Universitat Politècnica de Catalunya

Ph.D. Program:

AUTOMATIC CONTROL, ROBOTICS AND COMPUTER VISION

Ph.D. Thesis

Combining task and motion planning for mobile manipulators

Aliakbar Akbari

Thesis Advisor: Jan Rosell Gratacòs

15 December 2018

Combining task and motion planning for mobile manipulators

Submitted in partial fulfillment of the requirements for the degree of Ph.D. in

Automatic Control, Robotics and Computer Vision

Supervised by

Jan Rosell Gratacòs

Institut d'Organització i Control de Sistemes Industrials

Universitat Politècnica de Catalunya

15 December 2018

This thesis is dedicated to my parents

Acknowledgments

I would like to thank my advisor Prof. Jan Rosell Gratacòs for his guidance, motivation, and help to complete this thesis. I would also like to thank Dr. Fabien Lagriffoul for their guidance during my research stays.

I would like thank my colleagues Muhayyuddin and Mohammed Diab for their help and useful long discussions. I would also like to thank Raúl Suárez and all the members of Service of Industrial Robotics research group for their help.

This work is partially supported by the grant FPI-Ministry 2015 provided by the Spanish government.

Finally, special thanks to my father Morteza Akbari, my mother Faranak Rashid and the rest of my family, it was not possible to complete the thesis without their support.

Contents

List of Figures

List of Tables

Abstract

This thesis addresses the combination of task and motion planning to deal with different types of robotic manipulation problems, ranging from single or multiple collaborative mobile robots navigating among movable obstacles to complex higher-dimensional table-top manipulation tasks carried out by dual-arm robots or mobile manipulators. The combination of task and motion planning pursues the obtention of a geometrically feasible manipulation plan through a symbolic and geometric search space. It has emerged as a challenging issue as the failures due to geometric constraints lead robots to dead-end tasks.

Manipulation tasks in which there may be interactions between robots and objects are considered. To cope with them, task planning is combined with physics-based motion planning and information about the physics of the environment, allowing to deal with push and pull actions and also to look for low-cost plans in terms of power. Moreover, some of the proposed task and motion planning frameworks are enriched with the use of manipulation knowledge in order to facilitate the planning process and to aid in providing the ways of executing symbolic actions.

Problems with uncertain information (in the initial state of the robot world or in the result of symbolic actions) and problems where humans and robots interact are also considered. To deal with such issues, a contingent-based task and motion planner is proposed, which assumes the availability of a perception system (to evaluate the actual state of the environment) and the collaboration of the human operator (robots can ask humans for those tasks which are difficult or infeasible for them).

An implementation framework to combine different types of task and motion planners is presented. All the required modules and tools are illustrated, including the explanations on the flow of information between the different languages used, Prolog and C++.

Chapter 1

Introduction

Increasing complexity of daily manipulation tasks requires a robot to become more capable, robust, as well as autonomous in order to carry out various manipulation actions in human-like environments. Towards solving intricate manipulation problems, robotic systems need to apply *Task and Motion Planning (TAMP)* which looks for a discrete sequence of symbolic actions along with a motion plan solution for each of them. The scope of high-level task planning has been substantially studied in the Artificial Intelligence community. It works well to obtain a symbolic plan for large state spaces although task planning is not aware of geometric constraints imposed by the environment. On the other hand, motion planning deals with finding motion solutions or reporting geometric constraints in the case of failure if any. Therefore, the combination of task and motion planning is required for conducting robots towards appropriate manipulation solutions in a successful way. Various techniques in combining task and motion planning levels have been investigated in Robotics and the main challenging issue is the efficient way of interacting along sharing information between both planning levels.

Basically, generic motion planning is devoted to find collision-free trajectories. However, some manipulation actions (e.g. pushing objects away) exist involving interactions between robots and objects and, in this direction, physics-based motion planning have a valuable significance in order to allow motion planning strategies to incorporate possible interactions between rigid bodies, hence permitting the purposeful manipulation of objects.

Moreover, the use of knowledge-based techniques may enhance both task and motion planning levels by handing over sufficient information to robots. The modelling of knowledge by means of ontologies make it more flexible the use and the reasoning over the knowledge, providing fruitful information for the aid of actions selection and the way of their execution regarding the way to interact with the objects.

The present thesis develops different strategies of combining task and motion planning algorithms to cope with robotic manipulation tasks of increasing complexity. The strategies used include interleaved and simultaneously ways of solving task and motion planning. Task planners used are based on a planning graph technique and heuristic-based state space search. Motion planners used are based on physics-based or geometry approaches according to the type of actions. Manipulation knowledge has been introduced to guide action selection process and the way of actions execution. This thesis addresses the following research problems:

- **Navigation among movable obstacles *(NAMO)*:** *NAMO* problems are manipulation problems in which a mobile robot needs to push some objects away in order to reach the goal region. The tasks are challenging because of the following reasons:
  - *NAMO* problems where one robot requires to solve the task. There could be some situations that manipulation actions are infeasible due to geometric constraints from the environment, so they have to be identified while a robot is looking for a feasible manipulation plan. Moreover, finding a low-cost manipulation plan in terms of power can be a challenging issue.
  - *NAMO* problems where multi mobile robots may require to share the task or collaborate with each other in order to satisfy the manipulation tasks. Although geometrically feasible and low-cost plan have to be taken into account, the coordination of sub-tasks becomes challenging. For instance, a robot must be required to push away some objects to clear the path for another robot to travel to different regions in the workspace, or the execution of cooperative actions between several robots (the push of a heavy obstacle may require the use of more than one robot).
- **Higher dimensional manipulation problems:** They are referred to as manipulation problems where bi-manual autonomous fixed robots or mobile manipulators require to solve table-top manipulation tasks in the presence of different challenges which are summarized as follows:
  - Table-top problems may include a bi-manual robot where challenges are subject to strong geometric constraints (lack of space for placing objects, occlusions). 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 provide a geometrically feasible manipulation plan.
  - The problems can be even more challenging in human environments where mobile manipulators are involved and they need human-interactions. Human-interactions can be viewed in two phases: a) transferring knowledge to the robots when they have uncertain information about some situations of objects, b) asking an operator to perform some actions as a cooperator in the case that the robots encounter some difficulties or objects are located in human-workspace.
- **Manipulation knowledge:** To provide better information in task and motion planning, semantic manipulation knowledge is used. For example, a robot may know which actions are out of its capability using knowledge processing and may be able to identify the way

of interactions with objects holding manipulation constraints. Accordingly, task and motion planning components are represented using ontologies and a lightweight reasoning process is proposed.

1.1 Contributions

The main contributions of the thesis are listed below.

- *Semantic manipulation knowledge.* The manipulation knowledge framework is presented in terms of ontology-based modelling and lightweight reasoning process. It enriches the combination of task and motion planning by providing components of planning process and semantic information concerning the robot workspace. This framework is also integrated with physics-based motion planning that aims mainly to provide the way of interactions between a robot and objects holding specific manipulation constraints.

- *Combining task and motion planning for mobile robots using a planning graph technique.* Two frameworks of combined task and physics-based motion planning are presented using different combination and reasoning strategies. The semantic manipulation knowledge are integrated within the planning process. The framework deals with the scope of the navigation among movable obstacles problems in which mobile robots require to push manipulatable objects away in order to reach goal regions.

- *Heuristic-based task and motion planning for collaborative multiple mobile robots.* This approach solves simultaneously task and physics-based motion planning equipped with the semantic manipulation knowledge and different type of reasoning processes. The approach deals with a set of mobile robots in the navigation among movable obstacles problems where they require to collaborate or share the manipulation tasks with each other in order to satisfy the goals of the problem. It proposes also physics-based heuristics to obtain an efficient global manipulation plan for all the robots.

- *Efficient heuristic-based task and motion planning method for bi-manual robots.* This method combines task and motion planning simultaneously and offers a set of reasoning processes to deal with bi-manual robots. The heuristic computation addresses a geometrically relaxed problem (reasoning upon objects placements, grasp poses, and inverse kinematics solutions), and motion paths are evaluated lazily. This reduces the number of calls to a motion planner, while backtracking is reduced because the heuristic captures most of the geometric constraints. The method considers *Pick* and *Place* manipulation problems with the extension of *Push* actions.

- *Task and motion planning for human-robot interaction under uncertainty.* This approach relies also on combining simultaneously task and motion planning using a heuristic-based method. It tackles the type of uncertainty considered at the task level, and also the necessity of collaborating a mobile robot with human if demanded. Interacting with human can be accomplished in two phases: the robot requires human knowledge to determine its uncertain information and for the robot tasks being difficult or infeasible. So, the robot needs human collaboration in completing the manipulation tasks.

- *Implementation framework for combining task and motion planning.* To implement the aforementioned task and motion planning approaches, an implementation framework is proposed. The approaches are implemented using either C++ and or the Prolog languages and need different integration techniques. Hereupon, the implementation framework provides a flexible way to handle the communication between the languages using mainly *Robotic Operating System* shell and the *SWI-Prolog* library. It enables to visualize manipulation plans in simulation, and moreover to execute them in a real environment.

1.2 Outline of the Thesis

The rest of the thesis is structured below. Chapter 2 summarizes some related work. Chapter 3 illustrates the semantic manipulation knowledge in terms of modelling and reasoning process. Regarding the navigation among movable obstacles problems, Chapter 4 concentrates on different strategies of combining task and motion planning for mobile robots based on a planning graph technique and Chapter 5 presents an heuristic-based combination approach to cope with multiple mobile robots. The approaches apply different set of reasoning processes and use the semantic manipulation knowledge. Moreover, Chapter 6 represents an efficient heuristic-based task and motion planning approach for bi-manual robots taking into account efficient geometric reasoning process. Chapter 7 presents a combined approach to tackle uncertainty and human-robot interactions. Chapter 8 shows the implementation framework used in combining task and motion planning using different techniques. Eventually, Chapter 9 sketches the thesis conclusions and comes up with some future works.

1.3 List of Publications

The list of all publications which I participated as authors or co-authors is shown as follows:

Journal Publications:

- Akbari, A., Diab, M., and Rosell, J. (2019). Contingent task and motion planning under uncertainty for human-robot interactions. Submitted to IEEE Robotics and Automation Letters (under review).
- Diab, M., Akbari, A., Muhayyuddin, and Rosell, J. (2019). PMK- a knowledge processing framework for autonomous robotics perception and manipulation. Submitted to Sensors (under review).
- Akbari, A., Muhayyuddin, and Rosell, J. (2018). Knowledge-oriented task and motion planning for multiple mobile robots. Journal of Experimental & Theoretical Artificial Intelligence, pages 1-26, Taylor & Francis.
- Akbari, A., Lagriffoul, F., and Rosell, J. (2018). Combined heuristic task and motion planning for bi-manual robots. Autonomous Robots, pages 1-16, Springer US.
- Lagriffoul, F., Dantam, N., Garrett, C., Akbari, A., Srivastava, S., and Kavraki, L. (2018). Platform-independent benchmarks for task and motion planning, pages 3765 - 3772, IEEE Robotics and Automation Letters.
- Muhayyuddin, Akbari, A., and Rosell, J. (2017). k-pmp: Enhancing physics-based motion planners with knowledge-based reasoning. Journal of Intelligent and Robotic Systems, pages 1-19, Springer Netherlands.

Conference and Workshop Publications:

- Bebler, D., Pomarlan, M., Akbari, A., Diab, M., Rosell, J., Bateman, J., and Beetz, M. (2018), Assembly planning in cluttered environments through heterogeneous reasoning, KI 2018: Advances in Artificial Intelligence, pages 201-214, Springer International Publishing.
- Rosell, J., Akbari, A., Muhayyuddin,and Diab, M. (2018). A Knowledge-based Planning Framework for Smart and Autonomous Manipulation Robots. Workhsop on Combining Task And Motion Planning In The Frame Of Cloud Robotics, 2018 IEEE International Conference on Simulation, Modelling, and Programming for Autonomous Robots (SIMPAR).
- Muhayyuddin, Akbari, A., and Rosell, J. (2017), Knowledge-oriented physics-based motion planning for grasping in the presence of uncertainty. In Robot: Third Iberian Robotics Conference, Springer, Advances in Robotics, pages 502-515, Springer International Publishing.

- Muhayyuddin, Akbari, A., and Rosell, J. (2017), A Tool for Knowledge oriented Physics-based Motion Planning and Simulation. Proceedings of the EAI International Conference on Future Intelligent Vehicular Technologies, Springer International Publishing.
- Diab, M., Muhayyuddin, Akbari, A., and Rosell, J. (2017), An ontology framework for physics-based manipulation planning. In Robot: Third Iberian Robotics Conference, Advances in Robotics, pages 452-464, Springer International Publishing.
- Muhayyuddin, Akbari, A., and Rosell, J. (2017). Physics-based motion planning with temporal logic specification, IFAC-PapersOnline, pages 8993-8999, Elsevier.
- Rosell J, Nuño E, Claret JA, Zaplana I, García N, Akbari A, Muhayyuddin, Palomo L, Pérez A, Mas O, and Basanez L., (2017), Mobile manipulators as robot co-workers: autonomy and interaction in the human-robot collaboration, Comité Español de Automática (CEA-IFAC), pages 1-6.
- Akbari, A., Muhayyudin, and Rosell, J. (2016). Task planning using physics-based heuristics on manipulation actions. In IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), pages 1-8, IEEE.
- Akbari, A., Ud Din, M., and Rosell, J. (2016). Integrated task and motion planning using physics-based heuristics. In RSS Workshop on Task and Motion Planning: posters.
- Akbari, A., Muhayyuddin, and Rosell, J. (2015). Reasoning-based evaluation of manipulation actions for efficient task planning. In ROBOT: Second Iberian Robotics Conference, pages 69-80, Springer International Publishing.
- Akbari, A., Muhayyudin, and Rosell, J. (2015). Task and motion planning using physics-based reasoning. In IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), pages 1-7, IEEE.
- Muhayyuddin, Akbari, A., and Rosell, J. (2015). Physics-based motion planning: evaluation criteria and benchmarking, In Robot: Second Iberian Robotics Conference, pages 43-55, Springer International Publishing.
- Muhayyuddin, Akbari, A., and Rosell, J. (2015). Ontological physics-based motion planning for manipulation, In IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), pages 1-7, IEEE.
- Rosell, J., Pérez, A., Akbari, A., Palomo, L. and García, N. (2014), The Kautham Project: A teaching and research tool for robot motion planning, In IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), pages 1 - 8, IEEE.

1.4 Publication Note

Chapter 3 covers semantic knowledge presented in (Akbari et al., 2018b) and (Muhayyuddin et al., 2018). Chapter 4 explains the approaches appearing in (Akbari et al., 2015b) and (Akbari et al., 2015a) regarding mobile manipulation tasks. The approach related to collaborative manipulation tasks for multiple mobile robots presented in chapter 5 appears in (Akbari et al., 2018b) and (Akbari et al., 2016a). Chapter 6 develops the task and motion planning approach for bi-manual robots presented in (Akbari et al., 2018a) and (Rosell et al., 2018). The most parts of the implementation approach presented in chapter 8 appear in (Akbari et al., 2016b) and (Rosell et al., 2014).

Joint Works: Two joint works have been done with the collaboration with other research centers. The first one has been done with the collaboration with the *KnowRob* group from Bremen university. The geometric reasoning module proposed for combining task and motion planning for bi-manual robots has been offered to an ontology knowledge planner developed by *KnowRob* group. The result is published in (Beßler et al., 2018). The second one is about platform-independent benchmarks for task and motion planning which is proposed by some of the *TAMP* authors. This is the first attempt to propose some benchmarking problems that some of them have been tested in the present thesis. The work is published in (Lagriffoul et al., 2018).

Chapter 2

Background

2.1 Task Planning

Task planning is typically devoted to find a discrete sequence of symbolic actions to transition from a given initial state to a desired goal condition for a given problem. There are different approaches in *Artificial Intelligence (AI)* such as classical planning, logic programming, or constraint satisfaction which perform task planning under fully observable initial state. They 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. Basically, an action has preconditions which is a set of atoms which must hold for the action to be applied, and also positive and negative effects which are a set of atoms added and deleted respectively after the action is applied. Each state-level consists of the union of literals 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. Regarding mutual exclusion, different constraints between actions and between literals may exist in the *planning graph*. Two actions are constrained when:

- An effect of one action negates an effect of the other action. It is called the *inconsistent effect* constraint.
- An effect of one action deletes a precondition of the other action. It is called the *interference*

constraint.

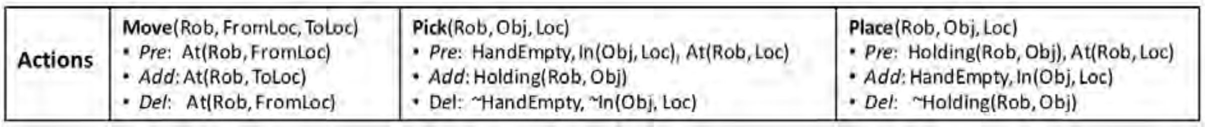
- They have mutually exclusive preconditions. It is called the *competing needs* constraint.

Two literals, moreover, are constrained if:

- All the ways of achieving them are pairwise mutex. It is called the *inconsistent support* constraint.

For instance, it is assumed that a robot *Rob* is going to deliver an object *Obj* to a new location using the manipulation actions *Move, Pick,* and *Place* shown in Fig. 2.1 with their preconditions and effects. The initial state is *[At(r, A), HandEmpty, In(obj, B)]* and the goal condition is *[In(obj, A)]*. In this case, the robot must move to the object location in order to pick up the object at its first location B, and then the robot needs to move back to the location A in order to place the object in the target location. Therefore, the solution plan will be $\pi =$ (*Move, Pick, Move, Place*).

Fig. 2.2 shows the complete levels of the *planning graph* process for the mentioned problem and some constraints applied at first levels. State-level and action-level depict necessary predicates and actions at each level respectively. Each action-level also contains maintenance actions for predicates to remain unchanged for the next level. The *planning graph* space starts with the initial state and looks for actions whose preconditions appear in the present level. At the state-level 1, there are predicates *At(r, A)* and *∼At(r, A)* that cannot happen at the same time and because of this reason, their action and maintenance action are constrained at the action-level 1. This type of constraint is called *"inconsistent effect"* and acts in such a way that the effect of one negates the effect of the other. As it can be seen, all the goal conditions are not satisfied by this level, so the search space grows by one level more. The process is repeated for the next level, whereas *Pick* and *Move* actions are likewise constrained due to *"competing needs"* constraint. This constraint is posed because of the actions have mutually exclusive preconditions. The search space is expanded to find the level in which all goal conditions occur that is the state-level 3. Backtracking search is attempted to find the solution from the the current level, but it fails because there is no way of satisfying all the constraints. Therefore, the *planning graph* is extended one more level. Consequently, the complete solution plan is able to be extracted from the state-level 4. As represented in the example, the backtracking search must satisfy all the constraints, so it could be sometimes difficult to find a solution plan from the first state-level where all the goal conditions appear and the *planning graph* requires to be expanded more in order to simply the constraints.



Actions	**Move**(Rob, FromLoc, ToLoc)	**Pick**(Rob, Obj, Loc)	**Place**(Rob, Obj, Loc)
	• *Pre*: At(Rob, FromLoc)	• *Pre*: HandEmpty, In(Obj, Loc), At(Rob, Loc)	• *Pre*: Holding(Rob, Obj), At(Rob, Loc)
	• *Add*: At(Rob, ToLoc)	• *Add*: Holding(Rob, Obj)	• *Add*: HandEmpty, In(Obj, Loc)
	• *Del*: At(Rob, FromLoc)	• *Del*: ~HandEmpty, ~In(Obj, Loc)	• *Del*: ~Holding(Rob, Obj)

Figure 2.1: Describing action templates in terms of preconditions, positive effects, and negative effects.

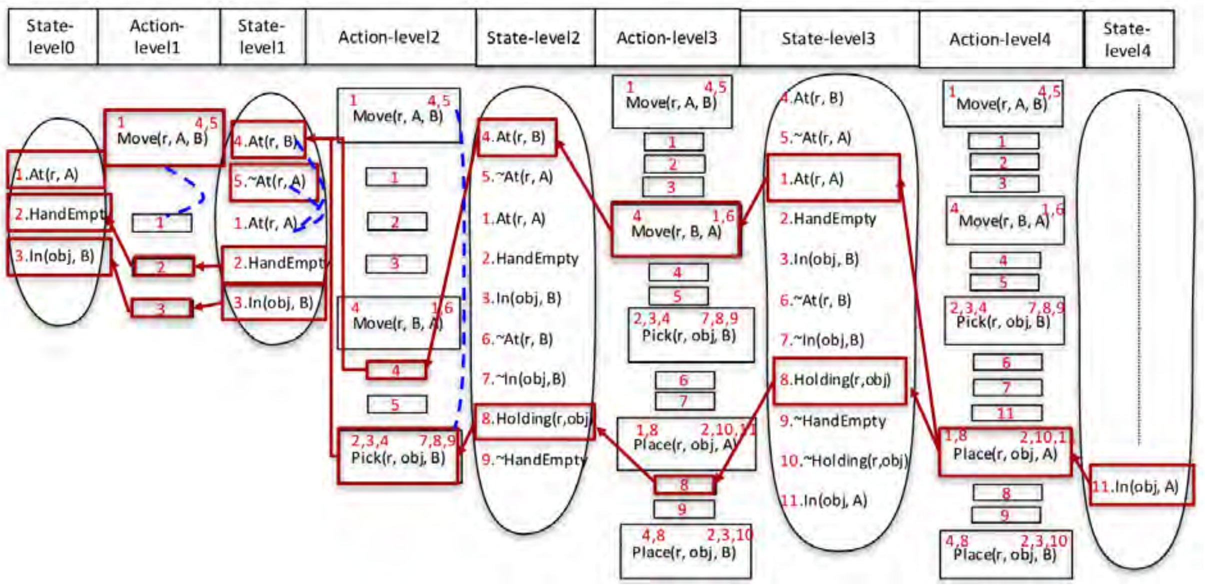


Figure 2.2: The complete levels of the *planning graph* search space. Non-labeled rectangles refer to maintenance actions, dashed arcs refer to the constraints, and the selected actions and the states are highlighted.

Alternatively, search can be done in state space. In this line, one of the most efficient task planning approaches is the *Fast-Forward (FF)* planner (Hoffmann and Nebel, 2001) which performs a heuristic search. The architecture of the planner is represented in Fig. 2.3. 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 negative 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 minimum number of 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 a complete heuristic search is invoked to solve the task from scratch. This strategy is done using *Best-First Search (BFS)* that simply expands all applicable actions and search nodes.

In planning based on the hierarchical structure, the purpose is to incorporate dependency among actions in the form of hierarchical structure in order to satisfy the task. In principle, hierarchical-based approaches apply the concept of task networks, that is called *Hierarchical Task Network (HTN)* (Erol et al., 1995). In this case, various tasks and constraints between them are

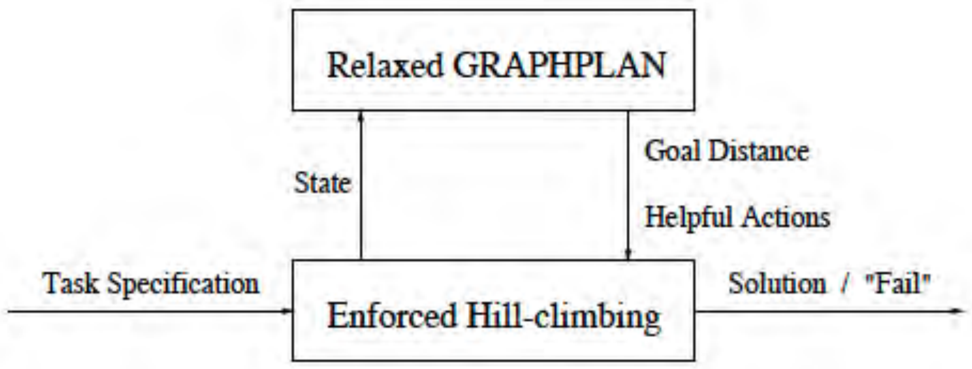


Figure 2.3: The Fast-Forward planner architecture (Hoffmann and Nebel, 2001) .

expressed and formed in a task network. Such a network aims to provide possible precondition of a goal task. HTN-based planning includes three main parts; primitive tasks, compound tasks, and goal tasks. A primitive task is the action which is directly executed. A sequence of primitive actions can be merged as a compound task. A goal task is a target condition which has to be satisfied in planning. The procedure is that a planning problem is given to hierarchical planning, then tasks are decomposed and grow in a search space. It performs searching until a sequence of primitive tasks are achieved with respect to satisfying preconditions.

Other task planning approaches exist that use constraint-based techniques. *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. The main components of *PDDL* are *Objects*, *Predicates*, *Initial-State*, *Goal-Specification*, and *Actions*. *Objects* are things in the planning world. *Predicates* are the properties of objects in the world that can be either true or false. *Initial-State* is the current state of the word at which planning begins. *Goal-Specification* is the target which has to be solved by planning. Actions are the ways of changing the world state. Planning tasks are structured in two files in *PDDL*; a domain file in which *Predicates* and *Actions* are defined, and then a problem file where *Objects*, *Initial-State*, and *Goal-Specification* are represented.

2.2 Task Planning under Uncertainty

Task planning can be done under incomplete information; therefore, it needs to plan under uncertainty. Task planning under uncertainty is also a well-established field in *AI*. There are various classes of planning in this field like conformant, contingent, or probabilistic planning.

Conformant planning looks for plans under given uncertainty concerning the start state and the effects of symbolic actions, assuming no sensing capabilities during the execution of the plan. The plan should be successful regardless of which is the start state. Contingent planning also considers uncertainty regarding the start state and the effects of actions. However, it has the ability to provide some sort of observation over a conditional plan in execution. Probabilistic planning does planning under probabilistic uncertainty regarding the start state and the effects of actions.

More details on some approaches following conditional-based task planning are commented next as one of the proposals of this thesis uses that. We are interested in this type of planner due to its feature of providing observation over a conditional plan. Some conditional task are: *Contingent-FF* (Hoffmann and Brafman, 2005), *POND* (Bryce et al., 2006), and *PKS* (Petrick and Bacchus, 2004). They plan in the belief space and compute conditional plans in the offline mode. Sensing actions are provided in the conditional plans which can be guided by the result of sensing actions. On the other hand, there are some conditional task planners like *K-Planner* (Bonet and Geffner, 2011), *SDR* (Brafman and Shani, 2012), and *HCP* (Maliah et al., 2014) solving conditional plans online. Although these planners can prune some contingency branches by considering online sensing actions, satisfying the goal of task may not be possible and the planners may face with dead-end even if there is a solution.

We will propose a planner based on the modified version the *Contingent-FF* planner which is detailed next. Basically, the *Contingent-FF* handles uncertainty in the initial state and for the nondeterministic actions. The initial state is a belief state represented by *Conjunctive Normal Form (CNF)* which expresses formulas as conjunctions of clauses with an *AND* or *OR*. Besides, the normal actions, which have preconditions and effects, and observation actions observing the value of uncertain facts and may have preconditions are considered. The task planner has two main components which are a search space and a heuristic computation.

The search space is based on a standard *And-Or* forward search mechanism in belief space. It is an *And-Or* tree whose nodes become alternatively belief states (*Or* nodes) and actions (*And* nodes). The *Or* children of a belief state associate with applicable actions and the *AND* children of an observation action associate with its two different outcomes. Regarding actions without observation, there is only one child; therefore, the *And* node trivializes. While the belief space search is exploring, the sets of known and negatively known facts are computed. Known facts correspond with those which hold in a world state and negatively known facts are the ones which do not hold in any world state. The facts which are not neither known and negatively known are called unknown. The sets of known and negatively known facts are computed in a belief state using the *SAT* solver which checks whether the given formula is satisfiable or unsatisfiable. Applicable actions are those whose preconditions are all known in a belief state. A belief state satisfies the goal if all goal conditions become known. From each belief state, the heuristic function is called to compute the heuristic value and the helpful actions.

For the heuristic computation, the planner uses a modified version of the *RPG* in the *FF* planner that is called *CRPG*. In the normal *RPG* process, the state-levels include known facts. They are extended in *CRPG* with unknown facts. A reasoning has been introduced about unknown facts such that they are shifted to known facts when they become known based on the reasoning done. The action-layers contain those actions whose preconditions are either achievable or observable with earlier actions. The *RPG* graph expansion process terminates from either the state-level in which the goal condition becomes known, so a relaxed plan can be extracted, or the graph reaches a fix point, indicating that a relaxed plan does not exist and the corresponding belief state can be skipped. This process is called *CRPG*. Once *CRPG* is built successfully, the backward search is initiated to find a sequence of actions from the known goal facts.

The result of the planning process provides a tree-shaped conditional plans. Such plans may involve a variety of observation actions whose outcome causing different plan branches. With respect to the observation binary value of the observation actions, the conditional plans switches from one plan branch to the other one.

2.3 Motion Planning

Motion planning deals with detecting collision-free paths to convey a robot to a configuration state. It 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 $C_{obstacle}$. Motion planning in $\mathcal{C}$-space consists in finding a path in $\mathcal{C}_{free}$ between two configurations.

With regard to classical motion planning, researches have investigated how to deal with geometric constraints in planning (they do not impose differential dynamic constraints). Traditional methods comprise different techniques such as cell decomposition, roadmap-based methods, and potential fields. In motion planning based on cell decomposition techniques, the first step is to decompose free space, either in an exact or an approximated way, and represent the set of cells as a graph. Then, some search algorithms such as *A** (Hart et al., 1968) or *Dijkstra* (Dijkstra, 1959) methods can be employed to find a path of minimal cost. The roadmap-based approaches build a graph to connect the initial and goal states in free space and they usually find the solution paths based on the shortest distance. Regarding this case, the obstacles in the $\mathcal{C}$-space are represented as polygons, the visibility graph methods (De Berg et al., 2000) generates a graph whose nodes are the vertices of the polygons and whose edges are line segments with no interference of obstacles. The last approach uses the potential fields computed in the $\mathcal{C}$-space to provide attraction to the goal configuration and repulsion from obstacles (Khatib, 1986). The drawback of these approaches is that for problems with more than two or three degrees of freedom, they are hard or impossible to implement and computationally prohibitive because they require the construction of the $\mathcal{C}_{obstacle}$ in the configuration space.

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. It results in probabilistic complete planning. The core of sampling-based motion planning is to use random-based techniques to sample the $\mathcal{C}$-space and to interconnect those collision-free configurations as roadmaps or trees to capture the connectivity of $\mathcal{C}_{free}$. To cope with high dimensional degrees of freedom, it is found as an appropriate strategy to reduce the complexity of problems. Some sampling-based motion planners are investigated by (Elbanhawi and Simic, 2014). The most popular approaches are tree-based techniques, which provide a single query, and roadmap-based techniques, which provide multiple queries as described below.

If planning is carried out in static environments that remain constant at each manipulation step, roadmap-based approaches provide better solutions (even though they are costly to be implemented due to the need of exploring the whole $\mathcal{C}$-space) because they make it possible to set multiple queries. Path planning based on the *Probabilistic Roadmap Method (PRM)* (Kavraki et al., 1996) that works 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.

In manipulation planning, however, since the status of the environment is altered after executing each action, tree-based algorithms have better solutions because they provide specific explorations of the $\mathcal{C}$-space, e.g., focused on a given query. In this case, the environment is reconfigured and the $\mathcal{C}$-space must be again explored. Here, some of the well-known single query sampling-based motion planners are reviewed as they fit better in manipulation problems and some of them will be used later in this thesis.

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* planner (Kuffner and LaValle, 2000) is a variant of *RRT*. There are two trees which are based on the *RRT* data structure, that are rooted at the start and goal configurations in order to meet each other. In this case, one tree is extended and attempts to connect the closest node of another tree to the new node in each iteration. Afterwards, the procedure is reversed for the next attempt by swapping both trees. A greedy heuristic method is applied to convey two trees. The method provides substantial improvements in the search efficiency. The *RRT** planner (Karaman and Frazzoli, 2011) has been developed that minimizes cost of the returned solution comparing with the *RRT* algorithm. After growing the tree as done in the basic *RRT* algorithm, the *RRT** captures the set neighbor nodes $\mathcal{N}$ of each new added nodes to verify if it can be reached through them with a less cost and, in this case, edges are rewired. Then, this procedure is repeated for the nodes in $\mathcal{N}$, which may to be rewired accordingly.

Kinodynamic Motion Planning by Interior-Exterior Cell Exploration (KPIECE) planner is particularly designed for complex dynamical systems (Şucan and Kavraki, 2009). *KPIECE* grows a tree of motions by applying randomly sampled controls for a randomly sampled time duration from a tree node selected as follows. The state space is projected onto a lower dimensional space that is partitioned into cells in order to estimate the coverage. As a result of this projection, each motion will be part of a cell, being each cell classified as an interior or exterior cell depending on whether the neighboring cells are occupied or not. Then, the selection of the cell is performance based on the importance parameter that is computed based on: 1) the coverage (the cells that are less covered are preferred over the others); 2) the selection (the cells that have been selected less number of time are preferred); 3) the neighbors (the cells that have less neighbors are preferred); 4) the selection time (recently selected cells are preferred); 5) the expansion (easily expanded cells are preferred over the cells that expand slowly). The cell that has maximum importance will be chosen, and a node of one of the motions of the cell will be randomly selected. The process continues until the tree of motions reaches the goal region.

Synergistic Combination of Layers of Planning (SyCLoP) planner (Plaku et al., 2010) is a meta approach that considers motion planning as a search problem in a hybrid space (of a continuous and a discrete layer) for efficiently solving the problem under kinodynamical constraints (Plaku et al., 2010). The continuous layer is represented by the state space (that is explored by a sampling-based motion planner like *RRT*) and the discrete layer is determined by the decomposition of the workspace. The decomposition is used to compute a cost parameter called lead that guides the motion planner towards the goal. *SyCLoP* works based on the following steps lead computation and region selection. The lead is computed based on the coverage and the frequency of the selection. The former is obtained by the sampling based motion planner (continuous layer) and the latter is computed by determining how many times a cell has been selected from discrete space. The selection of the region will be performed based on the available free volume of the region (high free volume regions are preferred for the exploration). The process continues until the planner finds a sample in the goal region. *SyCLoP* could be recalled *SyCLoP-RRT* based on the planner used in the continuous layer.

2.4 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. Manipulation problems are referred to as problems in which robots manipulate objects using a set of primitives, e.g., pushing, picking, or placing. One manipulation approach 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 (Muhayyudin et al., 2015).

A more general manipulation planning approach using several *PRMs* has been developed 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)*, at which a robot needs to manipulate some objects in the workspace in order to reach its goal region, has been addressed by 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 multi-modal 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., 2017) propose a geometric approach based on detecting a complete set of object subsurfaces in a cluttered scene, 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 (Muhayyudin et al., 2018) has been proposed, which 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.

The reviewed works much concentrate on geometry challenges in manipulation with no high-level reasoning, while there are other manipulation problems which require to search in a symbolic search space to find a sequence of actions. The approaches, furthermore, lack from underlying discrete symbolic relations (e.g. among robots and objects) which can break down the complexity of the problem. Therefore, combining task and motion planning makes a significant role to deal with different challenges of manipulation problems for robots.

2.5 Knowledge Representation and Reasoning Process

Knowledge-based representation techniques, like ontologies, can enhance manipulation planning by providing informative data with respect to the robot's world. Ontologies are one of the best techniques to represent knowledge because they are able to integrate semantic information (concluded by a set of rules defined by semantic languages for objects) that facilitates a better task reasoning process. By finding and relating facts in an ontology, robots can understand a task in a similar way a human does and answer questions such as *"should this task be performed or not?"*.

Several projects have pursued ontological modelling in robotics. The work in (Tosello et al., 2015) presents a semantic knowledge base that can be adopted as an information resource for autonomous cognitive robots performing manipulation tasks. The working group on ontologies for robotics and automation *(ORA WG)*, sponsored by the *IEEE Robotics & Automation Society*, proposed standardized ways of representing knowledge for service, industrial and autonomous robots (Schlenoff et al., 2012a). *ORA WG* has proposed the *Core Ontology for Robotics and Automation (CORA)* (Prestes et al., 2013), a conceptual structure to be used and integrated within other specific ontologies developed for the robotics and automation domain, i.e., with a main focus on reusability. The structure of *CORA* is based on the *Suggested Upper Merged Ontology (SUMO)* (Niles and Pease, 2001), that is a top-level ontology that aims to define the main ontological categories describing the world. Later, in order to provide the more generalized concepts of the robotic domain, *CORA* was extended by incorporating *CORAX* (Fiorini et al., 2015), *RPARTS* (robot parts) and *POS* (position) (Schlenoff et al., 2012b). *CORAX* is an ontology that defines the concepts not explicitly or completely covered by *SUMO* and *CORA*, such as robot motion (that categorizes the motion of the robot according to its type such as robot rolling or walking). *RPARTS* provides a general information of robot parts, also defining the attached parts on the robot such as robot sensing parts. *POS* captures general information about position and orientation.

The application of the knowledge has been used in different domains. In the navigation area, some works such as (Lim et al., 2011) and (Gemignani et al., 2016) use a metric map and a topological map to define the robot environment. The metric map is used for the geometrical representation of the robot workspace in terms of free and occupied areas, while the topological map is used to capture the topology of the workspace. Moreover, knowledge concerning housework activities was elaborated in (Tenorth and Beetz, 2009) and (Tenorth and Beetz, 2012) describing and reasoning over the knowledge in order to, e.g., reason about ontological actions effects and to know how to perform the actions. The work by (Beetz et al., 2011) presented an experiment in which robots retrieved a simple instruction for a meal preparation task from the web.

Ontological knowledge has also been integrated within task and motion planning framework. The works of (Di Marco et al., 2013; Freitas et al., 2014) apply the use of knowledge in HTN-based planning. In principle, the works represent the planning domain knowledge (such as goal or actions description) in a semantically expressive way using the *OWL* for the purpose of task specifications. The knowledge also encoded inside a high-level planning that combines *FF* with *HTN* task planning (Klusch et al., 2005). The work also proposed the conversation model to map information between the *PDDL* action description and the *OWL* information.

Most of reviewed approaches focus on various manipulation activities for robots. However, action feasibility and how a robot can interact with an object, that can be addressed using knowledge processing, has not been covered yet. In this thesis, semantic knowledge using the ontology method is provided in terms of modelling and reasoning process to facilitate manipulation planning process. It will be focused mainly to represent and reason over robot capability, objects type, and the way of manipulation with objects which hold various features and constraints. For instance, regarding robot capability, a robot is asked to pick a heavy object, but it cannot do it due to the weight of the object or some objects are manipulated using more than one robot. Also, planning components in terms of action description and query conditions will be represented in the ontology.

2.6 Combining Task and Motion Planning

In general, combination of task and motion planning needs to search in symbolic and geometric space. A task plan, including a sequence of symbolic actions, does not promise that actions become feasible in terms of geometry conditions. This requires the combined search and motivates the researchers to investigate efficient ways of the combination at both levels in order to acquire a feasible plan. Combining task and motion planning is an active research area in *Robotics* and *AI*. This section explains different strategies employed to combine task and motion planning with respect to different types of applications where the initial state and action effects are fully observable. Different studies have dealt with various strategies to combine *TAMP* with the aim of finding a feasible plan to solve a given task.

One way of the combination method is to consider a generic interface between symbolic and geometric planning levels to determine whether a collision-free motion for symbolic actions exists or not. 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 like (Erdem et al., 2011) and (Srivastava et al., 2014) a symbolic plan is computed *first*, and then geometric constraints are evaluated *afterwards*. This is done by calling a motion planner for each symbolic action in the plan obtained by the task planner. In the 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.

Moreover, there are two main ways of combining task and motion planning information: *simultaneously* or *interleaved*. The *simultaneously* approach accounts for geometric information by calling a motion planner while task planning is being pursued. The manipulation plan is then available after the task planning process is terminated. On the contrary, the *interleaved* approach decouples motion planning from the task planning part. Task planning is first done, and then call a motion planner to evaluate the feasibility of the plan. Upon failure, geometric constraints are fed back to the task planner and the process resumes. This can be repeated several times until a feasible manipulation plan is achieved. The study of (Erdem et al., 2016) investigated different combination 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. Next, the *TAMP* approaches are presented in which task planning and geometric reasoning are more tightly intertwined based on the proposed ways of combination.

FF-based TAMP. In this line, these studies are based on the *simultaneously* combination method. The study in (Cambon et al., 2009) presents 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.

Hierarchical-based TAMP. The works in this direction follow the *simultaneously* strategy. 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 application of combining hierarchical task and motion planning has been used by (Alami et al., 2014) and (Alili et al., 2010) for a teammate robot and robotic assistant. They use different geometric reasoning processes to find out a feasible plan. Moreover, the hierarchical model has also been presented for task and motion planning by introducing the concept of planning in the now (Kaelbling and Lozano-Pérez, 2011a). The main idea is to make a plan in the abstract level, commit to it, and provide recursive plan to execute actions without making the rest of the plan in detail. It also implements the *RRT* motion planner for primitive actions.

LTL-based TAMP. The work in (He et al., 2015) applied *TAMP* using the *LTL* task planner. This approach is done following the *interleaved* approach. 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. In this direction, all the approaches are based on the *interleaved* combination method. The work in (Lagriffoul et al., 2012) and (Lagriffoul et al., 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. The work in (Dantam et al., 2016) proposed the *Iteratively Deepened Task and Motion Planning* method using the *Satisfiability Modulo Theories (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 the case of failure at the geometric level, a logical explanation is computed. This explanation is fed back to the *Answer Set Programming (ASP)* task planner, which prunes entire families of plans leading to similar failures. The cycle repeats until a feasible plan is found.

The main focus of this thesis will be to propose different ways of combining task and motion planning levels with respect to various geometric challenges arising in manipulation problems.

2.7 Combining Task and Motion Planning under Uncertainty

Recently, some approaches have studied how to plan for task and motion levels under uncertainty. A number of strategies has been considered in high-level planning that cope with finding the feasible manipulation plan according to two types of uncertainty, which may arise from the geometry information or the task information, considered.

There are some approaches focusing on uncertainty about the geometry information. The concept of uncertainty in *Solving Simultaneous Task and Motion Planning (STAMP)* was introduced in (Şucan and Kavraki, 2012). The study aims to manage uncertain information (which propagates from motion planning to task planning). Therefore, the *Markov Decision Processes* technique is applied to cope with sequences of actions based on the simultaneously *TAMP* strategy. A probability of feasible plans is reported by a motion planner, which computes and evaluates a trajectory according to its uncertainty with the purpose of action selection. As a result, choosing a path connecting the initial state to the goal state of a robot is performed by comparing probability of action's outcomes. The concept of hierarchical belief-space task and motion planning has been motivated by (Kaelbling and Lozano-Pérez, 2011b) and (Kaelbling and Lozano-Pérez, 2013). They recommend the search space for planning to determine nodes costs (at the task level) based on a probability distribution technique to select the best candidate in order to pursue planning. The proposed search space is not a part of hierarchical planning and this is an external module that can be integrated to the hierarchical structure. Moreover, at required steps, the robot obtains an observation to update its information and estimate the position of the object to perform an executable task.

On the other hand, there are other approaches following the contingent-based task and motion planning to tackle uncertainty about the task information. The work in (Gaschler et al., 2013) has considered incomplete information using the *Planning with Knowledge and Sensing (PKS)* planner and perform contingent planning. The approach considers two main scenarios, namely a *force sensing* and *visual sensing*. The work in (Nouman et al., 2016) has introduced offline-based hybrid conditional task and motion planning, i.e., task planning is foremost performed, and then geometric evaluation is considered. So, the approach incorporates low-level feasibility checks inside conditional planning. It is assumed that actuation actions are deterministic.

Following the direction of contingent-based task and motion planning under uncertainty, this thesis also develops an approach for manipulation problems at which human interactions may be required to complete a manipulation task.

Chapter 3

Semantic Manipulation Knowledge

3.1 Introduction

This chapter introduces semantic manipulation knowledge for robotics system in terms of modelling and reasoning processes. The use of abstract knowledge can enhance the planning capabilities and give more autonomy to the robots to perform tasks. Many approaches exist for the representation of knowledge and one of them is ontology which is used to well structure the knowledge in terms of concepts and relations. Reasoning process over semantic knowledge can be then applied in order that robots infer the particular situation in their workspace. The use of ontologies for robotic manipulation is considered here in combination with task and motion planning (Akbari et al., 2018b) and (Akbari et al., 2015a), and also for physics-based motion planning (Muhayyuddin et al., 2018) and (Muhayyudin et al., 2015).

3.2 Knowledge Representation using Ontologies

In the main, an ontology tackles the concern about reality of things existence and categorizes conceptual knowledge regarding objects in the world upon a particular domain. Ontologies have emerged as a notable technique to explicitly expose knowledge in the artificial intelligence field at expressing the abstract knowledge in the form of concepts along relations. They are able to be encoded and stored in the *Web Ontology Language (OWL)* (Dean and Schreiber, 2002), with the intention to collect and organize ontology information on a world wide accessible database. The *OWL* Language has been allocated to process available information on an ontology model (e.g. connecting individuals or classes together according to relations components) as a useful interpreter, besides being good at representing such information to humans. Ontology models can be done using *Protégé* tool[1], which is a free open-source platform providing an editor to develop domain models and knowledge-based applications. The editor has a good interface to arrange the whole ontology information and classify it into ontology components. It allows to assign numeric values with a specific data type to ontology properties (like object poses) although these values can be alternatively stored in a data base and can be accessed when required. *Protégé* has the capability to reason about ontology information.

OWL enables the structure of knowledge representation by proposing classes, individuals, and object properties:

- *Classes* are collections of various objects sharing common properties.
- *Individuals* are allocated to describe particular instances of classes.
- *Object properties* determine how individuals can be related with each other.

3.3 Manipulation Ontology Components

This section describes abstract semantic knowledge considered for the mobile robots manipulation purposes. The proposed manipulation ontology model, developed with the *Protégé* editor in terms of *OWL*, entails two main classes called *ManipulationWorld* and *ManipulationPlanning*[2], that are represented by $\mathcal{K}_w$ and $\mathcal{K}_p$ respectively. Class *ManipulationWorld*, illustrated in Fig. 3.1, is structured in the following subclasses:

- *Obstacles*: Class that retains necessary information with respect to all the obstacles in the robot workspace. The information included comprises geometry and physical properties like objects masses, friction coefficients, etc. The type of obstacles *ObstacleType* is classified into two categories: *FixedObstacle*, for those obstacles with which a robot must avoid collisions, and *ManipulatableObstacle* for those with which a robot can interact. Class *ObstaclePose* includes information on the obstacle pose coded as a transformation matrix composed of orientation and translation values for each object.
- *Path*: Class representing predefined symbolic paths between different regions in the workspace.
- *Regions*: Class used to represent various regions. Subclasses *InitialRegion* and *GoalRegion* are used to represent the initial and the goal regions of the robots, respectively. The subclass *ManipulatableRegion* is used to represent the region associated to manipulatable obstacles where the robot must be placed in order to perform push/pull actions.
- *Robots*: Class used to represent the robots and their properties. Geometric parameters of the robots are stored by the subclass *KinematicProperties*; differential properties of the robot such as bounds on forces, torques, velocities, and accelerations (global properties

[1]http://protege.stanford.edu/
[2]OWL files are accessible at: https://sir.upc.edu/projects/ontologies/.

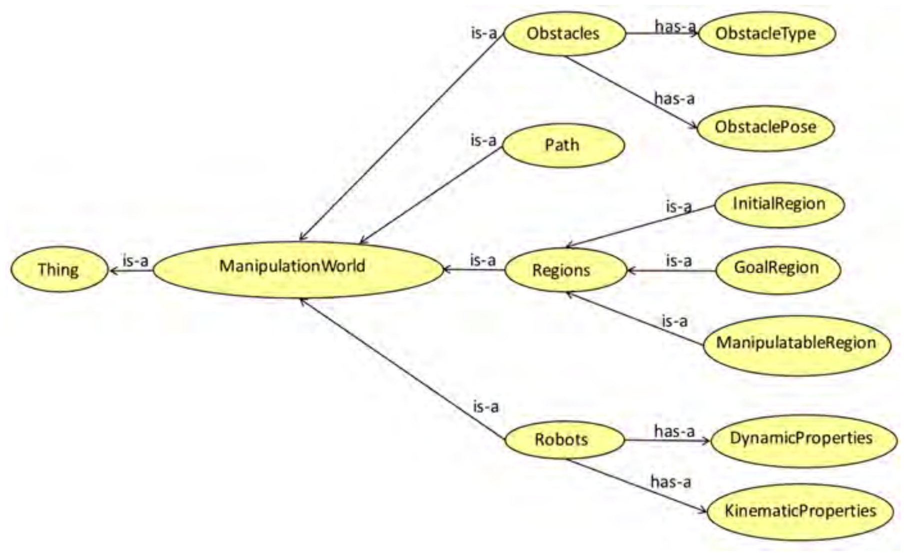


Figure 3.1: The taxonomy of the class *ManipulationWorld*
.

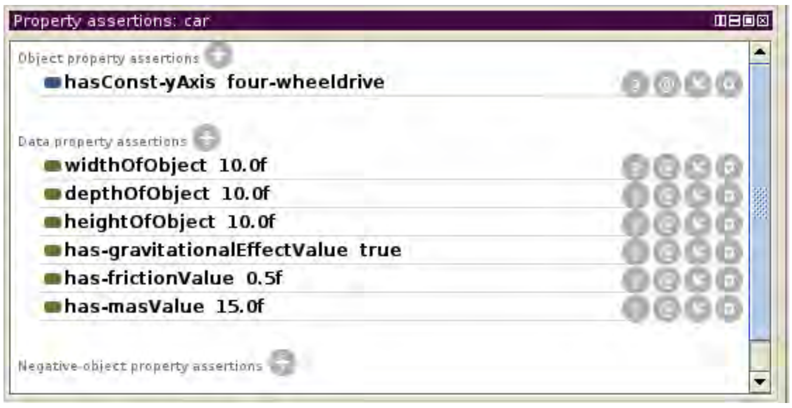
that condition the maximum capacity of the robot) are stored by the subclass *DynamicProperties*.
The access to the ontological knowledge can be done using *Description Logic (DL)*. For instance, the following *DL* description represents the relations for robot instances *Robot* explaining that each robot has dynamics properties which involve force bounds, velocity bounds, and acceleration bounds.

$Class \;\; Robots := Robot$
$\exists hasSuperclass(Robot, ManipulationWorld),$
$\wedge \exists hasDynamicProperties(Robot, DynamicProperties),$
$\wedge \exists isDynamicProperties(ForceBounds, DynamicProperties),$
$\wedge \exists isDynamicProperties(VelocityBounds, DynamicProperties),$
$\wedge \exists isDynamicProperties(AccelerationBounds, DynamicProperties).$

where $\wedge$ and $\exists$ represent *conjunction* and *exist*, respectively.
In a similar way, the next *DL* description illustrates the dimension of each sphere-like robot, the response to gravity, the friction, mass, and color values, respectively.

$Class \;\; Robots := Robot$
$\exists hasRadius(Robot, Value),$
$\wedge \exists hasHeight(Robot, Value),$
$\wedge \exists hasResponseToGravity(Robot, Value),$
$\wedge \exists hasFriction(Robot, Value),$
$\wedge \exists hasMass(Robot, Value),$
$\wedge \exists hasColor(Robot, Value).$

In a similar way, the information of manipulatable obstacles is classified in the ontology. As an example, the semantic properties of the car-like object stored in the *OWL* are depicted by Fig. 3.2, including the manipulation constraint, the dimensions, and physical attributes such as mass, friction, and the effect of gravity on the object.



Property assertions: car

Object property assertions

hasConst-yAxis four-wheeldrive

Data property assertions

widthOfObject 10.0f

depthOfObject 10.0f

heightOfObject 10.0f

has-gravitationalEffectValue true

has-frictionValue 0.5f

has-masValue 15.0f

Negative object property assertions

Figure 3.2: Screen shot of the *Protégé* editor showing the semantic properties of a car-like object.

On the other hand, class *ManipulationPlanning*, also illustrated in Fig. 3.3, is structured in the following subclasses:

- *ProblemQueryConditions*: Class that uses the information of *InitialState* and *GoalState* classes regarding the locations of the robots at the initial and goal states, respectively.
- *ActionProperties*: Class used to define the different actions templates and bind them with their preconditions and side effects.
- *Predicates*: Class that represents the predicates like *At* which contains where a robot is located.

3.4 Reasoning Process on Manipulation Knowledge

Reasoning over the *OWL* knowledge is done using the *KnowRob* tool (Tenorth and Beetz, 2009), a potent knowledge processing tool that enables a flexible access to the *OWL* knowledge and the *SWI-Prolog* library (Wielemaker et al., 2012). It is mainly developed in the Prolog language and provides fundamental predicates to fetch knowledge, e.g., the query *owl_subclass_of(?SubClass, ?Class)* explores all available subclasses of a class, *owl_individual_of(?Indv, ?Class)* seeks to list

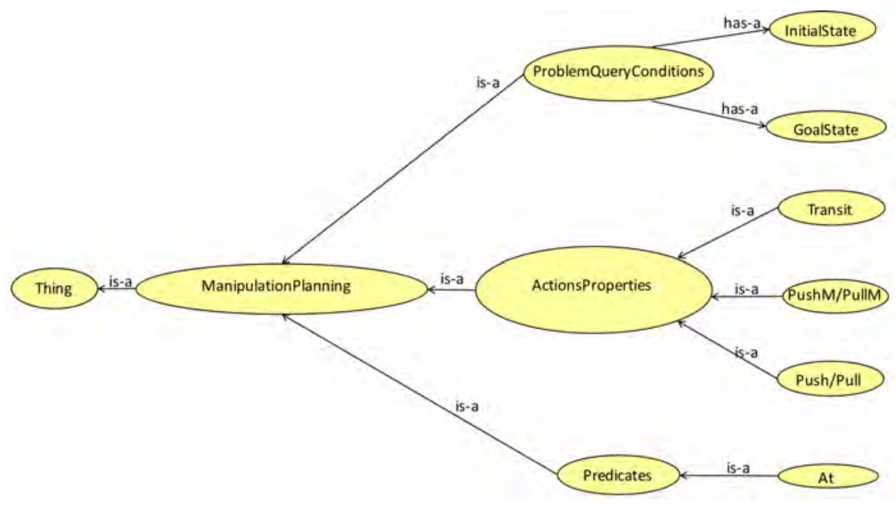


Figure 3.3: The taxonomy of the class *ManipulationPlanning*.

all individuals of a class, and *class_properties(?Class, ?Properties, ?Value)* determines the value of a class under particular properties. The tool has been extended with more predicates tailored to extract the necessary information. Some of them are defined as follows:

- *object_classification(?Obj, ?ObjType, ?Const)*: Given an object *Obj* returns the type of the associated object *ObjType* and the constraints may hold by evaluating its category and manipulation features. Those manipulatable objects that are too heavy for the robot to be manipulated are set to fixed.
- *manipulatable_region(?Obj, ?ManipRgns)*: Given a manipulatable object *Obj* computes the set of associated manipulatable regions *ManipRgns* taking into account the manipulation constraints, if any.
- *object_properties(?Obj, ?PhysicalProps, ?Dimension)*: Given an object *Obj* returns its physical properties *PhysicalProps*, including mass and friction values, as well as gravitational effect and the dimension *Dimension* of the object.
- *robot_properties(?DynamicsProps, ?KinematicProps)*: Returns the dynamics properties of the robot (forces and velocities limits) in *DynamicsProps*, and the kinematic properties (joint limits) in *KinematicProps*.
- *action_feasibility(?Rob, ?Obj, ?Action)*: Given a robot *Rob* and a manipulatable object *Obj* returns the corresponding feasible actions with respect to robot capabilities and object features.

3.5 Applications of Semantic Knowledge in Manipulation Planning

The proposed ontological knowledge has been used to enrich the combination of task and motion planning, and also to enhance physics-based motion planning to address different sort of manipulation problems. For the combination of task and motion planning, the usage will be discussed in details in Chapters 4 and 5, for physics-based motion planning, which is not the main scope of the thesis, it is briefly commented next.

Regarding applying semantic knowledge in motion planning, a simple scenario presented in (Muhayyudin et al., 2015) has been implemented that concentrates on how physics-based motion planning makes use of the ontological knowledge to manage a situation involving a single manipulatable obstacle. It consists of: a) the walls of a room which are defined as fixed bodies; b) a spherical shaped robot with two degrees of freedom; and c) a car-like manipulatable object which is defined as a constraint-oriented manipulatable body that can only be moved in the forward and backward directions. Using the knowledge-based reasoning over the physical properties of the car, the robot knows how much force it needs to apply when it is going to manipulate the car. The goal region $\boldsymbol{Q}_{goal}$ is occupied by the car-like object, so no collision-free path exists, i.e., the robot needs to push the car-like manipulatable object away in order to reach $\boldsymbol{Q}_{goal}$. This examples uses a physics-based motion planner based on *KPIECE*. The sequence of snapshots of the solution is represented in Fig. 3.4.

3.6 Conclusions

This chapter represented semantic manipulation knowledge by modelling it using ontologies and proposed a reasoning process over the ontologies in order to provide better information for manipulation planning. The proposed framework has been applied in the combined task and motion planning approach introduced in this thesis, and the usage will be discussed in Chapters 4 and 5. Out of the scope of this thesis, it has been used also in physics-based motion planning where a robot needs to interact with objects in workspace.

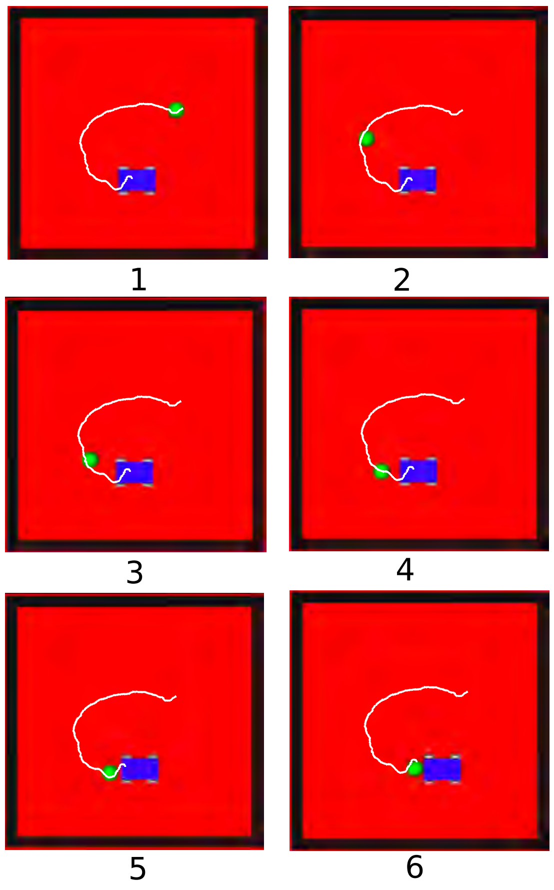


Figure 3.4: Simulation results of a physics-based motion planner using knowledge.

Chapter 4

GraphPlan-based Task and Motion Planning for Mobile Robots

4.1 Introduction

This chapter copes with the combination of task planning and physics-based motion planning levels using ontological knowledge for mobile robots in the scope of *NAMO* problems. Daily manipulation tasks require a robot to become more capable, robust, as well as autonomous in order to carry out various manipulation actions, e.g., pushing different sort of objects holding unique characteristics in human-like environments. In solving a given manipulation problem, task (high-level) beside motion (low-level) planning are required, and their combination plays a crucial role in realizing a solution plan in terms of finding a sequence of actions and a way of execution them. Manipulation actions comprise contacts between a robot and manipulatable objects, so motion planning has to be aware of the possible interaction among rigid bodies. To deal with this issue, a physics-based reasoning engine is employed that enables the access to such information aiming to evaluate feasibility or effect of actions.

Contributions. Two type of combinations of task and physics-based motion planning using the ontological knowledge are proposed that have been presented in the research articles (Akbari et al., 2015b) and (Akbari et al., 2015a). The task planner used is based on modified versions of the *GRAPHPLAN* algorithm. The prior work focuses on solving task first with different plans, and then calling a physics-based motion planner to analyze the manipulation plan in order to obtain the best one in terms of power. On the contrary, the latter one presents an approach that interleaves physics-based motion planning within the task planner simultaneously.

4.2 Problem Formulation

Consider the *NAMO* manipulation problems where there is no direct collision-free path between the initial and the goal state of the robot and it needs to push some objects away. Mobile robots are considered that can perform two types of actions: to move around freely and to push manipulatable obstacles. Then, the problem to be tackled is to find a feasible sequence of actions to bring the robot, among fixed and manipulatable obstacles, from an initial region towards a goal one, such that it has a good behavior in terms of power. Collisions with fixed obstacles are not allowed while, if necessary, manipulatable obstacles can be pushed away. The following regions in the workspace, available in the knowledge $\mathcal{K}_w$, are considered:

- *Initial region*: It is the region where a robot is initially located.
- *Goal region*: It is the region where a robot is going to be located eventually.
- *Connection region*: There could be different zones, which can be different rooms or particular regions, in the robot workspace. So, *connection region* is the region in which two separated zones are connected to each other, e.g., if the robot is located in the zone one, it can travel to the zone two through this region.
- *Manipulatable region*: It is the region in which a robot must be located in order to interact with a manipulatable object.

Those regions which are occupied by some objects are called *critical regions*. It is assumed that there is a path between two regions located in the same zone and it is activated if the region is not *critical region*. Otherwise, a push action is required to make it activated. The literals assumed here are summarized below:

- *HasAccess(Robot, Path)*: Holds true if the *Robot* has access to *Path*.
- *At(Robot, Region, Path)*: Holds true if the *Robot* is located at *Region* through *Path*.
- *In(Obj, CriticalRegion)*: Holds true if a manipulatable object *Obj* is positioned in a critical region *CriticalRegion*.

The following symbolic action templates are considered to deal with the *NAMO* problems:

- *Move(Robot, Region, ThroughPath, FromRegion, FromPath):*

  Precondition: *HasAccess(Robot, ThroughPath), At(Robot, FromRegion, FromPath)*

  Add: *At(Robot, Region, ThroughPath)*

  Delete: *At(Robot, FromRegion, FromPath)*

- *Push(Robot, Obj, ManipRegion, CriticalRegion, ToAccessPath, ThroughPath):*

  Precondition: *In(Obj, CriticalRegion)*, *At(Robot, ManipRegion, ThroughPath)*

  Add: *HasAccess(Robot, ToAccessPath)*

  Delete: *In(Obj, CriticalRegion)*

It is worth noting that all the literals and actions along their conditions are modelled in the manipulation ontology and can be accessible during planning when required.

4.3 Combining Task and Physics-based Motion Planning in Two Phases

In this approach, the combination of task and motion planning is done in two steps, i.e., high-level plans are first retrieved and physics-based motion planning is used afterwards to evaluate the symbolic plans.

4.3.1 Retrieval of High-Level Plans

The standard *GRAPHPLAN* algorithm builds the *planning graph* until the last state level satisfies all the goal conditions, and then the backtracking process searches for the sequence of actions satisfying all the constraints. If found, the solution plan is the shortest sequence of actions that solve the problem, irrespective of its feasibility or cost (e.g. the solution may content a push action which cannot be executed due to the weight of the object to be pushed). To allow the *GRAPHPLAN* algorithm to retrieve all the possible plans, a lightweight reasoning process has been implemented to identify all goal conditions (called possible conditions) that should be met at the initial and the goal states. The possible conditions representing all ways of satisfying the goal conditions. The *planning graph* grows until a state level is found that satisfies all possible conditions and the backward search is then applied from each possible condition in order to find all alternative plans.

As an example, Fig. 4.1 shows a two-room *NAMO* scenario with a wall as a fixed obstacle, three manipulatable obstacles (A, B, and C with associated regions from where they can be pushed) and an initial and a goal region.

The problem in Fig. 4.1 shows that the initial region has access to two alternative paths, *P1* and *P3* (i.e. the initial action of a plan may be to move the robot towards object A following path *P1* or alternatively moving towards object C following path *P3*). Therefore, these paths become possible conditions for the initial state that can be represented by *HasAccess(robot, p1), HasAccess(robot, p3)* in the initial state of the task planner.

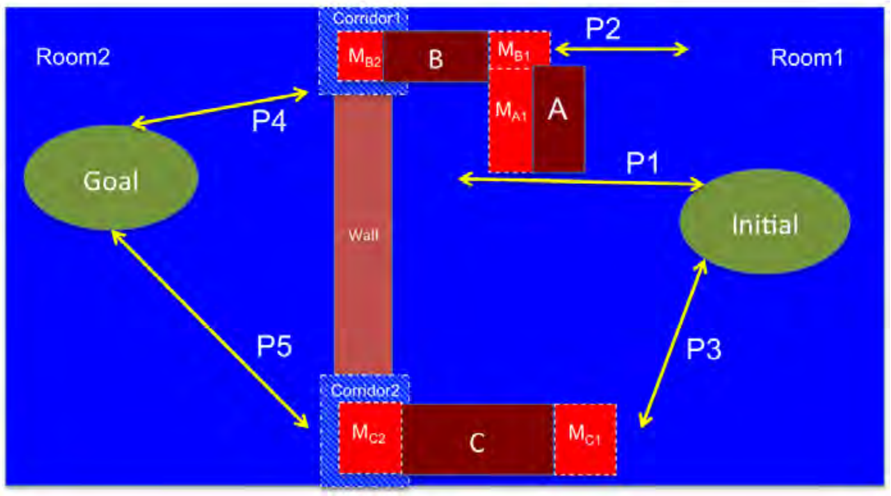


Figure 4.1: A *NAMO* manipulation problem where a robot must find the best path from an initial region to a goal one avoiding the fixed wall in the middle and pushing the manipulatable objects A, B and C, if necessary, to clear the corridors (dashed regions). Here, corridor 1 and corridor 2 are the connecting regions to connect both rooms together.

A similar procedure can be applied for the goal conditions as well, where it can be seen that the goal region can be reached through two alternative paths, *P4* and *P5*. Then:

G =[At(robot, goalregion, p4), At(robot, goalregion, p5)]

For the example in Fig. 4.1, some of th information of the resulting *planning graph* generated by this variant of *GRAPHPLAN* is shown in Fig. 4.2. At each state level, predicates are labeled with numbers, and at each action level actions are labeled with the predicates they connect (not counting the maintenance action that applies to all the predicates). The search space provides five state levels. From the last state level, backtracking is done starting at two possible goal conditions, *At(robot, goalregion, p4)* and *At(robot, goalregion, p5)*. The states encountered during backtracking in which the robot plays a role have been highlighted in Fig. 4.2 and the corresponding symbolic plans are depicted in Fig. 4.3. They are achieved from backtracking through each goal conditions in the goal layer of the *planning graph*.

4.3.2 Physics-Based Reasoning on High-Level Plans

A physics-based motion planner will be used to plan both move and push actions, and an associated cost, detailed in the following subsection, will be set to evaluate the feasibility of a symbolic plan.

The physics-based reasoning about a high-level plan (composed of a sequence of move and

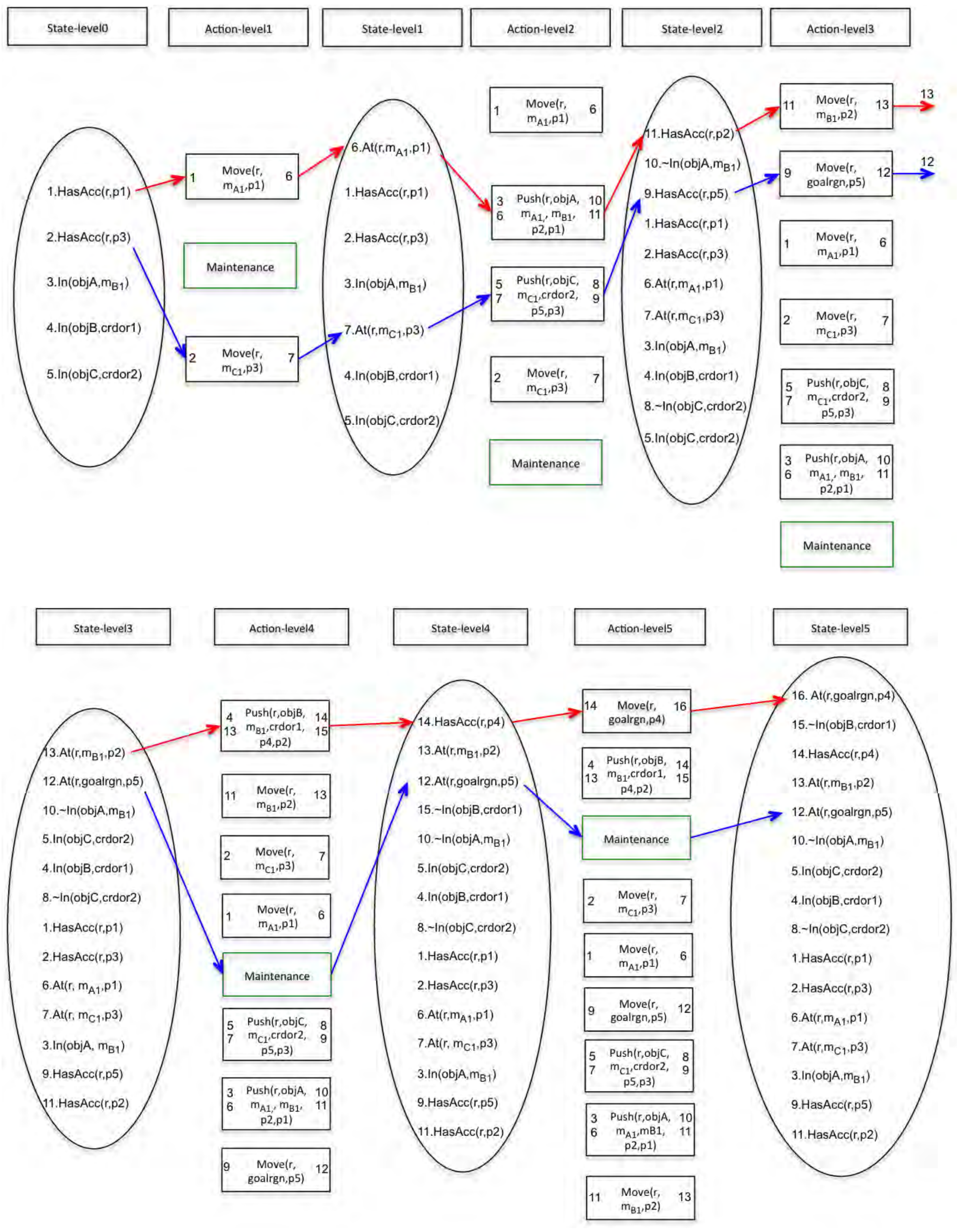


Figure 4.2: Some of the information of the search space of *GRAPHPLAN*. Here, some of the conditions and the first three parameters of the move action are depicted.

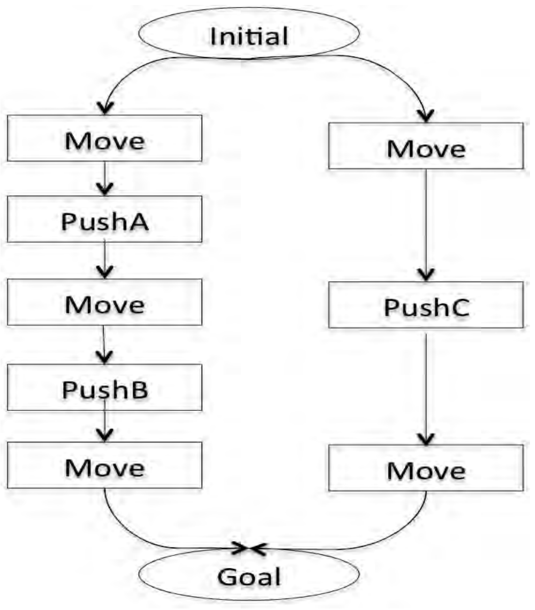


Figure 4.3: Plan A (left) and plan B (right) resulting from the modified *GRAPHPLAN* algorithm.

push actions) involves the dynamical properties of the robot and of the environment, such as masses of the objects, required manipulation forces, friction, etc. As discussed in Chapter 3, dynamical properties for the manipulation task can be queried by Prolog predicates (like *object_properties* or *robot_properties*) through the ontological knowledge and applied for the motion planner. The feasibility of a plan will be set according to its cost that will be computed according to the following *cost functions*:

- *Push cost*: It is the total amount of power consumed to complete the push action (i.e. to clear the region the object is occupying) by applying repetitively the same force:

$$c_p = \frac{n\mathbf{f}\mathbf{d}}{\Delta t}, \tag{4.1}$$

  where $\mathbf{f}$ represents the force applied by the robot for Δt duration, $\mathbf{d}$ is the corresponding displacement covered by the object, and n is the number of times the force is applied in order to complete the push action and clear the region. If the object is too heavy to be pushed by the robot, then this cost will be set to infinity.

- *Move cost:* It is the total amount of “action" (i.e. the dynamical attribute of a physical system that considers the history of moves and that has units of *Newton-meter-second*):

$$c_m = \sum_{i}^{n} |\mathbf{f_i}| \Delta t_i \varrho_i \tag{4.2}$$

  where $\mathbf{f}_1, \ldots, \mathbf{f}_n$ are the controls (forces) to be applied to follow the path and ϱ_i represents the distance covered when applying force $\mathbf{f}_i$. If the motion planner is not able to find a path, then this cost will be set to infinity.

Let C_p^j and C_m^j be the total push and move costs of plan j:

$$\begin{aligned} C_p^j &= \sum_{k=1}^{N_{p_j}} c_{p_k}^j \\ C_m^j &= \sum_{k=1}^{N_{m_j}} c_{m_k}^j \end{aligned} \tag{4.3}$$

with N_{m_j} and N_{p_j} being the total number of push and move actions of the jth plan, respectively. Then, based on these costs, a *Task-feasibility* parameter is defined for each plan as follows:

$$\alpha_j = \mu \frac{C_p^j}{\max_{\forall k \in [1,N]} C_p^k} + (1-\mu) \frac{C_m^j}{\max_{\forall k \in [1,N]} C_m^k} \tag{4.4}$$

with $\mu \in [0,1]$ being a weighting factor and N the number of alternative plans.

The best plan will be the one with the lowest *Task-feasibility* parameter.

4.3.3 Simulation Results and Discussion

The *KPIECE* motion planning algorithm (Şucan and Kavraki, 2009) (with the physics-based state propagator based on *ODE*) has been used, and some snapshots of the executions of both plans are illustrated in Fig. 4.4 and Fig. 4.5. Table 4.1 shows the cost of each action and the *Task-feasibility* parameter of each plan using $\mu = 0.5$. The total move and push costs of plan A are lower than those of plan B and result in a lower *Task-feasibility* parameter. Plan A is, therefore, the selected one. Note that plan B has less actions and would have been the unique plan found by the standard *GRAPHPLAN* algorithm.

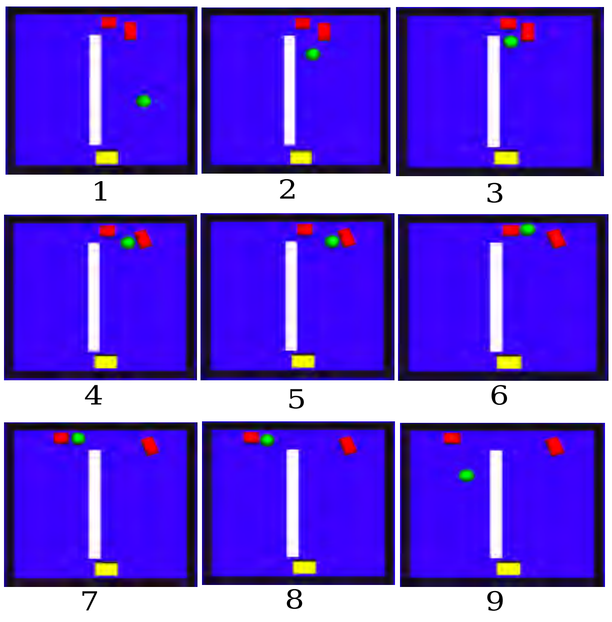


Figure 4.4: Simulation results of the execution of plan A for the example in Fig. 4.1. The solution can be visualized in `https://sir.upc.edu/projects/kautham/videos/efta15_TaskPlanning.mp4`

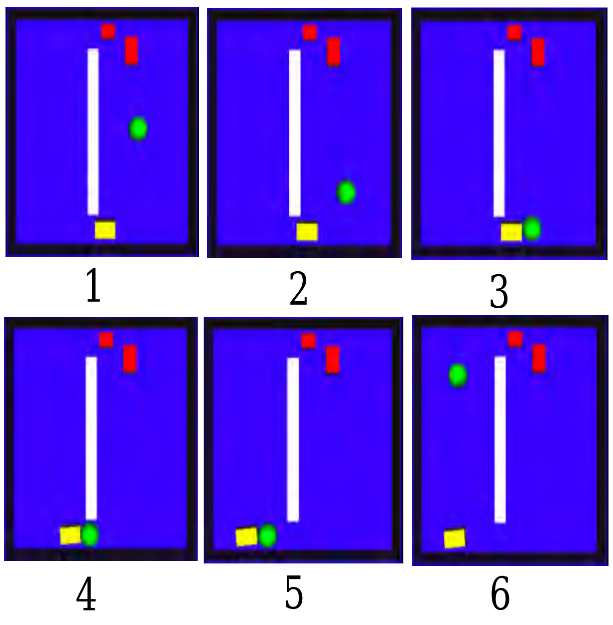


Figure 4.5: Simulation results of the execution of plan B for the example in Fig. 4.1.

Cost	**Plan A**			**Total**
Move (N·m·s)	0.07903	0.01401	0.03955	0.13259
Push (J/s)	7.46064	9.46279		16.92343
Task-feasibility parameter	0.91585			
Cost	**Plan B**			**Total**
Move (N·m·s)	0.05736	0.09372		0.15108
Push (J/s)	17.73759			17.73759
Task-feasibility parameter	1.0			

Table 4.1: Evaluation of cost plans.

According to the way of combining task and motion planning levels, the proposed approach collects primarily high level symbolic plans, and afterward calls the physics-based motion planner in order to reason over the plans. Therefore, it could be expensive to check all possible plans which may have, e.g., some infeasible actions that can be identified while the planning process is taken place.

4.4 Simultaneous Task and Physics-based Motion Planning using Reasoning Process

Unlike the previous section, this section presents an approach of combined task and motion planning using the low-level along the high-level reasoning process in order to prune infeasible and unnecessary symbolic actions when required. This method may lead to cut-off infeasible and dispensable task branches while task planning is taken place comparing to the approach presented in the previous section.

4.4.1 Reasoning Process about Manipulation Actions

The use of reasoning process plays an important role in integrating task and motion planning in order to find a geometrically feasible manipulation plan. To illustrate this issue, the *NAMO* example represented in Fig. 4.6 is assumed in which there are two rooms separated by a corridor and a mobile manipulator that must traverse to move from the initial to the goal region. Several paths are also considered among the regions of the robot workspace as follows:

- *P1, P2,* and *P3*: Paths that connect the initial region to the regions m_D, m_{A1}, and m_{A2}, respectively.

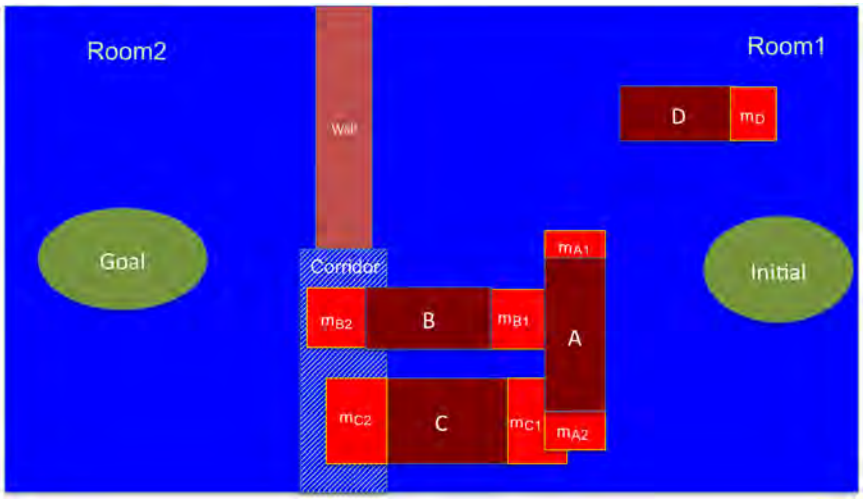


Figure 4.6: A *NAMO* manipulation problem where a robot must find a feasible path to reach the goal region avoiding the fixed wall. corridor is a connecting region to connect both rooms together.

- *P4* and *P5*: Paths that connect the region m_D to the regions m_{A1} and m_{A2}, respectively.
- *P6* and *P7*: Paths that connect the region m_{A1} to the regions m_{B1} and m_{C1}, respectively.
- *P8* and *P9*: Paths that connect the region m_{A2} to the regions m_{B1} and m_{C1}, respectively.
- *P10*: Path that connect the region m_{B1} to the goal region when the robot is in room 2.
- *P11*: Path that connect the region m_{C1} to the goal region when the robot is in room 2.

To achieve the manipulation goal, the robot needs to push away some objects in order to free its path towards the goal. The following constraints hold for the manipulatable objects: they can only be pushed from their manipulatable regions as defined in the knowledge and, according to them, along the x or y directions. Regarding the possible pushing actions, it can be seen that, for instance, pushing object A from m_{A1} becomes an ineffective action because it does not clear the access to either manipulatable region m_{B1} or m_{C1}. Thus, such type of actions should be identified and omitted in the task planning search space.

Therefore, three types of reasoning processes are proposed to evaluate manipulation actions: essential, feasibility, and geometric reasoning. They are applied while the *planning graph* is constructed.

The *essential* reasoner determines whether an action is relevant or not. The reasoner prunes those actions which are not relevant for the task to be solved, i.e., those that are inessential and applied to objects which do not occupy any regions in the workspace. This is done for *Push* actions by evaluating, in a given action-layer, whether the post-conditions of a non-maintenance action can only lead to actions already found in the action-layer and the corresponding manipulated object in the action does not occupy any critical regions. If this is the case, the corresponding action is marked as inessential and pruned from the *planning graph*. Also, the pre-conditions of the pruned action are deleted, as well as all the maintenance branches leading to them and they do not appear again. In this way, only fruitful actions are kept and any dispensable action branch is removed.

The *feasibility* reasoner evaluates the feasibility of an applicable action by determining whether the robot is capable or not to perform it using the knowledge about the robot specifications and manipulated objects using the Prolog predicate *action_feasibility* and with no call to the motion planner. It is considered for the push actions. If the robot is not able to perform the task, the action is considered as infeasible and it is pruned from planning along its branches.

The *geometric* reasoner calls a physics-based motion planner to check whether the effects of an action are reachable or not. If the motion planner is not able to find a motion, the action is considered as unattainable and is pruned from the planning process along its branches.

4.4.2 Planning using Reasoning Process

The proposal of combining task and motion planning using the reasoning process is sketched in Algorithm 1. It takes the knowledge $\mathcal{K}_w$ and $\mathcal{K}_p$ concerning the robot workspace and the planning information and returns the solution plan π.

The core of the computation is to appraise each selected action along the corresponding effects by the proposed reasoning process in order to prune dispensable actions. This results in a smaller *planning graph*, making both the construction phase and the search phase more efficient. When the reasoning process determines that an action is inessential, infeasible, or unattainable, then it is pruned. Fig. 4.7 illustrates the constructed *planning graph* for the example represented in Fig. 4.6. For the sake of saving space, only the relevant information has been exposed (e.g. maintenance actions are neglected).

The *planning graph* is established with the initial state defined in the knowledge ($\mathcal{K}_p.\boldsymbol{s}_{\text{init}}$) [line 1] and the construction phase procedure is launched until a layer is found that satisfies the goal constraints [line 26]. Action space $\boldsymbol{A}$ is available on demand from the ontological knowledge. At every iteration i, the set $\boldsymbol{A}_{\text{i}}$ of actions whose pre-conditions appear in the previous state-level $\boldsymbol{S}_{\text{i-1}}$ is selected [Line 5] and then evaluated [Lines 6-22]. The action is going to be forwarded to the sequence of reasoning process:

- *essentialReasoner*($\boldsymbol{a}$,$\mathcal{K}_w$) checks whether an action is essential or not [lines 7-10], e.g., the action *PushD* from m_D becomes inessential regarding the existing actions *PushA* from m_{A1} or from m_{A2} since they can always be done irrespective of performing the *PushD* action.
- *feasibilityReasoner*($\boldsymbol{a}$,$\mathcal{K}_w$) evaluates the feasibility of an action by determining whether the

Algorithm 1: Task and motion planning using efficient reasoning

```
inputs : $\mathcal{K}_w$ and $\mathcal{K}_p$
output: The solution plan $\pi$
1  $\boldsymbol{S}_0 \leftarrow \mathcal{K}_p.\boldsymbol{s}_{\text{init}}$
2  $\boldsymbol{A}_{\text{valid}} = \emptyset$
3  $i \leftarrow 1$
4  while $i < m$ do
5      $\boldsymbol{A}_{\text{i}} \leftarrow \{\boldsymbol{a} \in \mathcal{K}_p.\boldsymbol{A} \mid \text{precond}(\boldsymbol{a}) \subseteq \boldsymbol{S}_{\text{i-1}}\}$
6      foreach $\boldsymbol{a} \in \boldsymbol{A}_i$ AND $\boldsymbol{a} \notin \boldsymbol{A}_{valid}$ do
7          essentialReasoner($\boldsymbol{a}$,$\mathcal{K}_w$)
8          if $\boldsymbol{a}$ is inessential then
9              pruneActions($\boldsymbol{a}$)
10             continue
11         feasibilityReasoner($\boldsymbol{a}$,$\mathcal{K}_w$)
12         if $\boldsymbol{a}$ is infeasible then
13             pruneActions($\boldsymbol{a}$)
14             continue
15         geometryReasoner($\boldsymbol{a}$,$\mathcal{K}_w$)
16         if $e$ is unattainable then
17             pruneActions($\boldsymbol{a}$)
18             continue
19         $\boldsymbol{A}_{\text{valid}} \leftarrow \boldsymbol{a}$
20     $\boldsymbol{A}_{\text{i}} \leftarrow$ maintenanceActions($\boldsymbol{S}_{\text{i-1}}$)
21     $\boldsymbol{S}_{\text{i}} \leftarrow (l \mid l \in \text{effects}(\boldsymbol{A}_{\text{i}}))$
22     $M \leftarrow$ appendMutexes($\boldsymbol{A}_{\text{i}}, \boldsymbol{S}_{\text{i}}, \boldsymbol{S}_{\text{i-1}}$)
23     if goalFound($\mathcal{K}_p.\boldsymbol{s}_g, \boldsymbol{S}_i$) then
24         $\boldsymbol{S}_{\text{g}} = \boldsymbol{S}_{\text{i}}$
25         backtrack($\mathcal{K}_p.\boldsymbol{s}_{\text{g}}, M$)
26         if constraintSatisfaction($\boldsymbol{S}_{\text{g}}, \boldsymbol{S}_0, M$) then
27             $\pi \leftarrow$ extractPlan($\boldsymbol{S}_{\text{g}}, \boldsymbol{S}_0, M$)
28             return $\pi$
29     $i \leftarrow i + 1$
30 return NULL
```

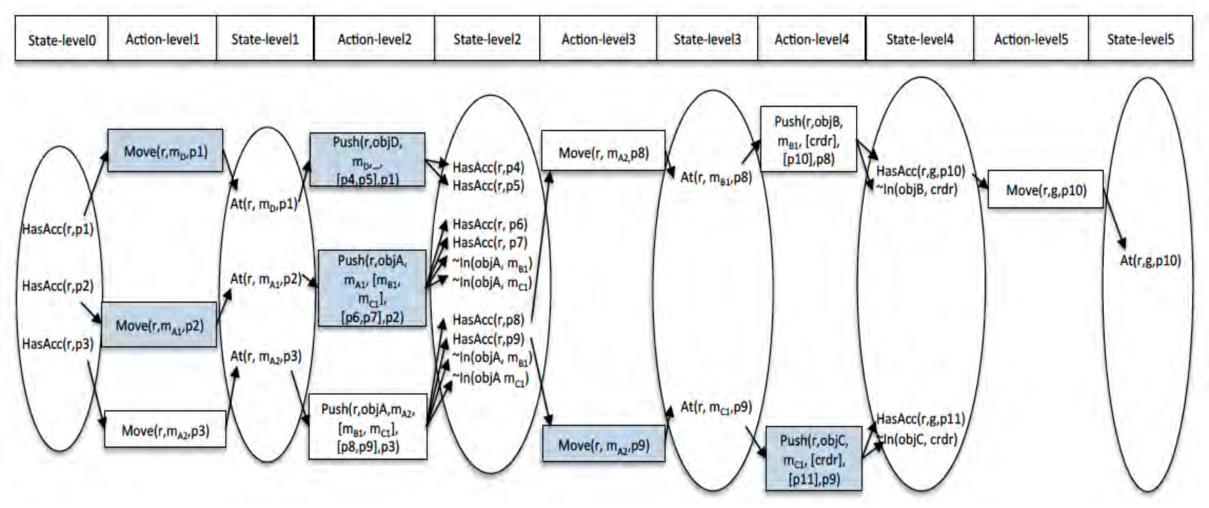


Figure 4.7: Some of the information of the constructed *planning graph*. Pruned actions are specified in blue. Here, the first three parameters of the move action are depicted.

robot is capable or not to perform it [lines 11-14], e.g., the action *PushC* is infeasible with respect to the robot's capability since it is a very big and heavy object.

- *geometryReasoner*($\boldsymbol{a}$,$\mathcal{K}_w$) checks whether the effects of an action are reached or not [lines 15-18], i.e., the action *PushA* from m_{A1} is ineffective because after pushing the object downward, regions m_{B1} and m_{C1} are not freed because the object collides with the wall and gets stuck over these regions.

The action that is not valid is pruned using function pruneActions($\boldsymbol{a}$), that deletes the action from the *planning graph* as well as the pre-conditions and all the maintenance branches leading to them. The actions that are valid are stored in the set $\boldsymbol{A}_{\text{valid}}$ [line 19], in order not to evaluate them again if the action further appears, and maintenance actions are later added [line 20]. Then the next state-level is created from the set $\boldsymbol{A}_{\text{i}}$ and the constraints between actions and literals is computed by function *appendMutexes(*$\boldsymbol{A}_i$, $\boldsymbol{S}_i$, $\boldsymbol{S}_{i\text{-}1}$*)* as done in the standard *GRAPHPLAN* algorithm.

In the case of goal conditions are met, backtracking is performed. If the initial state-level $\boldsymbol{S}_0$ is reached in this backtracking process and all the constraints are met the plan is extracted [lines 25-27].

4.5 Simulation Results and Discussion

Simulation results corresponding to the scenario represented in Fig. 4.6 are shown in Fig. 4.8 as a sequence of snapshots. *SyCLoP-RRT* (Plaku et al., 2010) is used as kinodynamic motion planner for physics-based planning because recent benchmarking study of the kinodynamic motion planners for physics based planning (Muhayyuddin et al., 2016) shows that *SyCLoP-RRT* computes better path in terms of power and smoothness. The solution plan and pruning actions are found in Fig. 4.7 and the average planning time is about 35 seconds for solving the manipulation problem.

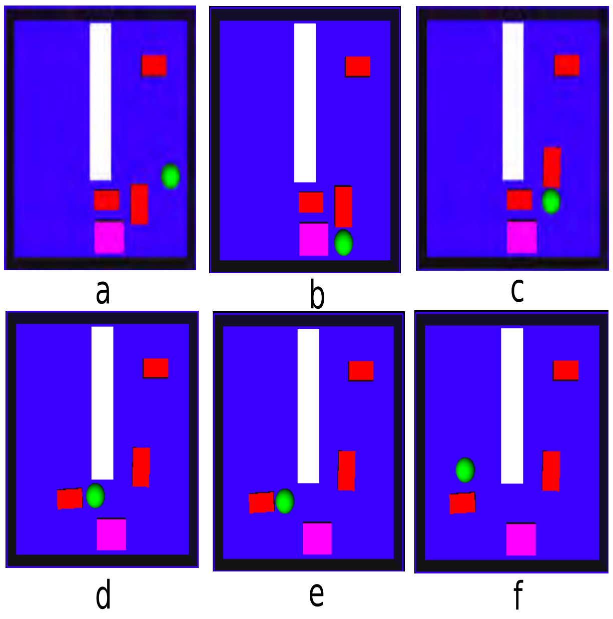


Figure 4.8: Simulation results of the execution for the example in Fig. 4.6.

4.6 Implementation Framework

The proposed framework for task and motion planning, shown in Fig. 4.9, consists of a knowledge module, a task planing module, and a motion planning module. All of them communicate with each other through a *Robot Operation System* (*ROS*, http://www.ros.org/, (Quigley et al., 2009)) communication layer:

- The knowledge is coded in the form of the *OWL* ontology and it consists of: a) knowledge about the world, i.e., information on the type of objects (manipulatable or fixed) and if manipulatable, on the regions from where they can be pushed (manipulatable regions), and their poses (position and orientation); b) physical properties (objects masses, friction coefficients, minimum manipulation forces, etc.) as represented in Chapter 3.

- The task planning module: it employs high-level reasoning process and uses the variants of the *GRAPHPLAN* approach.

- The motion planning module is used to determine the effect of each action in the dynamic world and to execute the plan. The implementation of this module is performed using *The Kautham Project* (Rosell et al., 2014) (`https://sir.upc.edu/projects/kautham/`) that is a motion planning tool based on the *Open Motion Planning Library* (*OMPL* (Sucan et al., 2012)) which provides the implementation of many sampling-based planners like *RRT* and *KPIECE*. The dynamic simulations are performed using *ODE*. As the motion planner has access to the physical properties of objects through the knowledge, it will be able find a motion with a low cost.

- The communication layer shares information between different layers. This layer involves ROS (Quigley et al., 2009) to provide the basic communication protocols, and the *KnowRob* (Tenorth and Beetz, 2009) and the *Json-prolog* library (provided by *KnowRob*) to enable the access to the ontological knowledge and Prolog predicates through the *ROS* communication protocols.

4.7 Conclusions

This chapter has proposed two frameworks to take into account lightweight high-level and physics-based reasoning for the *NAMO* manipulation problems. Two modified versions of the *GRAPHPLAN* algorithm are proposed. The prior one collects all possible symbolic plans, and, afterwards a physics-based reasoning process is applied to determine the feasibility of each plan in terms of the action costs. The latter one is based on simultaneous task and motion planner that has been computationally more efficient. This is achieved by reasoning over the action and pruning those that are dispensable. This reasoning process is performed at two levels. A high level that allows to find those actions that are not relevant and a low-level using a physics-based motion planner that allows to find those that are either infeasible or unattainable.

The proposals have been implemented and illustrated with some simple examples. The present proposal considers a mobile robot with move and push action capabilities. The limitation of the current proposals are when the manipulation problems become more complex in terms of more number of mobile robots or manipulation actions, because then they can be computationally expensive in terms of number of calls to the motion planner.

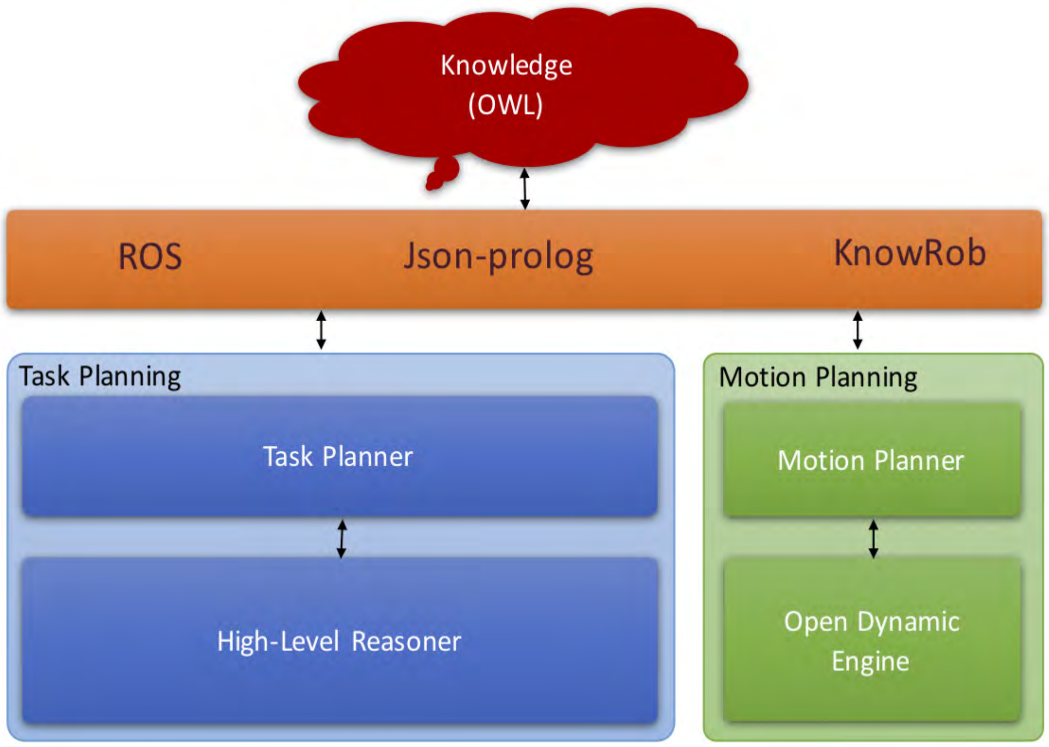


Figure 4.9: Implementation framework for task and motion planning using physics-based reasoning

Chapter 5

Heuristic-based Task and Motion Planning for Collaborative Multiple Mobile Manipulation

5.1 Introduction

The chapter is going to deal with the *NAMO* problems in which multiple mobile robots are required to collaborate or share the tasks with each other in order to satisfy the goals of the problem. The fulfillment for robotic systems composed of several mobile robots moving in environments with fixed and movable obstacles is a great challenge, mainly due to the need to find a sequence of motions that are feasible, i.e., motions that do not make the robot collide with fixed obstacles and that if necessary interact with movable obstacles to remove them and clear the path. If it is assumed that mobile robots can perform *Transit*, *Push*, and *Pull* actions, the problem can be efficiently tackled if a smart combination of high-level (symbolic) planning and low-level (geometric and physical) planning is proposed. At the high level, different symbolic task planners developed by the artificial intelligence community have been proposed. Among them, the Fast Forward planner, that is a heuristic-based planner, has demonstrated to be very efficient. At the low level, different motion planners have been proposed. Among them, the physics-based motion planners allow to plan motions of the robot from an initial configuration to a goal one, being the interaction with some objects possible (and hence permitting the purposeful manipulation of objects).

The combination of the Fast Forward task planner and a physics-based motion planner is therefore appealing, and is proposed here. Also, the use of knowledge-based techniques may enhance both planning levels, e.g., by handing over sufficient information to the robots regarding the way to interact with the obstacles. The coding of knowledge by means of ontologies make it more flexible the use and the reasoning over the knowledge, providing fruitful information for the selection and execution of actions.

When the robotic system is composed of more than one robot, planning is even more challenging, since in this case the problems require the coordination of sub-tasks (a robot must be required to push away some obstacle to clear the path for another robot to travel to different regions in the workspace), or the execution of cooperative actions between several robots (the push of a heavy obstacle may require the use of more than one robot).

Contributions. This chapter presents a knowledge-oriented task and motion planning method, called κ-TMP, based on the use of physics-based motion planning and information to compute the heuristic to search a feasible plan using the Fast-Forward task planner. The proposal is designed to cope with several mobile robots sharing the tasks and collaborating with each other in order to obtain the most efficient feasible global plan. By incorporating ontological knowledge, the method offers, together with the physics-based motion planner, off-line and on-line reasoning processes on symbolic literals to determine the actions feasibility and side-effects that guide the search of the plan. As a consequence, the proposed planning approach empowers robots to be more autonomous and have the capability to accomplish goals in complex scenarios. The approach is originally presented in articles (Akbari et al., 2016a) and (Akbari et al., 2016b) involving one robot and (Akbari et al., 2018b), considering collaborative tasks for multiple mobile robots.

This proposal assumes that the information of the configuration space connectivity is available beforehand, which is feasible for a mobile robot in a 2D workspace, that all the robots are equal and that they move one at a time. The direct extension of the proposed method to multiple manipulator robotic systems with high-dimensional configuration spaces is not possible since the first assumption may not hold, although a variant of the method will be proposed in Chapter 6.

5.2 Problem Formulation and Solution Overview

5.2.1 Scope and Motivating Example

The scope of the present proposal is to deal with collaborative tasks for the *NAMO* problems in which mobile robots may interact with obstacles in the environment (by pushing or pulling them) in order to fulfill the goal of traveling to their target regions. For this purpose, robots are required to share the tasks by assisting each other for clearing the path towards the goal or by executing cooperative actions.

Many task and motion planners cope with manipulation problems involving pick and place actions, i.e., without considering push/pull actions. Alternatively, some other approaches that consider those actions are purely based on motion planning, i.e., do not include high-level reasoning. With respect to other approaches, the scope considered here pose challenging situations (even more when considering several robots) where, on the one hand, the availability of high-level reasoning is required (like when more than one object may be obstructing the path) and, on the other, the planning of robot motions with interaction with the obstacles is also necessary (push actions are required because the robot cannot pick the objects because they are too heavy or simply because no robotic arm is available).

As a motivation example, it is assumed that several robots are able to execute the following actions:

- *Transit*: To travel freely in an indoor environment.
- *Push/Pull*: To change the position of an obstacle by pushing or pulling it. Depending on the weight of the object this task must be executed simultaneously by several robots. The use of those will be constrained to objects whose removal are potentially required to solve the task (e.g. objects blocking corridors or doors).

An indoor environment cluttered with obstacles is considered. The following classification of obstacles and regions is established:

- *Fixed obstacles*: Obstacles whose location cannot be changed by the robots, either because they are attached to the environment, like walls, or because they are too heavy to be moved by them. Collisions with fixed obstacles is not allowed.
- *Manipulatable obstacles (MObs)*: Obstacles that can be pushed or pulled by the robot. Some constraints may exist regarding the directions in which they can be moved.
- *Manipulatable regions (MRgn)*: Regions next to the manipulatable obstacles from where the robot can interact with them (no collisions are allowed from elsewhere).
- *Disjointed configuration space region (C_i)*: Region of the configuration space such that a collision-free path exists between any two of its configurations.
- *Critical Objects*: Those *MObs* whose removal may connect two disjointed regions together.

Two disjointed configuration space regions C_i and C_j are said to be neighboring regions if the removal of a critical obstacle makes them connected. It will be assumed that displacing an obstacle will not end creating a new disjointed configuration space region, i.e., the effect of a push/pull action will not partition any C_i into two, nor will this happen due to the positioning of a robot.

As a motivating example, consider the problem shown in Fig. 5.1 where two robots have to transit from their initial locations towards a goal region. To solve the task, several obstacles shall be removed since no collision-free paths exist. The set of *MObs* are labeled from A to M in an increasing order with respect to their weight, and are also colored according to it (the heavier, the darker). It is assumed that obstacle M is beyond the capacity of the robots and that obstacle L can be manipulated only if both robots do it simultaneously, while the other obstacles can be

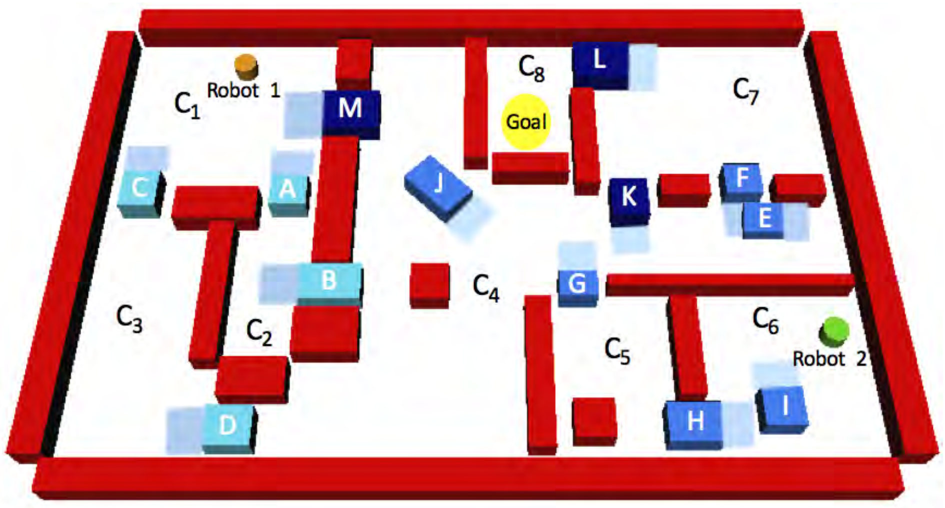


Figure 5.1: A *NAMO* manipulation example: two robots (shown as small cylinders) are required to reach the goal region by pushing or pulling some obstacles in the way (manipulatable obstacles are shown as boxes and are labeled in an increasing order with respect to their weight). Workspace regions are labelled with the name of the corresponding disjointed configuration space regions.

manipulated by a single robot. Besides, *MObs* must be manipulated through the manipulation regions (highlighted in light blue) where the robot must be located in order to pull or push them. It must be noted that, on the one hand, the execution of cooperative actions is required (pushing object L) and, on the other, the coordination of sub-tasks is also a need (obstacle G must be removed by robot 1 in order to allow robot 2 to move towards the goal).

In the planning phase, a great number of potential actions have to be considered, being their actual applicability and feasibility under appraisal. It is worth noting that some manipulation actions do not provide fruitful effects to solve the problem (e.g. there is not enough room to either push or pull object B in order to connect regions C_2 and C_4), and that such type of actions must be detected in advance in order to avoid any dead-end plan. Furthermore, among the set of feasible solution plans, the least-cost one is the one sought. These aforementioned issues pose substantial challenges that can be properly handled by considering an efficient combination of task and physics-based motion planning.

5.2.2 Problem Formulation

A task planning domain $\mathcal{D}$ is formalized as $\langle \mathcal{R}, S, \mathcal{K}_w, \mathcal{K}_p \rangle$, where:

- $\mathcal{R}$ is a graph describing the connectivity of the configuration space of any robot (all are considered equal and moving one at a time so the graph is unique irrespective of the robot used to compute it). The nodes of the graph describe the disjointed regions and the edges represent the potential connectivity between regions.

- S is a set of states containing both literals and geometric information.
- $\mathcal{K}_w$ represents, as an ontology, the semantic knowledge about the robots and the environment.
- $\mathcal{K}_p$ represents, as an ontology, the knowledge about the planning components.

Then, a task planning problem, $\mathcal{T}$, is formally defined by the tuple $\langle \mathcal{D}, \mathcal{I}, \mathcal{G} \rangle$ where $\mathcal{D}$ is the task planning domain explained above, $\mathcal{I}$ describes the initial state of the manipulation problem and $\mathcal{G}$ the set of goal conditions that have to be satisfied in the final state.

Configuration space information

The connectivity of the configuration space of a mobile robot in a 2D workspace, which can be obtained using the approach in (Stilman and Kuffner, 2005) for navigation problems, is represented by a graph $\mathcal{R}$. The nodes of $\mathcal{R}$ are the disjointed configuration space regions, and the edges connecting them are labeled with the name of the critical object(s) whose removal connects two disjointed regions into a single one. A small circle at the end of an edge illustrates that the corresponding critical obstacle can be manipulated from there. Fig. 5.2 represents the graph associated to the example of Fig. 5.1, where the first and second robots are initially located at C_1 and C_6, respectively, and their goal region is placed in C_8 (workspace regions are named as the corresponding disjointed configuration space regions). The manipulation of obstacles may change the connectivity of the configuration space. Nevertheless, the topology of the graph will remain fixed and this information will be introduced by removing the label on the corresponding edges. Also, the following functions will be used to retrieve information from $\mathcal{R}$:

- *Map($MRgn$)*: Returns the disjointed configuration space region C_i where the configuration of the robot lies when the robot is placed in the manipulatable region $MRgn$.
- *Path(C_i, C_j)*: Returns true if either $C_i = C_j$ or a path in $\mathcal{R}$ exists between nodes C_i and C_j and the edges in the path have no labels (i.e. the critical objects associated to the edges in this path have been removed). Otherwise it returns false.
- *Neigh(MObs,i)* with $i = 1, 2$: Returns one of the two disjointed neighboring regions separated by the critical object *MObs*.

States

A state S is represented by the tuple $S = \langle L, \mathcal{W} \rangle$ containing a conjunction of literals L formed based on predicates applied to arguments and that are true in the state, and the geometric

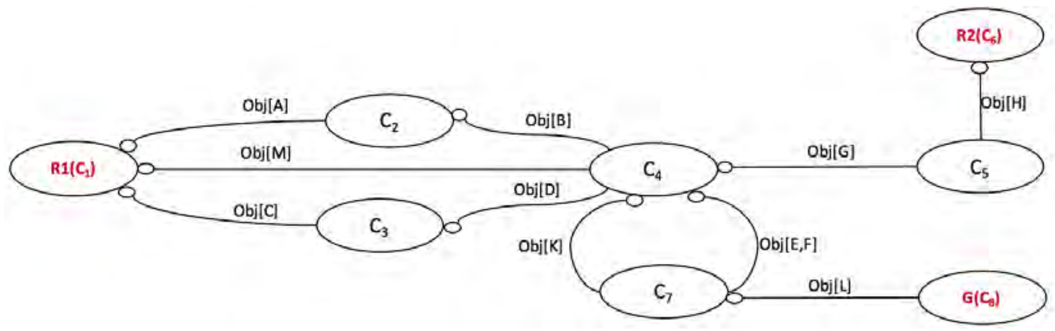


Figure 5.2: Graph $\mathcal{R}$ representing the connectivity of the configuration space for the example in Fig. 5.1. The nodes are labeled with the disjointed configuration space regions; the edges with the critical obstacles.

information of the workspace $\mathcal{W}$ representing the obstacles location and the configuration of the robots. The following set of literals are considered:

- *At(Robot, Region)*: Holds true if the *Robot* is in *Region*.
- *IsCrit(MObs)*: Holds true if *MObs* is a critical manipulatable obstacle.
- *HasAcc($MRgn_i$, $MRgn_j$)*: Holds true if the function *Path($Map(MRgn_i)$, $Map(MRgn_j)$)* returns true, informing that a path exists to move the robot from $MRgn_i$ to $MRgn_j$.

A finite set of states is considered, i.e., each geometric information corresponds with one literal. For example, if there is is a literal *At(Rob*$_1$*, R*$_{init1}$*)*, a unique transformation representing the robot position is assigned to that literal in the corresponding state.

Knowledge about the world

$\mathcal{K}_w$ encodes knowledge referring to obstacles (their geometry, pose and physical properties) and robots (the capability and the constraints). Two literals are defined whose truth values are evaluated using a lightweight reasoning process over the knowledge:

- *IsManp(MObs, Robot)*: Holds true if *MObs* can be manipulated by *Robot* according to its capability.
- *IsManpMRob(MObs, RobotA, RobotB)*: Holds true if *MObs* can be manipulated simultaneously by *RobotA* and *RobotB* according to their capability, but cannot be displaced by any of them separately.

The information in $\mathcal{K}_w$ regarding the poses of the obstacles, manipulatable regions, and their geometry is used to build the connectivity graph $\mathcal{R}$.

Knowledge about the planning components

$\mathcal{K}_p$ encodes the information regarding the initial state and the goal conditions to be met, as well as the action space $\mathcal{A}$ describing the actions. Namely, an action a is defined by a tuple ⟨*name(a), pre(a), effect*$^+$*(a), effect*$^-$*(a)*, $\mathcal{Q}$⟩ where:

- *name(a)* is a symbolic name for the action.
- *pre(a)* is the set of preconditions of the action defined as a conjunction of literals which must hold for the action to be performed.
- *effect*$^+$*(a)* is the set of positive effects defined as a conjunction of literals to be added as a result of applying an action.
- *effect*$^-$*(a)* is the set of negative effects defined as a conjunction of literals to be deleted after the action is performed.
- $\mathcal{Q}$ is a query to a physics-based motion planner acting on $\mathcal{W}$, that computes a path and its actual cost, and returns the new state of the workspace.

To map the current state s_c to the new one s_n using a given action a, the successor literals are:

$$s_n.L = \{(s_c.L \cup \mathit{effect}^+(a)) \backslash \mathit{effect}^-(a)\},$$

and, when required, the geometric information of the workspace ($s_n.\mathcal{W}$) is updated by $\mathcal{Q}$, i.e., $s_n.\mathcal{W} = \mathcal{Q}(s_c.\mathcal{W})$.

Five actions are defined: *Transit, Push, Pull, PushM*, and *PullM*, the last two introduced for the manipulation actions with multiple robots (to be applied when a single robot is not capable enough to displace the obstacle).

Since the manipulation world is represented using an ontology, it is convenient to represent the manipulation domain in the same way following the manner how actions and their conditions are defined in *PDDL*. The use of ontologies provides a well structured way of representing explicit formal specifications and gives more flexibility to describe sets of rules for different classes of objects. The actions along their preconditions, and positive and negative effects are:

Transit*(Rob, FRgn, TRgn):*

Pre: *At(Rob, FRgn), HasAcc(FRgn, TRgn)*
Add: *At(Rob, TRgn)*
Delete: *At(Rob, FRgn)*

Push/Pull*(Rob, MObs, MRgn):*

Pre: *At(Rob, MRgn), IsManp(MObs, Rob), IsCrit(MObs)*

Add: *HasAcc*(R_a,R_b) $\forall R_a, R_b | Map(R_a){=}Neigh(MObs, 1)$ and $Map(R_b){=}Neigh(MObs, 2)$
Delete: *IsCrit(MObs)*

PushM/PullM*(Rob1, Rob2, MObs, MRgn):*

Pre: *At(Rob1, MRgn), At(Rob2, MRgn), IsCrit(MObs), IsManpMRob(MObs, Rob1, Rob2)*
Add: *HasAcc*(R_a,R_b) $\forall R_a, R_b | Map(R_a){=}Neigh(MObs, 1)$ and $Map(R_b){=}Neigh(MObs, 2)$
Delete: *IsCrit(MObs)*

Note that push/pull actions are constrained to critical obstacles only, because it is their displacement that may change the connectivity of the configuration space. Note also that transit actions are always feasible provided that the preconditions hold, whereas the feasibility of push/pull actions will need to be checked.

5.2.3 Solution Overview

A knowledge-oriented task and motion planning method, called κ-TMP, for solving collaborative manipulation tasks is proposed. It is an enhanced version of the original Fast-Forward task planner as it has been explained in Chapter 2. The new version (sketched in Fig. 5.3) uses the *OWL* knowledge, reasoning processes and a physics-based motion planner to determine the actions feasibility and applicability during the computation of the heuristic that guides the search.

Manipulation Knowledge

The manipulation knowledge, comprising $\mathcal{K}_w$ and $\mathcal{K}_p$, will be coded as an ontology using the *Ontology Web Language (OWL)* and will be accessed during the planning phase as shown in Chapter 3.

Reasoning Process on Symbolic Literals

A reasoning process on symbolic literals is proposed to allow the robot reasoning upon actions. This reasoning process, detailed in Section 5.3, will allow pruning unnecessary or unfeasible actions and making better decisions during the planning phase. Offline and online reasoning processes are proposed, which are executed before and during planning, respectively:

- The *offline reasoning process* is responsible of using the manipulation knowledge to build the graph $\mathcal{R}$ and to set the planning problem $\mathcal{T}$.

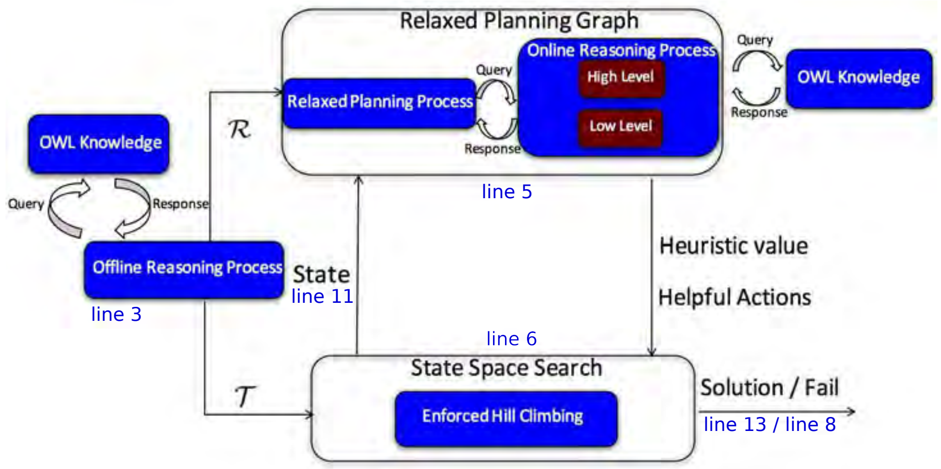


Figure 5.3: The κ-TMP planner architecture (indicated lines correspond to Algorithm 2).

- The *online reasoning process* consists of high-level and low-level modules: The high-level module determines the side effects of actions by taking into account $\mathcal{R}$; the low-level module evaluates the feasibility of actions using a physics-based motion planner.

Simultaneous Task and Motion Planning

The proposed task and motion planner is composed of the two main components of the Fast Forward method: the *Relaxed Planning Graph (RPG)* and the *State Space Search* with the following features.

- The *Relaxed Planning Graph* is the responsible of computing the heuristic value and the helpful actions. We propose a technique to compute the heuristic value regarding the physics-based information of the actions, in contrast to the original *FF* that computes the heuristic value based on the number of symbolic actions in the relaxed plan. The module constructs the *RPG* considering a cost for the push/pull actions determined by the physical properties of the obstacles, and extracts the *RPG* plan. Afterwards, the physics-based motion planner is called for the push/pull actions of the plan, in order to evaluate their feasibility as well as their actual cost, that is used to determine the heuristic value of the cost to reach the goal. Upon failure, the cause is fed back to the current state (if a new manipulatable obstacle is occluding the connectivity) or to the relaxation planning process (if there is no enough room for the push/pull action to remove the obstacle), and then the planning process resumes. Finally, the helpful actions, which are the actions that appear in the first-level of the RPG plan, are extracted.
- The *State Space Search* uses the heuristic values to select the next state, in the same way as the original *FF* procedure does.

5.3 Reasoning Process on Symbolic Literals

The reasoning process enables the robotic system to efficiently carry out the evaluation of symbolic literals required in high-level planning. The reasoning is done over the OWL knowledge, as well as over the graph $\mathcal{R}$ representing the connectivity of the configuration space and the queries answered by the motion planner. The reasoning process comprises an offline step performed before planning, and an online step performed while planning.

5.3.1 Offline Reasoning Process

The offline reasoning process primarily constructs the graph $\mathcal{R}$, and requests the query conditions of the problem from $\mathcal{K}_p$, in order to acquire $\mathcal{I}$ and $\mathcal{G}$. Furthermore, the process is responsible to reason on action preconditions according to $\mathcal{R}$ and $\mathcal{K}_w$, leading to the assertion of predicates that hold at the initial state:

- *IsCrit* is asserted for the obstacles that label the edges of $\mathcal{R}$.
- *IsManp* is asserted for the critical obstacles according to the robots capabilities and the features of the obstacles. This process is done through the knowledge-based reasoning process. The process computes the force which the robot requires to assert using physical properties of an obstacle in order to displace an obstacle. It also knows the maximum force that each robot can apply in the environment. If the computed force is under the maximum robot force, *IsManp* is asserted for the requested obstacle.
- *IsManpMRob* is asserted for the critical obstacles according to the capability of the robots and the physical properties of the obstacles. It is also computed using the knowledge-based reasoning process.
- *At* is asserted for each robot regarding the initial regions where they are located.
- *HasAcc* is asserted for any two regions of the same C_i where the robots are initially located, like *HasAcc(init1, MRgnC)* or *HasAcc(init2, MRgnC)*.

5.3.2 Online Reasoning Process

The online reasoning process contains a high-level reasoning module and a low-level reasoning module, as shown in Fig. 5.4, to reason on the effects of actions during the computation of the heuristics. The modules are responsible of the following. On the one hand, the high-level reasoning module uses the information in $\mathcal{R}$ to assert the effects of push/pull actions while constructing the *RPG*. On the other, the low-level reasoning module uses the motion planner to evaluate the feasibility of push/pull actions selected for the plan extracted form the *RPG*. In case a push/pull action is infeasible due to a collision with a fixed obstacle (condition 1 in Fig. 5.4), then the cost of this action is increased and the search of an alternative relaxation plan is launched. If otherwise the infeasibility is due to a collision with a manipulatable obstacle

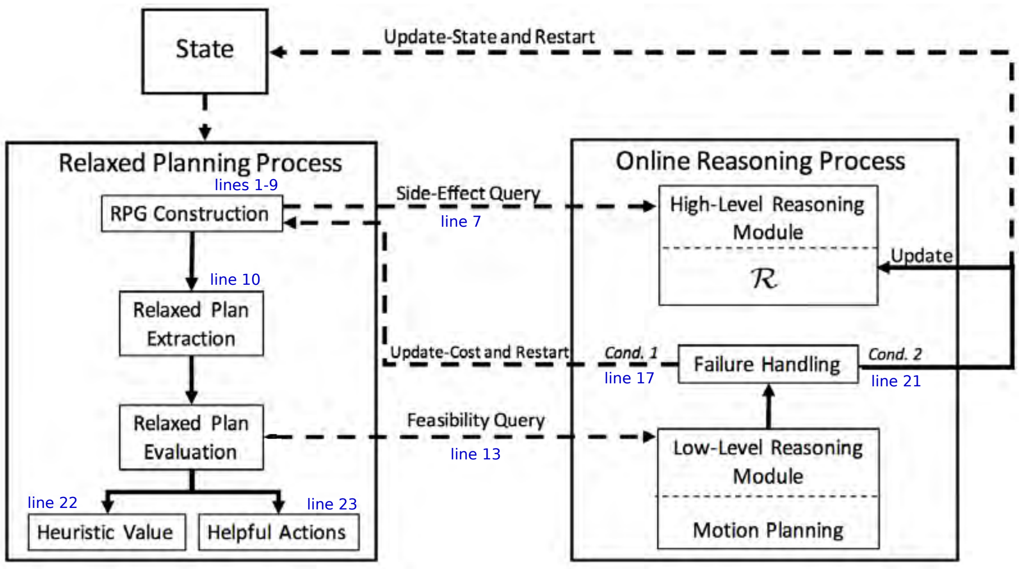


Figure 5.4: Information flow between the relaxed planning process and the online reasoning process, corresponding to the module *'Relaxed Planning Graph'* in Fig. 5.3. Indicated lines correspond to Algorithm 3.

(condition 2 in Fig. 5.4), then the state must be updated by adding this obstacle as a critical obstacle, the graph $\mathcal{R}$ must be updated accordingly, and the *RPG* construction must be restarted. The modules are detailed next.

The *high-level module* computes the positive effect of the push/pull actions, that consists in asserting the predicates *HasAcc*(R_a,R_b) between any two regions such that they belong to the neighboring disjointed configuration space regions that become connected by the removal of the critical object that makes them disconnected, i.e.: *HasAcc*(R_a,R_b) $\forall R_a, R_b | Map(R_a) = Neigh(MObs, 1)$ and $Map(R_b) = Neigh(MObs, 2)$. For instance, consider two disjointed configuration space regions, C_i and C_j blocked by an obstacle A, and two robots initially located at $R_{init1} \in Map(C_i)$ and $R_{init2} \in Map(C_i)$, respectively, and willing to travel to $R_{goal1} \in Map(C_j)$ and $R_{goal2} \in Map(C_j)$. Upon removal of A, the high-level reasoning process asserts *HasAcc*(R_{init1},R_{goal1}), *HasAcc*(R_{init1},R_{goal2}), *HasAcc*(R_{init2},R_{goal1}), and *HasAcc*(R_{init2},R_{goal2}) showing that both robots have access to the goal regions. The same process can be applied to any number of robots.

The *low-level module* evaluates the push/pull actions that appear in the relaxed plan, by calling the motion planner. Let two disjointed configuration space regions, C_i and C_j, be disjointed due to the presence of a critical obstacle A, i.e., the graph $\mathcal{R}$ has an edge labelled with A between nodes C_i and C_j. Then, the evaluation of the feasibility of the push/pull action applied over A is done in two interleaved queries. The first, called $\mathcal{Q}_d$, displaces A a given predefined small distance (with one or two robots depending on the type of push/pull action being analyzed); the second, called $\mathcal{Q}_{tr}$, appraises whether there is a path for the robot between any two arbitrary regions R_a and R_b such that $R_a \in Map(C_i)$ and $R_b \in Map(C_j)$. The two steps are repeated until a path for the $\mathcal{Q}_{tr}$ query is found or a collision occurs between A and another obstacle in the environment, or between the robot and another obstacle in the environment. Then:

- If the collided obstacle is a fixed obstacle, it means that the action is not feasible. Therefore, the *RPG* construction must be restarted and an alternative plan has to be searched by setting the cost of the action at a high value in the *RPG* in order not to select it.
- If the collided obstacle is a manipulatable obstacle, e.g., B, it means that it should be removed before trying to move A. Therefore:
    1. The state that was being explored must be updated by asserting B as a critical obstacle.
    2. The graph $\mathcal{R}$ must be updated by modifying the edge labelled with A with a label including both A and B, which expresses the fact that in order to make C_i and C_j connected, it is necessary to remove first B and then A.
    3. The *RPG* construction must be restarted.

For the feasible actions, the motion solutions returned by $\mathcal{Q}_d$ as well as their associated cost, computed as follows, are stored to be retrieved if these actions appear in the final plan. The cost for any transit or any push/pull action solved by the motion planner is computed according to the power consumed. The motion planner returns a trajectory as a sequence of controls, where each control is a force to be applied during a time interval. Then, the power consumed for a robot is:

$$\mathcal{C} = \sum_{j}^{n} \frac{\mathbf{f}_j \cdot \mathbf{d}_j}{\Delta t_j}, \tag{5.1}$$

where n is the number of controls of the trajectory, $\mathbf{f}_j$ and Δt_j the force and time interval corresponding to the j-th control, and $\mathbf{d}_j$ the resultant displacement. If the action involves more than one robot, the power consumed is the sum of the individual values for each one.

5.4 Knowledge-based Task and Motion Planning

The κ-TMP approach integrates the components of reasoning process and the ontological knowledge, altogether, with the modified version of *FF*. This section details the algorithms for the state space search module and of the relaxed planning graph module, as graphically introduced in Fig. 5.3.

5.4.1 State Space Search

Algorithm 2 outlines the procedure to obtain the final manipulation plan. The first step is the offline reasoning process, performed in function offlineProcess [line 3], that provides the initial state of the manipulation world S_0, the goal condition $\mathcal{G}$, and the connectivity graph $\mathcal{R}$. Then, the standard *Enhanced Hill Climbing (EHC)* technique proposed in *FF* is used to find the successor state in the plan [lines 4-11]: At each iteration, the RPG function is first called to obtain the helpful actions $H(S_i)$ and the heuristic value $h(S_i)$ that predicts the cost distance from the state S_i to the goal state [line 5] (this function is detailed in the next subsection). Then, using the helpful actions, the selectAction function [line 6] performs a search based on *EHC* for a successor state S_j such that $h(S_j) < h(S_i)$ (successor states are also evaluated using the RPG function). If such helpful action is not found [line 7], the algorithm fails and the search stops [line 8]. Otherwise, the helpful action leading to S_j is added to the plan π [line 10] and S_j is added as the next state in the plan, S_{i+1} [line 11]. The loop continues until $\mathcal{G}$ is satisfied and the plan π is returned, which is a sequence of push/pull and transit actions. If the *EHC* search fails, the *Best-First Search (BFS)* is considered as the original *FF* planner does. The motions corresponding to the push/pull actions are already known because the motion planner has been called to evaluate their feasibility. Nevertheless, the motions corresponding to the transit actions have to be found by calling now the motion planner [line 12] (no feasibility problems may arise since for these actions the satisfaction of the preconditions assures it).

```
Algorithm 2: $\kappa$-TMP
   inputs : $\mathcal{K}_p, \mathcal{K}_w$
   output: The feasible plan $\pi$
1  $i \leftarrow 0$
2  $\pi \leftarrow \emptyset$
3  $\{S_0, \mathcal{G}, \mathcal{R}\} \leftarrow$ offlineProcess$(\mathcal{K}_p, \mathcal{K}_w)$
4  while $\mathcal{G} \not\subseteq S_i$ do
5      $\{h(S_i), H(S_i\}) \leftarrow$ RPG$(S_i, \mathcal{G}, \mathcal{K}_p, \mathcal{K}_w, \mathcal{R})$
6      $\{H'(S_i))\} \leftarrow$ selectAction$(H(S_i), h(S_i))$
7      if $H'(S_i).empty()$ then
8          return *fail*
9      else
10         $\pi$.append$(H'(S_i))$
11         $S_{i+1} \leftarrow$ Succ$(S_i, H'(S_i))$
12 motionPlanning$(\pi)$
13 return $\pi$
```

5.4.2 The Relaxed Geometric Reasoning Process

The *RPG* is used to compute the heuristic value that estimates the cost to reach the goal state from the state being explored. The standard *FF* procedure does it in terms of number of actions. The κ-TMP, however, proposes to take into account the different costs of actions:

- Regarding the *RPG* construction and the extraction of the relaxed plan, κ-TMP proposes to consider, for the push/pull actions, an approximate cost according to the physical properties of the obstacles, and the unitary cost for the transit actions (the cost of maintenance actions that keep literals unchanged in the next state-level is set to zero). In this way, the *RPG* takes into account that the costs of actions may be different, but without the need of calling the motion planner to compute their actual cost. In particular, any push and pull action cost is set greater that the transit actions cost, and with a value that depends on the object mass:

$$cost(transit) = 1 \tag{5.2}$$
$$cost(push/pull) = m_i/m_j \tag{5.3}$$

  where m_i is the mass of the critical object being pushed/pulled and m_j is the mass of the lightest one.
  The cost of each literal l in a state-level is set from the cost of the action a_i that generates it plus the cost of the action preconditions $pre(a_i)$. Since several actions can generate the same literal, the minimum cost is selected:

$$cost(l) = min_{\forall a_i \mid l \in effect^+(a_i)}\{cost(a_i) + \sum cost(pre(a_i))\} \tag{5.4}$$

  Note that a_i is any of the actions introduced in Sec.5.2.2 and therefore may refer to either one or two robots.
- Regarding the computation of the heuristic value, a more exact cost of actions is used by applying Eq. (5.1): The cost of the push/pull actions is already known because, as detailed in the previous section, the motion planner has been called to evaluate the feasibility of the push/pull actions appearing in the relaxed plan. This is not the case for the transit actions, and the proposal is to estimate it by approximating the travelled distance by the Euclidean distance and assuming the use of the minimum force all along the path. The heuristic value of a *RPG* plan with n actions will be computed as:

$$h = \sum_{i}^{n} \mathcal{C}_i \tag{5.5}$$

The *RPG* construction procedure of the standard *FF* is terminated at the first state-level where $\mathcal{G}$ is satisfied. In the current proposal, however, the procedure keeps growing the relaxed planning graph some levels further (within a predefined maximum number of levels) until the cost of goal literals remains stable. By not stopping the growing procedure at the first state-level where the goal conditions are met, a lower cost value of goal literals can be possibly obtained. Then, the backward search for the relaxed plan is eventually performed from these conditions, yielding to the cheapest actions sequence.

Algorithm 3 sketches the computation of the modified *RPG* integrated with the online reasoning process, used to obtain the physics-based heuristic value and the feasible helpful actions. Its graphical flow was shown in Fig. 5.4. Its key points are as follows:

Algorithm 3: RPG($S, \mathcal{G}, \mathcal{K}_p, \mathcal{K}_w, \mathcal{R}$)

```
1  $\mathbb{S}_0 \leftarrow S$
2  $i \leftarrow 0$
3  while $i$ <MaxLevels do
4      $i \leftarrow i + 1$
5      $\mathbb{A}_i \leftarrow \{a \in \mathcal{K}_p.\mathcal{A} \mid pre(a) \subseteq \mathbb{S}_{i-1}\}$
6      $\mathbb{A}_i \leftarrow$ append(maintActions($\mathbb{S}_{i-1}$))
7      $\mathbb{S}_i \leftarrow \{l \mid l \in \text{effect}^+(\mathbb{A}_i)\}$
8      compCost($\mathbb{S}_i$)
9      if $\mathcal{G} \subseteq \mathbb{S}_i$ AND checkCost() then
10         $\pi' \leftarrow$RPGPlan()
11         $h \leftarrow 0$
12         for each $\{a \in \pi'\}$ $\{a \in \pi' \mid a.name \neq Transit\}$ do
13             $\{f.response, ColObj\} \leftarrow$ feasibilityChecker($a, \mathcal{K}_w$)
14             if $f.response = infeasible$ then
15                 if $ColObj \neq MObs$ then
16                     setHighCost($a$)
17                     GOTO 4      //Cond. 1 in Figure 5.4
18                 else
19                     update$\mathcal{R}$()
20                     updateState()
21                     GOTO 1      //Cond. 2 in Figure 5.4
22         $h \leftarrow$ heuristicValue()
23         $H(S) \leftarrow$ helpfulActions()
24         return $\{h, H(S)\}$
25 return $\{\infty, \emptyset\}$
```

- *Computing action-levels and state-levels* [lines 1-8]: The initial state-level $\mathbb{S}_0$ is created with respect to the information of the state S from where the *RPG* is to be computed [line 1]. Afterwards, action-levels $\mathbb{A}_i$ and state-level $\mathbb{S}_i$ are iteratively computed [lines 4-7]. At each level i, for the construction of $\mathbb{A}_i$, a is added if the preconditions appear in the level $i-1$ [line 5]; and any maintenance action is added too [line 6]. Then, the state-level $\mathbb{S}_i$ is constructed according to the effect$^+$ added by any action appearing in $\mathbb{A}_i$ [line 7] (negative effects are neglected since the relaxed version of the planning graph is being computed). Finally, the cost of literals is computed in compCost function based on Eq. (5.4) [line 8].
- *Terminating condition* [lines 3, 9]: The *RPG* construction ends and the plan extraction starts when both a state-level satisfies $\mathcal{G}$ and the function checkCost returns true (this latter condition happens when at the previous state-level the goal conditions were also satisfied and the cost value of goal literals has not changed up to some more levels). The extracted plan is used to compute the heuristic value and the helpful actions that are returned by the algorithm, if the *RPG* reaches a given maximum number of levels (MaxLevels) without satisfying $\mathcal{G}$, the algorithm terminates and returns the infinity heuristic value and the empty set of helpful action.
- *Relaxed plan computation and feasibility evaluation* [lines 10-23]: The RPGPlan function extracts the potential relaxed plan π' [line 10], and the included push/pull actions are forwarded to feasibilityChecker for the feasibility check [lines 12-13] (upon equal costs, pull actions are selected during the extraction of π'). This function calls the motion planner and returns whether the action is feasible or not, and in this latter case the obstacle that precluded it to be feasible. Infeasibility may be due to a collision with a fixed obstacle [lines 15-17] or with a *MObs* [lines 18-21]. In the first case, the cost of the action is set to high by setHighCost and the *RPG* construction is restarted in order to extract an alternative relaxed plan. In the second case, $\mathcal{R}$ and the current state are updated by the update$\mathcal{R}$ and updateState functions, respectively, and the *RPG* construction is restarted. In this way dead-end plans are avoided.
- *The computation of heuristic value and helpful actions* [lines 25-26]: When all evaluated actions are feasible, the heuristic value h is extracted by the heuristicValue function that computes the costs of the actions appearing in the relaxed plan as detailed above using Eq. (5.5). The helpful actions are extracted by helpfulActions.

5.5 Implementation and Results

5.5.1 Implementation

The proposed framework implementation consists of three major layers as depicted in Fig. 5.5: the ontological knowledge, task-level, and motion-level layers. The knowledge layer is coded in the form of an *OWL* ontology as detailed in Chapter 3. The task-level layer embraces the *Heuristic task planner* which is a modified version of the *FF* planner implemented using the *Prolog* language, and the *Action reasoning process* whose purpose is to determine actions conditions by calling online along offline reasoning processes. The motion-level layer comprises *The Kautham Project* (Rosell et al., 2014) (`https://sir.upc.edu/projects/kautham/`) that enables to plan under kinodynamic and physics-based constraints. It uses the *Open Motion Planning Library (OMPL)* (Sucan et al., 2012) as its core of planning algorithms, and is integrated with the *Open Dynamic Engine (ODE)* for the dynamic simulations. Although any kinodynamic motion planner can be selected, *KPIECE* (Şucan and Kavraki, 2010) has been used in the experiments because in a comparative study (Muhayyuddin et al., 2016) it showed the highest success rate and the best time-optimal solution as compared to other state-of-the-art kinodynamic planners. This planner does not minimize the distance and therefore the paths found will not be the shortest ones.

The task-level layer accesses the *OWL* knowledge using the *KnowRob* software (Tenorth and Beetz, 2009), a potent knowledge processing tool that enables a flexible access to the *OWL* knowledge. It is mainly developed in the *Prolog* language and provides fundamental predicates to fetch knowledge, e.g., the query *owl_subclass_of(?SubClass, ?Class)* explores all available subclasses of a class, *owl_individual_of(?Indv, ?Class)* seeks to list all individuals of a class, and *class_properties(?Class, ?Properties, ?Value)* determines the value of a class under particular properties. The communication between the task and motion layers is done using Robot Operation System (ROS, http://www.ros.org/, (Quigley et al., 2009)). The motion planner is encapsulated as a *ROS* service and the task planner as a *ROS* client (using the *SWI-Prolog* library (Wielemaker et al., 2012)).

5.5.2 Results and Discussion

The proposal has been first tested for a *NAMO* manipulation problem including one robot. Consider the task shown in Fig. 5.6 in which the robot is located in the initial region and must move towards the goal one. The initial and goal configurations of the robot do not belong to the same connected region of C_{free}, and therefore some manipulatable objects, labeled from A to H, must be moved away. These objects must be manipulated through a handle that determines the motion direction, as well as the manipulation region (highlighted in dark red) where the robot must be located in order to pull or push them. Moreover, these objects have different physical features and some of them, like object A, may be beyond the robot manipulation capacity.

The simulation result for the example in Fig. 5.6 and the sequence of actions for the execution of the plan is depicted in Fig. 5.7. *KPIECE* is used as kinodynamic motion planner for the physics-based planning since it has the highest success rate and computes the time-optimal solution as compared to other state-of-the-art kinodynamic planners such as *RRT* and *EST* (Muhayyuddin et al., 2016). While manipulation planning is performed, some potential actions are found in *RPG* plans, being their feasibility under investigation by the online reasoning process. It has been observed, for instance, that when *RPG* planning process is taking place, it confronts with several potential parallel actions like *Push/PullE* and *Push/PullF*, and the one of least cost is selected, in this case actions involving object F which is lighter than object E. Online

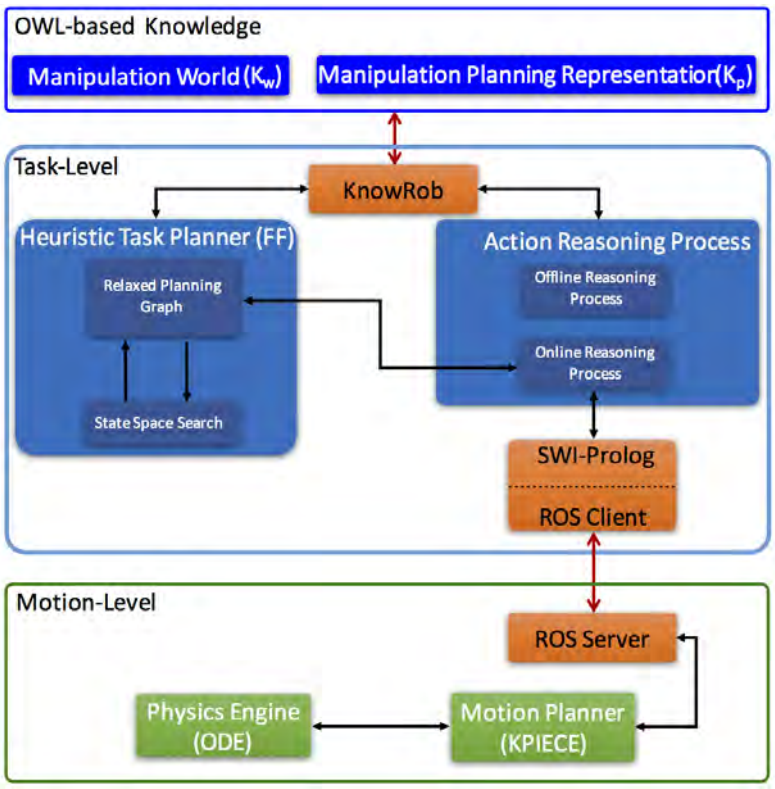


Figure 5.5: The proposed framework

reasoning process is then responsible to evaluate *RPG* plans and reject those ones containing infeasible actions (such as *PushC, PullF,* and *PushG*) because their effects cannot be satisfied. The simulation was run on an Intel Core i7-4500U 1.80GHz CPU with 16 GB memory and average planning time to compute the final manipulation plan was 89 s including several calls to the motion planner.

The problem posed in Fig. 5.1 where two robots are required to share a task and collaborate with each other for obtaining a global manipulation plan, has been solved using the κ-TMP approach. The example illustrates the challenges that appear in the navigation of robots among movable obstacles, that can be handled by the inclusion of a reasoning process (counting motion planning) within the computation of the heuristic that guides the state-space search of the *FF* task planner. This guiding allows to find a feasible sequence of actions, i.e., a sequence of actions that can be executed without colliding with the environment and taking into account the capability of the robots. Moreover, the problem posed makes it evident that the cooperation between robots is necessary.

The solution sequence of executive actions for the final plan, illustrated in Fig. 5.8, is:

a_1: *Rob1* transits from *Init* to *MRgn* of *MObs* C.

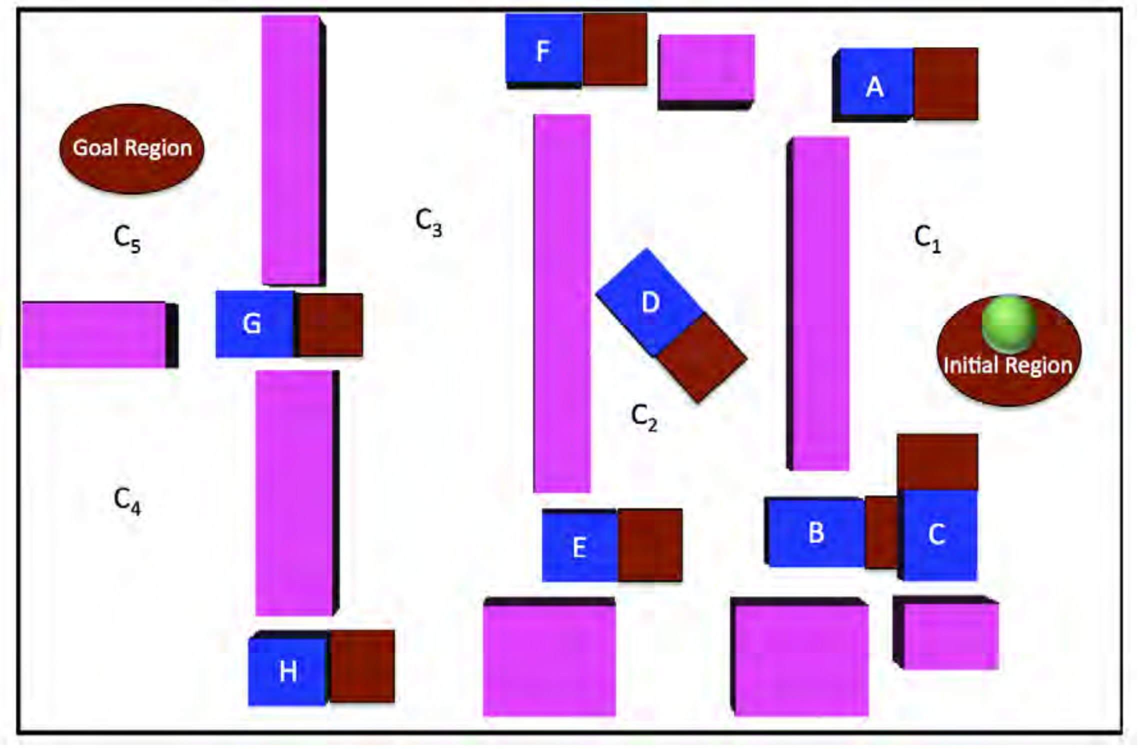


Figure 5.6: Manipulation problem: the robot (green sphere) has to move from the initial to the goal region among fixed and manipulatable obstacles (labeled according to their weight in a decreasing order). Workspace regions are labeled with the names of the associated configuration space being disconnected regions.

a_2: *Rob2* transits from *Init* to *MRgn* of *MObs* I.
a_3: *Rob2* pulls *MObs* I.
a_4: *Rob2* transits to *MRgn* of *MObs* H.
a_5: *Rob2* pulls *MObs* H.
a_6: *Rob1* pulls *MObs* C.
a_7: *Rob1* transits to *MRgn* of *MObs* D.
a_8: *Rob1* pushes *MObs* D.
a_9: *Rob1* transits to *MRgn* of *MObs* G.
a_{10}: *Rob1* pulls *MObs* G.
a_{11}: *Rob1* transits to *MRgn* of *MObs* E.
a_{12}: *Rob1* pushes *MObs* E.
a_{13}: *Rob1* transits to *MRgn* of *MObs* F.
a_{14}: *Rob1* pushes *MObs* F.
a_{15}: *Rob2* transits to *MRgn* of *MObs* L.
a_{16}: *Rob1* transits to *MRgn* of *MObs* L.
a_{17}: *Rob1* and *Rob2* pull *MObs* L.

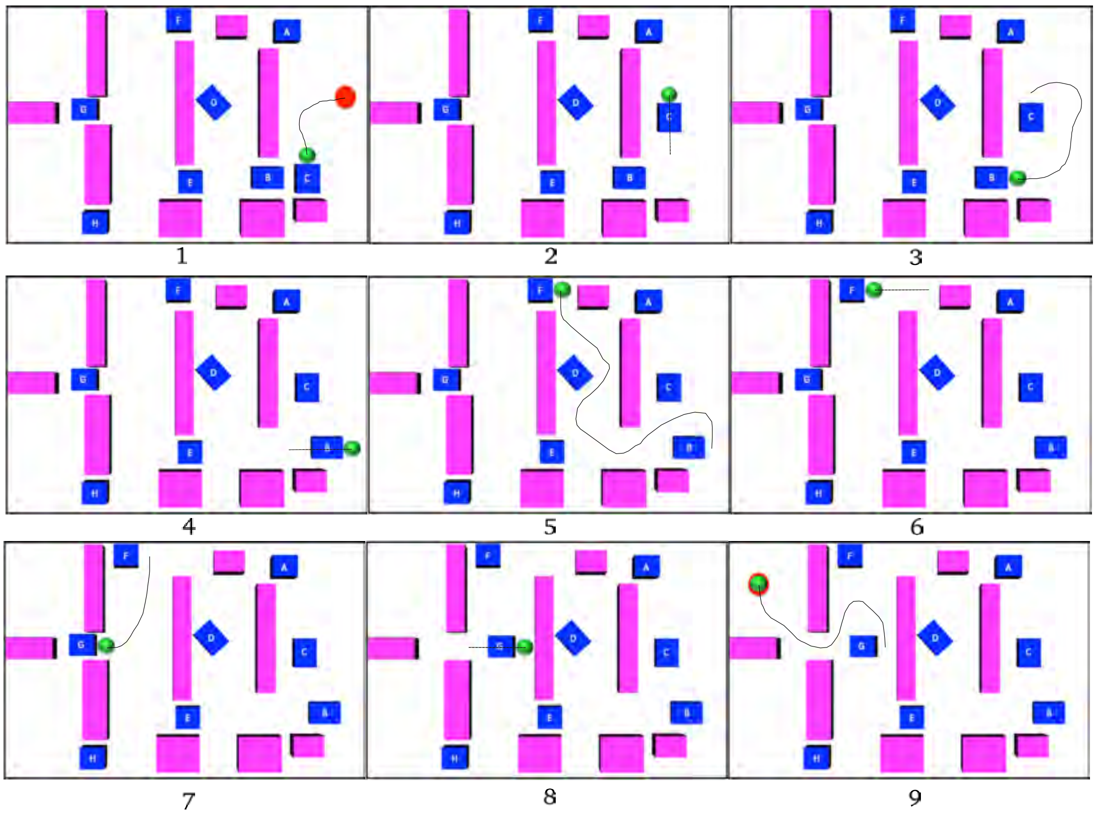


Figure 5.7: Snapshots of the task execution that involves the following actions: 1) transit to C, 2) pull C, 3) transit to B, 4) pull B, 5) transit to F, 6) push F, 7) transit to G, 8) pull G, 9) transit to Goal (The solution can be visualized in `https://sir.upc.edu/projects/kautham/videos/manip.mp4`).

a_{18}: *Rob2* transits to *Goal*.
a_{19}: *Rob1* transits to *Goal*.

Several challenges arise due to the following constraints regarding the geometry of the problem and the physical properties of the objects:

1. Robot 2 can transit towards the goal region only if robot 1 displaces the *MObs* G away.
2. Due to the presence of fixed obstacles (walls), there is not enough room to push/pull *MObs* B, or to pull *MObs* E, or to push *MObs* H and I.
3. Due to the presence of *MObs* I the *MObs* H cannot be pulled.
4. *MObs* M is too heavy to be moved, and *MObs* L can only be moved if both robots push/pull it simultaneously.

These constraints make the combination of task and motion levels a must, e.g., a plan that

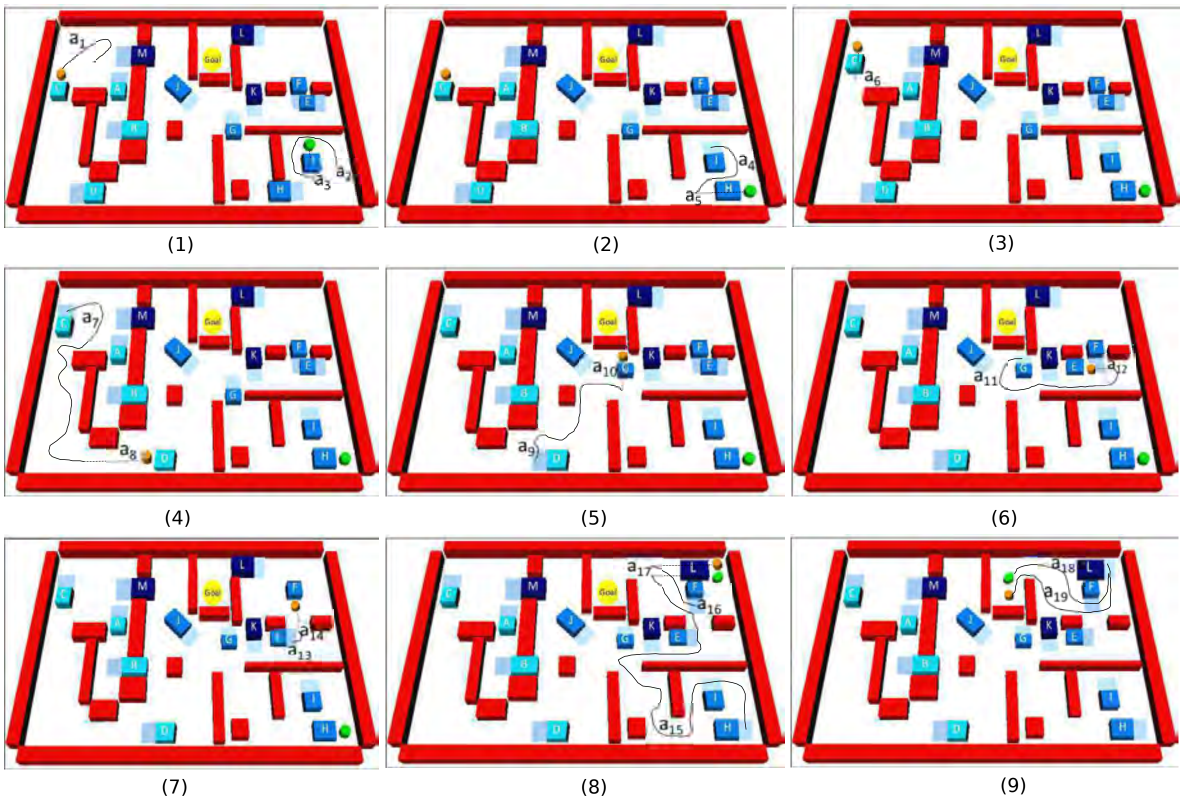


Figure 5.8: Snapshots of the task execution: 1) actions a_1, a_2, and a_3; 2) actions a_4 and a_5; 3) action a_6; 4) action a_7 and a_8; 5) actions a_9 and a_{10}; 6) actions a_{11} and a_{12}; 7) actions a_{13} and a_{14}; 8) actions a_{15}, a_{16} and a_{17}; 9) actions a_{18} and a_{19}. Video: `https://sir.upc.edu/projects/kautham/videos/k-TMP.mp4`

includes interactions with *MObs* M or B is not physically executable. The proposed κ-TMP planner is able to find a good solution in terms of power in a computationally efficient way due to the following features:

a) κ-TMP is able to identify infeasible actions like those of item 2 above by calling the motion planner during the computation of the heuristic and remove them, thus avoiding any dead-end plan.

b) κ-TMP is also able to identify infeasible actions like that of item 3 and set as critical object the one whose removal can make the action feasible.

c) During the computation of the heuristic, κ-TMP calls the motion planner only for the push/pull actions (those that may change the connectivity of the configuration space), thus avoiding unnecessary calls to the motion planner.

d) The construction of the *RPG* plan while computing the heuristic does not stop at the first level where the goal conditions appear but some levels further, thus allowing the possible extraction of plans with lower cost.

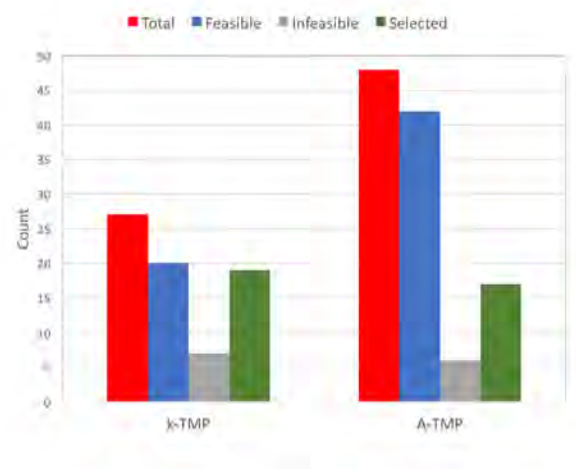


Figure 5.9: Histogram of actions evaluated by the motion planner: the total number of actions, feasible, infeasible, and selected actions.

e) κ-TMP is able to cope with multi-robot problems, since one of the effects of any push/pull action is to update the connectivity of the regions, thus potentially allowing the satisfaction of the preconditions of transit actions of other robots. Moreover, by managing the preconditions of the actions, collaborative tasks naturally arise if needed.

The performance of the present proposal is compared with a simplified version of it. Suppose that the *checkCost* function is skipped (in Algorithm 3 line[9]) and all actions are evaluated using physics-based motion planning instead of just *Transit*. Let call this modified approach as *A-TMP*. In this case, the performance will illustrate the advantages of some of the steps in the proposed *Relaxed Planning Graph* computation.

The results show that κ-TMP with respect to *A-TMP* is computationally more efficient due to feature **c** (*A-TMP* calls the motion planner for all the action in the *RPG* plan), and it is able to find a better path in terms of power consumption due to feature **d** (*A-TMP* finds a shorter plan in terms of number of actions by removing *MObs* K instead of both *MObs* E and F, but with a higher cost due to the heavy weight of *MObs* K).

The simulation was run on an Intel Core i7 2.50 GHz CPU with 16 GB memory for each planner. The performance of each manipulation planner is represented by Fig. 5.9 and Fig. 5.12. Fig. 5.9 (a) shows the number of calls to the motion planner. Accordingly, in the case of *A-TMP*, planning time has more than two fold increase as shown in Fig. 5.9 (b), but the execution time is less because it follows a shorter plan in terms of number of actions. Finally, κ-TMP finds the manipulation plan with less power consumed, approximately 36 (KJ/s), as compared with the other planner which obtains the plan with the cost approximately 42 (KJ/s).

The approach has been validated with different number of robots and obstacles, although

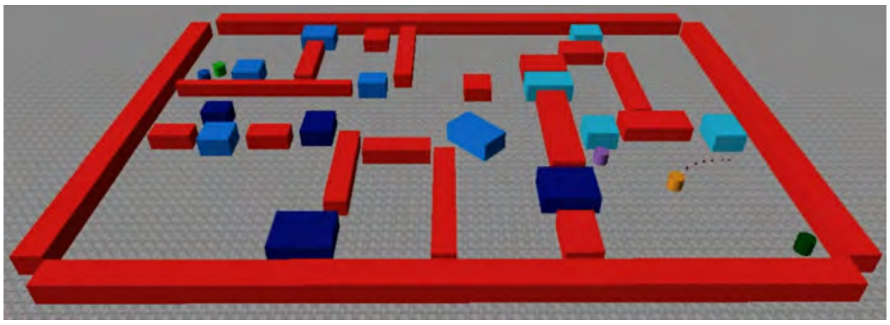


Figure 5.10: The initial state of the manipulation problem with five mobile robots.

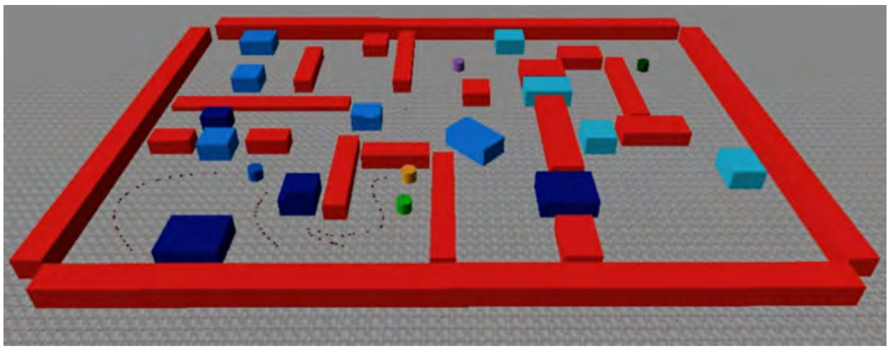


Figure 5.11: The goals state of the manipulation problem with five mobile robots. `Video: https://sir.upc.edu/projects/kautham/videos/k-TMP-5.mp4`

the action space has not been changed, i.e., actions are performed by only one or two robots (as an example, Fig. 5.10 and Fig. 5.11 show the initial and goals state of a manipulation problem with five robots respectively). Fig. 5.13 shows the results in terms of planning time, where a linear increases is detected according to both the number of robots and the number of obstacles.

It is worth noting that the approach uses a probabilistic complete motion planer (i.e. one that finds a solution if one exists provided enough time is left to the planner). Then, to evaluate the feasibility of the push/pull actions, if the motion planner times-out, the action is not pruned but its cost increased in the heuristic phase and the same action can later be tried again by providing more motion planning time. As a consequence, the proposed task and motion planner is probabilistic complete.

An interesting extension of the current work is to consider moving obstacles, like done in (Saha et al., 2014) and (da Silva et al., 2016) that solve mobile robot problems using different versions of task planners, focusing the reasoning in low-level planning problems where avoidance with moving obstacles is a critical issue. Also, we plan to consider mobile robots equipped with a manipulator to enhance the manipulation capabilities. In this line, the physics-based motion planner has already been tested for fixed robot manipulators in (Muhayyudin

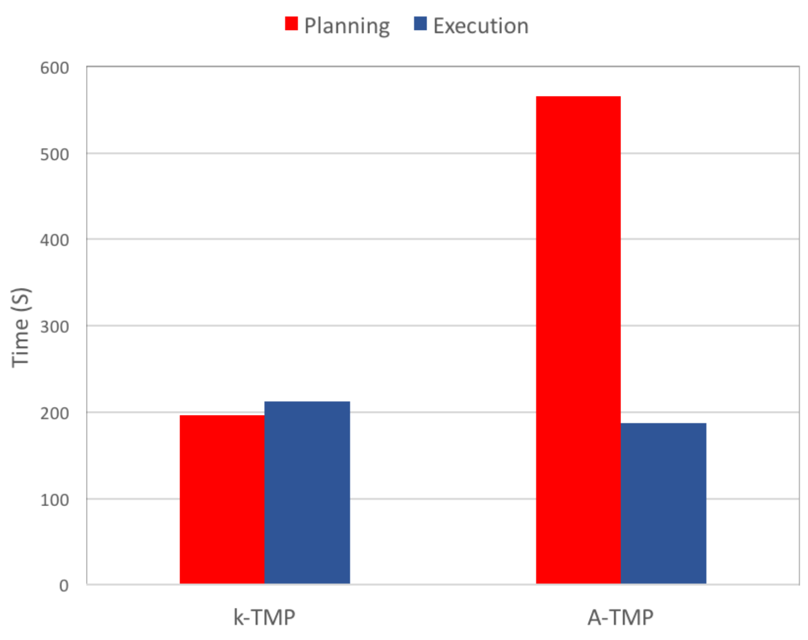


Figure 5.12: Histogram of planning and executive time.

et al., 2015; Muhayyuddin et al., 2018).

5.6 Conclusions

This chapter has presented an approach, called κ-TMP, to interweave task and motion planning for mobile manipulation problems, where multiple mobile robots are required to collaborate in order to navigate among movable obstacles. The framework is based on the *Fast Forward (FF)* task planner, on a physics-based motion planner, and on the use of knowledge represented with ontologies, which provide the capabilities required to face the challenges set at different levels.

The main challenge at task planning level is how to incorporate low-level geometric information. The proposal uses the *FF* task planner that does an heuristic search in state space, and naturally allows the inclusion of geometric reasoning in the heuristic computation. The main challenge at motion planning level is the capability to plan both collision-free motions and motions in contact with movable obstacles in order to pull or push them. In the current proposal, these capabilities are provided by a physics-based motion planner based on the *ODE* dynamic engine as state propagator and the *KPIECE* kinodynamic planner as a search algorithm. The physics-based motion planner has been mainly interleaved within the heuristic phase of the planner. Moreover, to provide the robots with the capacity to take good decisions at both task

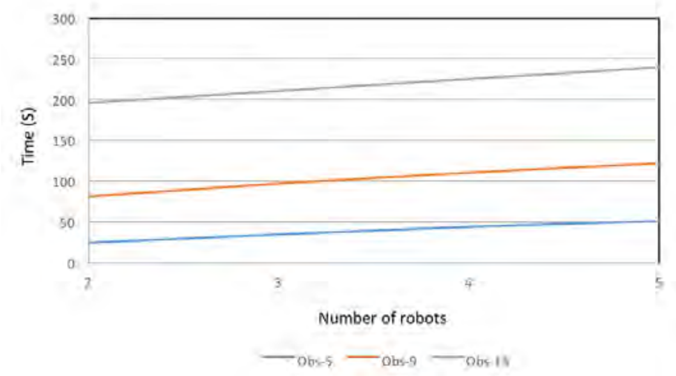


Figure 5.13: Experiments with different number of robots and obstacles including 5 obstacles (Obs-5), 9 obstacles(Obs-9), and 13 obstacles(Obs-13). Some details of problems can be found in https://sir.upc.es/projects/ontologies/.

and motion levels is also a key challenge. This can be faced using knowledge coded in terms of ontologies, that allow to well organize the knowledge and get access to it. The ontological knowledge has been integrated as a plug-in to both task and motion planning levels. Finally, it is important to highlight that the proposal is able to handle multi-robot systems in a very simple and direct way. On the one hand, solution plans may include cooperative actions since push/pull actions have been defined to be carried out simultaneously by two robots. On the other hand, the actions carried out by different robots are naturally coordinated in the plan because the heuristic guides the search towards the least cost plan and selects the actions only depending on the satisfaction of pre-conditions, irrespective of the robot that executes them.

The main contributions of the approach are: a) the integration of high- along low-level reasoning modules in the relaxed planning process of *FF* that computes the heuristic that guides the search; b) the use of a physics-based motion planner as a low level reasoning module; c) the use of physics-based information in the computation of costs, thus leading to power efficient solutions; d) the use of knowledge to set the problem and in the high-level reasoning module, that facilitates the collaboration between robots.

The approach has been implemented and validated for manipulation problems involving either one robot or several robots required to perform collaborative manipulation tasks. The results demonstrate the efficiency in the number of calls to the motion planner, the planning time, and the cost of the final plan.

The current proposal fits with the *NAMO* manipulation problems where multiple robots collaborations are required. Because the approach needs the precomputation of the configuration space connectivity, it would be computationally expensive to apply it for *Pick* and *Place* manipulation problems at which the robotic arms are involved. To overcome this limitation, a new proposal also on the *FF* task planner will be introduced in chapter 6.

Chapter 6

Efficient Heuristic-based Task and Motion planning for Bi-manual Robots

6.1 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 infeasible 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 et al., 2010; Hauser and Latombe, 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 (Cambon et al., 2009; Akbari et al., 2016a; 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 (Lagriffoul and Andres, 2016; Dantam et al., 2016; He et al., 2015; 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. These challenges are the subject of the present chapter.

Consider a manipulation problem including the bi-manual Yumi robot and a set of objects as depicted in Fig. 6.1. 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.

Contributions. The current chapter 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

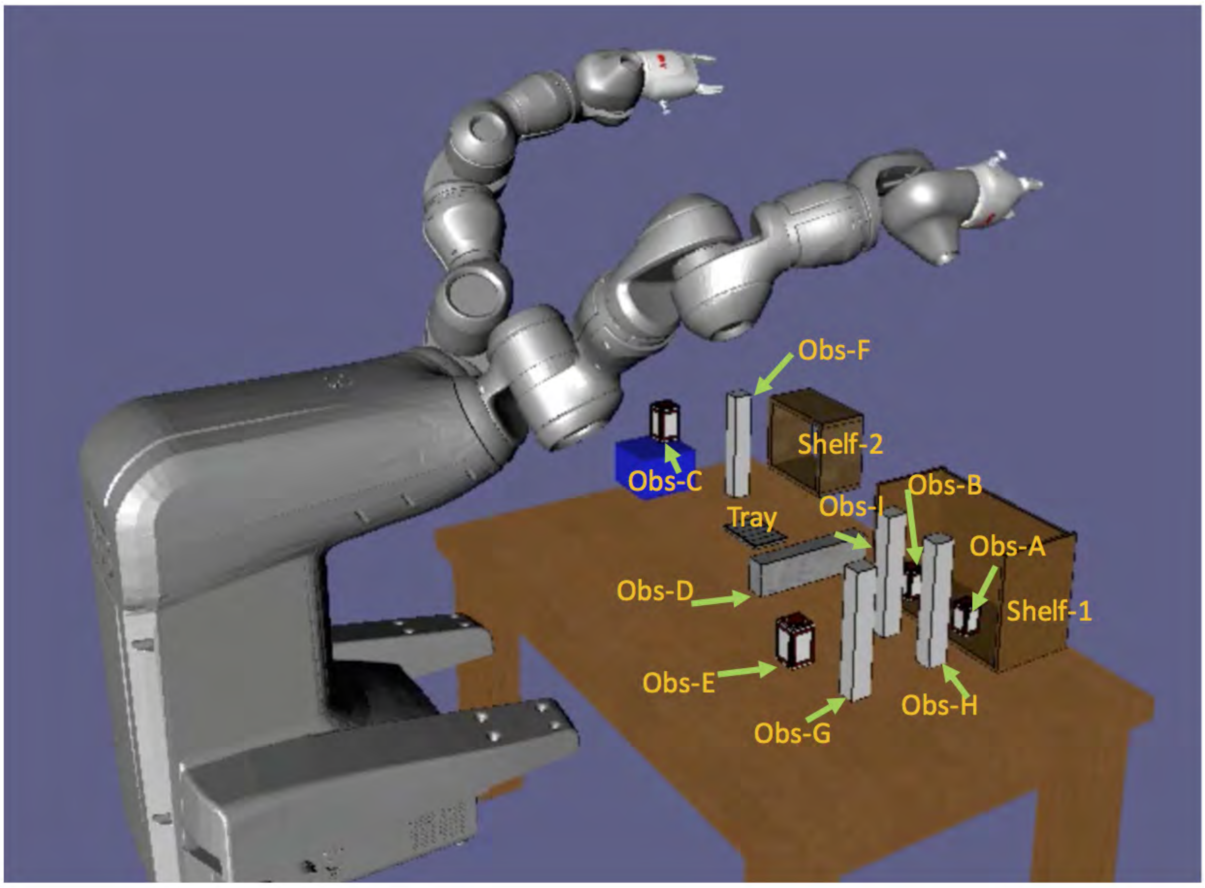


Figure 6.1: A motivating example: the Yumi robot must deliver objects A, B, and I to the tray, and object C within the shelf 2 in the presence of kinematics and placement constraints.

poses, and inverse kinematic solutions; b) geometric reasoning about motions. 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. The approach is originally presented in the research article (Akbari et al., 2018a) for *Pick* and *Place* problems and the extention with *Push* actions is presented in (Rosell et al., 2018).

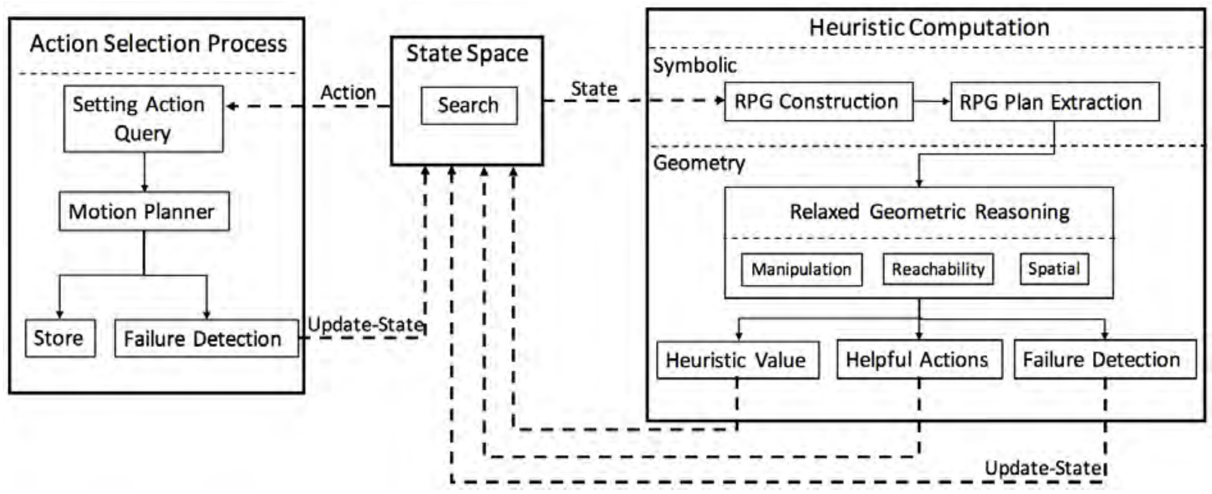


Figure 6.2: The proposed system overview of *TAMP*.

6.2 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. 6.2. 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 literals 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.

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 this chapter is structured as follows. First, Section 6.3 introduces the planning domains, Section 6.4 describes how the relaxed geometric reasoning process is accomplished for symbolic actions, and Section 6.5 illustrates the heuristic computation based on the relaxed geometric reasoning. Then, Section 6.6 demonstrates the planning part in the state space, Section 6.7 explains the possibility of the approach with push actions, Section 6.8 discusses implementation issues and the results of different manipulation problems, and finally Section 6.9 concludes the work.

6.3 Task and Motion Planning Formulation

6.3.1 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 literals.
- $\mathcal{W} = (\boldsymbol{\mathcal{R}}, \boldsymbol{\mathcal{O}})$ 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 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 literals 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:

$$\begin{aligned} S_2.\mathcal{P} &= \{(S_1.\mathcal{P} \cup \mathit{effect}^+(a)) \backslash \mathit{effect}^-(a)\} \\ S_2.\mathcal{V} &= \{(S_1.\mathcal{V} \cup \mathit{geom}^+(a)) \backslash \mathit{geom}^-(a)\} \end{aligned} \tag{6.1}$$

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 to compute heuristic values (see Sec. 6.5). 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:

$$\begin{aligned} S'_2.\mathcal{P}' &= \{S'_1.\mathcal{P}' \cup \mathit{effect}^+(a')\} \\ S'_2.\mathcal{V}' &= \{S'_1.\mathcal{V}' \cup \mathit{geom}^+(a')\} \end{aligned} \tag{6.2}$$

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'.

6.3.2 Manipulation Action Templates

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($\mathcal{O}^m_j$, 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′), $\forall \mathcal{O}_j^m$ ∼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′), $\forall \mathcal{O}_j^m$ ∼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), ∼Holding(Rob, $\mathcal{O}_i^m$, Q), ∀Q′∼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), ∼Clear($\mathcal{O}_j^m$), ∀Q′∼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),* ∼HandEmpty(Rob), ∼On($\mathcal{O}_i^m$, $\mathcal{O}_j^m$, Pos), ∼Clear($\mathcal{O}_i^m$)

Basically, to handle an action failure which could be due to blocking objects (*Critical Objects*), the condition *isCrit* underlies within the preconditions of *Move* and *MoveHolding* actions, i.e., avoiding to move the robot with or without an attached object to the occluded configurations. Then, the condition is able to be negated using the actions *PutDown* and *Stack*.

6.4 Action Evaluation using 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. 6.1, 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. This type of issue can be identified using the manipulation reasoning.

Algorithm 4 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 stores 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 $\mathcal{CO}$ 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 $\mathcal{CO}$.

Algorithm 4: $RelaxGeomReas(\mathcal{W}, a)$

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

6.5 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 placements. Algorithm 5 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.

Algorithm 5: $RPG(S, \mathcal{G})$

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

Evaluating π' [lines 3-17]: The relaxed state is initiated with the current geometric state of the world $S.\mathcal{W}$ and the 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.(6.2). 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 $\mathcal{CO}$, the ground atom *isCrit*($\mathcal{CO}$, *Q*$'$) is added to the current state. Otherwise, the literal *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 extracts 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. 6.1 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 [1] :

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

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:

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

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*

[1]For the sake of space, here just the name of action along with its associated object are represented. For example, *MoveA* is the move action transiting the robot towards the object A and is corresponding with the complete action parameters *Move(RArm,* Q_{init}, Q_{objA}*)*.

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:

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

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.

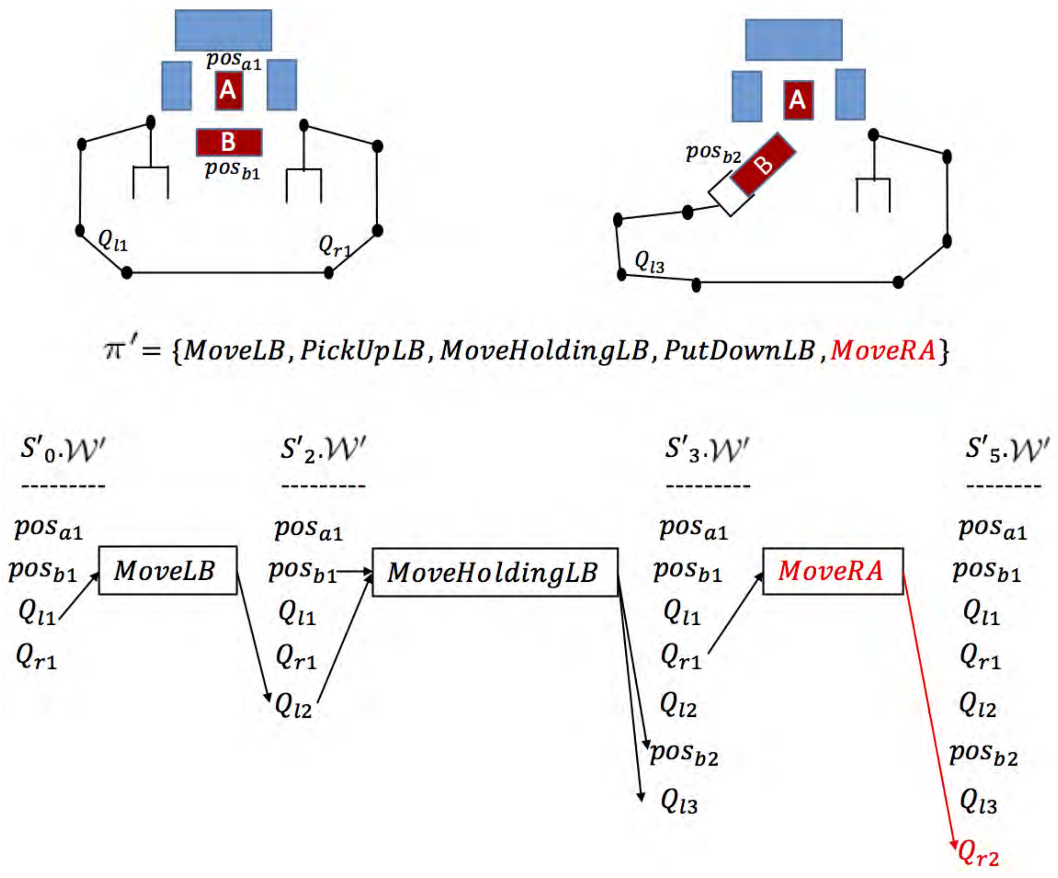
Example 2: Another advantage of considering geometric evaluation in the heuristic computation phase is to report failure in the case of selecting invalid geometric values for the actions. This case avoids the consideration of geometric backtracking which goes to the previous states in order to reselect their geometric values. Let's consider the example represented in Fig. 6.3.

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.

6.6 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 6. 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.(6.1)) 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 5), hence taking geometric constraints into account.

If no actions resulting in a state with lower heuristic value [line 7] can be found, the al-



pos_{a1}
A
B
pos_{b1}
Q_{l1}
Q_{r1}
A
pos_{b2}
B
Q_{l3}
$\pi' = \{MoveLB, PickUpLB, MoveHoldingLB, PutDownLB, MoveRA\}$
$S'_0.\mathcal{W}'$
$S'_2.\mathcal{W}'$
$S'_3.\mathcal{W}'$
$S'_5.\mathcal{W}'$
pos_{a1} pos_{b1} Q_{l1} Q_{r1}
$MoveLB$
pos_{a1} pos_{b1} Q_{l1} Q_{r1} Q_{l2}
$MoveHoldingLB$
pos_{a1} pos_{b1} Q_{l1} Q_{r1} Q_{l2} pos_{b2} Q_{l3}
$MoveRA$
pos_{a1} pos_{b1} Q_{l1} Q_{r1} Q_{l2} pos_{b2} Q_{l3} Q_{r2}

Figure 6.3: 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.

.

gorithm 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($\mathcal{CO}$, Q′)*, otherwise simply with the ground atom *UnReach(Rob, Q′)*, and the algorithm keeps searching until other H_{S_i} are considered.

Algorithm 6: 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: $\pi$
1  $trial \leftarrow 0$
2  $i \leftarrow 0$
3  $\pi \leftarrow \emptyset$
4  $S_{\mathrm{i}} \leftarrow S_{\mathrm{init}}$
5  while $\mathcal{G} \not\subseteq S_i$ do
6      $\{H_{S_{\mathrm{i}}}, S_{i+1}\} \leftarrow$ Search$(S_i, \mathcal{G}, \mathcal{A}))$
7      if $H_{S_i} = \emptyset$ then
8          $trial \leftarrow trial + 1$
9          if $trial < Max$ then
10             $i \leftarrow 0$
11             $S_i \leftarrow$ UpdateInitState$()$
12             Continue
13         else
14             return fail
15     else
16         $\{\mathcal{Q}, Res, \mathcal{CO}\}$ = MotionPlanner$(H_{S_{\mathrm{i}}})$
17         if $Res = True$ then
18             $\pi$.append$(H_{S_{\mathrm{i}}})$
19         else
20             $S_i \leftarrow$ UpdateState$(\mathcal{CO})$
21             Continue
22     $i \leftarrow i + 1$
23 return $\pi$
```

6.7 Extension with Push Actions

The proposed relaxed geometric reasoning has been also extended to consider feasible geometric information for push actions. In this case, the spatial reasoner is responsible to find a valid placement in the same direction in which the requested object is going to be pushed, and then the reachability reasoner is called to find a feasible configuration. Physics-based motion planning is here employed as a motion planner that finds a motion, having interaction with the requested object, between the initial robot configuration and the final one.

To differentiate between *Pick* and *Push* actions, the predicates *isPickable* and *isPushable* are required to be considered in the precondition of *Pick* and *Push* actions respectively. These predicates can be reasoned over the ontological knowledge processing or can be set by a user in the initial state of planning. In the symbolic action space, the action template ***Push****(Rob, $\mathcal{O}_i^m$, Q, Q′, Pos, Pos′)* is defined as follows:

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

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

6.8 Evaluation

6.8.1 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++. The 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.

Relaxed geometric reasoning and motion planning use *The Kautham Project* (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* (Kuffner and LaValle, 2000) 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 work-space. 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).

6.8.2 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.

Regarding the first class of manipulation problems, the scenario explained in Fig. 6.1 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. 6.4. 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. 6.5 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 (Muhayyudin 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. 6.6 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. 6.7. The clutter table problem where the goal is to hold the red object surrounded by 40 objects is displayed in Fig. 6.8. This problem is in essence similar to the problems used in (Srivastava et al., 2014) and (Muhayyudin et al., 2018).

Table 6.1 summarizes the comparison between the current proposal *(HTAMP)* and the work

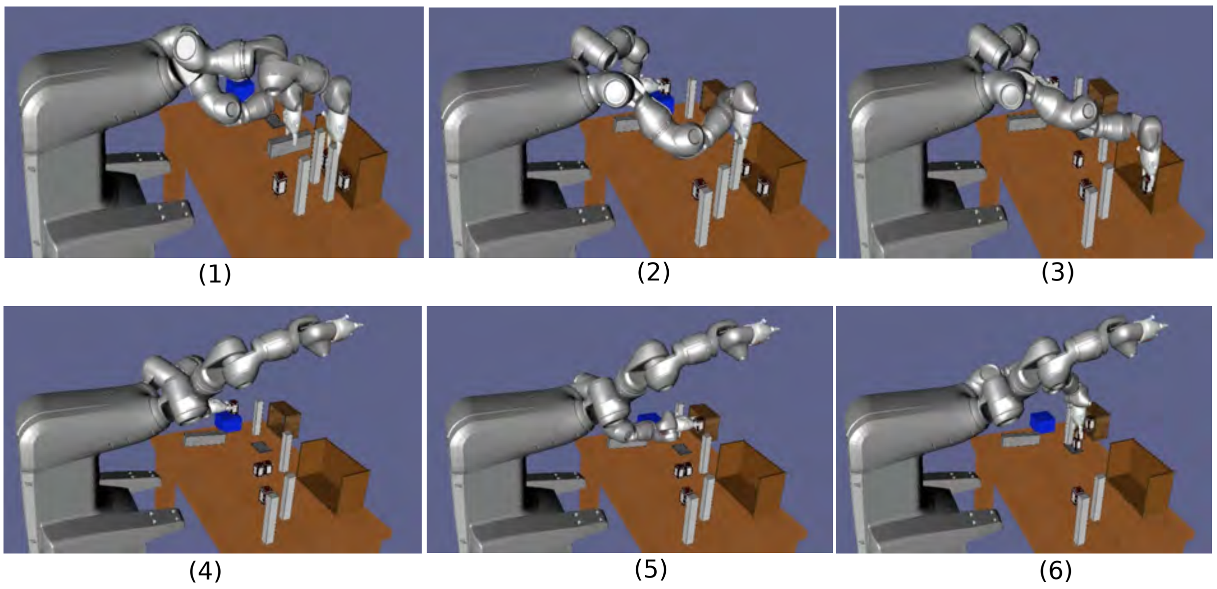


Figure 6.4: The sequence of the snapshots of the execution for the 3D world problem. Video: `https://sir.upc.es/projects/kautham/videos/HTAMP-3D.mp4`

in (Srivastava et al., 2014) (labeled here as *Ap1*) and the work in (Muhayyudin 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 6.2. As it can be seen, the relaxed geometric

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 6.1: Comparison of *HTAMP* with two different approaches. *Ap1* presented in (Srivastava et al., 2014) and *Ap2* presented in (Muhayyudin et al., 2018).

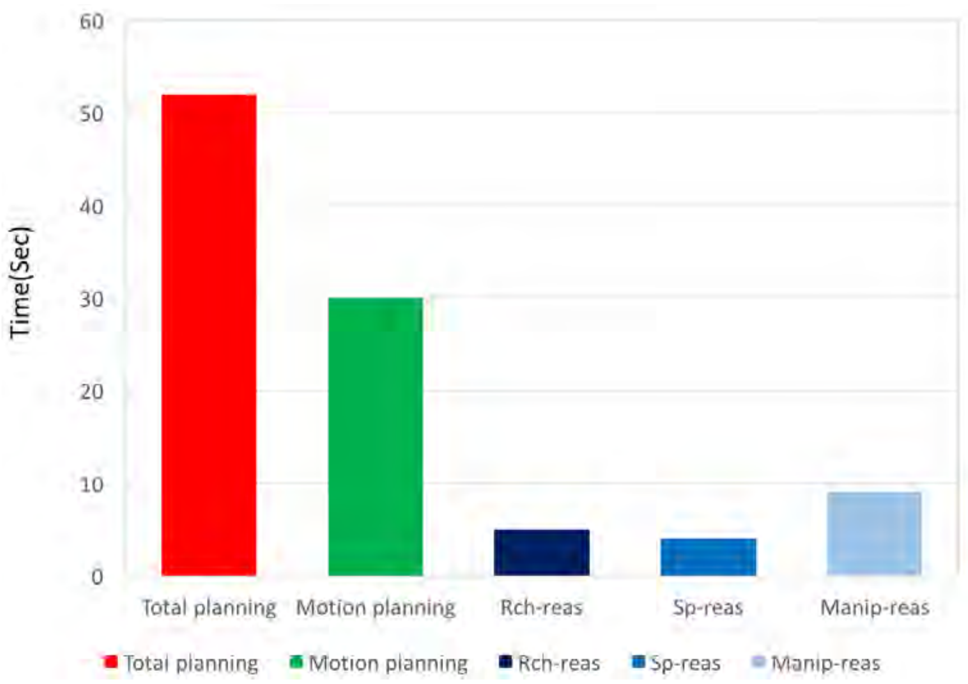


Figure 6.5: The average time related to total manipulation planning and motion planning, and also reachability, spatial, and manipulation reasoning for 30 runs.

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.

Moreover, the proposed planner has been evaluated with two other class of challenging problems which are *Tower of Hanoi* and *Blocks World* proposed in (Lagriffoul et al., 2018). The *Tower of Hanoi* is a classic problem in AI and it has been extended to robot manipulation. The base of the robot is fixed. The rods are set in a triangular fashion and the discs are very thick, so that stacking them on a rod may temporarily create an obstacle and prevent from picking/placing discs on other rods. We have tested the problem with the Yumi robot. This problem evaluates the *large task space* and *infeasible task actions* criteria. Although the rules are

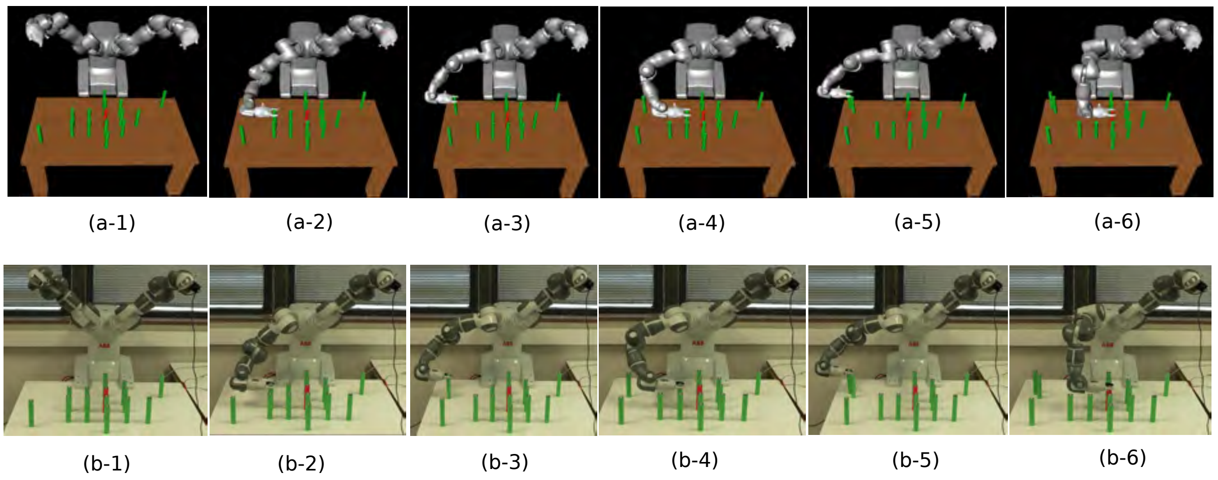


Figure 6.6: The sequence of the execution snapshots for the cluttered manipulation problem where the Yumi robot has to grasp the red object among 15 ones in the environment. The problem has been implemented in simulation (a-1 to a-6) and the real environment (b-1 to b-6). Video: `https://sir.upc.es/projects/kautham/videos/HTAMP-15.mp4`

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

simple, the optimal solution plan for n discs contains $2^n - 1$ steps, without considering geometry (large task space). The rules of the game require certain intermediate states, many of which are geometrically infeasible (infeasible task actions) if the chosen order creates occlusions. The initial state and the goal state of the problem are sketched in Fig. 6.9. The information about the planning domain can be found `http://tampbenchmark.aass.oru.se`.

Blocks World is another classical task planning problem, in which the goal is to stack the blocks in alphabetical order anywhere it is possible. The base is fixed, only top-grasps can be used, and hand-over is not allowed. This problem evaluates the *infeasible task actions*, *large task space*, and *motion/task trade-off* criteria. An obstacle is hovering over the table, such that the grippers would collide with it if more than two blocks are stacked on the table (infeasible task actions). Both initial piles need to be unstacked somewhere and re-stacked on one of the trays. During this process, half of the blocks need to transit from one tray to another, through the table. This requires many actions (large task space). The region on the table where

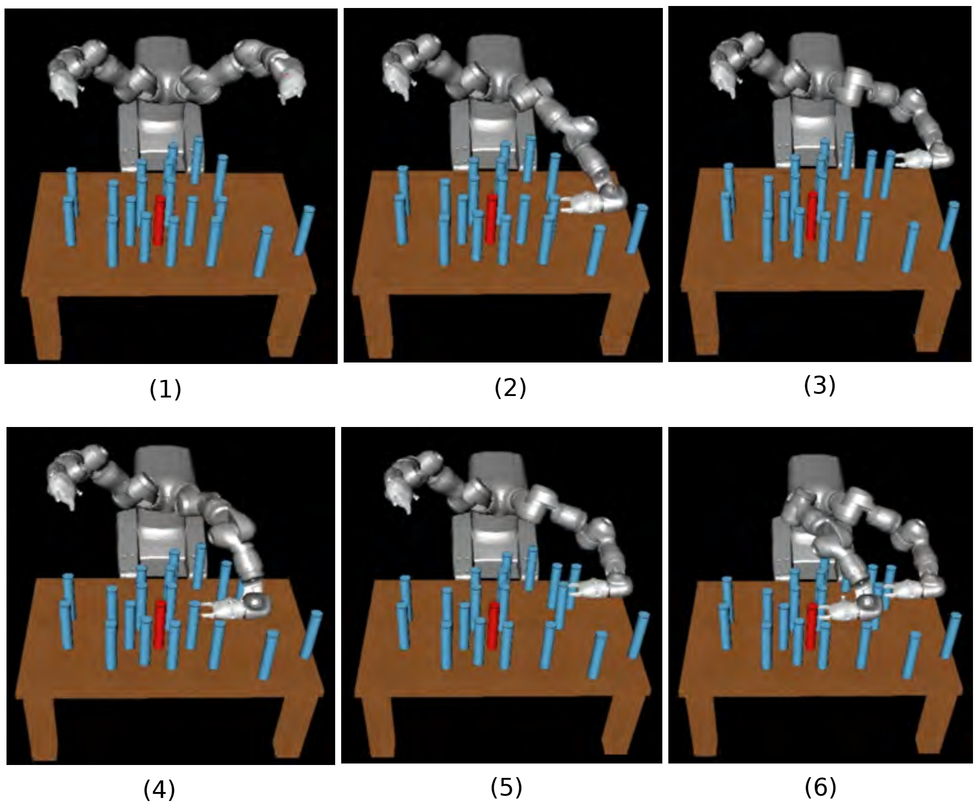


Figure 6.7: The sequence of the snapshots of the execution for the cluttered manipulation problem where the Yumi robot has to grasp the red object among 20 ones in the environment. Video: `https://sir.upc.es/projects/kautham/videos/HTAMP-20.mp4`

blocks transit has to be reachable by both arms; therefore, its size is limited. One must choose between few steps and cluttered table, or many steps and uncluttered table (motion/task trade-off). This has been also tested with the *Yumi* robot. The initial and the goal states of the problem are illustrated in Fig. 6.10. The information about the planning domain can be found `http://tampbenchmark.aass.oru.se`.

The performance of both examples in terms of planning time and selected number of actions in the solution plan is summarized in Table 6.3. *Tower of Hanoi 3* is the problem where three disks are involved and *Tower of Hanoi 6* is the problem in which six disks are considered. *Block World 4* and *Block World 10* are the problems in which four and ten blocks are used.

Besides, we tested our proposal with a manipulation problem which needs *Push* action as well as represented in Fig. 6.11. It is a simple manipulation problem where the robot requires to transfer the cube located on the shelf to the tray. In this scenario, all three levels have been involved. The relaxed geometric reasoning is able to detect the blue object which is blocking the way of reaching the cube on the shelf. The predicates *isPushable* and *isPickable* are inserted in the initial state. According to the predicate *isPushable(blueObj)* found in the state of the planner,

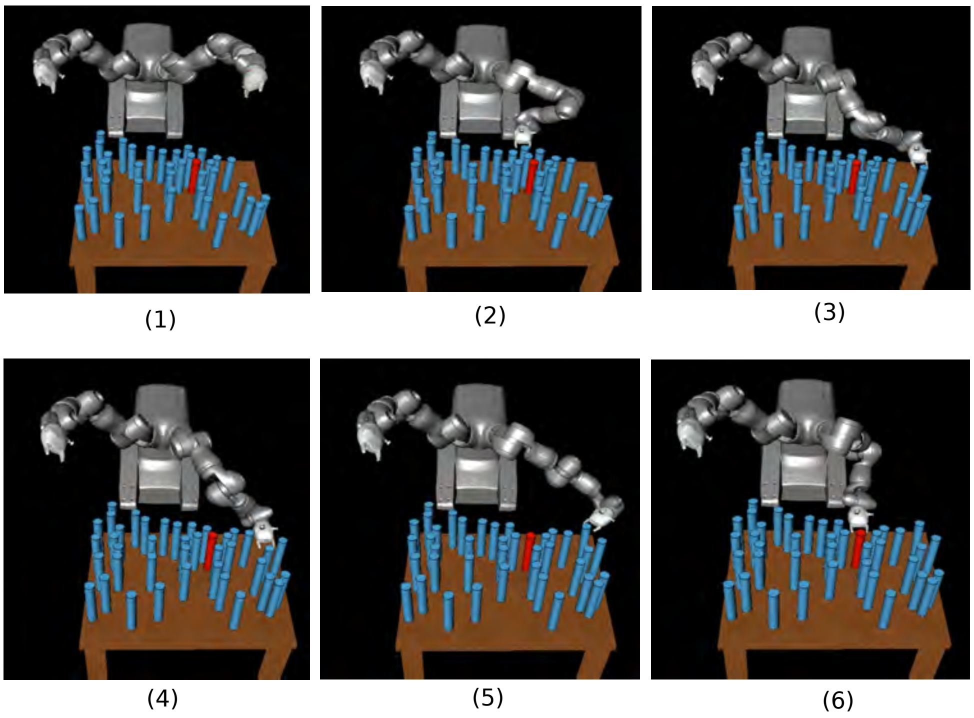


Figure 6.8: Some sequence of the snapshots of the execution for the cluttered manipulation problem where the Yumi robot has to grasp the red object among 40 ones in the environment. Video: `https://sir.upc.es/projects/kautham/videos/HTAMP-40.mp4`

the push action is applied using the physics-based motion planner to the blue object in order to free the access to the cube as the robot is not able to transfer it. transferring the cube to the tray is not able to be performed by the right arm. Therefore, the left arm collaborates in order to locate the target object over the tray.

Problem	Number of selected actions	Planning time
Tower of Hanoi 3	7	71
Tower of Hanoi 6	62	972
Block World 4	8	34
Block World 10	34	547

Table 6.3: The plan length and total planning time of *Tower of Hanoi* and *Block World* manipulation problems.

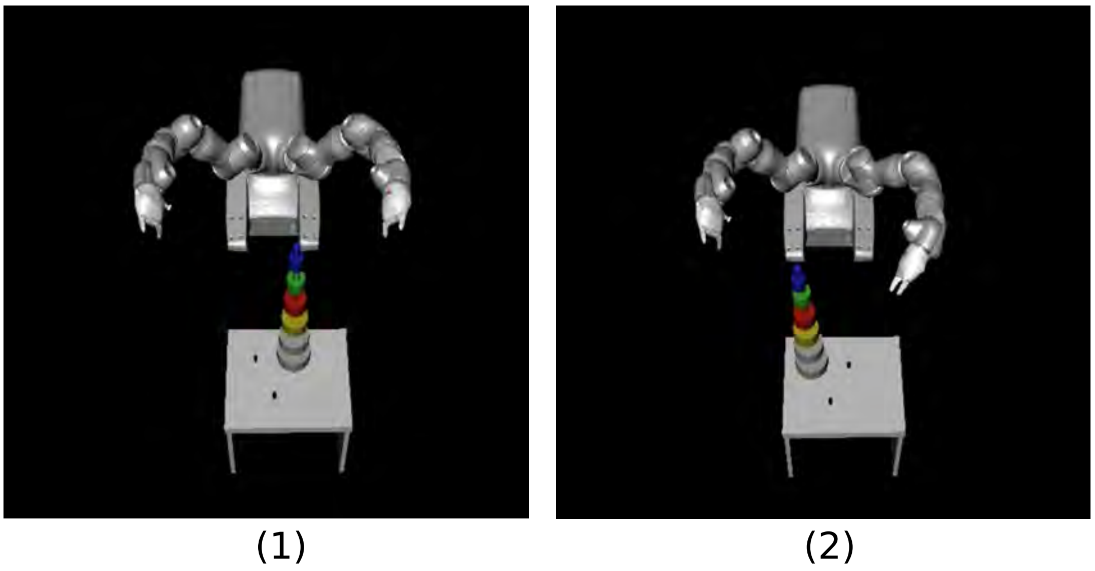


Figure 6.9: Tower of Hanoi problem: a) shows the initial state and b) shows the final state of the solution. Video: `https://sir.upc.es/projects/ontologies/Hanoi.mp4`

6.9 Conclusions

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 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.

Robotics manipulation problems become more challenging when the robot is required to

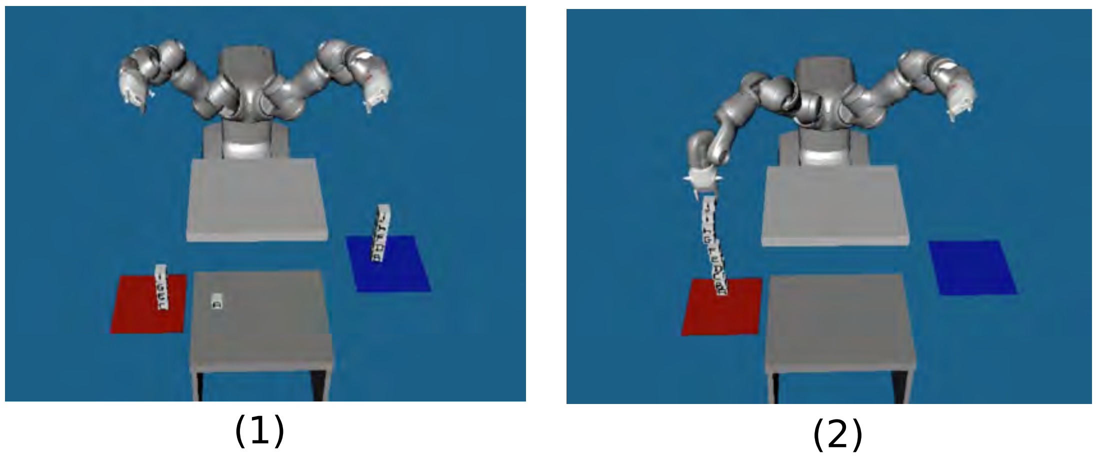


Figure 6.10: Block world problem: a) shows the initial state and b) shows the final state of the solution. Video: `https://sir.upc.es/projects/ontologies/Block.mp4`

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 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.

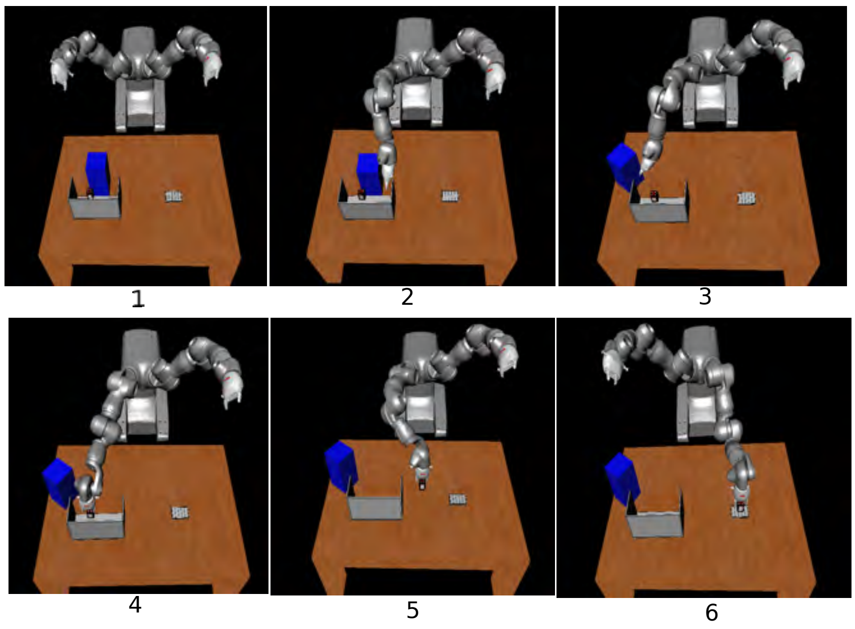


Figure 6.11: Some snapshots for the execution of the scenario. Video: `https://sir.upc.es/projects/ontologies/SIMPAR.mp4`

Chapter 7

Task and Motion Planning for Human-Robot Interactions under Uncertainty

7.1 Introduction

Robotics manipulation tasks become highly challenging when a mobile manipulator is required to obtain a feasible plan to solve a given problem under potential uncertainties. Uncertainty shall be viewed in the initial state of the robot environment, e.g., objects may rest in different positions or some objects features (like color) could be initially unknown for a robot. Uncertainty, moreover, has to be considered in the result of manipulation actions (as nondeterministic effects) since there could be possible action outcomes. To deal with such uncertainties, robots generally look for a sequence of actions for satisfying the goal of a task and perform replanning in the case of action execution failure or uncertain situations. This process may be costly while a robot requires to repeat expensive replanning.

To tackle those challenging issues, these problems can rely on contingent task planning which plans in belief space and has the ability to generate conditional plans under uncertainty in terms of initial state and action effects. Contingent-based task planners are able to provide a tree of plans rather than a number of executive actions. Therefore, uncertainty is observed during the plan execution, and accordingly the tree of plan is guided according to the binary observation values.

In general, task planners do not involve geometric information, so they may result in infeasible plans which cannot be executed by the robots due to geometric failures. This is the reason why geometric reasoning over symbolic actions has to be carried out with the purpose of identifying geometric constraints if any. For instance, objects may not be reachable directly or placed in their placement regions because of some geometric constraints in the environment. By adding a mobile base to a manipulator, the manipulation capabilities of the system are increased although an extra effort is required in the geometric reasoning in order to also consider the pose of the base.

There are some demanding or difficult tasks which are either not performable easily by robots or are out of their capability. In some cases, robots may also need to access some objects located in the human workspace. In these cases, robot can ask human as a collaborator to do some actions. Moreover, there could be various situations such that some type of knowledge about the environment is initially incomplete for the robot. Human can share the knowledge to the robot so that it is able to solve its task under certain amount of knowledge.

In this chapter, we are going to deal with manipulation tasks carried out by a mobile manipulator assisted by a human operator. The mobile manipulator will be responsible to execute the main task, while the human operator will be responsible some difficult actions, to transfer knowledge to the robot, to open some box-like containers which are not able to be opened by the robot, and to transfer objects to the robot where they are not reachable. Uncertainty in the initial state and in some action effects are considered. Some manipulation and sensing actions are considered in the current proposal which allow to illustrate the approach, which could be extended to handle a broader set of manipulation tasks. No geometric uncertainty is considered, e.g., in the robot motion or the object poses.

Contributions: To deal with aforementioned challenges, we propose a contingent-based task and motion planner based on *Contingent-FF* that works under uncertainty and considers human-robot collaboration. The *Contingent-FF* includes two main components, *heuristic evaluation* and *search space*, and results in a tree-shaped set of plans involving sensing actions. Here, a list of contributions are summarized that extends the basic *Contingent-FF* planner:

- *Robot action reasoning.* Two types of geometric reasoning are proposed: *relaxed geometric reasoning* and *lazy motion evaluation*. The former refers to *Reachability*, *Spatial*, and *Manipulation* reasoning. This reasoning process is embedded within the heuristic computation of the planner. Motion paths are lazily evaluated when actions are selected by the space search. If the reasoning processes fails, geometric constraints are fed back to the planner. This part of computation is done offline and aims to prune infeasible actions due to geometric constraints and to result in feasible actions in the tree of plans.

- *Human-robot collaboration.* There are some common actions which can be executed either by the robot or a human operator. The proposed relaxed geometric reasoning is also extended to inform the planner about which actions cannot be executed by the robot, and hand over them to the human operator. Furthermore, several cases may exist in which the planner decides to select actions required to be performed by a human operator. In these cases, the geometric world resulting by these actions is simulated and used in our planning system for further geometric reasoning evaluation.

- *Sensing action observation.* To observe the binary values of sensing actions, two modules are proposed: *perception* and *human knowledge*. Perception is used to detect the actual pose of objects and some objects feature (like color). The robot can also interact with a human operator to deal with its incomplete information, e.g., a can is filled or which glasses contain which drinks, etc. This part of computation is done online.

  One of the main advantages of the proposed framework is that the offline computation is valid and works despite the actual values of the uncertainties variables or the actual outcomes of the executable actions.

7.2 Contingent Task and Motion Planning Framework

This section describes a brief overview about the modifications applied to the *Contingent-FF* task planner in order to compute feasible manipulation conditional plans, and accordingly to consider observations on sensing actions in practice.

7.2.1 Planning Formulation

Our planning system domain $\mathcal{D}$ is a tuple $\langle \mathcal{A}, \Omega, \mathcal{F}, \mathcal{W}, S_g \rangle$ where $\mathcal{A}$ is the action space, Ω is the sensing action space, $\mathcal{F}$ a set of literals, $\mathcal{W} = (\boldsymbol{\mathcal{R}}, \boldsymbol{\mathcal{O}})$ is a workspace including a mobile manipulator $\boldsymbol{\mathcal{R}}$ (described by the pose of the base Pos_{rob} along the arm configuration Q_{rob}) and a set of movable and fixed objects $\boldsymbol{\mathcal{O}}$, and, 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 are denoted by *pos*, and whose features are denoted by *fe*.

An action $a \in \mathcal{A}$ is a tuple $\langle$*name(a), pre(a), effect(a), coneffect(a), geom(a)*, $\mathcal{Q}(a)\rangle$, where *name(a)* is the action symbolic name, *pre(a)* is a propositional formula which must hold for the action to be applied, *geom(a)* is the numerical counterpart of an action containing geometric information, *effect(a)* represents the negative and positive effects of a on the state it is applied to, and $\mathcal{Q}(a)$ is a query function to the motion planner which computes a motion between two robot configurations and stores the solution if any. A relaxed action $a' \in \mathcal{A}'$ (where $\mathcal{A}'$ is the relaxed action space) is similar to the action despite it does not consider any negative effects. Actions are executable action and can be done by either the robot or a person. Six actions types are considered to deal with some examples of mobile robot manipulation as follows:

- *Transit*: an action done by the robot to travel from one configuration to another one without an attached object.
- *Transfer*: an action done by the robot to move an attached object from one pose to another one.

- *Push*: an action done by the robot to push an object from one pose to another one.
- *Open*: an action done by the robot to open a box-like container (articulated cap with prismatic joint is assumed with two positions corresponding to fully closed and fully opened that will be stored in the containers objects features).
- *HumanManipulation*: an action done by a person to transfer/push an object to the robot workspace.
- *HumanOpen*: an action done by a person to open a box-like container.

Each sensing actions is a tuple ⟨*pre(a), o(a)*⟩, where *o(a)* is a literal having uncertainty. They are actions that do not involve motion devoted to observe the value of *o(a)*. The observation is done in run-time. Some sample sensing actions are considered in the proposed planning system. They are the following:

- *SenseColor*: a sensing action is done by a perception module to determine the color of an object.
- *SensePose*: a sensing action is done by a perception module to determine the pose of an object.
- *CheckContainer*: a sensing action is done by a person to evaluate whether a container is open or not.
- *CheckCan*: a sensing action is done by a person to evaluate whether can-like objects are filled or not.

A belief state S is a tuple $S = \langle \mathcal{P}, \mathcal{V} \rangle$ where $\mathcal{P}$ includes a set of known literals which hold in that state and a set of uncertain literals which may hold or not in the state, and $\mathcal{V}$ represents a full geometric description of the scene, i.e., configurations of robots and poses of objects corresponding to certain and uncertain literals. The executable action from a state S_1 results in a new world state using the transit functions $S_2.\mathcal{P} := S_1.\mathcal{P} - \mathit{effect}^{-}(a) + \mathit{effect}^{+}(a)$ and $S_2.\mathcal{V} := S_1.\mathcal{V} - \mathit{geom}^{-}(a) + \mathit{geom}^{+}(a)$. The sensing action splits a belief state and introduces two branches into the plan marked with *o(a)*, and also ¬*o(a)*.

A belief relaxed state $S' = \langle \mathcal{P}', \mathcal{V}' \rangle$ is similar to the belief state, where $\mathcal{P}'$ and $\mathcal{V}'$ contain the literals and geometric information of the manipulation workspace updated using the relaxed transition function (that not consider negative effects) $S'_2.\mathcal{P}' := S'_1.\mathcal{P}' + \mathit{effect}^{+}(a')$ and $S'_2.\mathcal{V} := S'_1.\mathcal{V}' + \mathit{geom}^{+}(a')$.

The 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 literals 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 under uncertainty is a hybrid tree-shaped conditional plan, i.e., a sequence of symbolic actions achieving $\mathcal{G}$, along with a feasible motion for each action, which we denote by π.

7.2.2 Geometric Constrains Predicates

Basically, three general predicates are allocated that set constraints, evaluated by geometric reasoning, to the state of the proposed planner if required: *isCrit($\mathcal{O}_j^m$, $\mathcal{O}'$, Pos)*, *infeasByRob($\mathcal{R}$, $\mathcal{O}'$, Pos)*, and *assist(Human, $\mathcal{O}'$, Pos)*. The first predicate indicates there is the blocking object $\mathcal{O}_j^m$ which is located towards the target object $\mathcal{O}'$ placed in the pose Pos. The second one shows the target object cannot be manipulated by the robot $\mathcal{R}$ in the corresponding pose Pos. Also, the last predicate shows the manipulation action with the target object $\mathcal{O}'$ and the corresponding Pos must be done by a human operator *Human*.

The proposed predicates are interleaved inside the actions conditions, similarly to the action templates representation described in chapter 6. Concerning the actions done by the robot, the predicates *∼isCrit* and *∼infeasByRob* are inserted within the preconditions of the actions *Transit, Open, Transfer,* and *Push* in order to avoid moving the robot, with or without holding an object, to any unreachable or infeasible configurations. However, the last two actions are able to negate the predicate *isCrit* if any. With respect to the actions performed by human, the preconditions of the actions *HumanTransfer* and *HumanOpen* are integrated with the predicate *assist* in order to know when human can collaborate with robot. For instance, the actions *Transit* and *HumanOpen*, considering some of the geometric constraints, are described below.

The action template *Transit($\mathcal{R}$, $\mathcal{O}_i^m$, Surface, Pos)* is designed to move the robot arm and base $\mathcal{R}$ with no object towards a grasp configuration of a manipulatable object $\mathcal{O}_i^m$ located in a surface *Surface* with the pose Pos. In the final configuration, the robot holds the target object. The action is applicable if the following preconditions hold: the object is located on a surface, top of the object is clear, the robot arm is empty, it can reach the grasp configuration if there is no movable objects blocking its way to $\mathcal{O}_i^m$. The last precondition is represented by fact *isCrit($\mathcal{O}_j^m$, $\mathcal{O}_i^m$, Pos)*; objects that make this fact to hold are called *Critical Objects* which are the objects blocking the way of reaching the object. As a result of the action, the robot holds an object.

***Transit**($\mathcal{R}$, $\mathcal{O}_i^m$, Surface, Pos):*

Pre: *onSurface($\mathcal{O}_i^m$, Surface, Pos), armEmpty($\mathcal{R}$), clear($\mathcal{O}_i^m$), ∼infeasByRob($\mathcal{R}$, $\mathcal{O}_i^m$, Pos), $\forall\mathcal{O}_j^m$ ∼isCrit($\mathcal{O}_j^m$, $\mathcal{O}_i^m$, Pos)*
Effect: *holding($\mathcal{R}$, $\mathcal{O}_i^m$, Pos), ∼clear($\mathcal{O}_i^m$), ∼armEmpty($\mathcal{R}$), ∼onSurface($\mathcal{O}_i^m$, Surface, Pos)*

The action template *HumanOpen(Human, $\mathcal{O}_i^m$, Pos, Closed, Open)* is used to open a box-like container $\mathcal{O}_i^m$ when it is closed. The action is applicable if the robot needs the assistance from a human operator that is represented by the *assist* predicate, and its status is closed shown by the predicate *status*. These conditions are within in the action preconditions. As a result of the action, the corresponding container will be open.

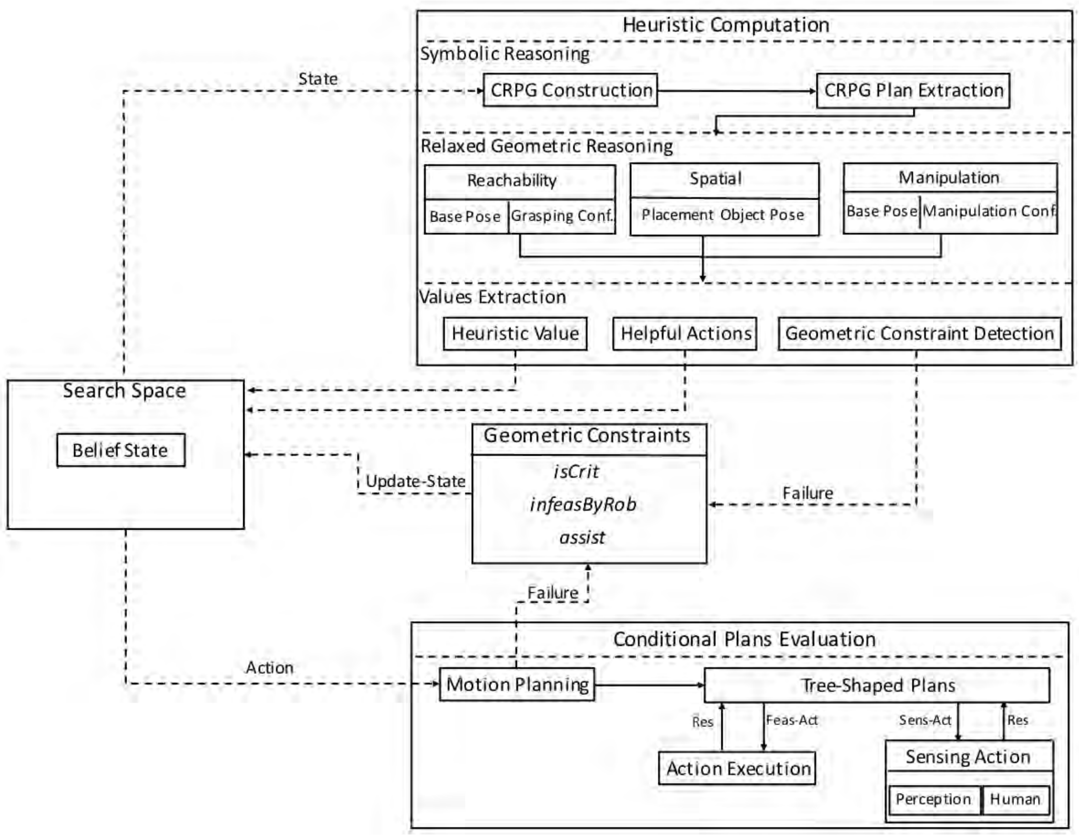


Figure 7.1: The proposed system overview of contingent task and motion planning using the extended version of *Contingent-FF*.

HumanOpen*(Human, $\mathcal{O}_i^m$, Pos, Closed, Open):*

Pre: *assist(Human, $\mathcal{O}_i^m$, Pos), status($\mathcal{O}_i^m$, Closed)*
Effect: *status($\mathcal{O}_i^m$, Open), ∼status($\mathcal{O}_i^m$, Closed)*

7.2.3 The Proposed Framework System

The proposed task and motion planning under uncertainty framework extends the basic *Contingent-FF* planner that aims to incorporate different sort of geometric reasoning, observations on sensing actions, as well as human-robot collaboration within planning. The overview of the system is sketched in Fig. 7.1. It involves three main parts: *Heuristic Computation, Space Search*, and *Conditional Plans Evaluation*.

Heuristic Computation provides a heuristic value and a set of helpful actions for a given belief state. The standard *CRPG* is first constructed (as detailed in Section 2.2) and the corresponding relaxed plan is extracted. The relaxed plan is then fed into the relaxed geometric reasoner which evaluates it versus certain geometric constraints, i.e., reachability, collisions, manipulation constraints and graspability. As some actions may be considered for a human, the geometric reasoner simulates the result of those actions, to be used later for evaluation of other actions. The heuristic value is returned along with helpful actions if such constraints are met. If a constraint is violated, the associated belief state is updated with facts 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.

Space Search keeps the standard search procedure of the *Contingent-FF* planner that is based on the *And-Or* search strategy. From each belief state, the action resulting in the state with lowest heuristic value is selected and is a candidate to be added to the conditional plans. The only difference is that the heuristic value now accounts for geometric constraints.

Conditional Plans Evaluation intents to find a motion solution for a candidate action at first. Using information from the relaxed geometric reasoning, it is possible to geometrically define feasible start and goal configurations for the *RRT-Connect* motion planner and compute a motion. If motion planning fails, the current belief state is updated with the cause of failure and the search resumes. Otherwise, the action is added to the tree-shaped conditional plans at hand. After finding the complete conditional plans, feasible actions are executed by a mobile robot or human in the real world and sensing actions are observed either using a perception module or the information provided by a human in run-time. Therefore, the robot plan is able to determine the correct branch in order to follow up its plan.

In general, the geometric failure could be due to collisionable objects, so the literal *isCrit* is added to the belief state. If the failure is because of fixed obstacles blocking the way of reaching an object, inverse kinematic problems, or motion planning time-out problem, the literal *infeasByRob* is added to the state. In such failure, if the type of the evaluated action is either transit or open, the literal *assist* is also inserted.

7.3 Relaxed Geometric Reasoning for Mobile Manipulators

Relaxed geometric reasoning is the evaluation of geometric conditions of actions with no call to motion planning. It indicates that a feasible motion is likely to be obtained for the selected actions if certain task constraints are satisfied. This is the extended relaxed geometric reasoning referred in chapter 6 to include mobile manipulation tasks. So, the relaxed geometric reasoning process contains three modules: *reachability reasoning* which looks for a feasible kinematic solution and pose for the robot to reach an object, *spatial reasoning* which determines a valid object placement for the robot or human actions, and *manipulation reasoning* which looks for a feasible kinematic solution to manipulate objects. They are explained in details as follows.

Reachability reasoning ($\mathcal{R}_{rch}$): This reasoning is applied for the *transit* action. To transit the robot to a location described by a set of potential pre-grasping poses, it is first required to obtain a valid goal configuration and robot pose. A set of robot poses is considered in the workspace of the robot. From each pose, an *Inverse Kinematic (IK)* solver is called for each potential pre-grasping pose, and then 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 associated pre-grasping pose. Failure may occur if no *IK* solution exists or if no collision-free *IK* solution exists through all tested robot poses. In this latter case, the reasoning procedure reports the collisionable objects, ordered such that those that are manipulatable are returned first.

Spatial reasoning ($\mathcal{R}_{sp}$): This reasoning is considered for the *transfer, push, open, humanOpen*, and *humanTransfer* actions. It is used to investigate 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 constraints are taken into consideration while the sample placement is accomplished. For example, in the case of the *push* action, the sample is considered in the direction in which the object has been grasped. In the case of *humanOpen* and *Open*, the valid object placement is extracted from the object feature. In the case of *humanTransfer* action, some predefined poses are considered in the robot workspace and the reasoner selects the feasible one.

Manipulation reasoning ($\mathcal{R}_{mnp}$): This reasoning 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 way, it is able to obtain the valid robot pose and grasping configuration when the robot manipulates an object.

Algorithm 7 describes the relaxed geometric function when applying the actions 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. Otherwise, the algorithm returns in $\mathcal{CO}$ the set of objects whose collision causes the failure of the action if any. Algorithm is detailed below:

- Reasoning about the robot actions [lines 5-15]: The *transit* action calls the reachability reasoning by the function $\mathcal{R}_{rch}(\mathcal{W}, a)$ [line 6]. The *transfer, push* and *open* actions call the spatial reasoning by the function $\mathcal{R}_{sp}(\mathcal{W}, a)$ [line 11], and then call the function $\mathcal{R}_{mnp}(\mathcal{W}, a, \mathcal{O}_i^m(pos_{goal}))$ [line 13]. If the reasoning processes are successfully done, the corresponding response is set to feasible and geometric details are appended to the evaluated action [line 8] and [line 15]. On the contrary, if the failure is due to manipulatable objects, the response is set to *infeasible-criticalObjects*. and the collisionable objects are stored. In other cases of failure, the response is set to *infeasible-infeasByRob* that could be because of collisions with fixed obstacles or the *IK* module is not able to find a configuration.

- Reasoning and finding the geometric values of the human actions [lines 16-23]: The *humanTransfer* action calls the spatial reasoning by the function $\mathcal{R}_{sp}(\mathcal{W}, a)$ [line 17]. This function is responsible to find the pose of the object placement for the human action and inserts it to the action [line 19]. For the *humanOpen* action, the pose of the container object being opened is extracted from the object feature by the spatial reasoner function $\mathcal{R}_{sp}(\mathcal{W}, a)$ [line 21] and is stored into the action details [line 23].

Algorithm 7: $RelaxGeomReas(\mathcal{W}, a)$

```
1  $CO \leftarrow \emptyset$
2  $a.geom^{+} \leftarrow \emptyset$
3  $i \leftarrow 0$
4  $Res = False$
5  if $a$.name = Transit then
6      $\{Res, Q_{rob}, Pos_{rob}, CO, g\} = \mathcal{R}_{rch}(\mathcal{W}, a)$
7      if $Res = feasible$ then
8          $a.geom^{+}.add(Q_{rob}, Pos_{rob}, g)$
9  else if $a$.name = Transfer or Push or Open then
10     while $i < Max$ do
11         $\{Res_{sp}, \mathcal{O}_j^m(pos_{goal}), CO\} = \mathcal{R}_{sp}(\mathcal{W}, a)$
12         if $Res_{sp} = feasible$ then
13             $\{Res, Q_{rob}, Pos_{rob}, CO, g\} = \mathcal{R}_{mnp}(\mathcal{W}, a, \mathcal{O}_i^m(pos_{goal}))$
14             if $Res = feasible$ then
15                 $a.geom^{+}.add(Q_{rob}, Pos_{rob}, \mathcal{O}_j^m(pos_{goal}), g)$
16 else if $a$.name = HumanTransfer then
17     $\{Res, \mathcal{O}_j^m(pos_{goal})\} = \mathcal{R}_{sp}(\mathcal{W}, a)$
18     if $Res = feasible$ then
19         $a.geom^{+}.add(\mathcal{O}_j^m(pos_{goal}))$
20 else if $a$.name = HumanOpen then
21     $\{Res, \mathcal{O}_j^m(pos_{open})\} = \mathcal{R}_{sp}(\mathcal{W}, a)$
22     if $Res = feasible$ then
23         $a.geom^{+}.add(\mathcal{O}_j^m(pos_{open}))$
24 else
25     //$a$ is not required to be checked;
26     return Null
27 return $\{a.geom^{+}, Res, CO\}$
```

7.4 Contingent 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, manipulation with constrained objects, and object placement. Algorithm 8 illustrates (for a given belief state S and goal $\mathcal{G}$) how the computation of the heuristic and helpful actions of the standard *Contingent-FF* is modified to include geometric information. This is done in three steps: computing the *CRPG* and the relaxed plan π', evaluating π', and computing the heuristic value along the helpful actions, as detailed next.

Computing the CRPG and π' [lines 1-2]: At the beginning, the *CRPG* graph *CRPG*$_{gr}$ containing state layers and action layers is constructed by the function CRPGConst [line 1]. The function CRPGPlan extracts π' from that graph [line 2]. This process is done as the standard *Contingent-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 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] using the relaxed transition function. 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 satisfying the precondition of the corresponding 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 *CRPG* 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 failure due to *infeasible-criticalObjects*, the literal *isCrit*($\mathcal{CO}$, $\mathcal{O}'$, Pos) with critical objects is added to the current belief state. Otherwise, the failure is because of *infeasible-infeasByRob* and the literal *infeasByRob*($\mathcal{R}$, $\mathcal{O}'$, Pos) is added to the state. In this case, if the type of action is either transit or open, the literal *assist*(*Human*, $\mathcal{O}'$, Pos) is also added to the state.

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

Algorithm 8: $CRPG(S, \mathcal{G})$

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

7.5 Tree-based Planning using Search Space

A manipulation plan is found using the *And-Or* state space search as in the *Contingent-FF* algorithm. The difference lies in the heuristic function which includes geometric reasoning, and the fact that action selection must be confirmed by the *RRT-Connect* motion planner. The pseudocode of this process is presented in Algorithm 9.

Similar to the approach presented in chapter 6, 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. The algorithm gets $\mathcal{T}$ as input and returns π if applicable. Initially, the trials counter $trial$ is initialized [line 1] and the state S_i gets the initial belief state [line 4]. The function Search implements the standard *Contingent-FF* search process [line 6]: it returns the next state to visit S_{i+1} along with the selected helpful action(s) or action(s) with $H_{S_{\mathrm{i}}}$. This value is computed with the modified CRPG function (see Algorithm 8), by taking geometric constraints into account.

If there is no such $H_{S_{\mathrm{i}}}$ action [line 7], 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].

For those actions that either do not belong to the set of sensing actions and are not assigned to human, the MotionPlanner function is used to compute a collision-free path for the currently selected action(s) [line 17]. If a path is found, Res is set to *feasible* and the path $\mathcal{Q}$ is returned. Afterwards, π is appended with the sensing, human, or normal action(s) [line 19]. In the case of failure due to *infeasible-criticalObjects*, the literal *isCrit(*$\mathcal{CO}$*,* $\mathcal{O}'$*,* Pos*)* with critical objects is added to the current belief state. Otherwise, the failure is because of motion planning time-out problem or collisionable fixed obstacles, the type of failure is *infeasible-infeasByRob* and the literal *infeasByRob(*$\mathcal{R}$*,* $\mathcal{O}'$*,* Pos*)* is added to the state. In this case, if the type of action is either transit or open, the literal *assist(Human,* $\mathcal{O}'$*,* Pos*)* is also added to the state.

7.6 Manipulation Plan Execution using Sensing and Human Interaction

When the manipulation conditional plan is achieved, it will be forwarded to for the execution module. Algorithm 10 outlines the process of actions execution performed by the robot or human, and also call to the sensing actions. The conditional plan is initialized from its root [line 1]. For each action of the plan, its type first identified whether it is execution or sensing one. In the case of execution action, if it has to be executed by human, the function executeByHuman

Algorithm 9: The Proposed Planning Algorithm

```
inputs : $\mathcal{T}=\langle \mathcal{D}, \mathcal{S}_0, \mathcal{G} \rangle$, $\mathcal{D}=\langle \mathcal{A}, \Omega, \mathcal{F}, \mathcal{W}, S_g \rangle$
output: $\pi$
1  $trial \leftarrow 0$
2  $i \leftarrow 0$
3  $\pi \leftarrow \emptyset$
4  $S_{\text{i}} \leftarrow S_{\text{init}}$
5  while $\mathcal{G} \nsubseteq S_i$ do
6      $\{H_{S_{\text{i}}}, S_{i+1}\} \leftarrow$ Search$(S_i, \mathcal{G}, \mathcal{A}, \Omega))$
7      if $H_{S_i} = \emptyset$ then
8          $trial \leftarrow trial + 1$
9          if $trial < Max$ then
10             $i \leftarrow 0$
11             $S_i \leftarrow$ UpdateInitState()
12             Continue
13         else
14             return fail
15     else
16         if $H_{S_i} \notin \Omega$ And $H_{S_i}.name \neq$ HumanTransfer And $H_S.name \neq$ HumanOpen then
17             $\{\mathcal{Q}, Res, C\mathcal{O}\}$ = MotionPlanner$(H_{S_{\text{i}}})$
18         if
             $H_{S_i} \in \Omega$ Or $H_{S_i}.name \neq$ HumanTransfer Or $H_S.name =$ HumanOpen Or $Res = feasible$
           then
19             $\pi$.append$(H_{S_{\text{i}}})$
20         else
21             $S_i \leftarrow$ UpdateState$(Res, C\mathcal{O})$
22             Continue
23     $i \leftarrow i + 1$
24 return $\pi$
```

asks a person to do the corresponding action [line 5]. Otherwise, the action is executed by the robot [line 7].

On the other hand, if the type of action becomes sensing, the function senseAct determines the binary value of the sensing action which is *True* or *False*. Depending on the type of uncertainty, the function may request to human or activate a sensing module in order to observe the action value. Regarding the *CheckCan* or *CheckOpen* sensing actions, human information is used, while a sensing module is used for the *SensePose* and *SenseColor* sensing actions.

Algorithm 10: Manipulation Plan Execution

```
   inputs : $\pi$
1  initializePlan($\pi$)
2  foreach {$H_S \in \pi$} do
3      if $H_S \in \mathcal{A}$ then
4          if $H_S.name$ = HumanTransfer Or HumanOpen then
5              executeByHuman($H_S$)
6          else
7              executeByRob($H_S$)
8      else
9          $Res$ = senseAct($H_S$)
10         selectBranch($\pi, H_S, Res$)
```

7.7 Empirical Results and Discussion

This section describes some manipulation problems solved using the proposed framework, and also the implementation issues. The mobile robot considered is *TIAGo*. It has 7 degrees of freedom arm, equipped with a gripper, mounted on a mobile platform through a lift torso. The scene depicted in Fig. 7.2, called *Problem-1*, shows the initial belief state of the mobile manipulation problem. The uncertainty considered for the gray objects and the can, i.e., whether the containers are open or not, the color of the gray cylinder is red or green, and the can is empty or filled. The goal is to transfer the cylinder A to either the red or the green tray. Some particular placements regions allocated for the manipulatable objects if required:

- The green cylinders must be just placed on the green tray.
- The red cylinders have to be only placed over the red tray.
- The blue cylinders must be placed only within the containers.
- The can objects may be optionally placed anywhere over the table.

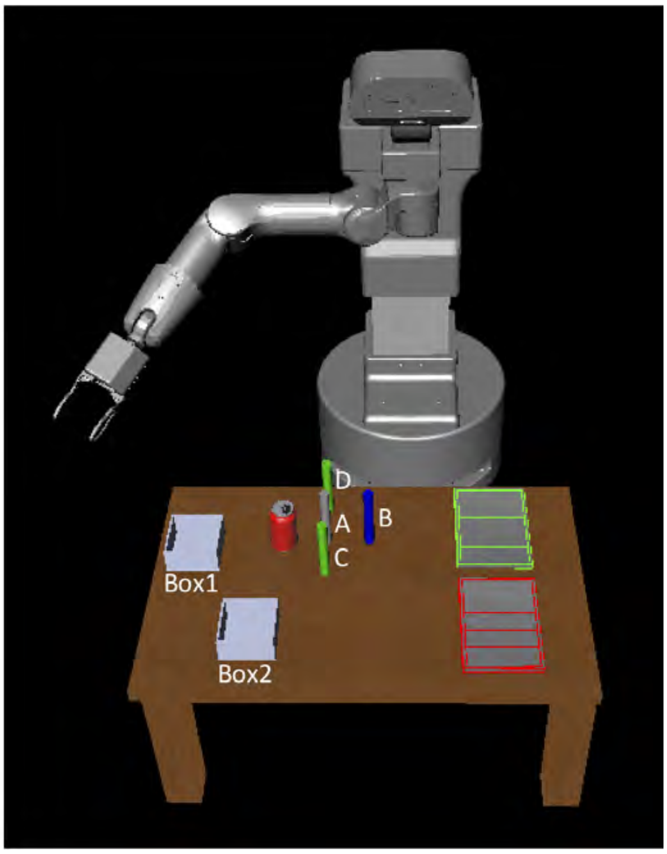


Figure 7.2: The manipulation example where the goal is to transfer cylinder A to one of the trays with respect to its color.

The complete conditional plan is represented in Fig. 7.3. It is briefly discussed how this plan is obtained. While the planning process is taken place, there are several challenges in terms of geometric constraints which are captured and handled by the proposed geometry reasoner. To reach the target object, the reachability reasoning process detects cylinder B and reports that the object is blocking the way of reaching object A in the heuristic computation. Therefore, the predicate *isCrit(cylinderB, cylinderA, posA)* is inserted to the initial belief state of of the planner. This predicate says that cylinder B blocks the way of reaching cylinder B. Furthermore, when the robot attempts to find a feasible configuration for opening box 1 in the case that the box is closed, the reachability reasoner fails due to colliding with the box. Here, it is the case that robot needs to ask a human operator for collaboration. Accordingly, the reasoner appends the predicates *infeasByRob(tiago, box1, posBox1)* and *assist(person, box1, posBox1)* to the corresponding state. The *humanOpen* action then appears.

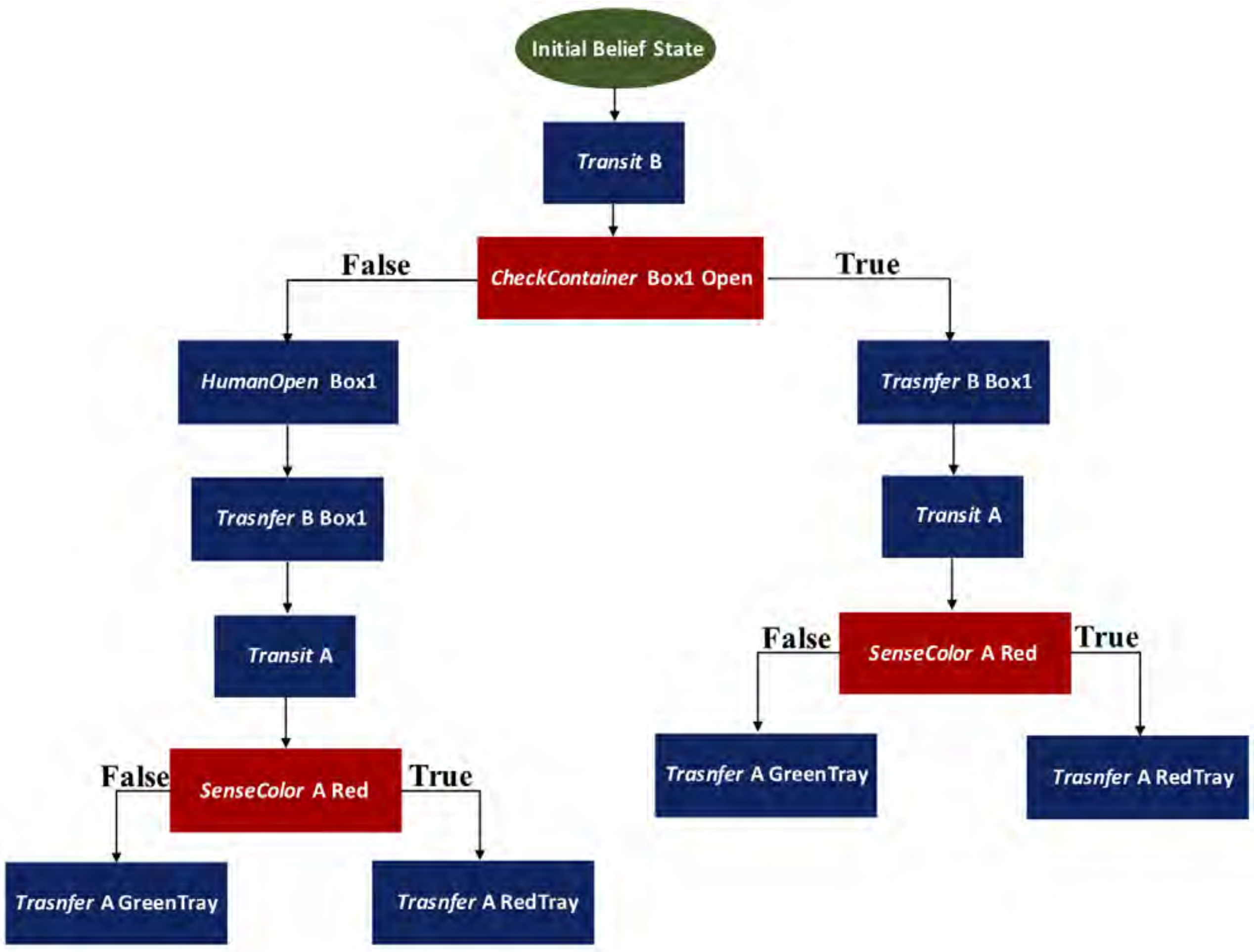


Figure 7.3: The conditional plan results from the proposed planning process. The actions highlighted with the blue color are the ones assigned to the robot or a human operator. Some important actions parameters are represented. The actions specified by the red color are sensing actions.

The executive simulated result of the proposed manipulation problem is shown in Fig. 7.4. We assume that the values of the uncertainty information are provided in run-time in simulation. So, the executive plan is provided below:

Executive Plan: { *Transit-B, CheckContainer-Box1-Open (False), HumanOpen-Box1, Transfer-B-Box1, Transit-A, SenseColor-A-Red (True), Transfer-A-RedTray* }

The states represented in the figure are classified as follows:

- *Part(a)* is the initial belief state of the robot and environment.

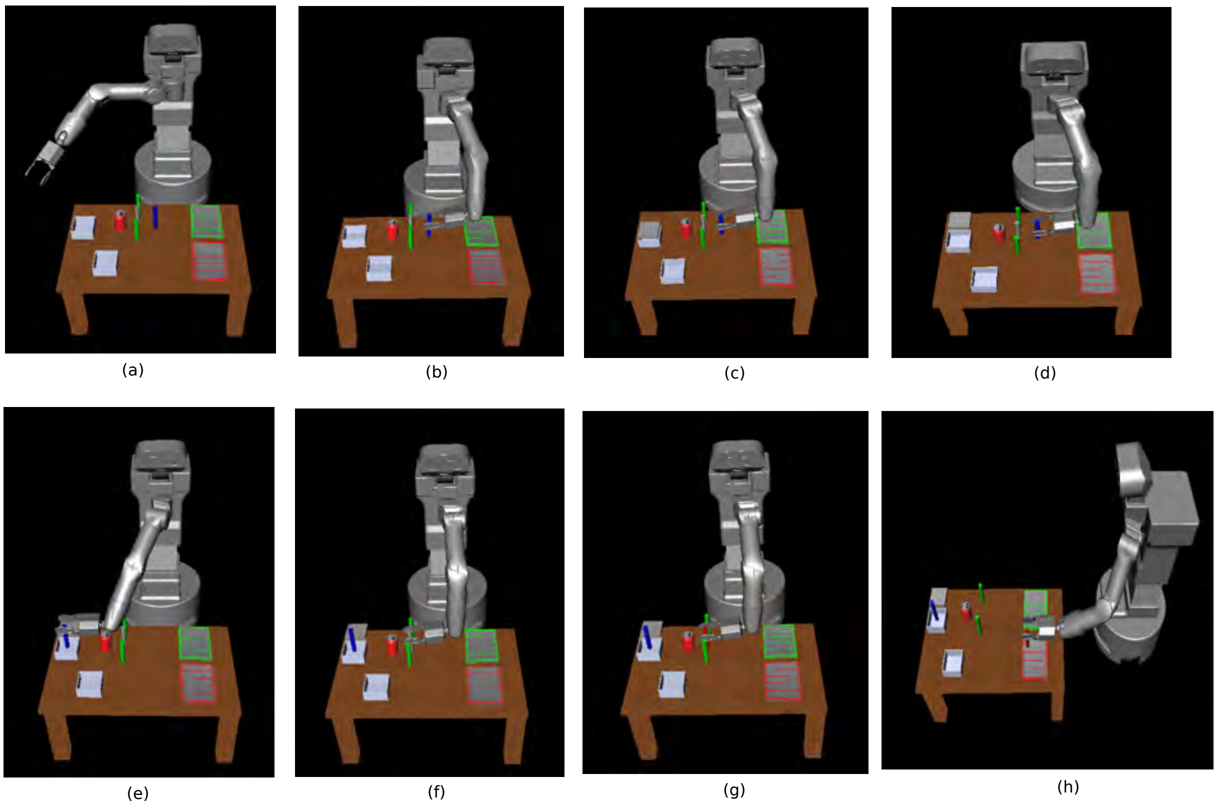


Figure 7.4: The simulation results of the executive plan performed by the *TIAGo* robot: a) is the initial robot and environment state, b) is the state after *Transit* action towards cylinder B, c) is the result of the sensing action *CheckContainer-Box1-Open*, d) is the state after applying *HumanOpen* action, e) is the state after the robot executes the *Transfer* action for cylinder B, f) is the state when the robot transits to cylinder A, g) is the state resulting from the sensing action *SenseColor-A-Red*, and h) is the state when the robot place cylinder A on the associated tray.

- *Part(b)* is the state where the robot applied *transit* action in order to reach cylinder B.
- *Part(c)* is the state where the result of the sensing action *CheckContainer-Box1-Open* is clear.
- *Part(d)* is the state where the *HumanOpen* action is executed.
- *Part(e)* is the state where the robot places cylinder B within box 1.
- *Part(f)* is the state where the robot transits to cylinder A.
- *Part(g)* is the state where the result of the sensing action *SenseColor-A-Red* is clear.
- *Part(h)* is the state where the robot moves its base and the arm configuration in order to place cylinder A over the red tray.

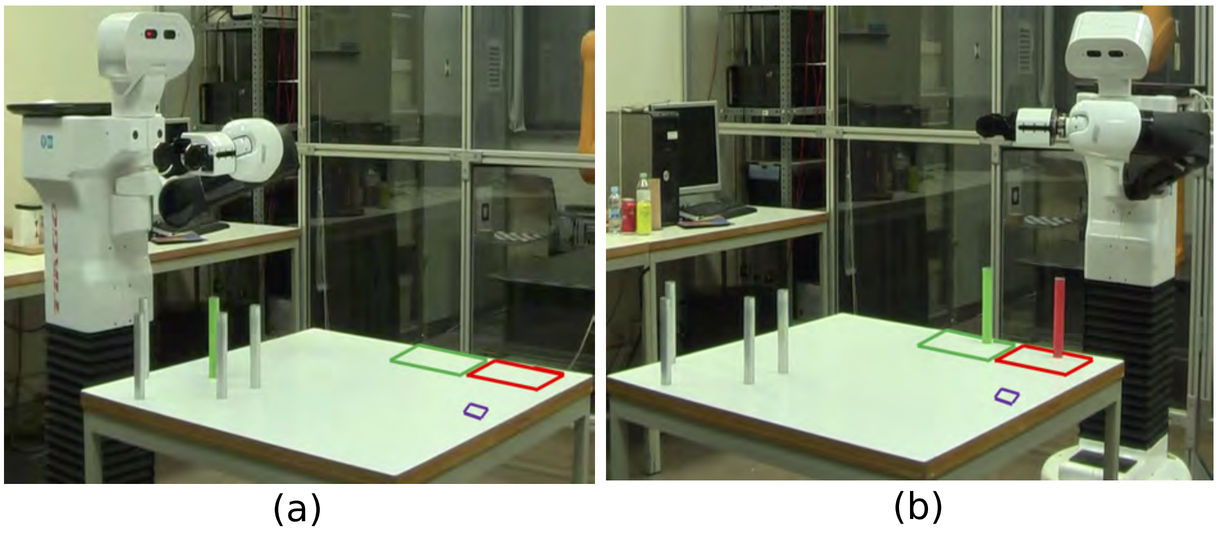


Figure 7.5: The manipulation example where green and red objects have to be placed in the green and red regions. The red object is not initially located in the robot workspace. The pink region is the place where human can transfer objects to the robot workspace. The solution can be visualized here: `https://sir.upc.es/projects/ontologies/GreenRedHuman.mp4`. The solution for the case that the red object is initially located on the table is visualized here: `https://sir.upc.es/projects/ontologies/TiagoRedGreenRob.mp4`.

In addition, the proposal has been evaluated for other cluttered problems where the robot needs to sort objects according to the colors. The problem represented in Fig. 7.5, called *Problem-2*, shows the initial and goal states of manipulation where the green and red objects must be located on the green and red regions respectively. The red object is not initially located on the table. The pink region is considered on the robot workspace where an operator can transfer objects. The planning uncertainties are the color of the green object which could be actually green or red and the location of the red object which could be on the robot table or in the human workspace. The proposal has been tested for similar problems by increasing the number of objects and varying color and/or location uncertainties. The problems performance are represented in Table 7.1 in terms of conditional and executive plan length, and moreover planning time. *Problems-3* includes a cluttered problem where there are nine objects and three of them need to be sorted. *Problem-4* is the one where twelve objects exist and four of them have to be sorted.

The proposed framework implementation consists of four components: task planning, relaxed geometric reasoning, motion planning, and executive module. Task planning is implemented using a modified version of the *Contingent-FF* planner developed in C++. The 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

Problem	Conditional plan		Executive plan		Planning time
	Sensing	Executive	Sensing	Executive	
Problem-1	3	10	2	5	35
Problem-2	3	13	2	5	59
Problem-3	3	19	2	8	163
Problem-4	7	41	3	12	449

Table 7.1: The conditional plan and executive plan length in terms of number of sensing and executive actions and planning time in seconds for the evaluated problems.

conditions (such as object placements or robot grasp configurations) are not pre-computed. They are sampled and assigned on demand during the planning process.

Relaxed geometric reasoning and motion planning use *The Kautham Project* (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* (Kuffner and LaValle, 2000) 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 work-space. Relaxed geometric reasoning uses these modules in order to find feasible sample geometric instances for symbolic actions. The executive module uses a sensing module which uses the 3D camera mounted inside the *TIAGo* robot, and also some components provided by *PAL Robotics* to send a motion path to the robot. The communication between task, relaxed geometric reasoning, motion planning, and executive modules is done via *Robotic Operating System (ROS)* (Quigley et al., 2009).

7.8 Conclusions

This chapter has presented a contingent-based task and motion planning approach that is able to cope with high-dimension mobile manipulation problems in the presence of high-level uncertainty and human-interactions. Human interactions are referred to as sharing knowledge and collaboration with the robot. A set of geometric reasoning processes have been offered to capture task constraints imposed in the robot environment while planning is done. So, the proposed approach results a tree-shaped conditional plan which is geometrically feasible.

Chapter 8

Implementation Framework for Combining Task and Motion Planning

8.1 Introduction

This chapter presents an implementation framework which makes the possibility of combining task and motion planning using different information techniques. The main purpose is to provide detailed information concerning implementation issues, e.g., how the information can be exchanged between task and motion planning levels although they have been implemented with different languages. Some parts of the implementation framework have been presented in (Akbari et al., 2016b) and (Rosell et al., 2014).

8.2 Implementation Framework

The proposed framework of combining task and motion planning levels is represented in Fig. 8.1. It is composed of 4 main layers: *Semantic Manipulation Knowledge*, *Task Planning*, *Motion Planning*, and *Task Executive*. This section is going to summerize the main role of each layer and the the way of communication among them.

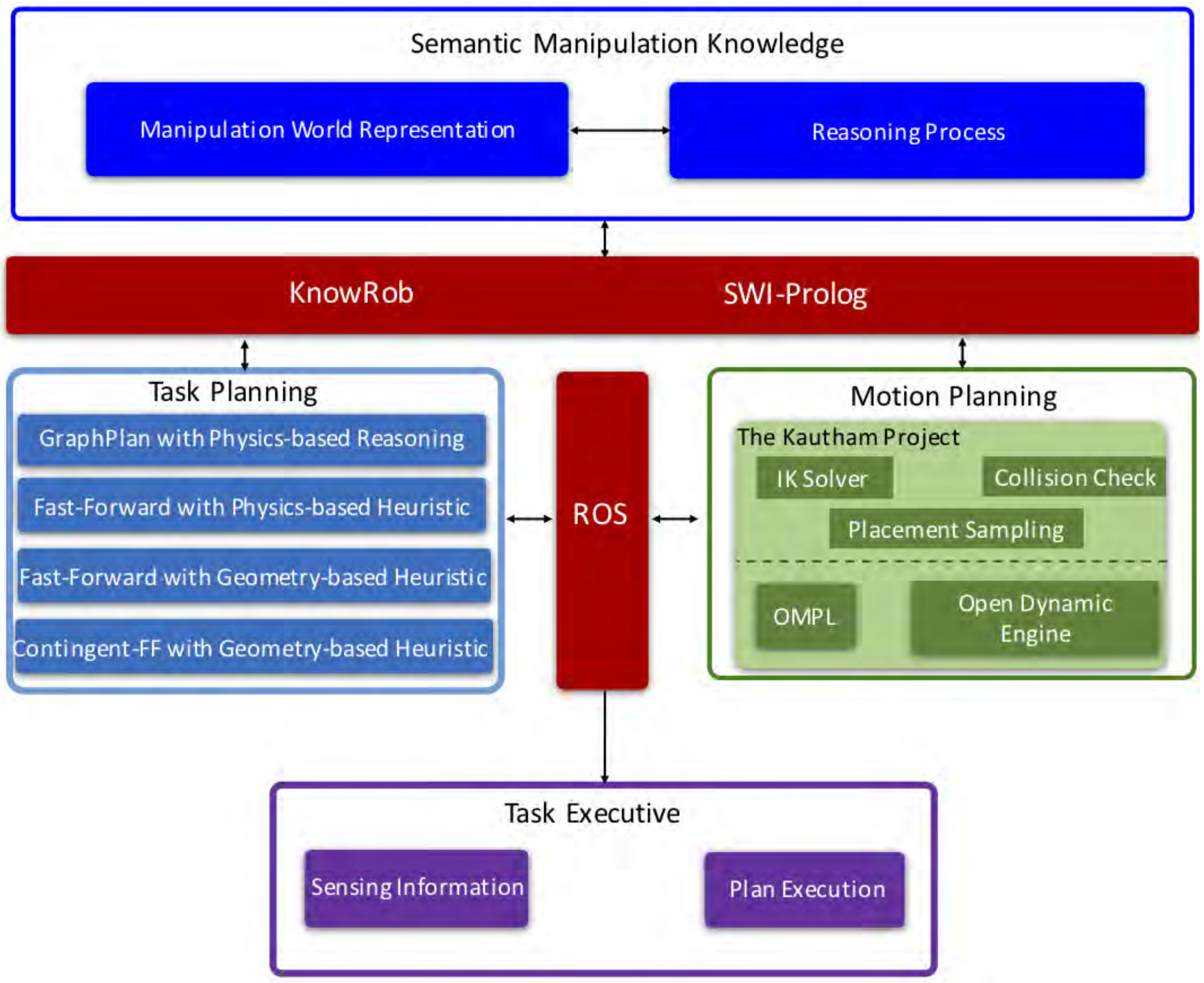
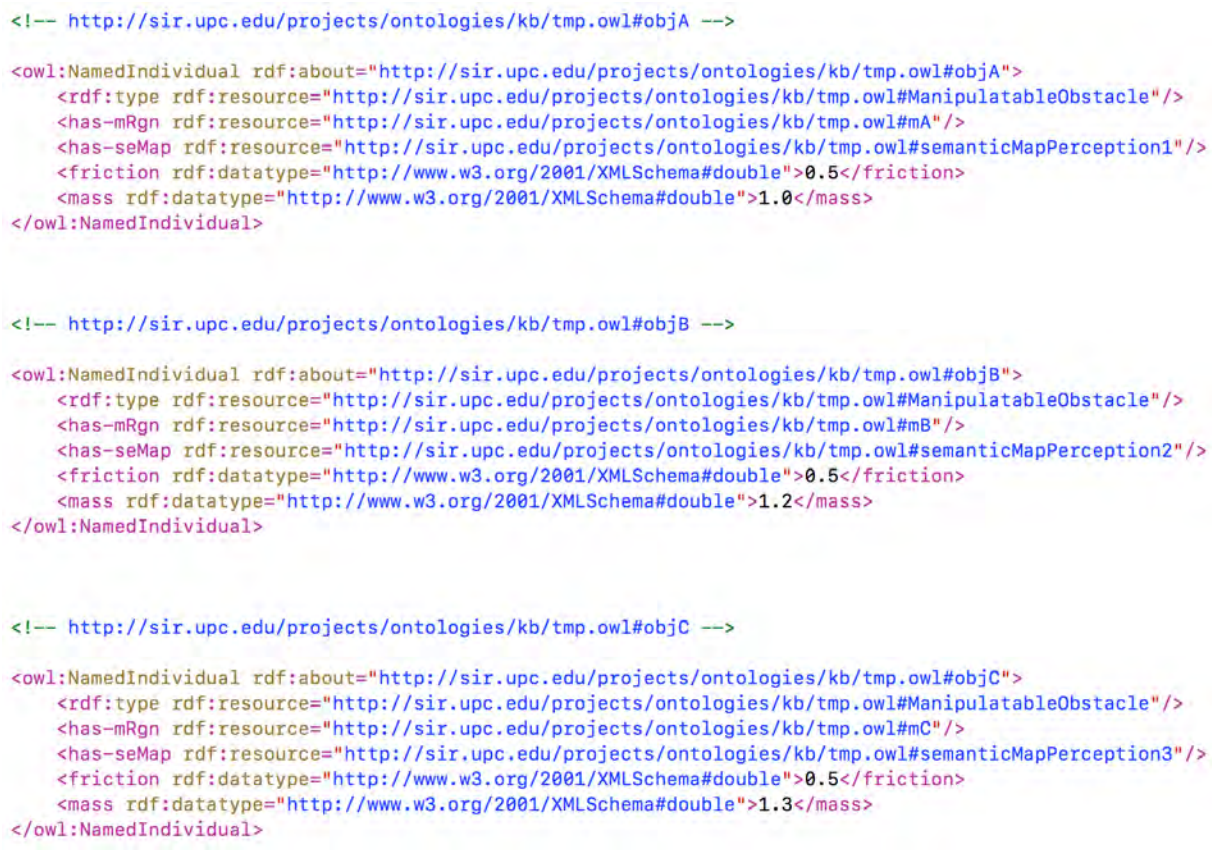

Semantic Manipulation Knowledge
Manipulation World Representation
Reasoning Process
KnowRob
SWI-Prolog
Task Planning
GraphPlan with Physics-based Reasoning
Fast-Forward with Physics-based Heuristic
Fast-Forward with Geometry-based Heuristic
Contingent-FF with Geometry-based Heuristic
ROS
Motion Planning
The Kautham Project
IK Solver
Collision Check
Placement Sampling
OMPL
Open Dynamic Engine
Task Executive
Sensing Information
Plan Execution

Figure 8.1: The framework of combining task and motion planning.

8.3 Semantic Manipulation Layer

Basically, the semantic manipulation layer includes the ontologies files related to the knowledge modelling stored in the cloud and reasoning process over the knowledge. The taxonomy of the knowledge has been demonstrated in Chapter 3. The knowledge is modelled using the *Protégé* editor[1]which is an open-source platform in terms of *OWL*. Some details of the *OWL* information is represented in Fig. 8.2. This example explains the information stored for three objects, namely *objA*, *objB*, and *objC* as the *OWL* individuals under the tag *owl:NamedIndividual*. Each object has also a number of tags. *has-mRgn* and *has-seMap*, which are object properties in the ontology, correspond with the associated manipulatable region and semantic map (storing the object pose) considered for an object respectively. Here, manipulatable regions are the individuals stored

[1]http://protege.stanford.edu/

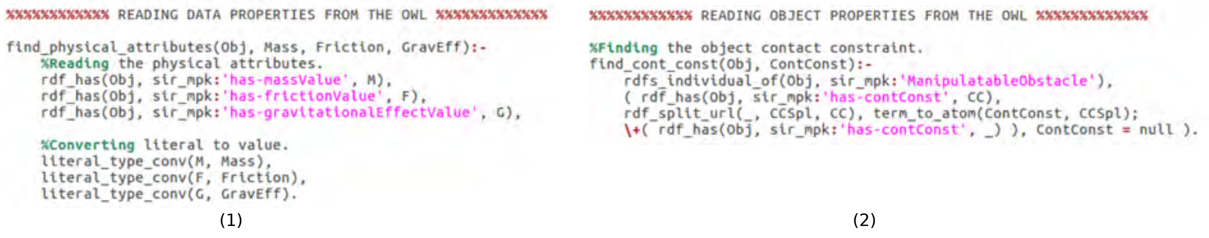


```
<!-- http://sir.upc.edu/projects/ontologies/kb/tmp.owl#objA -->

<owl:NamedIndividual rdf:about="http://sir.upc.edu/projects/ontologies/kb/tmp.owl#objA">
    <rdf:type rdf:resource="http://sir.upc.edu/projects/ontologies/kb/tmp.owl#ManipulatableObstacle"/>
    <has-mRgn rdf:resource="http://sir.upc.edu/projects/ontologies/kb/tmp.owl#mA"/>
    <has-seMap rdf:resource="http://sir.upc.edu/projects/ontologies/kb/tmp.owl#semanticMapPerception1"/>
    <friction rdf:datatype="http://www.w3.org/2001/XMLSchema#double">0.5</friction>
    <mass rdf:datatype="http://www.w3.org/2001/XMLSchema#double">1.0</mass>
</owl:NamedIndividual>


<!-- http://sir.upc.edu/projects/ontologies/kb/tmp.owl#objB -->

<owl:NamedIndividual rdf:about="http://sir.upc.edu/projects/ontologies/kb/tmp.owl#objB">
    <rdf:type rdf:resource="http://sir.upc.edu/projects/ontologies/kb/tmp.owl#ManipulatableObstacle"/>
    <has-mRgn rdf:resource="http://sir.upc.edu/projects/ontologies/kb/tmp.owl#mB"/>
    <has-seMap rdf:resource="http://sir.upc.edu/projects/ontologies/kb/tmp.owl#semanticMapPerception2"/>
    <friction rdf:datatype="http://www.w3.org/2001/XMLSchema#double">0.5</friction>
    <mass rdf:datatype="http://www.w3.org/2001/XMLSchema#double">1.2</mass>
</owl:NamedIndividual>


<!-- http://sir.upc.edu/projects/ontologies/kb/tmp.owl#objC -->

<owl:NamedIndividual rdf:about="http://sir.upc.edu/projects/ontologies/kb/tmp.owl#objC">
    <rdf:type rdf:resource="http://sir.upc.edu/projects/ontologies/kb/tmp.owl#ManipulatableObstacle"/>
    <has-mRgn rdf:resource="http://sir.upc.edu/projects/ontologies/kb/tmp.owl#mC"/>
    <has-seMap rdf:resource="http://sir.upc.edu/projects/ontologies/kb/tmp.owl#semanticMapPerception3"/>
    <friction rdf:datatype="http://www.w3.org/2001/XMLSchema#double">0.5</friction>
    <mass rdf:datatype="http://www.w3.org/2001/XMLSchema#double">1.3</mass>
</owl:NamedIndividual>
```

Figure 8.2: Some parts of the ontological knowledge representation.

under the class *ManipulatableRegion* and semantic maps are the individuals of class *ObstaclePose* as depicted in Fig. 3.1. The tags *friction* and *mass* display the values considered for the friction and weight of the corresponding object.

For the reasoning process over the knowledge, the Prolog language is used. It is a logic programming language that is particularly suited to programs which contain symbolic computation. For this reason, it is a frequently used language in *AI* for knowledge-based reasoning where manipulation of symbols and inference about them is a common task. Prolog consists of a series of rules and facts. A program is run by presenting some queries and seeing if this can be proved against these known rules and facts. To access the *OWL* knowledge from Prolog, the *KnowRob* tool (Tenorth and Beetz, 2009) and the *SWI-Prolog* library (Wielemaker et al., 2012) which provide flexible access to the knowledge are employed. The reasoning process proposed in Chapter 3.4 was implemented using these libraries and the Prolog language. Some parts of the process related to the way of accessing the ontological knowledge is shown in Fig. 8.3 in which physical properties of objects are retrieved using *find_physical_attributes* and contact constraints of objects are also found by *find_cont_const* predicates. The reasoning process about sorting of the objects type along its manipulation constraints (e.g. an object can be manipu-



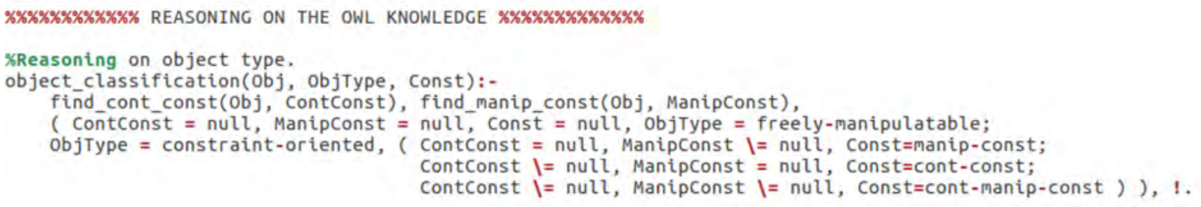
```
%%%%%%%%%%%% READING DATA PROPERTIES FROM THE OWL %%%%%%%%%%%%%

find_physical_attributes(Obj, Mass, Friction, GravEff):-
    %Reading the physical attributes.
    rdf_has(Obj, sir_mpk:'has-massValue', M),
    rdf_has(Obj, sir_mpk:'has-frictionValue', F),
    rdf_has(Obj, sir_mpk:'has-gravitationalEffectValue', G),

    %Converting literal to value.
    literal_type_conv(M, Mass),
    literal_type_conv(F, Friction),
    literal_type_conv(G, GravEff).
```

(1)

```
%%%%%%%%%%%% READING OBJECT PROPERTIES FROM THE OWL %%%%%%%%%%%%%

%Finding the object contact constraint.
find_cont_const(Obj, ContConst):-
    rdfs_individual_of(Obj, sir_mpk:'ManipulatableObstacle'),
    ( rdf_has(Obj, sir_mpk:'has-contConst', CC),
    rdf_split_url(_, CCSpl, CC), term_to_atom(ContConst, CCSpl);
    \+( rdf_has(Obj, sir_mpk:'has-contConst', _) ), ContConst = null ).
```

(2)

Figure 8.3: The Prolog predicates describing the accessibility of the knowledge: 1) Retrieves the physical properties of objects. 2) Finds contacts constraints of manipulatable objects.



```
%%%%%%%%%%%% REASONING ON THE OWL KNOWLEDGE %%%%%%%%%%%%%

%Reasoning on object type.
object_classification(Obj, ObjType, Const):-
    find_cont_const(Obj, ContConst), find_manip_const(Obj, ManipConst),
    ( ContConst = null, ManipConst = null, Const = null, ObjType = freely-manipulatable;
    ObjType = constraint-oriented, ( ContConst = null, ManipConst \= null, Const=manip-const;
                                     ContConst \= null, ManipConst = null, Const=cont-const;
                                     ContConst \= null, ManipConst \= null, Const=cont-manip-const ) ), !.
```

Figure 8.4: The Prolog predicate to reason over objects type along its constraints.

lated in x or y direction) is explained in Fig. 8.4. In this case, the predicates *find_cont_const* and *find_manip_const* first compute the contacts constrains and manipulation constraints respectively, if any. In the case of having constraints, the type of constraint is returned; otherwise, the object is marked as free manipulatable object.

8.4 Task Planning Layer

The task planning algorithms discussed throughout the present thesis, some using the ontological knowledge, have been developed either in C++ or the Prolog language.

GraphPlan with Physics-based Reasoning (presented in Chapter 4) is a modified version of the *planning graph* algorithm planner. It uses the ontological knowledge and is implemented in Prolog. The planner uses the information about the action description, some query conditions, and also workspace from the knowledge that can be used while planning is processed. *Fast-Forward with Physics-based Heuristic* (presented in Chapter 5) is a version of the basic *FF* planning system using the knowledge and developed in Prolog as well. In this case, The main reason of using Prolog language is to facilitate the accessibility of knowledge. The main advantage of the Prolog language implementation is to perform appropriately symbolic-based reasoning (like reasoning over the knowledge).

Fast-Forward with Geometry-based Heuristic (presented in Chapter 6) is a modified version of the original *FF* planner using *PDDL* to describe symbolic action, and also the initial state and goal conditions. The original software is based on C++ and presented in `https://fai.cs.uni-saarland.de/hoffmann/ff.html`. It has been extended to incorporate geometric reasoning process in the heuristic computation and the state space search. *Contingent-FF with Geometry-based Heuristic* (presented in Chapter 7) is a modified version of the *Contingent-FF* planner using *PDDL* as well. The original software is available in `https://fai.cs.uni-saarland.de/hoffmann/cff.html` developed in C++. The geometric reasoning and sensing actions have been incorporated in the modified version. The main advantage of C++ implementation is to provide fast and efficient computation.

8.5 Motion Planning Layer

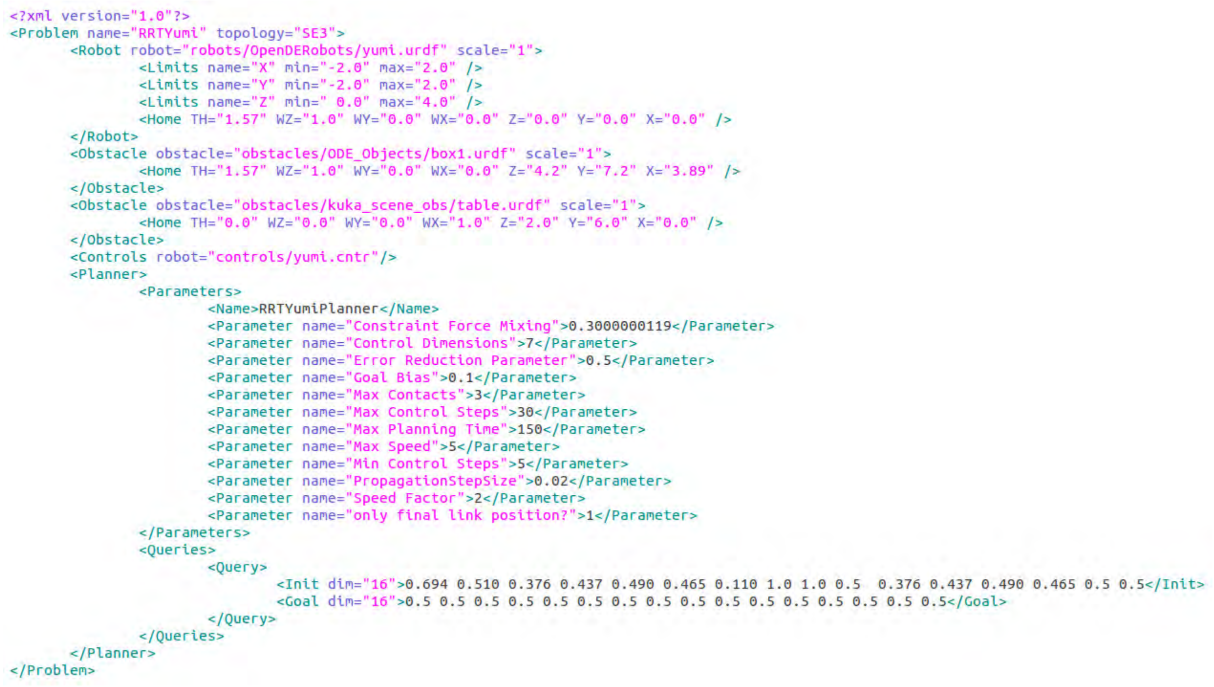
Basically, to perform different type of low-level geometric reasoning and motion planning, *The Kautham Project* which is a C++-based open source software is used. It is used for teaching and research purposes at the Institute of Industrial and Control Engineering (*IOC-UPC*).

8.5.1 Geometric Reasoning with The Kautham Project

The Kautham Project is mainly applied to develop and demonstrate the motion planning algorithms, which could be based on geometry, control, and physics-based families of motion planning. Motion planning problems are described using input XML four basic main sections (see Fig. 8.5). They are robot(s), obstacle(s), controls, and planner. The main parameters provided for each robot/obstacle are the path to a robot/obstacle description file, the translational limits of motion if the robot/obstacle has a mobile base, the home location with respect to the world reference frame. The information for controls which are numbers between zero and one is optional. In the end, the principle planning parameters provided are the query to be solved containing initial and goal configurations and the parameters of the planner (like goal bias and planning time).

A robot is defined as a kinematic tree with optional mobile base, its configuration space is $R = SE(3) \times \mathbb{R}^n$, where n represents the number of joints of the robot. If the base is fixed, $SE(3)$ part is represented as null. The kinematic structure of the robot is defined using *Unified Robotics Description Format* (*URDF*, `http://wiki.ros.org/urdf`). It contains the visual robot model, the collision model, transformations between the links, joints (along with limits) and dynamic parameters such as damping and masses. The visualization model is defined with triangular meshes that can be represented in *.wrl, .stl*, and *.dae* formats. The collision model can be represented either by a triangular mesh or by primitive shapes (cylinder, box, and sphere). Obstacles are also defined as robot data structures, in case of fixed obstacles, none of its degrees of freedom are actuated.

The main core of *The Kautham Project* consists of the workspace, the configuration space, and a set of planners. Once a problem is loaded, it fills the data structures of the workspace (that in-



```xml
<?xml version="1.0"?>
<Problem name="RRTYumi" topology="SE3">
        <Robot robot="robots/OpenDERobots/yumi.urdf" scale="1">
                <Limits name="X" min="-2.0" max="2.0" />
                <Limits name="Y" min="-2.0" max="2.0" />
                <Limits name="Z" min=" 0.0" max="4.0" />
                <Home TH="1.57" WZ="1.0" WY="0.0" WX="0.0" Z="0.0" Y="0.0" X="0.0" />
        </Robot>
        <Obstacle obstacle="obstacles/ODE_Objects/box1.urdf" scale="1">
                <Home TH="1.57" WZ="1.0" WY="0.0" WX="0.0" Z="4.2" Y="7.2" X="3.89" />
        </Obstacle>
        <Obstacle obstacle="obstacles/kuka_scene_obs/table.urdf" scale="1">
                <Home TH="0.0" WZ="0.0" WY="0.0" WX="1.0" Z="2.0" Y="6.0" X="0.0" />
        </Obstacle>
        <Controls robot="controls/yumi.cntr"/>
        <Planner>
                <Parameters>
                        <Name>RRTYumiPlanner</Name>
                        <Parameter name="Constraint Force Mixing">0.3000000119</Parameter>
                        <Parameter name="Control Dimensions">7</Parameter>
                        <Parameter name="Error Reduction Parameter">0.5</Parameter>
                        <Parameter name="Goal Bias">0.1</Parameter>
                        <Parameter name="Max Contacts">3</Parameter>
                        <Parameter name="Max Control Steps">30</Parameter>
                        <Parameter name="Max Planning Time">150</Parameter>
                        <Parameter name="Max Speed">5</Parameter>
                        <Parameter name="Min Control Steps">5</Parameter>
                        <Parameter name="PropagationStepSize">0.02</Parameter>
                        <Parameter name="Speed Factor">2</Parameter>
                        <Parameter name="only final link position?">1</Parameter>
                </Parameters>
                <Queries>
                        <Query>
                                <Init dim="16">0.694 0.510 0.376 0.437 0.490 0.465 0.110 1.0 1.0 0.5  0.376 0.437 0.490 0.465 0.5 0.5</Init>
                                <Goal dim="16">0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5</Goal>
                        </Query>
                </Queries>
        </Planner>
</Problem>
```

Figure 8.5: XML problem description file that includes information of the robot, the obstacle, the controls, and the planner.

cludes the robot/obstacle models, their kinematic limits), and of the configuration space. Moreover, it contains the methods for collision checking using *PQP* (Gottschalk et al., 1999) or *FCL* (Pan et al., 2012), and forward kinematics to move the robot to the particular configuration. To sample the configuration space various state samplers (such as random, Gaussian and Halton) are included. The project, moreover, contains different inverse kinematics algorithms for different robots and the sampling-based planners (such as *RRT, PRM*, and *EST*) offered by *Open Motion Planning Library* (*OMPL* (Sucan et al., 2012)). The detailed explanation regarding the implementation of planners can be found here `https://sir.upc.edu/projects/kautham/`.

8.5.2 The Kautham Project Extention with Knowledge Processing

The Kautham Project has been extended to fetch the ontological knowledge and reason over that. Once the problem is loaded, the knowledge processing is carried out through the Prolog engine. A data structure is considered in the low-level in which the result of knowledge processing is classified and can be retrieved during the motion planning process. The essential stored knowledge is summarized in *Robot Capability Properties*, *Object Type*, *Objects Features*, and *Object Physical Properties*.

Robot Capability Properties includes knowledge like robots constraints or capabilities (e.g.

maximum weight that can be loaded by the robot's gripper or hands). *Object Type* embraces the type of objects (like fixed or manipulatable). *Objects Features* involves the the features of objects like color or manipulation constraints (using the predicate *find_cont_const* as shown in Fig. 8.3). *Object Physical Properties* stores the physical properties of objects like mass or friction as shown in Fig. 8.3 using the predicate *find_physical_attributes*.

According to the knowledge-based motion planning scope, we have recently proposed some physics-based motion planners (such as approaches presented in (Muhayyuddin et al., 2018) and (Muhayyudin et al., 2015)) in solving manipulation problems at which a robot needs to interact with some manipulatable obstacles in order to find a feasible plan.

8.6 Task Executive Layer

When task and motion planning is successfully terminated, the complete manipulation plan, stored in the task planning layer, needs to be executed in the real environment or simulation. The task executive layer is used to deal with executing the plan by a robot, and moreover the sensing information required to monitor the result of the sensing actions (used in the case of uncertainty) if any. Regarding perception, the *Kinect* camera is employed using the *ROS* driver (`http://wiki.ros.org/kinect`). Objects are tagged with the *AR* tags and the *ROS* library (`http://wiki.ros.org/ar_track_alvar`) is used in order to identify their poses. In terms of simulation, *DrawStuff* (a part of *Open Dynamic Engine* `http://www.ode.org/`), The *Gazebo* simulator (`http://gazebosim.org/`), *Rviz* (`http://wiki.ros.org/rviz`), as well as *The Kautham Viewer* are used. *The Kautham Viewer* is originally implemented in the *The Kautham Project* and provides the visualization of the robot model, the collision model and the visualization of the configuration pace. Its limitation is that it can visualize only the solution path of a single query, with no consideration of object manipulation. To improve this, it has been extended to visualize the full manipulation plan with different set of actions (e.g. *Pick, Place*, or *Push*). The snapshots of the plan execution for mobile manipulation and the bi-manual robot have been accomplished using *DrawStuff* and *The Kautham Viewer* respectively in simulation, and also the mobile manipulator example has been presented using *The Kautham Viewer*.

8.7 Communication Layer

As different programming languages and libraries are associated with planners and modules, on of the challenging issues is the question of how task and motion planning levels are negotiating with each other and other modules.

8.7.1 Communications with The Motion Planning Layer

The main functionalities of the motion planning level such as motion planning query or collision check between obstacles have been encapsulated as services inside a node and provided in the *ROS* shell. In the case of calling motion planning from the task planning level, some of the information of robot (e.g. robot configurations or pose), workspace (e.g. objects home locations), query to be solved are filled through the task planning process using the *Robot Operating System* (*ROS*, `https://ros.org`) communication.

Therefore, the task planners need to define client nodes and call the services for different geometric reasoning processes.

To grant the access of motion planning to the knowledge and the reasoning process, the communication layer includes the *SWI Prolog* library. This library provides the access to the Prolog queries to the knowledge and the queries defined in the reasoning process module. It provides an interface to the Prolog language by opening the Prolog engine through C++, and then the Prolog files can be loaded in order to enable the reasoning process.

8.7.2 Communications with the Task Planning Layer

The Prolog-based task planning approaches are considered to apply knowledge processing within the planning process. They have been developed in Prolog due to the fact that the knowledge-based reasoning process is implemented in the Prolog language; therefore, it facilitates to access the knowledge and the reasoning process. Through *KnowRob* and the *SWI-Prolog* library, the task planners are able to ask queries to the knowledge according to planning domains and manipulation world where a robot is acting.

The role of *KnowRob* is to provide advanced predicates to access the ontological knowledge, and also to call some reasoning modules proposed by them, e.g., spatial reasoning and semantic map (storing object poses) through the task planning process.

Two types of communications, depending on the language of the task planner algorithms, are proposed to exchange information between task and motion levels. One is based on the Prolog and the other one is based on C++ implementation. According to the Prolog implementation, the task planners can be encapsulated as clients or services using *SWI-Prolog* and can be called from an application manager. It communicates with motion planning through the *ROS* middleware. The architecture of this process is shown in Fig. 8.6.

In the point of view of implementation, an interface between task and motion planning, which is called *Motion Planner Client*, is required to exchange the information. This client is developed in C++ and uses the *SWI-Prolog* library providing access to the C++ from the Prolog environment. Some of the predicate macro considered in the *Motion Planner Client* file is presented in Fig. 8.7. Each predicate has two arguments. i.e., the first one shows the name of a Pro-

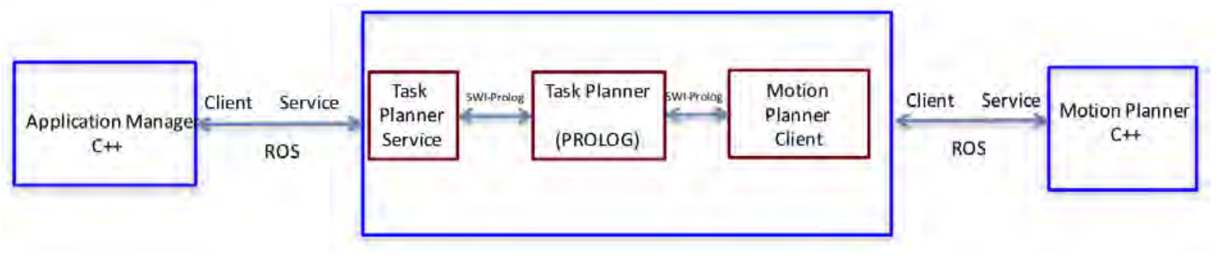
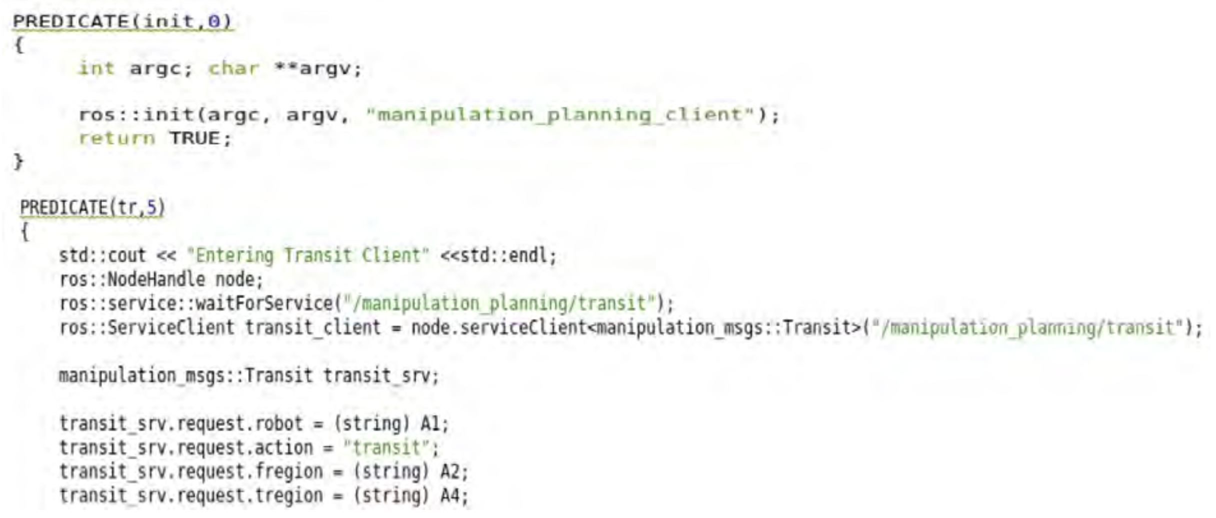


Figure 8.6: Prolog-based task planning as either service or client to communicate with motion planning.



```
PREDICATE(init,0)
{
      int argc; char **argv;

      ros::init(argc, argv, "manipulation_planning_client");
      return TRUE;
}
 PREDICATE(tr,5)
 {
     std::cout << "Entering Transit Client" <<std::endl;
     ros::NodeHandle node;
     ros::service::waitForService("/manipulation_planning/transit");
     ros::ServiceClient transit_client = node.serviceClient<manipulation_msgs::Transit>("/manipulation_planning/transit");

     manipulation_msgs::Transit transit_srv;

     transit_srv.request.robot = (string) A1;
     transit_srv.request.action = "transit";
     transit_srv.request.fregion = (string) A2;
     transit_srv.request.tregion = (string) A4;
```

Figure 8.7: Some predicate macros defined in *Motion Planner Client*.

log predicate and the second one is the number of arguments for that predicate. For instance, in Fig. 8.7, the first predicate macro shows the way of initializing the *manipulation_planning_client* node from the *init* predicate and the second one represents the *tr* predicate which is used to evaluate *Transit* actions from Prolog.

The predicate macros, developed in *Motion Planner Client*, can be called in the task planning procedure. The detailed information of the task and motion planning communication is exhibited in Fig. 8.8. It is shown that *Motion Planner Client* is loaded as a foreign library inside task planning using the predicate *load_foreign_library*. In this way, the Prolog-based task planners are able to call motion planning services and wait for their responses.

The communication of the C++-based task planners is straightforward. A client node which is able to call motion planning services is considered for sending and receiving information. In this case, as all the involved modules are developed in C++, there is no any issue of transferring information between the modules. After the task planning process is successfully terminated,

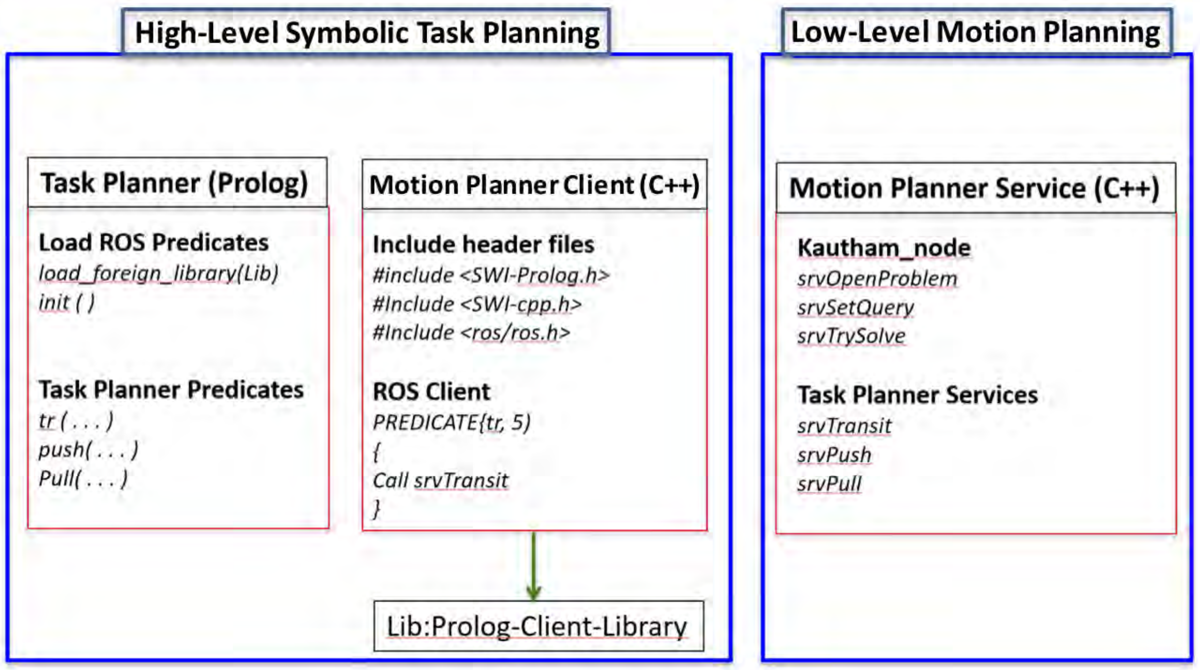


Figure 8.8: Detailed information about the communication between task and motion planning.

the manipulation plan will be forwarded to the task executive layer. This communication is fulfilled through the *ROS* communication.

8.8 Conclusions

This chapter has presented some details on the implementation framework proposed to solve the combining of task and motion planning. It discussed different task planners tools used throughout the thesis, *The Kautham Project* which is a tool to perform motion planning, and the communication layer among the proposed system that mainly shows how information could be exchanged from different languages.

Chapter 9

Conclusions and Future Work

9.1 Conclusions

The present thesis has developed a number of combined task and motion planning algorithms with respect to the increasing challenges of robotic manipulation problems. Different sort of reasoning processes have been also proposed inside the framework of combining task and motion planning levels in order to come up with a feasible manipulation plan. To sum up, the challenges of robotic manipulation problems considered in the thesis are summarized as follows:

- **Mobile manipulation problems**: They refer to the problems where there is no direct solution between the initial and the goal configurations of mobile robot, so they need to push some objects away in order to reach its goal. In this line, the main challenging task correspond with finding a geometrically feasible manipulation plan with a low-cost.
- **Collaborative multi mobile manipulation problems**: They hold the challenge of *mobile manipulation problems* although several mobile robots can share a task and collaborate with each other in order to solve the problem. Therefore, the coordination of sub-tasks remain challenging.
- **Higher dimensional manipulation problems**: Such problems are referred to as manipulation where bi-manual robots or mobile manipulators are going to set a table with various objects or grasping an object in a cluttered environment. These problems are also very challenging due to the fact that they are subject to strong geometric constraints, e.g., occlusions, lack of space in placing objects, objects reachability, or kinematic constraints of manipulators.

To cope with mobile manipulation problems, physics-based motion planning has been employed that mainly considers interactions between mobile robots and objects. This provides how much push or pull action is required to satisfy the actions conditions if possible and is able to measure transit actions of the robots in terms of power. Such information can guide task planning process towards selecting the actions having low-cost.

To enrich the combination of task and motion planning frameworks, manipulation knowledge can be developed and used within the planning process. This aims to facilitate the way of interaction with objects for the case of action execution, and also reason over some symbolic actions conditions (like robot capabilities issues) in order to guide the action selection process.

Particularly, this thesis concluded the following points at each section:

- The semantic manipulation knowledge has been presented in terms of representation and reasoning process. The knowledge is modelled using ontologies. The knowledge is basically divided in two categories; manipulation world and manipulation planning. The prior one associates with the information about the limits of the robots and objects features and properties. The latter one describes the conditions of planning components. Reasoning over this knowledge aids to evaluate actions and the way of executing each of them.

- To deal with *mobile manipulation problems*, combined task and physics-based motion planning using a planning graph technique has bee proposed. This combination provide some sort of high along low level reasoning to find a low-cost and geometrically feasible manipulation plan.

- In the case of *collaborative multi mobile manipulation problems*, a heuristic-based task and physics-based motion planning has been presented. The offered reasoning processes are able to assign sub-tasks to various mobile robots, and also find a low-cost and feasible manipulation plan.

- A heuristic-based task and motion planning has been proposed to tackle *higher dimensional manipulation problems*. The approach is able to capture most of geometric constraints in heuristic and assigns geometric choices carefully to symbolic actions. This process reduces geometric backtracking and a number of call to a motion planner.

- Furthermore, in *higher dimensional manipulation problems*, there could be the case that there is imperfect information about the initial state of the robot world and a robot may be required to interact with human to perform some tasks. Accordingly, a contingent-based task and motion planning has been presented to cope with these issues.

- Finally, an implementation framework has been illustrated to show the main tools and the flow of information among modules used to combine task and motion planning levels. Concerning the robotic systems, we have tested our results with the robots *Yumi ABB* and *TIAGo PAL*.

9.2 Future Work

Along with the conclusion points stated above, the current thesis, moreover, opens new research problems that require further consideration in order to achieve the latter objective of combined task and motion planning that is to deal with geometric uncertainty, the planning system using learning techniques, and also human-like executing actions of manipulation plans.

This thesis has throughly investigated different combinations of task and motion planning using only automated planning. Following the direction of task and motion planning under uncertainty, geometric uncertainty can be also considered inside the combined task and motion planning framework in order to deal with different execution failures that could be either due to robot or objects poses uncertainty or unpredicted situations like slipping grasped object from the robot gripper.

However, a robot can learn and experience about problems and retrieves the useful information for similar situations or problems encountered before. In this case, robot is able to learn from task failure and can handle similar problems which will face in its future tasks. So, one of the future research directions could be to develop a mechanism to use learning and experience features in order to facilitate planning process.

The other future research direction could be to use deep learning techniques in assigning geometric information to symbolic actions. In this case, while selecting, e.g., grasping configurations of object placement poses, robot can select more human-like geometric information for the task execution. Learning techniques can be also used to detect objects along their physical properties in unknown environments, and then assert them in an ontology. This way makes robots capable in order to perform manipulation with unknown objects by reasoning over the extensible ontological knowledge.

Bibliography

Akbari, A., Lagriffoul, F., and Rosell, J. (2018a). Combined heuristic task and motion planning for bi-manual robots. *Autonomous Robots*, 32(9-10):1–16.

Akbari, A., Muhayyuddin, and Rosell, J. (2015a). Reasoning-based evaluation of manipulation actions for efficient task planning. In *ROBOT2015: Second Iberian Robotics Conference*, pages 69–80. Springer.

Akbari, A., Muhayyuddin, and Rosell, J. (2018b). Knowledge-oriented task and motion planning for multiple mobile robots. *Journal of Experimental & Theoretical Artificial Intelligence*, 31(1):137–162.

Akbari, A., Muhayyudin, and Rosell, J. (2015b). Task and motion planning using physics-based reasoning. In *Emerging Technologies & Factory Automation (ETFA), 2015 IEEE 20th Conference on*, pages 1–7. IEEE.

Akbari, A., Muhayyudin, and Rosell, J. (2016a). Task planning using physics-based heuristics on manipulation actions. In *Emerging Technologies and Factory Automation (ETFA), 2016 IEEE 21st International Conference on*, pages 1–8. IEEE.

Akbari, A., Ud Din, M., and Rosell, J. (2016b). Integrated task and motion planning using physics-based heuristics. In *RSS 2016 Workshop on Task and Motion Planning: posters*, pages 1–4.

Alami, R., Gharbi, M., Vadant, B., Lallement, R., and Suarez, A. (2014). On human-aware task and motion planning abilities for a teammate robot. In *Human-Robot Collaboration for Industrial Manufacturing Workshop, RSS 2014*.

Alili, S., Pandey, A. K., Sisbot, E. A., and Alami, R. (2010). Interleaving symbolic and geometric reasoning for a robotic assistant. In *ICAPS Workshop on Combining Action and Motion Planning*, volume 3, pages 4–3. Citeseer.

Azizi, V., Kimmel, A., Bekris, K., and 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*, pages 764–769. IEEE.

Beetz, M., Klank, U., Kresse, I., Maldonado, A., Mosenlechner, L., Pangercic, D., Ruhr, T., and Tenorth, M. (2011). Robotic roommates making pancakes. In *Humanoid Robots (Humanoids), 2011 11th IEEE-RAS International Conference on*, pages 529–536. IEEE.

Beßler, D., Pomarlan, M., Akbari, A., Diab, M., Rosell, J., Bateman, J., Beetz, M., et al. (2018). Assembly planning in cluttered environments through heterogeneous reasoning. In *Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz)*, pages 201–214. Springer.

Blum, A. L. and Furst, M. L. (1997). Fast planning through planning graph analysis. *Artificial intelligence*, 90(1):281–300.

Bonet, B. and Geffner, H. (2011). Planning under partial observability by classical replanning: Theory and experiments. In *Twenty-Second International Joint Conference on Artificial Intelligence*.

Brafman, R. I. and Shani, G. (2012). Replanning in domains with partial information and sensing actions. *Journal of Artificial Intelligence Research*, 45:565–600.

Bryce, D., Kambhampati, S., and Smith, D. E. (2006). Planning graph heuristics for belief space search. *Journal of Artificial Intelligence Research*, 26:35–99.

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

Clarke, E. M., Grumberg, O., and Peled, D. (1999). *Model checking*. MIT press.

Şucan, I. A. and Kavraki, E. (2012). Accounting for uncertainty in simultaneous task and motion planning using task motion multigraphs. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 4822–4828. IEEE.

da Silva, R. R., Wu, B., and Lin, H. (2016). Formal design of robot integrated task and motion planning. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pages 6589–6594. IEEE.

Dantam, N., Kingston, Z. K., Chaudhuri, S., and Kavraki, L. E. (2016). Incremental task and motion planning: A constraint-based approach. In *Robotics: Science and Systems*, pages 1–6. Ann Arbor, MI, USA.

De Berg, M., Van Kreveld, M., Overmars, M., and Schwarzkopf, O. C. (2000). *Computational geometry*. Springer.

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

de Silva, L., Pandey, A. K., Gharbi, M., and 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*.

Dean, M. and Schreiber, G. (2002). Owl 2 web ontology language. http://www.w3.org/TR/owl2-overview. [Online; accessed 20-May-2014].

Di Marco, D., Levi, P., Janssen, R., van de Molengraft, R., and Perzylo, A. (2013). A deliberation layer for instantiating robot execution plans from abstract task descriptions. In *Int. Conf. on Automated Planning and Scheduling: Workshop on Planning and Robotics. AAAI Press*.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271.

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

Elbanhawi, M. and Simic, M. (2014). A review: motion planning. *IEEE Transactions*, 2:56–77.

Erdem, E., Haspalamutgil, K., Palaz, C., Patoglu, V., and 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*, pages 4575–4581.

Erdem, E., Patoglu, V., and 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.

Erol, K., Hendler, J. A., and Nau, D. S. (1995). Semantics for hierarchical task-network planning. Technical report, DTIC Document.

Fiorini, S. R., Carbonera, J. L., Gonçalves, P., Jorge, V. A., Rey, V. F., Haidegger, T., Abel, M., Redfield, S. A., Balakirsky, S., Ragavan, V., et al. (2015). Extensions to the core ontology for robotics and automation. *Robotics and Computer-Integrated Manufacturing*, 33:3–11.

Freitas, A., Schmidt, D., Panisson, A., Meneguzzi, F., Vieira, R., and Bordini, R. H. (2014). Semantic representations of agent plans and planning problem domains. In *Engineering Multi-Agent Systems*, pages 351–366. Springer.

Garrett, C. R., Lozano-Pérez, T., and Kaelbling, L. P. (2015). FFRob: An efficient heuristic for task and motion planning. In *Algorithmic Foundations of Robotics XI*, pages 179–195. Springer.

Gaschler, A., Petrick, R., Kröger, T., Knoll, A., and Khatib, O. (2013). Robot task planning with contingencies for run-time sensing. In *IEEE International Conference on Robotics and Automation (ICRA) Workshop on Combining Task and Motion Planning*.

Gemignani, G., Capobianco, R., Bastianelli, E., Bloisi, D. D., Iocchi, L., and Nardi, D. (2016). Living with robots: Interactive environmental knowledge acquisition. *Robotics and Autonomous Systems*, 78:1–16.

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

Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107.

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

Hauser, K. and Latombe, J.-C. (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., and Gonzalez-Baños, H. (2010). Multi-modal motion planning for a humanoid robot manipulation task. In *Robotics Research*, pages 307–317. Springer.

He, K., Lahijanian, M., Kavraki, L. E., and Vardi, M. Y. (2015). Towards manipulation planning with temporal logic specifications. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 346–352. IEEE.

Hertle, A. and 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*, pages 6053–6058. IEEE.

Hoffmann, J. and Brafman, R. (2005). Contingent planning via heuristic forward search with implicit belief states. In *Proc. ICAPS*, volume 2005.

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

Kaelbling, L. P. and Lozano-Pérez, T. (2011a). Hierarchical task and motion planning in the now. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1470–1477. IEEE.

Kaelbling, L. P. and Lozano-Pérez, T. (2011b). Planning in the know: Hierarchical belief-space task and motion planning. In *Workshop on Mobile Manipulation, IEEE International Conference on Robotics and Automation, Shanghai, China*.

Kaelbling, L. P. and Lozano-Pérez, T. (2013). Integrated task and motion planning in belief space. *The International Journal of Robotics Research*, 32(9-10):1194–1227.

Karaman, S. and Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894.

Kavraki, L. E., Svestka, P., Latombe, J.-C., and Overmars, M. H. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12(4):566–580.

Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. *The international journal of robotics research*, 5(1):90–98.

Klusch, M., Gerber, A., and Schmidt, M. (2005). Semantic web service composition planning with owls-xplan. In *Proceedings of the AAAI Fall Symposium on Semantic Web and Agents, Arlington VA, USA, AAAI Press*, pages 55–62. sn.

Kuffner, J. J. and LaValle, S. M. (2000). Rrt-connect: An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 2, pages 995–1001. IEEE.

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

Lagriffoul, F., Dantam, N. T., Garrett, C., Akbari, A., Srivastava, S., and Kavraki, L. E. (2018). Platform-independent benchmarks for task and motion planning. *IEEE Robotics and Automation Letters*, 3(4):3765–3772.

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

Lagriffoul, F., Dimitrov, D., Saffiotti, A., and 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*, pages 957–964. IEEE.

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

LaValle, S. M. and Kuffner, J. J. (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.

Lim, G. H., Suh, I. H., and Suh, H. (2011). Ontology-based unified robot knowledge for service robots in indoor environments. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 41(3):492–509.

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

Maliah, S., Brafman, R. I., Karpas, E., and Shani, G. (2014). Partially observable online contingent planning using landmark heuristics. In *ICAPS*.

Muhayyuddin, Akbari, A., and Rosell, J. (2016). Physics-based motion planning: Evaluation criteria and benchmarking. In *Robot 2015: Second Iberian Robotics Conference*, pages 43–55. Springer.

Muhayyuddin, Akbari, A., Rosell, J., and Akbari, A. (2018). k-pmp: Enhancing physics-based motion planners with knowledge-based reasoning. *Journal of intelligent and robotic systems*, 91(3-4):459–477.

Muhayyudin, Akbari, A., and Rosell, J. (2015). Ontological physics-based motion planning for manipulation. In *IEEE Int. Conf. on Emerging Technologies and Factory Automation*, pages 1–7. IEEE.

Muhayyudin, Moll, M., Kavraki, L., Rosell, J., et al. (2018). Randomized physics-based motion planning for grasping in cluttered and uncertain environments. *IEEE Robotics and Automation Letters*, 3(2):712–719.

Niles, I. and Pease, A. (2001). Towards a standard upper ontology. In *Proceedings of the international conference on Formal Ontology in Information Systems-Volume 2001*, pages 2–9. ACM.

Nouman, A., Yalciner, I. F., Erdem, E., and Patoglu, V. (2016). Experimental evaluation of hybrid conditional planning for service robotics. In *International Symposium on Experimental Robotics*, pages 692–702. Springer.

Pednault, E. P. D. (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*, pages 324–332, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Petrick, R. P. and Bacchus, F. (2004). Extending the knowledge-based approach to planning with incomplete information and sensing. In *ICAPS*, pages 2–11.

Plaku, E., Kavraki, L. E., and Vardi, M. Y. (2010). Motion planning with dynamics by a synergistic combination of layers of planning. *IEEE Transactions on Robotics*, 26(3):469–482.

Prestes, E., Carbonera, J. L., Fiorini, S. R., Jorge, V. A., Abel, M., Madhavan, R., Locoro, A., Goncalves, P., Barreto, M. E., Habib, M., et al. (2013). Towards a core ontology for robotics and automation. *Robotics and Autonomous Systems*, 61(11):1193–1204.

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

Rosell, J., Akbari, A., Ud Din, M., and Diab, M. (2018). A knowledge-based planning framework for smart and autonomous manipulation robots. In *Workhshop on Combining Task And Motion Planning In The Frame Of Cloud Robotics, 2018 IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*, pages 1–7.

Rosell, J., Pérez, A., Aliakbar, A., Muhayyuddin, Palomo, L., and 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*, pages 1–8. IEEE.

Saha, I., Ramaithitima, R., Kumar, V., Pappas, G. J., and Seshia, S. A. (2014). Automated composition of motion primitives for multi-robot systems from safe ltl specifications. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 1525–1532. IEEE.

Schlenoff, C., Prestes, E., Madhavan, R., Goncalves, P., Li, H., Balakirsky, S., Kramer, T., and Miguelanez, E. (2012a). An IEEE standard ontology for robotics and automation. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 1337–1342. IEEE.

Schlenoff, C., Prestes, E., Madhavan, R., Goncalves, P., Li, H., Balakirsky, S., Kramer, T., and Miguelanez, E. (2012b). An ieee standard ontology for robotics and automation. pages 1337–1342.

Siméon, T., Laumond, J.-P., Cortés, J., and 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., and 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*, pages 639–646.

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

Stilman, M. and Kuffner, J. J. (2005). Navigation among movable obstacles: Real-time reasoning in complex environments. *International Journal of Humanoid Robotics*, 2(04):479–503.

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

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

Şucan, I. A. and Kavraki, L. E. (2009). Kinodynamic motion planning by interior-exterior cell exploration. In *Algorithmic Foundation of Robotics VIII*, pages 449–464. Springer.

Şucan, I. A. and Kavraki, L. E. (2010). Kinodynamic motion planning by interior-exterior cell exploration. In *Algorithmic Foundation of Robotics VIII*, pages 449–464. Springer.

Tenorth, M. and Beetz, M. (2009). Knowrob knowledge processing for autonomous personal robots. In *Int. Conf. on Intelligent Robots and Systems*, pages 4261–4266.

Tenorth, M. and Beetz, M. (2012). A unified representation for reasoning about robot actions, processes, and their effects on objects. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 1351–1358.

Tosello, E., Fan, Z., and Pagello, E. (2015). A semantic knwoledge base for cognitive robotics manipulator. In *Workshop in Current Advances in Cognitive Robotics*.

Wielemaker, J., Schrijvers, T., Triska, M., and Lager, T. (2012). SWI-Prolog. *Theory and Practice of Logic Programming*, 12(1-2):67–96.

Zaplana, I., Claret, J. A., and nez, L. B. (2018). Análisis cinemático de robots manipuladores redundantes: Aplicación a los robots kuka lwr 4+ y abb yumi. *Revista Iberoamericana de Automı́ca e Informática industrial*, 15(2):192–202.