Combined Heuristic Task and Motion Planning for Bi-manual Robots

Aliakbar Akbari · Fabien Lagriffoul · Jan Rosell

Abstract Planning efficiently at task and motion levels allows the setting of new challenges for robotic manipulation problems, like for instance constrained table-top problems for bi-manual robots. In this scope, the appropriate combination of task and motion planning levels plays an important role. Accordingly, a heuristic-based task and motion planning approach is proposed, in which the computation of the heuristic addresses a geometrically relaxed problem, i.e., it only reasons upon objects placements, grasp poses, and inverse kinematics solutions. Motion paths are evaluated lazily, i.e., only after an action has been selected by the heuristic. This reduces the number of calls to the motion planner, while backtracking is reduced because the heuristic captures most of the geometric constraints. The approach has been validated in simulation and on a real robot, with different classes of table-top manipulation problems. Empirical comparison with recent approaches solving similar problems is also reported, showing that the proposed approach results in significant improvement both in terms of planing time and success rate.

Keywords Combined task and motion planning · Robot manipulation · Geometric reasoning · Path planning

This work is partially supported by the Spanish Government through the project DPI2016-80077-R. It is also supported by Swedish Knowledge Foundation (KKS) project "Semantic Robots". Aliakbar Akbari is supported by the Spanish Government through the grant FPI 2015.

Aliakbar Akbari, Jan Rosell
Institute of Industrial and Control Engineering (IOC), Universitat Politècnica de Catalunya (UPC) – Barcelona Tech
E-mail: aliakbar.akbari@upc.edu, jan.rosell@upc.edu

Fabien Lagriffoul
Örebro Universitetet - 70182 Örebro, Fakultetsgatan 1, Örebro
E-mail: Fabien.Lagriffoul@oru.se

1 Introduction

Solving robotic manipulation problems like setting a table with various objects requires a planning system which is capable of finding a complete sequence of actions along with feasible paths. Such planning problems becomes more challenging if the actions required to perform the task are subject to strong geometric constraints from the environment (lack of space for placing objects, occlusions) and the robot (reachability of objects, kinematic constraints of the manipulators). Then, the choice of geometric values like object placements, grasp poses, or inverse kinematic solutions (which may depend on each other) is crucial in order to avoid any dead-end task or unfeasible plan. The problem is even more relevant for bi-manual robots where the different reachability of each arm has to be accounted for.

To address manipulation planning problems, variants of probabilistic roadmaps have been proposed, like the manipulation graph by (Siméon et al, 2004), or multimodal motion planning techniques (Hauser and Latombe, 2010; Hauser et al, 2010). However, roadmap methods do not cope well with the curse of dimensionality inherent to manipulation problems with many objects. Task and Motion Planning (TAMP), which looks for a discrete sequence of symbolic actions along with a motion plan for each of them, is another possible approach. The underlying discrete structure of manipulation tasks (grasps and placements) is explicitly represented in the symbolic domain, which breaks down the complexity of the problem. The main difficulty when using TAMP for constrained manipulation planning problems is to avoid calling the motion planer for unfeasible actions, which is a challenging issue for efficiently solving manipulation problems.

There are two main ways of integrating task and motion planning information recently studied: simultaneously or interleaved. Approaches like (Akbari et al, 2016; Cambon et al, 2009; Garrett et al, 2015) account for geometric information by calling a motion planner *while* task planning

is being pursued. Therefore in these approaches, a manipulation plan is available when the task planning process is terminated. On the contrary, the methods investigated by (Dantam et al, 2016; He et al, 2015; Lagriffoul and Andres, 2016; Srivastava et al, 2014) decouple motion planning from the task planning part. They first do task planning, and then call the motion planner to evaluate the feasibility of the plan. Upon failure, geometric constraints are fed back to the task planner and the procedure resumes. This can be repeated several times until a feasible manipulation plan is found.

The integration of task and motion planning is challenging when the problem is highly constrained in terms of robot kinematics and obstacles' placement, due to the fact that it may require many calls to the motion planner while task planning is being pursued, or a large number of restarts of a task planner which has a high computational cost.

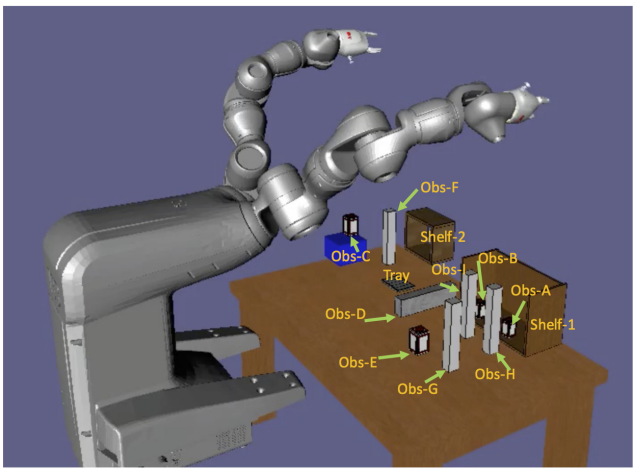

Fig. 1: A motivating example: the Yumi robot must deliver objects A, B, and I to the tray, and object C within the shelf 2 in the presence of kinematics and placement constraints.

1.1 A motivating example

Consider a manipulation problem including the bi-manual Yumi robot and a set of objects as depicted in Fig 1. The task is to place objects A, B, and I on the tray and object C inside shelf 2. Initially, the left arm cannot access objects A, B, I, and H because they are located out of its reachability space. Therefore, both arms have to collaborate with each other to solve the task. Objects A and B are not directly accessible due to the critical position of objects I and H that cause collisions with the robot when their grasp is attempted. In order to allow the robot to transfer the objects from one side of the table to the other, a tiny region is defined in the middle of the table to which both arms can have access. Initially, this region is entirely occupied by object D and we assume that no objects can be stacked on top of it. This situation imposes a constraint on the placement of any objects in this region. Furthermore, only two objects can be placed over the tray at most, due to its limited capacity, and the other ones can be piled on top of each other if needed, i.e., objects A and I can be placed on the tray surface and B on top of A. The order in which the goals are achieved is a critical issue. For instance, if object I is located on the tray at first, there is no feasible kinematic solution for transferring object C within shelf 2 because its entrance would be occluded. Goal ordering is also critical regarding actions including both arms and the middle region.

It must be noted that the aforementioned challenges do not require reasoning upon the details of robot motions to be addressed. Most of them can be detected using lightweight geometric reasoning processes with respect to arms inverse kinematic solutions, collision detection for possible choices of objects placements, performed at initial and final configurations of each action.

1.2 Contributions

The current study proposes a simultaneous TAMP approach to efficiently deal with bi-manual robot manipulation problems in constrained environments. The proposed approach is a heuristic-based planner, which searches for a plan in state space, and takes into consideration geometric constraints while computing heuristic values. Two types of geometric reasoning processes are used: a) geometric reasoning about placements of objects, grasping poses, and inverse kinematic solutions; b) geometric reasoning about motion. The former involves *Spatial, Reachability,* and *Manipulation* reasoning, which are used to account for geometric constraints in heuristic values. The proposed heuristic is able to cover a large range of tabletop manipulation problems. It figures out and excludes unfeasible motion planning queries which have kinematic problems or collisions in their start and goal configurations, and guides state space search. The latter calls a motion planner for validating state transitions and either returns a path or provides feedback in case of failure. The state of planner is updated by the constraints which are detected using both geometric reasoning processes.

2 Related Work

2.1 Task planning approaches

There are different approaches in Artificial Intelligence (AI) like classical planning, logic programming, or constraint satisfaction, which have been used by robotics researchers to devise TAMP planning approaches.

Planning can be done by searching in plan space, like the *GRAPHPLAN* task planner (Blum and Furst, 1997), by

iterative expansion of a *Planning Graph*. A planning graph is a data-structure interleaving state-levels and action-levels. Each state-level consists of the union of atoms (see Def. 1) achievable by the actions in the previous level, and each action-level is the set of actions whose preconditions are present in the previous state-level. The graph is expanded until goal conditions appear in the last state-level, while mutual exclusion relations among actions and states are maintained. A plan is then looked for by backtracking from the last state-level towards the initial state-level taking mutual exclusions into account. If no plan can be found, the graph is expanded and the procedure is repeated.

Alternatively, search can be done in state space. In this line, one of the most efficient task planning approaches is the *FastForward* (FF) (Hoffmann and Nebel, 2001) which performs a heuristic search. This is the task planner used in this paper. It has two main components: the *Enforced Hill Climbing* (EHC) module devoted to select the more promising successor state using the heuristic values, and the Relaxed *GRAPHPLAN* module that computes the heuristic value in terms of the estimated number of actions. This later module also computes the set of helpful actions (i.e. those actions that executed from that state have a high probability of being in the solution plan), which allows making the exploration more efficient. The *Relaxed GRAPHPLAN* module is based on a relaxed version of the *Planning Graph*. The relaxed version of the *Planning Graph* (called RPG) ignores the delete lists of the actions, so mutual exclusion relations do not take effect in the planning phase. From the first state-level at which all the problem goals appear, a backward search is applied that results in the relaxed plan composed of the sequence of cheapest actions connecting the initial state to the final one. The heuristic value is then computed as the number of actions in the relaxed plan, and the helpful actions are those actions of the RPG that appear in the first action level. If EHC fails, everything done so far is skipped and the FF restarts considering *Best-First Search* (BFS) instead of EHC.

Other task planning techniques like hierarchical-based planning recursively decompose tasks into sub-tasks, down to ground actions. Dependency among actions are modeled with so called *Hierarchical Task Network* (HTN) (Ghallab et al, 2004). *Linear Temporal Logic* (LTL), *Satisfiability Modulo Theories* (SMT), and *Answer Set Programming* (ASP) tackle task planning with logic programming. LTL (Clarke et al, 1999) is a formalism used to specify tasks by combining logical propositions and temporal operators (LTL formula), for which models are found using dedicated solvers. SMT (De Moura and Bjørner, 2011) and ASP (Lifschitz, 2002) are constraint-based languages capable of expressing boolean satisfiability problems combined with other theories, e.g., integers. SMT problems are solved with constraint programing, while ASP problems are solved with *Satisfiability* (SAT) solvers.

Usually, task planning domain is represented by the *Planning Domain Definition Language* (PDDL) (Ghallab et al, 1998) whose purpose is to standardize the setup of AI planning problems.

2.2 Motion planning approaches

Motion planning is mostly done in the configuration space ($\mathcal{C}$-space) (Lozano-Perez, 1983). The $\mathcal{C}$-space has as many dimensions as degrees of freedom the robot has, and therefore each point represents a configuration of the robot. The subspace corresponding to collision-free configurations is called $\mathcal{C}_{free}$ and the subspace corresponding to collision configurations is called C_{obs}. Motion planning in $\mathcal{C}$-space consists in finding a path in $\mathcal{C}_{free}$ between two configurations.

Recently, much study is centered in sampling-based motion planning to provide efficient solutions for path planning by avoiding the need to compute the whole $\mathcal{C}$-space. Path planning based on the *Probabilistic Roadmap Method* (PRM) (Kavraki et al, 1996) is done in two phases. The first phase is the construction phase, that spends a specific amount of time sampling $\mathcal{C}_{free}$ and interconnecting samples with simple collision-free paths forming a roadmap. The second phase is the query process, which connects a start configuration to a goal configuration by using graph search techniques. Alternatively, path planning based on the *Rapidly Exploring Random Tree* (RRT) (LaValle and Kuffner, 2001) explores the configuration space by expanding several branches of a tree. In the generic RRT algorithm, a tree is initialized at the root where the initial state is placed and it incrementally grows towards the goal configuration along random directions biased by the less explored areas. The RRT-Connect (Kuffner and LaValle, 2000) is a variant of RRT that has two trees rooted at the start and goal configurations that try to meet each other. This is the motion planner used in this paper.

2.3 Manipulation planning

The limitation of generic motion planning emerges when a robot requires to displace objects when there is no feasible path between two robot configurations due to task constraints. Accordingly, various versions of motion planning have been enhanced and applied for solving a manipulation problem. One is called physics-based motion planning and considers dynamic interactions between rigid bodies. Physics-based motion planning has been applied in manipulation problems where pushing actions are required, for instance (Muhayyuddin et al, 2015). A more general manipulation planning approach using several PRMs has been de-

veloped by (Siméon et al, 2004) that considers multiple possible grasps (that can be used for re-grasping the objects) and stable placements of the movable objects to solve the problem. The manipulation problem of *Navigation Among Movable Obstacles* (NAMO) has been addressed in (Stilman et al, 2007) and (Stilman and Kuffner, 2008) using a backward search algorithm from the goal in order to move the objects occluding the way between two robot configurations. The works in (Hauser et al, 2010) and (Hauser and Latombe, 2010) have investigated multimodal motion planning for the application of climbing robots and push-planning by a humanoid robot, respectively, without consideration of causal reasoning. The work in (Hauser, 2014) deals with the *Minimum Constraint Removal* (MCR) problem while searching for a motion path. The algorithm incrementally grows a PRM for estimating the connectivity of the configuration space, and results in a path that violates the fewest object-collision constraints.

In the related field of grasp planning, Azizi et al. propose a geometric approach based on detecting a complete set of object subsurfaces in a cluttered scene (Azizi et al, 2017), which allows the end-effector to safely approach and grasp the object. In a similar vein, (Hertle and Nebel, 2017) present techniques for sampling appropriate geometric configurations for object placements, grasping poses, or robot positions, in order to perform a specific action. And recently, a method for grasp planning in cluttered environments has been proposed (Muhayyuddin et al, 2018), that uses randomized physics-based motion planning to account for robot-object and object-object interactions. This allows a robot to push obstructing objects away while reaching a target grasp pose.

2.4 Combining task and motion planning approaches

Different studies have dealt with various strategies to combine task and motion planning with the aim of finding a feasible plan to solve a given task.

The work in (Dornhege et al, 2012) introduces *semantic attachments*, which are external reasoners called when the preconditions of actions are evaluated (a motion planner in this case). Other approaches employ a generic interface between symbolic and geometric planning levels to determine whether a collision-free motion for symbolic actions exists or not (Erdem et al, 2011; Srivastava et al, 2014). This is done by calling a motion planner for each symbolic action in the plan obtained by the task planner. In case of failure, obstructing objects are identified and the state of the task planner is updated with no-good constraints referring to these objects. The process is resumed and repeated until a feasible manipulation plan is found. Lagriffoul et al (2012, 2014) introduce the concept of *geometric backtracking*, which denotes the systematic search process in the space of grasps and placements when instantiating a symbolic plan. They use linear constraints generated from symbolic actions and the kinematic model of the robot in order to prune the space of grasps and placements. In all these approaches (except the one by Dornhege et al.), a symbolic plan is computed *first*, and geometric constraints are evaluated *afterwards*. By contrast, our approach evaluates geometric constraints for individual actions *while* task planning is performed. Next, we present approaches in which task planning and geometric reasoning are more tightly intertwined.

Planning Graph-based TAMP. The problem of planning push and pull actions by a mobile robot has been addressed by (Akbari et al, 2015a,b). It combines different knowledge-based task planners with physics-based motion planners. This approach evaluates the feasibility of actions while planning, allowing to cut off unfeasible action branches at the task level (Akbari et al, 2015a). A modified version of *GRAPHPLAN* (Akbari et al, 2015b) was proposed to allow the retrieving of several potential plans which were subsequently evaluated by a physics-based motion planner to find the least-cost feasible one. These approaches can be computationally expensive in terms of number of calls to the motion planner.

FF-based TAMP. (Cambon et al, 2009) present an algorithm which interleaves search at symbolic and geometric levels, where a PRM motion planner calls the task planner to guide roadmap sampling. Upon failure of a selected action, the PRM is left for further exploration (thus keeping the probabilistic completeness of sampling-based motion planners). Guidance is provided by a heuristic value based on the symbolic distance to the goal. On the other hand, the work in (Garrett et al, 2015) proposed an approach, called *FFRob*, which computes the heuristic value by analyzing the feasibility of actions with a *Conditional Reachability Graph* (CRG) based on a modification of PRM planner. It requires a pre-processing step to initialize the CRG by sampling objects poses and robot configurations, and determining conditions under which these samples are reachable or not. The study in (Akbari et al, 2016) proposes the κ*-TMP* for multi mobile robots. The approach calls a physics-based motion planner during heuristic computation only for actions manipulating an object. In the last two approaches, the heuristic function considers geometric information, by calling the motion planner or by evaluating different accessibility queries which can be expensive in constrained environments. To mitigate this drawback, the current study only performs relaxed geometric information in the heuristic computation.

Hierarchical-based TAMP. The work in (de Silva et al, 2013) focuses on a combination based on the HTN planner. It facilitates backtracking at different levels, also including an interleaved backtracking procedure. The work in (Kaelbling and Lozano-Pérez, 2011) has addressed an aggressively hierarchical approach that constrains the abstract

plan steps so that they are serializable (i.e. so that the particular way of performing the first step does not make it impossible to carry out subsequent steps), and handles the integration by operating on detailed, continuous geometric representations.

LTL-based TAMP. The work in (He et al, 2015) applied TAMP using the LTL task planner. Motion planning evaluation launches after a task plan is provided. In the case of failure, task planning input can be updated by a set of constraints in order to find another plan. The authors claim the planner is capable enough in moving away objects that block desired executions without requiring backtracking.

Constraint-based TAMP. The work in (Dantam et al, 2016) proposed the *Iteratively Deepened Task and Motion Planning* method using the SMT. It incrementally detects constraints and keeps dynamically adding or eliminating a number of task constrains based on the feedback obtained from the RRT-Connect motion planner. The approach is able to find an alternative plan when an unfeasible one is identified. It first finds the task plan, and then motion planning is employed to evaluate its feasibility. The work presented by (Lagriffoul and Andres, 2016) addresses TAMP by solving a culprit detection problem. In case of failure at the geometric level, a logical explanation is computed. This explanation is fed back to the ASP task planner, which prunes entire families of plans leading to similar failures. The cycle repeats until a feasible plan is found.

Erdem et al (2016) investigated different integration methods between task and motion planning, focusing on when and how feasibility checks should be performed. Computational benefits and drawbacks of different method are systematically compared. A similar analysis was done by Lagriffoul et al (2013), in which various degrees of postponing feasibility checks are mathematically and empirically compared. Both studies point out the importance of the type of problem considered for choosing an appropriate method, e.g., large task spaces are not amenable to pre-computation or, geometrically constrained problems benefit from tight task-motion integration because it allows early pruning.

3 Overview of the Proposed Task and Motion Planning

The proposed TAMP extends the basic FF planner in order to consider geometric information while planning. The approach uses hybrid planning information for representing the effects of actions in terms of symbolic and geometric information. The architecture of the system is sketched in Fig 2. It consists of three main parts: *Heuristic Computation, State Space Search*, and *Action Selection Process*.

Heuristic Computation returns a heuristic value and a set of helpful actions for a given state. The standard RPG is first constructed and the relaxed plan is extracted. The relaxed plan is then fed into a geometric reasoner which checks it against certain geometric constraints, i.e., reachability, collisions, and graspability. If these constraints are satisfied, the heuristic value is returned along with helpful actions. If a constraint is violated, the state is updated with a set of atoms(s) describing the cause of failure, and an alternative relaxed plan is looked for. Hence the heuristic function is informative both in terms of symbolic and geometric constraints.

State Space Search keeps the standard search procedure of the FF planner, i.e., enforced hill climbing (EHC). From each state, the action resulting in the state with lowest heuristic value is selected. The only difference is that the heuristic value now accounts for geometric constraints.

The *Action Selection Process* attempts to find a motion path for an action. Using information from geometric reasoning, it is possible to define start and goal configurations for the RRT-Connect motion planner, and compute a path. If motion planning fails, the current state is updated with the cause of failure and the search resumes. Otherwise, the action is added to the partial plan at hand.

The rest of the paper is structured as follows. First, Section 4 introduces the planning domains, Section 5 describes how the relaxed geometric reasoning process is accomplished for symbolic actions, and Section 6 illustrates the heuristic computation based on the relaxed geometric reasoning. Then, Section 7 demonstrates the planning part in the state space, Section 8 discusses implementation issues and the results of different manipulation problems, and finally Section 9 concludes the work.

4 Task and Motion Planning Formulation

4.1 Definitions

Definition 1 (Hybrid TAMP Domain) A hybrid TAMP domain $\mathcal{D}$ is a tuple $\langle \mathcal{A}, \mathcal{F}, \mathcal{W}, Acc, S_g \rangle$ where $\mathcal{A}$ is the action space, $\mathcal{F}$ a set of ground atoms, $\mathcal{W} = (\boldsymbol{\mathcal{R}}, \boldsymbol{\mathcal{O}})$ is a workspace including a set of robots $\boldsymbol{\mathcal{R}}$ and a set of movable and fixed objects $\boldsymbol{\mathcal{O}}$. Acc represents potential accessible workspace regions w.r.t. robot arms, S_g is a set of predefined grasping poses for objects. Objects are denoted as:
$\boldsymbol{\mathcal{O}}$=$\{\mathcal{O}_1^m$*(pos,fe)*$\ldots \mathcal{O}_j^m$*(pos,fe),* $\mathcal{O}_1^f$*(pos,fe)*$\ldots \mathcal{O}_k^f$*(pos,fe)*$\}$
where *j* and *k* are the number of *Movable* and *Fixed* objects respectively, whose initial position and orientation is denoted by *pos*, and whose features are denoted by *fe*. In the present case, the bi-manual Yumi robot is represented by $\boldsymbol{\mathcal{R}} = \{LArm, RArm\}$, for left and right arms respectively, and its configuration (a vector of joint angles) is denoted by Q. $Acc = \{Acc_l, Acc_r, Acc_m\}$ represent *predefined* workspace regions which can be kinematically reachable by the left arm, right arm, and both arms, respectively. In order for the task planner to assign reachable targets to

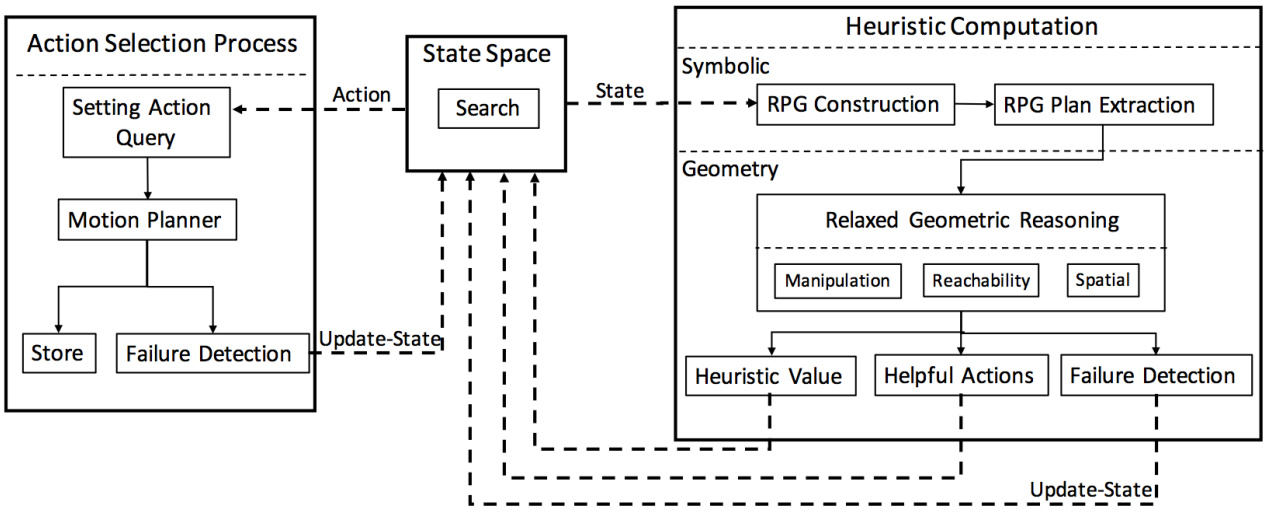

Fig. 2: The proposed system overview of TAMP

each arm, configuration symbols are typed by *ConfRobLeft* or *ConfRobRight*, which map to $Acc_l \cup Acc_m$ or $Acc_r \cup Acc_m$, respectively. In this way, some unfeasible actions are pruned from the search space of planning. For example, if an object is located on the left side of a table which is not reachable by the right arm, task planning applies the move action considering the left arm.

Definition 2 (Hybrid Action) A hybrid action $a \in \mathcal{A}$ is a tuple $\langle$*name(a), pre(a), effect(a), coneffect(a)*, $\mathcal{Q}(a)\rangle$ where:

- *name(a)* is the action symbolic name.
- *pre(a)* is a set of atoms which must hold for the action to be applied. It includes:
  - *geometric details*: *geom(a)* is the component and numerical counterparts of *pre(a)*. It represents geometric values in $\mathcal{W}$, i.e., how object poses and/or robot configuration should be before executing the action. This information is important for geometric reasoning and setting a motion planning query.
- *effect(a)* represents the effects of a on the state it is applied to. It consists of:
  - *effect*$^+$*(a)*, positive effects, i.e., a set of atoms added to the current state;
  - *effect*$^-$*(a)*, negative effects, i.e., a set of atoms deleted from the current state.
  - *geometric details*: *geom*$^+$*(a)* and *geom*$^-$*(a)* are the components and numerical counterparts of *effect*$^+$*(a)* and *effect*$^-$*(a)* computed during geometric reasoning. They represent changes in $\mathcal{W}$, i.e., how object poses and/or robot configuration are modified by executing the action. For instance, when an object is transferred to a new location, the new pose is represented in *geom*$^+$*(a)*. A data-structure is so considered to store the geometric information such as grasping pose, object pose placement, and robot configurations.
- *coneffect(a)* is the set of conditional effects, i.e., pairs $\langle con(a)$, *effect(a)*$\rangle$ such that *effect(a)* is applied when the formula $con(a)$ holds.
- $\mathcal{Q}(a)$ is a query function to the motion planner which computes a motion between two robot configurations and stores the solution if any.

Definition 3 (Hybrid Relaxed Action) A hybrid relaxed action $a' \in \mathcal{A}'$ (where $\mathcal{A}'$ denotes the relaxed action space) is a tuple $\langle$*name*(a'), *pre*(a'), *effect*(a'), *coneffect*(a'), $\emptyset\rangle$ where *effect*(a') contains only *effect*$^+(a')$ and *geom*$^+(a')$. In other words, a hybrid relaxed action does not incorporate any negative effect. A hybrid relaxed domain $\mathcal{D}'$ is a tuple $\langle \mathcal{A}', \mathcal{F}, \mathcal{W}, Acc, S_g \rangle$.

Definition 4 (Hybrid State) A hybrid state S is a tuple $S = \langle \mathcal{P}, \mathcal{V} \rangle$ where $\mathcal{P}$ is a set of ground atoms which hold in that state, and $\mathcal{V}$ represents a full geometric description of the scene, i.e., configurations of robots and poses of objects. Applying action a to state S_1 results in state S_2 as follows:

$$\begin{aligned} S_2.\mathcal{P} &= \{(S_1.\mathcal{P} \cup \mathit{effect}^+(a)) \backslash \mathit{effect}^-(a)\} \\ S_2.\mathcal{V} &= \{(S_1.\mathcal{V} \cup \mathit{geom}^+(a)) \backslash \mathit{geom}^-(a)\} \end{aligned} \quad (1)$$

The hybrid state is required to represent the effects of actions in terms of symbolic and geometric information. A data-structure for storing geometric information is considered that is needed to compute heuristic values (see Sec. 6). The data-structure of a state includes information like all poses of the fixed and manipulatable objects along the configurations of both robotic arms.

Definition 5 (Hybrid Relaxed State) A hybrid relaxed state is a tuple $\langle \mathcal{P}', \mathcal{V}' \rangle$ where $\mathcal{P}'$ is a set of ground atoms which hold in that state, $\mathcal{V}'$ represents a full geometric description of the scene. $\mathcal{P}'$ and $\mathcal{V}'$ are the result of applying a *relaxed* action (see Def. 3). Hence, since no negative effects are considered, applying the relaxed action a' to state S'_1 results in state S'_2 as follows:

$$\begin{aligned} S'_2.\mathcal{P}' &= \{S'_1.\mathcal{P}' \cup effect^+(a')\} \\ S'_2.\mathcal{V}' &= \{S'_1.\mathcal{V}' \cup geom^+(a')\} \end{aligned} \tag{2}$$

Accordingly, there can be many relaxed worlds depending on the number of geometric details recorded for each symbol related to objects' placements and robot configurations, as symbols can have different values at the same time. For instance, if there is one movable object which is initially grasped by the robot and it is transferred to a new position (by a relaxed action), the object will have two geometric placements values, i.e., its initial position and the transferred position, and the robot will also have two geometric configuration values in this state. The following two possible worlds exist for the example:

- The object is in the initial position and the robot is in the initial configuration.
- The object is in the final position and the robot is in the final configuration.

This concept allows the planner to evaluate all possible situations when requested.

Definition 6 (Hybrid TAMP Planning Problem) A hybrid TAMP planning problem $\mathcal{T}$ is represented by a tuple $\langle \mathcal{D}, \mathcal{S}_0, \mathcal{G} \rangle$ where $\mathcal{D}$ is a hybrid domain, $\mathcal{S}_0$ consists of a set of ground atoms representing the initial symbolic state $\mathcal{I}$ such that $\mathcal{I} \subseteq \mathcal{F}$ along their geometric assignments regarding the initial state of the world $\mathcal{W}_0$, and $\mathcal{G} \subseteq \mathcal{F}$ is the set of symbolic goal conditions. The solution of a TAMP problem is a hybrid plan, i.e., a sequence of symbolic actions achieving $\mathcal{G}$, along with a feasible motion for each action, which we denote by π.

Definition 7 (Hybrid Relaxed Planning Problem) A hybrid relaxed planning problem $\mathcal{T}'$ is described as a tuple $\langle \mathcal{D}', \mathcal{S}_i, \mathcal{G} \rangle$ where $\mathcal{D}'$ is the hybrid relaxed domain, and $\mathcal{S}_i$ is the current state from which the relaxed plan is computed. We denote the relaxed plan by π'.

4.2 Manipulation Action Templates

The following action templates and all the symbols (like those considered for the reachability of robot arms) to solve the pick and place manipulation tasks are defined using PDDL. Note that PDDL can use ADL (Action Description Language, Pednault (1989)) which allows us to write operators in a more compact way, using quantifiers and conditional effects. Geometric details of symbolic actions conditions (such as object placements or robot grasp configurations) are not pre-computed. They are sampled and assigned on demand during the planning process.

The action template *Move(Rob, Q, Q′)* is designed to move the robot arm *Rob* between configurations *Q* and *Q′* without holding any object. The action is applicable if the following preconditions hold: the robot arm is located at *Q*, its hand is empty, it can reach configuration *Q′* and no movable objects are blocking its way to *Q′*. The last precondition is represented by fact *isCrit*($\mathcal{O}_j^m$, *Q′*); objects that make this fact to hold are called *Critical Objects*. As a result of the action, conditional effects (specified by *when*), whose purpose is to classify the symbolic configuration space for the right arm *RArm* and the left arm *LArm*, are specified. When the move action is applied to *RArm*, *Q′* belongs to the symbolic configurations considered for the right arm, *ConfRobRight*. When the move action is applied to *LArm*, *Q′* belongs to symbolic configurations considered for the left arm, *ConfRobLeft*.

***Move**(Rob, Q, Q′):*

Pre: *At(Rob, Q), HandEmpty(Rob), ∼UnReach(Rob, Q′), ∀*$\mathcal{O}_j^m$ *∼isCrit(*$\mathcal{O}_j^m$*, Q′)*
Effect: *when (con: Rob = RArm) ⇒ {At(Rob, Q′), Q′ ∈ ConfRobRight}, when (con: Rob = LArm) ⇒ {At(Rob, Q′), Q′ ∈ ConfRobLeft}, ∼At(Rob, Q)*

The action template *MoveHolding(Rob,* $\mathcal{O}_i^m$*, Q, Q′, Pos, Pos′)* is designed to move the robot *Rob* between configurations *Q* and *Q′* while holding object $\mathcal{O}_i^m$, changing its pose from *Pos* to *Pos′* if *Pos′* is feasible. The action uses conditional effects like those of the move action.

***MoveHolding**(Rob,* $\mathcal{O}_i^m$*, Q, Q′, Pos, Pos′):*

Pre: *At(Rob, Q), Holding(Rob,* $\mathcal{O}_i^m$*, Q, Pos), ∼Infeas(Pos′,* $\mathcal{O}_i^m$*), ∼UnReach(Rob, Q′), ∀*$\mathcal{O}_j^m$ *∼isCrit(*$\mathcal{O}_j^m$*, Q′)*
Effect: *when (con: Rob = RArm) ⇒ {At(Rob, Q′), Q′ ∈ ConfRobRight}, when (con: Rob = LArm) ⇒ {At(Rob, Q′), Q′ ∈ ConfRobLeft}, Holding(Rob,* $\mathcal{O}_i^m$*, Q′, Pos′), ∼At(Rob, Q), ∼Holding(Rob,* $\mathcal{O}_i^m$*, Q, Pos)*

The action template *PickUp(Rob,* $\mathcal{O}_i^m$*, Q, Pos)* is described to pick $\mathcal{O}_i^m$ by *Rob*. It is responsible to attach $\mathcal{O}_i^m$ to the robot gripper when the robot is located at *Q* with no attached object, and $\mathcal{O}_i^m$ lays on a position *Pos* on the surface and its top side is free.

***PickUp**(Rob,* $\mathcal{O}_i^m$*, Q, Pos):*

Pre: *At(Rob, Q), HandEmpty(Rob), Clear(*$\mathcal{O}_i^m$*), On-surface(*$\mathcal{O}_i^m$*, Pos, Q)*
Effect: *Holding(Rob,* $\mathcal{O}_i^m$*, Q, Pos), ∼HandEmpty(Rob), ∼On-surface(*$\mathcal{O}_i^m$*, Pos, Q)*

The action template *PutDown(Rob,* $\mathcal{O}_i^m$*, Q, Pos)* is described to put down $\mathcal{O}_i^m$ over a surface at *Pos* by detaching

the object from the robot when *Rob* holds $\mathcal{O}_i^m$ in Q. The action is also responsible to negate *isCrit* facts for the associated $\mathcal{O}_i^m$ if any.

PutDown*(Rob, $\mathcal{O}_i^m$, Q, Pos):*

Pre: *Holding(Rob, $\mathcal{O}_i^m$, Q, Pos)*
Effect: *HandEmpty(Rob), Clear($\mathcal{O}_i^m$), On-surface($\mathcal{O}_i^m$, Pos, Q), $\sim$Holding(Rob, $\mathcal{O}_i^m$, Q), $\forall Q' \sim$isCrit($\mathcal{O}_i^m$, Q')*

The action template *Stack(Rob, $\mathcal{O}_i^m$, $\mathcal{O}_j^m$, Q, Pos)* is used to stack $\mathcal{O}_i^m$ on the top of $\mathcal{O}_j^m$. It is applicable when *Rob* is holding $\mathcal{O}_i^m$ and the top side of $\mathcal{O}_j^m$ where $\mathcal{O}_i^m$ is going to be located is free.

Stack*(Rob, $\mathcal{O}_i^m$, $\mathcal{O}_j^m$, Q, Pos):*

Pre: *Clear($\mathcal{O}_j^m$), Holding(Rob, $\mathcal{O}_i^m$, Q, Pos)*
Effect: *HandEmpty(Rob), Clear($\mathcal{O}_i^m$), On($\mathcal{O}_i^m$, $\mathcal{O}_j^m$, Pos), $\sim$Clear($\mathcal{O}_j^m$), $\forall Q' \sim$isCrit($\mathcal{O}_i^m$, Q')*

Finally, the action template *UnStack(Rob, $\mathcal{O}_i^m$, $\mathcal{O}_j^m$, Q, Pos)* is applied to pick a pilled object $\mathcal{O}_i^m$.

UnStack*(Rob, $\mathcal{O}_i^m$, $\mathcal{O}_j^m$, Q, Pos):*

Pre: *At(Rob, Q), HandEmpty(Rob), Clear($\mathcal{O}_i^m$), On($\mathcal{O}_i^m$, $\mathcal{O}_j^m$, Pos)*
Effect: *Clear($\mathcal{O}_j^m$), Holding(Rob, $\mathcal{O}_i^m$, Q, Pos),* $\sim$HandEmpty(Rob), $\sim$On($\mathcal{O}_i^m$, $\mathcal{O}_j^m$, Pos), $\sim$Clear($\mathcal{O}_i^m$)

5 Relaxed Geometric Reasoning

Relaxed geometric reasoning consists in evaluating geometric conditions of actions without calling a motion planner. It indicates that motion planning is likely to be feasible for the selected actions if certain task constraints are met. The reasoning process involves three modules: *reachability reasoning* which looks for a feasible kinematic solution for the robot regardless of any attached object, *spatial reasoning* which determines a valid pose for the manipulated object, and *manipulation reasoning* which determines if grasping is possible. They are described in detail next.

Reachability reasoning ($\mathcal{R}_{rch}$): To move the robot to a location described by a set of potential pre-grasping poses, it is first required to find a valid goal configuration. This is done by calling an Inverse Kinematic (IK) solver for each potential pre-grasping pose and evaluating whether the IK solution is collision-free or not. The reasoning procedure returns the first collision-free IK solution found, if any, and the corresponding pre-grasping pose. Failure may occur if no IK solution exists or if no collision-free IK solution exists. In this latter case, the reasoning procedure reports the collisionable objects, ordered such that those that are manipulatable are returned first.

Spatial reasoning ($\mathcal{R}_{sp}$): It is used to determine a proper placement for an object within a given region without considering the robot. For the desired object, a pose is sampled such that the object lies in the surface region and the initial stable posture is maintained. The sampled object pose is then evaluated by a collision-checking module to determine whether it collides with other objects. If the sampled pose is feasible, it is recorded in the geometry details of the action which moves the object. Otherwise, another sample will be tried. If all attempted samples are unfeasible, the reasoner reports failure and the collisionable objects are returned. Moreover, some other constraints are taken into consideration while the sample placement is accomplished with respect to the object features or final placement region limitation, e.g., big or heavy objects are not allowed to be located on the top of small or light objects.

Manipulation reasoning ($\mathcal{R}_{mnp}$): It is used to choose the grasp to be used to transfer an object from its current placement to a valid goal placement (determined by $\mathcal{R}_{sp}$). From a set of grasps, the reasoning process uses $\mathcal{R}_{rch}$ reasoning to verify that one of the possible ways to grasp the object has a collision-free IK solution both at the current and the goal placements, and it is returned. If there is no grasp satisfying these constraints and the reason is due to collisions and not to the IK problem, then the collisionable objects are returned. In this work, prismatic objects are used with the predefined top and side grasps, the latter at different heights according to the object height.

For instance in Fig 1, it is supposed that the robot is going to place object C within the shelf 2. If the object is initially grasped from the top, there will be no valid configuration for the robot to locate the object due to collision with the shelf. On the contrary, if it grasps the object from the side, it can find a feasible configuration to position the object on the surface. Therefore, the reasoning process leads to avoiding or reducing re-grasping operations.

Algorithm 1 describes the relaxed geometric function when applying either the *Move* or *MoveHolding* action to a given world $\mathcal{W}$ (describing the pose of the objects and the configuration of the robot). Upon success, the positive geometry details of the action are updated with the robot configuration and the gripper pose. For the *MoveHolding* action, the pose of the object is also added. Otherwise, the algorithm returns in $\mathcal{CO}$ the set of objects whose collision causes the failure of the action. This geometric reasoning is not required for other manipulation actions as they only attach or detach the object to/from the gripper. The evaluation of the *Move* and *MoveHolding* actions is done as follows:

- Reasoning about the *Move* action [lines 5-8]: the process is done by the function $\mathcal{R}_{rch}(\mathcal{W}, a)$ [line 6], which searches for a feasible robot configuration and store it in Q_{rob} if it finds one. In this case, Res is set to $True$ and the geometric information is stored in the geometric details of the action [lines 6-8]. Otherwise, it is set to $False$ and the $\mathcal{CO}$ causing the failure, is returned if any.
- Reasoning about the *MoveHolding* action [lines 9-16]: the action is appraised by the functions $\mathcal{R}_{sp}(\mathcal{W}, a)$ and $\mathcal{R}_{mnp}(\mathcal{W}, a, \mathcal{O}_i^m(pos_{goal}))$. The function $\mathcal{R}_{sp}(\mathcal{W}, a)$ searches

for a feasible object placement, sets Res_{sp} to $True$ if found, and returns $\mathcal{O}_j^m(pos_{goal})$. Upon failure, it returns $False$ and $C\mathcal{O}$ may be returned [line 11]. Then, the function $\mathcal{R}_{mnp}(\mathcal{W}, a, \mathcal{O}_i^m(pos_{goal}))$ looks for a feasible goal configuration and stores it in Q_{rob} if found. Otherwise, potential objects causing failure are returned in $C\mathcal{O}$.

Algorithm 1: $RelaxGeomReas(\mathcal{W}, a)$

1 $C\mathcal{O} \leftarrow \emptyset$
2 $a.geom^+ \leftarrow \emptyset$
3 $i \leftarrow 0$
4 $Res = False$
5 **if** a.name $=$ Move **then**
6 $\quad \{Res, Q_{rob}, C\mathcal{O}, g\} = \mathcal{R}_{rch}(\mathcal{W}, a)$
7 $\quad$ **if** $Res = True$ **then**
8 $\quad\quad a.geom^+.add(Q_{rob}, g)$
9 **else if** a.name $=$ MoveHolding **then**
10 $\quad$ **while** $i < Max$ **do**
11 $\quad\quad \{Res_{sp}, \mathcal{O}_j^m(pos_{goal}), C\mathcal{O}\} = \mathcal{R}_{sp}(\mathcal{W}, a)$
12 $\quad\quad$ **if** $Res_{sp} =$ True **then**
13 $\quad\quad\quad \{Res, Q_{rob}, C\mathcal{O}, g\}$
$\quad\quad\quad = \mathcal{R}_{mnp}(\mathcal{W}, a, \mathcal{O}_i^m(pos_{goal}))$
14 $\quad\quad\quad$ **if** $Res =$ True **then**
15 $\quad\quad\quad\quad a.geom^+.add(Q_{rob}, \mathcal{O}_j^m(pos_{goal}), g)$
16 $\quad\quad\quad\quad$ break
17 **else**
18 $\quad a$ is not required to be evaluated;
19 $\quad Res = True$
20 **return** $\{a.geom^+, Res, C\mathcal{O}\}$

6 Heuristic Computation using Relaxed information

Both the heuristic value and helpful actions are computed using relaxed symbolic information and relaxed geometric reasoning. The purpose of using relaxed information is to find a relaxed plan satisfying actions' conditions in terms of robot kinematics and object placement. Algorithm 2 illustrates (for a given state S and goal $\mathcal{G}$) how the computation of the heuristic and helpful actions of the standard FF is modified to include geometric information. This is done in three steps: computing the RPG and the relaxed plan π', evaluating π', and computing the heuristic value along the helpful actions, as detailed next.

Computing the RPG and π' [lines 1-2]: At the beginning, the RPG graph RPG_{gr} containing state layers and action layers is constructed by the function RPGConst [line 1]. The function RPGPlan extracts π' from that graph [line 2]. This process is done as the standard FF carries out.

Evaluating π' [lines 3-17]: The relaxed state is initiated with the current geometric state of the world $S.\mathcal{W}$ and the

Algorithm 2: $RPG(S, \mathcal{G})$

1 $RPG_{\text{gr}} \leftarrow$ RPGConst($\mathcal{G}$)
2 $\pi' \leftarrow$ RPGPlan(RPG_{gr})
3 $S'.\mathcal{W}'_0 = S.\mathcal{W}$ and $S'.\mathcal{F}' = S.\mathcal{F}$
4 **foreach** $\{a' \in \pi'\}$ **do**
5 $\quad$ **while** *True* **do**
6 $\quad\quad \mathcal{W}' =$ SetRelaxWorld(a')
7 $\quad\quad$ **if** $\mathcal{W}' \neq$ *NULL* **then**
8 $\quad\quad\quad \{a'.geom^+, Res, C\mathcal{O}\} \leftarrow$RelaxGeomReas($\mathcal{W}', a'$)
9 $\quad\quad\quad$ **if** $Res = True$ **then**
10 $\quad\quad\quad\quad S' \leftarrow$NextRelaxState($a'$)
11 $\quad\quad\quad\quad$ break
12 $\quad\quad$ **else**
13 $\quad\quad\quad$ **if** MaxUpdates(S) $<$ *Max* **then**
14 $\quad\quad\quad\quad S \leftarrow$ UpdateState($S, C\mathcal{O}$)
15 $\quad\quad\quad\quad$ **return** RPG($S, \mathcal{G}$)
16 $\quad\quad\quad$ **else**
17 $\quad\quad\quad\quad$ **return** $\{\infty, \emptyset\}$
18 $h(S) \leftarrow$ HeuristicValue()
19 $H(S) \leftarrow$ HelpfulActions()
20 **return** $\{h, H(S)\}$

symbolic information of the state $S.\mathcal{F}$ [line 3]. Actions appearing in π' are forwarded to the relaxed geometric reasoning for the feasibility check [line 8]. Upon success, action a' is applied to compute the new relaxed state [line 10] according to Eq.(2). The key question is in which relaxed world the corresponding action has to be evaluated because multiple copies of poses and configurations for objects and robot may exist. This is tackled in the function SetRelaxWorld [line 6] which chooses one of the feasible relaxed worlds, i.e., with no collisions and meeting the precondition of the associated relaxed action. At each successive call, the function returns a different feasible relaxed world and returns *NULL* if no more exist. In this latter case, the function MaxUpdates [line 13] evaluates whether a predefined maximum number of trials to update the current state and find another RPG plan is reached or not. A new relaxed plan is then required [line 15] after updating the current state by the function UpdateState [line 14] according to the feedback of the geometric reasoner: in the case of finding any $C\mathcal{O}$, the ground atom *isCrit($C\mathcal{O}$, Q')* is added to the current state. Otherwise, the ground atom *UnReach(Rob, Q')* is added to the state regarding the corresponding arm, and also *Infeas(Pos', $\mathcal{O}_i^m$)* is inserted if the evaluated action is of type *MoveHolding*. Two examples are given below to illustrate how task constraints (like critical objects) and action details are determined.

Computing The heuristic value along with helpful actions [lines 18-19]: After the relaxed plan becomes feasible in terms of lightweight geometry information, the heuristic value and the set of helpful actions of the current state are returned. The function HeuristicValue returns the heuristic value $h(S)$ [line 18] and the function HelpfulActions ex-

tracts helpful actions $H(S)$ [line 19]. This process is also performed in a similar way to the standard FF.

Example 1: To illustrate the heuristic computation using relaxed information, consider the scene in Fig 1 and let the goal be to transfer object A to the middle region of the table that it is occupied by object D. From the initial state of planning, the following relaxed plan is first extracted:

$\pi'_0 = \{$*MoveA, PickUpA, MoveHoldingA, PutDownA*$\} \implies h(S_{init}) = 4$

The selected actions are then forwarded to the relaxed geometric reasoning check. The action *MoveA* is first evaluated using $\mathcal{R}_{rch}$. The reasoner tries to acquire a feasible Q in order to grasp object A. All attempted samples have collisions with object H and there is not any other relaxed world such that object H is in another position in order to find a feasible grasping pose of object A. Then, the constraint *isCrit(*H*,* Q_a*)* is asserted to the initial state and the heuristic computation is restarted again and the following plan is retrieved:

$\pi'_1 = \{$*MoveH, PickUpH, MoveHoldingH, PutDownH,* $\pi'_0\} \implies h(S_{init}) = 8$

As it can be seen, the heuristic value is increased. The feasibility of actions are investigated. The reasoning process is able to find geometric details for the actions *Move* and *MoveHolding* applied to object H. But, the spatial reasoning process is not able to find any valid placement for object A when the action *MoveHoldingA* is being evaluated due to collisions with the object D and there is not any relaxed world where this object is located away from this region. Therefore, the constraint *isCrit(*D, Q_a*)* is added to the state. Similarly, the heuristic computation is repeated and the following relaxed plan is obtained:

$\pi'_2 = \{$*MoveD, PickUpD, MoveHoldingD, PutDownD,* $\pi'_1\} \implies h(S_{init}) = 12$

When this relaxed plan is forwarded to the reasoning process, it can find feasible geometric details for all the relaxed actions. So, the corresponding heuristic value and helpful actions are achieved. In this way, the integration of the relaxed reasoning process with the heuristic function leads to provide the informed heuristic value taking into account task constraints.

Example 2: Another advantage of considering geometric evaluation in the heuristic computation phase is to report failure in the case of selecting invalid geometric values for the actions. This case avoids the consideration of geometric backtracking which goes to the previous states in order to reselect their geometric values. Let's consider the example represented in Fig 3.

The goal is to move the right arm to a feasible grasping configuration of object A. The left part of the scene shows the initial state and the right side presents one feasible manipulation relaxed world after applying the *MoveHolding* relaxed action. The symbolic relaxed solution plan taking into account geometric constraints is shown as well. The action locates the object in pos_{b2}. When the geometric information of the last relaxed action *MoveRA* is going to be evaluated, it figures out there is no feasible world making this action feasible either with object B placed in position pos_{b1} or pos_{b2}. The reasoning process so reports the collisionable object B, and again the heuristic computation is restarted to find a feasible geometric assignment for locating object B.

7 Planning in the state space

A manipulation plan is found using state space search as in the classical FF algorithm. The difference lies in the heuristic function which includes geometric reasoning, and the fact that action selection must be confirmed by the motion planner. The pseudocode of this process is presented in Algorithm 3. It is worth noting that there is no pre-processing step assigning geometric details (object placements or robot grasp configurations) to the symbols: geometric details are resolved during relaxed geometric reasoning. It gets $\mathcal{T}$ as input and returns π if applicable. First, the trials counter $trial$ is initialized [line 1] and the state S_i gets the initial state [line 4]. The function Search implements the standard FF search process using EHC or BFS [line 6]: it returns the next hybrid state (see Eq.(1)) to visit S_{i+1} along with helpful action(s) or action(s) with lowest heuristic value H_{S_i}. This value is computed with the modified RPG function (see Algorithm 2), hence taking geometric constraints into account.

If no actions resulting in a state with lower heuristic value [line 7] can be found, the algorithm tries the whole process again by taking into account previous trials. As long as the maximum number of iterations is not reached [line 9], the process is repeated with the initial state updated by the function UpdateInitState [line 11]. This function samples random geometric states while taking all the constraints identified so far into account. If the maximum number of trials is reached, the process returns failure [line 14].

With an arbitrarily high value for Max, the probability that all possible object poses and robot configurations be considered gets close to 1. However, although Search [line 6] is complete and MotionPlanner [line 16] is probabilistically complete, overall completeness is lost since motion planning runs with a cut-off time, and it is never queued for further refinement in case of failure [lines 20-21].

The MotionPlanner function is used to compute a collision-free path for the currently selected action(s) [line 16]. If a path is found, Res is set to $True$ and the path $\mathcal{Q}$ is returned. Then π is appended with the action(s) [line 18]. Otherwise, Res is false and $\mathcal{CO}$ may contain colliding object(s). If collision(s) are detected, the state is updated with ground atom(s) *isCrit(*$\mathcal{CO}$*,* $\mathcal{Q}'$*)*, otherwise simply with the ground atom *UnReach(Rob,* $\mathcal{Q}'$*)*, and the algorithm keeps searching until other H_{S_i} are considered.

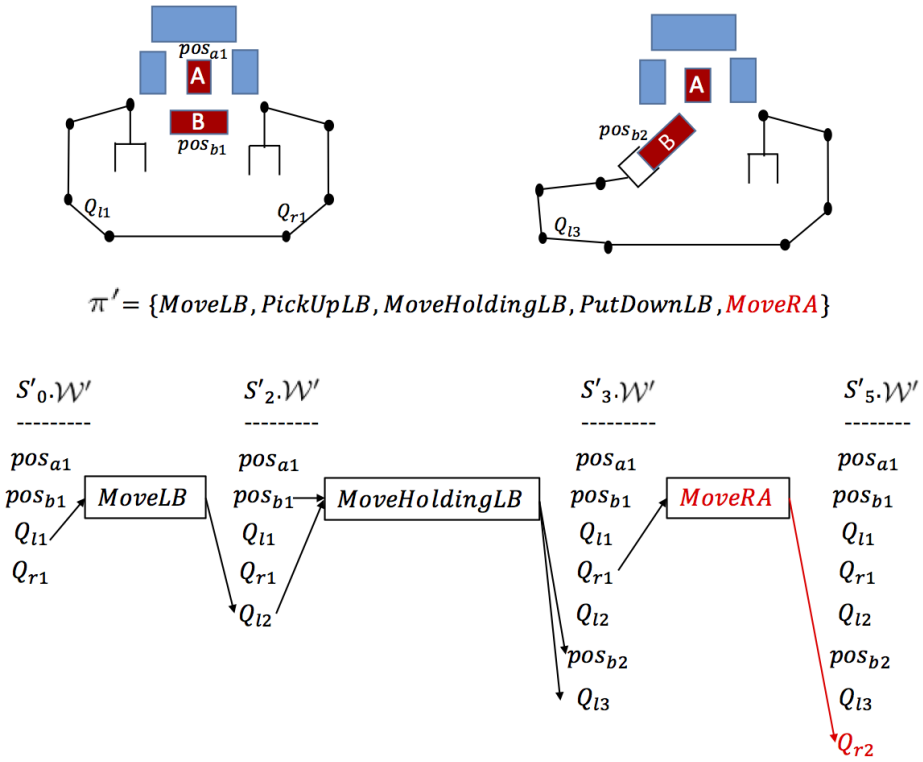

Fig. 3: Example of reporting an unfeasible geometric value for the placement of an object. Blue objects are fixed and the red ones are manipulatable and the labels in the scene involve the geometric details of objects. The characters *L* and *R* show the action applied to the left and right arms respectively. The goal is to transfer the right arm to object A. The successive relaxed worlds that result from relaxed actions in π' that add geometric details are shown.

8 Evaluation

8.1 Implementation

The proposed framework implementation consists of three components: task planning, relaxed geometric reasoning, and motion planning. Task planning is implemented using a modified version of the FF planner developed in C++. Relaxed geometric reasoning and motion planning use the Kautham Project[1] (Rosell et al, 2014), a C++ based open-source tool for motion planning, that enables to plan under geometric and kinodynamic constraints. It uses the Open Motion Planning Library (OMPL) (Sucan et al, 2012) as a core set of sampling-based planning algorithms. In this work, the RRT-Connect motion planner is used for motion planning. This planner is one of the most efficient motion planners, but it does not guarantee optimal motions. The Kautham Project involves different collision checking modules to detect robot-object and object-object collisions, and features a placement sampling mechanism to find feasible object poses in the workspace. For IK computations, it uses the approach developed by (Zaplana et al, 2018). Relaxed geometric reasoning uses these modules in order to find feasible sample geometric instances for symbolic actions. The communication between task, relaxed geometric reasoning, and motion planning modules is done via Robotic Operating System (ROS) (Quigley et al, 2009).

[1] https://sir.upc.edu/projects/kautham/

8.2 Empirical Results and Discussion

All experiments were run on an Intel Core i7-4790U 4.00 GHz CPU machine with 32 GB memory. The platform used is the two-arm Yumi robot (with 7 degree-of-freedoms per arm and two finger grippers). Two different classes of table-top manipulation problems were used for validation: constrained problems, and cluttered table manipulation problems in which the object to be grasped is surrounded by many other objects.

Heuristic computation was made more efficient by caching the computation of geometric details. As there are multiple calls to the heuristic function from each state, geometric values can be stored and reused for similar relaxed actions in different relaxed plans.

Algorithm 3: The Proposed TAMP Algorithm

inputs: $\mathcal{T}$=$\langle \mathcal{D}, \mathcal{S}_0, \mathcal{G} \rangle$, $\mathcal{D}$=$\langle \mathcal{A}, \mathcal{F}, \mathcal{W}, Acc, S_g \rangle$
output: π

$trial \leftarrow 0$
$i \leftarrow 0$
$\pi \leftarrow \emptyset$
$S_{\text{i}} \leftarrow S_{\text{init}}$
while $\mathcal{G} \not\subseteq S_i$ **do**
  $\{H_{S_{\text{i}}}, S_{i+1}\} \leftarrow$ Search$(S_i, \mathcal{G}, \mathcal{A}))$
  if $H_{S_i} = \emptyset$ **then**
    $trial \leftarrow trial + 1$
    if $trial < Max$ **then**
      $i \leftarrow 0$
      $S_i \leftarrow$ UpdateInitState()
      Continue
    else
      return fail
  else
    $\{\mathcal{Q}, Res, C\mathcal{O}\}$ = MotionPlanner$(H_{S_{\text{i}}})$
    if $Res = True$ **then**
      π.append$(H_{S_{\text{i}}})$
    else
      $S_i \leftarrow$ UpdateState$(C\mathcal{O})$
      Continue
  $i \leftarrow i + 1$
return π

Regarding the first class of manipulation problems, the scenario explained in Fig 1 has been implemented and validated in simulation. The problem consists of various types of objects and is non-monotonic, i.e., target objects have to be temporarily placed in regions which are not specified as goal regions, and occluding objects need to be moved out of their initial pose in order to free space in these temporary locations. Some snapshots of the execution are displayed in Fig 4. Detecting various types of task constraints, the appropriate geometric details, as well as the correct order of actions play an important role in this manipulation problem, which is mostly done in heuristic computation. The performance of the problem is illustrated in Fig 5 in terms of average time of total manipulation planning and motion planning, and also the reachability, spatial, and manipulation reasoning. The solution task plan is composed of 33 actions and the success rate is 96%. The parameters used are the following: the number of pre-grasping poses per object used for reachability reasoning is 20; the number of sample poses for spatial reasoning is 20; the *Max* threshold is 20 in algorithms 1 and 2, and 5 in algorithm 3.

Regarding the second class of problems, several scenes containing from 15 to 40 objects have been used and compared with the approach presented in (Srivastava et al, 2014) and (Muhayyuddin et al, 2018). In order to make the problems more challenging, only side-grasps are used to pick up objects. The clutter table problem for 15 objects is represented in Fig 6 which has been evaluated in simulation and in the real environment. In this type of problems, it has been observed that both robot arms can collaborate to free the path towards the target object, like the problem shown in Fig 7. The clutter table problem where the goal is to hold the red object surrounded by 40 objects is displayed in Fig 8. This problem is in essence similar to the problems used in (Srivastava et al, 2014) and (Muhayyuddin et al, 2018).

Table 1 summarizes the comparison between the current proposal (HTAMP) and the work in (Srivastava et al, 2014) (labeled here as Ap1) and the work in (Muhayyuddin et al, 2018) (labeled as Ap2) in terms of success rate and planning time. We observed that the HTAMP approach has efficiently solved the clutter table problems in both terms as compared with the Ap1 approach. One of the main reasons is that HTAMP detects the task constraints in the heuristic computation phase and selects carefully the objects placements in advance of calling motion planning. On the contrary, Ap1 does first symbolic task planning, calling a motion planner that feeds back constraint to the state. Moreover, Ap1 may require to manipulate the objects more than once. The average time of the reachability, spatial, and manipulation reasoning, and also motion planning for this type of problem is shown in Table 2. As it can be seen, the relaxed geometric reasoning time increases according to the difficulty of the problem when the number of objects increases. This is due to the fact that the relaxed geometric process requires more effort to find valid geometric details for grasping as well as for placement of objects on the table.

In comparison to Ap2, HTAMP is also able to solve more efficiently cluttered problems with increasing number of obstacles in the workspace. The approach Ap2 is a randomized physics-based motion planner designed to plan grasping motions in cluttered environments, when the exact final placement of the objects is not relevant. The approach does not rely in a combination of task and motion levels, and no high-level explicit reasoning is done. Instead, robot-object and object-object interactions are allowed to push away those objects obstructing the way toward the object to be grasped. We observe that HTAMP scales better with the increasing number of objects regarding planning time, although the execution time can be worse if many objects have to be removed.

The proposed approach can be extended to consider other manipulation problems in which push or pull actions are required. To tackle this, those symbolic actions have to be defined in the action space, and also the relaxed geometric reasoning can be extended in order to compute feasible geometry information for the new actions.

Robotics manipulation problems become more challenging when the robot is required to place several objects in limited goal regions where placements of objects are critical. In such case, a large number of different object pose samples may be needed to determine the best feasible ones. This

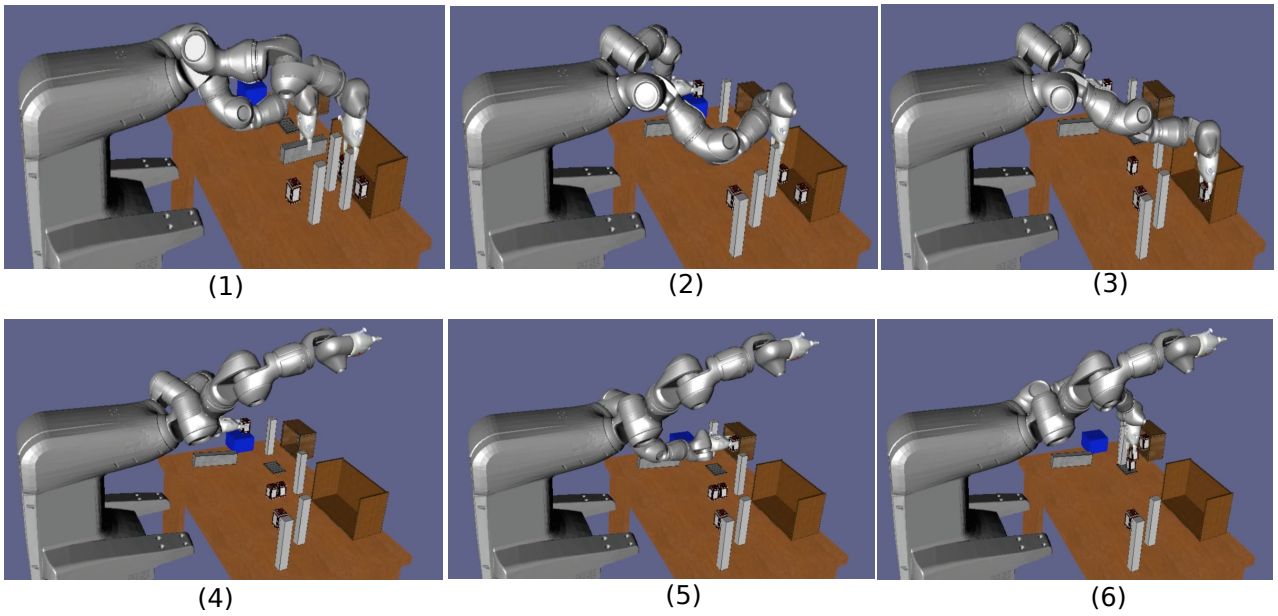

Fig. 4: The sequence of the snapshots of the execution for the 3D world problem. Video: `https://sir.upc.es/projects/kautham/videos/HTAMP-3D.mp4`

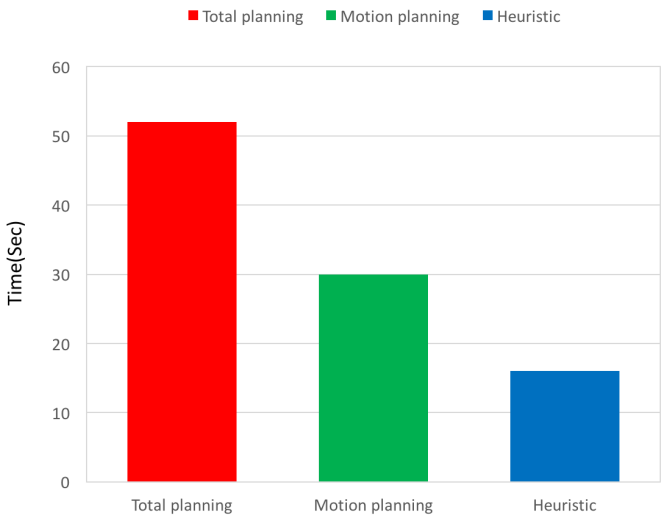

Fig. 5: The average time related to total manipulation planning and motion planning, and also reachability, spatial, and manipulation reasoning for 30 runs.

Problem	Success rate %			Av. planning time		
	Ap1	Ap2	HTAMP	Ap1	Ap2	HTAMP
Clutter 15	100	100	100	32	9.1	10.1
Clutter 20	94	100	100	57	16.7	11.8
Clutter 25	90	96	100	69	26	15.3
Clutter 30	84	90	100	77	41.2	18.2
Clutter 35	67	73	95	41	49.6	19.3
Clutter 40	63	60	95	68	71.6	21.8

Table 1: Comparison of HTAMP with two different approaches. Ap1 presented in (Srivastava et al, 2014) and Ap2 presented in (Muhayyuddin et al, 2018).

Problem	Rch-reas	Sp-reas	Manip-reas	Motion planning
Clutter 15	0.8	0.11	1.01	5.74
Clutter 20	1.12	0.21	1.9	6.41
Clutter 25	1.45	0.32	2.45	8.42
Clutter 30	1.59	0.49	3.6	10.63
Clutter 35	2.17	0.71	3.95	9.81
Clutter 40	3.07	1.05	5.17	10.1

Table 2: The average total reachability, spatial, and manipulation reasoning, and also motion planning time of the cluttered environment problems.

could decrease the performance of HTAMP since the number of combinations can grow very large (as it is the case for packing problems). Therefore heuristic computation could be slower, or the planner could have to restart several times if no feasible combination of object poses is found.

9 Conclusion

To solve manipulation problems for bi-manual robots, a combined heuristic-based task and motion planning approach that looks for the plan in the state space has been proposed. The main challenge was to provide low-level geometric information for guiding the task planner. The proposed idea

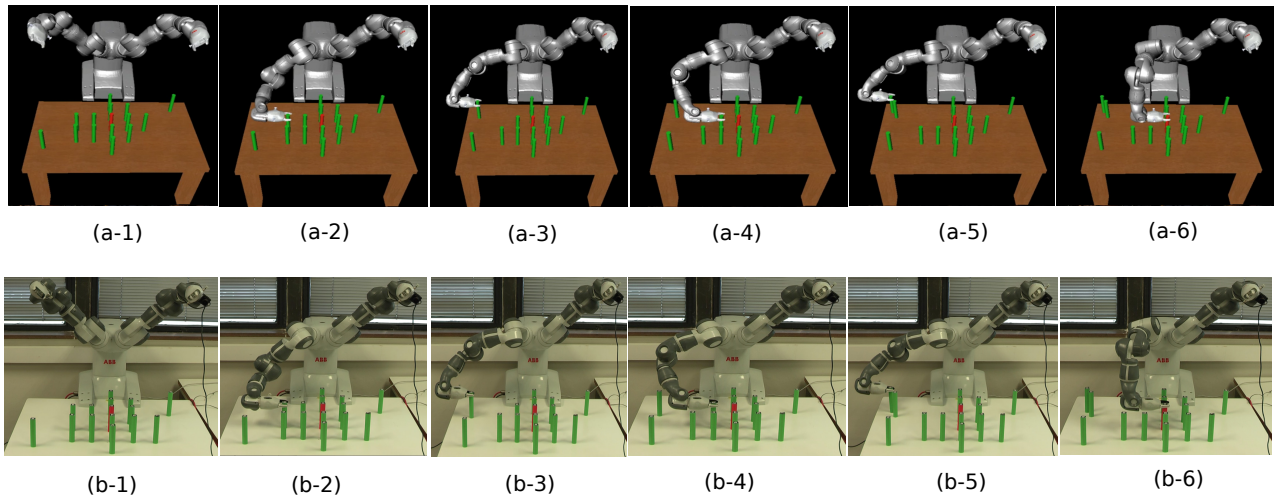

Fig. 6: The sequence of the execution snapshots for the cluttered manipulation problem where the Yumi robot has to grasp the red object among 15 ones in the environment. The problem has been implemented in simulation (a-1 to a-6) and the real environment (b-1 to b-6). Video: `https://sir.upc.es/projects/kautham/videos/HTAMP-15.mp4`

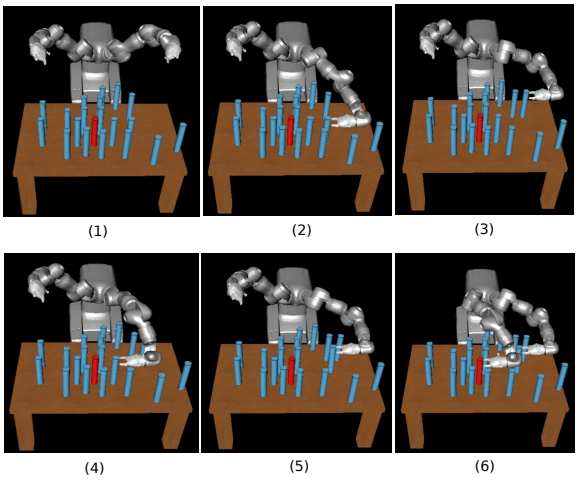

Fig. 7: The sequence of the snapshots of the execution for the cluttered manipulation problem where the Yumi robot has to grasp the red object among 20 ones in the environment. Video: `https://sir.upc.es/projects/kautham/videos/HTAMP-20.mp4`

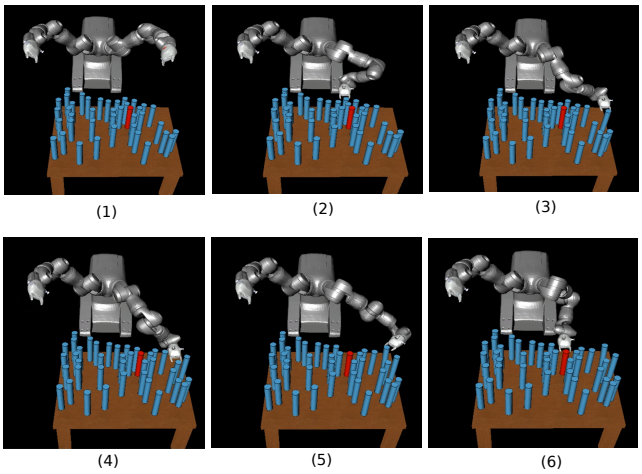

Fig. 8: Some sequence of the snapshots of the execution for the cluttered manipulation problem where the Yumi robot has to grasp the red object among 40 ones in the environment. Video: `https://sir.upc.es/projects/kautham/videos/HTAMP-40.mp4`

is to use the FF heuristic state-space task planner, which was modified so that its heuristic function takes geometric constraints into account through various geometric reasoning procedures.

During the heuristic computation, a relaxed geometric problem is addressed, which considers only certain types of task constraints. Therefore, the heuristic function is able to guide state space search towards geometrically feasible states. Once an action has been selected in the state space, the motion planner is called to confirm this choice by computing a collision-free path.

Using this technique, the proposed approach is able to find feasible manipulation plans without geometric pre-computation, and without geometric backtracking. The proposed approach has been validated with various classes of table-top manipulation problems of different complexity, and a thorough comparison with recent approaches is presented. The proposal has been implemented and evaluated in simulation and through real experiments. The results show that the approach

can solve manipulation tasks efficiently both in terms of planning time and success rate.

As future work, the integration of semantic knowledge with the planning process is envisioned, which will allow to flexibly apply the proposed approach to problems involving manipulation of objects with various attributes and specific manipulation constraints. Also, we may consider alternative task planning techniques, such as contingent planning, as a way to cope with uncertainty and achieve better performance in real situations.

References

Akbari A, Muhayyuddin, Rosell J (2015a) Reasoning-based evaluation of manipulation actions for efficient task planning. In: ROBOT2015: Second Iberian Robotics Conference, Springer

Akbari A, Muhayyuddin, Rosell J (2015b) Task and motion planning using physics-based reasoning. In: IEEE Int. Conf. on Emerging Technologies and Factory Automation

Akbari A, Muhayyuddin, Rosell J (2016) Task planning using physics-based heuristics on manipulation actions. In: IEEE Int. Conf. on Emerging Technologies and Factory Automation

Azizi V, Kimmel A, Bekris K, Kapadia M (2017) Geometric reachability analysis for grasp planning in cluttered scenes for varying end-effectors. In: Automation Science and Engineering (CASE), 2017 13th IEEE Conference on, IEEE, pp 764–769

Blum AL, Furst ML (1997) Fast planning through planning graph analysis. Artificial intelligence 90(1):281–300

Cambon S, Alami R, Gravot F (2009) A hybrid approach to intricate motion, manipulation and task planning. The International Journal of Robotics Research 28(1):104–126

Clarke EM, Grumberg O, Peled D (1999) Model checking. MIT press

Dantam N, Kingston ZK, Chaudhuri S, Kavraki LE (2016) Incremental task and motion planning: A constraint-based approach. In: Robotics: Science and Systems

De Moura L, Bjørner N (2011) Satisfiability modulo theories: introduction and applications. Communications of the ACM 54(9):69–77

Dornhege C, Eyerich P, Keller T, Trüg S, Brenner M, Nebel B (2012) Semantic attachments for domain-independent planning systems. In: Towards Service Robots for Everyday Environments, Springer, pp 99–115

Erdem E, Haspalamutgil K, Palaz C, Patoglu V, Uras T (2011) Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation. In: IEEE Int. Conf. on Robotics and Automation Robotics and Automation, pp 4575–4581

Erdem E, Patoglu V, Schüller P (2016) A systematic analysis of levels of integration between high-level task planning and low-level feasibility checks. AI Communications 29(2):319–349

Garrett CR, Lozano-Pérez T, Kaelbling LP (2015) FFRob: An efficient heuristic for task and motion planning. In: Algorithmic Foundations of Robotics XI, Springer, pp 179–195

Ghallab M, Howe A, Knoblock C, Mcdermott D, Ram A, Veloso M, Weld D, Wilkins D (1998) PDDL—The Planning Domain Definition Language

Ghallab M, Nau D, Traverso P (2004) Automated planning: theory & practice. Elsevier

Hauser K (2014) The minimum constraint removal problem with three robotics applications. The International Journal of Robotics Research 33(1):5–17

Hauser K, Latombe JC (2010) Multi-modal motion planning in non-expansive spaces. The International Journal of Robotics Research 29(7):897–915

Hauser K, Ng-Thow-Hing V, Gonzalez-Baños H (2010) Multi-modal motion planning for a humanoid robot manipulation task. In: Robotics Research, Springer, pp 307–317

He K, Lahijanian M, Kavraki LE, Vardi MY (2015) Towards manipulation planning with temporal logic specifications. In: Robotics and Automation (ICRA), 2015 IEEE International Conference on, IEEE, pp 346–352

Hertle A, Nebel B (2017) Identifying good poses when doing your household chores: Creation and exploitation of inverse surface reachability maps. In: Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on, IEEE, pp 6053–6058

Hoffmann J, Nebel B (2001) The FF planning system: Fast plan generation through heuristic search. Journal of Artificial Intelligence Research pp 253–302

Kaelbling LP, Lozano-Pérez T (2011) Hierarchical task and motion planning in the now. In: IEEE Int. Conf. on Robotics and Automation Robotics and Automation, pp 1470–1477

Kavraki LE, Svestka P, Latombe JC, Overmars MH (1996) Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE transactions on Robotics and Automation 12(4):566–580

Kuffner JJ, LaValle SM (2000) Rrt-connect: An efficient approach to single-query path planning. In: Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on, IEEE, vol 2, pp 995–1001

Lagriffoul F, Andres B (2016) Combining task and motion planning: A culprit detection problem. The International Journal of Robotics Research 35(8):890–927

Lagriffoul F, Dimitrov D, Saffiotti A, Karlsson L (2012) Constraint propagation on interval bounds for dealing with geometric backtracking. In: Intelligent Robots and

Systems (IROS), 2012 IEEE/RSJ International Conference on, IEEE, pp 957–964

Lagriffoul F, Karlsson L, Bidot J, Saffiotti R (2013) Combining task and motion planning is not always a good idea. In: RSS Workshop on Combined Robot Motion Planning

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

LaValle SM, Kuffner JJ (2001) Randomized kinodynamic planning. The International Journal of Robotics Research 20(5):378–400

Lifschitz V (2002) Answer set programming and plan generation. Artificial Intelligence 138(1-2):39–54

Lozano-Perez T (1983) Spatial planning: A configuration space approach. IEEE transactions on computers C-32:108–120

Muhayyuddin, Akbari A, Rosell J (2015) Ontological physics-based motion planning for manipulation. In: IEEE Int. Conf. on Emerging Technologies and Factory Automation, IEEE

Muhayyuddin, Moll M, Kavraki LE, Rosell J (2018) Randomized physics-based motion planning for grasping in cluttered and uncertain environments. IEEE Robotics and Automation Letters 3(2):712–719, DOI 10.1109/LRA.2017.2783445, URL https://doi.org/10.1109/LRA.2017.2783445

Pednault EPD (1989) Adl: Exploring the middle ground between strips and the situation calculus. In: Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp 324–332

Quigley M, Conley K, Gerkey B, Faust J, Foote T, Leibs J, Wheeler R, Ng AY (2009) ROS: an open-source robot operating system. In: ICRA Workshop on Open Source Software, vol 3, p 5

Rosell J, Pérez A, Aliakbar A, Muhayyuddin, Palomo L, García N (2014) The Kautham Project: A teaching and research tool for robot motion planning. In: IEEE Int. Conf. on Emerging Technologies and Factory Automation

de Silva L, Pandey AK, Gharbi M, Alami R (2013) Towards combining HTN planning and geometric task planning. In: RSS Workshop on Combined Robot Motion Planning and AI Planning for Practical Applications

Siméon T, Laumond JP, Cortés J, Sahbani A (2004) Manipulation planning with probabilistic roadmaps. The International Journal of Robotics Research 23(7-8):729–746

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

Stilman M, Kuffner J (2008) Planning among movable obstacles with artificial constraints. The International Journal of Robotics Research 27(11-12):1295–1307

Stilman M, Schamburek JU, Kuffner J, Asfour T (2007) Manipulation planning among movable obstacles. In: Robotics and Automation, 2007 IEEE International Conference on, IEEE, pp 3327–3332

Sucan I, Moll M, Kavraki LE, et al (2012) The open motion planning library. Robotics & Automation Magazine, IEEE 19(4):72–82

Zaplana I, Claret JA, nez LB (2018) Análisis cinemático de robots manipuladores redundantes: Aplicación a los robots kuka lwr 4+ y abb yumi. Revista Iberoamericana de Automt́ica e Informática industrial 15(2):192–202