High-Level Planning for Object Manipulation with Multi Heterogeneous Robots in Shared Environments

Alessandro Palleschi*, George Jose Pollayil*, Mathew Jose Pollayil*, Manolo Garabini*, Lucia Pallottino*

***Abstract*— Multi-robot systems are becoming increasingly popular in warehouses and factories, since they potentially enable the development of more versatile and robust systems than single robots. Multiple robots allow performing complex tasks with greater efficiency. However, this leads to increased complexity in planning and dispatching actions to robots. In this paper, we tackle such complexity using a hierarchical planning framework: the task is first planned at an abstract level and then refined by local motion planning. We propose a framework based on a state-transition system formalism that abstracts the problem by removing unnecessary details and, hence, considerably reduces planning space complexity. Forward search from an initial state allows the robot to find a sequence of actions to accomplish the assigned task. These actions can be planned at a lower level employing any motion planning technique available in the literature. The proposed method is validated through experiments in several operating conditions and scenarios.**

I. Introduction

The surge in e-commerce demands and the Industry 4.0 revolution are changing warehouses towards increasing automation [1]. This will promote multiple heterogeneous robots being employed in the same warehouse to increase performance in tasks, such as handling and moving goods. When deploying multi-robot systems, guaranteeing efficient operations requires the ability to plan, coordinate, and dispatch actions so as to accomplish desired goals. Such a class of problems is known as task and motion planning (TAMP) [2] and involves planning operations for robots sharing the same environment while interacting with objects.

A major challenge in TAMP lies in the high complexity of the planning problem itself. Indeed, traditional motion planning techniques, such as the ones relying on random-sampling-based algorithms or constrained optimizations [3], struggle when the planning space has to be augmented to include the state of several robots and of other objects in the environment [2]. Some authors try to overcome the complexity of the problem by exploiting its modal structure [4]: feasible changes belong to submanifolds of the full space (the *modes*). This leads to the definition of a hybrid search problem. At first, a sequence of discrete modes is planned. Then, a continuous set of mode parameters and motion paths to connect the different modes are found [5].

This work has received funding from the European Union's Horizon 2020 research and innovation program under agreements no. 73273 (ILIAD) and no. 101017274 (DARKO), and from the Italian Ministry of Education and Research (MIUR) in the framework of the CrossLab project (Departments of Excellence).

*Centro di Ricerca E. Piaggio and Dipartimento di Ingegneria dell'Informazione, Università di Pisa, Largo Lucio Lazzarino 1, Pisa, Italy {alessandro.palleschi, georgejose.pollayil, mathewjose.pollayil}@phd.unipi.it, manolo.garabini@gmail.com, lucia.pallottino@unipi.it

Corresponding author: Alessandro Palleschi

Research in TAMP also tries to integrate artificial intelligence (AI) approaches into robotic motion and multi-modal planning, since they can efficiently carry out task planning in large discrete spaces [6], [7].

Classical task planning has been exploited in manipulation planning for single robots: e.g., see [8]. Such approaches do not easily extend to multi-robot setup. A possible solution is shown in [9], which presents a TAMP method for heterogeneous multi-robot systems dedicated to transportation-like tasks. Other works focus on bridging high-level task planning and control. For instance, [10] divides manipulation tasks into sequences of sub-tasks using contact as the core element to build a contact graph, but it considers only manipulators. In [11], an autonomous manipulation framework for multi-robot planning is presented. It uses a semantic graph, where nodes derive from a discretization of grasp poses and workspaces, and arcs represent elementary actions that the robots can perform. However, a limiting assumption is that objects can be successfully grasped in order to be moved, which might not always be the case, e.g., for heavy or bulky objects that need to be pushed or pulled [12] or when direct grasping is hindered by environmental constraints [13]. Nonetheless, there are still many challenges to be faced when dealing with the different levels of abstractions and devising solutions to effectively pass information among them [14].

In this paper, we propose a framework for object manipulation planning with a team of multi heterogeneous robots. It is based on a multi-level structure that decomposes the problem over several levels of abstractions. We characterize the planning domain using an abstract representation of the world in which the robots are working. A formalization of this level through a state-transition system and a set of basic entities is presented. These entities model the states of the objects-agents system and the admissible actions so that the proposed high-level structure is both scenario and platform independent. Our approach autonomously generates a high-level plan to move the goods to the desired locations, combining the skills of the robots. The plan is then passed to the lower levels for refinement and execution. We propose a flexible structure that does not make any assumption nor has specific requirements on the class of search or planning algorithms used for the different levels of the hierarchy. In addition, we also propose interfaces between the highest abstract layer and the lower levels, to handle possible feasibility issues and failures, eventually finding alternative high-level plans with backtracking strategies. The framework has been tested through experiments on three different multi-robot systems.

II. Planning Domain

We wish to address high-level task planning for multi-robot objects manipulation: i.e. to find the sequence in which the robots should be used, which actions they should perform, and in which areas the actions should take place to move each object to the specified locations.

We model the problem through a state-transition system

$$\Sigma = \langle \mathrm{States},\ \mathrm{Actions},\ \phi \rangle, \tag{1}$$

where, States is a representation of the system composed by objects and robots, $\mathrm{Actions}$ represents the finite set of actions that can be performed by the robots on the objects, and ϕ is a state-transition function.

The States representation is carried out by modeling a set of entities, and their specific properties, which are used for planning purposes. To formalize the state-transition system (1), three finite sets are introduced and defined in the remainder of the section:

- E: a set of *entities* with the associated mathematical constants to represent their properties;
- Ψ: a set of *rigid relations* between the entities representing the constant properties of the system;
- Ξ: a set of *state variables* representing the changing properties of the system;

In this work, we use the *classical planning* assumptions provided in [6]: i) the environment is finite and static: the sets of states and actions are finite, and changes occur only in response to specific actions; ii) there is no specific model of time; iii) the system is deterministic, i.e., the state produced by applying a particular action in a given state is predictable and known.

Entities: For the problem of manipulating objects with multi-robot systems, we define three basic entities: *Objects*, *Agents*, and *Sectors*.

1) Objects: a set $\mathscr{O}$ of objects that can be moved by means of manipulation actions.

2) Agents: a set $\mathscr{A}$ of autonomous systems and parts of the environment that can be exploited for manipulating the *objects*. We classify the *agents* in $\mathscr{A}$ into robot, $\mathrm{transport}$, and $\mathrm{passive\ surface}$. The class robot represents the typical autonomous manipulation systems such as robotic manipulators (with both fixed or mobile base). The class $\mathrm{transport}$ includes mobile devices, such as AGVs or conveyor belts, that can be used to move and transport the object, or even surfaces such as pallets. The class $\mathrm{passive\ surface}$ contains surfaces that can be exploited as statically stable support during object manipulation

It is noteworthy that, unlike robot *agents*, $\mathrm{transport}$ *agents* are not able to directly transfer an *object* from/to another *agent*. Moreover, $\mathrm{passive\ surfaces}$ can only be used as support for an *object* while performing hand-off operations or non-prehensile manipulations, e.g., pushing, tilting, or top sliding. This type of *agent* cannot move an *object* by itself.

Given the three types of *agents*, the following subsets of $\mathscr{A}$ can be defined: $\mathscr{R} = \{a \in \mathscr{A} \,|\, a = \mathrm{robot}\}$, $\mathscr{T} = \{a \in \mathscr{A} \,|\, a = \mathrm{transport}\}$, $\mathscr{P} = \{a \in \mathscr{A} \,|\, a = \mathrm{passive\ surface}\}$.

3) Sectors: a set $\mathscr{S}$ of regions in which the *agents* operate. A *sector* $\sigma \in \mathscr{S}$ is an area of the global workspace in which a given subset of the *agents* can work and interact with an *object*. In summary, while the *agents* model the entities that can interact with each other and with the *objects*, *sectors* model the areas in which interactions can occur.

Having defined the above basic entities, the set E of the *entities* is defined as $\mathrm{E} = \mathscr{O} \cup \mathscr{R} \cup \mathscr{T} \cup \mathscr{P} \cup \mathscr{S} \cup \{\mathrm{null}\}$.

It is worth remarking that an exhaustive definition of the *entities* is crucial for the planning problem to be well-posed. It is good practice to define as $\mathrm{passive\,surface}$ any planar surface that is reachable by at least one robot, while the *sectors* should cover any part of the working area that is reachable by at least one robot or transfer agent. Thus, any area where object manipulation is possible is included into the state space.

Rigid Relations: Given the *entities* E, we introduce the following *rigid relations* among them:

A) **Adjacency**: spatial adjacency relation between ordered pairs of *sectors*. Adjacent *sectors* are in $\mathrm{adj} \subseteq \mathscr{S} \times \mathscr{S}$;
B) **Reachability**: the reachability relation is between a *sector* $\sigma_i \in \mathscr{S}$ and a $\mathrm{transport}$ agent or a robot that can reach σ_i. Entities in reachability relation are in $\mathrm{canAct} \subseteq \{(a, \sigma) | a \in \{\mathscr{R} \cup \mathscr{T}\}, \sigma \in \mathscr{S}\}$;
C) **Stability**: the stability relation is between a set of *agents* $A \subseteq \mathscr{A}$ and an object, that can be statically supported when in contact with agents in A. Entities in stability relation are in $\mathrm{stable} \subseteq \{\mathscr{O} \times A | A \subseteq \mathscr{A}, A \neq \emptyset\}$;
D) **Transportability**: the transportability relation is between a $\mathrm{passive\,surface}$ and a $\mathrm{transport}$ *agent*, that can move the surface, e.g., an automated forklift moving a pallet. Entities in transportability relation are in $\mathrm{canMove} \subseteq \{\mathscr{T} \times \mathscr{P}\} \cup \{\mathrm{null}\}$.

Finally, the overall set of *rigid relations* is defined as $\Psi = \{\mathrm{adj},\ \mathrm{canAct},\ \mathrm{stable},\ \mathrm{canMove}\}$.

State Variables: The *state variables* formally describe the present condition of the system, specifying the contacts between the *objects* and the *agents* and the *sectors* in which they are currently located

- $\mathrm{hold}[a] \subseteq \mathscr{O}_\emptyset$: the set of *objects* $o \in \mathscr{O}$ that are in contact with the *agent* $a \in \mathscr{A}$;
- $\mathrm{onA}[o] \subseteq \mathscr{A}$: the set of *agents* $a \in \mathscr{A}$ in contact with the *object* $o \in \mathscr{O}$;
- $\mathrm{load}[t] \in \mathscr{P}_\emptyset$: the $\mathrm{passive\,surface}$ $p \in \mathscr{P}$ loaded on the $\mathrm{transport}$ *agent* $t \in \mathscr{T}$;
- $\mathrm{onT}[p] \in \mathscr{T}_\emptyset$: the $\mathrm{transport}$ *agent* $t \in \mathscr{T}$ on which the $\mathrm{passive\ surface}$ $p \in \mathscr{P}$ is loaded;
- $\mathrm{at}[o] \in \mathscr{S}$: the location $\sigma \in \mathscr{S}$ of each *object* $o \in \mathscr{O}$;
- $\mathrm{at}[a] \in \mathscr{S}$: the location $\sigma \in \mathscr{S}$ of each *agent* $a \in \mathscr{A}$;

where $S_\emptyset \equiv S \cup \{\mathrm{null}\}$.

Thus, the *state variables* are:

$$\begin{aligned}\Xi = \{&\mathrm{hold}[a], \mathrm{onA}[o],\ \mathrm{load}[t],\ \mathrm{onT}[p],\ \mathrm{at}[a],\ \mathrm{at}[o] \\ &| \, a \in \mathscr{A},\ t \in \mathscr{T},\ p \in \mathscr{P},\ o \in \mathscr{O}\}.\end{aligned} \tag{2}$$

For taming the complexity, we assume that a robot can only contact one object at a time; therefore $\mathrm{hold}[r] = o$ if the robot $r \in \mathscr{R}$ is in contact with the object $o \in \mathscr{O}$, otherwise $\mathrm{hold}[r] = \{\mathrm{null}\}$. Instead, we allow $\mathrm{passive\ surfaces}$ and

TABLE I

ACTION DEFINITIONS FOR THE PROPOSED PLANNING DOMAIN.

name	**moveF**	**moveP**	**moveOnP**	**moveH**
ent	$(a,\sigma) \in \mathscr{A} \times \mathscr{S}$	$(p,t) \in \mathscr{P} \times \mathscr{T},\ \sigma \in \mathscr{S}$	$o \in \mathscr{O},\ (R,p) \in \wp(\mathscr{R}) \times \mathscr{P},\ \sigma \in \mathscr{S}$	$o \in \mathscr{O}\,, (A,\sigma) \in \wp(\mathscr{A}) \times \mathscr{S}$
pre	$\mathbf{adj}(\mathbf{at}[a],\sigma)$ $\mathbf{canAct}(a,\sigma)$ $\mathbf{hold}[a] = \text{null}$ $\mathbf{at}[a] \neq \sigma$ $a \notin \mathscr{P}$	$\mathbf{adj}(\mathbf{at}[p],\sigma)$ $\mathbf{canAct}(t,\sigma)$ $\mathbf{onT}[p] = t$ $\mathbf{hold}[p] = \mathbf{hold}[t] = \text{null}$ $\mathbf{at}[t] \neq \sigma,\ \sigma \notin \mathbf{at}[p]$	$\mathbf{canAct}(R,\sigma)$ $\sigma \in \mathbf{at}[p]$ $\mathbf{onA}[o] = \{R,p\}$ $\mathbf{hold}[r] = o\, \forall r \in R$ $\mathbf{at}[r] = \mathbf{at}[o] \neq \sigma\ \forall r \in R$	$\mathbf{adj}(\mathbf{at}[o],\sigma)$ $\mathbf{canAct}(a,\sigma)\ \forall a \in A$ $\mathbf{onA}[o] = A$ $\mathbf{hold}[a] = o\ \forall a \in A$ $\mathbf{at}[a] = \mathbf{at}[o] \neq \sigma\ \forall a \in A$
eff	$\mathbf{at}[a] \leftarrow \sigma$	$\mathbf{at}[t], \mathbf{at}[p] \leftarrow \sigma$	$\mathbf{at}[r], \mathbf{at}[o] \leftarrow \sigma$	$\mathbf{at}[o], \mathbf{at}[r] \leftarrow \sigma\, \forall r \in R$

name	**transport**	**transportOnP**	**pickFromP**	**pickFromT**	**loadP**
ent	$t \in \mathscr{T}$ $\sigma \in \mathscr{S}$	$(p,t) \in \mathscr{P} \times \mathscr{T}$ $\sigma \in \mathscr{S}$	$o \in \mathscr{O}$ $(p,R) \in \mathscr{P} \times \wp(\mathscr{R})$	$o \in \mathscr{O}$ $(t,R) \in \mathscr{T} \times \wp(\mathscr{R})$	$p \in \mathscr{P}$ $t \in \mathscr{T}$
pre	$\mathbf{canAct}(t,\sigma)$ $\mathbf{hold}[t] \neq \text{null}$ $\mathbf{at}[t] \neq \sigma$	$\mathbf{canAct}(t,\sigma)$ $\mathbf{hold}[p] \neq \text{null}$ $\mathbf{hold}[t] = \text{null}$ $\mathbf{onT}[p] = t$ $\mathbf{at}[t] \neq \sigma,\ \sigma \notin \mathbf{at}[p]$	$\mathbf{hold}[p] = o$ $\mathbf{hold}[r] = o\, \forall r \in R$ $\mathbf{isStable}(o, \mathbf{onA}[o] \setminus \{p\})$	$\mathbf{hold}[t] = o$ $\mathbf{hold}[r] = o\, \forall r \in R$ $\mathbf{isStable}(o, \mathbf{onA}[o] \setminus \{t\})$	$\mathbf{at}[p] = \mathbf{at}[t]$ $\mathbf{hold}[t] = \text{null}$ $\mathbf{load}[t] = \text{null}$ $\mathbf{onT}[p] = \text{null}$
eff	$\mathbf{at}[t] \leftarrow \sigma$ $\mathbf{at}[o_i] \leftarrow \sigma\ \forall o_i \in \mathbf{hold}[t]$	$\mathbf{at}[t], \mathbf{at}[p] \leftarrow \sigma$ $\mathbf{at}[o_i] \leftarrow \sigma\ \forall o_i \in \mathbf{hold}[p]$	$\mathbf{hold}[p] \leftarrow \mathbf{hold}[p] \setminus \{o\}$ $\mathbf{onA}[o] \leftarrow \mathbf{onA}[o] \setminus \{p\}$	$\mathbf{hold}[t] \leftarrow \mathbf{hold}[t] \setminus \{o\}$ $\mathbf{onA}[o] \leftarrow \mathbf{onA}[o] \setminus \{t\}$	$\mathbf{load}[t] \leftarrow p$ $\mathbf{onT}[p] \leftarrow t$

name	**placeOnP**	**placeOnT**	**positionR**	**removeR**	**unloadP**
ent	$o \in \mathscr{O}$ $(p,R) \in \mathscr{P} \times \wp(\mathscr{R})$	$o \in \mathscr{O}$ $(t,R) \in \mathscr{T} \times \wp(\mathscr{R})$	$o \in \mathscr{O}$ $r \in \mathscr{R}$	$o \in \mathscr{O}$ $r \in \mathscr{R}$	$p \in \mathscr{P}$ $t \in \mathscr{T}$
pre	$\mathbf{hold}[r] = o\, \forall r \in R$ $\mathbf{at}[r] \in \mathbf{at}[p]\, \forall r \in R$ $\mathbf{onA}[o] \cap \mathscr{P} = \text{null}$ $\mathbf{onA}[o] \cap \mathscr{T} = \text{null}$	$\mathbf{hold}[r] = o\, \forall r \in R$ $\mathbf{at}[t] = \mathbf{at}[r]\, \forall r \in R$ $o \notin \mathbf{hold}[t]$ $\mathbf{onA}[o] \cap \mathscr{P} = \text{null}$ $\mathbf{onA}[o] \cap \mathscr{T} = \text{null}$	$\mathbf{hold}[r] = \text{null}$ $r \notin \mathbf{onA}[o]$ $\mathbf{at}[o] = \mathbf{at}[r]$	$\mathbf{hold}[r] = o$ $r \in \mathbf{onA}[o]$ $\mathbf{isStable}(o, \mathbf{onA}[o] \setminus \{r\})$	$\mathbf{onT}[p] = t$
eff	$\mathbf{hold}[p] \leftarrow \mathbf{hold}[p] \cup o$ $\mathbf{onA}[o] \leftarrow p$	$\mathbf{hold}[t] \leftarrow \mathbf{hold}[t] \cup o$ $\mathbf{onA}[o] \leftarrow t$	$\mathbf{hold}[r] \leftarrow o$ $\mathbf{onA}[o] \leftarrow \mathbf{onA}[o] \cup r$	$\mathbf{hold}[r] \leftarrow \text{null}$ $\mathbf{onA}[o] \leftarrow \mathbf{onA}[o] \setminus \{r\}$	$\mathbf{load}[t] \leftarrow \text{null}$

We have used the notation $S(x)$ to indicate $x \in S$ and $\wp(S)$ for the powerset of S

transport *agents* to be in contact with multiple objects. Hence, the variables $\mathrm{hold}[p]$ and $\mathrm{hold}[t]$ are the lists, possibly empty, of objects in contact with a passive surface *agent* $p \in \mathscr{P}$ and a transport *agent* $t \in \mathscr{T}$, respectively. On the other hand, an *object* needs to be in contact with at least an *agent* to be statically stable. Thus, $\mathrm{onA}[o]$ is the non-empty list of *agents* that are in contact with an *object* o. The variable $\mathrm{onT}[p]$ is the transport *agent* $t \in \mathscr{T}$, if any, that is holding the passive p. Finally, the *state variables* $\mathrm{at}[\cdot]$ gives the current location of objects and *agents*.

To summarize, a state of our system represents a particular configuration of the *objects* and *agents* in which each *object* o is in contact with a subset of the *agents* in a sector σ.

***Actions and State-Transition Function**:* The set Actions in (1) is a representation of the finite set of actions that can be performed by the agents in $\mathscr{A}$ on the objects in $\mathscr{O}$.

Action models can be derived using action templates [6]. An action template for States can be defined as a tuple $\alpha = \langle \mathrm{name}, \mathrm{ent}, \mathrm{pre}, \mathrm{eff} \rangle$. Here, name is the name of the action and $\mathrm{ent} \in \mathrm{E}$ is a set of entities involved in the action. The preconditions pre are a set of conditions that must be verified for the action to be applicable. Finally, the predicted outcome of the action is specified by the effect eff, e.g., the *state variables* that are affected by the action and their newly assigned values. The state resulting from an action can be computed using the state-transition function $\phi(s,\nu)$. This function takes as inputs a state s and the action ν, and checks the preconditions in pre for the parameters in ent. If pre are verified, ϕ outputs a new state specified by eff.

For our problem the list of *actions* that captures the basic object handling capabilities for a large class of multi heterogeneous robot systems is the following:

- ***moveF***: Move an agent, being it a robot or a transport agent, that is not holding an object.
- ***moveP***: Move a passive surface (with the aid of a transport agent) that is not holding an object.
- ***moveH***: Move an object while it is held by an agent (or a group of agents).
- ***moveOnP***: Move an object on a passive surface with the aid of a robot in such a way that contact with the surface is maintained.
- ***transport***: Transport an object while it is placed on a transport agent.
- ***transportOnP***: Transport an object while it is on a passive surface with the aid of a transport agent.
- ***pickFromP/pickFromT-placeOnP/placeOnP***: Pick/place an object from/on a passive surface or a transport agent.
- ***positionR/removeR***: Position/remove contacts of a robot with an object.
- ***loadP/unloadP***: Load/Unload a passive surface with a transport agent.

Table I provides the formal definitions of the fields of the template α for the above basic actions. It is also possible to associate a cost to each action. This would clearly depend on the particular action and the entities involved.

***Planning problem**:* Having defined all the components of our planning domain Σ in (1), the *task planning problem* $\mathbb{P}$ is defined as a triple $\langle \Sigma, s_0, G_s \rangle$, where $s_0 \in$ States is the initial state of the system, and $G_s \subseteq$ States represents the set of goal states. The solution of $\mathbb{P}$ is a *task plan*, i.e., a finite sequence of *actions* $\pi = \{\nu_1, \ldots, \nu_N\}$, applicable from s_0 and such that $\phi(s_0, \pi) \in G_s$. We recall that a sequence

of *actions* is said to be applicable in a generic state s if there exists a finite sequence of states $\{s_0, \dots, s_N\}$ such that $\phi(s_{i-1}, \nu_i) = s_i$ for $i = 1, \dots, N$ [6]. The initial state s_0 needs to be identified from the scenario by means of lower level sensing, or might even be given as an input by a human operator. Now, given a set of final desired positions of the objects to be moved, there are multiple approaches that can be pursued to choose the goal set G_s. A simple choice would be to set G_s as the subset of States such that the locations of the *objects* are in the required *sectors*, i.e., $\mathrm{at}[o] \in \mathscr{S}_{G_s} \subseteq \mathscr{S}$.

III. Task Planning and Execution

In this section, we first describe how the *planning problem* $\mathbb{P}$, presented in Sec. II-.3, can be solved in order to find a high-level plan π for the multi-robot system. Then, considerations on how to relate the plan π to motion planning and low-level control, are reported.

***Task Planning**:* A straightforward approach to solve $\mathbb{P}$ is to use state-space search. We focus our attention on forward-search of which many deterministic implementations exist [15]. These can be broadly categorized into two classes: *uninformed* methods, such as breadth-first or Uniform Cost Search, do not consider any prior information on the goal location; *heuristic-based* methods, such as A* or greedy best-first, exploit a heuristic function $h(s)$ to guide the search by selecting the most promising states.

In general, if optimality is not required, a valid solution might be to use a greedy best-first method. Instead, if optimal solutions are needed, it might be better to employ A*-like algorithms. Note that one can either use optimality to obtain plans with minimum number of actions or consider different costs for each action and state. Costs can take into account several aspects, such as energy consumption, robustness of manipulation, and physical properties of the object. The specific choice for the search algorithm will influence the retrieved plan, in terms of number of actions and used agents.

***Simultaneous actions**:* In this work, we are implicitly constraining only one agent to take an action at each step. Even though this approach has the advantage of reducing planning complexity, the solutions found do not allow concurrent execution of actions typical of multi-robot systems. A possible way to lift this constraint is to allow each agent to execute at most one action at each step. Clearly, it should be ensured that the combinations of simultaneous actions are not in conflict, i.e., the preconditions of an action ν_j executed by an agent a_i are not violated by the effects of a simultaneous action ν_k by an agent a_h, and vice versa.

This can be achieved by augmenting the set of actions to include all possible combinations of feasible simultaneous actions. The preconditions, entities, and effects of these new actions would be the union of the preconditions, entities, and effects of each action, while the cost could be the sum of the single costs. Another approach is to inspect the plan π returned by the planner so as to identify sequences of actions that can be executed simultaneously. As stated previously, these can be identified by checking the preconditions and effects of consecutive actions so as to find the ones that can be executed concurrently.

***Plan Refinement**:* Once the employed state-space search algorithm has solved the *planning problem*, the solution π will be a sequence of symbolic actions, as those presented in Table I, that the agents should execute to achieve a desired goal. These high-level actions should be passed to lower-level solvers to actually plan the motion of the robots.

While the presented high-level framework retrieves long-horizon plans, the other levels of the hierarchy will deal, on shorter horizons, with all the low-level constraints, which cannot be considered at an abstract level. Examples are the presence of possible obstacles, the characteristics of the robots and their kinematics. Indeed, w.r.t. the classic formalism, in TAMP the actions template is augmented to include continuous parameters and a set of constraints on them that specify whether a certain transition is feasible [2]. These parameters and constraints are not explicitly reported for the sake of space. For instance, the initial and goal configurations of the robots and the trajectories connecting them should be continuous and collision-free, or fulfill kinematic constraints if the robots are in contact with objects. These can be found by solving constraint satisfaction problems (CSP) [2], or with learning methods [16]. Even though the development of an integrated framework for task and motion planning (TAMP) is beyond the scope of this paper, hereafter, we present some considerations on how the symbolic actions in π can be planned at lower levels. We use the sequencing first approach, in which a complete plan skeleton π is first found by our high-level framework, and then this sequence of actions is passed to lower-level solvers for execution. The plan π is decomposed into a series of subplans $\pi = \{\pi^0, \dots, \pi^{L-1}\}$ that are planned and then executed. Each subplan is associated to the relative subsequence of actions $\pi^i = \{\nu_1^i, \dots, \nu_{N_i}^i\}$ and the relative subset of states $\mathrm{States}(\pi^i) = \{s_0^i, \dots, s_{N_i}^i\}$, where $s_0^i \equiv s_{N_{i-1}}^{i-1}$ $\forall i = 0, \dots L-1$.

In general, low values of L correspond to longer planning horizon and longer (w.r.t the size of the overall plan) subsequences of actions to be planned and executed. This could raise the computational effort and planning times. Besides, a higher number of L corresponds to shorter action sequences that might be faster to be processed by the lower-level solvers. However, these local solutions are unaware of the rest of the plan, and the chance of ending up in configurations affecting the feasibility of the next subplans increases.

We propose the following procedure to decompose π into subplans. Starting from the first action ν_1, inspect the actions ν_i of π until a ν_f equal to $\mathrm{placeOnP}$, $\mathrm{placeOnT}$, or $\mathrm{removeR}$ is found. Then, search the remaining actions until a ν_{f+j} different from a $\mathrm{removeR}$ is found, or the plan is terminated. These are actions where an object is placed on a passive surface - e.g., for regrasping - or on a mobile base - e.g., to move it to another sector - and all the agents that will not be used for the following manipulations are removed. The resulting states are good candidates for separation between subplans. Thus, the set of actions $\{\nu_1, \dots, \nu_f, \dots, \nu_{f+j}\}$ is the first subplan π^0. The next ones are constructed iterating the same procedure from ν_{f+j+1}.

The lower planning of a subplan π^i starts only when the previous subplan π^{i-1} has been planned and correctly

Algorithm 1 Backtracking Procedure

procedure BACKTRACKING($s^i_{k-1}, s^i_{N_i}, \nu^i_k, method$)
  $curr\_state \leftarrow s^i_{k-1}$, $plan \leftarrow \emptyset$
  $actions \leftarrow curr\_state$.applicable_actions $\setminus \{\nu^i_k\}$
  while true **do**
    if $curr\_state = s^i_{N_i}$ **then**
      return *plan*
    if not *actions* **then**
      return NONE
    $action \leftarrow$ choose_action($actions, method$)
    plan.append(*action*)
    $curr\_state \leftarrow$ predict_state($curr\_state, action$)
    $actions \leftarrow curr\_state$.applicable_actions

Algorithm 2 Replanning Procedure

procedure REPLAN($s^i_0, s^i_{N_i}, G_s, method$)
  $curr\_state \leftarrow s^i_0$, $plan \leftarrow \emptyset$
  $s^i_{N_i}$.applicable_actions $\leftarrow \emptyset$
  while true **do**
    if $curr\_state \in G_s$ **then**
      return *plan*
    $actions \leftarrow curr\_state$.applicable_actions
    if not *actions* **then**
      return NONE
    $action \leftarrow$ choose_action($actions, method$)
    plan.append(*action*)
    $curr\_state \leftarrow$ predict_state($curr\_state, action$)

executed. However, it might not be always possible to convert every high-level subplan into feasible low-level solutions or a feasible solution might not be correctly executed by an agent. We will denote these two different scenarios as **planning** failures and **execution** failures, respectively. In the following, we discuss some solutions to recover from such failures.

Planning Failures: This type of failure occurs when one of the low-level planners finds a generic action $\nu^i_j \in \pi^i$ to be unfeasible. In this case, we implement a backtracking strategy (Algorithm 1), as common for other top-down approaches [17], to find an alternative sequence of high-level actions. More in detail, after identifying the unfeasible action ν^i_j and the previous state s^i_{j-1}, our high-level planning module is invoked to find an alternative sequence of actions connecting the last feasible state and the final state (s^i_{j-1} and $s^i_{N_i}$) associated to π^i. To avoid ending up with a plan equal to π^i, the search is executed removing ν^i_j from the admissible actions when in s^i_{j-1}. If such a sequence exists, it is appended to the feasible part of the plan π^i, and the new subplan $\tilde{\pi}^i$ is sent to the low-level planners.

If it does not exist, we mark the action ν^i_{j-1} as unfeasible (since it would lead only to unfeasible paths). Hence, we perform the same backtracking procedure starting with s^i_{j-2} as initial node and $s^i_{N_i}$ as goal and removing ν^i_{j-1} from the admissible actions from s^i_{j-2}. Backtracking is performed until a feasible subplan is found or no more actions are left to be checked in π^i, i.e., $s^i_{j-k} = s^i_0$. It is worth remarking that the proposed solution allows in principle to reduce the complexity of the search, since we replan only the part of policy π which is currently unfeasible, without the need to find a whole new plan. In the worst case, the procedure will explore all the paths connecting s^i_0 to $s^i_{N_i}$.

It might also occur that a low-level feasible plan from s^i_0 to $s^i_{N_i}$ does not exist. In this case, the high-level planner is asked to find a new plan $\tilde{\pi}$ starting from s^i_0 to a state $\tilde{s}_N \in G_s$, that does not include $s^i_{N_i}$. As reported in Algorithm 2, all the actions in $s^i_{N_i}$ are marked as not admissible, i.e., making its frontier empty. Clearly, it is not possible to have guarantees on the optimality of the new plan, or even that a solution exist, depending on the initial state s^i_0.

Execution Failures: As specified in Sec. II, we consider a deterministic system; the state produced by an action is unique and known. However, in many practical cases, an action may either succeed or fail when executed because of, e.g., incorrect or partial information on the environment, execution failures, or unmodelled events. Thus, the execution may lead to one among a set of different states. It could still make sense to use a deterministic model as long as the system is able to monitor the action execution and detect a failure, and use failure-recovery mechanisms (e.g., by replanning or by re-acting). This can be achieved by providing the agents with the capability of sensing/estimating the actual state resulting from executing actions. If the perceived state is different from the expected one, it is necessary to call again the high-level planner to find a new plan updating the initial conditions accordingly to the information gathered at execution time.

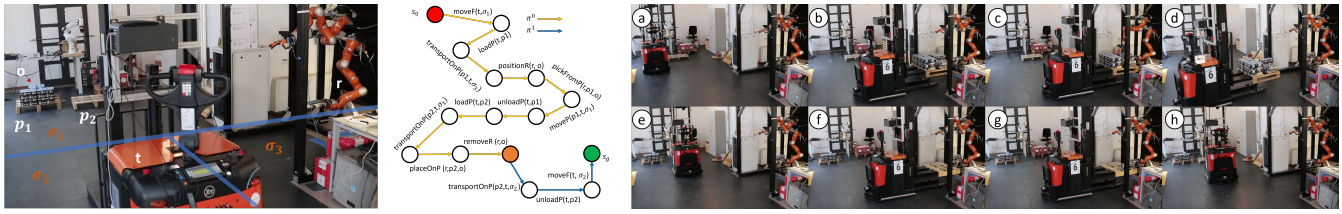
Nonetheless, an erroneous execution of an action could theoretically steer the system into a configuration, which is not part of the created state space, thereby precluding the possibility of finding a solution to the planning problem. However, as mentioned in Sec.II, an exhaustive definition of the entities should include any configuration where object manipulation is possible in the state space. In this way, only the configurations in which manipulation actions are not possible (e.g., because the area is not reachable by any agent) are left out of the state space.

Discussion on plan refinement: Given the proposed high-level framework and the mechanisms to connect it to the lower layers of the planning structure, specific choices of the lower planning and control algorithms will influence the performance and the robustness of the overall architecture, or even the possibility of devising a feasible plan. Indeed, using inefficient low-level strategies will increase failures, whereas effective and/or reactive methods will increment robustness and the chances of correctly planning and executing the computed high-level strategy. Nevertheless, these kinds of issues are specifically dependent on the environment, the employed robots, the used planning and control algorithms, and even the sensors the robots are equipped with.

IV. VALIDATION

In this section, we report the validation of the proposed framework on three different robotic systems: (A) **Two-robot system** for picking and palletizing - an autonomous guided forklift and a bimanual manipulation system; (B) **Three-robot system** for pick and place - two manipulators and a mobile base; (C) **One-robot system** for rearrangement planning - a bimanual manipulation system.

The high-level plans are retrieved using three different and largely used search algorithms [15]: i) Breadth-first:



(a) Setup. (b) Planned Task. (c) Plan Execution.

Fig. 1. Task A: Task plan using Uniform Cost Search and execution for picking and palletizing operations in a warehouse.

an uninformed search algorithm that does not consider the costs, ii) Uniform Cost Search (UCS): an uninformed search algorithm that does consider the costs, and iii) A*: an informed search algorithm. For A*, the used heuristics $h(s)$ is equal to the number of state variables in s that are different from a goal state g. The search algorithms were implemented in MATLAB on a Laptop PC equipped with an Intel Core i7 Processor (6x2.20 GHz) and 16 GB DDR4 RAM.

The different computation times (average time over 1000 runs), length and cost of the retrieved plans, and the number of visited states for each algorithm are reported in Table II. In the table we report the plans' nominal times, length, and cost, while between brackets the actual values considering also possible backtracking procedures. The number of asterisks $*$ specifies the number of backtracking operations. The plan retrieved with UCS is then refined by the low-level solvers and executed. The average time needed by each agent to plan an action is finally reported in Table III.

(A) Two-robot system for picking and palletizing: Here, we model and plan a picking and palletizing operation.

Planning Domain: In this scenario, we have four agents: an automated forklift, called *t*; two pallets, called *p1* and *p2*; a fixed base dual-arm robotic platform, WRAPP-up [18], designed to perform autonomous picking and palletizing operations, called *r*. We have a single object, $\mathscr{O} = \{o\}$, and three sectors, i.e., $\mathscr{S} = \{\sigma_1,\ \sigma_2,\ \sigma_3\}$, as shown in Fig.1a.

The set of rigid relations Ψ is given by $\mathbf{adj} = \{(\sigma_1, \sigma_2), (\sigma_2,\ \sigma_1),\ (\sigma_1,\ \sigma_3),\ (\sigma_3,\ \sigma_1),\ (\sigma_2,\ \sigma_3),\ (\sigma_3,\ \sigma_2)\}$, $\mathbf{canAct} = \{(\mathrm{r},\ \sigma_3),\ (\mathrm{t},\ \sigma_1),\ (\mathrm{t},\ \sigma_2),\ (\mathrm{t},\ \sigma_3)\}$, $\mathbf{stable} = \{(o,\ \mathrm{r}),\ (o,\ \mathrm{p1}),\ (o,\ \mathrm{p1},\ \mathrm{r}),\ (o,\ \mathrm{p2}),\ (o,\ \mathrm{p2},\ \mathrm{r})\}$ $\mathbf{canMove} = \{(\mathrm{t}, \mathrm{p1}), (\mathrm{t}, \mathrm{p2})\}$.
This planning domain corresponds to a state-space of dimension 375.

Planning and Execution: The initial state has the object on the pallet $p1$, with both pallets in sector σ_1, the unloaded forklift in σ_2, the manipulator in σ_3. This formally corresponds to the state:

$$s_0 = \{\mathrm{hold[r]} = \mathrm{null},\ \mathrm{hold[p1]} = o,\ \mathrm{hold[p2]} = \mathrm{null},\ \mathrm{hold[t]} = \mathrm{null},\ \mathrm{load}[t] = \mathrm{null},\ \mathrm{onA}[o] = \mathrm{p1},\ \mathrm{onT}[p] = \mathrm{null},\ \mathrm{at}[o] = \sigma_1,\ \mathrm{at[r]} = \sigma_3,\ \mathrm{at[p1]} = \sigma_1,\ \mathrm{at[p2]} = \sigma_1,\ \mathrm{at[t]} = \sigma_2\}.$$

In the desired final configuration the object is on $p2$, with both pallets in sector σ_1, the unloaded forklift is in σ_2. This matches the set of goal states G_s given by $G_s = \{s \in \mathrm{States} \mid \mathrm{hold}[p2] = o \wedge \mathrm{at}(o) = \sigma_1 \wedge \mathrm{at}(p2) = \sigma_1 \wedge \mathrm{at}(p1) = \sigma_1 \wedge \mathrm{at}(t) = \sigma_2 \wedge \mathrm{onT}[p2] = \mathrm{null} \wedge \mathrm{load}[t] = \mathrm{null}\}$.

For this case, the three algorithms return the same high-level plan (depicted in Fig. 1b), with Breadth-first being faster than UCS and A*. A* is the one exploring fewer states, as its search is guided by the heuristics to estimate how close a state is to the goal. Fig. 1b shows also the two subplans in which the plan is separated. As expected for this simple case, the output of the planner first commands the forklift to move to the correct sector to load the pallet with the object to pick. Then, the forklift moves the pallet to the robot station, where the robot picks the object. To plan at the lower level the picking action, we use the reactive planner described in [18] and [19], which allows robust and efficient picking and placing operations for cuboids and cylinders. The average times needed to plan an action for the two agents (t and r) are reported in Tab. III. The robot holds the object, waiting for the forklift to unload the pallet and load the other one. Eventually, the robot places the object on the target pallet and the forklift unloads it in the target sector. A photo sequence of an execution of this plan is reported in Fig.1c.

(B) Three-robot system for pick and place: We now consider the task of moving an object, a roll of duct tape, between non-adjacent surfaces in presence of two manipulators and a mobile platform (see Fig. 2).

Planning Domain: In this scenario, we have six agents: three passive surfaces, called p_1, p_2, and p_3; two robots, a Panda arm by Franka Emika and a Universal Robot UR10e, both equipped with the Pisa/IIT SoftHand, called r_1 and r_2; one transport agent called t, a Robotnik SUMMIT-XL STEEL. There is one object to move $\mathscr{O} = \{o\}$, and four sectors, i.e., $\mathscr{S} = \{\sigma_1,\ \sigma_2,\ \sigma_3,\ \sigma_4\}$, as shown in Fig.2a.

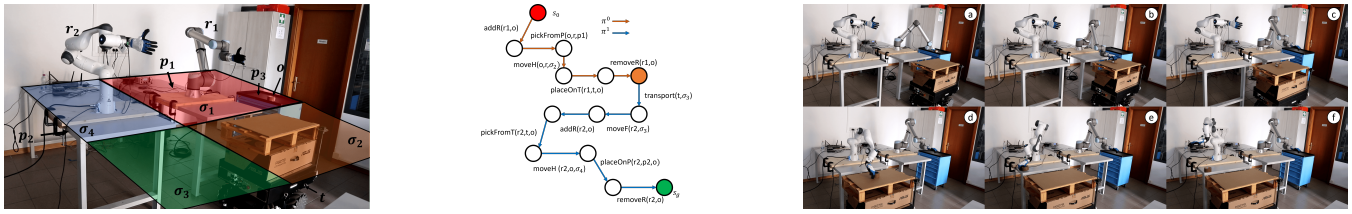
The set of rigid relations Ψ is given by $\mathbf{adj} = \{(\sigma_1, \sigma_2), (\sigma_2, \sigma_1), (\sigma_2, \sigma_3), (\sigma_3, \sigma_2), (\sigma_3, \sigma_4), (\sigma_4, \sigma_3), (\sigma_4, \sigma_1), (\sigma_1, \sigma_4)\}$, $\mathbf{canAct} = \{(\mathrm{r}_1, \sigma_1), (\mathrm{r}_1, \sigma_2), (\mathrm{t}, \sigma_2), (\mathrm{t}, \sigma_3), (\mathrm{r}_2, \sigma_3), (\mathrm{r}_2, \sigma_4)\}$, $\mathbf{stable} = \{(o, \mathrm{r}_1), (o, \mathrm{r}_2), (o, \mathrm{r}_1, \mathrm{r}_2), (o, \mathrm{p}_1), (o, \mathrm{p}_2), (o, \mathrm{p}_3), (o, \mathrm{p}_1, \mathrm{r}_1), (o, \mathrm{p}_2, \mathrm{r}_2), (o, \mathrm{p}_3, \mathrm{r}_1), (o, \mathrm{t}), (o, \mathrm{t}, \mathrm{r}_1), (o, \mathrm{t}, \mathrm{r}_2)\}$, $\mathbf{canMove} = \{\mathrm{null}\}$.
This planning domain corresponds to a state-space of dimension 84.

Planning and Execution: Initially, the object is on the surface p_3 corresponding to the initial state

$$s_0 = \{\mathrm{hold}[\mathrm{r}_1] = \mathrm{null},\ \mathrm{hold}[\mathrm{r}_2] = \mathrm{null},\ \mathrm{hold}[p_1] = \mathrm{null},\ \mathrm{hold}[p_2] = \mathrm{null},\ \mathrm{hold}[p_3] = o,\ \mathrm{hold}[t] = \mathrm{null},\ \mathrm{onA}[o] = \mathrm{p}_3,\ \mathrm{at}[o] = \sigma_1,\ \mathrm{at}[\mathrm{r}_1] = \sigma_1,\ \mathrm{at}[\mathrm{p}_1] = \sigma_1,\ \mathrm{at}[\mathrm{p}_2] = \sigma_4,\ \mathrm{at[t]} = \sigma_2,\ \mathrm{load}[t] = \mathrm{null}\},.$$

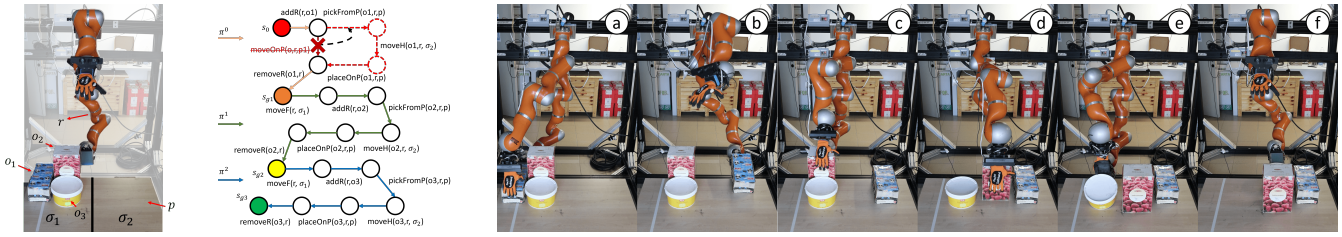
Since we want to move the object to p_2, the set of goal states is $G_s = \{s \in \mathrm{States} \mid \mathrm{hold}[\mathrm{p}_2] = o \wedge \mathrm{hold}[\mathrm{r}_2] = \mathrm{null}\}$.

Again, the three algorithms return the same solution, shown in Fig. 2b, with breadth-first being the fastest. Also, the algorithms explore the same number of states. We also highlight in Fig. 2b the two subplans in which the plan has been divided. As a classic hand-over of the object is not



(a) Setup. (b) Planned Task. (c) Plan Execution.

Fig. 2. Task B: Task plan using Uniform Cost Search and execution for moving an object with two robots and a transport agent.



(a) Setup. (b) Planned Task. (c) Plan Execution.

Fig. 3. Task C: Task plan using Uniform Cost Search and execution for the objects' rearrangement case.

TABLE II

HIGH-LEVEL PLANNING: STATISTICS

	Breadth-First				Uniform Cost Search				A*			
	T[ms]	Visited	L_π	Cost	T[ms]	Visited	L_π	Cost	T[ms]	Visited	L_π	Cost
A	0.09	326/375	14	22	0.14	319/375	14	22	0.16	257/375	14	22
B	0.09	78/84	12	36	0.11	78/84	12	36	0.13	76/84	12	36
C	0.07* (0.14)	61/64	11* (13)	62* (71)	0.08* (0.16)	61/64	15* (17)	44* (53)	0.12* (0.23)	60/64	15* (17)	44* (53)

TABLE III

LOW-LEVEL PLANNING: STATISTICS

	A		**B**			**C**
	r	t	r_1	r_2	t	r
Times	12.1ms	37.3ms	9.20ms	10.2ms	21.1ms	11.2ms

possible, the robot r_1 has first to grasp the tape and place it on the mobile platform. The platform moves to bring the object to a sector reachable by r_2 (σ_3), that eventually picks it from the platform and places it on the table p_2. The grasp poses for the manipulators were generated using [20], the motion of the manipulators were planned using the RRT-Connect [21] algorithm provided by OMPL with MoveIt! [22], while for the mobile platform a classical point-to-point navigation is used. The average time needed by each agent to plan an action using RRT-Connect is reported in Tab. III. We chose RRT-Connect for planning the motion of the two manipulators as it is, in general, more efficient than RRT [21], and faster than RRT* [23]. RRT would require, in average, 0.0092s and 0.013s for planning an action for r_1 and r_2, respectively. RRT* takes instead 5.003s and 5.004s. All the algorithms had a maximum planning time of 5 seconds. A photo sequence of the plan execution is shown in Fig.2c

(C) One-robot system for rearrangement planning: Here, we consider a scenario in which three objects must be rearranged with a bimanual system (see Fig. 3) with the need for backtracking procedure from the lower-level to the high-level planner, due to unfeasibility of subplans.

Planning Domain: In this scenario, we have two agents: one passive surface, called p; a dual-arm robotic platform, WRAPP-up, called r; three objects $\mathscr{O} = \{o_1, o_2, o_3\}$; and two sectors, i.e., $\mathscr{S} = \{\sigma_1,\ \sigma_2\}$, as in Fig.3a.

The rigid relations are $\mathbf{adj} = \{(\sigma_1, \sigma_2), (\sigma_2, \sigma_1)\}$, $\mathbf{canAct} = \{(\mathrm{r}, \sigma_1), (\mathrm{r}, \sigma_2)\}$, $\mathbf{canMove} = \{\text{null}\}$, $\mathbf{stable} = \{(o_i, \mathrm{r}), (o_i, \mathrm{p}),\ (o_i, \mathrm{p}, \mathrm{r})\}\, \forall o_i \in \mathscr{O}$. This planning domain corresponds to a state-space of dimension 64.

Planning and Execution: Initially, all the objects are on p in σ_1, corresponding to the state

$$s_0 = \{\text{hold[r]} = \text{null},\ \text{hold[p]} = \mathscr{O},\ \text{onA}[o_1] = \mathrm{p},\ \text{onA}[o_2] = \mathrm{p},\ \text{onA}[o_3] = \mathrm{p},\ \text{at}[o_1] = \sigma_1,\ \text{at}[o_2] = \sigma_1,\ \text{at}[o_3] = \sigma_1,\ \text{at[r]} = \sigma_1,\ \text{at[p]} = \{\sigma_1, \sigma_2\}\}.$$

The set of goal states G_s corresponds to the rearranged configuration of all the objects on p in σ_2: $G_s = \{s \in \text{States} \mid \text{hold[p]} = \mathscr{O} \wedge \text{at}(o_1) = \sigma_2 \wedge \text{at}(o_2) = \sigma_2 \wedge \text{at}(o_3) = \sigma_2 \wedge \text{hold[r]} = \text{null}\}$.

The three search algorithms find the same picking sequence (first pick o_1, then o_2, and eventually o_3), but the found path is different. Indeed, Breadth-first finds a shorter, but more expensive, plan where all the objects are simply moved on the table. Instead, both A* and UCS retrieve a plan where o_2 and o_3 are picked from the table (the pickFromP action), since this is less expensive than moving them on the table due to the dimension (o_2 is taller than larger) and shape (o_3 is a cylinder). Fig. 3b shows the composition of the plan when UCS is used, where the three subplans π^0, π^1, and π^2 - obtained using the procedure described in Sec. III - are highlighted. It can be noted that for the first object, the planner initially derives a subplan π^0 that would require the robot to push it on the table in order to move it to the correct location. However, the presence of the other objects, which are acting as obstacles, does not allow the low-level planner to find a feasible execution of the moveOnP action.

The first subplan and the associated initial and final states are backtracked to the high-level planner to find a different

plan, which requires the robot to pick the object from the table and place it on the correct sector. This plan is again sent to the low-level planner of the robot that, this time, is able to find a feasible execution for the planned actions. A photo sequence of the correct execution of this plan is reported in Fig. 3c.

The total time took by the high-level planner to find a feasible solution and the actual cost of the path after the backtracking are reported in Tab. II between brackets. The plan obtained using Breadth-first was not actually executed, but it would have certainly required at least one backtracking action, as the first subplan is equal for the three algorithms. Hence, we reported the planning time and the cost of the plan also considering this replanning.

Discussion on validation: For tasks (A) and (B), no differences can be noted on the high-level plan, except for different time performances of the three search algorithms. Instead, the plan retrieved by breadth-first for (C) (even after the first replanning) is more expensive than the ones retrieved by UCS and A*. In general, a search algorithm that considers costs might find plans that are indeed longer, but composed of actions that are perhaps less risky and expensive, and thus more robust. This can reduce the risk of failures both in planning and execution, and reduce the calls to the high-level for retrieving new plans. Informed algorithms (such as A*) can be meaningful if a good heuristic function is defined. Otherwise, it might be better to use UCS, as we did.

In all the considered scenarios, the use of different low-level approaches - e.g., changing algorithm in OMPL for (B) - did not produce significant effects or influence on the higher level. Indeed, we did not encounter low-level planning or execution errors that required high-level replanning.

Notwithstanding the above considerations, general conclusions cannot be drawn regarding the choice of certain high- or low-level algorithms over others from the three presented examples. In order to be able to draw general conclusions, it would be necessary to collect statistically meaningful data and analyze them. An extensive experimental campaign could be carried out in a significantly high number of cases, using several more robotic systems, and changing multiple low-level planners and search algorithms.

V. Conclusion

In this paper, we presented an attempt towards an autonomous high-level planning framework for material handling using multi-robot systems. The planning domain contains only the strictly necessary details for finding a sequence of actions to be performed by the robots on the objects in order to move them. The produced high-level plan is easier to be refined by state-of-the-art lower level motion planners. Experiments show the applicability of our approach to real case scenarios. Some aspects, such as a more integrated task and motion planning strategy, and formal guarantees of the high-level algorithm still need further investigation.

References

[1] Nils Boysen, René De Koster, and Felix Weidinger. Warehousing in the e-commerce era: A survey. *Eur. J. Oper. Res.*, 277, 2018.

[2] Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Integrated task and motion planning. *Annual Review of Control, Robotics, and Autonomous Systems*, 4:265–293, 2021.

[3] John Schulman, Yan Duan, Jonathan Ho, Alex Lee, Ibrahim Awwal, Henry Bradlow, Jia Pan, Sachin Patil, Ken Goldberg, and Pieter Abbeel. Motion planning with sequential convex optimization and convex collision checking. *Int. J. Rob. Res.*, 33(9):1251–1270, 2014.

[4] Jennifer Barry, Kaijen Hsiao, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Manipulation with multiple action types. In *Experimental Robotics*, pages 531–545. Springer, 2013.

[5] K. Hauser and V. Ng-Thow-Hing. Randomized multi-modal motion planning for a humanoid robot manipulation task. *Int. J. Rob. Res.*, 30(6):678–698, 2011.

[6] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated planning and acting*. Cambridge University Press, 2016.

[7] Erez Karpas and Daniele Magazzeni. Automated planning for robotics. *Annu. rev. control robot. auton. syst.*, 3(1):417–439, 2020.

[8] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Ffrob: Leveraging symbolic planning for efficient task and motion planning. *Int. J. Rob. Res.*, 37(1):104–136, 2018.

[9] James Motes, Read Sandström, Hannah Lee, Shawna Thomas, and Nancy M Amato. Multi-robot task and motion planning with subtask dependencies. *IEEE Robot. Autom. Lett.*, 5(2):3338–3345, 2020.

[10] Esmaeil Najafi, Anuj Shah, and Gabriel AD Lopes. Robot contact language for manipulation planning. *IEEE/ASME Trans. Mechatron.*, 23(3):1171–1181, 2018.

[11] H. Marino, M. Ferrati, A. Settimi, C. Rosales, and M. Gabiccini. On the problem of moving objects with autonomous robots: A unifying high-level planning approach. *IEEE Robot. Autom. Lett.*, 1(1):469–476, Jan 2016.

[12] F. Ruggiero, V. Lippiello, and B. Siciliano. Nonprehensile dynamic manipulation: A survey. *IEEE Robot. Autom. Lett.*, 3(3):1711–1718, 2018.

[13] George Jose Pollayil, Giorgio Grioli, Manuel Bonilla, and Antonio Bicchi. Planning robotic manipulation with tight environment constraints. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9385–9392, 2021.

[14] Masoumeh Mansouri, Federico Pecora, and Peter Schüller. Combining task and motion planning: Challenges and guidelines. *Front. Robot. AI*, 8:133, 2021.

[15] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/.

[16] Anthony Simeonov, Yilun Du, Beomjoon Kim, Francois R. Hogan, Joshua Tenenbaum, Pulkit Agrawal, and Alberto Rodriguez. A long horizon planning framework for manipulating rigid pointcloud objects. In *Conf. Rob. Learn.*, 2020.

[17] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Sampling-based methods for factored task and motion planning. *Int. J. Rob. Res.*, 37(13-14):1796–1825, 2018.

[18] Manolo Garabini, Danilo Caporale, Vinicio Tincani, Alessandro Palleschi, Chiara Gabellieri, Marco Gugliotta, Alessandro Settimi, Manuel Giuseppe Catalano, Giorgio Grioli, and Lucia Pallottino. WRAPP-up: A dual-arm robot for intralogistics. *IEEE Robot. Autom. Mag.*, 28(3):50–66, 2021.

[19] Alessandro Palleschi, Marco Gugliotta, Chiara Gabellieri, Dinh-Cuong Hoang, Todor Stoyanov, Manolo Garabini, and Lucia Pallottino. Fully autonomous picking with a dual-arm platform for intralogistics. In *2020 I-RIM Conference*, pages 109–111. I-RIM, 2020.

[20] C. Gabellieri, F. Angelini, V. Arapi, A. Palleschi, M. G. Catalano, G. Grioli, L. Pallottino, A. Bicchi, M. Bianchi, and M. Garabini. Grasp it like a pro: Grasp of unknown objects with robotic hands based on skilled human expertise. *IEEE Robot. Autom. Lett.*, 5(2):2808–2815, 2020.

[21] J.J. Kuffner and S.M. LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Int. Conf. Robot. Autom. (ICRA)*, volume 2, pages 995–1001, 2000.

[22] S. Chitta, I. Sucan, and S. Cousins. Moveit![ros topics]. *IEEE Robot. Autom. Mag.*, 19:18–19, 03 2012.

[23] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *Int. J. Rob. Res.*, 30(7):846–894, 2011.