

Multi-Step Planning for Robotic Manipulation of Articulated Objects

Max Pflueger
University of Southern California
pflueger@usc.edu

Gaurav Sukhatme
University of Southern California
gaurav@usc.edu

Abstract—Multi-step planning is a process that allows us to solve complex problems by using a hybrid of a continuous kinematic planner and discrete search algorithm. In this paper we demonstrate how to formulate practical robotic manipulation tasks into this planning framework by applying the approach to that of a robot planning to fold a typical folding chair. We believe that this planning approach could have wide applications in allowing robots to perform tasks with more generalized and intuitive goal conditions.

I. INTRODUCTION

Robot path planning has been well studied in the context of moving the robot from one state to another, however the useful objective of most problems is to move an object in the world that is not a robot from one state to another. This presents a significant problem for kinematic path planners as the free axes of the objects in the scene cannot be directly actuated, and many concepts of locality are no longer accurate, i.e. if an object is only a small distance from the goal state, the path length to the goal state may actually be very far if the robot is not nearby.

Planners that explore the space through available control actions could theoretically solve this problem, however it would require incorporating the physics of the interaction between the robot and object, which could be quite complex. Additionally, the planner would be subject to a significant problem of 'narrow passages' at the states where interaction with the object occurs.

In this paper we have attempted to construct a planner that is useful for these sorts of problems. Current state-of-the-art kinematic robotic path planners are able, in most circumstances, to easily produce a plan from an initial state in joint space to a final state, if one exists. We combine this capability with a discrete search algorithm in a technique known as multi-step planning.

The multi-step planning process solves this problem by creating classes of manifolds in the world state space, known as stances, where we know how to plan across each class of manifold. We use a discrete search algorithm that can choose the transitions between stances that allow us to reach our goal state, combined with continuous planners that can plan within our stances to determine if a transition is feasible, and if so present a solution.

Multi-step planning has been used before in other domains of robotics [1], our contribution in this paper is to show how

to adapt this planning approach to manipulation problems with multi-armed robots. We demonstrate the usefulness of this approach by developing a multi-step planner that can handle the task of unfolding typical folding chair.

II. RELATED WORK

Multi-step planning has been used before in some other specialized applications, Bretl [1] used it to plan a path for a wall climbing robot, and Hauser [2] demonstrated applications of machine learning to improve the efficiency of the planner.

Path planners for robot arms have been well studied and multiple good solutions exist including PRM's [3] and RRT's [4] [5]. Practical implementations of multiple such algorithms exist in the OMPL library [6].

III. PROBLEM DESCRIPTION

A. General Problem

The full planning space of the problem combines the full pose of the robot along with the poses of all objects in the scene. Planning in that space in a raw form would normally be very difficult due to the high dimensionality of the space, and the need to model state transitions that take into account the physical interaction of objects. We simplify this space by assuming that the scene (non-actuated degrees of freedom) is fixed, and only moved under special circumstances, in this case, objects in the scene only move when the robot is holding on to them.

B. The Chair

We examine the problem of a robot that wishes to fold or unfold a typical folding chair. The initial and goal states will be specified as poses for the chair, leaving the planner to decide where to grab the chair and how to move. We allow ourselves full prior knowledge of the shape and kinematics of the chair, as well as a list of available grasp points on the chair.

We will be performing this manipulation with the PR2 robot from Willow Garage, as a practical matter the robot is not strong enough to manipulate the chair arbitrarily, so we add an extra constraint to the state of the chair that requires two legs of the chair to be on the ground at all times, and we assume they will not slip. In most practical configurations this will ensure the robot does not have to support any significant forces with its arms. In this initial implementation we have also assumed that the pose of the chair base link (the back of the chair) will not change.

IV. ALGORITHM

We consider the full state space of our world to be some element of \mathcal{R}^n . Some dimensions of our world can be directly controlled (i.e. robot joints), where as others cannot (e.g. the position of an object). We further define the idea of a stance in the world as some qualitative and discrete interaction of objects in the world. By restricting the robot to a certain stance, we create a manifold through our full state space, over which we are able to plan.

We consider a *stance* to be defined as the state of interaction between the robot and the objects in the world, i.e. where and if the robot grippers are attached to another object in the world. We define a *configuration* to be the combination of a *stance* and the current state of the object(s) in the world. Finally we say that a *state* or *world state* is the combination of a *configuration* with the current state of the robot (or *robot state*).

The main multi-step planner is run as method sketched out in Algorithm 1. It is essentially a breadth first search process, exploring from the start configuration to adjacent configurations and so on. With the development of a distance metric and a good heuristic we could use a more sophisticated discrete search process, though it would not fundamentally change the structure of the algorithm.

Algorithm 1 getPlan

Input: world states $init$ and $goal$
Output: trajectory, if one exists
 $state_queue.push_back(init)$
 $current \leftarrow state_queue.pop()$
for all $s \in adjacent(current)$ and not yet visited **do**
 if s is reachable from $current$ **then**
 mark s as visited
 $state_queue.push_back(reachable\ state\ in\ s)$
 if s is the goal stance **then**
 trace steps backwards to construct the trajectory and return
 end if
 end if
end for

Fig. 1 shows a simplified illustration of what this search graph might look like. We see that the initial state (state 1) is adjacent to a number of other states, though only some of them (solid lines) are reachable through the current stance. By exploring the graph the planner can find a path to the goal (state 5) by passing through stance 2.

A. Adjacency

An important abstraction here is the idea of an adjacency. Two stances are considered to be adjacent if there exists a world state that is valid in both stances. The implementation of this algorithm requires a way to find sample world states that are valid for both the current stance and an adjacent stance.

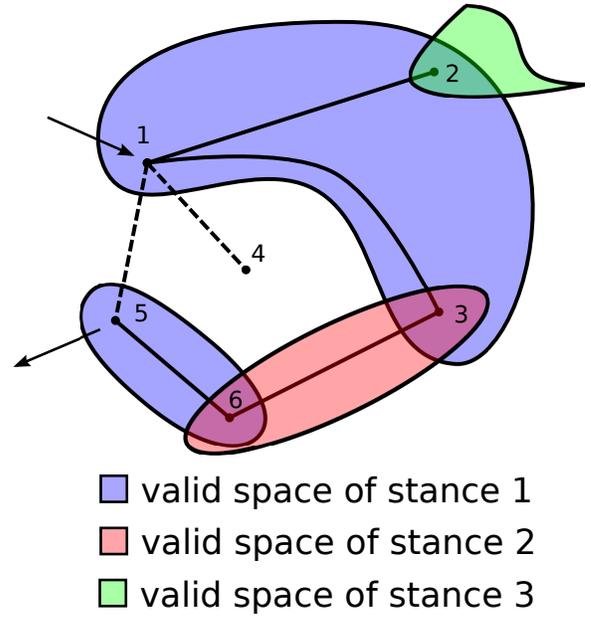


Fig. 1. A simplified example of a multi-step planning tree through multiple stances

B. Reachable

The reachable function is designed to tell the discrete search algorithm if a valid path exists from one world state to another. To do this it has to verify that any changes in the configuration are valid (i.e. objects cannot move unless the robot is holding them), and use a continuous planner to verify that a feasible path exists for the robot.

V. IMPLEMENTATION

To demonstrate a practical application of this algorithm we implemented it for the task of folding a typical folding chair. We chose this problem because it presents a difficult planning problem requiring multiple steps that is not immediately solvable with standard planning techniques.

A. Problem Specifications

We allow the algorithm to start with a complete kinematic model of a real chair, supplied in the standard URDF format. We also provide a list of valid grasp points on the chair. Each grasp point is tied to a particular link on the chair, so it will continue to be valid through manipulation actions. Fig. 2 shows a visualization of available grasp points being evaluated during the planning process, note that some of the grasp points are shown for hypothetical configurations of the chair. Grasps are shown green when a valid kinematic solution could be found to perform a grasp, and red when one could not be found, or because that grasp is considered in collision.

Currently the list of grasp points passed to the algorithm is hand specified, however it would be sensible to do this process automatically. The running time of the algorithm, as well as the existence and quality of a solution, can depend heavily on how many and where grasp points are provided, so having a principled process here would be quite useful.

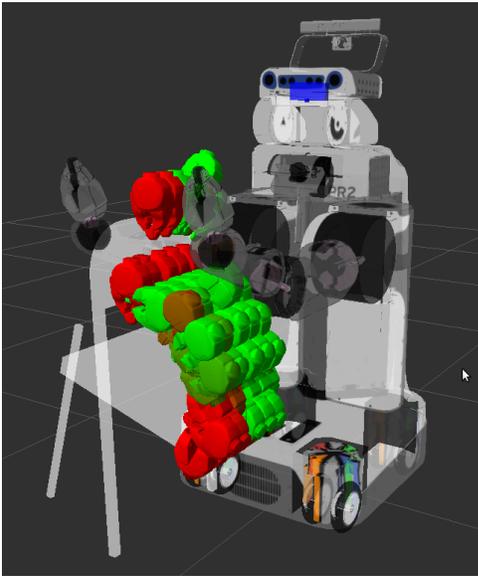


Fig. 2. Visualization of hypothetical grasp points during planning for different chair configurations. Grasps are shown as PR2 grippers, green for valid, red for invalid

The initial state of the object is observed at the beginning of the planning process, the final state is specified as the state of the object and robot. Currently we specify the pose of the root link of the object to be unchanged, and focus on changing the object articulation, but we hope to support changes in object pose soon.

B. Chair State Estimation

Perceiving the exact state of the chair is also a challenging problem, however a general solution to that problem is outside the scope of this work. We have attached AR Tag style markers to the chair that can be used with a calibrated 2D camera to perceive the pose of the parts of the chair in the frame of the robot. We used the `ar_pose` ROS package to do perception of these tags. In ROS we treat the chair as another robot with a URDF file, so it is necessary to extract from multiple observed tracking tags the joint angles of the chair. We calculate the observed rotation between two links of the chair, then project that rotation in quaternion space onto the constraint created by the rotational joint in the URDF specification of the chair. Note that an axis of rotation constraint becomes a hyperplane passing through the origin in 4D quaternion space, so we can use standard linear algebra techniques for this.

C. Adjacent Stance Expansion

We maintain an index of sampled world states for every stance intersection, and by choosing an upper limit on how many states may be sampled for a given stance intersection we avoid the issue of oversampling an intersection if it is revisited.

The sampled world states are maintained in a data structure that associates each world state with the stances it is in, and allows efficient queries of stance intersections to find all world states in that intersection.

D. Reachability Checking

Reachability checking involves determining when a state transition is feasible in 1 step. We have to verify 3 things, kinematics, collisions, and path validity.

To check kinematics we look at the initial and final states wherever the robot is holding an object, if the robot is holding an object, we use inverse kinematics to verify that a valid kinematic solution exists for the configuration of the object in that state. We use IK methods provided by the KDL library.

Currently we check for collision only by specifically excluding grasps that are known to cause collisions in certain object configurations, however it makes sense to use a more generalized collision checker here.

Checking path validity can be done by a standard robot arm path planner for whatever change in robot joint state is necessary from initial to final conditions. We are currently using joint space interpolation for unconstrained arms, but with further work we plan to incorporate a standard collision avoidance path planner here. Here we add some additional constraints that the configuration of the object may not change unless the robot is holding it with both arms on different links (this is a constraint specific to the chair, but it is conceptually simple to generalize to other objects, you can't move a joint unless you are holding both of the links). When the robot does change this configuration of the object, this creates some special constraints as a closed kinematic chain, discussed below.

1) *Closed Kinematic Chains*: Solving kinematics for closed chains can be difficult in general, we deal with this problem by prioritizing the pose of the object. When a path must be found where the robot changes the configuration of the object, we perform a linear interpolation in the joint space of the object, and then solve inverse kinematics for the robot to hold the object at points along that path. We keep the joint states of the robot smooth by seeding each IK solver with the solution from the previous state.

E. Discrete Search

We currently use breadth first search for the planning process, thus searching for a plan with a minimum of steps. This may be a significant factor slowing down the search, and we will be working to develop a good heuristic to enable other search algorithms such as A* in the future.

F. Path Execution

We assume that our environment is very predictable and so after finding a solution trajectory we simply execute it without sensor feedback.

VI. EXPERIMENTS

The full implementation is still a work in progress, but we have been able to demonstrate this planner running both in simulation and on the real robot, where we get reasonable planning results.

Fig. 3 shows a sequence of states from the planner running in simulation to fold a folding chair. A rough model of the

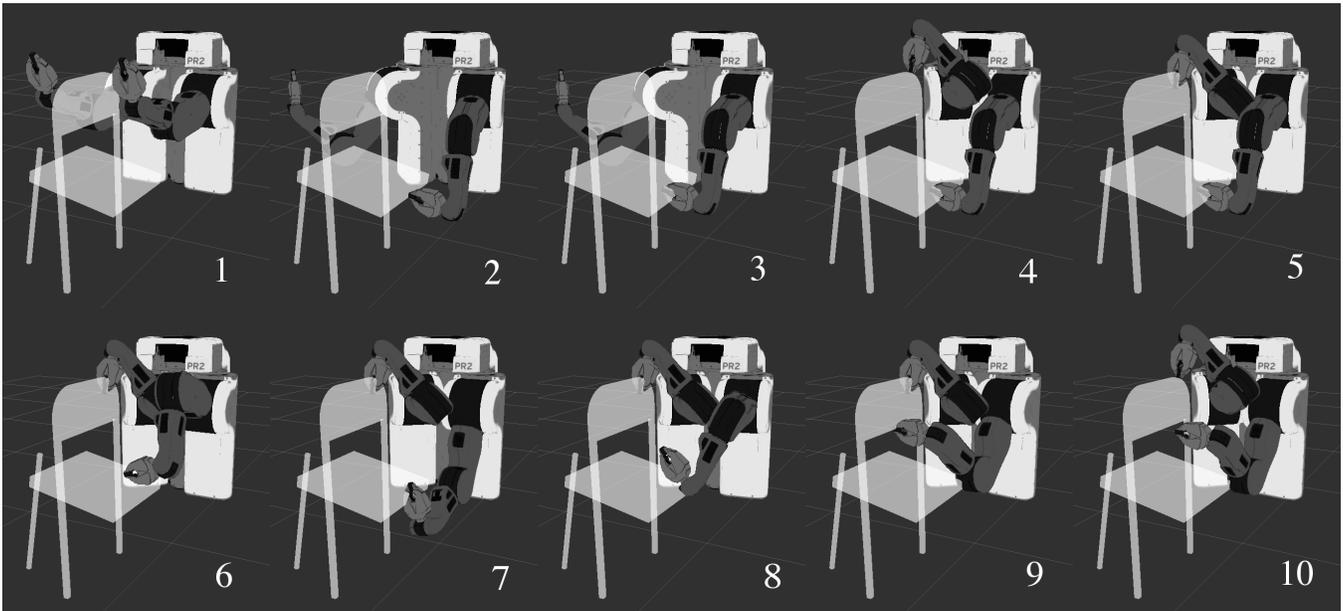


Fig. 3. A sequence of robot states from a plan generated in simulation. The initial state of the chair is shown transparent, the goal state is completely folded up. Observe how the robot moves the seat part way, then transitions to a different grip (state 7) from which it is possible to finish the manipulation. Note that gripper open/close is not implemented and not all intermediate states are shown.

chair is shown partially transparent in the initial state. The goal state is fully folded up. Note that the grippers are not shown to open or close because that part is not yet implemented. Also, since full collision detection has not been integrated into the planner, some of the states shown have the arms in collision with each other. Despite this, we can see that the planner has developed a solution that detects and deals with the issue of needing to change grasps part way through the folding action.

We are currently working to better characterize the performance characteristics of the algorithm, but initial trials have shown the computation time to find a 7 step solution at about 10 minutes on a quad-core desktop computer.

Implementation on the real robot is still very rough, but a video of one test can be seen here: [<http://youtu.be/ZeW7Rh4x81Q>].

VII. DISCUSSION & FUTURE WORK

In working with the folding chair we have made a number of assumptions and simplifications that should be removed to make this algorithm more useful. In addition to these relatively mechanical improvements to the algorithm we think some conceptual improvements could be made as well.

One of the biggest weaknesses with this approach is the large amount of information it requires about the object(s) in the planning scene, so finding a way to estimate kinematic models or work with less information would be useful. Also, the current system executes plans blindly, so modifying the algorithm to react to failures or moved objects could improve success rates, particularly in real world environments.

Generalizing this approach for new tasks will require, at least at first, a similarly detailed description of the environment, and the stances available to the robot. As with the chair

it will also require knowledge of what sort of motions are allowed in each stance, e.g. the chair is heavy, so it can't be fully lifted off the ground.

VIII. CONCLUSION

We have proposed an approach for adapting the technique of multi-step planning to the object manipulation class of problems. We developed an implementation of this approach and have presented some initial results in our evaluation of its performance. We believe that multi-step planning is promising paradigm for allowing robot path planning to explore problem areas that currently require significant amounts of application specific coding. Although our current implementation still has significant amounts of application specific code, we believe further work in this paradigm will allow for generalized planners that will work without application specific code.

REFERENCES

- [1] T. Bretl, S. Lall, J.-C. Latombe, and S. Rock, "Multi-step motion planning for free-climbing robots," in *Algorithmic Foundations of Robotics VI*. Springer, 2005, pp. 59–74.
- [2] K. Hauser, T. Bretl, and J.-C. Latombe, "Learning-assisted multi-step planning," in *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*. IEEE, 2005, pp. 4575–4580.
- [3] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *Robotics and Automation, IEEE Transactions on*, vol. 12, no. 4, pp. 566–580, 1996.
- [4] S. M. LaValle, "Rapidly-Exploring Random Trees: A New Tool for Path Planning," 1998.
- [5] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.
- [6] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, December 2012, <http://ompl.kavrakilab.org>.