Experience Guided Mobile Manipulation Planning

Tekin Meriçli[1] and **Manuela Veloso**[2] and **H. Levent Akın**[1]
{tekin.mericli,akin}@boun.edu.tr,veloso@cmu.edu

[1] Department of Computer Engineering, Boğaziçi University, Bebek, 34342, Istanbul, Turkey

[2] School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, United States

Abstract

The most critical moves that determine the success of a manipulation task are performed within the close vicinities of the object prior to grasping, and the goal prior to the final placement. Memorizing these state-action sequences and reusing them can dramatically improve the task efficiency, whereas even the state-of-the-art planning algorithms may require significant amount of time and computational resources to generate a solution from scratch depending on the complexity and the constraints of the task. In this paper, we propose a hybrid approach that combines the reliability of the past experiences gained through demonstration and the flexibility of a generative motion planning algorithm, namely RRT*, to improve the task execution efficiency. As a side benefit of reusing these final moves, we can dramatically reduce the number of nodes used by the generative planner, hence the planning time, by either early-terminating the planner when the generated plan reaches a "recalled state", or deliberately biasing it towards the memorized state-action sequences that are feasible at the moment. This complementary combination of the already available partial plans and the generated ones yield to fast, reliable, and repeatable solutions.

Introduction

Planning for fine reaching and manipulation is a difficult problem as the generated plans should be very well tuned for the robot to succeed in its task. Depending on the complexity and the constraints of the manipulation task at hand, even the state-of-the-art planners may require significant amount of time and computational resources to generate plans that will end with a success. The most critical parts of a manipulation task, though, are usually the moves performed within the close vicinities of the object prior to grasping or the goal prior to the final placement. Re-using parts of the solutions around those critical regions that are executed in the past and known to be successful can increase the manipulation efficiency significantly by reducing the overall computational demand for planning.

In this paper, we contribute an algorithm that utilizes demonstrated and memorized state-action sequences for fine manipulation, and complements them with a generative planner if necessary. Given target object and goal configurations, the robot searches its *memory* of demonstrated fine manipulation motions executed successfully in the past, checks their feasibilities for the current situation, and utilizes the generative planner to reach a state where it can recall the rest of the motion required to pick up or drop off the object. Once it reaches one of the *recalled* states, the robot proceeds with the execution of the corresponding sequence while actively monitoring its execution to prevent failures that may occur due to sensing and actuation noise. As the most constrained parts of the task are covered by the *remembered* execution sequences, utilization of these past experiences for fine manipulation significantly reduces the amount of search that the generative planner has to perform, hence improving the overall planning time and efficiency. In that case the generative planner is only used to "roughly reach" the critical motion region around the object or the goal, and then hand over the delicate motions to the memorized state-action sequences.

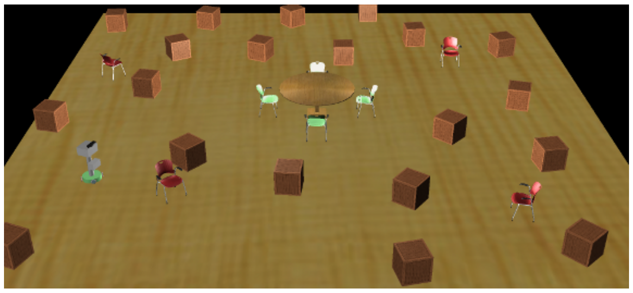


Figure 1: Simulated problem environment with randomly scattered chairs to be manipulated (red), desired final chair poses indicated around a round table (green), boxes that serve as obstacles, and an omni-directional robot used as the mobile manipulator (bottom left corner).

We use each of the demonstrated state-action pair *sequences* individually as they are, considering them to be the "fine-tuned final moves" that lead to the solution of the problem at hand, even though the common practice in the world of Learning from Demonstration (LfD) is to use a suitable machine learning algorithm to obtain a generalized execution policy out of the demonstrated state-action pairs that can be used for reproducing the demonstrated task (Argall et al. 2009). In our work, these sequences are used analogously to "cases" in a Case-Based Reasoning/Planning system (Bartsch-Spörl, Lenz, and Hübner 1999; Spalzzi 2001); that is, the robot knows how to handle the rest of the problem when it recalls the case. Therefore, all we need to do is to move the robot to the cases that it can recall and apply directly, similar to what was done in (Ros et al. 2009) with the introduction of the *controllable state features* concept. This concept was used to transform the currently perceived state to a familiar one rather than trying to adapt the case to the current state.

Our proposed approach is suitable for any problem that requires fine reaching and manipulation planning. As an example problem domain, in this work we picked the particular problem of manipulating chairs that are scattered in an environment cluttered with obstacles, and arranging them in a predefined seating formation. In order to achieve this goal, the robot first needs to *reach* the chairs, and then *manipulate* them in such a way to carry them to their desired poses while avoiding the obstacles and the other chairs in the environment. One constraint that we have in this problem is that the robot can only grab a chair if it approaches the chair from behind within certain distance and orientation limits. Not exceeding a certain velocity while trying to approach the chair is also important since it can accidentally be bumped into and pushed away from the robot, rendering it unreachable in some circumstances. Figure 1 shows a screenshot from the simulated problem environment, where the chairs to be manipulated are shown in red and their desired final poses are shown in green. The robot shown on the bottom left corner of the figure is composed of a round omni-directional base and a special attachment unit that grabs the chair when the robot is properly positioned behind it. On this example problem, we present extensive experimental evaluation of our algorithm using RRT* as the generative planner, and two different methods of combining past experience with the plan generation process.

The rest of the paper is organized as follows. First, we provide some brief background information on the Rapidly-exploring Random Trees (RRT) based approaches and give an overview of the related work. Next, we thoroughly explain the proposed approach, present extensive experimental evaluation, and discuss the results in depth. Finally, we conclude our paper and point out some possible future work we aim to tackle.

Background

In this work, we utilize a Rapidly-exploring Random Trees (RRT) (LaValle 1998; 2006) based algorithm, namely RRT*, as the generative planner component of our proposed approach since RRT is the most commonly used sampling based motion planning algorithm due to its simplicity, practicality, and probabilistic completeness property. Starting from the initial configuration, the RRT algorithm incrementally builds a tree by uniformly sampling points from the state space and growing the tree in the direction of the sample by extending the closest node of the tree towards the sample. It is also possible to bias the tree growth towards the goal by using the goal itself as a sample with probability p, and sampling randomly with probability $1 - p$.

As the number of nodes in the tree increases, finding the closest node to the randomly sampled point takes more time since this operation requires the entire tree to be traversed. There has been attempts to use more efficient data structures for faster nearest neighbor computation as well as heuristics to speed up the algorithm. Bruce and Veloso (Bruce and Veloso 2003) contributed the execution extended RRT (ERRT) algorithm, where the plan generated in the previous iteration is used to guide the progress of the plan in the current iteration by keeping a waypoint cache with the assumption that the environment did not change significantly between the two iterations. This approach proves very useful when used in highly dynamic environments that require intensive re-planning, such as robot soccer. Urmson and Simmons (Urmson and Simmons 2003) proposed heuristically-guided RRT (hRRT), where they shape the probability distribution to make the likelihood of selecting any particular node based on the node's potential exploratory or lower cost path contributions.

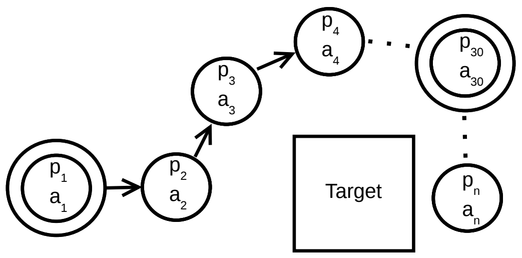
A very recent study that is focused on the optimality of the paths generated by RRT was contributed by Karaman and Frazzoli (Karaman and Frazzoli 2010), in which they present RRT*. In an RRT* tree, each tree node stores its distance from the start node in terms of path length, and instead of looking for the closest single node to the sampled point, RRT* looks for a set of *near nodes*. Therefore, when a new point is sampled, not the closest node but the node with the lowest cost among the near nodes is extended towards the sample. Also, the near nodes neighborhood is restructured by modifying the parents and children of the nodes in order to end up with lowest cost paths. In our experiments, we used RRT* since the optimality and directedness of the generated paths better complemented the idea in our proposed approach, which is reaching the critical region as fast and directly as possible and fine tuning within that region.

Related Work

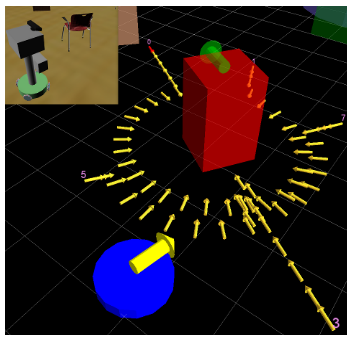
Reusing previously constructed paths or motion segments has been investigated in various forms in the literature. We review the most recent related studies here.

Cohen *et al.* (Cohen et al. 2011) proposed constructing a graph with predefined motion primitives and performing a search on that graph to plan a path while satisfying the constraints via the solutions provided by primitives that are generated on the fly by various analytical solvers. In order to improve the planning efficiency, they initially plan for only 4 of the total of 7 degrees of freedom (DoF) of the manipulator, and then switch to full 7 DoF planning towards the end, which resembles the approach that we propose in this paper.

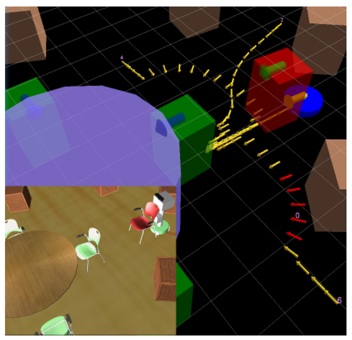
Berenson *et al.* (Berenson, Abbeel, and Goldberg 2012) contributed a framework which utilizes stored end-to-end paths and a generative planner in parallel to plan the required motion for a given problem. They simultaneously start planning from scratch and looking for a similar stored path which they can repair and reuse, and use the path returned by whichever method finds a solution first. The similarity of a stored path to the given situation is determined by comparing the start and end points, and BiRRT is used



(a)



(b)



(c)

Figure 2: (a) Diagram illustrating how the pose-action pairs are chained one after the other. The concentric circles represent the entry points. Snapshot from the simulation and the visualization environments showing the robot trying to reach the chair (b) and the goal after the chair is grabbed (c). The orientations of the robot, the chair, and the goal are indicated with thick arrows, and the entry points of the fine reaching sequences are depicted as little arrows around the chair and the goal.

to repair infeasible paths by filling in the gaps caused by the obstacles along the path.

Ye and Alterovitz (Ye and Alterovitz 2011) combined LfD with motion planning. The demonstrated motions are recorded relative to the robot's torso as well as the objects of interest in the task, and dynamic time warping is used for aligning multiple demonstration sequences. Implicitly encoded constraints, such as keeping the spoon full of sugar level to avoid spilling, are automatically extracted from the task execution sequences by looking at the low variance portions of the data. Recorded and processed trajectories are reused in relatively similar scenarios to the ones used for learning, and generative planning is utilized to bridge the gaps when obstacles are present along the way. As in the work of Berenson *et al.* (Berenson, Abbeel, and Goldberg 2012), they also store full, end-to-end paths. In order for the full length paths to be meaningful for reusing, a large number of them covering various situations should be stored. This is where our proposed approach differs from the presented methods in the literature. We store only a very little number of partial relative paths (i.e. sequences) that cover the critical region around the object and the goal; hence, these paths can be reused in any scenario setup.

Approach

Our algorithm consists of the following components:

- A set of execution sequences performed successfully in the past
- An execution monitoring system to ensure accurate execution of the remembered action sequences
- A generative planner

In the remainder of this section, we elaborate on how we define and obtain the set of fine-tuned final moves (i.e. sequences) for the reaching and manipulation sub-problems, how we ensure that these sequences are executed in such a way to obtain the desired results, and how we merge the solutions provided by a generative planner and the available sequences so that they complement each other effectively to yield faster and more reliable executions.

Execution Sequences for Fine Manipulation

Even though there are infinitely many possible reaching and grasping configurations for a particular object, humans usually follow a rough, object-independent reaching pattern that is as direct as possible, and switch to object-specific fine manipulation moves when they get close enough to the object. Those object-specific fine moves are usually finite in number; for example, a bottle is grasped from the side or on its cap, a mug is grasped from the side or on its handle, etc. In some cases, as in our problem domain, reaching and manipulation configurations may have additional constraints; that is, a chair can only be reached and grabbed when approached from behind within certain orientation limits.

Trying to generate plans that will achieve those fine moves from scratch each and every time the object needs to be manipulated becomes burdensome and inefficient. Therefore, we make the robot *memorize* these finite amount of object-specific fine reaching and manipulation moves represented as *sequences* of state-action pairs as we demonstrate the robot how to perform them.

In our example problem domain of chair manipulation, we joystick the robot within the close vicinity of the chair to demonstrate it how to approach the chair from different directions and end up right behind it for a successful pick up. Similar demonstrations are provided within the close vicinity of the goal pose to teach the robot how to approach the destination from different directions. These recorded sequences are composed of the robot's poses relative to the target (i.e. either the object or the goal) together with the corresponding motion commands. Therefore, a sequence takes the form

$$(\wp_1, a_1), (\wp_2, a_2), (\wp_3, a_3), ..., (\wp_n, a_n)$$

where $\wp_i$ is the relative pose of the robot to the target denoted as a 3-tuple $< x, y, \theta >$, and a_i is the action associated with $\wp_i$, also denoted as a 3-tuple $< v_x, v_y, v_\theta >$ indicating the translational and rotational velocities of the robot. Using relative poses provides the flexibility of pose invariance; for example, if a bottle to be reached and grasped lays on its side on a table, the robot can still perform the task even if the demonstrations were given when the bottle was standing upright on the table.

The demonstrations are recorded at each step of the robot's perception cycle, in our case at a frequency of 30Hz. As the number and the length of the recorded sequences increase, such high recording rate may pose problems in terms of efficient processing and scalability. To address this problem, we grant access to a sequence at every n^{th} frame, called an *entry point*. The value of n can be adjusted depending on the requirements of the task. Figure 2 illustrates a diagram representation of a sequence and shows snapshots of a certain configurations of the robot trying to reach the chair and carry it to the goal in the simulated environment, together with the corresponding sequences defined within the close vicinities of the chair and the goal for providing fine reaching moves. The entry points are visualized as arrows that indicate the robot's pose at that particular frame of the sequence at $n = 30$; that is, the entry points are present every 30 frames.

Since these sequences are defined relative to the target, some of the entry points may not be reachable due to obstruction depending on the target's global pose and the state of its immediate surrounding. This is indicated by the *feasibility* flag of the entry point, which is set according to the reports of a collision model. Only the feasible set of entry points are considered when making use of the sequences. An example visualization of the feasible (yellow) and infeasible (red) entry points can be seen in Figure 2(c) and Figure 4.

Execution Monitoring

Due to the uncertainty in sensing and actuation that affects the execution, there may be discrepancies between the expected outcome of a particular action and the actually observed one. Therefore, actively monitoring the execution is necessary in robotics in order to detect and handle problems caused by the uncertainty rooted in both the robot itself and the environment (Pettersson 2005).

Since we make the robot use the demonstrated fine reaching and manipulation moves without any generalization, we need to ensure that those moves are performed as demonstrated to lead to successful completion of the task. Therefore, we make the robot keep track of its execution along the sequence by computing the difference between its actual pose relative to the target and the pose stored in the current frame of the sequence, and checking whether the location and orientation differences are within certain tolerance thresholds. Based on its memory of the sequence, at each time step the robot knows what state it should be in after performing the action associated with the current state. If it ends up in an unexpected state; that is, if the current frame of the sequence being followed suggests that the robot's relative pose at that moment should be $\wp$ but it is actually $\wp'$ and $||\wp - \wp'|| > \varepsilon$, then the robot computes a linear interpolation between $\wp$ and $\wp'$, and moves accordingly to get back to the expected state. After correcting its pose, the robot continues following the sequence. As opposed to blindly executing the demonstration, execution monitoring increases the chance of successful operation.

Generative Planner

The fine reaching and manipulation sequences are demonstrated only within the close vicinity of the target; therefore, the robot first needs to reach these sequences to be able to use them. However, since the demonstrated sequences are taking care of the fine moves that need to be performed to achieve the task, reaching the sequences can be done roughly and as directly as possible, hence saving computation and execution time. To fill in this gap, we use the RRT* generative motion planning algorithm even though any kind of motion/path planner could have been used for this purpose. RRT variants in general provide fast solutions, and RRT* in particular provides an optimal solution in terms of path length.

We utilize the feasible entry points of the sequences to guide the generative planner in two ways. The first approach is to early-terminate plan generation when an entry point (i.e. a recalled state) is by chance reached first while trying to reach the goal directly. The second approach is to treat entry points as sub-goals and deliberately direct the generative planner towards them as well as the goal itself, considering that each sequence, hence each entry point, leads to the goal. This guidance results in much more direct paths to the entry points and it significantly reduces the number of RRT* nodes required to find the path. In RRT-based algorithms, the cost of finding the nearest node of the generated tree to the randomly sampled point goes up with the increasing number of nodes in the tree; therefore, by keeping the number of generated nodes low, we reduce the planning and execution time significantly. Figure 3 shows the difference between using the generative planners alone, early-terminating the plan when an entry point is reached by coincidence, and deliberately guiding the planner by directing it towards the entry points.

In addition to reaching the sequences when they are far away, the generative planner is also used during execution of the sequences to hop to another feasible entry point when the next entry point in the followed sequence is observed to be infeasible (i.e. obstructed), which in a way similar to the methods used by Berenson *et al.* (Berenson, Abbeel, and Goldberg 2012) and Ye and Alterovitz (Ye and Alterovitz 2011) to bridge the gaps. In order to prevent infinite loops, all entry points prior to the current entry point of a sequence are marked as infeasible as the robot moves along the sequence. Figure 4 illustrates both of these concepts.

Algorithm

Initially, the remembered sequences are placed around the target based on its global pose. The entry points of the sequences are tested for collisions with the environment, and marked as either feasible or infeasible accordingly. Then, the set of feasible entry points are passed to the generative planner along with the actual desired final pose of the robot. When the nodes of the generative planner coincide with an entry point within the distance and orientation thresholds

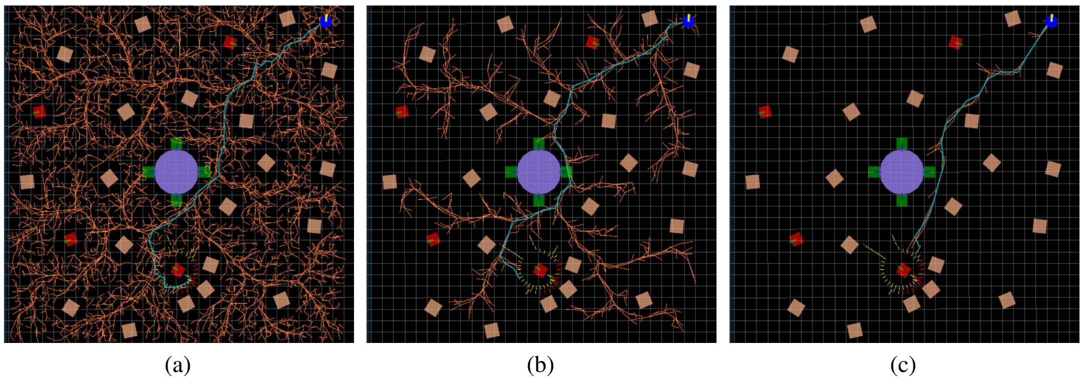


Figure 3: Different ways of combining the generative planner and the sequences to achieve the task; (a) using RRT* alone without any guidance (4480 nodes), (b) early termination of the plan when an entry point is hit by coincidence (716 nodes), and (c) deliberately directing the planner towards the entry points (67 nodes).

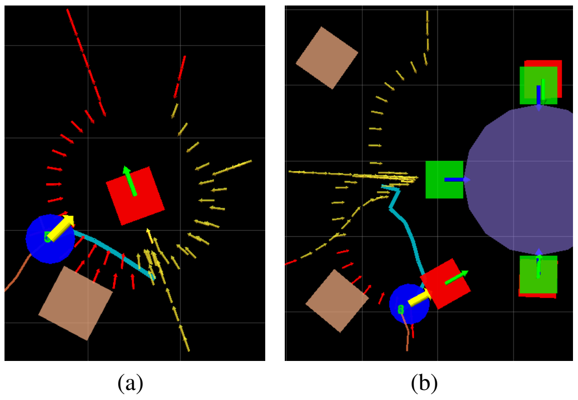


Figure 4: Using the generative planner to bridge feasible entry points during reaching (a) and manipulation (b). Feasible entry points are shown in yellow, infeasible ones are shown in red, and the generated path is shown in blue.

used for execution monitoring purposes, planning process is terminated and the generated waypoints are followed to get to the target entry point (Figure 3). Once the entry point is reached, the robot starts executing the actions stored in the sequence frame by frame while monitoring its execution at each step. If the next entry point of the sequence happens to be blocked while the robot is following a sequence, then the selected sequence is dropped and generative planning is initiated again to form a bridge between the current entry point and another feasible one (Figure 4).

The benefits of the proposed approach are two-fold:

1. By utilizing the local solutions obtained through demonstration within the close vicinity of the target, the generative planner is relieved from the challenging task of generating plans that will provide very fine moves to solve the entire problem. Therefore, it is only used to reach the critical region as roughly and directly as possible.
2. The generative planner serves as a gap filler either between points where no previously known (i.e. demonstrated) solutions exist, or between feasible entry points of the sequences in case the next entry point is observed to be obstructed while following a sequence.

Also, by increasing the number of possible goal points via treating the entry points as sub-goals, the generative planner is helped to generate more directed solutions in much less time since it is much easier to reach a bigger region of many possible goals than a single point with tight orientation constraints. This complementary combination of the already available partial plans and the generated ones yield to fast yet reliable solutions.

Experimental Evaluation

Experiments are performed in Webots mobile robot simulation environment [1]. The simulated worlds were $15m \times 15m$ in dimensions, and the robot had a maximum translational velocity of 0.5 m/s and a rotational velocity of $\pi/2$ rad/s. We first demonstrated the robot 5 sequences for reaching the chair and grabbing it, and 4 sequences for delivering the chair to a goal pose. Then we created 10 challenging setups for the problem of chair manipulation, and ran 100 experiments for each of the three possible uses of the generative planner.

The RRT* generative planner is allowed to expand trees of up to 12000 nodes, each of which represented the 3-DoF pose of the robot since the the orientation as well as the location is important in manipulation tasks. A goal bias of $p = 0.01$ is used during tree generation process and the search is terminated when a node is within 0.05 m of location and $\pi/36$ rad of orientation difference limits. Due to these tight limits, the generative planner often failed to generate a valid path (i.e. exceeded the maximum number of allowed nodes) when it is used without any guidance by the entry points, either as early termination conditions or subgoals to be reached; however, when guidance was utilized,

[1]Webots Mobile Robot Simulation Software, http://www.cyberbotics.com

the planner never failed. The average failure rates of the RRT* planner when used alone was 41.29%. The number of nodes generated when the generative planner was used alone is still substantially higher (Figure 3) even if we leave out the failed cases and compute the mean over 100 successful cases. Figure 5 shows the mean number of nodes used by plain RRT as well as the RRT* generative planners in each of the 10 experiment environments. The target chair was very close to the edge of the world and had an obstacle right behind it in Environment 7, making it particularly challenging to generate a collision-free path from scratch.

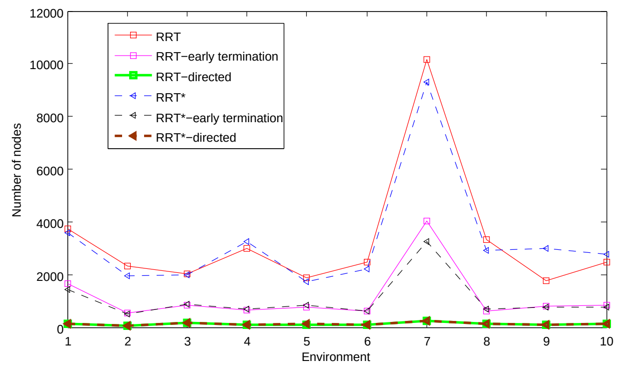


Figure 5: Mean number of nodes generated by the RRT and RRT* planners in various guidance levels.

In order to demonstrate the advantage of utilizing the bigger region created by the local sequences around the target over merely increasing the single goal bias, we blocked the direct path to the back of the chair, as shown in Figure 3, and tested all three approaches in that setup by running each of them 10 times. Table 1 reports the average required time and the number of nodes to generate a solution. The reason that RRT* alone with a high goal bias ($p = 0.75$) spent such a long time is because it failed several times before a valid path is finally generated.

Table 1: Performance with blocked direct path to the object.

Method	# nodes	Planning time (sec)
RRT* alone	7690	309.682
Early termination	1249	0.821
Deliberate directing	44	0.081

We also measured the average execution time for completing the entire task of arranging all four of the chairs in their desired final configurations, which is shown in Table 2.

Table 2: Average task completion times.

Method	Task completion time (sec)
RRT* alone	364.011
Early termination	280.779
Deliberate directing	259.439

Conclusion and Future Work

In this work, we presented an algorithm for reaching and fine manipulation tasks, which utilizes previously experienced locally defined fine-tuned motion sequences to achieve the task, and uses a generative planner to reach the region created by these sequences as fast and directly as possible. We show that generative planning can be significantly reduced by past experience guidance utilized as either early termination conditions or sub-goals that the robot is trying to reach. This complementary combination of the already available partial plans and the generated ones yield to fast yet reliable and repeatable solutions.

Application of the proposed approach to both prehensile and non-prehensile manipulation problems in higher dimensional setups, handling uncertainty in perception explicitly in addition to execution monitoring, accumulating new experiences over time, and transferring learned fine reaching and manipulation sequences among objects with similar properties are some of the problems to tackle in the future.

References

Argall, B. D.; Chernova, S.; Veloso, M.; and Browning, B. 2009. A survey of robot learning from demonstration. *Robotics and Automation Systems* 57(5):469–483.

Bartsch-Spörl, B.; Lenz, M.; and Hübner, A. 1999. Case-based reasoning - survey and future directions. In *Proc. of the 5th German Biennial Conference on Knowledge-Based Systems*, 67–89. Springer Verlag.

Berenson, D.; Abbeel, P.; and Goldberg, K. 2012. A robot path planning framework that learns from experience. In *Proc. of ICRA*.

Bruce, J., and Veloso, M. M. 2003. Real-time randomized path planning for robot navigation. *Lecture Notes in Computer Science* 288–295.

Cohen, B. J.; Subramania, G.; Chitta, S.; and Likhachev, M. 2011. Planning for manipulation with adaptive motion primitives. In *ICRA'11*, 5478–5485.

Karaman, S., and Frazzoli, E. 2010. Incremental sampling-based algorithms for optimal motion planning. In *RSS*.

LaValle, S. M. 1998. Rapidly-Exploring Random Trees: A New Tool for Path Planning. Technical report.

LaValle, S. M. 2006. *Planning Algorithms*. New York, NY, USA: Cambridge University Press.

Pettersson, O. 2005. Execution monitoring in robotics: A survey. *Robotics and Autonomous Systems* 53:73–88.

Ros, R.; Arcos, J. L.; Lopez de Mantaras, R.; and Veloso, M. 2009. A case-based approach for coordinated action selection in robot soccer. *Artificial Intelligence* 173:1014–1039.

Spalzzi, L. 2001. A survey on case-based planning. *Artificial Intelligence Review* 16:3–36. 10.1023/A:1011081305027.

Urmson, C., and Simmons, R. 2003. Approaches for heuristically biasing RRT growth. In *IROS 2003*, volume 2.

Ye, G., and Alterovitz, R. 2011. Demonstration-guided motion planning. In *Proc. of ISRR*.