Knowledge-oriented Task and Motion Planning for Multiple Mobile Robots

Aliakbar Akbari*, Muhayyuddin, and Jan Rosell

Institute of Industrial and Control Engineering (IOC), Universitat Politècnica de Catalunya (UPC), Barcelona, Spain

Robotic systems composed of several mobile robots moving in human environments pose several problems at perception, planning and control levels. In these environments, there may be obstacles obstructing the paths, which robots can remove by pushing or pulling them. At planning level, therefore, an efficient combination of task and motion planning is required. Even more if we assume a cooperative system in which robots can collaborate with each other by e.g. pushing together a heavy obstacle or by one robot clearing the way to another one. In this paper, we cope with this problem by proposing κ-TMP, a smart combination of an heuristic task planner based on the Fast Forward method, a physics-based motion planner, and reasoning processes over the ontologies that code the knowledge on the problem. The significance of the proposal relies on how geometric and physics information is used within the computation of the heuristics in order to guide the symbolic search, i.e., how an artificial intelligence planning method is combined with low-level motion planning to achieve a feasible sequence of actions (composed of collision-free motions plus physically-feasible push/pull actions). The proposal has been validated with several simulated scenarios (using up to five robots that need to collaborate with each other to reach the goal state), showing how the method is able to solve challenging situations and also find an efficient solution in terms of power.

Keywords: Task and motion planning; manipulation planning; knowledge-based representation; reasoning process.

1. Introduction

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

*Corresponding author. Email: aliakbar.akbari@upc.edu

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: The present study proposes 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.

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 is currently under development as pointed out in the conclusions.

The rest of the paper is structured as follows. First, Section 2 summarizes some related work and Section 3 formulates the problem and explains the solution overview. Then, Section 4 presents and illustrates the semantic manipulation knowledge, Section 5 the off-line and on-line reasoning processes on symbolic literals, and Section 6 details how the knowledge-based task level and the motion planning level are combined. Finally, Section 7 shows some implementation issues and simulation results, and Section 8 sketches the conclusions and future works.

2. Related Work

2.1. *Heuristic Task Planning*

One of the most efficient task planning approaches among those that search in the state space is the Fast Forward (FF, (Hoffmann & Nebel, 2001)), which performs a heuristic search. It has two main components, as depicted in Figure 1, 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 these heuristic values in terms of the estimated number of actions needed to reach the goal. 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* (Blum & Furst, 1997), that is a graph that interleaves state-levels (involving a number of literals) and action-levels (representing a set of actions). Mutual exclusion relations are defined among actions as well as among literals (indicating how the combination of literals can be true at each state-level). Action-levels contain actions whose preconditions are met in the previous state-level, and they may add or delete some literals in the subsequent state-level. The construction phase is launched from a state-level that includes the initial state of the problem, and continues till a state-level is found where all

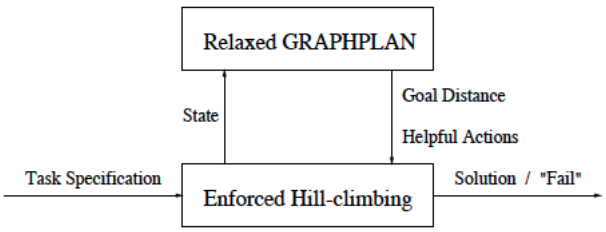


Figure 1.: The Fast-Forward planner architecture (from (Hoffmann & Nebel, 2001)).

goal conditions are satisfied. The relaxed version of the Planning Graph (called RPG) ignores the delete lists of the actions (so mutual exclusion relations do not take effect in the planning phase). The heuristic value is then computed as the number of actions in the plan extracted from the RPG, and the helpful actions are those actions of the RPG appearing in the first action level. If EHC fails, everything done so far is skipped and the FF restarts considering Best-First Search (BFS) instead of EHC.

2.2. *Task and Motion Planning*

Different approaches have dealt with different strategies to combine task planning (based on high-level symbolic reasoning) with motion planning (based on low-level geometric computations), with the aim of finding a feasible plan to solve a given task. Some of them like (Erdem, Haspalamutgil, Palaz, Patoglu, & Uras, 2011; Srivastava et al., 2014), consider an independent module as a generic interface between both planning levels, or define a semantic attachment module to a planning domain description as a tool to appraise the truth value of grounded predicates by operating on geometric information of the states (Dornhege et al., 2012). Other approaches also deal with ways to relate both planning levels, like (Galindo, Fernández-Madrigal, González, & Saffiotti, 2008), (Dearden & Burbridge, 2014), and (Mansouri & Pecora, 2016) that use semantic map, learning techniques, and hybrid reasoning for that purpose, respectively. There are also some approaches such as (Cashmore et al., 2015; Kimmel et al., 2012) proposing an implementation framework which provides an environment to develop task and motion planning problems. On the other hand, however, there are approaches not that general, but more dependant on the task planner used, like for instance those based on Hierarchical Task Networks (HTN), on heuristic search methods as FF, or based on constraint solving.

Among those based on HTN, the work of de Silva, Pandey, Gharbi, and Alami (2013) focuses on a combination that facilitates backtracking at different levels, also including an interleaved backtracking procedure. Also based on HTN, the work in Kaelbling and Lozano-Pérez (2011) presents an aggressively hierarchical approach that constrains the abstract plan steps so that they are serializable (i.e. so that the particular way that the first step is carried out does not make it impossible to carry out subsequent steps), and handles the integration by operating on detailed, continuous geometric representations.

There are some planning approaches using different variants of the FF planner, which is also used for the current proposal. The work in Cambon, Alami, and Gravot (2009) presents an interleaved search at the symbolic and geometric levels, where the motion planner, which is based on the PRMs, calls the task planner to guide roadmap sampling. Upon failure of a selected action, the PRM is left for further exploration (thus considering the probabilistic completeness of sampling-based motion planners). The approach computes the heuristic value which relies on symbolic distance to goal. Therefore, the heuristic function is not informed in terms of geometric

information. To compensate this lack, the work of Garrett, Lozano-Pérez, and Kaelbling (2015) proposes an approach, called FFRob, that when computing the heuristics analyses the action feasibility by using a conditional reachability graph based on a probabilistic roadmap motion planner.

Finally, among those based on constraint solving, the approaches (da Silva, Wu, & Lin, 2016; Saha, Ramaithitima, Kumar, Pappas, & Seshia, 2014) have embedded different motion primitives within constraint-based task planners. These approaches focus on difficulties found in motion planning problems for mobile robots like avoiding dynamic obstacles when a robot is doing its task. With focused on manipulation tasks, the work in Neil T. Dantam and Kavraki (2016) proposes the Iteratively Deepened Task and Motion Planning method that employs an incremental constraint solver and keeps dynamically adding or eliminating a number of task constrains according to the feedback received from the motion planner. The approach is able to find an alternative plan when an infeasible one is identified. It first finds the task plan, and then motion planning is employed to evaluate the feasibility of actions in the plan. The concept of geometric backtracking has been investigated by the works of Lagriffoul, Dimitrov, Saffiotti, and Karlsson (2012) and Lagriffoul, Dimitrov, Bidot, Saffiotti, and Karlsson (2014) in which a set of linear constraints is generated from the kinematic model of the robot and the symbolic actions computed by the task planner. These constraints are used for pruning during geometric backtrack search. These approaches may restart the whole planning process in the case of identifying geometric constraints while evaluating geometrically the final plan. In contrast to these approaches, the current proposal considers simultaneously task and motion planning, where the FF task planner is also guided by physics-based motion planning information by taking into account geometry constraints while planning. Therefore, the proposed combination avoids to restart the whole planning process when a new geometric constrain is detected.

To the best of our knowledge, there are no approaches within the framework of simultaneous task and motion planning that use a physics-based motion planner and information to guide task planning. This paper aims to contribute in this line by taking into account how the task planning search can be enhanced using the physics-based information, resulting a power-efficient and feasible manipulation plan. Moreover, it considers how multi mobile robots can collaborate or share a task together in order to solve a manipulation problem.

2.3. *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) whose main purpose is to classify knowledge on a world-wide accessible database (developing and depicting ontology models can be done using the *Protégé* tool[1] which is a powerful and flexible editor to represent ontology applications).

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.

Knowledge-based representation techniques, like ontologies, can enhance manipulation planning by providing informative data with respect to the robot's world. In this regard, for instance, knowledge concerning housework activities was elaborated in (Tenorth & Beetz, 2009)

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

and (Tenorth & Beetz, 2012) describing and reasoning over ontological actions along their effects, and the approach in (Provine, Schlenoff, Balakirsky, Smith, & Uschold, 2004) used ontologies for path planning. Ontological knowledge has also been integrated in HTN-based planning (Di Marco, Levi, Janssen, van de Molengraft, & Perzylo, 2013; Freitas et al., 2014), or to encode information inside a high-level planning that combines the FF with HTN planning (Klusch, Gerber, & Schmidt, 2005).

2.4. *Physics-based Motion Planning*

Kinodynamic motion planning refers to the planning of a collision-free trajectory from a start to a goal state while satisfying a given set of (kinematic and dynamic) constraints. The planning is performed in the state space that records the robot dynamics. At any time, the state of the system is described as $x = (q, \dot{q})$ where q represents the configuration of the robot in the configuration space. A forward propagation step is performed using the propagation function described as $\dot{x} = f(x, u)$, with u representing the control inputs. The sampling-based motion planners, especially tree-based planners such as the Rapidly-Exploring Random Tree (RRT, (LaValle & Kuffner, 2001)), the Kinodynamic Motion Planning by Interior-Exterior Cell Exploration (KPIECE, (Sucan & Kavraki, 2012)) and the Synergistic Combination of Layers of Planning (SyCLoP, (Plaku, Kavraki, & Vardi, 2010)), can plan efficiently in the existence of kinodynamic constraints.

Physics-based motion planning is an enhanced form of kinodynamic motion planning that incorporates a physics engine (such as Bullet (Erwin, 2013) or ODE (Russell, 2007)) as state propagator. Therefore, it allows the handling of manipulation tasks since it considers the dynamic interaction between rigid bodies along with the kinodynamic and physics-based constraints, like in the approach presented in (Stilman & Kuffner, 2005) that considers the problem of navigating among movable obstacles that may be moved away to clear the path for the robot. To deal with the computational complexity of physics-based motion planners, a few approaches have been proposed to reduce the planner search space, like the work in (Zickler & Veloso, 2009) that uses a non-deterministic finite state machine to guide control sampling. With the same aim, the approach in (Muhayyuddin, Akbari, & Rosell, 2017a) performed a reasoning process over the high-level knowledge (stored in the form of ontologies) to guide the low-level motion planner by delimiting from where the objects can be interacted.

2.5. *Integrating Task Planning and Physics-based Motion Planning*

Recently, we addressed the manipulation problem of a mobile robot that is able to push and pull movable objects by combining different knowledge-based task planners with physics-based motion planners. First, a modified version of the *Planning Graph* algorithm was proposed (Akbari, Muhayyuddin, & Rosell, 2015b) to allow the retrieving of a number of potential plans that were then evaluated by a physics-based motion planner to find the least-cost feasible one. Then, the approach was modified to evaluate the feasibility of actions while planning (Akbari, Muhayyuddin, & Rosell, 2015a), allowing to cut off some infeasible action branches at the task level. These approaches are computationally expensive in terms of number of calls to the motion planner. To mitigate the aforementioned drawback, a more efficient combination of task and physics-based motion planning was suggested based on a version of the FF planner (Akbari, Muhayyudin, & Rosell, 2016) that called the physics-based motion planner on the actions selected in the RPG plan computed to obtain the heuristic. The approach was designed for a single robot and also considered, in order to determine the pre- and post-condition of the push and pull actions, the geometric reasoning concerning the connectivity of the configuration space.

The present study extends this latest approach to tackle problems in which multiple robots may either share tasks (e.g. one robot moves an object away to free the path for the other robot), or cooperate with each other (e.g. by pushing the same object). Accordingly, the graph of configuration space connectivity is dynamically updated with respect to the results of the actions applied by each robot during the planning process. Moreover, the current proposal is more efficient computationally in terms of the number of queries set to the motion planner.

3. Problem Formulation and Solution Overview

3.1. *Scope and Motivating Example*

The scope of the present proposal is to deal with collaborative tasks 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.

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

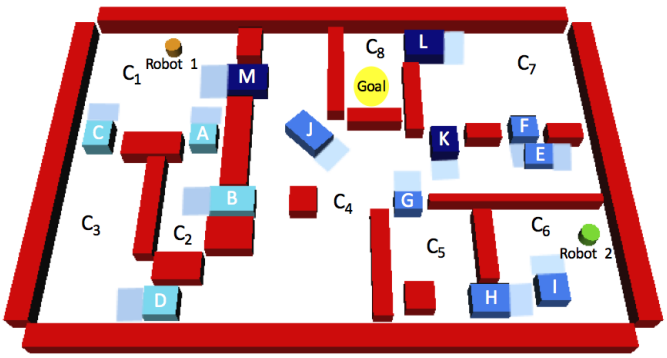


Figure 2.: A manipulation example: two robots (shown as small cylinders) are required to reach the goal region by pushing or pulling some obstacles in the way (manipulatable obstacles are shown as boxes and are labeled in an increasing order with respect to their weight). Workspace regions are labelled with the name of the corresponding disjointed configuration space regions.

a robot.

As a motivating example, consider the problem shown in Figure 2 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 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.

3.2. *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 (Section 3.2.1).
- S is a set of states containing both literals and geometric information (Section 3.2.2).
- $\mathcal{K}_w$ represents as an ontology the semantic knowledge about the robots and the environment (Section 3.2.3).
- $\mathcal{K}_p$ represents as an ontology the knowledge about the planning components (Sec-

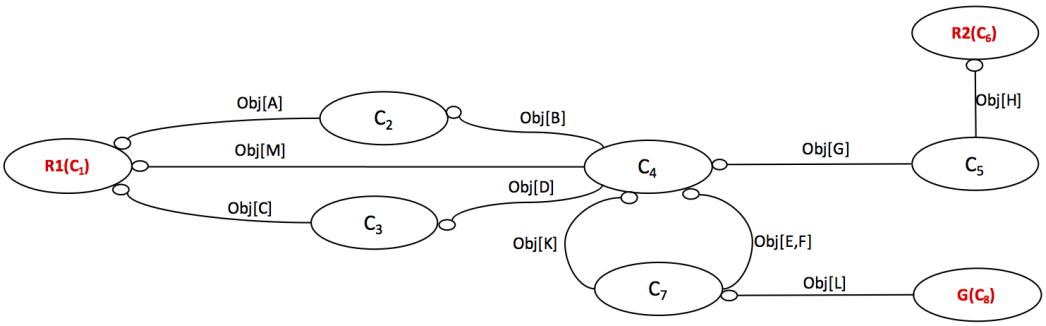


Figure 3.: Graph $\mathcal{R}$ representing the connectivity of the configuration space for the example in Figure 2. The nodes are labeled with the disjointed configuration space regions; the edges with the critical obstacles.

tion 3.2.4).

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.

3.2.1. 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 & 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. Figure 3 represents the graph associated to the example of Figure 2, 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*.

3.2.2. 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 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.

3.2.3. 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 joint capability, but cannot be displaced by any of them separately.

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

3.2.4. 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 $\langle$*name(a), pre(a), effect$^+$(a), effect$^-$(a)*, $\mathcal{Q}\rangle$ 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:

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

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 (but in a similar manner as actions and their conditions are defined in PDDL (Ghallab et al., 1998), a standard language used to represent task planning problems). 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.

3.3. *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 (Fig. 1). The new version (sketched in Figure 4) uses 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.

3.3.1. 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. It is described in Section 4.

3.3.2. 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, 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 $\mathcal{T}$.
- 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.

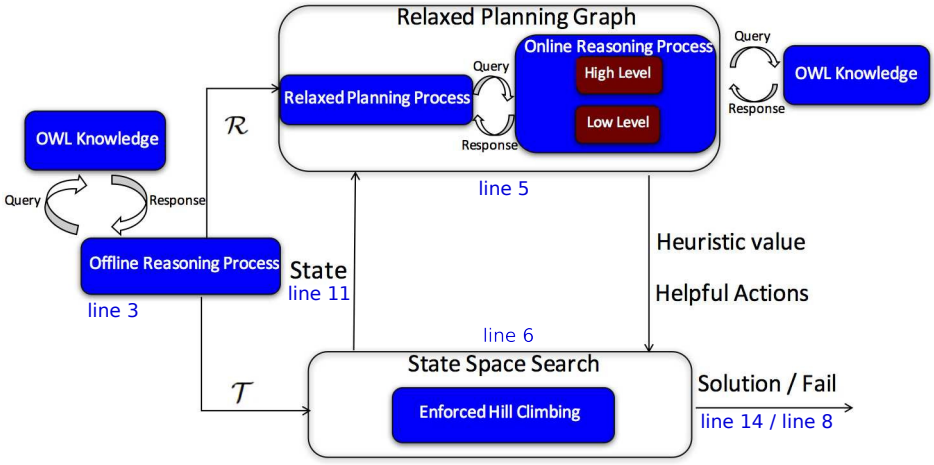


Figure 4.: The κ-TMP planner architecture (indicated lines correspond to Algorithm 1).

3.3.3. 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.

4. Semantic Manipulation Knowledge

The proposed manipulation ontology model, developed with the *Protégé* editor in terms of OWL, entails two main classes called *ManipulationWorld* and *ManipulationPlanning*[2], in order to code $\mathcal{K}_w$ and $\mathcal{K}_p$. Class *ManipulationWorld*, illustrated in Figure 5, is structured in the following subclasses:

- *Obstacles*: Class that retains necessary information with respect to all the obstacles in the world. The information included comprises geometry and physical properties like objects

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

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.

- *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 the corresponding push/pull action.
- *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 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 robot, the response to gravity, the friction, mass, and color values respectively.

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

On the other hand, class *ManipulationPlanning*, illustrated in Figure 5, 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 and bind them with their preconditions and side effects.
- *Predicates*: Class that represents the predicates *At*, *IsCrit*, *HasAcc*, *IsManp*, as well as *IsManpMRob* used to define the actions (only predicate *At* is shown in Figure 5).

Figure 5.: Manipulation taxonomy

5. 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.

5.1. *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 physical properties of the obstacles.
- *IsManpMRob* is asserted for the critical obstacles according to the joint capability of the robots and the physical properties of the obstacles.
- *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)*.

5.2. *Online Reasoning Process*

The online reasoning process contains a high-level reasoning module and a low-level reasoning module, as shown in Figure 6, 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 Figure 6), 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 (condition 2 in Figure 6), 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

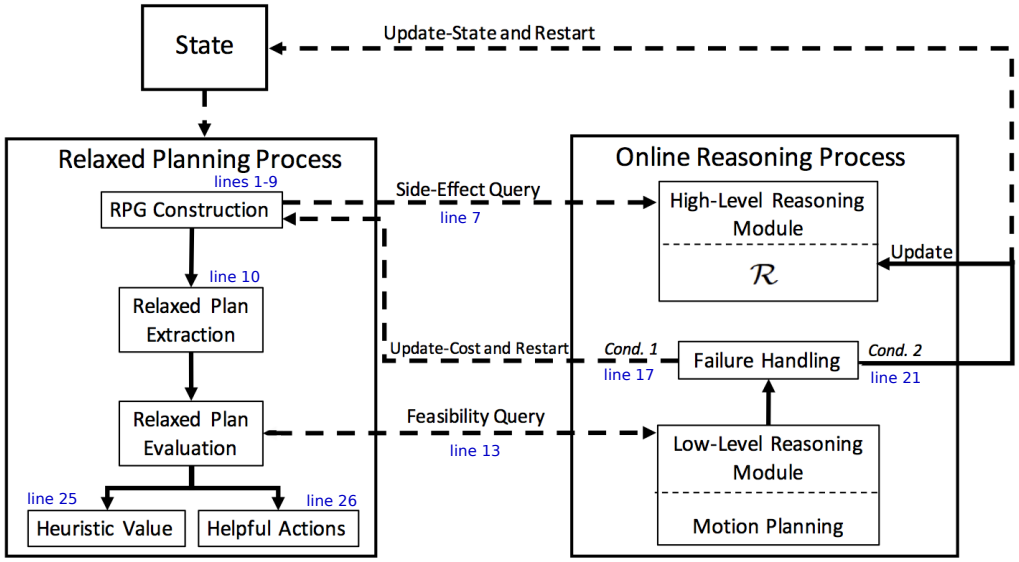


Figure 6.: Information flow between the relaxed planning process and the online reasoning process, corresponding to the module 'Relaxed Planning Graph' in Fig. 4. Indicated lines correspond to Algorithm 2.

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:

$$\mathcal{C} = \sum_{j}^{n} \frac{\mathbf{f}_j \cdot \mathbf{d}_j}{\Delta t_j}, \tag{1}$$

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.

6. 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 Figure 4.

6.1. *State Space Search*

Algorithm 1 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 4-13]: 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 selectHAction_usingEHC 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 (no feasibility problems may arise since for these actions the satisfaction of the preconditions assures it).

Algorithm 1 κ-TMP

Input: $\mathcal{K}_p, \mathcal{K}_w$
Output: The feasible plan π

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** $\mathcal{G} \not\subseteq S_\text{i}$ **do**
5: $\quad\{h(S_\text{i}), H(S_\text{i})\} \leftarrow$ RPG$(S_\text{i}, \mathcal{G}, \mathcal{K}_p, \mathcal{K}_w, \mathcal{R})$
6: $\quad\{H'(S_\text{i}))\} \leftarrow$ selectHAction_usingEHC$(H(S_\text{i}), h(S_\text{i}))$
7: $\quad$**if** $H'(S_\text{i}).empty()$ **then**
8: $\quad\quad$**return** fail
9: $\quad$**else**
10: $\quad\quad\pi$.append$(H'(S_\text{i}))$
11: $\quad\quad S_\text{i+1} \leftarrow$ Succ$(S_\text{i}, H'(S_\text{i}))$
12: $\quad$**end if**
13: **end while**
14: **return** π

6.2. *The Relaxed Planning Graph 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:

$$cost(transit) = 1 \quad (2)$$
$$cost(push/pull) = m_i/m_j \quad (3)$$

  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:

$$cost(l) = min_{\forall a_i \mid l \in effect^+(a_i)}\{cost(a_i) + \sum cost(pre(a_i))\} \quad (4)$$

  Note that a_i is any of the actions introduced in Sec.3.2.4 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. (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:

$$h = \sum_{i}^{n} \mathcal{C}_i \quad (5)$$

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 2 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 key points are as follows:

- *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

Algorithm 2 RPG$(S, \mathcal{G}, \mathcal{K}_p, \mathcal{K}_w, \mathcal{R})$

```
1: 𝕊_0 ← S
2: i ← 0
3: while i <MaxLevels do
4:    i ← i + 1
5:    𝔸_i ← {a ∈ K_p.A | pre(a) ⊆ 𝕊_{i-1}}
6:    𝔸_i ← append(maintActions(𝕊_{i-1}))
7:    𝕊_i ← {l | l ∈ effect+(𝔸_i)}
8:    compCost(𝕊_i)
9:    if G ⊆ 𝕊_i AND checkCost() then
10:      π' ←RPGPlan()
11:      h ← 0
12:      for each {a ∈ π' | a.name ≠ Transit} do
13:         {f.response, ColObj} ← feasibilityChecker(a, K_w)
14:         if f.response = infeasible then
15:            if ColObj ≠ MObs then
16:               setHighCost(a)
17:               GOTO 4      //Cond. 1 in Figure 6
18:            else
19:               updateR()
20:               updateState()
21:               GOTO 1      //Cond. 2 in Figure 6
22:            end if
23:         end if
24:      end for
25:      h ← heuristicValue()
26:      H(S) ← helpfulActions()
27:      return {h, H(S)}
28:   end if
29: end while
30: return {∞, ∅}
```

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 equation (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 18-22]. 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). The helpful actions are extracted by helpfulActions.

7. Implementation and Results

7.1. *Implementation*

The proposed framework implementation consists of three major layers as depicted in Figure 7: ontological knowledge, task-level and motion-level layers. The knowledge layer is coded in the form of an OWL ontology as detailed in Section 4. 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, Moll, & Kavraki, 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 & Kavraki, 2010) has been used in the experiments because in a comparative study (Gillani, Akbari, & Rosell, 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 & Beetz, 2009), a potent knowledge processing tool that enables a flexible access to 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, Schrijvers, Triska, & Lager, 2012)).

7.2. *Results and discussion*

The problem posed in Figure 2 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 Figure 8, is:

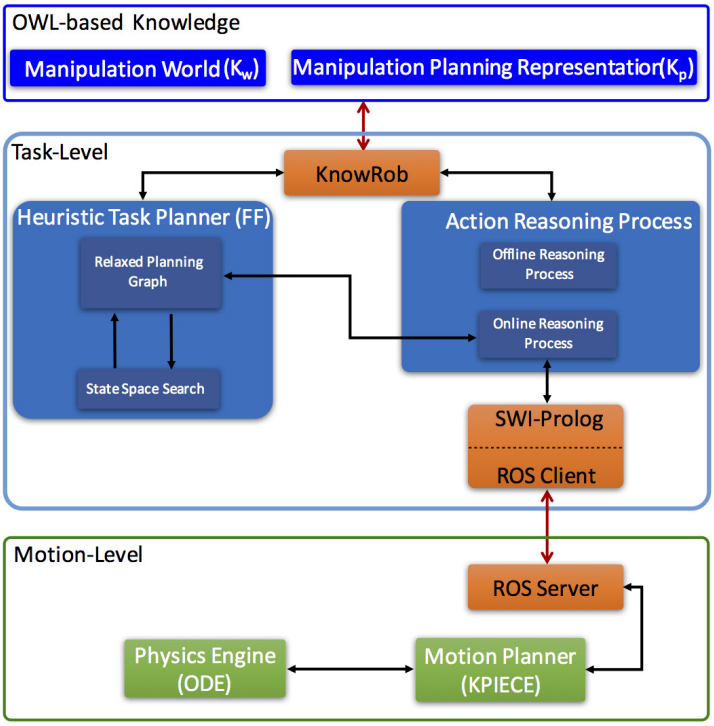


Figure 7.: The proposed framework

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

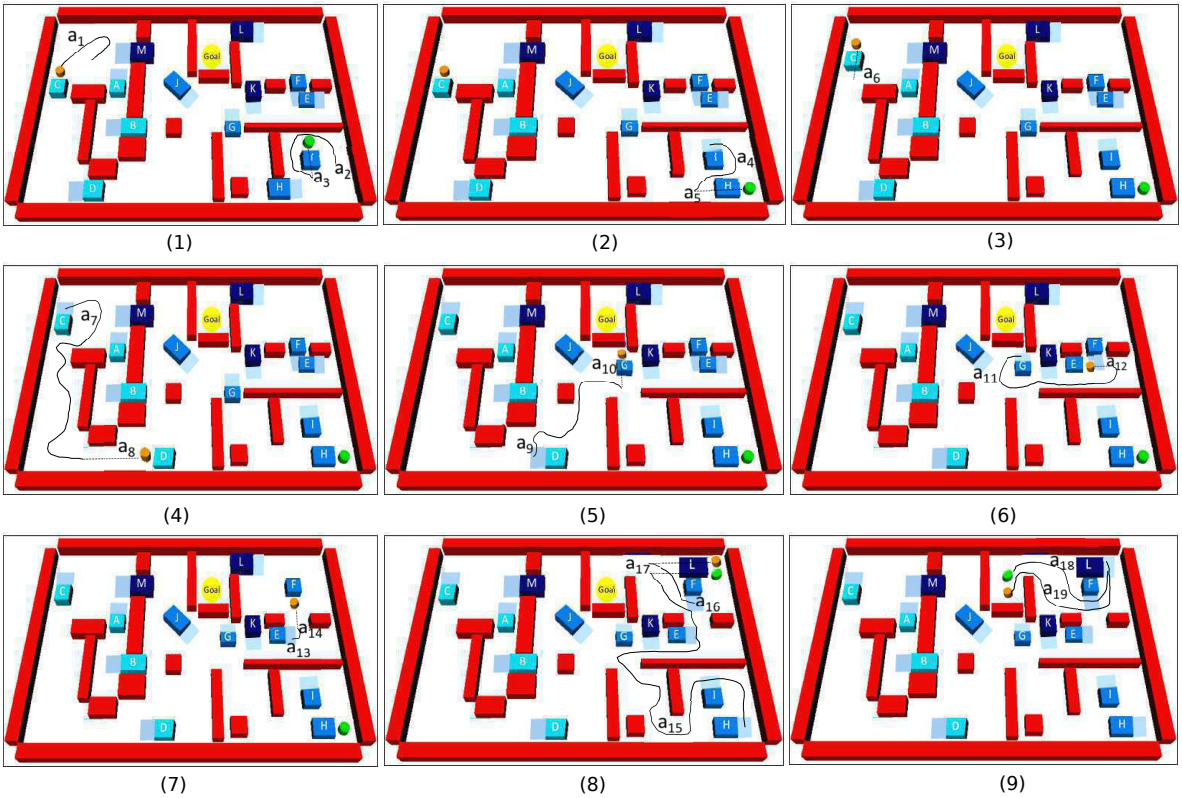


Figure 8.: Snapshots of the task execution: 1) actions a_1, a_2 and a_3; 2) actions a_4 and a_5; 3) action a_6; 4) action a_7 and a_8; 5) actions a_9 and a_{10}; 6) actions a_{11} and a_{12}; 7) actions a_{13} and a_{14}; 8) actions a_{15}, a_{16} and a_{17}; 9) actions a_{18} and a_{19}. Video: https://sir.upc.edu/projects/kautham/videos/k-TMP.mp4

(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 includes interactions with *MObs* M or B is not physically executable. The proposed κ-TMP planner is able to find a power-optimal solution 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.

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.

To compare the performance of the present proposal we have extended the approach in Akbari et al. (2016) (here called *A-TMP*) by adding features **b** and **e** from κ-TMP to allow *A-TMP* to find a feasible plan (*A-TMP* already shared the feature **a**), otherwise it could not find a solution. 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 Figure 9. Figure 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 Figure 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 the action space has not been changed, i.e., actions are performed by only one or two robots (as an example, Figure 10 shows a scenario with five robots). Figure 11 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.

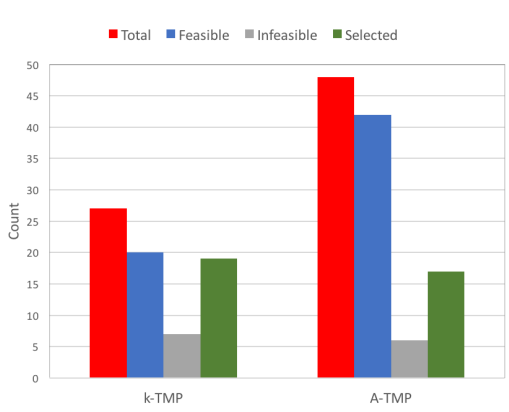
It is worth noting that the approach uses a probabilistic complete motion planer (i.e. one that finds a solution if one exists provided enough time is left to the planner). Then, to evaluate the feasibility of the push/pull actions, if the motion planner times-out, the action is not pruned but its cost 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 (da Silva et al., 2016; Saha et al., 2014) 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 (Muhayyuddin et al., 2017a; Muhayyuddin, Akbari, & Rosell, 2017b).

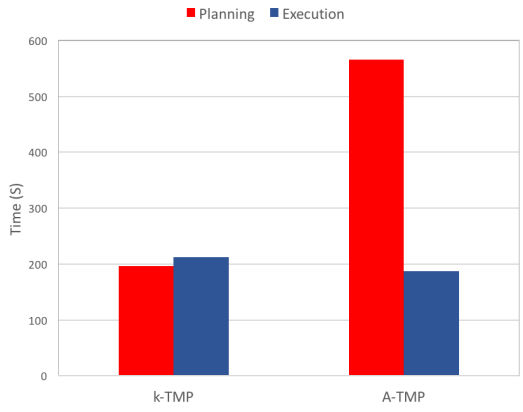
8. Conclusions

An approach, called κ-TMP, has been presented 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

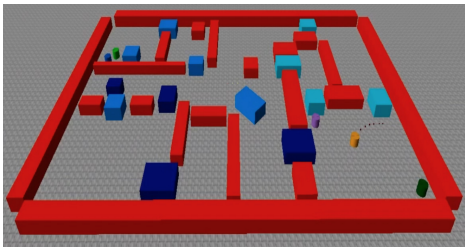


(a) Histogram of actions evaluated by the motion planner: the total number of actions, feasible, infeasible, and selected actions.

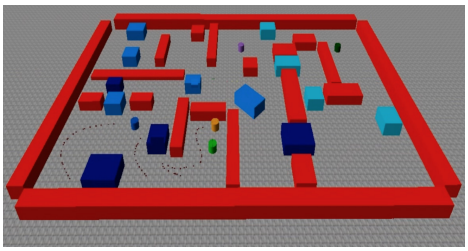


(b) Histogram of planning and executive time.

Figure 9.: Performance of each manipulation planner.



(a) The initial state of the manipulation problem.



(b) The goal state of the manipulation problem.

Figure 10.: The manipulation problem where five robots collaborate between them to solve the task and reach their target regions. Video: https://sir.upc.edu/projects/kautham/videos/k-TMP-5.mp4

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 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 rea-

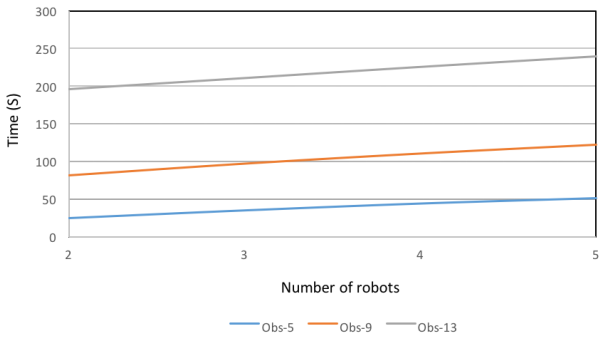


Figure 11.: Experiments with different number of robots and obstacles including 5 obstacles (Obs-5), 9 obstacles(Obs-9), and 13 obstacles(Obs-13). Some details of problems can be found in https://sir.upc.es/projects/ontologies/.

soning 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 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.

Currently, the approach is being extended to consider manipulation planning for pick and place problems for bi-manual robots. To deal with that, the reasoning process has been augmented to consider collaboration between robotic arms and to reason about various task constraints, like finding appropriate grasps, inverse kinematic solutions, and objects placement. The proposal focuses on the use a sampling-based mechanism inside task and motion planning, i.e., manipulatable objects blocking the connectivity of configuration space are detected while planning. This way makes the planner efficient in solving high dimension manipulation problems by skipping the precomputation of the configuration space connectivity. The proposal will also focus in including the handling of uncertainty which may arise from the motion level to the task level using a contingency-based task planner.

Funding

This work is partially supported by the Spanish Government through the project DPI2016-80077-R. Aliakbar Akbari is supported by the Spanish Government through the grant FPI 2015.

References

Akbari, A., Muhayyuddin, & Rosell, J. (2015a). Reasoning-based evaluation of manipulation actions for efficient task planning. In *Robot2015: Second iberian robotics conference.*

Akbari, A., Muhayyuddin, & Rosell, J. (2015b). Task and motion planning using physics-based reasoning. In *Emerging technologies & factory automation (etfa), 2015 ieee 20th conference on* (pp. 1–7).

Akbari, A., Muhayyudin, & Rosell, J. (2016). Task planning using physics-based heuristics on manipulation actions. In *Ieee int. conf. on emerging technologies and factory automation.*

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

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

Cashmore, M., Fox, M., Long, D., Magazzeni, D., Ridder, B., Carrera, A., ... Carreras, M. (2015). Rosplan: Planning in the robot operating system. In *Icaps* (pp. 333–341).

da Silva, R. R., Wu, B., & Lin, H. (2016). Formal design of robot integrated task and motion planning. *arXiv preprint arXiv:1604.05657*.

Dearden, R., & Burbridge, C. (2014). Manipulation planning using learned symbolic state abstractions. *Robotics and Autonomous Systems*, *62*(3), 355–365.

de Silva, L., Pandey, A. K., Gharbi, M., & Alami, R. (2013). Towards combining HTN planning and geometric task planning. In *Rss workshop on combined robot motion planning and ai planning for practical applications.*

Di Marco, D., Levi, P., Janssen, R., van de Molengraft, R., & 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.*

Dornhege, C., Eyerich, P., Keller, T., Trüg, S., Brenner, M., & Nebel, B. (2012). Semantic attachments for domain-independent planning systems. In *Towards service robots for everyday environments* (pp. 99–115). Springer.

Erdem, E., Haspalamutgil, K., Palaz, C., Patoglu, V., & Uras, T. (2011). Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation. In *Ieee int. conf. on robotics and automation robotics and automation* (pp. 4575–4581).

Erwin, C. (2013, October). *Bullet physics library, http://bulletphysics.org.*

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

Galindo, C., Fernández-Madrigal, J.-A., González, J., & Saffiotti, A. (2008). Robot task planning using semantic maps. *Robotics and autonomous systems*, *56*(11), 955–966.

Garrett, C. R., Lozano-Pérez, T., & Kaelbling, L. P. (2015). FFRob: An efficient heuristic for task and motion planning. In *Algorithmic foundations of robotics xi* (pp. 179–195). Springer.

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

Gillani, M., Akbari, A., & Rosell, J. (2016). Physics-based motion planning: Evaluation criteria and benchmarking. In *Robot 2015: Second iberian robotics conference* (pp. 43–55).

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

Kaelbling, L. P., & Lozano-Pérez, T. (2011). Hierarchical task and motion planning in the now. In *Ieee int. conf. on robotics and automation robotics and automation* (pp. 1470–1477).

Kimmel, A., Dobson, A., Littlefield, Z., Krontiris, A., Marble, J., & Bekris, K. E. (2012). Pracsys: An extensible architecture for composing motion controllers and planners. In *International conference on simulation, modeling, and programming for autonomous robots* (pp. 137–148).

Klusch, M., Gerber, A., & 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.*

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

Lagriffoul, F., Dimitrov, D., Saffiotti, A., & Karlsson, L. (2012). Constraint propagation on interval bounds for dealing with geometric backtracking. In *Intelligent robots and systems (iros), 2012 ieee/rsj international conference on* (pp. 957–964).

LaValle, S. M., & Kuffner, J. J. (2001). Randomized kinodynamic planning. *The International Journal of Robotics Research*, *20*(5), 378–400.

Mansouri, M., & Pecora, F. (2016). A robot sets a table: a case for hybrid reasoning with different types of knowledge. *Journal of Experimental & Theoretical Artificial Intelligence*, *28*(5), 801–821.

Muhayyuddin, Akbari, A., & Rosell, J. (2017a). κ-pmp: Enhancing physics-based motion planners with knowledge-based reasoning. In (pp. 1–19). Springer.

Muhayyuddin, Akbari, A., & Rosell, J. (2017b). Knowledge-oriented physics-based motion planning for grasping under uncertainty. In *Iberian robotics conference* (pp. 502–515).

Neil T. Dantam, S. C., Zachary K. Kingston, & Kavraki, L. E. (2016). Incremental task and motion planning: A constraint-based approach. In *Rss workshop on task and motion planning.*

Plaku, E., Kavraki, L., & Vardi, M. (2010, June). Motion planning with dynamics by a synergistic combination of layers of planning. *IEEE Tran. on Robotics*, *26*(3), 469-482.

Provine, R., Schlenoff, C., Balakirsky, S., Smith, S., & Uschold, M. (2004). Ontology-based methods for enhancing autonomous vehicle path planning. *Robotics and Autonomous Systems*, *49*(1), 123–133.

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

Rosell, J., Pérez, A., Aliakbar, A., Muhayyuddin, Palomo, L., & García, N. (2014). The Kautham Project: A teaching and research tool for robot motion planning. In *Ieee int. conf. on emerging technologies and factory automation.*

Russell, S. (2007). *Open Dynamic Engine.* http://www.ode.org/.

Saha, I., Ramaithitima, R., Kumar, V., Pappas, G. J., & 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* (pp. 1525–1532).

Srivastava, S., Fang, E., Riano, L., Chitnis, R., Russell, S., & Abbeel, P. (2014). Combined task and motion planning through an extensible planner-independent interface layer. In *Ieee int. conf. on robotics and automation robotics and automation* (pp. 639–646).

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

Sucan, I., & Kavraki, L. E. (2012). A sampling-based tree planner for systems with complex dynamics. *Robotics, IEEE Transactions on*, *28*(1), 116–131.

Şucan, I. A., & Kavraki, L. E. (2010). Kinodynamic motion planning by interior-exterior cell exploration. In *Algorithmic foundation of robotics viii* (pp. 449–464). Springer.

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

Tenorth, M., & Beetz, M. (2009). Knowrob knowledge processing for autonomous personal robots. In *Int. conf. on intelligent robots and systems* (pp. 4261–4266).

Tenorth, M., & 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* (pp. 1351–1358).

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

Zickler, S., & Veloso, M. (2009). Efficient physics-based planning: sampling search via non-deterministic tactics and skills. In *8th int. conf. on autonomous agents and multiagent systems-volume 1* (pp. 27–33).