

for the growing diffusion of service robots. Given the high dexterity and reasoning needed for accomplishing this task, it is currently performed mainly by human workers [1]. For this reason, many researchers have focused their efforts in the last years in designing methods and algorithms to be employed by robots for objects relocation.

The main aspects that make this research topic remarkably challenging are the unstructured nature of these environments combined to the $\mathcal{NP}$-hard complexity of the problem [2, 3]. In detail, retrieving a target from clutter might be intractable from a computational standpoint [4, 5], due to the combinatorial nature of the problem of rearranging the position of several obstacles to free up a path toward a target object [6, 7, 8].

In literature, the problem has been mainly addressed by relying on geometric methods that potentially make use of specific heuristics to reach effective solutions while reducing the computation complexity. In [4], the Authors propose a geometric method with the aim of minimizing the number of obstacles to be relocated, and consequently the time (or the energy) spent to the scope. Their algorithm is shown to be complete and efficient, outperforming other methods in terms of execution time, but the optimality of the solution is not guaranteed. Similarly, the planner in [9] solves the rearrangement problem exploiting dynamic nonprehensile actions guaranteeing only the feasibility of the plan.

In [10] a probabilistic solution, scalable with respect to the number of objects, is analyzed. In the proposed method the Authors choose the constraints to take into consideration depending on the motion feasibility. In case of an unfeasible motion, it is necessary to remove some of the constraints or to increase the motion planning timeouts in order to make the algorithm complete.

In [11], the Authors provide a geometric method for multiple objects reorganization in clutter, minimizing the number of objects to move. Differently from the above-mentioned works, here the optimal solution is obtained by splitting a continuous 2D plane into discrete cells, which are then used in a hybrid planner to generate the motion plan. However, the elementary motions that compose the plan are determined not considering the reachability of the objects and the robot kinematic constraints. For this reason, the plan might not be feasible for real robots. Alternatively, in [12] the Authors consider a novel approach based on Task Motion Planning (TMP) for unknown object rearrangements, relying on graphs that are built online in order to retrieve the target object. Their approach does not make use of heuristics in the exploration, potentially resulting in a large graph and not computationally efficient solution.

Recently the TMP problem, including object manipulation and grasping in a cluttered environment, have made use of machine and deep learning [13, 14] techniques. For instance, in [15] a neural approach is proposed, with a particular focus on considering unknown objects. In detail, they describe the Neural Rearrangement Planning, an approach to rearrange unknown objects from perceptual data in the real world. It is capable of rearranging previously unseen objects, exploiting segmented point-clouds coming from a RGB-D sensors.

Among the machine-learning techniques, Reinforcement Learning (RL) has been considered by many researchers as a base for solutions of this kind of problems [16]. It is a technique based on a figure named *agent*, which interacts with a space named *environment* choosing a possible *action*, which provides a feedback named *reward*. In particular, the agent objective is to collect the maximum reward values over time. The aim of this learning technique is to provide robots abilities like learning, improving over time, adapting and reproducing tasks [17]. The approach proposed in [13] combines the action planning with a goal-independent reinforcement learning approach considering a sparse reward. The problem is to find high-level actions to send to a low-level layer as trajectories for robots to solve random puzzles. The obtained results prove that the proposed approach is able to solve the task if the considered solution space is not too large. In [18], the Authors propose a data-driven method to be employed in case of an occluded target. They assign a probability distribution to the target object pose considering partial observations and an occlusion-aware heuristic, and then they exploit a receding horizon approach. They present an architecture that allows learning a generative model used to update the target pose probability distribution in a continuous action space.

There is a large amount of approaches that exploit a visual input, e.g. RGB or RGB-D images, mapping it into feasible actions to bring the agent towards the goal [19, 20]; to the scope, they make use of Convolutional Neural Networks (CNNs) combined with a policy-based Reinforcement Learning [21, 22, 23].

In our previous works [24, 25], we proposed a method to solve pick-and-place problems in clutter, exploiting a combination between Task and Motion Planning using an algorithm named Q-tree which is based on Q-learning [26]. In particular, this approach relies on a dynamic tree structure instead of the well-known Q-matrix, providing advantages like limitation of computational burden and easy implementation. The considered tree is built on a Control-Aware Motion Planner that relies on the feedback from a Task-Priority Inverse Kinematics Framework [27] that takes into account several kinematic constraints of the robotic manipulator, e.g. joint limits and obstacle avoidance.

In this paper we extend the formulation by adding the following contributions:

- the objects in the considered scenarios have different characteristics and the Reinforcement Learning (RL) agent learns the optimal sequence of objects to relocate;
- we perform an analysis on the learning parameters variation;
- Observation about the optimal choice of ε parameters related to the methods H-εG$_b$ and H-εG$_d$.

Additionally, we present experimental results on a KINOVA Jaco^2 7-DoFs manipulator to show the effectiveness of the proposed approach in a realistic scenario.

The rest of the paper is organized as follows: Sect. 2 gives the necessary mathematical background on Reinforcement Learning and rooted trees; in

Sect. 3 we present the formulation of the problem that we address; Sect. 4 describes the proposed solution; Sect. 5 focuses on the obtained simulative and experimental results, while in Sect. 6 we report conclusions and future work.

2 Mathematical background

2.1 Reinforcement Learning

Reinforcement Learning is the design of an agent which acts into an environment with the aim of maximizing cumulative reward signals.

In detail, at each timestep k, for a particular state s_k the agent selects an action a_k, receiving a scalar reward r_k. The idea is to maximize the expected reward over time. In particular, the decision-making problem is modelled as a Markov Decision Process (MDP), in which:

- $\mathcal{S}$ denotes the state space;
- $\mathcal{A}$ denotes the action space;
- $p = Pr(s_{k+1} = s' | s_k = s, a_k = a)$ is the transition probability;
- r is the reward function;
- $\gamma \in [0, 1]$ is the discount factor.

The agent estimates the goodness of a state in order to decide which action to perform at a particular time-step. To the scope, a possible method is to consider the *action-value* function, also known as the *Q-function*, defined as the expected sum of rewards after performing action a_k in the state s_k following a policy π:

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

The optimal action-value function is given by:

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

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

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

where $s_k = s, s_{k+1} = s', a_k = a, a_{k+1} = a'$.

Equation (3) assumes the knowledge of the state transition probability $p(s'|s, a)$, i.e., it is a *model-based* problem.
The same equation can be applied in a *model-free* problem ignoring the state transition probability. This is achieved by setting $p(s'|s, a) = 1$ and, therefore, obtaining:

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

Equation (4) is a recursive nonlinear function without a closed-form solution. The following iterative update law can be adopted:

$$Q_{k+1}(s_k, a_k) = Q_k(s_k, a_k) - \alpha\delta_k \ , \tag{5}$$

where $\alpha \in [0, 1]$ is known as *learning rate* and δ_k is the temporal difference error, defined as:

$$\delta_k = Q_k(s_k, a_k) - r_{k+1} - \gamma \max_{a \in \mathcal{A}} Q_k(s_{k+1}, a) \ . \tag{6}$$

Finally, by combining Eq. (6) and Eq. (5), one obtains:

$$Q_{k+1}(s_k, a_k) = Q_k(s_k, a_k) + \alpha(r_{k+1} + \gamma \max_{a \in \mathcal{A}} Q_k(s_{k+1}, a) - Q_k(s_k, a_k)) \ , \tag{7}$$

which is an iterative approach named *Q-learning* [26].

2.2 Rooted trees

A directed graph is an ordered pair $\mathcal{G} = \{\mathcal{V}, \mathcal{X}\}$, where $\mathcal{V} = \{\nu_1, \ldots, \nu_l\}$ is the set of nodes, or vertices, and $\mathcal{X} = \{\chi_1, \ldots, \chi_m\}$ is the set of oriented pairwise edges from node ν_i to ν_j. A scalar value might be assigned to edges; in this case the edge value is assumed to be 1 unless specified otherwise. A path between two nodes ν_i and ν_j is the set of directed edges through which a node ν_j can be reached from node ν_i. A graph $\mathcal{G}$ is defined cyclic if it contains a cycle, i.e., there is a subset of the edge set that forms a path such that the first node of the path corresponds to the last. On the opposite, if no cycle exists a graph is defined acyclic.

A tree $\mathcal{T}$ is an undirected acyclic graph such that there is a unique path between every pairs of vertices; this implies that in a tree of l nodes, $m = l - 1$ edges exist. A directed tree is a Directed Acyclic Graph (DAG) whose underlying undirected graph is a tree. In particular, in a directed rooted tree, given a node ν_i, there is exactly one edge from another node ν_j, called *parent*, to ν_i that is, then, a *child* of ν_j; then, every node has a unique parent except the root which has no parent and from which exactly one path exists to any other node of the tree; furthermore, a node with no child is a terminal node.

Moreover, the *depth* or *level* of a node ν_i is its distance from the root, i.e., the length of the unique path from the root to ν_i computed by summing the weights associated to the path. Thus, the root has depth 0.

3 Problem formulation

The aim of this work is to design a learning agent for solving the task of retrieving a target object T in clutter and moving it from an initial position $\boldsymbol{p}_{t,0} \in \mathbb{R}^3$ to a final position $\boldsymbol{p}_{t,f} \in \mathbb{R}^3$, relying on a Reinforcement Learning approach. Given the presence of N_o obstacles $\mathcal{O} = \{O_1, \ldots, O_{No}\}$ in the scene with assigned position $\boldsymbol{p}_{o,i} \in \mathbb{R}^3$ $(i = 1, 2, \cdots, N_o)$, the Reinforcement

Learning agent should relocate the obstacles that make T unreachable, in order to free up a path to reach it. An object is considered unreachable if it is not possible to find a trajectory that allows the robot to grasp and relocate it without hitting any obstacle. In the initial state s_H, the robot starts in a predefined joints configuration, all the obstacles O_i are in initial positions $\boldsymbol{p}_{o,i}$ and the target T is in the initial position $\boldsymbol{p}_{t,0}$.

The sequence $\mathcal{S}_u$ represents any sequence of objects with cardinality $|\mathcal{S}_u| = u$, and $\mathcal{S}_u^T$ is the sequence obtained from $\mathcal{S}_u$ adding as last element the target T, i.e. $\mathcal{S}_u^T = \{\mathcal{S}_u, T\}$. An object $O_i \in \mathcal{S}_u$ is considered relocatable if all the previous objects in the sequence $\mathcal{S}_u$ can be relocated. If all the objects into the sequence $\mathcal{S}_u$ are relocatable, the sequence is defined feasible.

Given that, in general, it is possible to find multiple feasible sequences, an optimization procedure can be designed in order to minimize a metric related to the energy spent during the relocation procedure. In this work, without loss of generality we consider that each obstacle has a weight ϕ_i in the range $(0, 1]$ Kg, that is the standard object weight normalized with respect to the KINOVA Jaco^2 7-DoFs maximum payload. It is worth noticing that the weight can be associated to any other physical characteristics of the objects, e.g. fragility or volume. In this perspective, the aim of the optimization problem is to find a feasible sequence $\mathcal{S}_u^T$ that minimizes the cost function:

$$\Phi_{\mathcal{S}_u} = \sum_{i \in \mathcal{S}_u} \phi_i. \tag{8}$$

Finally, we consider the following assumptions:

- the positions of the obstacles and the target are known beforehand;
- the relocation of an obstacle does not affect the positions of the other ones;
- it is not possible to grasp the obstacles and the target from the top, due to the presence, for example, of a shelf over the objects.

4 Proposed solution

In this section we describe our proposed solution to the above-mentioned problem. We have designed a two-layered architecture, as shown in Fig. 1 . In detail, the high-level is represented by the *RL-Task Planner* block, which is in charge of learning the optimal sequence of actions, i.e. obstacles to relocate, in order to free up a path toward the target; the low-level layer is the *Motion Planner* block, which provides a feedback about the feasibility, g_k, of a specific action, a_k, aimed at relocating an object in terms of fulfillment of robot kinematic constraints.

An action a_k represents a given object to relocate, which is considered unfeasible if the Motion Planner cannot find in a predefined amount of time an obstacle-free path that connects the end-effector initial configuration and the object. In case of a feasible action, the Motion Planner outputs the joint

velocities $\dot{\boldsymbol{q}}(t)$ that make the robot actually perform the object relocation, obtaining a new state s_k. In both cases, a reward r_k, depending on the feasibility g_k of the action a_k is generated.

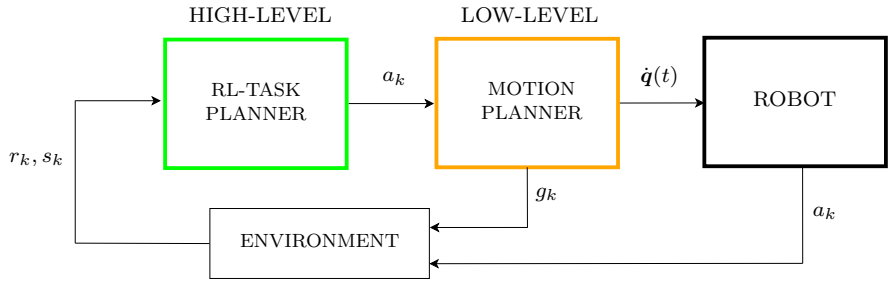

Fig. 1: Representation of the proposed architecture: the RL-Task Planner chooses the action a_k with an appropriate policy, while the Motion Planner provides information about the feasibility g_k of the chosen action a_k in terms of fulfillment of kinematic constraints. In case of a feasible action, the joint velocities vector $\dot{\boldsymbol{q}}(t)$ is sent to the Robot that actually performs it, relocating the object. The environment elaborates the information related to the feasibility g_k of the action and generates a reward signal r_k and the new state s_k, which are used to update the RL agent.

In the following subsections we will give algorithmic details about the two layers. It is worth noticing that the word *task* is used in the both the layers with different meanings. Within the low-level layer it represents the generic elementary control objectives used in the inverse kinematics framework, whereas for the high-level one, it represents a discrete planning for the RL agent.

4.1 Low-level: Motion Planner

As previously mentioned, the low-level Motion Planner provides information about the feasibility of the actions a_k requested by the high-level RL-Task Planner and computes the joints velocity vector $\dot{\boldsymbol{q}}(t)$ that allow the robot to reach and relocate the selected object. More in detail, it is composed by three processes (see Fig. 2): the actions-objects mapping, the sampling-based algorithm and the Set-based Task-Priority Inverse Kinematics (STPIK)-check [27].

First of all, the action a_k is translated in a desired end-effector pose $\boldsymbol{\eta}_{\mathrm{ee,d}}$, in which the desired position is set as the constant position of the object to relocate and the desired orientation is chosen in order to be suitable for the grasping phase. This information feeds the sampling algorithm, which is in charge of finding an obstacle-free, time-varying trajectory $\boldsymbol{\eta}_{\mathrm{ee,d}}(t)$ between the end-effector initial position $\boldsymbol{\eta}_{\mathrm{ee},0}$ and $\boldsymbol{\eta}_{\mathrm{ee,d}}$. The considered sampling algorithm is the Rapidly-exploring Random Tree (RRT) Connect [28] which is a bidirectional method based on growing two graphs rooted in $\boldsymbol{\eta}_{\mathrm{ee},0}$ and $\boldsymbol{\eta}_{\mathrm{ee,d}}$, respectively. Therefore, the algorithm is able to find a global path connecting the start and final points more efficiently than single tree search approaches.

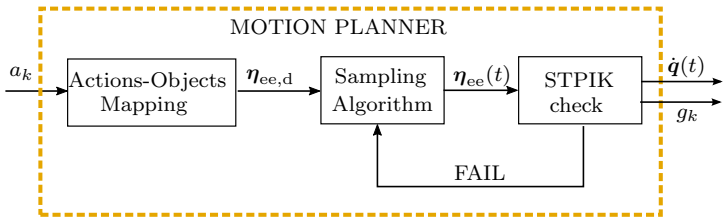

Fig. 2: Motion planner architecture, designed as three blocks: the Action-Objects Mapping translates the actions in constant desired end-effector poses; the Sampling Algorithm computes obstacle-free trajectories for the end-effector; the STPIK (Set-based Task-Priority Inverse Kinematics) checks the feasibility of the trajectories in terms of joint-level kinematic constraints (joint limits and self-hits).

In this step the sampling is performed in cartesian-space, and the constraints taken into account are defined only in cartesian-space as well, i.e. the output of the sampling-based algorithm is an obstacle-free path, obtained without considering all the joint-space safety tasks (e.g., joint limits and self-hits). They are handled by the STPIK-check block, which simulates the candidate trajectory $\boldsymbol{\eta}_{\mathrm{ee,d}}(t)$ and checks their fulfillment.

Considering a robot manipulator with n degrees of freedom and being $\boldsymbol{q} = [q_1 \dots q_n]^T$ its joint position vector, the STPIK algorithm allows to perform several tasks simultaneously. In detail, for a generic task $\sigma \in \mathbb{R}^v$, where $v \in \mathbb{N}^+$ is the task dimension, the Closed Loop Inverse Kinematics (CLIK) [29] algorithm can be applied in order to compute the needed joint velocities for achieving a specific desired value $\boldsymbol{\sigma}_d(t)$:

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^{\dagger}(\dot{\boldsymbol{\sigma}}_d + \boldsymbol{K}\tilde{\boldsymbol{\sigma}}) \,, \tag{9}$$

where $\boldsymbol{J}^{\dagger}$ is the Moore-Penrose pseudoinverse of the task Jacobian matrix $\boldsymbol{J}(\boldsymbol{q}) \in \mathbb{R}^{6\times n}$, $\dot{\boldsymbol{\sigma}}_d \in \mathbb{R}^v$ is the time derivative of $\boldsymbol{\sigma}_d$, $\boldsymbol{K} \in \mathbb{R}^{v\times v}$ is a positive-definite gain matrix and $\tilde{\boldsymbol{\sigma}} = \boldsymbol{\sigma}_d(t) - \boldsymbol{\sigma}(t)$ is the task error.

For a redundant robot the number of joints n is greater than the task dimension v, and such redundancy can be exploited to perform multiple tasks simultaneously. In particular, the elementary tasks can be arranged in a hierarchy $\mathcal{H}$ and the solution can be computed by projecting the velocity components of the lower priority tasks onto the null space of the higher priority ones, in order to filter out the components that will affect them. In this way, the accomplishment of the primary task is always guaranteed, while the lower-priority ones are executed *at best*. Therefore, considering a hierarchy composed by K tasks, the joint velocities $\dot{\boldsymbol{q}}$ can be computed recursively as [30]:

$$\dot{\boldsymbol{q}}_K = \sum_{i=1}^{K} (\boldsymbol{J}_i \boldsymbol{N}_{i-1}^A)^{\dagger} (\dot{\boldsymbol{\sigma}}_{i,d} + \boldsymbol{K}_i \tilde{\boldsymbol{\sigma}}_i - \boldsymbol{J}_i \dot{\boldsymbol{q}}_{i-1}) \,, \tag{10}$$

where $\boldsymbol{N}_i^A$ is the null space of the augmented Jacobian matrix $\boldsymbol{J}_i^A$, obtained by stacking the task Jacobian matrices from task 1 to i. This task priority-framework has been extended to handle also tasks in which the control objective is to keep the task value $\boldsymbol{\sigma}$ within a certain set, i.e. above a lower threshold

and below an upper one. This is the case of tasks such as joint limits, where the control objective is to keep the joints position within their physical limits, or obstacle avoidance, where the control objective is to keep the end-effector of the manipulator at a minimum distance from potential obstacles. This kind of tasks has been defined as *set-based*, and their handling within classical task-priority framework is managed through a proper insertion/removal of tasks in the hierarchy. For more details about the specific employed algorithm, the reader is referred to [31] and [32].
In case of a constraint violation, a *fail* signal is sent to the sampling algorithm block that generates a new $\boldsymbol{\eta}_{\mathrm{ee,d}}(t)$, starting a new iteration. The process terminates when the sampling algorithm finds a path that overcomes the IK-check, or when it cannot find any feasible path after a predefined amount of time.

After the execution of the algorithm, the label g_k is set according to the result of the planning. In particular:

$$g_k = \begin{cases} g_U & \text{if } a_k \text{ is unfeasible} \\ g_O & \text{if } a_k \text{ is feasible and it is related to the relocation of obstacle } O_i \\ g_T & \text{if } a_k \text{ is feasible and it is related to the relocation of the target } T \end{cases} \tag{11}$$

4.2 High level: RL-Task Planner

The high-level RL-Task Planner is in charge of learning the optimal sequence of obstacles O_i to relocate in order to reach the target T. Defining as $\mathcal{S}$ the manifold of all the possible sequences composed by $j \leq N_o$ obstacles and the target T, its cardinality is:

$$\xi_t = \sum_{j=1}^{N_o} j! \binom{N_o}{j} , \tag{12}$$

Let us define as $a_k \in \mathcal{A}_k = \{a_1, \cdots, a_{N_o}, a_T\}$ the action chosen at a specific timestep k. Within the set $\mathcal{A}_k$, the terms a_i $(i = 1, \cdots, N_o)$ are related to the relocation of the objects O_i, whereas a_T represents the action to relocate the target T.

When the RL-Task Planner selects the action a_k in a given state s_k, it receives a reward r_k that depends on its feasibility g_k:

$$r_k(s_k, a_k) = \begin{cases} -\phi_i, & \text{if } g_k = g_O \\ -10, & \text{if } g_k = g_U \\ 100, & \text{if } g_k = g_T \end{cases} , \tag{13}$$

where ϕ_i is the weight of the object related to the action a_k.

The RL-Task Planner has to choose the action a_k that maximizes the reward r_k for a given state s_k.

In our previous work [24, 25], we presented the Q-Tree, a technique that replaces the well-known static Q-Matrix of the Q-learning algorithm with a dynamic rooted tree structure that is built over $E_{\max}$ episodes of the algorithm. The advantage of this approach regards the computational burden. Indeed, for how we represent the state s_k (e.g. sequences of relocated objects until timestep k), considering the classical Q-Learning, all the possible combination must be allocated a-priori. Due the combinatorial nature of the problem, it is very hard, while considering the Q-Tree, the pre-allocation is not necessary because the structure is built dynamically.

Each node of the tree represents a state s_k, with an associated sequence $\mathcal{S}_{s_k} = \{O_{i_1}, O_{i_2}, \cdots, O_{i_U}\}$, that contains information about the objects O_{i_x} relocated until timestep k. In this work, the value associated to edge χ_k between s_k and s_{k+1} is set as the Q-value Q_{χ_k}, and updated at each time-step as in Eq. (7). At the beginning of the first episode, the tree is initialized with a root node s_H, that represents the initial state in which the robot is in a predefined joints configuration and all the objects are in their initial positions. Within each episode $E_h (h = 1, \ldots, E_{\max})$, the tree gets updated at each time-step k after the choice of an action a_k as follows:

1. if $O_{i_x} \notin \mathcal{S}_{s_k}$, meaning that the action a_k related to the object O_i has not been chosen in any previous episode, a new node s_{k+1} with associated sequence $\mathcal{S}_{s_{k+1}} = \{\mathcal{S}_{s_k}, a_k\}$ is allocated, updating the edge value between s_k and s_{k+1};
2. if $O_{i_x} \in \mathcal{S}_{s_k}$, meaning that the action a_k related to the object O_i has already been chosen in a previous episode, the edge that connects the nodes s_k and s_{k+1} is updated without allocating a new node.

The episode terminates when the robot reaches the target T, or when a maximum number of iteration $I_{\max}$ is reached.

Then, at the beginning of each one of the following episodes, the algorithm starts again from the node s_H keeping the tree structure built until that episode. It is worth noticing that given a specific s_k, a_k is selected from a set that does not contain actions already considered unfeasible in previous episodes in order to speed up the process. This is valid for all the proposed exploration policies.

An example of the Q-Tree algorithm is reported in Fig. 3.

4.2.1 Exploration policies

In this Section we describe the Q-Tree exploration policies that will be compared in Sec. 5. Given a node s_k with an associated $\mathcal{S}_{s_k}$, the RL-Task Planner chooses a_k with the following possible policies:

- **Learning Random Exploration**-(**LRND**): it chooses $a_k \in \mathcal{A}_k$ in a completely random manner.
- **Random Exploration with Heuristics**-(**H-LRND**): it at first tries to reach the target (choosing a_{k_T}). If it is unfeasible, a_k is chosen randomly among the other actions in $\mathcal{A}_k$.

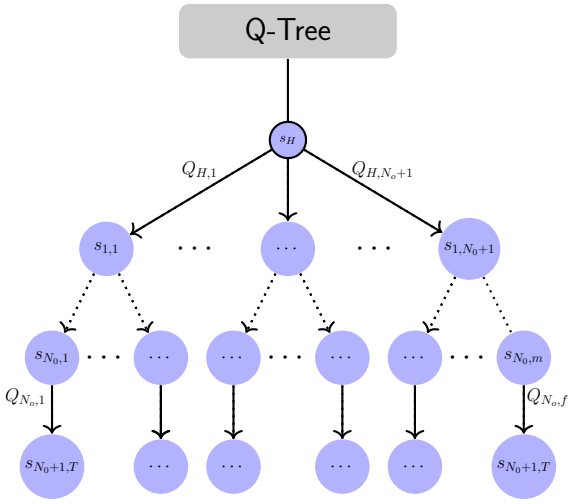

Fig. 3: Example of complete Q-Tree: The node root s_H represents the initial configuration where all objects are in the initial location. The last nodes contain the target T.

– **ε-Greedy Exploration with Heuristics**-(H-εG): it chooses a_k exploiting the ε-Greedy technique. In detail, it chooses a random action a_k with probability ε and the action associated with the maximum edge value with probability $1-\varepsilon$ (this corresponds to the $\max_{a'} Q_*(s', a')$ in Eq. (3)).

Regarding the ε-Greedy Exploration with Heuristics policy, in order to ease the exploitation, ε is updated at the beginning of each episode E_h as:

$$\varepsilon(h) = \varepsilon_0 \left(\frac{\varepsilon_{\min}}{\varepsilon_0} \right)^{\frac{h-1}{E_{\max}-1}}, \tag{14}$$

where ε_0 and $\varepsilon_{\min}$ are the initial and the minimum ε value respectively.

4.2.2 Search methods

For each one of Q-Tree exploration policies, we compare two different search methods in order to investigate their effect on the learning dynamics:

– *Breadth*: the RL-Task Planner explores an entire tree level before moving on the following levels;
– *Depth:* the RL-Task Planner prefers moving towards nodes at following levels rather than the ones at the same level.

In other words, the employed search method affects the behavior of the RL-Task Planner in case of an unfeasible action is chosen. Considering the breadth method the current episode is terminated, whereas considering the depth method it continues choosing another action among the possible ones.

4.2.3 Q-Tree learning algorithm

In this subsection we give details about the learning algorithm employing the Q-Tree structure. The inputs of the algorithm are:

1. number of obstacles;
2. robot end-effector pose;
3. search method;
4. exploration policy.

At the beginning of the first episode, the Q-Tree is initialized allocating the root node s_H. At each timestep k, the RL-Task Planner chooses an action $\overline{a}_k$ following the selected policy and search method and it queries the low-level Motion Planner, which tries to plan a trajectory for reaching the object associated to a_k. It returns the information about the action feasibility g_k that is then used to compute the reward r_k according to Eq. (13). At this point the tree is updated adding a new node and updating the corresponding edge value. The procedure gets iterated until one of the termination conditions of the episode is met.

The training phase terminates when the maximum number of episodes $E_{\max}$ is reached, or when the Q-Tree is not significantly updated anymore over consecutive episodes, both in terms of new nodes allocation and edge values. In detail, defining as:

$$\overline{Q}_h = \sum_{j=1}^{l-1} |Q_{\chi_j}| \,, \tag{15}$$

the sum of all the edge values at the end of the episode h, the termination condition is:

$$|\overline{Q}_h - \overline{Q}_{h-1}| \leq \beta \,, \tag{16}$$

where $\beta > 0$ is a predefined threshold.

4.2.4 Optimality and complexity analysis

Upon completion of the training phase, the tree is completely built and it is possible to find a number of sequences of obstacles to relocate in order to reach the target. The optimal sequence is the one corresponding to the nodes related to the highest edge values, i.e. the maximum of the Q-function.

The computational complexity is strictly related to the number of queries to the Motion Planner. It is worth noticing that during the training, the motion planner is queried only if a new node must be added, due to the fact that each computed trajectory is stored and exploited in future episodes. Upon termination of training, the number of motion planner queries is equal to the tree edges number.

Algorithm 1: Q-Tree

```
Data:
    ObstaclesNumber
    RobotPose // End-Effector position
    SearchMethods // Breadth, Depth
    ExplorationPolicy // LRND,H-LRND,H-εG
Result: ActionsSequence
// Allocation of the Tree root node
CreateRootNode();
Q̄_0 = 0 // Starting training for the RL-Task Planner
while h ← 1 < E_max and |Q̄_h − Q̄_{h−1}| > β do
    // Reset scene to initial condition
    CurrentState = s_H ;
    for j ← 1 to I_max do
        // Selection next action
        NextAction = TaskPlanner(Policy);
        // Feedback Motion Planner
        MotionPlanner(CurrentState,NextAction);
        r = getReward();
        // Update
        if IsInTree(CurrentState) == false then
            AddNode(CurrentState);
        end
        UpdateTreeEdgesValue();
    end
end
ActionsSequence = SelectSequenceWithMaximumEdgeValues();
```

5 Simulation and experiments

The solution described in Sect. 4 is here validated via an extensive simulation campaign and experiments. In detail, in Sect. 5.1 the exploration strategies presented in Sect. 4.2 are compared on scenarios with growing complexity, while experimental results in a realistic case study are presented in Sect. 5.2.

5.1 Simulation results

All the simulation are performed using the visualizer RViz. Three different scenarios are simulated, differentiated in terms of objects number N_o to relocate:

- **Scenario 1 (easy):** $N_o = 5$ obstacles are considered, $\xi_p = 10$ feasible sequences from the root node s_H to the target nodes s_T in Fig. 3, where 9 are sub-optimal and 1 is optimal;
- **Scenario 2 (medium):** $N_o = 10$ obstacles are considered with $\xi_p = 20$ possible sequences from the initial state s_H to the target nodes s_T, where 19 are sub-optimal and 1 is optimal;

– **Scenario 3 (hard):** $N_o = 15$ obstacles are considered with $\xi_p = 20$ possible sequences from the initial state s_H to the target nodes s_T, where 19 are sub-optimal and 1 is optimal.

In order to assess the performance of the proposed relocation strategy, a parametric analysis has been conducted considering the following parameters:

– learning rate α, variable in the interval $[0.3 - 1]$;
– discount factor γ, variable in the interval $[0.1 - 0.9]$.

In addition, ε_0 and $\varepsilon_{\min}$ in Eq.(14) are set to 1 and 10^{-3}, respectively.

Three performance indices are considered: (i) the number of episodes $\mathrm{E_{ss}}$ effectively required to train the agent bringing at steady state condition the tree; (ii) the number of queries MP_q to the motion planner described in Sect. 4.1; (iii) the number of episodes $\mathrm{E}_{1_{\mathrm{st}}}$ required to find the first optimal solution.

The maximum number of episodes $\mathrm{E_{max}}$ for each training in Algorithm 1 is set to $\mathrm{E_{max}} = 10000$ for the LRND and H-LRND techniques (both breadth and depth search methods), while for the ε-greedy techniques, $\mathrm{E_{max}}$ is set to the maximum number of episodes required by the LRND to converge.

Figure 4 reports, as an example, the evolution of $\overline{Q}_h$ in Eq. (15) normalized with respect to its steady state value, $\overline{Q}_\infty$, in the case of H-LRND$_b$ and Scenario 1 for different values of α and γ. The figure shows how the tree reaches a steady value after a certain number of episodes and how α and γ affect the converge in this specific training instance.

The simulation results are reported both numerically in Tables 1–3 and graphically in Figs. 5–7; they are computed running each case study 50 times and averaging the obtained results. In the following, the subscripts b and d represent the Breadth and Depth exploration, respectively. As a general consideration, the H-εG$_d$ algorithm exhibits optimal or slightly sub-optimal performance for each of the three considered metrics and scenarios. More in detail, H-εG$_d$ significantly outperforms the other methods concerning E_{ss} especially in case of complex scenarios (see Table 3). With regards to motion planning queries MP_q, which equals the number of explored tree branches, H-εG$_b$ outperforms the other ones and slightly H-εG$_d$. Furthermore, regarding the number of episodes $\mathrm{E}_{1_{st}}$ necessary for the learning agent to find the first optimal path, it is worth noticing that depth approaches exhibits significantly better performance with respect to breadth ones with very similar results for the three exploration policies LNRD, H-LNRD, H-εG.
Finally, increasing the learning rate α or decreasing the discount factor γ lead to better performance for each of the proposed exploration strategies and performance indexes; in detail, by adopting $\alpha = 1$ combined with heuristics approaches, the agent learns more quickly the optimal solution.

In support of Fig. 8 the three exploration policies are described in Sect. 4.2. Those are compared with respect to the performance indexes described above. As it is possible to notice, the magnitude of each indicator is reduced significantly with respect to the Breadth approach by using the Depth approach.

In Figs. 9 and 10, the minimum and maximum average values related to the first time in which the agent reaches the target (through the optimal

Table 1: Parametric analysis considering (α, γ) parameters and as evaluation criteria: the number of episodes E_{ss} necessary to converge, the number of motion planning queries MP_q and the the episode number $E1_{st}$ in which the optimal solution is reached for the first time. This case is for the Scenario 1 with obstacles number $N_o = 5$.

$[\alpha, \gamma]$	Algorithm	E_{ss}	MP_q	$E1_{st}$	$[\alpha, \gamma]$	Algorithm	E_{ss}	MP_q	$E1_{st}$	$[\alpha, \gamma]$	Algorithm	E_{ss}	MP_q	$E1_{st}$
[0.3,0.9]	LRND$_b$	511	84	50	[0.3,0.5]	LRND$_b$	441	84	51	[0.3,0.1]	LRND$_b$	283	84	43
	H-LRND$_b$	487	74	32		H-LRND$_b$	422	74	32		H-LRND$_b$	267	74	28
	H-εG$_b$	365	69	32		H-εG$_b$	289	67	39		H-εG$_b$	235	73	28
	LRND$_d$	449	84	30		LRND$_d$	388	84	26		LRND$_d$	233	84	28
	H-LRND$_d$	414	74	14		H-LRND$_d$	372	74	7		H-LRND$_d$	203	74	8
	H-εG$_d$	364	72	12		H-εG$_d$	315	71	15		H-εG$_d$	165	73	12
[0.5,0.9]	LRND$_b$	355	84	52	[0.5,0.5]	LRND$_b$	296	84	48	[0.5,0.1]	LRND$_b$	246	84	48
	H-LRND$_b$	344	74	31		H-LRND$_b$	282	74	32		H-LRND$_b$	212	74	29
	H-εG$_b$	229	64	34		H-εG$_b$	245	67	34		H-εG$_b$	176	71	32
	LRND$_d$	291	84	22		LRND$_d$	245	84	18		LRND$_d$	176	84	18
	H-LRND$_d$	272	70	12		H-LRND$_d$	226	74	11		H-LRND$_d$	149	74	12
	H-εG$_d$	271	70	14		H-εG$_d$	214	70	9		H-εG$_d$	141	73	12
[1,0.9]	LRND$_b$	170	84	50	[1,0.5]	LRND$_b$	173	84	51	[1,0.1]	LRND$_b$	177	84	45
	H-LRND$_b$	151	74	29		H-LRND$_b$	152	74	34		H-LRND$_b$	142	74	33
	H-εG$_b$	146	65	39		H-εG$_b$	148	64	29		H-εG$_b$	131	69	30
	LRND$_d$	122	84	26		LRND$_d$	104	84	23		LRND$_d$	114	84	20
	H-LRND$_d$	94	74	13		H-LRND$_d$	95	74	11		H-LRND$_d$	87	74	10
	H-εG$_d$	82	70	12		H-εG$_d$	86	70	13		H-εG$_d$	79	72	13

Table 2: Parametric analysis considering (α, γ) parameters and as evaluation criteria: the number of episodes E_{ss} necessary to converge, the number of motion planning queries MP_q and the the episode number $E1_{st}$ in which the optimal solution is reached for the first time. This case is for the Scenario 2 with obstacles number $N_o = 10$.

$[\alpha, \gamma]$	Algorithm	E_{ss}	MP_q	$E1_{st}$	$[\alpha, \gamma]$	Algorithm	E_{ss}	MP_q	$E1_{st}$	$[\alpha, \gamma]$	Algorithm	E_{ss}	MP_q	$E1_{st}$
[0.3,0.9]	LRND$_b$	1606	623	449	[0.3,0.5]	LRND$_b$	1383	622	437	[0.3,0.1]	LRND$_b$	1043	622	470
	H-LRND$_b$	1596	598	410		H-LRND$_b$	1279	598	426		H-LRND$_b$	952	598	414
	H-εG$_b$	1026	518	427		H-εG$_b$	868	572	407		H-εG$_b$	930	593	421
	LRND$_d$	1035	622	19		LRND$_d$	813	622	16		LRND$_d$	485	622	16
	H-LRND$_d$	974	598	16		H-LRND$_d$	688	598	15		H-LRND$_d$	382	598	16
	H-εG$_d$	673	594	15		H-εG$_d$	553	595	13		H-εG$_d$	299	596	15
[0.5,0.9]	LRND$_b$	1275	622	451	[0.5,0.5]	LRND$_b$	1118	622	451	[0.5,0.1]	LRND$_b$	939	622	444
	H-LRND$_b$	1211	598	414		H-LRND$_b$	1061	598	420		H-LRND$_b$	887	598	430
	H-εG$_b$	853	491	411		H-εG$_b$	718	548	398		H-εG$_b$	861	593	412
	LRND$_d$	647	622	15		LRND$_d$	564	622	14		LRND$_d$	396	622	13
	H-LRND$_d$	594	598	19		H-LRND$_d$	458	598	13		H-LRND$_d$	293	598	13
	H-εG$_d$	468	585	15		H-εG$_d$	388	592	13		H-εG$_d$	285	597	16
[1,0.9]	LRND$_b$	910	622	432	[1,0.5]	LRND$_b$	894	622	433	[1,0.1]	LRND$_b$	823	621	451
	H-LRND$_b$	855	597	419		H-LRND$_b$	867	598	400		H-LRND$_b$	800	597	423
	H-εG$_b$	690	456	375		H-εG$_b$	630	519	414		H-εG$_b$	771	590	416
	LRND$_d$	268	622	16		LRND$_d$	255	622	21		LRND$_d$	217	621	10
	H-LRND$_d$	227	597	15		H-LRND$_d$	226	598	12		H-LRND$_d$	190	598	19
	H-εG$_d$	219	572	21		H-εG$_d$	222	581	13		H-εG$_d$	181	592	14

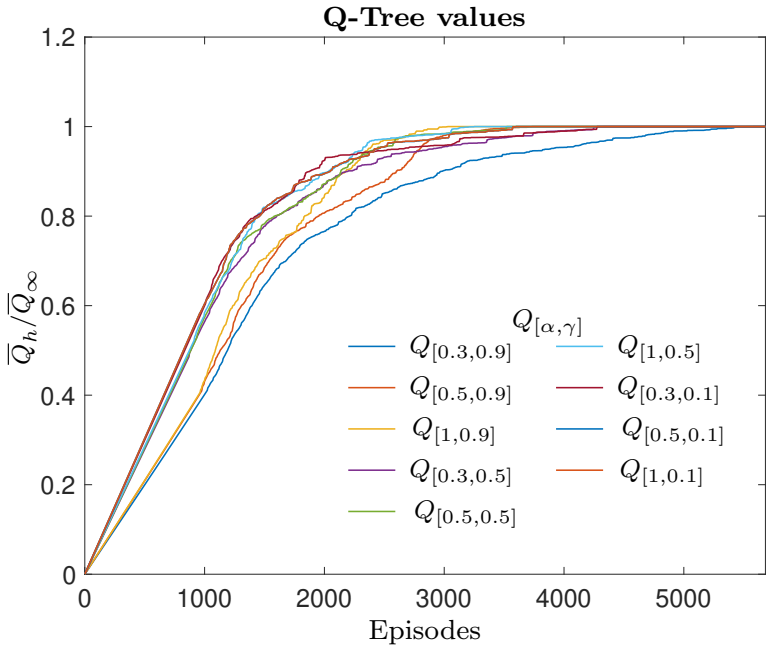

Fig. 4: Plots represent the sum of all tree edge values $\overline{Q}_h$ normalized with respect to the steady state value $\overline{Q}_\infty$ for different values of α and γ in the case of H-LRND$_b$ approach and Scenario 3.

sequence) for each scenarios are reported. As can be seen, in terms of reaching for the first time the optimal solution, the Depth drastically outperforms the Breadth approach.

In order to verify the robustness of the methods with respect to the objects position, we have validated our algorithm considering 10 additional simulations with random objects position for Scenario 1 with $N_o = 5$.
Table 4 shows the obtained results, which demonstrates that the considerations above are still valid in these new scenarios.

5.2 Experimental results

In the experimental case study, $N_o = 10$ obstacles (bottles) are considered. Furthermore, we assume that the perception module is enabled to detect and recognize obstacles and target, providing their poses accurately. The scenario is shown in Fig. 12 and two shelves are present: the top shelf is the starting location for both objects and target, which needs to be relocated to the bottom shelf. Objects are initially positioned on the top shelf as in Fig. 12; in addition, objects have different colors meaning they have different weights ϕ_i such as the green bottle is the target, while $\phi_1 = 0.15, \phi_2 = 0.13, \phi_3 = 0.23, \phi_4 = 0.44$, $\phi 5 = 0.65$, $\phi_6 = 0.78$, $\phi_7 = 0.13$, $\phi_8 = 0.34$, $\phi_9 = 0.56$, $\phi_{10} = 0.89$ Kg.

Table 3: Parametric analysis considering (α, γ) parameters and as evaluation criteria: the number of episodes E_{ss} necessary to converge, the number of motion planning queries MP_q and the the episode number $E1_{st}$ in which the optimal solution is reached for the first time. This case is for the Scenario 3 with obstacles number $N_o = 15$.

$[\alpha,\gamma]$	Algorithm	E_{ss}	MP_q	$E1_{st}$	$[\alpha,\gamma]$	Algorithm	E_{ss}	MP_q	$E1_{st}$	$[\alpha,\gamma]$	Algorithm	E_{ss}	MP_q	$E1_{st}$
[0.3,0.9]	LRND$_b$	5035	1674	1049	[0.3,0.5]	LRND$_b$	4519	1672	1021	[0.3,0.1]	LRND$_b$	3542	1658	1045
	H-LRND$_b$	4802	1605	958		H-LRND$_b$	3965	1600	1018		H-LRND$_b$	3213	1585	968
	H-εG$_b$	2829	1353	987		H-εG$_b$	2809	1520	993		H-εG$_b$	3051	1572	978
	LRND$_d$	2511	1675	11		LRND$_d$	1712	1675	13		LRND$_d$	708	1673	11
	H-LRND$_d$	2454	1606	12		H-LRND$_d$	1694	1606	11		H-LRND$_d$	684	1605	13
	H-εG$_d$	1342	1587	10		H-εG$_d$	777	1597	11		H-εG$_d$	658	1600	10
[0.5,0.9]	LRND$_b$	4050	1670	1040	[0.5,0.5]	LRND$_b$	3730	1664	1069	[0.5,0.1]	LRND$_b$	3293	1653	1054
	H-LRND$_b$	3985	1599	963		H-LRND$_b$	3579	1593	971		H-LRND$_b$	3047	1581	995
	H-εG$_b$	2397	1299	984		H-εG$_b$	2593	1506	992		H-εG$_b$	2981	1571	955
	LRND$_d$	1590	1675	14		LRND$_d$	1712	1674	13		LRND$_d$	565	1672	13
	H-LRND$_d$	1562	1606	10		H-LRND$_d$	1694	1606	11		H-LRND$_d$	530	1603	8
	H-εG$_d$	906	1572	13		H-εG$_d$	553	1588	11		H-εG$_d$	499	1596	9
[1,0.9]	LRND$_b$	3278	1655	1027	[1,0.5]	LRND$_b$	3245	1653	1025	[1,0.1]	LRND$_b$	3154	1649	1067
	H-LRND$_b$	3027	1580	987		H-LRND$_b$	3038	1581	980		H-LRND$_b$	2931	1578	956
	H-εG$_b$	2380	1268	1022		H-εG$_b$	2452	1500	995		H-εG$_b$	2832	1566	982
	LRND$_d$	618	1672	12		LRND$_d$	564	1670	10		LRND$_d$	451	1669	13
	H-LRND$_d$	594	1604	15		H-LRND$_d$	634	1605	11		H-LRND$_d$	430	1602	12
	H-εG$_d$	414	1530	12		H-εG$_d$	382	1556	11		H-εG$_d$	383	1591	11

Objects are relocated by the 7-DoF Jaco^2 robot equipped by a three finger gripper, manufactured by KINOVA company and available in the LAI-Robotics laboratory of the University of Cassino and Southern Lazio, Italy.

Figure 11 shows the Denavit-Hartenberg parameters (also see Table 5) summarizing the kinematics parameters of the robot, whereas Table 6 and Table 7 report the mechanical joint position and velocity limits, respectively.

In order to acquire the objects' position, an INTEL RealSense D455 RGB-D camera is adopted together with an ArUco marker [33] which provides a common reference frame and YOLO (You Only Look Once) software [34] for object segmentation. A desktop PC with CPU Intel(R) Core(TM) i9-9900KF 3.60GHz and GPU GeForce RTX 2070 Super equipped with Ubuntu 18.04 and MATLAB 2020b is adopted in order to find the optimal sequence by running the RL-Task Planner in Fig. 1. The same PC is equipped with the ROS (*Robot Operating System*) framework which is in charge to execute the Motion Planner and control the real robot. The software-hardware architecture is reported in Fig. 13. More in detail, the perception module receives the real scene from the camera and provides information on the objects pose to the RL-Task Planner implemented in MATLAB. This latter selects an action according with its policy and sends it to the Motion Planner implemented in C++. Finally, if there is a free-obstacle path that satisfies all the joint constraints, the robot receives the computed joint velocities to perform the task.

Alg.	E_{ss}	MP_q	$E_{1_{st}}$
$LRND_b$	177	97	70
H-$LRND_b$	168	84	41
H-εG_b	127	59	55
$LRND_d$	98	97	10
H-$LRND_d$	89	84	11
H-εG_d	84	75	11

(a) Case n. 1

Alg.	E_{ss}	MP_q	$E_{1_{st}}$
$LRND_b$	200	84	48
H-$LRND_b$	185	73	23
H-εG_b	114	57	24
$LRND_d$	130	83	5
H-$LRND_d$	100	55	5
H-εG_d	47	48	4

(b) Case n. 2

Alg.	E_{ss}	MP_q	$E_{1_{st}}$
$LRND_b$	167	98	45
H-$LRND_b$	153	88	36
H-εG_b	107	66	26
$LRND_d$	96	98	14
H-$LRND_d$	83	88	16
H-εG_d	76	79	12

(c) Case n.3

Alg.	E_{ss}	MP_q	$E_{1_{st}}$
$LRND_b$	167	94	37
H-$LRND_b$	151	83	27
H-εG_b	129	64	33
$LRND_d$	94	93	24
H-$LRND_d$	80	83	8
H-εG_d	69	76	5

(d) Case n. 4

Alg.	E_{ss}	MP_q	$E_{1_{st}}$
$LRND_b$	200	104	31
H-$LRND_b$	170	87	13
H-εG_b	112	54	12
$LRND_d$	119	103	12
H-$LRND_d$	99	87	7
H-εG_d	69	76	6

(e) Case n. 5

Alg.	E_{ss}	MP_q	$E_{1_{st}}$
$LRND_b$	187	84	26
H-$LRND_b$	114	57	14
H-εG_b	93	42	17
$LRND_d$	122	84	5
H-$LRND_d$	58	57	3
H-εG_d	46	47	4

(f) Case n. 6

Alg.	E_{ss}	MP_q	$E_{1_{st}}$
$LRND_b$	166	91	32
H-$LRND_b$	135	71	20
H-εG_b	111	55	17
$LRND_d$	100	92	12
H-$LRND_d$	77	71	8
H-εG_d	66	63	7

(g) Case n. 7

Alg.	E_{ss}	MP_q	$E_{1_{st}}$
$LRND_b$	130	71	30
H-$LRND_b$	94	49	16
H-εG_b	84	41	16
$LRND_d$	82	71	18
H-$LRND_d$	53	49	8
H-εG_d	41	44	8

(h) Case n. 8

Alg.	E_{ss}	MP_q	$E_{1_{st}}$
$LRND_b$	206	82	25
H-$LRND_b$	166	55	17
H-εG_b	112	40	15
$LRND_d$	117	83	4
H-$LRND_d$	84	55	4
H-εG_d	56	46	2

(i) Case n. 9

Alg.	E_{ss}	MP_q	$E_{1_{st}}$
$LRND_b$	162	82	15
H-$LRND_b$	138	61	5
H-εG_b	95	38	5
$LRND_d$	103	82	6
H-$LRND_d$	85	61	5
H-εG_d	58	51	5

(j) Case n. 10

Table 4: Additional Simulation considering 10 random cluttered different scenarios with $N_o = 5$.

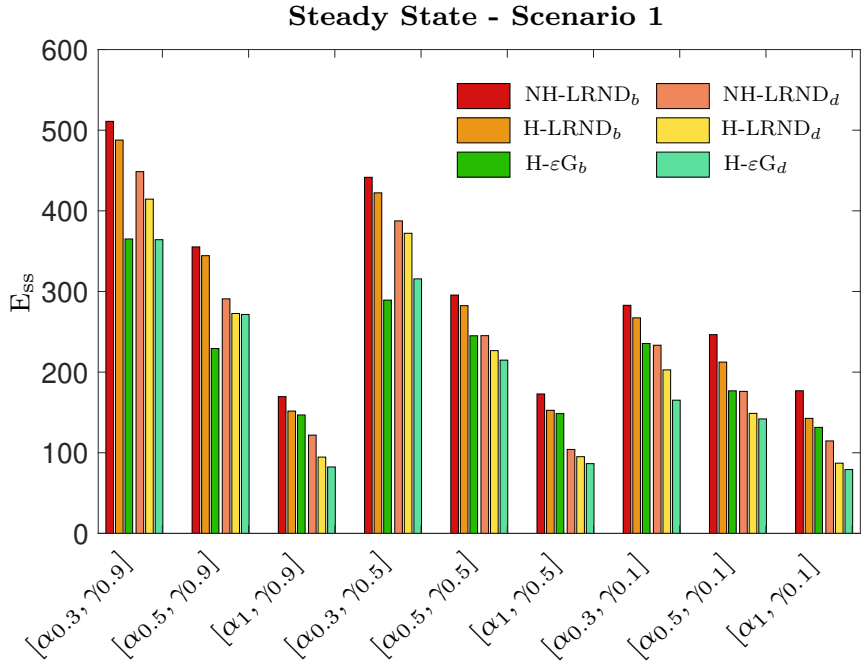

Fig. 5: Steady State Parametric Analysis: This bar graph is relative to the reaching of optimal solution varying α and γ for the Scenario 1.

joint	$\boldsymbol{a}$ (m)	$\boldsymbol{\alpha}$ (°)	$\boldsymbol{d}$ (m)	$\boldsymbol{\theta}$ (°)
1	0	90°	0.2755	θ_1
2	0	90°	0	θ_2
3	0	90°	−0.410	θ_3
4	0	90°	−0.0098	θ_4
5	0	90°	−0.3111	θ_5
6	0	90°	0	θ_6
7	0	0°	0.2638	θ_7

Table 5: DH parameter for the KINOVA Jaco^2.

Each one of the actions selected by the RL-Task Planner is related to the relocation of one of the objects in the scene. Regarding the motion planner in Sect. 4.1, the following task hierarchy is considered:

1. mechanical joint limits avoidance;
2. self-collision avoidance;
3. obstacle avoidance between the end-effector and the other objects;
4. position and orientation of the arm's end-effector.

Once the agent is trained, the robot starts relocating objects according to the found sequence, namely $\mathcal{S}_4^T = \{O_9, O_3, O_1, O_6, T\}$; a sequence of snap-

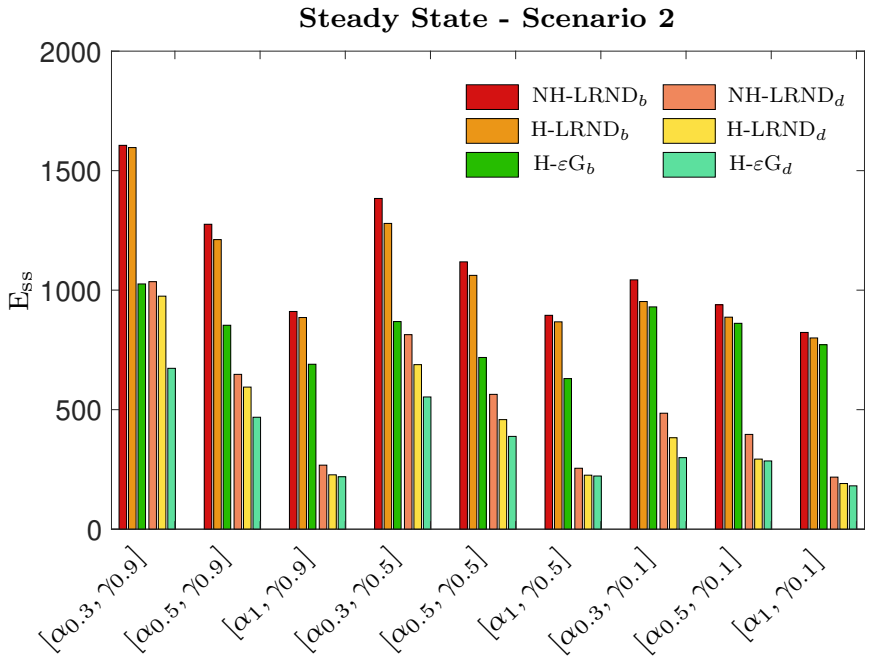

Fig. 6: Steady State Parametric Analysis: This bar graph is relative to the reaching of optimal solution varying α and γ for the Scenario 2.

joint	**Min** (rad)	**Max** (rad)
1	$-\infty$	∞
2	0.82	5.46
3	$-\infty$	∞
4	0.52	5.76
5	$-\infty$	∞
6	1.13	5.14
7	$-\infty$	∞

Table 6: Positions limits for the KINOVA Jaco[2]

shots relative to object relocation is reported in Fig. 14, while a video of the experiment can be found here[1].

6 Conclusions

This work considered the problem of retrieving a target from clutter through a robot manipulator. A two-layered architecture is devised to the scope made

[1] `http://www.youtube.com/watch?v=2aTqmWzmiJ8`

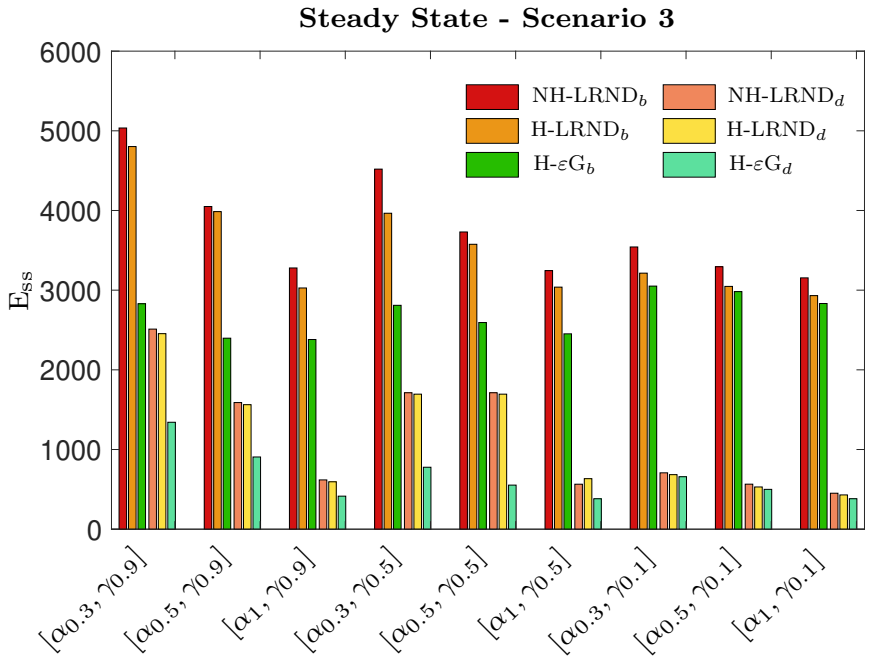

Fig. 7: Steady State Parametric Analysis: This bar graph is relative to the reaching of optimal solution varying α and γ for the Scenario 3.

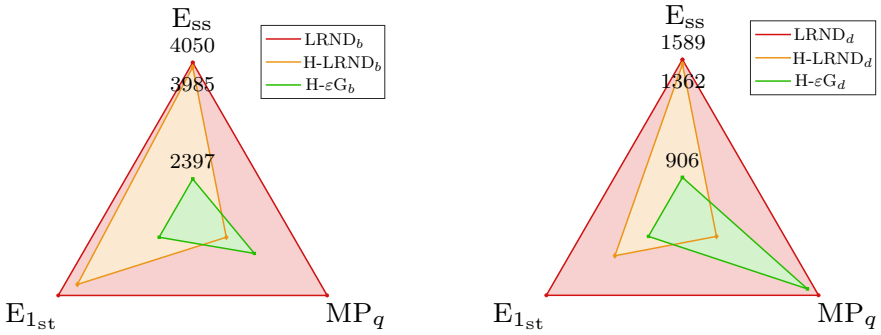

Fig. 8: Comparison between Breadth (left) and Depth (right) approaches, considering the proposed tree exploration strategies for the Scenario 3, with learning rate $\alpha = 0.5$ and discount factor $\gamma = 0.9$.

by a Task and Motion Planners. The former represents the high level and is responsible for learning the optimal sequence of objects to relocate, whereas the latter, at a lower level, is in charge of planning robot trajectories taking into account robot and environmental constraints. In detail, the Task planner makes use of a Q-learning algorithm on a dynamic tree structure, and three

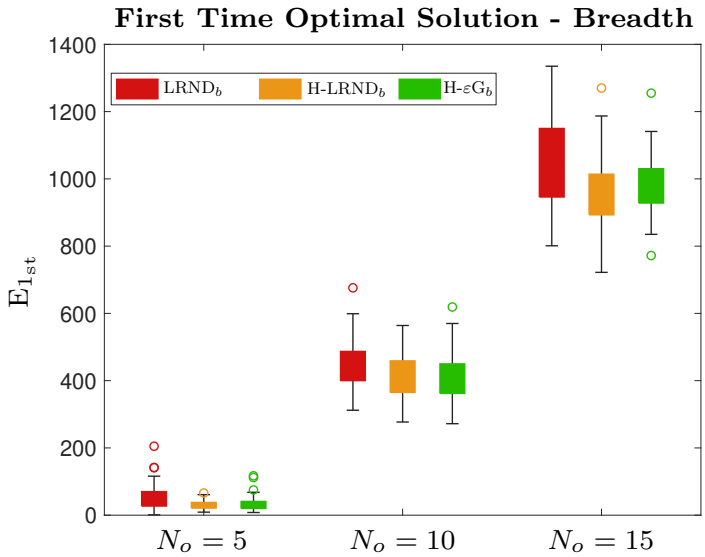

Fig. 9: Three learning policies comparison: Average, minimum and maximum episode relative at the first time that the agent reaches the target T through the optimal sequence considering Breadth search approach. The statistics are based on 50 training with $\alpha = 0.5$ and $\gamma = 0.9$.

joint	**Min** (rad/s)	**Max** (rad/s)
1	−0.81	0.81
2	−0.81	0.81
3	−0.81	0.81
4	−0.81	0.81
5	−1.11	1.11
6	−1.11	1.11
7	−1.11	1.11

Table 7: Velocities limits for the KINOVA Jaco[2]

different RL-policies with two tree exploration strategies are compared in scenarios with different complexity. The extensive simulation campaigns shows that the ε-greedy approach with Depth search and heuristics allows to obtain on average the best performance.
This approach was, then, applied on a real case study where bottles have to be relocated by a robot manipulator. As future work, DQN (*Deep Q-Network*) will be investigated to leverage their generalization capabilities together with a POMDP (*Partially Observable Markov Decision Process*) modeling to tackle the case of hidden objects.

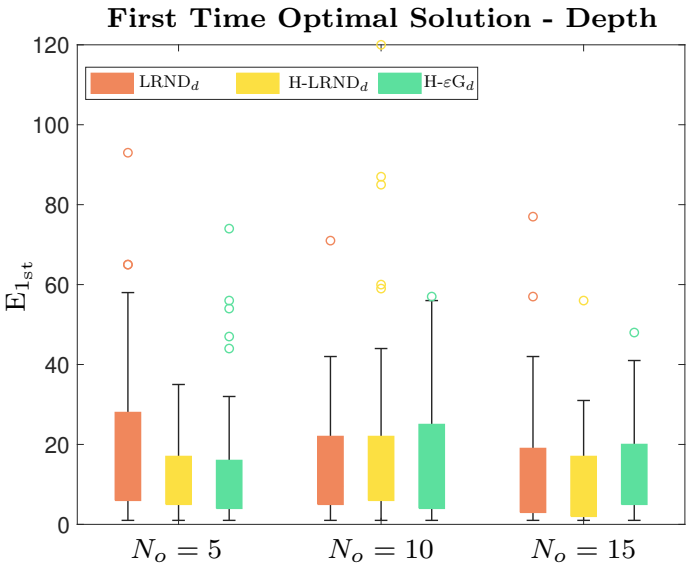

Fig. 10: Three learning policies comparison: Average, minimum and maximum episode relative at the first time that the agent reaches the target T through the optimal sequence considering Depth search approach. The statistics are based on 50 training with $\alpha = 0.5$ and $\gamma = 0.9$.

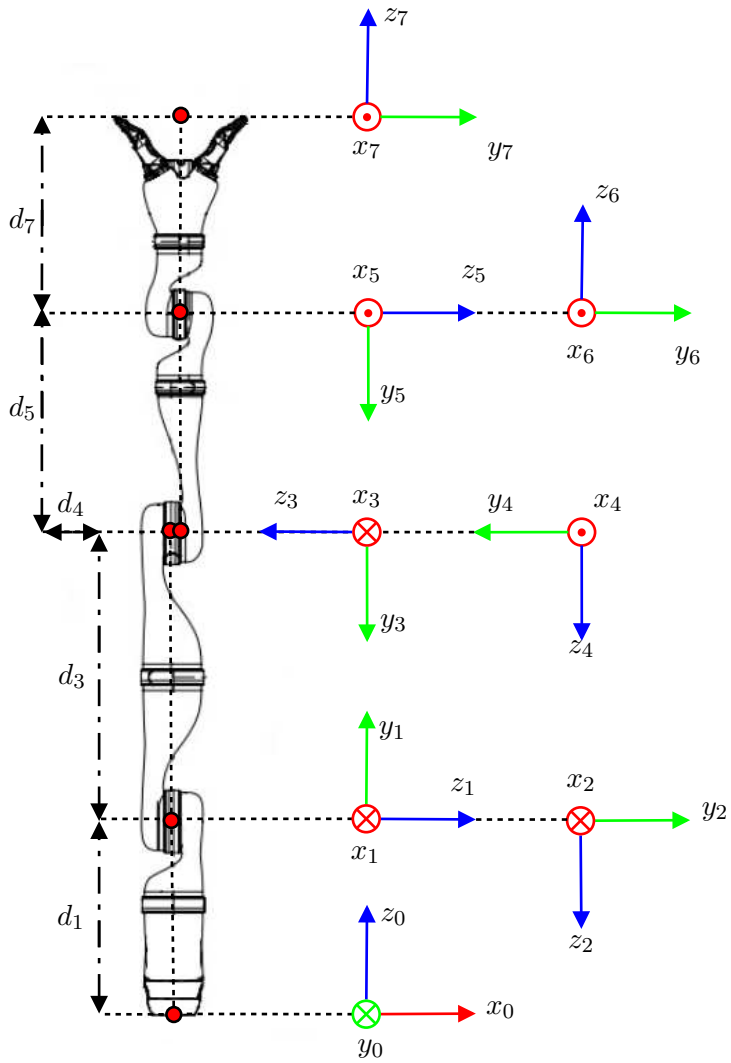

Fig. 11: KINOVA Jaco^2 with reference frames in Denavit-Hartenberg convention.

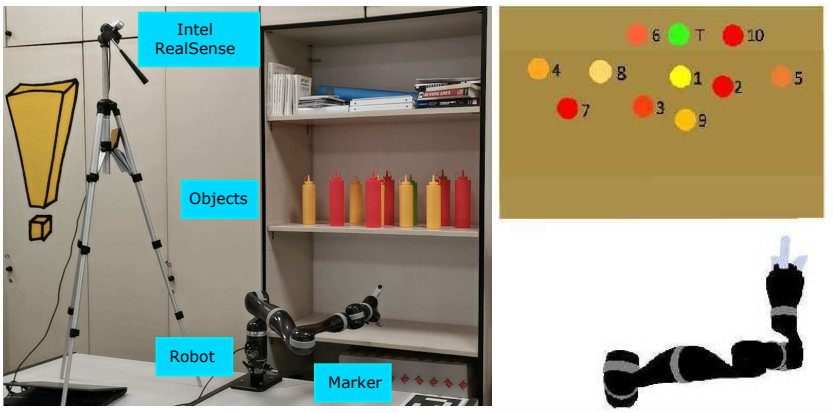

Fig. 12: Left. The robotic setup adopted to demonstrate the devised approach. Right. A top view representation of target (in green) and obstacles in their initial configuration.

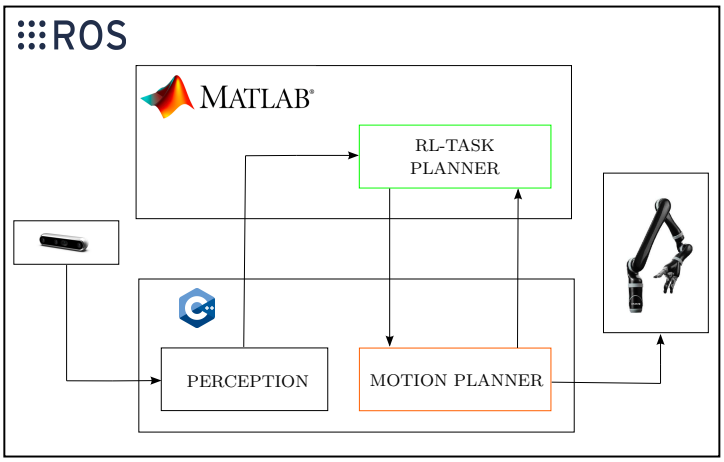

Fig. 13: Software-Hardware Architecture: The perception module receives the scene from the real camera and provides information on the objects pose to the RL-Task Planner (MATLAB). This latter selects an action according with its policy and sends the action to the Motion Planner (C++). Finally, if there is a free-obstacle path that satisfies all the joint constraints, the robot receives the computed joint velocities to perform the task.

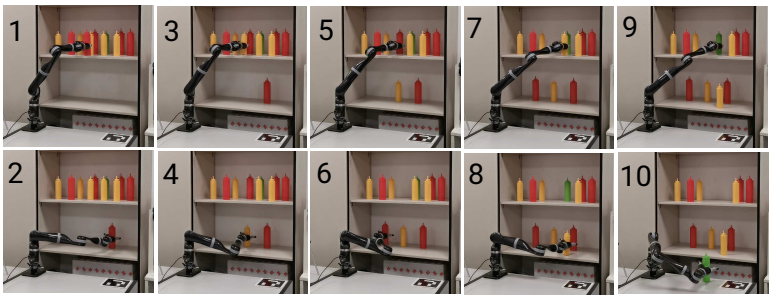

Fig. 14: Robot manipulator during the validation phase: The KINOVA Jaco^2 is moving the objects in the real world scenario.

Declarations

– **Funding**: The Authors declare that this work was supported by Dipartimento di Eccellenza granted to DIEI Department, University of Cassino and Southern Lazio, by H2020-ICT project CANOPIES (Grant Agreement N. 101016906) and by POR FSE LAZIO 2014-2020, Project DE G06374/2021.

– **Conflict of interest**: The Authors declare that they have no conflict of interest.

– **Code or data availability**: Not applicable

– **Authors' Contributions**: All Authors have contributed equally to the ideas, theories and analysis of results. The first draft of the manuscript was written by Giacomo Golluccio.
Paolo Di Lillo, Daniele Di Vito, Alessandro Marino and Gianluca Antonelli commented and revised this first version. All authors read and approved the final manuscript.

– **Ethical approval**: Not applicable.

– **Consent to participate**: Not applicable.

– **Consent for publication**: Not applicable.

References

1. Nikolaus Correll, Kostas E Bekris, Dmitry Berenson, Oliver Brock, Albert Causo, Kris Hauser, Kei Okada, Alberto Rodriguez, Joseph M Romano, and Peter R Wurman. Analysis and observations from the first amazon picking challenge. *IEEE Trans. on Automation Science and Engineering*, 15(1):172–188, 2016.
2. Federico Ceola, Elisa Tosello, Luca Tagliapietra, Giorgio Nicola, and Stefano Ghidoni. Robot task planning via deep reinforcement learning: a tabletop object sorting application. In *2019 IEEE Int. Conf. on Systems, Man and Cybernetics (SMC)*, pages 486–492. IEEE, 2019.
3. Changjoo Nam, Jinhwi Lee, Sang Hun Cheong, Brian Y Cho, and ChangHwan Kim. Fast and resilient manipulation planning for target retrieval in clutter. In *2020 IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 3777–3783. IEEE, 2020.

4. Jinhwi Lee, Younggil Cho, Changjoo Nam, Jonghyeon Park, and Changhwan Kim. Efficient obstacle rearrangement for object manipulation tasks in cluttered environments. In *2019 Int. Conf. on Robotics and Automation (ICRA)*, pages 183–189. IEEE, 2019.
5. Mike Stilman and James Kuffner. Planning among movable obstacles with artificial constraints. *The Int. Journ. of Robotics Research*, 27(11-12):1295–1307, 2008.
6. Kaiyu Hang, Johannes A Stork, Florian T Pokorny, and Danica Kragic. Combinatorial optimization for hierarchical contact-level grasping. In *2014 IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 381–388. IEEE, 2014.
7. Mike Stilman, Jan-Ullrich Schamburek, James Kuffner, and Tamim Asfour. Manipulation planning among movable obstacles. In *Proceedings 2007 IEEE Int. Conf. on Robotics and Automation*, pages 3327–3332. IEEE, 2007.
8. Weihao Yuan, Kaiyu Hang, Danica Kragic, Michael Y Wang, and Johannes A Stork. End-to-end nonprehensile rearrangement with deep reinforcement learning and simulation-to-reality transfer. *Robotics and Autonomous Systems*, 119, 2019.
9. Joshua A Haustein, Jennifer King, Siddhartha S Srinivasa, and Tamim Asfour. Kinodynamic randomized rearrangement planning via dynamic transitions between statically stable states. In *2015 IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 3075–3082. IEEE, 2015.
10. Neil T Dantam, Zachary K Kingston, Swarat Chaudhuri, and Lydia E Kavraki. Incremental task and motion planning: A constraint-based approach. In *Robotics: Science and systems*, volume 12, page 00052. Ann Arbor, MI, USA, 2016.
11. Giray Havur, Guchan Ozbilgin, Esra Erdem, and Volkan Patoglu. Geometric rearrangement of multiple movable objects on cluttered surfaces: A hybrid reasoning approach. In *2014 IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 445–452. IEEE, 2014.
12. Hossein Karami, Antony Thomas, and Fulvio Mastrogiovanni. A task-motion planning framework using iteratively deepened and/or graph networks. *arXiv preprint arXiv:2104.01549*, 2021.
13. Manfred Eppe, Phuong DH Nguyen, and Stefan Wermter. From semantics to execution: Integrating action planning with reinforcement learning for robotic causal problem-solving. *Frontiers in Robotics and AI*, 6:123, 2019.
14. Blai Bonet and Héctor Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1-2):5–33, 2001.
15. Ahmed H Qureshi, Arsalan Mousavian, Chris Paxton, Michael C Yip, and Dieter Fox. Nerp: Neural rearrangement planning for unknown objects. *arXiv preprint arXiv:2106.01352*, 2021.
16. Marwan Qaid Mohammed, Kwek Lee Chung, and Chua Shing Chyi. Review of deep reinforcement learning-based object grasping: Techniques, open challenges and recommendations. *IEEE Access*, 2020.
17. Petar Kormushev, Sylvain Calinon, and Darwin G Caldwell. Reinforcement learning in robotics: Applications and real-world challenges. *Robotics*, 2(3):122–148, 2013.
18. Wissam Bejjani, Wisdom C Agboh, Mehmet R Dogar, and Matteo Leonetti. Occlusion-aware search for object retrieval in clutter. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4678–4685. IEEE, 2021.
19. Yuhong Deng, Xiaofeng Guo, Yixuan Wei, Kai Lu, Bin Fang, Di Guo, Huaping Liu, and Fuchun Sun. Deep reinforcement learning for robotic pushing and picking in cluttered environment. In *2019 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 619–626. IEEE.
20. Bohan Wu, Iretiayo Akinola, and Peter K Allen. Pixel-attentive policy gradient for multi-fingered grasping in cluttered scenes. In *2019 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 1789–1796. IEEE, 2019.
21. Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.
22. Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.
23. Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods

for deep reinforcement learning. In *Int. Conf. on Machine Learning*, pages 1928–1937. PMLR, 2016.
24. Giacomo Golluccio, Daniele Di Vito, Alessandro Marino, Alessandro Bria, and Gianluca Antonelli. Task-motion planning via tree-based q-learning approach for robotic object displacement in cluttered spaces. In *Proceedings of the 18th Int. Conf. on Informatics in Control, Automation and Robotics - ICINCO*, pages 130–137. INSTICC, SciTePress, 2021.
25. Giacomo Golluccio, Daniele Di Vito, Alessandro Marino, and Gianluca Antonelli. Robotic weight-based object relocation in clutter via tree-based q-learning approach using breadth and depth search techniques. In *2021 20th Int. Conf. on Advanced Robotics (ICAR)*, pages 676–681. IEEE, 2021.
26. Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3-4):279–292, 1992.
27. Daniele Di Vito, Mathieux Bergeron, David Meger, Gregory Dudek, and Gianluca Antonelli. 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. IEEE.
28. James J. Kuffner and Steven M. LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Proceedings 2000 ICRA. Millennium Conference. IEEE Int. Conf. on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, volume 2, pages 995–1001. IEEE, 2000.
29. Stefano Chiaverini. Singularity-robust task-priority redundancy resolution for real-time kinematic control of robot manipulators. *IEEE Trans. on Robotics and Automation*, 13(3):398–410, 1997.
30. Bruno Siciliano and J.-J. E. Slotine. A general framework for managing multiple tasks in highly redundant robotic systems. In *Proc. Fifth Int. Conf. on Advanced Robotics (ICAR)*, pages 1211–1216, Pisa, Italy, 1991. IEEE.
31. Paolo Di Lillo, Filippo Arrichiello, Daniele Di Vito, and Gianluca Antonelli. BCI-controlled assistive manipulator: developed architecture and experimental results. *IEEE Trans. on Cognitive and Developmental Systems*, pages 1–1, 2020.
32. Paolo Di Lillo, Enrico Simetti, Francesco Wanderlingh, Giuseppe. Casalino, and Gianluca. Antonelli. 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, 2021.
33. Sergio Garrido-Jurado, Rafael Muñoz-Salinas, Francisco José Madrid-Cuevas, and Manuel Jesús Marín-Jiménez. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6):2280–2292, 2014.
34. Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition*, pages 779–788, 2016.