Flexibly configuring task and motion planning problems for mobile manipulators*

Siddhant Saoji
Department of Mechanical Engineering
Indian Institute of Technology
Jodhpur, India
saoji.1@iitj.ac.in

Jan Rosell
Institute of Industrial and Control Engineering
Universitat Politcnica de Catalunya
Barcelona, Spain
jan.rosell@upc.edu
ORCID: 0000-0003-4854-2370

***Abstract*—Robotic manipulation requires the planning at a symbolic level (task planning) and at a geometric level (motion planing). This paper presents a planning framework for both levels of planning that includes an easy way to configure their interconnection. Motion planning is done using The Kautham Project, which is equipped with the Open Motion Planning Library suite of sampling-based motion planners, and task planning is done using the Fast Forward task planner. Both planning levels can be accessed through Robotic Operating System interfaces using services. A client program then uses these task and motion planning services and an XML configuration file that defines the linkage between symbolic actions and geometric values, to compute the sequence of feasible robot motions that allow to successfully execute a manipulation task. An illustrative example using the Tiago mobile manipulator in a kitchen environment is presented where the flexibility in configuring different instances of manipulation tasks is shown.**

***Index Terms*—Robotic manipulation, task planning, motion planning.**

I. Introduction

Mobile manipulators acting as robot co-workers at the factory floor or as robot helpers at home may face different problems at the perception level (to sense and interpret the current state of the environment), at the action level (to successfully navigate and execute manipulation motions despite the uncertainty) and at the planning level (to determine the sequence of actions to be done and the feasible motions to perform them). Also, some other advanced manipulation tasks would require reasoning and interaction with human operators. To implement solutions to these problems and to integrate them into a robotic system is a difficult task that can be alleviated by the use of the Robotic Operation System (ROS, [1]).

Regarding planning, on the one hand there is the problem to find the sequence of collision-free motions that allow to safely move the robot from one configuration to another. For many degrees-of-freedom robots, like the mobile manipulators, sampling-based methods are usually used, like the Probabilistic Roadmaps (PRM, [2]) or the Rapidly-exploring Random Trees (RRT, [3]) and their variants. By taking samples of the configuration space, these methods capture the connectivity of the free regions of the configuration space with either roadmaps or tree-like structures.Very efficient implementations of these families of motion planning algorithms is available in the Open Motion Planning library (OMPL, [4]).

This work was partially supported by the Spanish Government through the project DPI2016-80077-R

On the other hand, there is the need to plan at task level, i.e. the robot has to be able to determine which is the sequence of actions to be done to perform a given task. Many problems can be tackled with classical planning approaches, that assume known initial values of variables, deterministic actions, and a set of goals defined over the variables [5]. These can be well modeled using the Planning Domain Definition Language (PDDL, [6]), which is an action-centered language that use pre- and post-conditions to describe the applicability of actions (atoms describing the state satisfy the pre-conditions) and effects (positive/negative effects add/delete atoms to the current state). Planning tasks specified in PDDL are separated into two files, a domain file for predicates and actions, and a problem file for objects, initial state and goal specification.

is Both planning levels are interconnected, starting with the linkage between the symbolic variables used at task level and the geometric values required for motion level. This grounding is not unique, for instance there may be different configurations of the arm of the robot that allow to grasp a given object, and that may correspond to a pick action that can be executed when the robot and the object are located at a given region labeled e.g. with a given symbolic variable A. The feasibility of a sequence of actions obtained from a task plan may depend on the grounding used, and if no feasible solutions exist, the information obtained at the geometric level may be used to guide the search of a new sequence [7], [8]. Therefore, task and motion planning (TAMP) is a difficult issue because interactions between planning levels have to be carefully handled.

This paper presents a simple way to easily link both planning levels. First, Sections II and III briefly present, respectively, the motion planning and task planning tools implemented as ROS servers. Then, Section IV presents the proposal to interface both levels, and Section V how to use it. Finally, the proposal is illustrated in section VI and discussed in SectionVII, where some future works are pointed out.

II. Motion planning server

As a motion planning server we will use *The Kautham Project* (sir.upc.edu/projects/kautham), a C++ based open-source software for teaching and research in robot motion planning [9], that fits the needs of mobile manipulation problems as those considered here. This tool, whose main core of planners is provided by OMPL, allows to cope with problems with one or several robots (generally considered as kinematic trees with a mobile base), and to easily configure the set of controls that shall move the degrees of freedom of the robot (with the possibility to define coupling between them).

Motion planning problems are modeled using an XML file with information on the models of the robot(s) and obstacles (using URDF and different geometric formats like .wrl, .stl or .dae), the controls (name of a separate XML file where they are defined) and the planner (type, parameters and query to be solved). In the examples used here, the RRT-connect planner [10] will be used since it is one of the more efficient single-query planners (which is the type of planner that fits the manipulation domain where the locations of the objects in the scene keep changing).

Kautham provides console, GUI and ROS interfaces. Some of the utilities offered by Kautham as ROS services are:

- *OpenProblem / CloseProblem*
- *SetPlannerByName / SetQuery / GetPath*
- *SetRobotsConfig / GetObstaclePos / SetObstaclePos*
- *AttachObstacle2RobotLink / DetachObstacle*
- *SetRobControls*

The Kautham ROS interface allow to easily integrate Kautham in complex robotic systems requiring planning at different levels. The *SetRobControls* service allows to change the control file that defines which is the set of degrees of freedom that are controlled, e.g. those moving the base or those moving the arm.

III. Task planning server

One of the most efficient classic task planning approaches among those that search in the state space is the Fast Forward (FF, [11]), which performs a heuristic search. It has two main components. One is the search module that is responsible to select the more promising successor state using the heuristic values and the Enforced Hill-Climbing search method. The other module computes these heuristic values as the estimated number of actions needed to reach the goal, computed as the length of a relaxed plan obtained using the *Relaxed Graphplan*, which is a graph that interleaves state-levels (involving a number of literals) and action-levels (representing a set of actions whose negative effects will be ignored).

The implementation of the FF method is open-source (fai.cs.uni-saarland.de/hoffmann/ff.html). We have wrapped it as a ROS service, so that using the PDDL domain and problem files as request, it returns the sequence of actions as a vector of strings.

```
<?xml version="1.0"?>
<Config>
  <Problemfiles>
        <pddldomain name="PDDL_domain_file.xml" />
        <pddlproblem name="PDDL_problem_file.xml" />
        <kautham name="kautham_problem_file.xml" />
  </Problemfiles>
  <States>
    <Initial>
      <Object name="object_id_1"> x y z qx qy qz qw </Object>
      <Object name="object_id_2"> x y z qx qy qz qw </Object>
        ……..
      <Object name="object_id_n"> x y z qx qy qz qw </Object>
      <Robot name="obot_id_1" controlfile="control_file_1"> c11  c12 </Robot>
      <Robot name="obot_id_2" controlfile="control_file_2"> c11  c12 </Robot>
        ……..
      <Robot name="obot_id_n" controlfile="control_file_n"> c11  c12 </Robot>
    </Initial>
  </States>
  <Actions>
    <Pick robot="PDDL_robot" object="PDDL_object" region="PDDL_region">
        <Obj> object_id </Obj>
        <Rob> robot_id </Rob>
        <Link> link_id </Link>
        <Cont>  control_file_manip </Cont>
        < Regioncontrols > c11  c12</Regioncontrols>
        < Graspconf grasp="grasp_id_1" > c11  c12</Graspconf>
        < Graspconf grasp="grasp_id_2" > c21  c21</Graspconf>
        ……..
        < Graspconf grasp="grasp_name_n" > cn1  cn1</Graspconf>
        <Pose> x y z qx qy qz qw <Pose>
    </Pick>
    <Place robot="PDDL_robot" object="PDDL_object" region="PDDL_region">
        <Obj> object_id </Obj>
        <Rob> robot_id </Rob>
        <Cont>  control_file_manip </Cont>
        < Regioncontrols > c11  c12</Regioncontrols>
        < Graspconf grasp="grasp_id_1" > p11  p12</Graspconf>
        < Graspconf grasp="grasp_id_2" > p21  p21</Graspconf>
        ……..
        < Graspconf grasp="grasp_id_n" > pn1  pn1</Graspconf>
        <Pose> x y z qx qy qz qw <Pose>
        <Poseregion> xmin xmax ymin ymaz z qx qy qz qw <Poseregion>
    </Place>
    <Move robot="PDDL_robot" region="PDDL_region_from" region="PDDL_region_to">
        <Rob> robot_id </Rob>
        <Cont>  control_file_move </Cont>
        <Conf > c11  c12</Conf>
        <Conf > c21  c21</Conf>
    </Move>
  </Actions>
</Config>
```

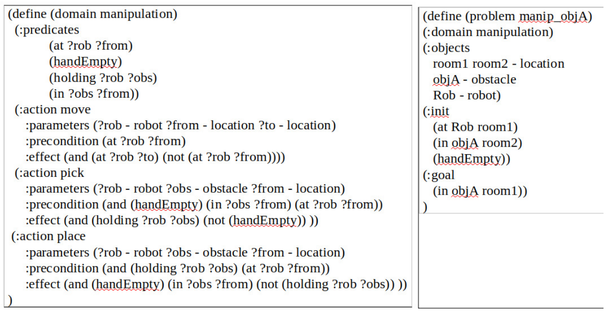
Fig. 1. TAMP configuration file template.

IV. Interfacing layers

An XML file, called TAMP configuration file, is proposed to act as an intermediate layer for the flexible linkage between symbolic actions and geometric data involved in task and motion planning (see Fig 1). The TAMP configuration file allows for flexibly setting the parameters required to define the problem, which includes the PDDL domain and problem files, along with the kautham motion planning problem file. For instance, here a simple manipulation domain will be used describing actions like PICK, PLACE, and MOVE, along with predicates like *at*, *in*, *holding*, and *handEmpty*, as shown in Fig. 2, where a simple PDDL problem file involving the manipulation of a single object is also shown.

The XML configuration file also includes the setting of the initial state: the objects to be manipulated and their poses along with the robots, their control files and the controls values that define the robot initial configurations. By modifying the initial state, we can flexibly create different instances of a manipulation task.

The main part of the TAMP configuration file provides flexibility in defining the actions. A XML tag is defined per action, with the PDDL symbols of the action as attributes and the associated geometric information defined in child XML tags. For the simple manipulation domain we have:



```
(define (domain manipulation)
  (:predicates
          (at ?rob ?from)
          (handEmpty)
          (holding ?rob ?obs)
          (in ?obs ?from))
  (:action move
    :parameters (?rob - robot ?from - location ?to - location)
    :precondition (at ?rob ?from)
    :effect (and (at ?rob ?to) (not (at ?rob ?from))))
  (:action pick
    :parameters (?rob - robot ?obs - obstacle ?from - location)
    :precondition (and (handEmpty) (in ?obs ?from) (at ?rob ?from))
    :effect (and (holding ?rob ?obs) (not (handEmpty)) ))
 (:action place
    :parameters (?rob - robot ?obs - obstacle ?from - location)
    :precondition (and (holding ?rob ?obs) (at ?rob ?from))
    :effect (and (handEmpty) (in ?obs ?from) (not (holding ?rob ?obs)) ))
)
```

```
(define (problem manip_objA)
(:domain manipulation)
(:objects
  room1 room2 - location
  objA - obstacle
  Rob - robot)
(:init
  (at Rob room1)
  (in objA room2)
  (handEmpty))
(:goal
  (in objA room1))
)
```

Fig. 2. PDDL domain and problem files.

The PICK action is defined for a robot and an object when located in a given region. Here, first the identifiers of the object and the robot in Kautham are given, together with the control file to be used to move the robot. Then, the set of Kautham controls that define the robot configuration when the robot is at the region is provided and, optionally, multiple robot configurations (also defined in kautham controls) to grasp the object. These configurations will be checked sequentially and the first feasible one (i.e. with no collisions) will be chosen. Alternatively, if no grasp configuration is given, the pose of the object has to be provided. In this case, an internal procedure will be used to compute the grasp to pick the object at that pose.

The PLACE action is defined for a robot and an object when located in a given region. Here, first the identifiers of the object and the robot in Kautham are given, together with the control file to be used to move the robot. Then, the set of Kautham controls that define the robot configuration when the robot is at the region is provided and, optionally, multiple robot configurations (defined in kautham controls) to place the object. These configurations will be checked sequentially and the first feasible one (i.e. with no collisions) will be chosen. Alternatively, if no robot configuration is given, the pose where to place the object has to be provided, or a region from where to sample a potential placement pose. In these latter cases, an internal procedure will be used to compute the robot configurations to reach the placement poses.

The MOVE action is defined for a robot to move between two specified regions. Here the identifier of the robot in Kautham is first given together with the control file to be used to move it. Then, the two robot configurations (defined in kautham controls) is provided, corresponding to each of the two regions.

V. The Task and Motion Planning Client

A Python client has been impemented[1] as a planning manager, sketched in Algorithm 1. It is responsible of reading the TAMP configuration file and using the ROS services required for solving the given task and motion planning problem. First, the PDDL domain and problem files and the Kautham file

Algorithm 1 Task and Motion Planning Client

```
Input fin: XML TAMP configuration file
Output fout: XML Task File
    Initialization
 1: domain ← fin.read(PDDL domain file)
 2: problem ← fin.read(PDDL problem file)
 3: kautham ← fin.read(Kautham problem file)
 4: obstacles ← fin.read(initial state)
 5: srvOpenProblem(kautham)
 6: srvSetObstaclePos(obstacles)
    Task planning
 7: TaskPlan ← srvFF.plan(domain, problem)
    Loop all actions in the task plan
 8: for Action in TaskPlan do
 9:   if (Action = PICK) then
10:     obj ← fin.read(Pick.Obj)
11:     rob ← fin.read(Pick.Rob)
12:     link ← fin.read(Pick.Link)
13:     controlfile ← fin.read(Pick.Cont)
14:     atregion ← fin.read(Pick.Regioncontrols)
15:     atpick ← selectGrasp(fin.read(Pick.Graspconf),
                              fin.read(Pick.Pose))
16:     srvSetRobControls(controlfile)
17:     path1 ← srvGetPath(srvSetQuery(atregion, atpick))
18:     srvAttachObstacle2RobotLink(obj,rob,link)
19:     path1 ← srvGetPath(srvSetQuery(atpick, atregion))
20:     fout.append(path1,path2)
21:   else if (Action = PLACE) then
22:     obj ← fin.read(Place.Obj)
23:     rob ← fin.read(Place.Rob)
24:     controlfile ← fin.read(Place.Cont)
25:     atregion ← fin.read(Place.Regioncontrols)
26:     atplace ← selectPlaceConf(fin.read(Place.Graspconf),
          fin.read(Place.Pose),fin.read(Place.Poseregion))
27:     srvSetRobControls(controlfile)
28:     path1 ← srvGetPath(srvSetQuery(atregion, atplace))
29:     srvDetachObstacle(obj,rob)
30:     path1 ← srvGetPath(srvSetQuery(atplace, atregion))
31:     fout.append(path1,path2)
32:   else if (Action = MOVE) then
33:     rob ← fin.read(Move.Rob)
34:     controlfile ← fin.read(Move.Cont)
35:     atreg1 ← fin.read(Move.Conf)
36:     atreg2 ← fin.read(Move.Conf)
37:     path ← srvGetPath(srvSetQuery(atreg1, atreg2))
38:     fout.append(path)
39:   end if
40: end for
41: return fout
```

are read. The PDDL files are passed as a request to the Fast Forward task planning service, which returns the sequence of actions. Then, the Kautham problem is opened using the *OpenProblem* service and the object poses modified using the *SetObstaclePos* service, according to the values given in the TAMP configuration file. Finally, the actions of the plan are processed, calling the Kautham *SetQuery* and *GetPath* services to find the robot paths, and the *AttachObstacle2RobotLink* and *DetachObstacle* services to update the state when pick and place actions are involved. The sequence of paths obtained

[1]Code can be found in the *python-branch* of the git repository https://gitioc.upc.edu/rostutorials/task_and_motion_planning.git.

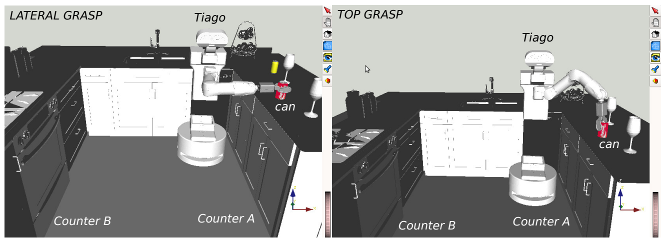


Fig. 3. Grasp configurations to pick the can.

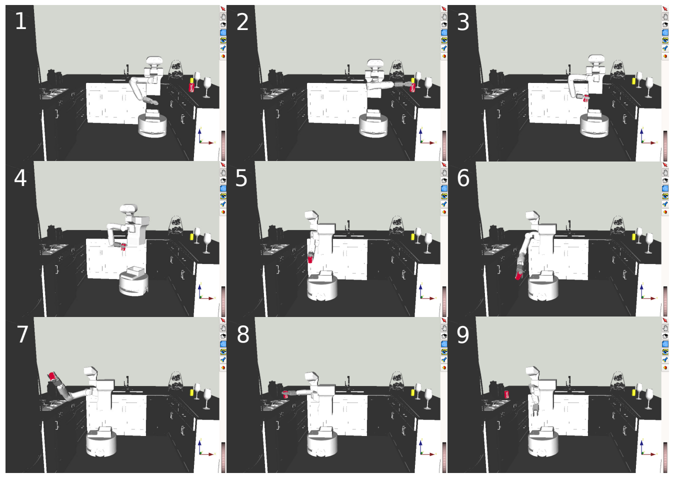


Fig. 4. Snapshots of the complete task execution. Video available at sir.upc.edu/projects/kautham/videos/tampconfig.webm

are stored in a single XML file as a sequence of Transit and Transfer actions that can be visualized with the Kautham GUI.

VI. An example

The Tiago-kitchen problem demonstrates the task and motion planning of a PAL-Robotics TIAGo robot in a kitchen environment with two different counters and two cans and glasses. The example demonstrates the flexibility in configuring different instances of the manipulation task. The TIAGo robot uses two different control files, as specified in the TAMP configuration file. The manipulation tasks use a control file that defines controls for the torso, the arm and the gripper, while the base remains fixed, Whereas for the navigation tasks, a control file defines controls only for the base while the arm remains in a fixed position as the robot navigates through the environment. For grasping the objects, two grasp configurations, namely the lateral grasp and the top grasp, are checked sequentially for feasibility and chosen, while using the controller that controls only the arms and the gripper. Also, while placing the manipulated red can in the example, these configurations are again considered. The example is available in the python-branch branch of the git repository given above, where the readme file provided describes the steps to reproduce the example and the requirements.

VII. Discussion and future work

The availability of a flexible and easy way to set task and motion planning problems is a key point towards the exploration of the tight coupling between planning levels that exists in some manipulation problems, and in the proposal and benchmarking of combined ways of planning.

The framework presented here gives the tools to get easy access to task and motion planing servers and the simple interconnection through an XML file. This XML file is currently configured by hand, and the challenge is now to automatically generate it. With this in mind, we are focusing the future working in:

- the use of deep learning vision techniques to automatically set the initial state (which objects are in the scene and which are their poses);
- the use of reasoning procedures and ontological knowledge to automatically set the PDDL problem file (the objects involved in the planning, the initial symbolic state, and the conditions that the goal has to satisfy);
- the use of methods to automatically generate action-oriented grasp configurations, like those explored in [12], that use strategies to simultaneously satisfy the stability and the semantic constraints that have to be satisfied for an action to be feasible.

References

[1] 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, 2009, p. 5.

[2] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[3] S. M. Lavalle and J. J. Kuffner, "Rapidly-Exploring Random Trees: Progress and Prospects," *Algorithmic and Computational Robotics: New Directions*, pp. 293–308, 2001.

[4] I. Sucan, M. Moll, L. E. Kavraki *et al.*, "The open motion planning library," *Robotics & Automation Magazine, IEEE*, vol. 19, no. 4, pp. 72–82, 2012.

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

[6] M. Ghallab, A. Howe, C. Knoblock, D. Mcdermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins, "PDDL—The Planning Domain Definition Language," 1998.

[7] N. Dantam, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki, "Incremental task and motion planning: A constraint-based approach," in *Robotics: Science and Systems*, 2016.

[8] A. Akbari, F. Lagriffoul, and J. Rosell, "Combined heuristic task and motion planning for bi-manual robots," *Autonomous Robots*, pp. 1–16, 2018.

[9] 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 *IEEE Int. Conf. on Emerging Technologies and Factory Automation*, 2014.

[10] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, vol. 2. IEEE, 2000, pp. 995–1001.

[11] J. Hoffmann and B. Nebel, "The FF planning system: Fast plan generation through heuristic search," *Journal of Artificial Intelligence Research*, pp. 253–302, 2001.

[12] M. U. Din, M. U. Sarwar, I. Zahoor, W. M. Qazi, and J. Rosell, "Learning action-oriented grasping for manipulation," in *IEEE Int. Conf. on Emerging Technologies and Factory Automation*, 2019, pp. 1575–1578.