Reasoning-Based Evaluation of Manipulation Actions for Efficient Task Planning

Aliakbar Akbari, Muhayyuddin Gillani and Jan Rosell

Abstract To cope with the growing complexity of manipulation tasks, the way to combine and access information from high- and low-planning levels has recently emerged as an interesting challenge in robotics. To tackle this, the present paper first represents the manipulation problem, involving knowledge about the world and the planning phase, in the form of an ontology. It also addresses a high-level and a low-level reasoning processes, this latter related with physics-based issues, aiming to appraise manipulation actions and prune the task planning phase from dispensable actions. In addition, a procedure is contributed to run these two-level reasoning processes simultaneously in order to make task planning more efficient. Eventually, the proposed planning approach is implemented and simulated through an example.

Keywords Task planning · Reasoning process · Manipulation

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, e.g., pushing different sort of objects holding unique characteristics in human-like environments. In solving a given complex manipulation problem, high-level beside 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. In this scope, many studies apply recent and efficient task

A. Akbari · M. Gillani · J. Rosell(✉)
Institute of Industrial and Control Engineering,
Universitat Politècnica de Catalunya, Barcelona, Spain
e-mail: {aliakbar.akbari,muhayyuddin.gillani,jan.rosell}@upc.edu

J. Rosell—This work was partially supported by the Spanish Government through the projects DPI2011-22471, DPI2013-40882-P and DPI2014-57757-R. Muhayyuddin is supported by the Generalitat de Catalunya through the grant FI-DGR 2014.

L.P. Reis et al. (eds.), *Robot 2015: Second Iberian Robotics Conference*,
Advances in Intelligent Systems and Computing 417,
DOI: 10.1007/978-3-319-27146-0_6

planning approaches such as hierarchical-based task planning or GraphPlan in order to combine them with motion planning by different techniques (e.g., [1][2][3]).

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. In this line, the work in [4] presented an approach, called *smart motion planning*, that is able to incorporate ontology-based knowledge within a physics-based motion planner with the purpose of allowing contact between the robot and the objects only from where these objects can be manipulated. Also in a similar way, a lightweight reasoning process has been proposed by [5] that evaluates stability, reachability, and also visibility with respect to manipulation actions that conditions the motion planning.

Solving mobile manipulation problems by evaluating several possible alternative plans has been recently deliberated as an interesting research line. For instance, the Task Motion Multigraph (TMM [6]) has been introduced to simultaneously plan the motions of alternative task plans, although without considering physics-based issues. On the contrary, the recent work [7] suggested the integration of task planning with a physics-based reasoning module. The approach first collects all possible plans and then compares them based on their feasibility, in order to realize the best one according to the minimum total action cost (the pruning of unnecessary actions while planning is not considered). Following this latter work, the present study introduces a framework based on a version of GraphPlan that uses a reasoning process (based on an ontological knowledge modelling) and a physics-based motion planner to identify unnecessary actions (inessential, ineffective, or infeasible) and exclude them from the planning graph. This pruning of the task planning search space makes task planning more efficient.

2 Problem Statement and Solution Overview

A mobile manipulator is considered that is able to deal with two actions, to push removable objects and to freely move around. The problem to be solved is to efficiently find a sequence of actions to perform a given task. Then, the first point to consider is to discriminate between feasible actions and those which do not provide useful results. To illustrate this issue, the example represented in Figure 1 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. To achieve this, 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 and, according to them, along the x or y directions. Regarding the possible pushing actions, it can be seen that, for instance, pushing obstacle A from m_{A1} becomes an ineffective action because it does not clear the

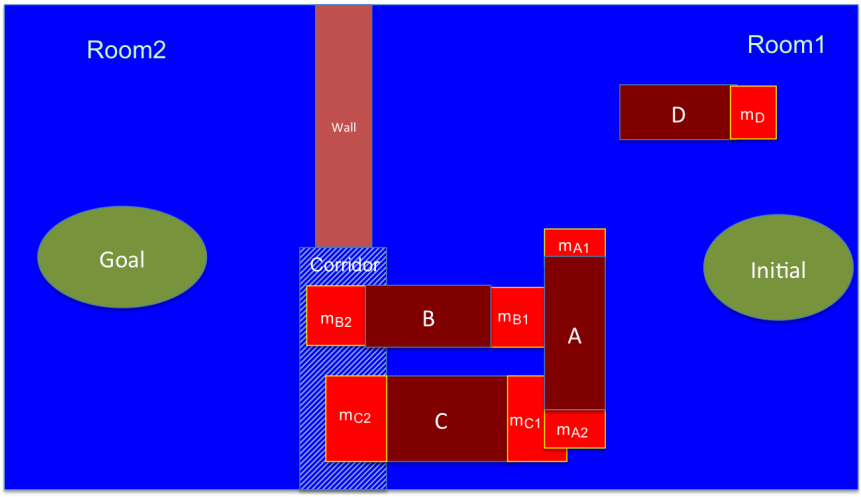

Fig. 1 A manipulation problem where a robot must find a feasible path to reach the goal region avoiding the fixed wall.

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.

To cope with this type of problem, the use of manipulation knowledge in the form of an ontology is proposed in order to represent all the manipulation problem components (e.g. manipulatable regions or constraints) and to facilitate the reasoning process. Also, the use of a physics-based engine is envisioned to reason about the feasibility of actions and its interleaving with the task planning based on GraphPlan can reduce the planning graph by pruning those actions that are inessential, ineffective, or infeasible, and the corresponding branches. In this way, the high-level planning module will maintain only valid actions and the search of the plan will be easier.

3 Manipulation Modeling Using Ontologies

An ontological knowledge-based management system can be integrated within task planning in order to improve its efficiency. The current proposal envisions the use of ontology-based knowledge in order to hand over to the task planning algorithm sufficient knowledge concerning the manipulation of objects, i.e., knowledge concerning the way an object can be manipulated. This may be determined by the object type and be given by its properties, e.g., *free manipulatable objects* can be pushed in any direction and from any of its sides, while *constraint-oriented manipulatable objects* can only be pushed in some directions and from some given sides (such as object D in Figure 1 that can only be pushed horizontally from the manipulatable region m_D).

3.1 Ontology Concept

Ontology deals with the concern regarding things existence. It has been recognized as an approach to expose explicit knowledge about the world in Artificial Intelligence with respect to characterizing concepts as well as relations. Ontologies are able to be stored in the Web Ontology Language (OWL) [8] being a family of knowledge representation on the basis of description logics. The intention of OWL is to access and share ontological knowledge among several systems through the world wide web. This knowledge is modeled over classes and individuals. Classes (declared by *owl:Class*) are dedicated to comprise those objects sharing analogous knowledge, while individuals (declared by *owl:NamedIndividual*) describe particular elements of classes. Correspondence between classes, between classes and individuals, as well as between individuals are performed with properties (declared by *owl:ObjectProperty*) in the OWL. OWL can be designed by one of the most popular ontology editors called *Protégé* [9] enabling knoweledge development. On the advantages of this editor is to visualize ontology as a form of graph to represent relations between knowledge in the database.

The management of an ontology-based knowledge described in the OWL and its later use for reasoning purposes can be done using the Knowrob tool [10]. This software tool is developed based on SWI Prolog and the Semantic Web library and enables fundamental predicates to access such knowledge using Prolog language, 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.

3.2 Components of the Manipulation Ontology

A manipulation ontology using OWL has been designed as shown in Figure 2. The classes *ManipulationWorld* and *ManipulationPlanning* with their subclasses represent the knowledge about the world and about high-level planning components, respectively. The OWL files are accessible at the following web adress: `https://sir.upc.edu/projects/ontologies/`.

The subclasses of the *ManipulationWorld* class are:

- *"Regions"*: Class that represents various types of regions. For instance, *ManipulatableRegion* belongs to an object and is the region where the robot should be located in order to apply forces to push it. *CriticalRegion* corresponds with regions which should be free from obstacles, e.g., the corridor depicted in Figure 1.
- *"ObjectsType"*: Class that collects information about the objects type.
- *"Path"*: Class that involves a number of abstract paths connecting two different regions (the actual path can be acquired by querying a motion planner). A property is considered for the individuals of the class that defines whether the regions which

Fig. 2 Taxonomy of knowledge-based manipulation

the path connects are critical or not. This property is updated after each push action. Related to Figure 1, the following individuals of this class are used in the solution.

- *P1, P2,* and *P3*: Paths that connect the initial region to the regions m_D, m_{A1}, and m_{A2}, respectively.
- *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 the room 2.
- *P11*: Path that connect the region m_{C1} to the goal region when the robot is in the room 2.

Likewise, the subclasses of the *ManipulationPlanning* class are:

– *"Predicates"*: Class that expresses a number of parameters with associated names devoted to describe preconditions and side effects of actions beside conditions of initial and goal states. The following predicates have been defined:
  - *HasAccess(Robot, Path)*: Returns true if the *Robot* is located at the initial region of the *Path*.
  - *At(Robot, Region, Path)*: Says whether the *Robot* has reached the *Region* through the *Path*.
  - *In(Object, Region)*: Returns true if the *Object* is at the *Region*.
– *"States"*: Class that contains the conditions about the initial and goal states of the problem.
– *"ActionProperties"*: Class used to bind actions with their needs and side effects. Two actions have been defined to solve manipulation problems as that of Figure 1: the *Move* action used to convey the robot to different regions and the *Push* action whose purpose is to clear critical regions.
  - *Move(Robot, Region, ThroughPath):*
    Precondition: *HasAccess(Robot, ThroughPath)*
    Add: *At(Robot, Region, ThroughPath)*
    Delete: _
  - *Push(Robot, Obj, ManipRegion, CriticalRegions, ToAccessPath, ThroughPath):*
    Precondition: *At(Robot, ManipRegion, ThroughPath)*
    Add: *HasAccess(Robot, ToAccessPath)*
    Delete: *In(Obj, CriticalRegions)*

4 Reasoning Process about Manipulation Actions

Two types of reasoning processes are proposed to evaluate manipulation actions. First, a high-level inference process determines whether an action is relevant or not. It is called essential reasoning and is performed based on the pre- and post-conditions of actions available through the Planning Graph generated by the task planning algorithm used (GraphPlan [11]). Second, a low-level reasoning based on a physics-based engine determines if an essential action can be executed or not, called feasibility reasoning, and if feasible whether the expected results are accomplished or not, called geometric reasoning. The integration of both layers is discussed in the next Section.

4.1 High-Level Reasoning

Essential reasoning process is performed in the Planning Graph generated by the GraphPlan algorithm. The Planning Graph is a layered graph with two types of levels: state-levels with sets of literals representing conditions, and action-levels with sets of possible actions. Edges connecting a state-level with the following action-level represent the pre-conditions of an action; edges connecting an action-level with the following state-level represent the post-conditions (or effects) of an action. Maintenance action is always included to retain literals for the subsequent level.

There are different constraints between actions and between literals. Two actions are constrained when:

- An effect of one action negates an effect of the other action called **"inconsistent effect"** constraint.
- An effect of one action deletes a precondition of the other action called the **"interference"** constraint.
- They have mutually exclusive preconditions called **"competing needs"** constraint.

Two literals, furthermore, are constrained if:

- One of them is the negation of the other one. It is called the **"inconsistent support"** constraint.

The construction phase of the GraphPlan procedure begins from the initial state and expands consecutively by interleaving state and action levels forming the Planning Graph, until the goal state conditions are met. Then, the search phase of GraphPlan uses a backward search to find the solution sequence. If this search fails because not all the planning constraints are satisfied, then the construction phase resumes.

The high-level reasoner can be applied during the construction phase to prune those actions that are not relevant for the task to be solved, i.e., those that are inessential. This is done 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. 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.

4.2 Low-Level Reasoning

The physics-based engine is embedded inside the low-level planning layer with the purpose of providing robust inference capability over actions. Physics-based motion planning is the more realistic approach for robot motion planning because it is able to evaluate interactions between rigid bodies and, therefore, can both be used for move actions to plan collision-free paths as well as for push actions where collisions with manipulatable objects occur.

Then, in order to determine whether an action is feasible or not, it is planned using a physics-based motion planner and the feasibility of the resulting path can then be evaluated as follows. Based on the control forces, applied duration, and distance covered, the dynamic costs of push and move actions, c_p and c_m respectively, are computed in terms of power consumed and action [7]:

$$c_p = \frac{n\mathbf{fd}}{\Delta t} \qquad c_m = \sum_{i}^{n} |\mathbf{f_i}| \Delta t_i \varrho_i \tag{1}$$

where, **f** is the applied force, Δt is the time duration, **d** represents the displacement covered in the result of **f**, and n is the number of times the force shall be applied to complete the push action. In case of c_m, ϱ represent the distance. If the planner is not able to find a path or the required power is above the robot specifications, then the action is considered infeasible.

For those actions that are feasible, their effects are analyzed by geometrically evaluating the position of the objects and regions, e.g., it is verified if after a push action a critical region is free from obstacles. Then, an action is called ineffective if its effects are not met. In case of finding infeasible or ineffective action, the same pruning process is performed to remove corresponding actions.

5 Efficient Task and Motion Planning

Task planning is devoted to determining the sequence of actions to perform a given task, while motion planning is devoted to determine the motions to execute a given action. The interleaving of both planning levels may result in a robust and computationally efficient solution. The proposal is sketched in Algorithm 1 that takes the initial and goal states, $\boldsymbol{s}_{\text{init}}$ and $\boldsymbol{s}_{\text{goal}}$ respectively, and the whole set of possible actions $\boldsymbol{A}$ as input 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 ineffective, then it is pruned. Figure 3 illustrates the constructed Planning Graph for the example represented in Figure 1. 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 ($\boldsymbol{s}_{\text{init}}$) [line 1] and the construction phase procedure is launched until a layer is found that satisfies the goal constraints [line 27]. 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 4] and then evaluated [Lines 5-23]. First, the effects e are obtained [line 6] to be forwarded to the sequence of reasoning process:

Algorithm 1. Efficient task and motion planning

Input: The initial and goal states s_{init}, s_{goal}. Set of possible actions $\boldsymbol{A}$. The maximum allowed iteration m

Output: The solution plan π

```
1: S_0 ← s_init
2: A_valid = ∅
3: for i ← 1 to m do
4:    A_i ← {a ∈ A | precond(a) ⊆ S_i-1}
5:    for each a ∈ A_i AND a ∉ A_valid do
6:       e ← effects(a)
7:       essentialReasoner(a,e)
8:       if a is inessential then
9:          pruneActions(a)
10:         continue
11:      end if
12:      feasibilityReasoner(a,e)
13:      if a is infeasible then
14:         pruneActions(a)
15:         continue
16:      end if
17:      geometryReasoner(a,e)
18:      if e is ineffective then
19:         pruneActions(a)
20:         continue
21:      end if
22:      A_valid ← a
23:   end for
24:   A_i ← maintenanceActions(S_i-1)
25:   S_i ← (l | l ∈ effects(A_i))
26:   checkMutexes(A_i, S_i, S_i-1)
27:   if goalFound(s_g, S_i) then
28:      S_g = S_i
29:      backtrack(s_g)
30:      if constraintSatisfaction(S_g, S_0) then
31:         π ← extractPlan(S_g, S_0)
32:         return π
33:      end if
34:   end if
35:   i ← i + 1
36: end for
37: return NULL
```

– *essentialReasoner*($\boldsymbol{a}$,e) checks whether an action is essential or not [lines 7-11], 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*($\boldsymbol{a}$,e) evaluates the feasibility of an action by determining whether the robot is capable or not to perform it [lines 12-16], e.g., the action *PushC* is infeasible with respect to the robot's capability since it is a very big and heavy object.

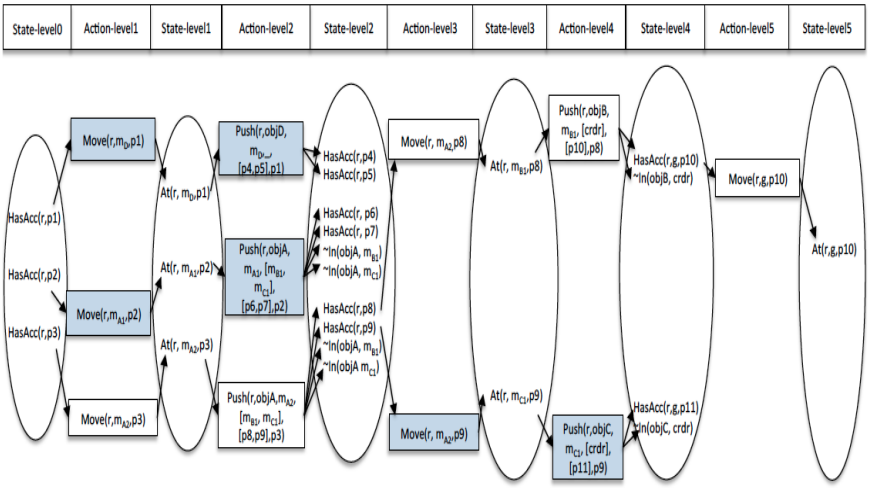

Fig. 3 The constructed Planning Graph. Pruned actions are specified in blue.

– *geometryReasoner*(***a***,*e*) checks whether the effects of an action are reached or not [lines 17-21], 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(***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 22], in order not to evaluate them again if the action further appears, and maintenance actions are later added [line 24]. Then the next state-level is created from the set $\boldsymbol{A}_{\text{i}}$ and the constraints between actions and literals is computes by function *checkMutexes*($\boldsymbol{A}_i$, $\boldsymbol{S}_i$, $\boldsymbol{S}_{i-1}$) 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 29-33].

6 Implementation and Simulation Results

The implementation of the proposed algorithm entails four phases: high-level planning, low-level planning, middleware, and executive. In the high-level planning, ontological knowledge concerning the manipulation world is represented using OWL and developed with *Protégé*. Knowrob software is applied to manage such knowledge and use it for the task planning algorithm. In the low-level planning, The Kautham

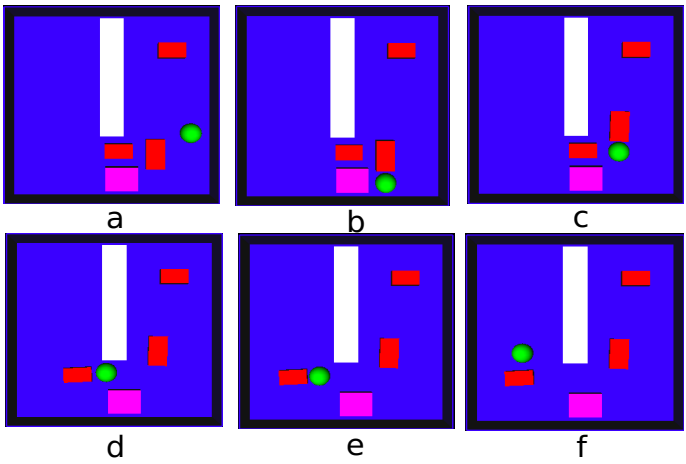

Fig. 4 Simulation results of the final plan

Project [12], an open-source tool for motion planing environment, is employed that uses the *Open Motion Planning Library (OMPL)* [13] as the core set of planning algorithms. OMPL allows planning under geometric constraints as well as under differential constraints, including those that required dynamic simulations (OMPL uses the *Open Dynamic Engine* for the dynamic simulation). Transferring information between both planning levels is performed by the middleware phase using ROS-based communications [14]. The high-level module is encapsulated as a ROS client and the low-level module works as a ROS service in order to evaluate manipulation actions. Finally, the plan is forwarded to an executive module that is the responsible of executing the whole task.

Simulation results corresponding to the scenario represented in Figure 1 are shown in Figure 4 as a sequence of snapshots. The SyCLoP-RRT [15] is used as kinodynamic motion planner for physics-based planning because recent benchmarking study of the kinodynamic motion planners for physics based planning [16] shows that SyCLoP-RRT computes the power optimal and smooth solution. The average planning time is about 35 seconds for the executive plan.

7 Conclusion and Future Work

The current paper presents a simultaneous task and motion planner. Task planning is done with the GraphPlan algorithm, that has been modified to make it 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 ineffective. The proposed algorithm has been implemented and illustrated through a manipulation problem. As a future work, to tackle several feasible plans, heuristics-based task planning is going to be considered that takes into account the costs of actions to find the most feasible actions while planning the whole task.

References

1. Kaelbling, L.P., Lozano-Pérez, T.: Hierarchical task and motion planning in the now. In: International Conference on Robotics and Automation (ICRA), pp. 1470–1477. IEEE (2011)
2. Shafii, N., Passos, L.S., Reis, L.P., Lau, N.: Humanoid soccer robot motion planning using graphplan. In: Proceedings of the 11th International Conference on Mobile Robots and Competitions, pp. 84–89 (2011)
3. Di Marco, D., Levi, P., Janssen, R., van de Molengraft, R., Perzylo, A.: A deliberation layer for instantiating robot execution plans from abstract task descriptions. In: Proceedings of the International Conference on Automated Planning and Scheduling: Workshop on Planning and Robotics (PlanRob) (2013)
4. Muhayyudin, Akbari, A., Rosell, J.: Ontological physics-based motion planning for manipulation. In: Proceedings of the International Conference on Emerging Technologies and Factory Automation (ETFA). IEEE (2015)
5. Mosenlechner, L., Beetz, M.: Fast temporal projection using accurate physics-based geometric reasoning. In: International Conference on Robotics and Automation (ICRA), pp. 1821–1827. IEEE (2013)
6. Şucan, I.A., Kavraki, L.E.: Mobile manipulation: encoding motion planning options using task motion multigraphs. In: International Conference on Robotics and Automation (ICRA), pp. 5492–5498. IEEE (2011)
7. Akbari, A., Muhayyudin, Rosell, J.: Task and motion planning using physics-based reasoning. In: Proceedings of the IEEE International Conference on Emerging Technologies and Factory Automation (ETFA) (2015)
8. McGuinness, D., Van Harmelen, F., et al.: Owl web ontology language overview. W3C recommendation **10**(10) (2004)
9. Stanford 2007: Protégé (2007). http://protege.stanford.edu/
10. Tenorth, M., Beetz, M.: Knowrob knowledge processing for autonomous personal robots. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 4261–4266 (2009)
11. Blum, A.L., Furst, M.L.: Fast planning through planning graph analysis. Artificial Intelligence **90**(1), 281–300 (1997)
12. Rosell, J., Pérez, A., Aliakbar, A., Muhayyuddin, Palomo, L., García, N.: The kautham project: a teaching and research tool for robot motion planning. In: Emerging Technology and Factory Automation (ETFA), pp. 1–8. IEEE (2014)
13. Şucan, I.A., Moll, M., Kavraki, L.E.: The Open Motion Planning Library. Robotics & Automation Magazine **19**(4), 72–82 (2012)
14. Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A.Y.: Ros: an open-source robot operating system. In: ICRA Workshop on Open Source Software, vol. 3, p. 5 (2009)
15. Plaku, E., Kavraki, L., Vardi, M.: Motion planning with dynamics by a synergistic combination of layers of planning. IEEE Transactions on Robotics **26**(3), 469–482 (2010)
16. Muhayyuddin, Akbari, A., Rosell, J.: Physics-based motion planning: evaluation criteria and benchmarking. In: ROBOT2015: Second Iberian Robotics Conference. Springer (2015)