INTECH
open science | open minds

OPEN ACCESS
ARTICLE

International Journal of Advanced Robotic Systems

Improving Prehensile Mobile Manipulation Performance through Experience Reuse

Regular Paper

Tekin Meriçli[1,*], Manuela Veloso[2] and H. Levent Akın[1]

[1]Department of Computer Engineering, Bogazici University, Istanbul, Turkey
[2]Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA
* Corresponding author E-mail: tekin.mericli@boun.edu.tr

Received D M 2014; Accepted D M 2014

DOI: 10.5772/chapter.doi

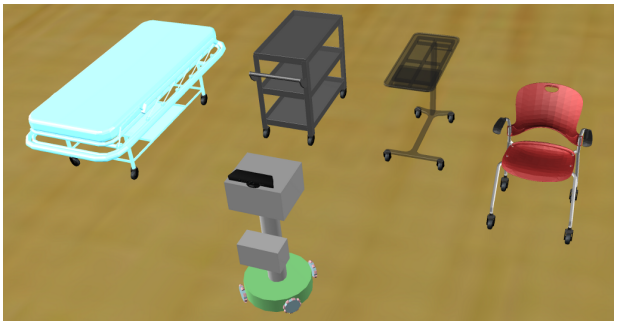
Abstract During pick and place tasks, a mobile manipulator performs recurring relative moves within the close proximities of the object of interest and the destination independent of their global poses. These moves are usually critical to the success of the manipulation attempt and hence need to be executed delicately. Considering the critical yet recurring nature of these moves, we let the robot memorize them as state-action sequences and reuse them whenever possible to guide manipulation planning and execution. When combined with a sampling-based generative planner, this guidance helps reduce planning time by deliberately biasing the planning process towards the feasible sequences. Additionally, monitoring the execution while reiterating the reached sequences improves task success rate. Our experiments show that this complementary combination of the already available partial plans and executions with the ones generated from scratch yields to fast, reliable, and repeatable solutions.

Keywords Experience-guided manipulation planning, mobile manipulation

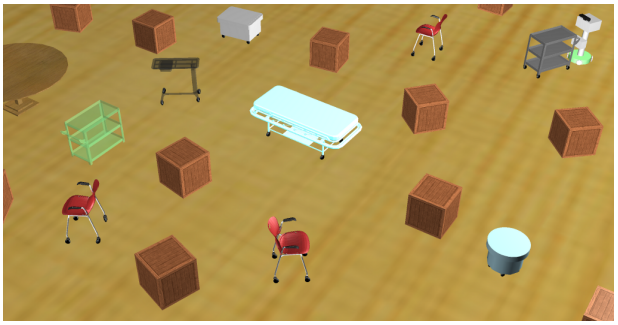
1. Introduction

Majority of the activities that are performed in everyday living contexts are instances of pick and place tasks. These types of tasks usually require careful planning and delicate execution as the manipulator has to be finely aligned with the object of interest (OOI) during pick up and the destination during placement. However, depending on the constraints of the tasks and the complexity of the task environments, even the state-of-the-art planners may require significant amount of time and computational resources to generate trajectories that will yield successful executions. On the other hand, careful observation of the recurring pick and place actions that take place in everyday scenarios led to the following conclusions, which motivated the ideas behind our contribution:

1. Reaching towards the OOI and/or the destination is usually done as directly, roughly, and quickly as possible. The most critical parts of a manipulation task are the moves performed within close proximities of the OOI prior to pick up or the destination prior to placement, and they need to be executed delicately.
2. Even though there could be infinitely many ways of picking up or placing an object, the general tendency is to repeat a finite and discrete number of alternatives for finely approaching a particular OOI and/or its destination. For example, a bottle is grasped from the side or on its cap, a mug is grasped from the side or on its handle, or a chair is usually grabbed from its back.



(a) The omni-directional mobile manipulator robot and several manipulable hospital and office objects used in the experiments.



(b) A random simulated task environment cluttered with various manipulable objects, stationary obstacles, and furnitures.

Figure 1. The omni-directional mobile robot capable of manipulating several hospital and office objects (a) is transporting a utility cart from the top right of the scene to its desired pose on the middle left of the scene, indicated as the pale green ghost shape of the cart (b).

Building on these observations and conclusions, we contribute an experience-based mobile manipulation planning and execution method that allows the robot to memorize and later reuse the critical reaching moves performed within close proximities of the OOIs and their destinations. We show that this approach reduces the overall computational demand for planning as the critical manipulation regions are handled by the memorized solutions. The main role of generative planning is to move the robot towards the critical manipulation region as quickly and directly as possible. This paradigm results in considerable efficiency improvements compared to planning from scratch every time the OOIs need to be manipulated. We also observe that reiterating the moves that have been executed in the past and are known to be successful improves execution reliability. We run our experiments in a simulated setup where our mobile service robot has to pick up and transport several office and hospital objects, such as chairs, overbed tables, utility carts, and stretchers to their desired destinations while avoiding collisions in cluttered environments (Figure 1).

The rest of the paper is organized as follows. Section 2 provides some brief background information on the Rapidly-exploring Random Tree (RRT) variant generative motion planning algorithms. Section 3 gives an overview of the related work while Section 4 elaborates on the contributed method. Section 5 presents the results of our extensive experimental evaluation. Finally, Section 6 summarizes and concludes the paper while pointing out some potential future work.

2. Background

In this work, we utilize a set of RRT variant algorithms, namely the original RRT itself [1, 2], RRT-Connect [3], and RRT* [4], as the generative planner component of our proposed approach. In addition to their simplicity, practicality, and probabilistic completeness property, the sampling-based natures of RRT-based algorithms align very well with the way our method defines and stores the robot's manipulation experiences.

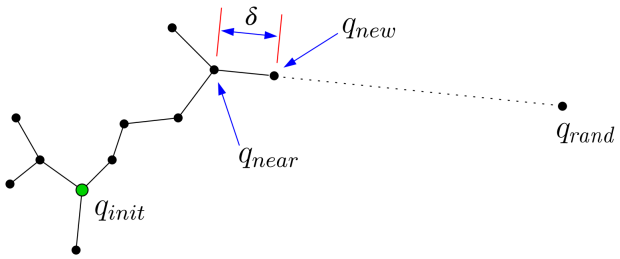


Figure 2. Tree construction process of the RRT algorithm [3].

Starting from the initial configuration q_{init}, RRT-based algorithms incrementally build a tree composed of random configurations that could potentially be bounded by certain constraints. In each iteration, a random configuration q_{rand} is picked by uniformly sampling the configuration space, the "nearest" node to the sample (q_{near}) is computed, the tree is extended a δ amount from q_{near} towards q_{rand}, and a new node q_{new} is added to the tree if that configuration is collision-free (Figure 2). Given enough time to explore the configuration space, the constructed random tree eventually reaches the goal configuration within some similarity boundaries, hence the probabilistic completeness property of the algorithm. Instead of following a purely random approach, it is also possible to inform the search process by biasing the tree growth towards the goal configuration using the goal itself as a sample with probability p, while sampling randomly with probability $1 - p$. Algorithm 1 presents the pseudocode of the RRT algorithm in its simplest form.

Algorithm 1 The basic RRT algorithm.

```
1:  function BUILDRRT(q_init, q_goal)
2:      Tree ← q_init;
3:      q_new ← q_init;
4:      while dist(q_new, q_goal) > THRESHOLD do
5:          q_rand ← Sample();
6:          q_near ← Nearest(Tree, q_rand);
7:          q_new ← Extend(q_near, q_rand);
8:          if CollisionFree(q_new) then
9:              Tree.add(q_new);
10:         end if
11:     end while
12:     return Path(Tree, q_new);
13: end function
```

As the number of nodes in the tree increases, however, finding the nearest node to the randomly sampled configuration takes longer since the entire tree needs to be traversed. There has been attempts to address this problem by using more efficient data structures for faster nearest neighbor computation as well as heuristics for informed exploration. Utilizing a kd-tree for partitioning the configuration space and searching for the nearest neighbor on the kd-tree instead of the random tree itself results in significant speed ups [5]. Heuristics, such as shaping the probability distribution to alter the likelihood of selecting any particular node based on the node's potential exploratory or lower cost path contributions, lead to higher quality paths [6]. In addition to improving the performance of an individual planning session, reusing previously generated plans to guide current ones proves very useful especially in highly dynamic environments, such as robot soccer, that require intensive re-planning [7]. 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 would not change significantly between consecutive iterations.

The RRT-Connect algorithm [3] uses a greedy heuristic that aggressively tries to connect the tree to the samples by moving over larger distances during configuration space exploration and tree expansion. Instead of performing a single step that extends the tree some δ amount towards the sample, the Connect heuristic iterates the extension operation until the sample is reached or an obstacle is encountered, resulting in quicker and more direct traversal of the distances between configurations. This property of the RRT-Connect algorithm aligns well with our motivating idea (1) about reaching the critical manipulation region as quickly and directly as possible.

Even though the RRT algorithm rapidly explores the configuration space and eventually reaches the goal, it does not guarantee that the solution path is an optimal one. The RRT* algorithm [4], on the other hand, focuses on the optimality of the paths generated by the RRT methodology. Each RRT* 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 configuration, RRT* looks for a set of *near nodes*. Therefore, when a new configuration 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. The asymptotic optimality and directness properties of the solutions generated by RRT* also align well with our motivating idea (1).

In Section 4.2, we elaborate on how we utilize these planning algorithms in our framework while improving their performances via experience guidance and reuse.

3. Related Work

It has been shown in general problem solving domains that reusing past experience stored as derivational problem solving episodes improves planning and execution efficiency [8–10]. Also, reusing previously constructed paths or motion segments has been investigated in various forms in the motion and manipulation planning literature. In this section, we review the most recent related studies.

The adaptive motion primitives concept introduced in [11] allows combining predefined multi-dimensional actions with the primitives that are generated on the fly via analytical solvers during plan graph construction and search processes. Planning efficiency is improved by initially planning for only 4 out of 7 degrees of freedom (DoF) of the manipulator, and then switching to full 7 DoF planning towards the end. This hybrid and multi-resolution approach resembles the method we present in this paper, where the fine manipulation region is reached roughly and the critical motions are performed delicately within that region.

The Lightning framework [12] utilizes stored end-to-end paths in addition to running a generative planner to plan the required motion for a given problem. The processes of planning from scratch and looking for a similar stored path that can be repaired and reused for the given problem are run simultaneously and the path returned by the process that finds the solution first is used. The similarity of a stored path to the given situation is determined by comparing the start and end points, and Bi-directional RRT (BiRRT) is used to repair infeasible paths by filling in the gaps caused by the obstacles along the path. One of the uses of the generative planner component of our contributed method is to bridge the gaps along the blocked sequences in a similar manner.

The Learning from Demonstration (LfD) concept [13] is utilized in [14] to simplify programming for pick and place tasks. Using a magnetic tracker to follow the index finger of the human teacher, static trajectories are demonstrated to an industrial manipulator equipped with a vacuum gripper to pick and place objects with certain shape constraints in an obstacle-free tabletop manipulation scenario. The captured trajectories are transformed to the robot's frame of reference and segmented to extract task primitives, which are then translated into robot-specific codes to be executed in order to replicate the demonstrated trajectory. Even though it may be suitable for factory environments, this approach lacks flexibility and has limited scalability as it requires new demonstrations for each new task environment configuration that could potentially include static obstacles.

The demonstration-guided motion planning (DGMP) framework [15] combines LfD with motion planning, where the demonstrated motions are recorded relative to the robot's torso as well as the OOIs in the task environment, and dynamic time warping (DTW) [16] is used for aligning multiple demonstration sequences. Implicitly encoded constraints, such as keeping a full spoon level to the ground 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. This algorithm also stores full, end-to-end paths, as in [12].

In order for the full length paths to be meaningful for reuse, a large number of them covering various situations should be stored. This is one of the aspects where our proposed approach differs from the presented methods in the literature. Instead of storing a large set of complete end-to-end solutions, our algorithm stores only a few partial relative trajectories that cover the critical regions around the OOIs and the destinations so that these partial executions can be reused in any scenario independent of the actual configuration of the environment.

4. Experience-Based Mobile Manipulation

Pick and place tasks require planning and execution of delicate reaching moves. As the manipulator approaches the OOI and/or the destination, these moves become more important and critical to the success of the manipulation task. Likely due to that criticality, humans exploit a small set of those target-specific delicate reaching and manipulation moves when performing everyday pick and place tasks even though there are infinitely many ways of reaching for and manipulating objects [17]. Inspired by these observations, we develop the following components and bring them together harmoniously to achieve experience-based prehensile mobile manipulation:

- A set of local reaching moves represented as sequences of state-action pairs that have been successfully performed in the past,
- A generative planner to bridge the gaps at various stages of planning and execution,
- An execution monitoring system that ensures accurate reiteration of the memorized sequences.

In the remainder of this section, we elaborate on how the target-specific local reaching moves (i.e., sequences) are defined and acquired, how the stored motion sequences are reiterated in such a way to obtain the previously experienced results, and how the solutions provided by the generative planner are merged with the available sequences so that they complement each other effectively to yield faster and more reliable executions.

4.1. Motion Sequences for Delicate Reaching and Manipulation

Having observed the critical yet repetitive nature of the finite sets of target-specific delicate reaching and manipulation moves, we let our mobile manipulator robot simply *memorize* them. That way, they could be reused for planning and performing mobile manipulation for the same objects in the future instead of trying to come up with plans that will achieve similar delicate moves from scratch each and every time the OOIs need to be manipulated.

It is necessary to define a compact yet extensive representation for these moves in order to address several issues simultaneously. For instance, in some cases, as in our problem domain, these moves may also have additional pose and dynamics constraints; that is, a chair, a utility cart, or a stretcher can only be reached and grabbed when approached from behind within certain orientation and velocity limits, otherwise get bumped into and pushed away without a successful grasp. Therefore, it is important for the robot to capture the velocity profile of the motion as it performs those delicate moves and memorizes them. Another important point is to make these fine trajectories independent of the target's global pose in order to maximize their utilization through potential reuse in various environment configurations. This is achieved by defining and storing the trajectories with respect to the target's frame of reference; that is, if the robot is to pick up the OOI, then the trajectories are in the OOI's frame of reference, and if it is to place the OOI, then the trajectories are in the destination's frame of reference.

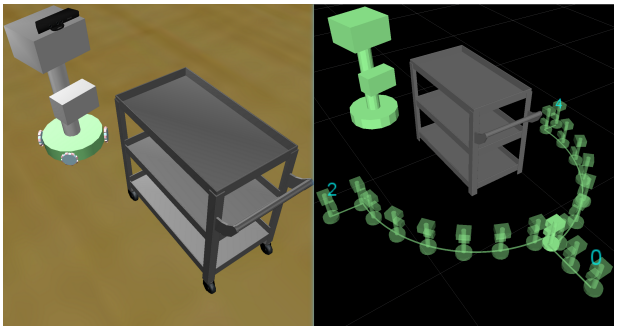
Formally, our algorithm attaches a static global frame of reference, φ_G, to the environment and separate frames of reference to the robot and the target (either the OOI or the destination), denoted as φ_R and φ_T, respectively, to define their poses within φ_G. Let $\wp_R$ be φ_R w.r.t. φ_T denoted as $\langle x, y, \theta \rangle$. Invariance to φ_T is achieved by storing $\wp_R$ instead of φ_R together with the motion command being executed at that pose. Therefore, a local delicate reaching and fine manipulation move, represented as a *sequence* of state-action pairs of length n takes the following form,

$$S_{i\in[0,L)} : ((\wp_{R_0}, a_0), (\wp_{R_1}, a_1), \ldots, (\wp_{R_{n-1}}, a_{n-1})),$$

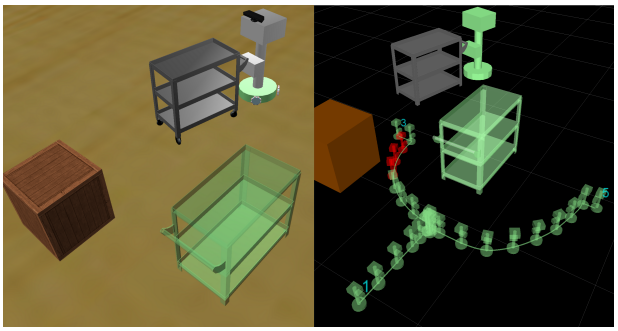
where $a_{j\in[0,n)}$ is the action associated with $\wp_{R_{j\in[0,n)}}$, denoted as $\langle v_x, v_y, v_\theta \rangle$, indicating the omni-directional motion command composed of the translational and rotational velocity components of the robot, and L is total number of sequences for a particular OOI. Figure 3 illustrates the visualization of the pick up (relative to the object) and placement (relative to the destination) sequences originating from poses located immediately around the target and leading the robot to the relative pose where it can pick up or drop off the OOI.

As the number of stored sequences and their lengths grow, processing efficiency and scalability starts to become a problem, especially if the sequences are being recorded at each step of the robot's perception cycle, which corresponds to 30Hz in our case. To address this problem, our algorithm sparsifies the sequences by granting access to them at every k^{th} frame, called an *entry point*. That is, the robot can merge into a sequence or leave it when it needs to only at the entry points. Figure 4 provides a close-up look at the visualization of the pick up sequences memorized for the chair object. The entry points are visualized as scaled-down robot figures that indicate the robot's relative pose at every k^{th} frame of the sequence. The value of k can be adjusted depending on the dimensions of the OOIs and the obstacles in the task environment as well as the motion precision requirements of the task. Sparser sequences with larger k values could be used in environments with larger obstacles and denser sequences with smaller k values could be used for checking and correcting for diversions from the sequences more frequently to meet precise movement requirements.

Since the sequences are defined relative to the target, some of the entry points may not be reachable due to direct obstruction or potential collisions depending on the target's global pose and the state of its immediate surrounding. One important role of the entry points is to provide a way to sparsely (hence quickly) check



(a) Three pick sequences obtained relative to the OOI.



(b) Three place sequences obtained relative to the destination.

Figure 3. Visualization of (a) pick and (b) place sequences associated with the utility cart object and its potential destination.

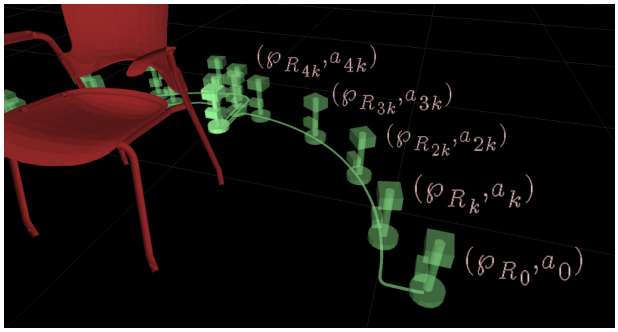


Figure 4. Visualization of the sequences around the chair object and their *entry points* depicted as scaled-down robot figures.

for collisions along the sequences to figure out which portions of them are usable for any given environment configuration. For that purpose, each entry point features a binary flag indicating its *feasibility*, the value of which is determined based on the collision reports provided by our custom-built collision checker. Depending on the type of a sequence, the collision checker tries to determine for each of the entry points whether there would be any collisions if the robot would have passed through it by itself (for pick sequences) or while carrying the OOI (for place sequences). In case the collision checker reports an obstruction or a potential collision on an entry point, it is marked as *infeasible* and excluded from the set of entry points to be utilized for planning and execution. An example visualization of the feasible (green) and infeasible (red) entry points can be seen in Figure 3(b) where an obstacle in the environment results in potential collisions although not directly obstructing the sequence.

Once reached (Section 4.2) and merged into through an entry point with index βk, $\beta \in \mathbb{N}^0$, the robot can start reiterating a sequence simply by executing $a_{j\in[\beta k,n)}$ that correspond to $\wp_{R_{j\in[\beta k,n)}}$ along the sequence. In order to guarantee successful completion of the task, the robot monitors the execution during the reiteration process (Section 4.3) to detect divergences from the expected outcome and compensate for them as well as to check for the feasibilities of the upcoming entry points to avoid potential collisions by planning an auxiliary path that would route the robot around the obstacle and merge it back into a feasible segment of one of the sequences (Section 4.2). Infinite loops during planning and execution are prevented by marking each traversed entry point as *infeasible* to remove them from the set of entry points to be considered for reuse in potential re-planning. Figure 5 depicts the process of reiterating a collision-free sequence.

4.2. Generative Planner

In our contributed method, the sequences representing local fine manipulation solutions are used analogously to "cases" in a Case-Based Reasoning/Planning (CBR/P) system [8–10, 18, 19]; that is, the robot knows how to handle the rest of the problem when it recalls the case. However, since our "cases" (i.e., sequences) are already fine-tuned solutions, we do not attempt to adapt them to the current problem configuration at hand as would be done in the *reuse* step of the original CBR/P approach. Instead, exploiting the fact that we can control the state of the robot, our algorithm simply moves the robot towards the cases that it can recall and apply directly. This approach resembles the *controllable state features* concept [20], which was used to transform the currently perceived state to a familiar one rather than trying to adapt the case to the current state.

Planning from scratch is only needed to navigate the robot along a collision-free path towards the sequences as the sequences take care of the delicate moves that need to be performed within the close proximity of the target to achieve the task. For that purpose, we utilize sampling-based generative planning, as the underlying representation aligns well with our definition of the robot's fine reaching and manipulation trajectories. We specifically use a set of RRT variant algorithms, namely the original RRT itself [1, 2], RRT-Connect [3], and RRT* [4], the working principles of which are provided in Section 2.

As mentioned in Section 2, it is possible to bias the growth of the RRT towards the goal by sampling the goal pose with probability p while sampling randomly with probability $1 - p$. We extend this concept to bias the tree growth towards the delicate manipulation *region* immediately around the target so that the robot can reach there as roughly and directly as possible and hand over the rest of the execution to the fine-tuned sequences to complete the task. That is, in the process of generating plans for the robot to reach the target, each feasible entry point on any sequence can potentially be considered as a *(sub)goal* to be reached, since merging into a sequence through any of those entry points would lead the robot to where it eventually wants to go. This approach also addresses indirectly, yet elegantly, one of the major

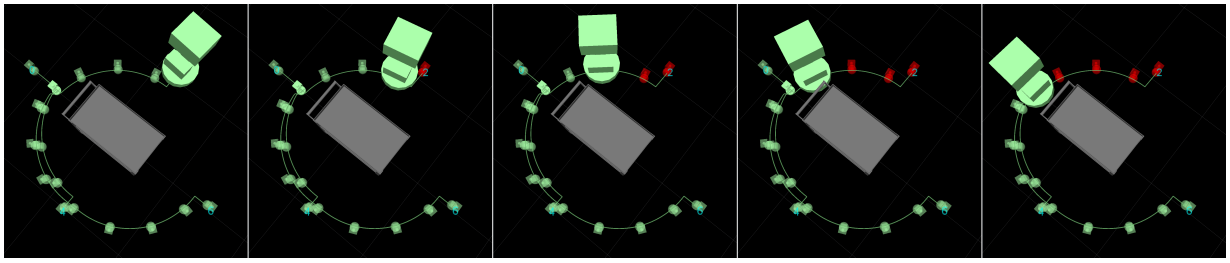


Figure 5. The robot reiterates a collision-free sequence simply by executing the motion commands associated with each frame of the sequence one after the other. Passed entry points are marked as *infeasible* (red) to prevent infinite loops of planning and execution.

problems of RRT-based algorithms, which is the increasing computational cost of the nearest neighbor search with the increasing number of nodes in the tree. Even though we represent the configuration space using a kd-tree and perform the nearest neighbor computations on the kd-tree in logarithmic time, keeping the number of generated RRT nodes as small as possible via proper biasing in sampling reduces computation and execution times considerably.

We define the following strategies for utilizing the feasible entry points to guide the generative planning process.

- **Sampling individual subgoals**: Let p_g be the probability of sampling the actual goal pose that the sequences lead to. Additionally, let p_{ep} be the probability of sampling an individual entry point from the set of all feasible entry points E. Provided that there is a small number of relatively sparse sequences, our algorithm can randomly sample an entry point (i.e., a subgoal) from E with probability $(|E|p_{ep})$ while sampling the goal with probability p_g and a random configuration with probability $(1-(|E|p_{ep})-p_g)$.
- **Sampling sequences**: Instead of sampling individual feasible entry points within sequences, with this strategy, our algorithm samples with probability p_s a sequence S_i from the set of sequences S associated with the target. Then an entry point is randomly sampled from the set of feasible entry points E_{S_i} that belong to S_i, which results in the total probability of sampling an entry point being $(|S|p_s)$. Similar to the previous strategy, a random configuration is sampled with probability $(1-(|S|p_s)-p_g)$.
- **Sampling subgoals within goal probability**: When this strategy is employed, our algorithm combines the actual goal with the set of feasible entry points, obtaining a total of $(|E|+1)$ (sub)goals. After deciding, with probability p_g, to use *any* goal as a sample, either the actual goal is picked with probability $p_{g^*} = 1/(|E|+1)$ or one of the feasible subgoals is randomly picked with probability $(1-p_{g^*})$.
- **Coincidental termination**: Instead of deliberately biasing the RRT growth towards the sequences, the feasible entry points can also be used for early-terminating the search process of the generative planner in case one of the entry points is coincidentally reached within some pose difference tolerance. Since the feasible entry points form a *region* of many subgoals to be reached, it becomes likely for the generative planner to arrive at one of those subgoals first while trying to reach the actual single goal pose.

When these various guidance strategies are enabled, we observe considerable reduction in the number of generated RRT nodes and more direct paths towards the fine manipulation regions around the targets, as we show in Section 5 as part of our extensive experimental evaluation. Algorithm 2 provides the pseudocode of our experience-based sampling function that utilizes the stored sequences based on the aforementioned strategies. This function replaces the *Sample* function used in line 5 of Algorithm 1.

Algorithm 2 Experience-based configuration sampling function that utilizes the sequences.

```
1:  function EXP_SAMPLE( )
2:      r ← rand()
3:      if STR_WITHIN_GOAL_PROB then
4:          if r ≤ p_g then
5:              r ← rand()
6:              if r ≤ 1.0/(|E| + 1) then
7:                  q_rand ← q_goal
8:              else
9:                  q_rand ← GetRandomEntryPoint()
10:             end if
11:         else
12:             q_rand ← GetRandomConfiguration()
13:         end if
14:     else
15:         if STR_INDIVIDUAL_SUBGOALS then
16:             p_total ← p_g + |E|p_ep
17:         else if STR_SEQUENCES then
18:             p_total ← p_g + |S|p_s
19:         end if
20:         if r ≤ p_g then
21:             q_rand ← q_goal
22:         else if r ≤ p_total then
23:             q_rand ← GetRandomEntryPoint()
24:         else
25:             q_rand ← GetRandomConfiguration()
26:         end if
27:     end if
28:     return q_rand
29: end function
```

Generative planning is not only used for navigating the robot towards the sequences that are far away, but also when the robot is reiterating a sequence and the upcoming entry point is observed to be infeasible (i.e., obstructed). In those cases, the generative planner guides the robot around the obstacle to merge into another feasible segment

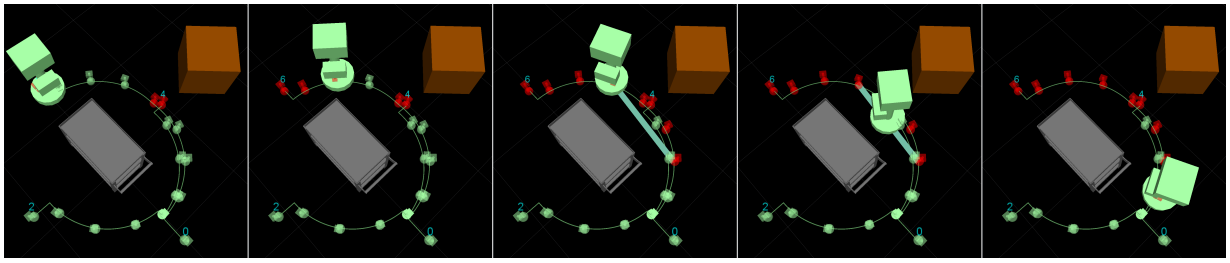


Figure 6. Reiteration of a sequence that is blocked by an obstacle. When the robot detects the upcoming entry point along the path to be *infeasible* (red), it uses the generative planner to hop to a feasible portion of either the same sequence or another one and proceed.

and proceed with the execution of the corresponding sequence, as was done in [12, 15] to bridge the gaps along the stored trajectories. Figure 6 illustrates this phenomenon. As mentioned before and shown in the figure, infinite planning and execution loops are prevented by marking the visited entry points as *infeasible* (red).

4.3. Execution Monitoring

Uncertainty is abundant in the world of robotics. Due to the uncertainty in sensing and actuation, there may be discrepancies between the expected outcome of a particular action and the actual observed one. Therefore, actively monitoring the robot during task execution is necessary in order to detect and handle problems caused by various sources of uncertainty that may be rooted in the task environment as well as the robot itself [21].

In order to achieve the task, we need to ensure that the sequences are reiterated as originally provided, mainly because those delicate reaching and manipulation moves are acquired and stored without any generalization. For that purpose, the robot keeps track of its execution by comparing its currently observed state to the expected one as it passes through each entry point while reiterating a sequence. The sequences provide the robot with the information that the observed state should be $\wp_{R_{j+1}}$ after executing a_j when in state $\wp_{R_j}$. If it ends up in an unexpected state, that is, if the currently visited entry point 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 ceases reiteration, computes a linear interpolation between $\wp$ and $\wp'$, moves accordingly to get back to the expected state, and resumes reiteration. An alternative implementation for the execution monitoring module could use a simple controller that would make slight modifications on the reiterated motion command to keep the robot on the corresponding trajectory at all times. Regardless of the implementation details, in general, execution monitoring increases the chance of successfully completing the task as opposed to open-loop reiteration.

5. Experimental Evaluation

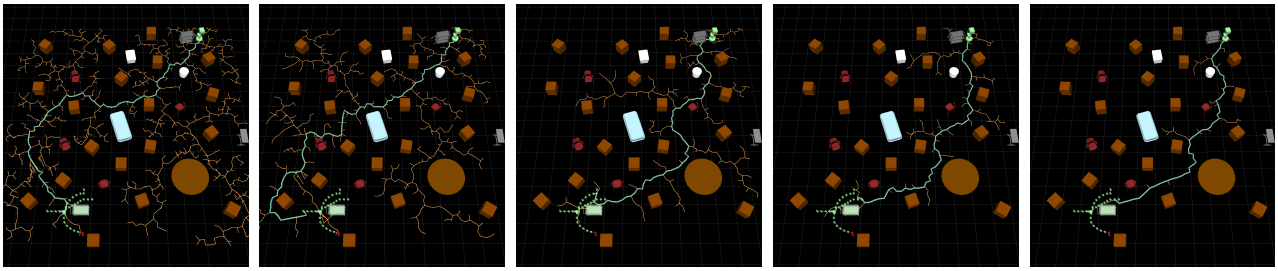
We conducted extensive experiments in the Webots [1] mobile robot simulation environment [22], where we modeled our omni-directional mobile manipulator robot and several passively-mobile, manipulable office and hospital objects shown in Figure 1. The robot has a footprint of roughly $0.27m$ in radius and it can navigate with a maximum translational velocity of $0.6m/s$ and a rotational velocity of $\pi/2rad/s$ in the simulated environments with dimensions $15m \times 15m$. Considering the dimensions of the task environments and the δ value used in the RRT expansion process (see Figure 2), which is set to be equal to the robot's radius, we placed a practical limit on the maximum number of nodes as 43200 for all of our generative planners, although we never observed the planners fail by exceeding that limit. RRT search process is terminated when the pose of the most recently added tree node gets within $0.05m$ distance and $\pi/36rad$ orientation difference limits to the pose of a (sub)goal.

For the acquisition of the fine reaching and manipulation sequences, we followed a LfD approach and joysticked the robot to demonstrate it how to pick up and place the manipulable objects available in the simulated environment. It would also be possible for the robot to acquire that experience through self exploration. Through these demonstrations, we provided the robot with four pick and four place sequences per OOI. Based on the dimensions of the objects in our task environments and the level of motion precision expected from the robot, we empirically decided to sparsify these sequences by defining entry points at every 30^{th} frame (i.e., $k = 30$). Pick up and placement of an OOI through the potential utilization of the sequences is considered successful if the robot (for pick up) or the OOI (for placement) gets within $0.05m$ distance and $\pi/36rad$ orientation difference limits to the target. The same tolerance values as the OOI pick up and placement are used for execution monitoring as we want to guarantee the robot to be within tolerable pose difference limits by the time it arrives at the target.

In utilizing the sequences for guiding the generative planning process, we set the bias for the entry points and the sequences to be $p_{ep} = p_s = 0.01$, which is smaller than the actual goal bias of $p_g = 0.05$, as the entry points and the sequences being greater in quantity provides the desired attraction to the fine manipulation region. Since there are only a few and relatively sparse sequences used for each OOI in our experiments, the total attraction probability of the feasible entry points, that is $|E|p_{ep}$, never exceeds 0.5.

Figure 7 provides a visual overview of the effects of various subgoal utilization strategies explained in Section 4.2 on the generated RRT branches (orange) as well as the solution paths (blue) for navigating the utility cart object from the upper right corner of the environment to the desired destination located at the bottom left corner. In these preliminary runs, the spread of the branches,

[1] http://www.cyberbotics.com



(a) No subgoals used. (b) Coincidental reach. (c) Sampling within goal. (d) Sampling sequences. (e) Sampling subgoals.

Figure 7. Different ways of combining the RRT generative planner and the sequences to achieve the task; (a) using none of the keyframes as subgoals, (b) early termination by coincidental reaching, (c) sampling subgoals within goal probability, (d) sampling sequences, and (e) sampling individual subgoals. The general tendency is observed to be towards a reduced number of generated nodes, hence shorter planning times, and an increased directness of the generated paths with the increased utilization of the entry points as subgoals.

hence the number of nodes, seems to decrease while the directness of the solution paths increases with the increased utilization of the sequences. This kind of effect is expected as the attraction of a fine manipulation region increases with the incorporation of more of the individual entry points into the planning guidance process. The numerical details of these five preliminary runs are provided in Table 1, where a correlation between the number of nodes and the planning time can be observed.

Subgoal Use	Nodes	Planning time (ms)
None	1196	84.108
Coincidental	632	42.851
Within goals	343	25.425
Sequences	216	17.208
Individual subgoals	119	11.977

Table 1. RRT planning statistics for various subgoal utilization strategies, the outcomes of which are visualized in Figure 7. 38 out of 41 subgoals were feasible in this scenario.

In order to thoroughly investigate how our experience-guided mobile manipulation approach performs under various conditions, we ran separate pick up and placement planning tests in five randomly configured task environments with four manipulable objects: a chair, an overbed table, a utility cart, and a stretcher (shown in Figure 1(a)). Due to the inherently random nature of the RRT-based algorithms, each of these tests were run 30 times for each combination of the three generative planners and the four subgoal utilization methods as well as the base case of planning only for reaching the actual goal (the "none" case). [2]

The box plots shown in Figure 8 present the planning performances obtained with each of these combinations for picking up and placing the chair object. Similar relative performances were observed for the other manipulable objects. Even though we present the number of nodes generated in the planning process as the measured metric, it is an implicit indicator of the relative time requirements of each of those combinations as the planning time decreases with the decreasing number of generated nodes in RRT-based planners, as previously shown in Table 1.

[2] Visualizations of the planning process with each planner and subgoal utilization strategy combination, where the shrinking effect of increased subgoal utilization on the spread of the generated trees and the solution paths can be seen in http://youtu.be/ePZYq41uTrA

Looking at these plots, we see that sampling individual subgoals and sampling sequences result in the best performances (i.e., least number of nodes), in that order, in all environments for all planners. On the other hand, sampling subgoals within goal probability has a varying relative performance depending on the environment and the planner. It performs the poorest in the majority of the task environments for the RRT and RRT* planners. Considering its definition, we see that this strategy actually decreases the probability of individual entry points being sampled since it essentially involves a two stage process, where the rarely occurring decision of sampling a goal is followed by which (sub)goal to sample. Therefore, instead of consistently directing the planner towards a (sub)goal in each rare occasion of deciding to sample a goal, it distracts the focus of the generative planner, diminishing the attractive properties of the fine manipulation region. In case the RRT-Connect planner is used, however, this strategy performs the third best in the majority of the task environments, though not significantly better than the base case. This effect can be explained by the way the RRT-Connect algorithm works, which is directly reaching the sample unless an obstacle is encountered. Therefore, once a (sub)goal is sampled, the algorithm tries to reach it along a straight line without any distraction or divergence. Coincidental termination has a comparable performance to the base case due to the very few number of sparse sequences, hence only a few alternative goals to reach coincidentally. Better performances with this strategy are observed when the number of sequences and their densities are increased.

Figure 9 provides sequence utilization statistics of the tests presented in Figure 8. The bars with the same colors as the ones in Figure 8 indicate in what percentage of the several planning trials with each combination the planner eventually ended up reaching an entry point. We see that sampling individual subgoals and sampling sequences have high sequence utilization percentages, which is expected. Even the coincidental termination strategy seems to be making decent use of the sequences. Although the percentages reported for the strategy of sampling subgoals within goal probability seems to be unexpectedly high, this situation has a perfect explanation. Since this strategy combines the actual goal with the set of subgoals and picks it with a probability of $p_{g^*} = 1/(|E| + 1)$, which is much smaller compared to the probability of

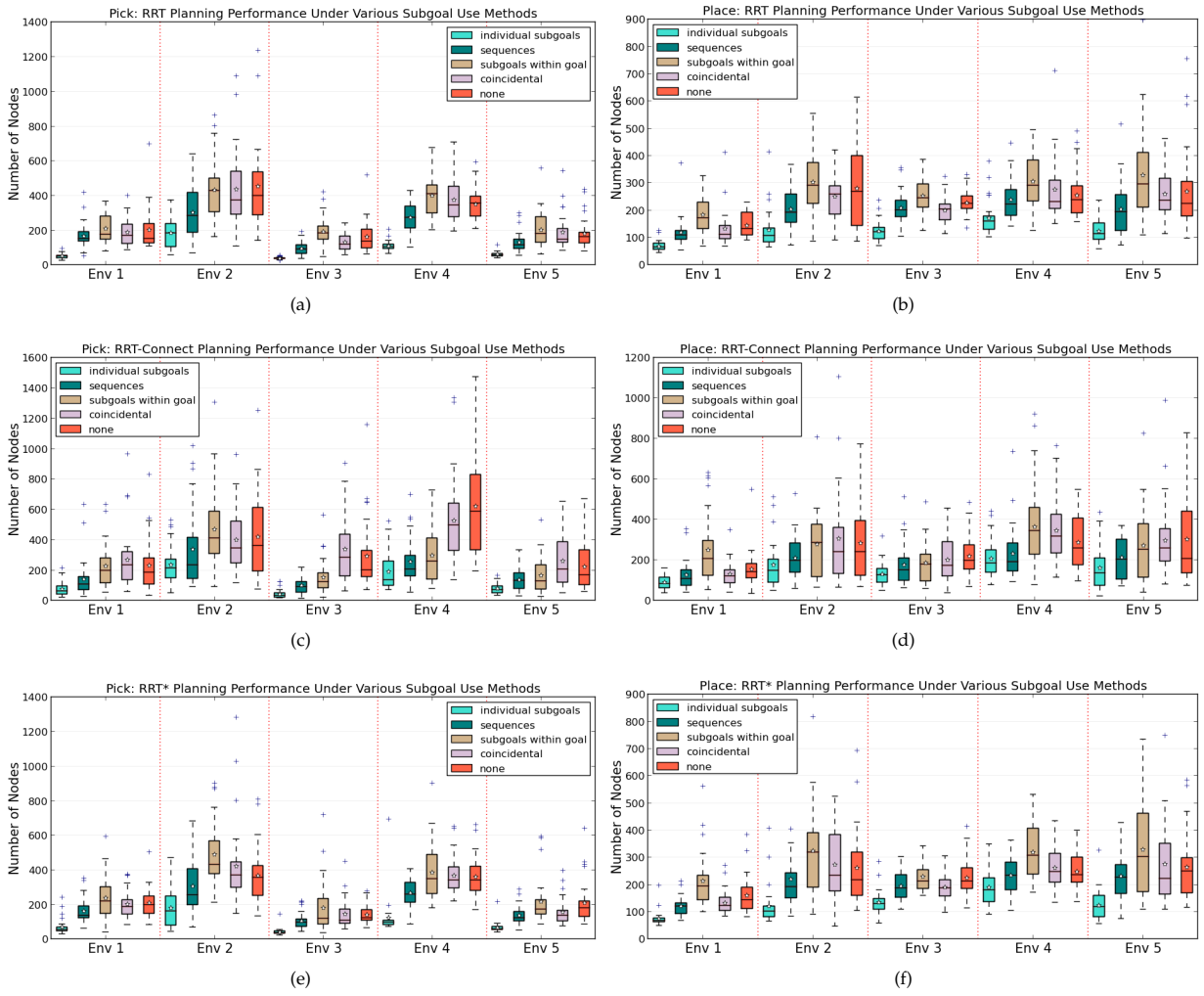


Figure 8. Planning performance for picking up ((a), (c), and (e)) and placing ((b), (d), and (f)) the chair object in five different task environments. The performance measure is the number of generated nodes by the RRT ((a) and (b)), the RRT-Connect ((c) and (d)), and RRT* ((e) and (f)) generative planners. The relative time requirements of each combination are implicitly reflected.

picking *any* subgoal with probability $(1 - p_{g^*})$, it is highly likely that the planner will end up reaching a subgoal rather than the actual goal.

Another experiment we conducted was to compare the effectiveness of utilizing every feasible segment of the sequences and hopping among them during reiteration to discarding the segments from the beginning of the sequences all the way to the last occluded part and utilizing only the remaining segments for planning and reiteration. We ran the complete task of picking up, navigating, and placing a particular OOI 10 times with each of the three generative planners for both cases, and measured the task completion times for each run. Table 2 provides the total number of entry points on the pick and the place sequences of the particular OOI used in this experiment as well as the amount utilized in different stages of the manipulation task, from where it can be inferred that the sequences were severely obstructed. Figure 10 illustrates task completion time statistics obtained for both scenarios. Looking at the mean values, marked with stars, we see that utilizing all feasible segments of the available sequences helps the robot perform better. The reason for greater variance is that sometimes the robot merges into a segment that is obstructed along the way to the goal and it has to hop to other feasible segments to proceed, whereas sometimes it ends up on an unobstructed segment of a sequence that leads it directly to the goal. On the other hand, when the blocked segments are discarded, the robot can only end up on a segment unobstructed all the way to the goal if the planner does not take it directly to the actual goal itself. When all feasible segments are utilized, due to the greater attraction effect created by the greater quantity of the entry points, the robot plans a more direct path quicker, hence gets to the fine manipulation region earlier, and completes the task in less time.

Operation	Entry points (Used / Total)	
	Segments utilized	Segments discarded
Pick	34 / 48	23 / 48
Place	18 / 39	13 / 39

Table 2. Number of entry points used when partial sequence segments are utilized versus when they are discarded.

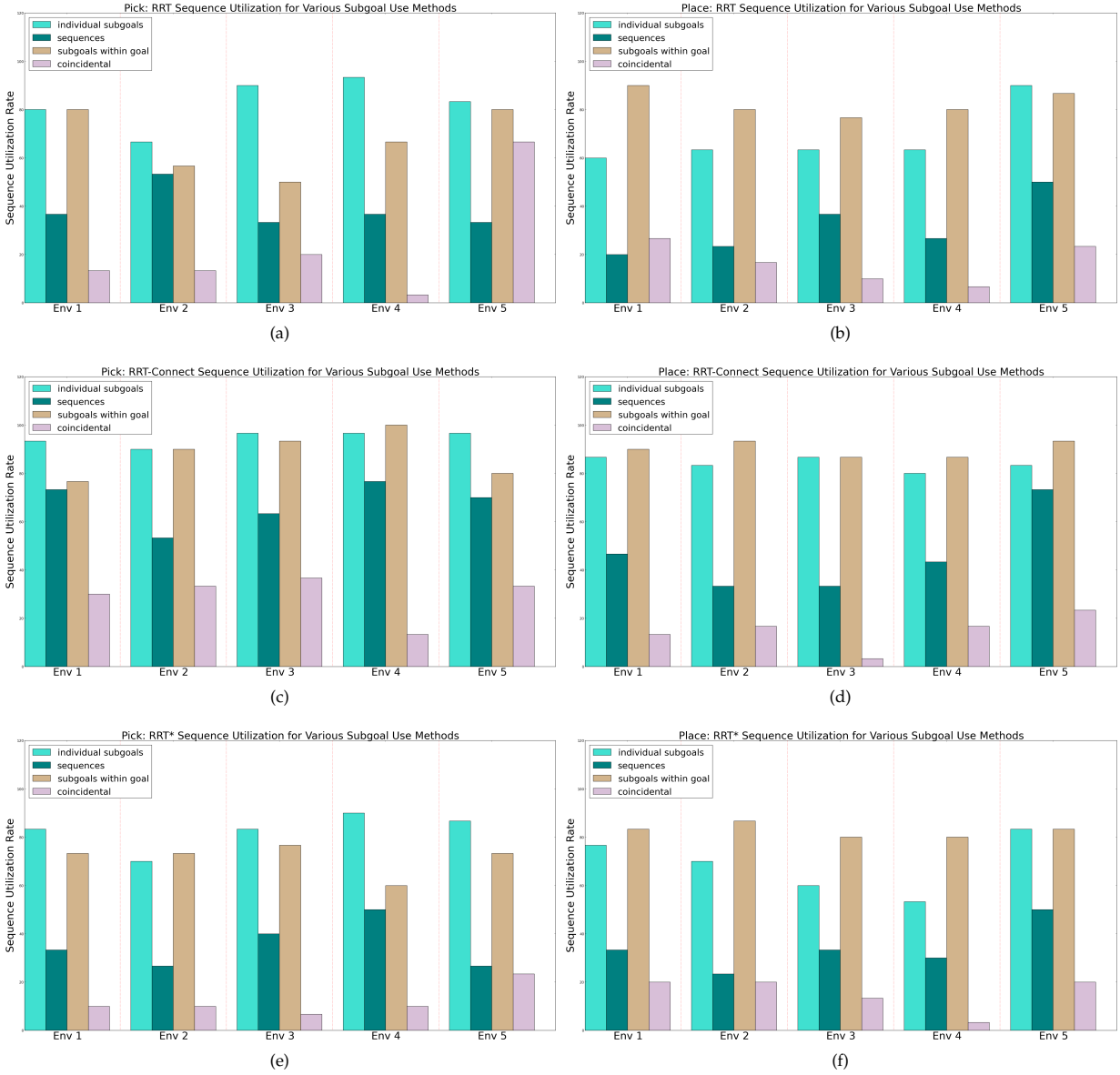


Figure 9. Sequence utilization while planning and executing pick ((a), (c), and (e)) and place ((b), (d), and (f)) tasks for the chair object in five different environments. Utilization percentage is measured for the RRT ((a) and (b)), the RRT-Connect ((c) and (d)), and RRT* ((e) and (f)) generative planners.

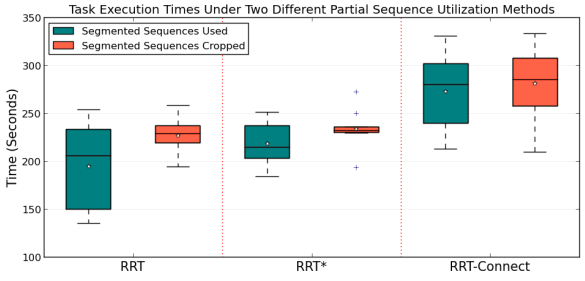


Figure 10. Durations for the complete pick and place task when the feasible segments of the partially occluded sequences are utilized in planning and execution versus when the segments from the beginning of the sequences to the occluded part are discarded.

The last experiment we conducted was to demonstrate the advantage of utilizing the attraction of a fine manipulation *region* over merely increasing the sampling probability of the single actual goal. For that purpose, we prepared a placement planning and execution scenario for the stretcher object where the direct path (i.e., line of sight) to the actual goal was blocked, as shown in Figure 11. In this scenario, a total of 42 out of 48 entry points were feasible.

We experimented with each combination of the three generative planners and the four subgoal utilization strategies as well as the base case with an increased goal bias. We ran 30 tests for each of these combinations. The sampling probability of the single actual goal used in the base case was set to be equal to the highest sampling probability of the fine manipulation region obtained with the individual subgoal sampling strategy through adding the sampling probabilities of individual subgoals p_{ep} on top of the sampling probability of the actual goal p_g, that is $(|E|p_{ep} + p_g)$. In our experiment scenario, where 42 feasible entry points are present, this value equals

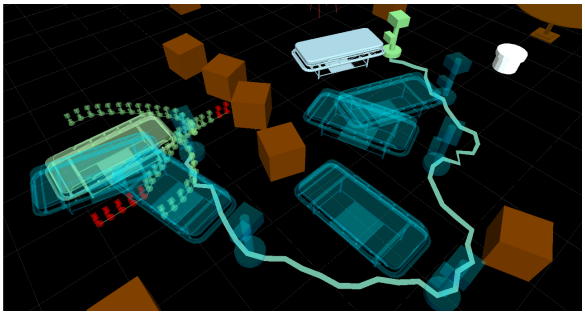


Figure 11. Visualization of the stretcher placement scenario where the direct path to the actual goal pose is blocked.

to $(42 \times 0.01 + 0.05 = 0.47)$; therefore, $p_g = 0.47$ for the base planning case. Figure 12 illustrates the relative performances of each of the combinations. Even in this challenging scenario, all of our experience-guided methods perform better than the base case planning strategy. In particular, the strategy of sampling individual subgoals results in the best performance. Even though the single goal of the base case planning strategy has the same attraction value as the entire fine manipulation region formed by the sequences, the entry points being distributed around the target helps the planner find its way around the obstacles easier and quicker.

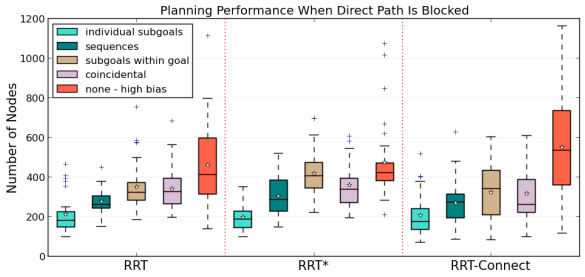


Figure 12. Planning performance using various subgoal utilization strategies versus merely increasing the goal bias in the scenario where the direct path to the goal is blocked (Figure 11).

6. Conclusion and Future Work

The delicacy and precision required by the pick and place activities performed in everyday living contexts may strain even the state-of-the-art planners, demanding significant amount of time and computational resources to generate successful solutions. However, careful examination of such tasks reveal recurring manipulation patterns, which usually occur within the close vicinity of the object and the destination. Fine manipulation within those regions is critical to the overall success of the task.

Motivated by this observation, we contribute an experience-based mobile manipulation method where the robot memorizes a few number of critical yet recurring target-specific fine reaching and manipulation moves as state-action sequences, and reuses them whenever possible to guide manipulation planning and execution. We show through extensive experimentation that this guidance helps reduce task completion times considerably when combined with a sampling-based generative planner, while increasing the chance of successful task completion by carefully reiterating previously executed and known-to-be-successful fine moves. Our approach harmoniously combines the already available partial plans and executions with the ones generated from scratch, yielding to fast, reliable, and repeatable solutions. [3]

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 monitoring the overall execution, accumulating new experiences over time, and transferring learned delicate reaching and manipulation sequences among objects with similar properties are some of the potential research problems to tackle in the future.

7. Acknowledgments

The first author was partly supported by The Scientific and Technological Research Council of Turkey under Programmes 2211 and 2214, and the Turkish State Planning Organization (DPT) under the TAM Project, number 2007K120610. This research was further supported by the National Science Foundation under grant number IIS-1012733, by the Office of Naval Research under grant number N00014-09-1-1031, and by the Air Force Research Laboratory under grant number FA87501220291. The views and conclusions contained herein are those of the authors only.

8. References

[1] Steven M. LaValle. Rapidly-Exploring Random Trees: A New Tool for Path Planning. Technical Report TR 98-11, Department of Computer Science. Iowa State University, 1998.

[2] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, New York, NY, USA, 2006.

[3] James J. Kuffner and Steven M. LaValle. RRT-connect: An efficient approach to single-query path planning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 995–1001, 2000.

[4] Sertaç Karaman and Emilio Frazzoli. Incremental Sampling-based Algorithms for Optimal Motion Planning. In *Proceedings of Robotics: Science and Systems (RSS)*, Zaragoza, Spain, June 2010.

[5] Anna Atramentov and Steven M. LaValle. Efficient nearest neighbor searching for motion planning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, volume 1, pages 632–637, 2002.

[6] Chris Urmson and Reid Simmons. Approaches for Heuristically Biasing RRT Growth. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 2, 2003.

[7] James Bruce and Manuela Veloso. Real-time randomized path planning for robot navigation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pages 2383–2388, 2002.

[8] Manuela M. Veloso. *Planning and Learning by Analogical Reasoning*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1994.

[3] An example video showing the robot performing experience-guided pick and place can be seen here: http://youtu.be/IUzffcQ15WU

[9] Manuela M. Veloso. Flexible Strategy Learning: Analogical Replay of Problem Solving Episodes. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, volume 1 of *AAAI '94*, pages 595–600, Menlo Park, CA, USA, 1994. American Association for Artificial Intelligence.

[10] Manuela M. Veloso. Merge strategies for multiple case plan replay. In David B. Leake and Enric Plaza, editors, *Case-Based Reasoning Research and Development*, volume 1266 of *Lecture Notes in Computer Science*, pages 413–424. Springer Berlin Heidelberg, 1997.

[11] Benjamin J. Cohen, Gokul Subramania, Sachin Chitta, and Maxim Likhachev. Planning for Manipulation with Adaptive Motion Primitives. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 5478–5485, 2011.

[12] Dmitry Berenson, Pieter Abbeel, and Ken Goldberg. A Robot Path Planning Framework that Learns from Experience. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3671–3678, 2012.

[13] Brenna D. Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A Survey of Robot Learning from Demonstration. *Robotics and Automation Systems*, 57(5):469–483, 2009.

[14] Alexander Skoglund, Boyko Iliev, Bourhane Kadmiry, and Rainer Palm. Programming by Demonstration of Pick-and-Place Tasks for Industrial Manipulators using Task Primitives. In *International Symposium on Computational Intelligence in Robotics and Automation, 2007. CIRA 2007*, pages 368–373, 2007.

[15] Gu Ye and Ron Alterovitz. Demonstration-guided Motion Planning. In *Proceedings of the International Symposium on Robotics Research (ISRR)*, August 2011.

[16] Hiroaki Sakoe and Seibi Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 26(1):43–49, Feb 1978.

[17] Carolyn R. Mason, Jose E. Gomez, and Timothy J. Ebner. Hand synergies during reach-to-grasp. *Journal of Neurophysiology*, 86(6):2896–2910, 2001.

[18] Brigitte Bartsch-Spörl, Mario Lenz, and André Hübner. Case-Based Reasoning - Survey and Future Directions. In *Proceedings of the 5th German Biennial Conference on Knowledge-Based Systems*, pages 67–89. Springer Verlag, 1999.

[19] Luca Spalazzi. A Survey on Case-Based Planning. *Artificial Intelligence Review*, 16:3–36, 2001. 10.1023/A:1011081305027.

[20] Raquel Ros, Josep Lluís Arcos, Ramon Lopez de Mantaras, and Manuela Veloso. A Case-based Approach for Coordinated Action Selection in Robot Soccer. *Artificial Intelligence*, 173:1014–1039, June 2009.

[21] Ola Pettersson. Execution Monitoring in Robotics: A Survey. *Robotics and Autonomous Systems*, 53:73–88, 2005.

[22] Olivier Michel. Webots: Professional Mobile Robot Simulation. *Journal of Advanced Robotics Systems*, 1(1):39–42, 2004.