Task and Motion Planning Using Physics-based Reasoning

Aliakbar Akbari, Muhayyuddin and Jan Rosell
Institute of Industrial and Control Engineering (IOC)
Universitat Politcnica de Catalunya (UPC) – Barcelona Tech
Barcelona, Spain, jan.rosell@upc.edu

***Abstract*—For everyday manipulation tasks, the combination of task and motion planning is required regarding the need of providing the set of possible subtasks which have to be done and how to perform them. Since many alternative plans may exist, the determination of their feasibility and the identification of the best one is a great challenge in robotics. To address this, this paper proposes: *a*) a version of GraphPlan (one of the best current approaches to task planning) that has been modified to use ontological knowledge and to allow the retrieval of all possible plans; and *b*) a physics-based reasoning process that determines the feasibility of the resulting plans and an associated cost that allows to select the best one among them. The proposed framework has been implemented and is illustrated through an example.**

***Index Terms*—Task planning, Physics-based motion planning, Reasoning, Manipulation.**

I. Introduction

Autonomous robots, like mobile manipulators, are currently setting new challenges for robot manipulation tasks in human-like environments. The complete performance of a manipulation task consists in the execution of a sequence of actions, which requires planning at task and motion levels. Both task and motion planning are, therefore, of great importance and play a considerable role in finding feasible solutions to manipulation problems, and the sharing and interaction of information through planning levels is of crucial importance for providing efficient solutions. The combination of task and motion planning has been centered mainly around the task planning techniques of Hierarchical Task Networks (HTN) and GraphPlan, reviewed in Section III, e.g. [1][2][3][4].

On the other hand, manipulation involves the interaction between objects and, in this scope, physics-based processes have a valuable significance in order to: a) allow motion planning strategies to incorporate (using dynamic simulation) possible interactions between rigid bodies; b) reason about the feasibility of the actions provided by the task planner. In this line, for instance, a physics-based temporal projection for manipulation tasks has been proposed to provide reasoning on stability, visibility, and reachability of robot motions [5], or a manipulation planning framework has been proposed where physics-based motion planning is enhanced with ontological knowledge reasoning on motion constraints [6].

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.

From what has been stated, the combination of task planning with physics-based motion planning and reasoning arises as an interesting issue. One of the possible strategies is to use physics-based motion planning and reasoning to evaluate the feasibility and the cost of a set of plans possibly provided by the task planner. A proposal in this line, but without considering physics-based issues, was the Task Motion Multigraph (TMM [7]), that runs in parallel the search of solution paths corresponding to alternative sequences of actions given by a task planner.

With this in mind, the current paper presents a framework to combine physics-based reasoning with task and motion planning to find the best feasible sequence of actions to solve a given task. The task planner employs a modified version of the GraphPlan algorithm that uses ontological knowledge and allows the retrieval of all possible plans. These plans are forwarded to the physics-based reasoning in order to determine their feasibility by measuring the cost of the corresponding sequence of actions. The proposed framework has been implemented and is illustrated with a simple manipulation problem.

II. Problem Statement and Solution Overview

Consider a mobile robot that can perform two types of actions: to move around freely and to push manipulatable obstacles. Then, the problem to be tackled is to find the best feasible sequence of actions to bring the robot, among fixed and manipulatable obstacles, from an initial region towards a goal one. Collisions with fixed obstacles are not allowed, while if necessary, manipulatable obstacles can be pushed away. As an example, Fig. 1 shows a two-room scenario with a wall as a fixed obstacle, three manipulatable obstacles (A B and C, with associated regions from where they can be pushed) and an initial and a goal region.

The proposed solution is based on a two-step procedure. The first one, described in Section III, finds all possible sequences of high-level actions; the second one, described in Section IV, computes how to perform these actions and evaluates the feasibility based on a physics-based simulation. To perform the first step, the GraphPlan task planning algorithm has been modified to retrieve all possible plans and to use ontological

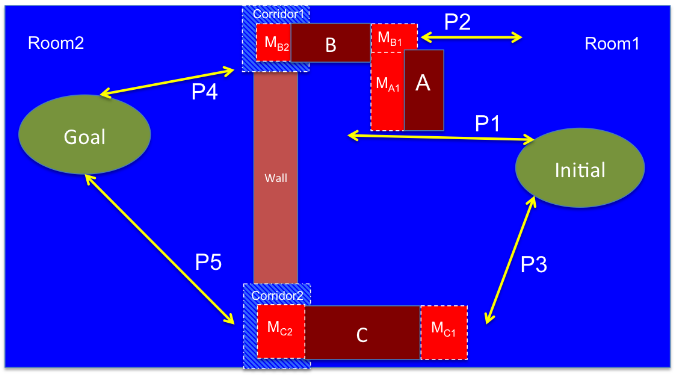

Fig. 1. A manipulation problem where a robot must find the best path from an initial region to a goal one avoiding the fixed wall in the middle and pushing the manipulatable obstacles A, B and C, if necessary, to clear the corridors (dashed regions).

knowledge. To perform the second step, a physics-based motion planner has been used and a cost for each type of action has been defined. Implementation details are presented in Section V where the results corresponding to the problem of Fig. 1 are discussed.

The present paper is a first step towards a more comprehensive framework that: 1) considers a wider set of actions, like pick and place; 2) tightly combines task and motion planning levels by interleaving the physics-based evaluation within the search of the sequence of actions.

III. Task Planning

Robot task planning is devoted to finding an ordered sequence of actions (i.e. organized in terms of constraint satisfaction between them), that allow a robot to perform a given task. There has been a significant amount of study in task planning, as reported in [8], comprising a variety of different automated task planning strategies. Some of the best ones are the Hierarchical Task Network (HTN) [9] and the GraphPlan [10]. The former approach establishes a network to assign possible preconditions of actions (which can be either primitive or compound). To find a plan, tasks are decomposed and grow in a search space until primitives actions satisfying preconditions are met. The GraphPlan technique, on the other hand, constructs a search space of plans with the form of a graph (detailed in the next subsection) and establishes a sequence of levels incrementally expanded through the search space. The main advantages of GraphPlan-based planners is the ability to analyze combinations of action to satisfy the task, the ability to evaluate multiple ways to reach the goal, and the ability to run parallel actions at a time. This paper proposes a variant of the GraphPlan planning strategy integrated with ontological knowledge. Next subsections first present an overview of the GraphPlan and of the concept of ontologies, and then explain how manipulation knowledge is coded using an ontology and how the standard GraphPlan algorithm is modified to allow the retrieval of all possible plans.

A. GraphPlan Overview

The GraphPlan constructs a planning structure, called Planning Graph, where state-levels (representing sets of literals) and action-levels (representing possible actions applied on the previous state-level) are interleaved. Each action-level also contains maintenance actions to maintain literals unchanged for the next level. Edges in the graph join literals and actions of consecutive levels. Different constraints between actions and between literals may exist. Two actions are constrained when:

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

Two literals, moreover, are constrained if:

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

The search space procedure extends consecutively the levels in the graph until all the goal conditions appear in the last created state-level. When this happens, a solution may be possibly found by a backward search. If it fails because not all the constraints are satisfied, then the extension procedure resumes. A Planning Graph with five state levels (corresponding to the problem shown in Fig. 1) is drawn in Fig. 2 where, for clarity, the constraints are not shown nor the edges connecting the levels. the graph will further be commented in Section III-D.

B. Concept of Ontology

An ontology classifies knowledge within a particular domain and enables a flexible access to it by describing things with associated relations. An ontology can be divided into components: classes, individuals, properties, attributes, and axioms. Classes (also called concepts) specify collections or types of different objects that share common properties. Individuals (also referred to as instances) demonstrate specific elements of classes. Properties express how classes and individuals are related to one another. Attributes define unique properties, features, and particular characteristics of objects. Axioms set constraints on the values of classes and individuals. Ontologies are encoded using the Web Ontology Language (OWL) [11]. The purpose of OWL is to collect and classify ontology-based knowledge on a world-wide accessible database represented by an XML-based file format aiming to share such knowledge over multiple systems or devices. Ontologies can be created using the *Protégé* editor [12]. *Protégé* is an open source platform that is able to provide a flexible ontology editor in order to facilitate design of knowledge-based applications.

To access knowledge encoded with the OWL, a knowledge processing framework for robotic systems called Knowrob [13] can be used. Knowrob is a potent reasoning tool that works over ontological knowledge. It has been implemented based on the Semantic Web library and SWI Prolog, enabling the access of Prolog predicates to obtain the knowledge accumulated within the OWL. Thus, Knowrob enables a flexible access to ontological knowledge to facilitate the inference process.

C. Ontology-based Task Planning for Manipulation

The present work codes the knowledge of a manipulation task by defining an ontology with OWL. This ontology will contain the information related to the objects (like for instance their dynamical properties or the parts from where they can be pushed), or to some regions of the workspace (like for instance the initial and goal regions, the critical regions that cannot be occluded, etc.), and will be accessed by the task and motion planners. The ontology is structured with the following classes:

1) Class *"States"* represents the conditions that are satisfies by a state of the world. It includes the subclasses *InitialState* and *GoalState* related to the problem to be solved.
2) Class *"Path"* defines an abstract path connecting two regions (the actual path can be obtained by querying a motion planner) such as *P1* and *P3* in Fig. 1.
3) Class *"Regions"* defines different types of regions: *ManipulatableRegion, CriticalRegion,* and *GoalRegion*. *ManipulatableRegion* associated to an object is the region where the robot should be located in order to apply forces to push it. *CriticalRegion* represents regions which should be free from obstacles, e.g. corridors shown in Fig. 1 or the manipulatable region of an object that should be pushed away. *GoalRegion* represents the region where the final position of the robot should be.
4) Class *"Predicates"* represents the predicates required to set the properties of the states and actions. Three predicates have been defined as subclasses:
   - *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*.
5) Class *"ActionProperties"* defines actions, action preconditions (what should be satisfied before executing an action) and action effects (what should be added/deleted to the world state after executing an action). Two actions, *Move* and *Push*, have been defined as subclasses of class *"ActionProperties"*, as well as their preconditions and postconditions. The *Move* action is used to transfer the robot to different regions (which may include the goal or manipulatable regions) through a path obtained by a motion planner. The *Push* action is applied to move objects to clear the way and facilitates the connection between paths.
   - *Move(Robot, Region, ThroughPath):*
     Precondition: *HasAccess(Robot, ThroughPath)*
     Add: *At(Robot, Region, ThroughPath)*
     Delete: _
   - *Push(Robot, Obj, ManipRegion, CriticalRegions, ToAccessPath, ThroughPath):*
     Precondition: *In(Obj, CriticalRegions), At(Robot, ManipRegion, ThroughPath)*
     Add: *HasAccess(Robot, ToAccessPath)*
     Delete: *In(Obj, CriticalRegions)*
6) Class *"ObjectProperties"* sets information about the type of objects (and the robot) and their locations in the world.

D. The GraphPlan variant

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

As an example, the problem in Fig. 1 shows that the initial region has access to two alternative paths, *P1* and *P3* (i.e. the initial action of a plan may be to move the robot towards object A following path *P1* or alternatively moving towards object C following path *P3*). Therefore these paths become possible conditions for the initial state:

I =[HasAccess(robot, p1), HasAccess(robot,p3), In(objA, manipulatableRegionM_{B1}), In(objB, corridor1), In(objC, corridor2)]

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

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

For the example in Fig. 1, the resulting Planning Graph generated by this variant of GraphPlan is shown in Fig. 2. At each state level, predicates are labeled with numbers, and at each action level actions are labeled with the predicates they connect (not counting the maintenance action that applies to all the predicates). The search space provides five state levels. From the last state level, backtracking is done starting at two possible goal conditions, *At(robot, goalregion, p4)* and *At(robot, goalregion, p5)*. The states encountered during backtracking in which the robot plays a role have been highlighted in Fig. 2, and the corresponding plans are depicted in Fig. 3.

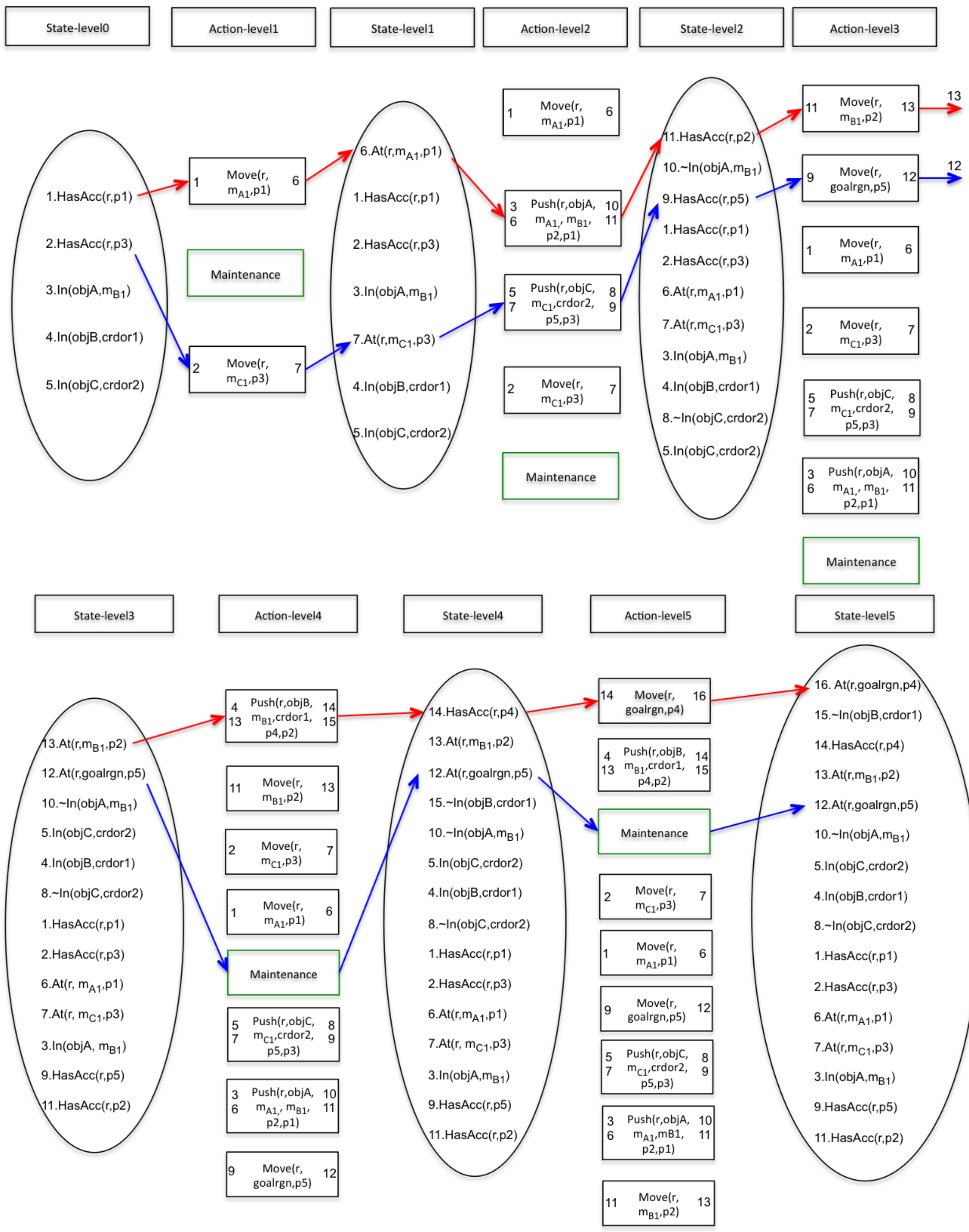

Fig. 2. The search space of GraphPlan

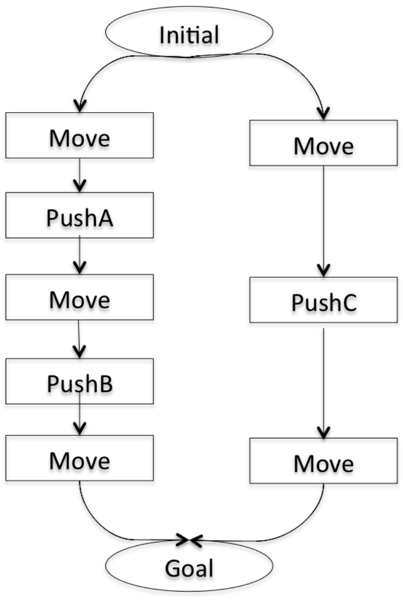

Fig. 3. Plan A (left) and plan B (right) resulting from the modified GraphPlan algorithm.

IV. Physics-Based Reasoning

A. Physics-Based Motion Planning

The basic purpose of motion planning is the computation of an appropriate trajectory from a start to a goal configuration in the configuration space, while satisfying a given set of constraints. Sampling-based motion planning algorithms, like the *Rapidly exploring Random Trees* (RRTs) or the *Kinodynamic Planning by Interior-Exterior Cell Exploration* (KPIECE), give very efficient results and are well suited for kinodynamic motion planning. They can easily consider systems with differential constraints and cope with nonlinear dynamics [14], [15].

Typically, motion planning algorithms focus on computing collision-free trajectories, but some variants, known as physics-based, have been designed to also consider the purposeful manipulation of the static objects in the environment, e.g. [16]. These algorithms modify the state propagator of a kinodynamic motion planer to consider the result of the interaction between rigid bodies using a rigid body dynamic simulators such as the Open Dynamic Engine (ODE) [17] or Bullet [18].

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

B. Cost functions

The physics-based reasoning about a high-level plan (composed of a sequence of move and push actions) involves the dynamical properties of the robot and of the environment, such as masses of the objects, required manipulation forces, friction, etc. The feasibility of a plan will be set according to its cost that will be computed according to the following *cost functions*:

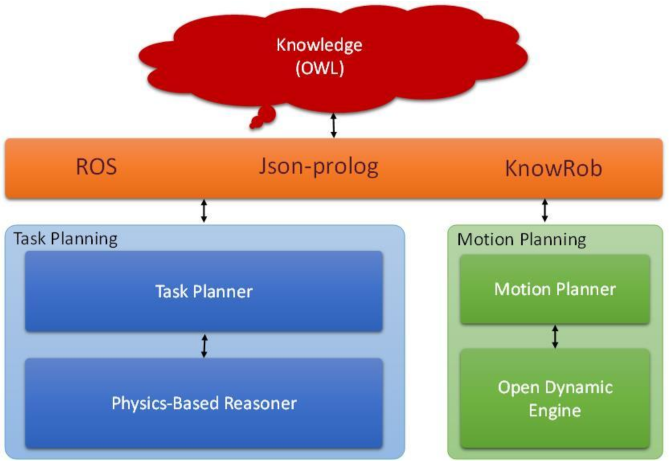

Fig. 4. Implementation framework for task and motion planning using physics-based reasoning

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

$$c_p = \frac{n\mathbf{f}\mathbf{d}}{\Delta t}, \tag{1}$$

  where $\mathbf{f}$ represents the force applied by the robot for Δt duration, $\mathbf{d}$ is the corresponding displacement covered by the object, and n is the number of times the force is applied in order to complete the push action and clear the region. If the object is too heavy to be pushed by the robot, then this cost will be set to infinity.
- *Move cost:* It is the total amount of "action" (i.e. the dynamical attribute of a physical system that considers the history of moves and that has units of *Newton-meter-second*):

$$c_m = \sum_{i}^{n} |\mathbf{f_i}|\Delta t_i \varrho_i \tag{2}$$

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

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

$$\begin{aligned} C_p^j &= \sum_{k=1}^{N_{p_j}} c_{p_k}^j \\ C_m^j &= \sum_{k=1}^{N_{m_j}} c_{m_k}^j \end{aligned} \tag{3}$$

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

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

TABLE I
EVALUATION OF COST PLANS.

follows:

$$\alpha_j = \mu \frac{C_p^j}{\max_{\forall k \in [1,N]} C_p^k} + (1-\mu) \frac{C_m^j}{\max_{\forall k \in [1,N]} C_m^k} \quad (4)$$

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

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

V. Implementation Framework and simulation results

The proposed framework for task and motion planning, shown in Fig. 4, consists of a knowledge module, a task planing module and a motion planning module. All them communicate with each other through a ROS-based communication layer:

- The knowledge is coded in the form of an OWL ontology and it consists of: a) knowledge about the world, i.e. information on the type of objects (manipulatable or fixed) and if manipulatable, on the regions from where they can be pushed (manipulatable regions), and their poses (position and orientation); b) physical properties (objects masses, friction coefficients, minimum manipulation forces, etc.).
- The task planning module consists of two sub-modules: the task planner (that uses the variant of GraphPlan explained in subsection III-D) and the physics-based reasoner (that evaluates the plans as detailed in IV-B in order to select the best one).
- The motion planning module is used to determine the effect of each action in the dynamic world and to execute the plan. The implementation of this module is performed using The Kautham Project [19], that is a motion planning tool based on the Open Motion Planning Library (OMPL [20]) which provides the implementation of many sampling-based planners like RRT and KPIECE. The dynamic simulations are performed using ODE.
- The communication layer shares information between different layers. This layer involves ROS [21] to provide the basic communication protocols, and the Knowrob and the Json-prolog library (provided by Knowrob) to enable the access to the ontological knowledge and Prolog predicates through the ROS communication protocols.

In order to illustrate the proposal, the problem set in Fig. 1 has been modelled using The Kautham Project. The scenario consists of a robot (green sphere), three manipulatable bodies (red and yellow blocks), and a fixed body (white block). As discussed before, the modified version of GraphPlan retrieves the two possible plans that were shown in Fig. 3. The KPIECE motion planning algorithm (with the physics-based state propagator based on ODE) has been used, and some snapshots of the executions of both plans are illustrated in Fig. 5 and Fig. 6. Table I shows the cost of each action and the *Task-feasibility* parameter of each plan using $\mu = 0.5$. The total move and push costs of plan A are lower than those of plan B and result in a lower Task-feasibility parameter. Plan A is, therefore, the selected one. Note that plan B has less actions and would have been the unique plan found by the standard GraphPlan algorithm.

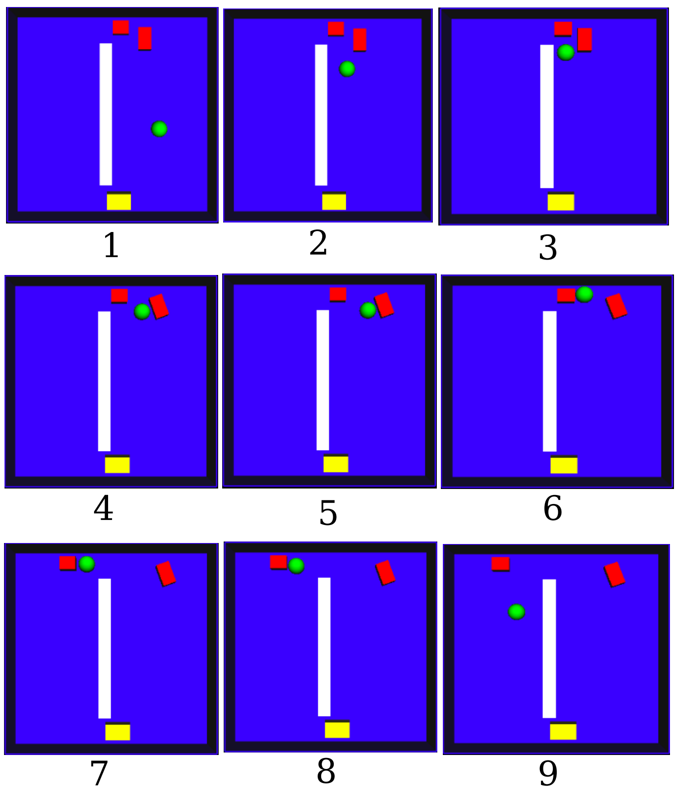

Fig. 5. Simulation results of the execution of plan A.

VI. Conclusion

This paper has proposed a framework to take into account physics-based reasoning in manipulation planning. A modified version of the GraphPlan algorithm is first proposed in order to provide all possible plans and, afterwards, a physics-based reasoning process is applied to determine the feasibility of each plan in terms of the action costs. The proposal has been implemented and illustrated with a simple example.

The present proposal considers a mobile robot with move and push action capabilities. The extension to manipulators with pick and place capabilities is under development. Also, the substitution of the two-step procedure by a tight combination of the task and motion planning levels is under study,

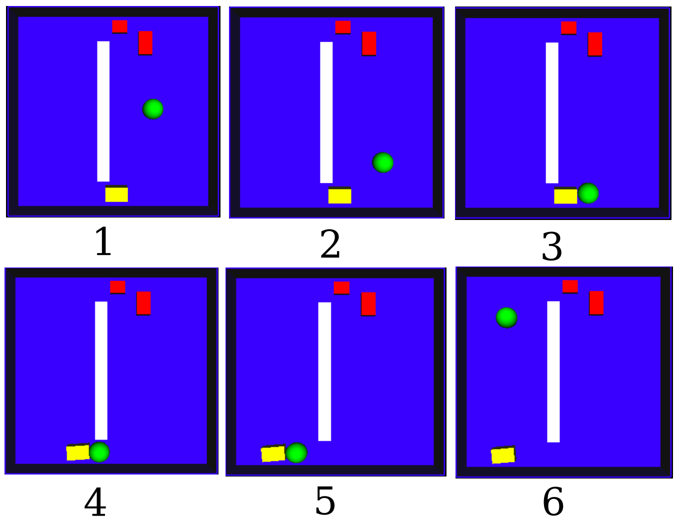

Fig. 6. Simulation results of the execution of plan B.

i.e. the use of the physics-based evaluation to condition and accelerate the search of the sequence of actions.

References

[1] J. Wolfe, B. Marthi, and S. J. Russell, “Combined task and motion planning for mobile manipulation.” in *ICAPS*, 2010, pp. 254–258.

[2] L. P. Kaelbling and T. Lozano-Pérez, “Hierarchical task and motion planning in the now,” in *International Conference on Robotics and Automation (ICRA)*. IEEE, 2011, pp. 1470–1477.

[3] N. Shafii, L. S. Passos, L. P. Reis, and N. Lau, “Humanoid soccer robot motion planning using graphplan,” in *Proceedings of the 11th International Conference on Mobile Robots and Competitions*, 2011, pp. 84–89.

[4] M. Klusch, A. Gerber, and M. Schmidt, “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*, 2005.

[5] L. Mosenlechner and M. Beetz, “Fast temporal projection using accurate physics-based geometric reasoning,” in *International Conference on Robotics and Automation (ICRA)*. IEEE, 2013, pp. 1821–1827.

[6] Muhayyudin, A. Akbari, and J. Rosell, “Ontological physics-based motion planning for manipulation,” in *Proceedings of the International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2015.

[7] I. A. Şucan and L. E. Kavraki, “Mobile manipulation: Encoding motion planning options using task motion multigraphs,” in *International Conference on Robotics and Automation (ICRA)*. IEEE, 2011, pp. 5492–5498.

[8] M. Ghallab, D. Nau, and P. Traverso, *Automated planning: theory & practice*. Elsevier, 2004.

[9] K. Erol, J. A. Hendler, and D. S. Nau, “Semantics for hierarchical task-network planning,” DTIC Document, Tech. Rep., 1995.

[10] A. L. Blum and M. L. Furst, “Fast planning through planning graph analysis,” *Artificial intelligence*, vol. 90, no. 1, pp. 281–300, 1997.

[11] D. McGuinness, F. Van Harmelen *et al.*, “Owl web ontology language overview,” *W3C recommendation*, vol. 10, no. 10, p. 2004, 2004.

[12] Stanford2007, “Protégé,” http://protege.stanford.edu/, 2007.

[13] M. Tenorth and M. Beetz, “Knowrob knowledge processing for autonomous personal robots,” in *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2009, pp. 4261–4266.

[14] E. Frazzoli, M. A. Dahleh, and E. Feron, “Real-time motion planning for agile autonomous vehicles,” *Journal of Guidance, Control, and Dynamics*, vol. 25, pp. 116–129, 2000.

[15] I. A. Şucan and L. E. Kavraki, “A sampling-based tree planner for systems with complex dynamics,” *Transactions on Robotics*, vol. 28, no. 1, pp. 116–131, 2012.

[16] S. Zickler and M. Veloso, “Efficient physics-based planning: sampling search via non-deterministic tactics and skills,” in *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, 2009, pp. 27–33.

[17] S. Russell, “Open Dynamic Engine,” http://www.ode.org/, 2007.

[18] C. Erwin, “Bullet physics library, http://bulletphysics.org,” 2013. [Online]. Available: http://bulletphysics.org

[19] J. Rosell, A. Pérez, A. Aliakbar, Muhayyuddin, L. Palomo, and N. García, “The kautham project: A teaching and research tool for robot motion planning,” in *Emerging Technology and Factory Automation (ETFA)*. IEEE, September 2014, pp. 1–8.

[20] I. A. Şucan, M. Moll, and L. E. Kavraki, “The Open Motion Planning Library,” *Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, December 2012, http://ompl.kavrakilab.org.

[21] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, “Ros: an open-source robot operating system,” in *ICRA workshop on open source software*, vol. 3, no. 3.2, 2009, p. 5.