Combining Motion Planning and Task Assignment for a Dual-arm System

Carlos Rodríguez and Raúl Suárez

Abstract— **This paper deals with the problem of combining motion and task assignment for a dual-arm robotic system. Each arm of the system performs independent tasks in a cluttered environment. Robot actions are determined to remove potential obstacles and obtain collision-free paths to grasp the target objects. The approach uses the information provided by the motion planner to build a graph structure in order to represent the obstacles to be removed. The graph is used, first, to decide which is the next motion path to be computed, and, second, to assign the tasks to each arm of the robotic system. The approach has been implemented for a dual-arm robotic system and real experiments are presented in the paper.**

I. INTRODUCTION

Object manipulation in cluttered environment is a task that humans perform daily. Different types of problems are found when humans are replaced with robots that have to perform this type of tasks, like the motion planning in order to move the robot from an initial to the goal configuration, the computation of the object grasping, and the computation of a path to move the object from its initial configuration to a new goal configuration. This kind of problems are considered classics in the literature in robotics.

On the other hand, there are higher level problems to be solved, like planning for task distribution or the precedence relationship between tasks. In the case of two or more robots that execute tasks in a shared environment, planning the task assignation/distribution to each robot is a relevant problem, i.e. defining which robot is in charge of each task, and how the tasks are prioritized in order to avoid collisions of the robots with the obstacles of the environment as well as collisions between the robots.

In this context, this work deals with the motion and task assignment problem for a dual-arm robot in cluttered environments. Each arm is used in an independent but coordinate way to grasp and manipulate target objects, and there are other objects in the environment acting as potential obstacles. The two arms can be used to remove these obstacles if it is necessary. The target objects can be pre-assigned to each robot or not, depending on the task to be performed.

The proposed approach uses a motion planner to find the robot movements to grasp and manipulate the target objects, and, at the same time, determines which are the obstacles that must be removed. With the information from the motion planning a graph structure is build, which allows to easily represent the tasks that the robots should execute.The graph allows the distribution and the prioritization of the tasks between the arms by using a backtracking search algorithm.

After this introduction the paper is organized as follows. Section II presents a review of related works, Section III presents the proposed approach, and Section IV presents the manipulation task graph. Then, Section V presents some application examples and, finally, Section VI summarizes the work and presents some topics deserving future work.

II. RELATED WORKS

The use of a dual-arm robotic system is becoming quite common to perform manipulation tasks, either involving manipulation of a single object using both arms [1], [2], or manipulation of multiple objects using each arm in an independent way [3], [4].

Motion planning for a dual-arm system or multiple robots with large number of degrees of freedom in a shared environment implies complex problems in high dimensional spaces. The approaches to deal with these problems can be classified into centralized and decoupled [5]. The approach proposed in this work belongs to the decoupled type. For each arm an independent motion plan is computed, and then, the motion plans are coordinated following the approach proposed by [6], in order to avoid collisions between the arms during the execution.

Combining motion and task assignment (hybrid planning) is an open problem in robotics [7], [8], [9], in the literature there are several relevant algorithms deal with hybrid planning, one of these works employs a high-level planner that acts as constraint and provides a heuristic cost function in the search algorithm in order to speed up the motion planner [10]. Furthermore, a grid-based discrete representation can be also used to combine the information from the task planner and the information from the motion planner in order to obtain a continuous free-collision trajectory for the robot [11]. Another way to combine the task and motion planners is formulated as a representational abstraction between them, where, each action in the task planner can have a multiple instantiations in the motion planner, this increases the possibility of finding a feasible trajectory for the robot [12].

The approach proposed in this work is different to the works mentioned above that are focused on high-level task planners (semantic and symbolic representation) and on how to combine them with the low level planners (motion planner) to solve task-oriented problems. Instead, the proposed approach uses the motion planner to obtain motion

The authors are with the Institute of Industrial and Control Engineering (IOC) - Universitat Politècnica de Catalunya (UPC), Barcelona, Spain (carlos.rodriguez.p, raul.suarez@upc.edu).

This work was partially supported by the Spanish Government through the projects DPI2013-40882-P and DPI2014-57757-R.

plans that have the information necessary to build a graph representation of the tasks that the robot must execute, as will be explained later in Section IV.

Our work is closely related with the works presented in [13], [14], which propose backtracking search algorithms in the real space in order to know which movable objects in the workspace block the access to a given goal; a related algorithm was also proposed in [15], using a high-level regression-based symbolic planner. Furthermore, Krontiris and Bekris [16] try to avoid the backtracking search by using *fast monotone rearrangement solve* algorithm.

Stilman et al. [13] solves the problem of rearrange a clutter workspace using a robot in a 2D workspace, and then, they have extended the work to a 3D workspace using a manipulator with high number of degree of freedom [17]. In order to rearrange the workspace and reach the target object, they employ swept volumes to determine, recursively, which objects must be moved. On the other hand, Dogar et. al [14] presents an approach focused on determining free volume space where to put the obstacles that directly block the access to the target object, and includes a non-prehensile manipulation such as pushing in order to move the obstacles that cannot be grasped (pick and place manipulation).

In contrast to [14] and [17], the approach proposed here uses a dual-arm system, and each robot arm may be able to solve the same tasks of removing obstacles and grasp the target objects. Combining the solutions of each arm, it is possible to obtain a better task execution, where the arms work in a independent but cooperative way to achieve a common goal. The proposed approach uses a backtracking search algorithm to explore the graph and obtain the sequence of actions to remove the minimum number of obstacles, and then, distributes the tasks between the robot arms. In contrast, [18] provides algorithms for removing the minimal number of obstacles in order to reach a position, but it does not do recursive removal like in this work.

III. PROPOSED APPROACH

Consider a dual-arm robotic system in a cluttered environment with fixed and removable objects. The goal of this work is to grasp two target objects with the dual-arm system. The objects can be grasped by any arm or can be assigned in advance, depending on the task to be performed. Due to the cluttered environment, the objects of interest may be blocked by other removable objects, which therefore must be removed.

A. Background of the proposed approach

The proposed approach follows the work in [3], which presents a variation of a probabilistic roadmaps planner to compute the robot paths and uses a *precedence graph* to represent the tasks to be executed by each robot arm. The nodes of the *precedence graph* represent the target objects and the removable objects O_j, $j = 1, ..., n$, and the arcs represent each robot R_i, $i = 1, 2$. The motion planner is used to compute a path from the initial configuration of R_i to the grasp configuration of O_j. The key point of the motion planner is that the samples generated in the configuration space that imply collisions with removable objects O_k $k = 0, ..., n$ are not rejected like in a classic motion planner; instead, these samples are added to the roadmap as valid samples, and the set of obstacles $SO_{i,j}$ contains the removable objects O_k that generate collisions is stored. At the end, a robot motion path $P_{i,j}$ is generated knowing the involved set of obstacles $SO_{i,j}$. The motion planner is used in a recursive way for each $O_k \in SO_{i,j}$ following a minimum cost strategy until finding a collision-free path $P_{i,j}$ that allows removing the corresponding O_j.

B. Contributions and limitations

The main contribution of this work is a framework to use two robot arms to perform independent manipulation tasks in a cluttered workspace considering the removal of obstacles if it is necessary. The framework uses a *manipulation task graph* that is based on a *precedence graph*. The graph contains the possible actions that the robot arms can execute, according to the information obtained from the motion planner. A backtracking search algorithm is used to explore the graph and determine the sequence of actions that the robot arms must execute to reach the target objects, having to remove the minimum number of removable obstacles.

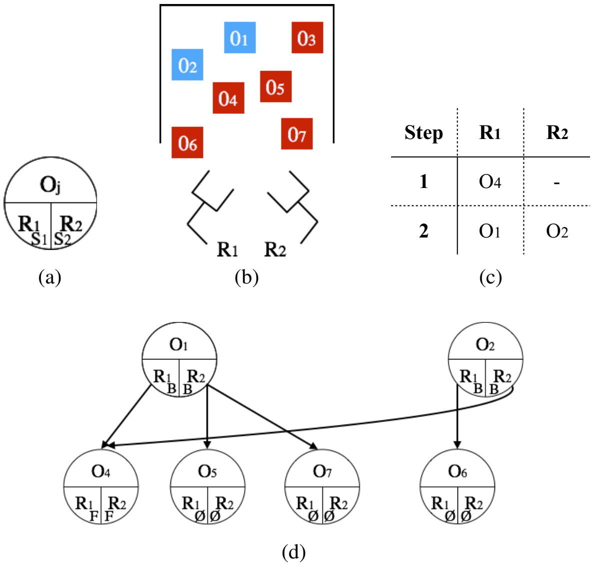
A potential limitation of the approach is that the workspace must be known in advance, including the distinction between fixed and removable obstacles. Beside, in the current implementation, a finite set of grasping configurations for each removable object (and the used hand) are given with the object model, then, there is no guarantee that the motion planner can find a solution to manipulate a given object whenever one actually exists using a non-included grasp.

IV. THE MANIPULATION TASK GRAPH

A. Description of the manipulation task graph

The *manipulation task graph G* is based on an AND/OR graph [19], where the obstacles O_k ($k = 1, ..., n$) of a set of obstacles $SO_{i,j}$ found on the motion plan $P_{i,j}$ for each robot arm R_i ($i = 1, 2$) are represented. Each node represents an object O_j ($j = 1, ..., n$) and each arc from that node represents the robot R_i used to manipulate O_j. The root nodes represent the target objects, and the descendant nodes represent the set of obstacles $SO_{i,j}$ of the parent nodes. Figure 1(a) shows the representation of a node of the *manipulation task graph G*. Each node is defined with the following properties: An *id* of the node name that indicates the represented object O_j; A *root id* that represents which is the corresponding root node of O_j; A *root cost* for each root node representing the number of nodes to be removed from O_j to reach a root node; A *local cost* for each R_i, representing the number of siblings nodes of O_j; A *state* represented by a flag for each R_i associated to O_j that indicates:

- Empty ($\emptyset$), when a path $P_{i,j}$ for R_i to grasp O_j was not computed yet.
- Null (N), when R_i cannot grasp O_j (i.e. O_j is not kinematically reachable by R_i).



Step	R1	R2
1	O4	-
2	O1	O2

Fig. 1: a) Node representing object O_j and the states of the accessibility to O_j with each robot R_i, b) Example of a workspace in $\Re^2$, c) Resulting sequence of actions to reach O_1 and O_2, d) Resulting *manipulation task graph G*.

- Free (F), when exists $P_{i,j}$ for R_i to grasp O_j and it has no obstacles (i.e. $SO_{i,j} = \emptyset$).
- Blocked (B), when exist $P_{i,j}$ for R_i to grasp O_j and has obstacles (i.e. $SO_{i,j} \neq \emptyset$).

Henceforth, the *robot states* are represented as a pair, i.e. $\langle$state of R_1, state of $R_2\rangle$.

B. Working with the manipulation task graph

The *manipulation task graph* G is built using the information obtained by the motion planner. First, the target objects are added as root nodes of G, then, a node is chosen as O_j (node to be explored) and a motion path $P_{i,j}$ is computed for each R_i. Then, each $O_k \in SO_{i,j}$ obtained from $P_{i,j}$ is added to G as descendant nodes of O_j. Then, the leaf node O_j with the minimum number of obstacles until the target object is chosen, and the motion plan to remove O_j is computed. The process is repeated and G is expanded from the leaf nodes until finding a path to remove each O_j from the target object plan. When there is a path $P_{i,j}$ to remove each obstacle, a backtracking search in G is done to obtain the sequence of action that the robot arms must execute.

Figure 1(b) shows an example in $\Re^2$, a cluttered environment is composed of removable square objects O_j in front of the dual-arm system represented by the robot arms R_1 and R_2. The blue squares represent the target objects O_1 and O_2 and the red squares the obstacles.

Figure 1(d) shows the resulting G to grasp O_1 and O_2 with each R_i: to grasp O_1 with R_1 resulted $SO_{1,1} = \{O_4\}$ and for R_2 $SO_{2,1} = \{O_5, O_7\}$. The branch of the manipulation task graph with the lower number of obstacles is chosen, in this case the branch that contains O_4. In order to remove O_4 a collision-free paths were found for both robot arms (i.e. $SO_{1,4} = \{\emptyset\}$, $SO_{2,4} = \{\emptyset\}$. On the other hand, to grasp O_2 with R_1 resulted $SO_{1,2} = \{O_6\}$ and for R_2 $SO_{2,2} = \{O_4\}$, since O_4 has a collision-free path this branch is chosen. Figure 1(c) shows the resulting sequence of actions.

Algorithm 1: Main

```
input : $W$, $SG$
output: $TS$
$SA, TS \leftarrow \emptyset$
create $G$ with the target objects $SG$ as roots
select a $target$ from $SG$
while $target \neq \emptyset$ do
    select an unexplored node $O_{aux}$ with root = $target$
    if $O_{aux} \neq \emptyset$ then
        for each $R_j$ do
            find the motion plan $P_{i,j}$ to grasp $O_{aux}$
            if $P_{i,j}$ has $SO_{i,j} \neq \emptyset$ then
                for each obstacle $O_k$ in $SO_{i,j}$ do
                    if $O_k$ is already in $G$ then
                        if $O_k$ not generates a loop then
                            update the properties of $O_k$
                    else
                        add $O_k$ as child of $O_{aux}$ in $G$
            update the properties of $O_{aux}$
        if $O_{aux}$ has at least one state = Free then
            find solution from $O_{aux}$ to $target$
            if there is a solution then
                add the solution to the set of actions $SA$
                select a new $target$ from $SG$
        if $O_{aux}$ has state = Null for each $R_j$ then
            update the properties of the parent nodes
    else
        error: not solution for $target$
        break
if solution for each target in $SG$ then
    $TS \leftarrow$ taskDistribution($G, SA$)
return $TS$
```

C. Implementation

Algorithm 1 shows the main procedure of the proposed approach. The algorithm receives as input the model of the workspace W, which includes the 3D models of the dual-arm system, the fixed and removable objects, as well as their corresponding location in the workspace, the set of target objects SG, and the grasping configurations of the removable objects, and returns the task sequence TS that the dual-arm system must execute. The algorithm is divided in two parts, building and exploring the *manipulation task graph G*. These two parts are executed sequentially until a set of robot actions SA is found to grasp the target objects.

G is created with each element of SG as a root node that represents a *target* object. Next, a leaf node O_j is chosen to be explored, which in the first iteration is obviously one of the root nodes. It is necessary to know which node O_j will be explored in order to find a solution for a given *target*. Only the leaf nodes that fulfill with the *robot states* = $\langle \emptyset, \emptyset \rangle$ can be selected, this means that the node has not been explored. Then, the terminal O_j that complies the previous condition and has the minimum *root cost* is selected. The *root cost* is given by the number of obstacles that should be removed from the obtained path to grasp a target object.

Then, a motion path is computed for each R_i in order to grasp and manipulate the corresponding O_j. The motion planner returns the path $P_{i,j}$ and also returns the set of removable obstacles $SO_{i,j}$ found along $P_{i,j}$.

In order to increase the probability of finding feasible paths with the motion planner, it is considered that the objects can be grasped in different ways. A set of grasp configurations is used to compute different motion paths, and the one with the minimum number of elements in $SO_{i,j}$ is chosen. The grasping configurations for each object can be obtained using different procedures (see for instance [20], [21]). Here we assume that the set of grasping configurations of an object for a given hand has been computed in advance and it is provided with the model of the object. Then, the grasping configurations of the hand are related to the object reference frame, but since the position of the object in the work environment is known, the grasping configurations can easily be mapped to the robot configuration space.

Then, after O_j is analyzed, the corresponding $SO_{i,j}$ is used to expand G. An empty $SO_{i,j}$ means that there is a free collision path to grasp and move O_j, otherwise the obstacles in $SO_{i,j}$ should be added to G as children of O_j. If $SO_{i,j}$ has only one element it generates an OR arc, while if $SO_{i,j}$ has more than one element they generate AND arcs.

Before adding an obstacle to G it is verified whether it has already been previously added, because it can produce loops in G. A loop means there is a collision between O_k and one of its predecessor nodes (e.g. father, grandfather or any other predecessor) and vice versa. The properties of O_k must be updated with a null state, i.e. O_k cannot be grasped by R_i. In the case that any R_i cannot grasp O_k, it is checked which nodes of the same branch are affected and they states are updated as null. On the other hand, if adding O_k does not generate loops the properties of O_k and O_j must be properly updated (e.g. the robot states of O_j change from unexplored to blocked if $SO_{i,j}$ is not empty).

When a path $P_{i,j}$ with $SO_{i,j} = \emptyset$ is found to remove O_j and its siblings (i.e a collision-free path), this means that a sequence of actions exists to remove O_j and the predecessors nodes that are blocking the path to a given target object.

The sequence of actions is computed using a backtracking search method, from O_j to the *target* object. The search in the graph is done as follows. Check if O_j has at least one collision-free path, then, check if it has sibling nodes for the same R_i and the same parent node. Recursively, check if each sibling node has a collision-free path, and if its child nodes (if there exist) have collision-free paths. Then, choose the parent of O_j and continue the exploration recursively until arriving to the given *target* object, which means that there exists a sequence of actions to remove all the obstacles to allow the grasp of the *target* object. Then, the sequence of action is added to the set of sequence of actions SA and a new *target* is selected.

When a sequence of action has been determined for each *target* object the algorithm computes the taskDistribution function. The input to the taskDistribution function is G and SA. The function distributes the actions of SA as pairs, i.e. $\langle$ task$_i$ for R_1, task$_j$ for R_2 $\rangle$ (in order to parallelize the actions to be executed by each robot arm) generating a set of tasks sequence TS. The tasks that correspond to the removal of the objects represented by leaf nodes of G that are included in SA are selected and, then, from these tasks, those that unlock a larger number of nodes in G are selected. This process is executed recursively until all the removal tasks in SA are assigned. In some cases a pair may has an empty element, this means that one of the arms can not perform any task in parallel with the other arm.

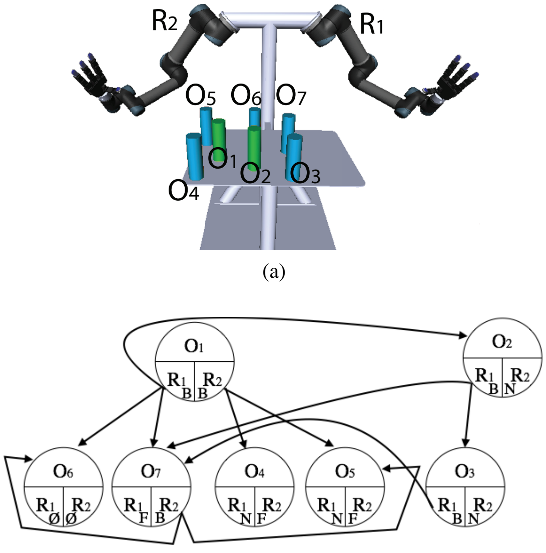
The output of the Algorithm 1 is TS. If the algorithm returns empty TS, it is because it does not exist a sequence of action for one of the *targets* (i.e. when the *target* is not reachable by any of the robot arms, or, when the obstacles block the access to the *target*.

V. EXPERIMENTAL RESULTS

The proposed approach has been implemented using a dual-arm robotic system in a simulated and real workspace. The dual-arm system is composed of two 6 DOF Universal Robots (UR5), each one is equipped with an Allegro Hand (AH) with 16 DOF. The simulated workspace includes the corresponding 3D models of the two robots and the two hands, and a data base with different types of objects.

The motion planning and graphical simulations have been performed using *The Kautham Project* software tool [22], which is a home-developed open source environment that provides several useful tools for the development of planners, like, for instance, random and deterministic sampling methods, metrics to evaluate the performance of the planners (number of generated samples, collision check callings, number of nodes in the graph solution, connected components) and simulation tools (including direct and inverse kinematic models of the robots) that use *Coin3D* for the graphical rendering, *PQP* for the collision detection, *ROS* for the communication layer, and more recently *OMPL* has been integrated providing the motion planning tool with more options. The experiments were run in a computer with a 2.3 GHz Intel Core i7 processor with 8 Gb RAM, Ubuntu OS 14.04 and ROS Hydro.

In the experiments presented below, several cylinders lie on a table in front of the dual-arm robot. The target objects are the green cylinders O_1 and O_2 and there are a set of removable objects (blue cylinders $O_j, j \geq 3$) that may act as obstacles that block the direct access to the target objects. The goal is that each robot arm grasps a green cylinder.



(a)

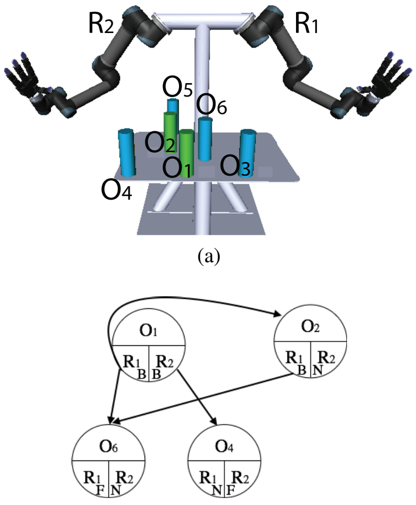
(b)

Fig. 2: Experiment 1, (a) Setup of the workspace, (b) The resulting manipulation task graph.

As mentioned above, a set of grasp configurations of the objects were computed in advance. In this case, the grasp configurations are such that the fingers wrap around the axis of the objects (grasping the objects from the top is not allowed). The set of graps contains 35 different configurations. The inverse kinematics of the robot arms is computed for each configuration of the set of grasps in order to know whether they are actually reachable. Then, the reachable configurations are used as goal configurations for the motion planner (i.e. the motion planner computes a motion path for each goal configuration). The paths with the minimum number of obstacles are selected. Then, a new set of motion paths are computed to remove the obstacles from the paths toward the target objects (the obstacles are thrown out of the workspace instead of being relocated).

Figure 2(a) and Figure 3(a) show two setups of the workspace used for the experiments. The resulting manipulation task graphs are shown in Figure 2(b) and Figure 3(b).

In the first experiment, the result was as follows. In order to grasp O_1 with R_1 the computed motion path has collisions with O_2, O_6 and O_7 , while to grasp O_1 with R_2 the objects O_4 and O_5 have to be removed. Since the paths obtained for both robots are not collision free, the branch of the manipulation task graph with the lower number of obstacles is chosen, in this case the branch that contains O_4 and O_5. After looking for the motion paths to grasp O_4 and O_5 it was found a collision-free path for R_2 while R_1 cannot reach O_4 nor O_5. On the other hand, to grasp the other target object, O_2, a path that collides with O_3 and O_7, was found for R_1 and a path that collide with O_1 was found for R_2, this generates a loop in G, i.e. O_1 has to be removed to grasp O_2 and vice versa. The path found for R_1 to remove O_3 has collision with O_7 while R_2 cannot reach O_3 due to a loop in G. Finally, in order to grasp O_7 a collision-free path was found with R_2.



(a)

(b)

Fig. 3: Experiment 2, (a) Setup of the workspace, (b) The resulting manipulation task graph.

TABLE I: Number of paths computed by the motion planner, total computation time and average time.

		# Paths	Total time (s)	Average time (s)
Experiment 1	R_1	29	2.785142	0.096039
	R_2	18	3.407077	0.189282
Experiment 2	R_1	17	3.742668	0.233916
	R_2	16	2.028555	0.135234

Figure 4(a) shows the resulting task sequence TS describing which object is assigned to each robot in each step of the work, and Figures 4(b-d) show snapshots of the real execution of experiment 1.

In the second experiment, the result was as follows. In order to grasp O_1 a motion path colliding with O_2 and O_6 was found for R_1, and a motion path colliding with O_4 was found for R_2. Then, a collision-free path was found to remove O_4 with R_2. On the other hand, in order to grasp O_2 with R_1 a path that collides with O_6 was found, and in order to remove O_6 a collision-free path was found for R_1 while R_2 cannot reach O_6 because the path has a collision with O_2, generating a loop in G.

Figure 5(a) shows the resulting task sequence TS, and Figures 5(b-d) show snapshots of the real execution of experiment 2.

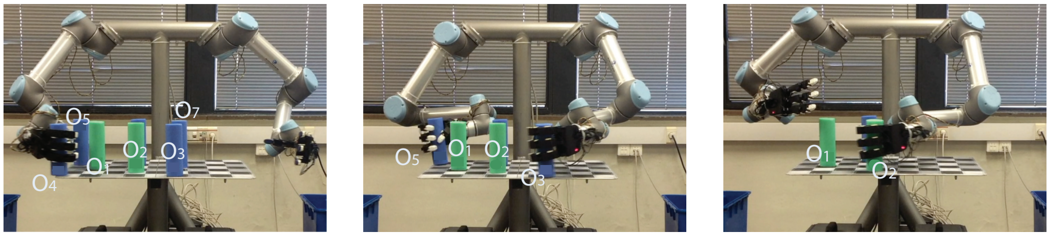
Table I shows the total number of paths computed for each experiment using a *RRT-Connect* planner, the total time used to compute all the paths and the average time per path.

VI. CONCLUSIONS AND FUTURE WORKS

This work has presented an approach that combines a motion planner and a task planner for a dual-arm system. One of the advantages of the proposed approach is that it can work in workspaces where there are removable obstacles. The approach generates motion paths for the robot arms (even though the target objects may be blocked by removable obstacles) and the set of motion paths to remove the obstacles. On the other hand, the approach includes a task planner which is in charge of determining the sequence of actions to

Step	R1	R2
1	O7	O4
2	O3	O5
3	O2	O1

(a)

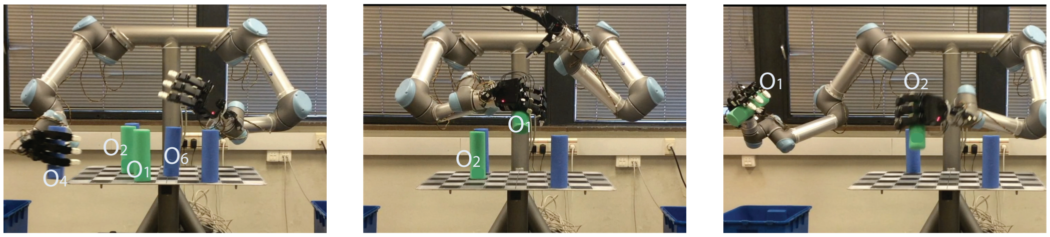


(b) (c) (d)

Fig. 4: Execution of the real experiment 1, a) The task sequence TS, b) R_1 going to grasp O_7 and R_2 moving O_4, c) R_1 grasping O_3 and R_2 grasping O_5, d) R_1 grasping O_2 and R_2 waiting to grasp O_1.

Step	R1	R2
1	O6	O4
2	O2	O1

(a)



(b) (c) (d)

Fig. 5: Execution of the real experiment 2, a) The task sequence TS, b) R_1 going to grasp O_6 and R_2 grasping O_4, b) R_2 moving O_1 and R_1 waiting to grasp O_2, c) R_1 moving O_2 and R_2 in the final configuration.

be performed by the robot arms to achieve their goals. The proposed approach has been implemented and successfully applied in a simulated and a real environment.

As future work, it is considered the transfer of the objects between the robot arms. Besides, it is considered the inclusion of a grasp planner to determine, when necessary, the grasp configurations on line during the manipulation planning.

References

[1] C. Smith, Y. Karayiannidis, L. Nalpantidis, X. Gratal, P. Qi, D. Dimarogonas, and D. Kragic, "Dual arm manipulation - A survey," *Robotics and Autonomous Systems*, vol. 60, no. 10, pp. 1340–1353, 2012.

[2] N. Vahrenkamp, T. Asfour, and R. Dillmann, "Simultaneous Grasp and Motion Planning," *IEEE Robotics and Automation Magazine*, vol. 19, pp. 43–57, 2012.

[3] C. Rodríguez, A. Montaño, and R. Suárez, "Planning manipulation movements of a dual-arm system considering obstacle removing," *Robotics and Autonomous Systems*, vol. 62, no. 12, pp. 1816 – 1826, 2014.

[4] R. Suarez, J. Rosell, and N. Garcia, "Using synergies in dual-arm manipulation tasks," in *Proc. IEEE Int. Conf. Robotics and Automation*, May 2015, pp. 5655–5661.

[5] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.

[6] A. Montaño and R. Suárez, "Coordination of several robots based on temporal synchronization," *Robotics and Computer-Integrated Manufacturing*, vol. 42, pp. 73 – 85, 2016.

[7] K. Hauser and J. C. Latombe, "Integrating task and PRM motion planning: Dealing with many infeasible motion planning queries," in *ICAPS Workshop on Bridging the Gap between Task and Motion Planning*, 2009.

[8] L. Karlsson, J. Bidot, F. Lagriffoul, A. Saffiotti, U. Hillenbrand, and F. Schmidt, "Combining task and path planning for a humanoid two-arm robotic system," in *Combining Task and Motion Planning for Real-World Applications (ICAPS workshop)*, 2012, pp. 13–20.

[9] D. Hadfield-Menell, E. Groshev, R. Chitnis, and P. Abbeel, "Modular task and motion planning in belief space," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, Sept 2015, pp. 4991–4998.

[10] S. Cambon, R. Alami, and F. Gravot, "A Hybrid Approach to Intricate Motion, Manipulation and Task Planning." *The International Journal of Robotics Research*, vol. 28, no. 1, pp. 104–126, 2009.

[11] E. Erdem, K. Haspalamutgil, C. Palaz, V. Patoglu, and T. Uras, "Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation," in *Proc. IEEE Int. Conf. Robotics and Automation*, May 2011, pp. 4575–4581.

[12] S. Srivastava, E. Fang, L. Riano, R.Chitnis, S. Russell, and P. Abbeel, "Combined Task and Motion Planning Through an Extensible Planner-Independent Interface Layer," in *Proc. IEEE Int. Conf. Robotics and Automation*, 2014.

[13] M. Stilman and J. Kuffner, "Planning among movable obstacles with artificial constraints," in *Workshop on the Algorithmic Foundations of Robotics*, July 2006.

[14] M. Dogar and S. Srinivasa.., "A framework for push-grasping in clutter," in *Proceedings of Robotics: Science and Systems*, Los Angeles, CA, USA, June 2011.

[15] L. Kaelbling and T. Lozano-Perez, "Hierarchical task and motion planning in the now," in *Proc. IEEE Int. Conf. Robotics and Automation*, May 2011, pp. 1470–1477.

[16] A. Krontiris and K. E. Bekris, "Efficiently solving general rearrangement tasks: A fast extension primitive for an incremental sampling-based planner," in *Proc. IEEE Int. Conf. Robotics and Automation*, May 2016, pp. 3924–3931.

[17] M. Stilman, J.-u. Schamburek, J. Kuffner, and T. Asfour, "Manipulation planning among movable obstacles," in *Proc. IEEE Int. Conf. Robotics and Automation*, 2007.

[18] K. Hauser, "The minimum constraint removal problem with three robotics applications," *The International Journal of Robotics Research*, vol. 33, no. 1, pp. 5 – 17, 2015.

[19] D. Stoffel, W. Kunz, and S. Gerber, "And/or graphs," *Technical Report*, 1995.

[20] C. Rosales, L. Ros, J. M. Porta, and R. Suárez, "Synthesizing grasp configurations with specified contact regions," *International Journal of Robotics Research*, vol. 30, no. 4, pp. 431–443, 2011.

[21] F. Gilart and R. Suarez, "Determining Force-Closure Grasps Reachable by a Given Hand," in *10th IFAC Symposium on Robot Control, SYROCO*, September 2012, pp. 235–240.

[22] J. Rosell, A. Pérez, A. Aliakbar, Muhayyuddin, L. Palomo, and N. García, "The kautham project: A teaching and research tool for robot motion planning," in *Proc. of the IEEE Int. Conf. on Emerging Technologies and Factory Automation, ETFA'14*, 2014.