Toward a Lifelong Motion Planning System that Learns from Experience

Dmitry Berenson Pieter Abbeel Ken Goldberg

University of California, Berkeley, Berkeley, CA
{berenson, pabbeel}@eecs.berkeley.edu, goldberg@berkeley.edy

One of the long-standing goals of research in robotics is to create a robot whose ability to do a task improves as the robot performs similar tasks. Consider a mobile manipulator operating in a domestic environment. A common task for such a robot may be to bring the user an item from the kitchen's cupboard. The motion planning capabilities required for such a task are non-trivial: the robot must be able to operate in a complex and changing environment that imposes a variety of constraints on the robot's motion. The robot must also be able to solve a given query within a reasonable amount of time. The most common approaches to solving such tasks is to plan with no prior knowledge of the task and the environment; a planning algorithm is only given the robot and environment models and the start and goal configurations, and asked to generate a path. While this approach, which we term *planning-from-scratch*, is general, it can produce unacceptably long planning times for difficult problems. Even if the robot previously generated a path for a very similar task, planning-from-scratch affords no way to take advantage of this previous computation.

Our current work seeks to create a system for planning paths in high-dimensional spaces that is able to learn from experience, with the aim of minimizing computation time. This system is intended for use on mobile manipulators such as the PR2 as well as in robotic surgery. We seek to create a system which leverages the generality of planning-from-scratch to produce solutions in situations the robot has not encountered previously and the efficiency of re-using previous experience in situations similar to previously-encountered ones. Our goal is to integrate these two approaches to create a system that is both general and efficient.

I. Overview

To this end, we are currently implementing a system that is based on a teacher-student model. In this system, we run two modules in parallel: a state-of-the-art general planner (the teacher), and the algorithm that re-uses previous paths (the student). Given a new query, both modules are started simultaneously and the first path produced by either module is executed on the robot while the other module is stopped.

If the teacher was the first to produce a path, this means that the student did not have a path sufficiently appropriate for the given query. Thus, to improve the student's performance on similar queries, we add the path produced by the teacher to the student's path library. If the student produces a path before the teacher, then the student has already surpassed the teacher's ability on the given query, so there is no need to add the student's path into the library if we are only interested in surpassing the teacher. However, there may be some situations where the teacher's performance is quite poor, and simply surpassing the teacher is not sufficient in terms of computation time. Thus, if the student was the first to produce a path and that path was significantly different[1] from the path it chose to re-use, we add that path to the student's path library. In this way, the student is able to surpass the teacher's performance on difficult problems while not cluttering the path library with redundant paths.

This system architecture has several advantages. First, we do not need to arbitrarily pre-generate some number of paths before running the system. The student can start with an empty path library and build that library as the robot performs its intended tasks. This is a significant advantage because it may be difficult to envision the types of tasks and environments the robot encounters before it actually encounters them.

Also, if we consider sampling-based motion planning, there can be a huge benefit in terms of computation time for difficult problems. Consider the reaching-into-the-cluttered-cupboard example described above. This is a typical narrow-passage problem, and solving such problems with sampling-based planners is usually quite time-consuming. However, since the geometry of the kitchen cupboard does not change between queries, previously-computed paths will be very close to valid (we only need to avoid the changing clutter in the cupboard, not navigate into the tight space of the cupboard). Thus the system will be able to solve the query much more quickly than a sampling-based planner.

Finally, a major advantage of this system is that it is *lifelong*; i.e. we can continue to run the system over the lifetime of the robot, even as the robot transitions between tasks. We loose nothing when transitioning to a new task because, in the worst case, we will always perform as well as the teacher, which is a state-of-the-art planner.

In the following sections we provide a description of our implementation of the student and teacher modules of the system. We emphasize that this is only one embodiment of

[1] We use Dynamic Time Warping to evaluate path similarity.

the system and that there are many ways to implement these modules. However, in our evaluations, these implementations performed the best in terms of computation time.

II. Teacher Module

We implement the teacher module as a BiDirectional RRT (BiRRT). This method is known for its fast planning times and a wide array of enhancements have been presented on how to make the algorithm more efficient. These enhancements can be used to create multiple teacher modules that all train the student by simply generalizing the rule for adding a new path to the library: If any teacher succeeds before the student, all other teachers and the student are stopped and the path is added to the student's library. Such an approach would of course require more computational resources, and may benefit from off-board computation.

III. Student Module

There are two core parts to the implementation of the student module:

1) Selecting an appropriate path
2) Transforming this path into a feasible one

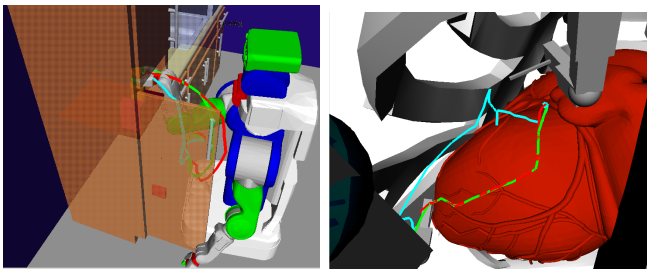
Of course, these steps are linked, and the efficiency of the second depends highly on the first. One issue that complicates selecting an appropriate path is that the paths in the library will not, in the general case, start and end at the appropriate configurations. We can amend this by simply connecting the endpoints of the paths to the given start and goal by straight lines. Once we do this, we need to decide which path is most likely to yield the fastest computation in the second step.

We employ a heuristic to inform this decision, which assumes that the amount a given path violates the constraints of the environment correlates with the time necessary to transform the path into a feasible one. This heuristic is motivated by our implementation of the second step.

If we had sufficient computational resources, we would evaluate how much each path in the library violates the constraints of the task and environment, i.e. how much the path is in collision and/or violates a kinematic constraint, and select the least-violating path. However, evaluating all the paths in a library is quite time-consuming and thus impractical for large libraries.

Thus we employ a second heuristic, which first selects n paths from the library based on their distance to the start and goal and then evaluate the violation of those n paths. The motivation for this heuristic is that the larger the segments we need to connect the end-points of path to the start and goal, the more chance there is to violate constraints. We've found that this heuristic produces good results in practice and allows us to save a great deal of computation time.

After selecting an appropriate path, the student then transforms this path into a feasible one. Again, there are many methods in motion planning and even trajectory optimization that are capable of this task. Since our aim is to minimize computation time, we use what we believe to be the fastest approach: repairing the path using multiple BiRRTs. This method is also general, in that it is guaranteed to find a solution if one exists. The implementation of this method is as follows:



PR2 in Kitchen Laparoscopic Heart Surgery

Fig. 1. Examples of smoothed paths generated by the teacher and student modules for two tasks (only the path of the end-effector is shown). Red: Path chosen from path library. Green: Path produced by the student. Light Blue: Path produced by the teacher. The student generated its path faster than the teacher in both of these examples.

First, we check each point in the discretized path for constraint violation. Any segments of the path that do not violate constraints are preserved. To get from one valid segment to another, we run BiRRT, whose start tree is rooted at the end of one valid segment and whose goal tree is rooted at the beginning of the next valid segment. The composit path consisting of originally-valid path segments and path segments produced by the BiRRT is then returned as the path.

Finally, it is important to note that both the teacher and student modules do not consider path quality in their computation. To optimize the quality of the path, the path of whichever module succeeds first is then passed to a path optimizer before execution. Again, there are many path optimization methods that could be used, and our current implementation uses a short-cut smoother. Examples of smoothed paths produced by the teacher and student modules for two tasks are shown in Figure 1.

IV. Summary

In summary, we have described our ongoing work in creating a system for planning paths in high-dimensional spaces that is able to learn from experience. Our goal in this work is to produce feasible paths while incurring minimal computation time on practical planning problems for manipulators. Our system leverages the generality of planning-from-scratch to produce solutions in situations the robot has not encountered previously and the efficiency of re-using previous experience in situations similar to previously-encountered ones. Our approach consists of a teacher-student framework, where the student stores examples generated by the teacher in its path library, depending on the student's performance. As a task is performed more often, the student can re-use paths from the library to achieve computation times that are better than those of the teacher. We are currently evaluating the performance of our system on domestic manipulation tasks using the PR2 robot and on surgical tasks for a laparoscopic surgery robot.