CASE-BASED MOBILE MANIPULATION

by

Tekin Meriçli

B.S., Computer Engineering, Marmara University, 2005

M.S., Computer Science, The University of Texas at Austin, 2007

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy

Graduate Program in Computer Engineering

Boğaziçi University

2014

ACKNOWLEDGEMENTS

Completing this thesis has been a long journey made possible by the unconditional help of many people to whom I owe my sincere thanks.

First and foremost, I am grateful to my advisors Levent Akın and Manuela Veloso for their guidance and support; this thesis would not have been possible without them. Levent Akın has played a key role in my becoming a roboticist by letting me be a part of the "Cerberus" RoboCup team as a junior undergraduate student. Him encouraging me to explore various research ideas and directions helped me build my broad experience in all aspects of intelligent robotics. Manuela Veloso deserves special thanks as she has been absolutely wonderful in giving me brilliant ideas, setting the boundaries of my research, keeping me on track, and helping me complete my thesis in a finite amount of time. I owe a lot to Manuela.

I would also like to thank the members of my committee, Ethem Alpaydın, Olcay Kurşun, Albert Ali Salah, and Hatice Köse for their constructive criticism and valuable feedback throughout the progression of this thesis.

I thank all my friends from CmpE of Boğaziçi University, who made my time there extremely fun and fruitful. CmpE has a unique atmosphere and I am very proud to be affiliated with it. Being a member of the Robotics Research Group at the AI Lab of CmpE in general and the Cerberus team in particular has been truly rewarding in all aspects.

I would also like to thank all my friends from the CORAL research group at Carnegie Mellon University, who made my brief visit to Carnegie Mellon a very pleasant and intellectually satisfying experience. Their invaluable feedback helped me improve both the content of my work and the way I present it.

My brother has been a spectacular role model for me and I have always followed his footsteps. Neither this thesis nor any of my other significant achievements would have been possible without his support and encouragement. “2 Meriçli” shall prevail.

My parents, Birgül Meriçli and İsmet Meriçli, deserve the greatest thanks of all. They have made so many sacrifices and supported me and my brother in every possible way throughout our education. My father, in particular, was the main source of inspiration for both me and my brother to choose and pursue a science and technology oriented career path. We miss him greatly.

Finally, I would like to thank my lovely wife Andrea for being extremely patient and supporting through the ups and downs of graduate school. I appreciate everything she has done for me.

Parts of this thesis study were supported by The Scientific and Technological Research Council of Turkey Programme 2214 and the Turkish State Planning Organization (DPT) under the TAM Project, number 2007K120610.

ABSTRACT

CASE-BASED MOBILE MANIPULATION

The ability to manipulate the environment is one of the primary skills that autonomous mobile service robots are expected to have, considering that the daily lives of humans heavily rely on this skill. There are various ways for a mobile robot to perform manipulation, the exact form of which is determined by the requirements of the task and the constraints imposed by the physical properties of the environment, the object, and the robot itself. Anecdotal evidence suggests that humans mostly reuse their manipulation experiences, acquired through interaction and observation, especially in recurring everyday manipulation tasks, both in prehensile and non-prehensile manipulation contexts. With this motivation, this thesis contributes a case-based approach to achieving practical and efficient mobile manipulation through the utilization of past experience, stored as object-specific, distinct, and potentially probabilistic *cases*. In scenarios where prehensile manipulation is possible, this guidance combined with sampling-based generative planners helps reduce planning time by deliberately biasing the planning process towards the feasible cases while increasing the overall robustness and repeatability of the method. When non-prehensile manipulation techniques, such as push-manipulation, need to be utilized, these probabilistic cases can be used as building blocks for constructing safe and *achievable* push plans to navigate the object of interest to the desired goal pose as well as to potentially push the movable obstacles out of the way in cluttered task environments. Additionally, incremental acquisition and tuning of the probabilistic cases allows the robot to adapt to the changes in the environment, such as increased mass due to loading of the object of interest for transportation purposes. The purely interaction and observation driven nature of our method makes it robot, object, and environment (real or simulated) independent, as we demonstrate through extensive testing and experimentation. We also verify the validity of our push-manipulation method in preliminary real world tests.

ÖZET

DURUM TABANLI HAREKETLİ MANİPÜLASYON

Ortamı manipüle edebilme, insanların günlük yaşamlarının önemli ölçüde bu yetiye dayandığı düşünülürse, özerk hareketli servis robotlarının sahip olması beklenen özelliklerin başında gelmektedir. Tam formu görevin gereksinimleri ile ortamın, nesnenin, ve robotun fiziksel özelliklerinden kaynaklanan kısıtlarca belirlenmekle birlikte, hareketli bir robot pek çok şekilde manipülasyon yapabilir. Çeşitli incelemeler insanların günlük manipülasyon görevlerinde çoğunlukla etkileşim ve gözlem sonucu elde ettikleri manipülasyon tecrübelerinden faydalandıklarını göstermektedir. Bu motivasyonla, bu tez çalışması, pratik ve etkili hareketli manipülasyonun nesneye özgü, ayrık, ve potansiyel olarak olasılıksal *durumlar* şeklinde saklanan geçmiş tecrübelerin yeniden kullanılması ile başarıldığı durum temelli bir yaklaşım sunmaktadır. İlgilenilen nesnenin kavranabildiği senaryolarda, bu kılavuzluk, örnekleme tabanlı planlama algoritmalarıyla birleştirildiğinde yöntemin gürbüzlüğünü ve tekrarlanabilirliğini arttırırken, planlama sürecini kasıtlı olarak uygulanabilir durumlara doğru yönlendirerek planlama zamanını azaltmaya yardımcı olmaktadır. İterek-manipülasyon gibi, nesnenin kavranmadan manipüle edilmesini gerektiren tekniklere ihtiyaç duyulduğunda, bu olasılıksal durumlar, ilgilenilen nesneyi istenen hedef konuma taşıyabilmek ve sıkışık görev ortamlarında potansiyel olarak karşılaşılabilecek engelleri yoldan çekebilmek için gereken güvenli ve *başarılabilir* itme planlarının yapıtaşları olarak kullanılabilirler. Buna ek olarak, olasılıksal durumların artımsal olarak edinilip iyileştiriliyor olmaları, robotun ilgilenilen nesnelerin nakliye amacıyla yüklenmeleri ya da boşaltılmaları sonucu kütlelerinin değişmesi gibi durumlara uyum sağlayabilmesine imkan tanımaktadır. Çok geniş kapsamlı deneylerle gösterdiğimiz üzere, tamamen etkileşim ve gözleme dayalı doğası, sunduğumuz yöntemi robottan, nesneden, ve ortamdan (gerçek ya da benzetim) bağımsız hale getirmektedir. Ayrıca geliştirdiğimiz iterek-manipülasyon yönteminin geçerliliğini bir takım öncü gerçek dünya deneyleriyle de doğrulamaktayız.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS iii
ABSTRACT v
ÖZET vi
LIST OF FIGURES ix
LIST OF TABLES xiv
LIST OF SYMBOLS xvi
LIST OF ACRONYMS/ABBREVIATIONS xviii
1. INTRODUCTION 1
1.1. Approach 3
1.1.1. Case-based Pick and Place Manipulation 4
1.1.2. Case-based Push Manipulation 5
1.2. Contributions 7
1.3. Evaluation 7
1.4. Thesis Outline 8
2. MOBILE PICK AND PLACE MANIPULATION 9
2.1. Motivation 9
2.2. Background 11
2.3. Related Work 13
2.4. Case-Based Mobile Pick and Place 15
2.4.1. Cases for Delicate Reaching and Manipulation 16
2.4.2. Generative Planner 20
2.4.3. Execution Monitoring 24
2.5. Experimental Evaluation 25
2.6. Discussion 36
3. MOBILE PUSH-MANIPULATION 38
3.1. Motivation 38
3.2. Related Work 40
3.3. Case-based Mobile Push-Manipulation 43
3.3.1. Probabilistic Cases for Push-Manipulation 44

3.3.2. Building Experimental Interaction Models 46
3.4. Case-Based Reactive Push-Manipulation 49
3.4.1. Experimental Evaluation 51
3.5. Case-Based Achievable Push-Manipulation 54
3.5.1. Execution Monitoring 57
3.5.2. Experimental Evaluation 59
3.5.3. Mobile Push-Manipulation in Real World 65
3.6. Discussion 67
4. ADAPTING PAST EXPERIENCE TO NOVEL SITUATIONS 69
4.1. Motivation 69
4.2. Related Work 70
4.3. Approach 70
4.4. Experimental Evaluation 71
4.5. Discussion 73
5. MOBILE PUSH-MANIPULATION AMONG MOVABLE OBSTACLES . . . 74
5.1. Motivation 74
5.2. Related Work 75
5.3. Push-Manipulation Among Movable Obstacles 75
5.4. Experimental Evaluation 76
5.5. Discussion 79
6. CONCLUSIONS AND FUTURE WORK 80
6.1. Case-Based Mobile Pick and Place 80
6.2. Case-Based Mobile Push-Manipulation 81
6.3. Contributions 82
6.4. Future Research Directions 83
APPENDIX A: BUILDING REALISTIC SIMULATION ENVIRONMENTS . 84
A.1. Simulated Robot and Object Models 84
A.2. Robot Kinematics and Control 86
APPENDIX B: REAL WORLD TEST SETUP 89
APPENDIX C: ADDITIONAL EXPERIMENTAL RESULTS 92
C.1. Experience-based Mobile Pick and Place 92
REFERENCES 108

LIST OF FIGURES

Figure 1.1. Carnegie Mellon's CoBot-2 robot and its simulated model. 3

Figure 2.1. The simulated mobile manipulator, several office and hospital objects, and a randomly populated cluttered task environment. . . . 10

Figure 2.2. Tree construction process of the RRT algorithm. 11

Figure 2.3. The basic RRT algorithm. 12

Figure 2.4. Visualization of pick and place sequences associated with the utility cart object and its potential destination. 17

Figure 2.5. Visualization of the sequences around the chair object and their *entry points* depicted as scaled-down ghost robot figures. 19

Figure 2.6. Reiteration of a collision-free sequence. 20

Figure 2.7. Configuration sampling function that utilizes the sequences. 23

Figure 2.8. Reiteration of an obstructed sequence. 24

Figure 2.9. Different ways of combining the RRT generative planner and the sequences to achieve the task. 27

Figure 2.10. The statistics for the number of generated nodes during planning for picking up the chair object. 29

Figure 2.11. Planning time statistics for picking up the chair object. 30

Figure 2.12. The statistics for the number of generated nodes during planning for placing the chair object. 31

Figure 2.13. Planning time statistics for placing the chair object. 32

Figure 2.14. The effect of fragmented segment utilization on task completion time. 34

Figure 2.15. Visualization of the stretcher placement scenario where the direct path to the actual goal pose is blocked. 35

Figure 2.16. Statistics for the number of plan tree nodes in the scenario where the direct path to the goal is blocked. 36

Figure 2.17. Planning time statistics for the scenario where the direct path to the goal is blocked. 36

Figure 3.1. Our simulated CoBot model together with a set of realistically simulated passively-mobile objects. 39

Figure 3.2. Various contacts between the robot and the object of interest. . . 43

Figure 3.3. Depictions of the reference frames used during sequence acquisition and reiteration, and the sequences acquired for the chair object. . 45

Figure 3.4. Actual distribution of the object's relative final pose (location and orientation) when one of the sequences is reiterated several times. 48

Figure 3.5. Approximation of the object trajectories with vectors. 50

Figure 3.6. The four different sets of sequences used in the reactive push-manipulation experiments. 53

Figure 3.7. The Exp-RRT algorithm. 55

Figure 3.8. Illustration of the Exp-RRT construction process. The sequences resulting in the most similar poses are highlighted. 56

Figure 3.9. Illustration of the sigma points. 57

Figure 3.10. Collision-free and achievable push-plans generated for various pushable objects using their corresponding probabilistic cases. 60

Figure 3.11. The effects of the growing variety of the available sequences for the chair object on the path length and its consistency. 62

Figure 3.12. The tendency towards a decreasing number of re-plans during task execution with the increasing number of practices per individual case of a particular object of interest. 63

Figure 3.13. The tendency towards an increasing plan success rate with the increasing number of cases. 63

Figure 3.14. The number of cases versus the number of expanded Exp-RRT nodes. 64

Figure 3.15. A snapshot from one of the real world tests, where the task of the robot is to arrange the chairs around the round table. 66

Figure 3.16. Snapshots from the case-based push-manipulation planning and execution test in a real world setup. 67

Figure 4.1. The effect of the changes in the dynamics of the object on the its behavior. 69

Figure 4.2. The effect of continuous adaptation to the observed changes on the robust execution of the generated plans. 72

Figure 5.1. The effect of movable obstacle utilization during planning. 74

Figure 5.2. State lattice construction for the movable obstacle. 76

Figure 5.3. Clearing movable obstacles along the path of the stretcher. 77

Figure 5.4. Planning performance in the presence of movable obstacles. 78

Figure A.1. Realistically simulated models of some common office and service objects, such as a chair, an overbed table, and a utility cart. . . . 85

Figure A.2. A standard generic caster wheel and its simulated counterpart. . . 85

Figure A.3. Simulated model of the omni-directional wheel with a lower resolution of rollers compared to the wheels of the physical platform. . 86

Figure A.4. Simulated CoBot base and its kinematics. 87

Figure B.1. The visualization of the AR Tag tracking process. 89

Figure B.2. Various reference frames used in computing the location of the object's center with respect to the robot. 90

Figure B.3. Visualizations of the Fast Sampling Plane Filtering based planar feature detection and Corrective Gradient Refinement based localization algorithms. 91

Figure C.1. The statistics for the number of generated nodes during planning for picking up the overbed table object. 93

Figure C.2. Planning time statistics for picking up the overbed table object. . 94

Figure C.3. The statistics for the number of generated nodes during planning for placing the overbed table object. 95

Figure C.4. Planning time statistics for placing the overbed table object. . . . 96

Figure C.5. The statistics for the number of generated nodes during planning for picking up the utility cart object. 98

Figure C.6. Planning time statistics for picking up the utility cart object. . . . 99

Figure C.7. The statistics for the number of generated nodes during planning for placing the utility cart object. 100

Figure C.8. Planning time statistics for placing the utility cart object. 101

Figure C.9. The statistics for the number of generated nodes during planning for picking up the stretcher object. 103

Figure C.10. Planning time statistics for picking up the stretcher object. 104

Figure C.11. The statistics for the number of generated nodes during planning for placing the stretcher object. 105

Figure C.12. Planning time statistics for placing the stretcher object. 106

LIST OF TABLES

Table 2.1. RRT planning statistics for various subgoal utilization methods. . . 27

Table 2.2. Sequence utilization while planning and executing pick task for the chair object in ten different environments. 33

Table 2.3. Sequence utilization while planning and executing place task for the chair object in ten different environments. 33

Table 2.4. Number of entry points used when partial sequence segments are utilized versus when they are discarded. 34

Table 3.1. Performances of different sets of motion sequences. 52

Table 5.1. Task completion statistics for stationary and movable obstacle configurations. 79

Table C.1. Sequence utilization while planning and executing pick task for the overbed table object in ten different environments. 97

Table C.2. Sequence utilization while planning and executing place task for the overbed table object in ten different environments. 97

Table C.3. Sequence utilization while planning and executing pick task for the utility cart object in ten different environments. 102

Table C.4. Sequence utilization while planning and executing place task for the utility cart object in ten different environments. 102

Table C.5. Sequence utilization while planning and executing pick task for the stretcher object in ten different environments. 107

Table C.6. Sequence utilization while planning and executing place task for the stretcher object in ten different environments. 107

LIST OF SYMBOLS

$a_{j\in[0,n)}$	Motion command component of a sequences of length n.
d_{max}	The maximum distance in the task environment.
E	Set of all feasible entry points.
E_{S_i}	Set of feasible entry points that belong to sequence S_i.
k	Keyframe/entry point frequency along a sequence.
K	Scaling factor used for adjusting the weight of the past experience over the newly observed outcome.
L	Dimensionality of the state space.
M	Total number of cases stored for a particular object of interest.
p	Probability of picking the goal configuration as a sample.
p_{ep}	Probability of sampling an individual entry point.
p_g	Probability of sampling the actual goal configuration.
p_{g^*}	Probability of picking the actual goal after deciding to sample a goal configuration.
p_s	Probability of sampling a sequence.
q_{init}	Initial configuration before planning.
q_{near}	Nearest configuration on the plan tree to the picked sample.
q_{new}	New configuration added to the plan tree.
q_{rand}	Random configuration picked during planning.
S	Set of sequences.
$S_{i\in[0,M)}$	Set of M sequences.
S^{new}	Set of newly acquired sequences.
δ	Length of the extension towards the sample configuration from the nearest node of the plan tree.
δ_{max}	The threshold for the maximum allowed final distance to the goal pose.
ε	Maximum allowed difference between the expected and the actual pose during sequence reiteration.
ζ	The scalar scaling factor that determines the spread of the sigma points around the mean.

$\mathcal{G}$	Direction vector defining the next waypoint along the guideline path.
ι	Set of index increments used for picking sequences during the push uncertainty modeling process.
$\wp_O$	Pose of the object relative to its last stationary pose.
$\wp_{O_g}$	The goal pose for the object.
$\wp_{O_o}$	The observed final pose of the object.
$\wp_R$	Pose of the robot relative to the target.
$\bar{\wp}_O$	The expected final pose of the object.
$\bar{\wp}_{O^f}$	The mean of the final object poses observed so far for a particular sequence.
φ_G	Global reference frame.
φ_O	Object's reference frame, defining its global pose.
φ_R	Robot's reference frame, defining its global pose.
φ_S	Auxiliary reference frame, defining the last stationary pose of the object before it starts being pushed.
φ_T	Target's reference frame, defining its global pose.
ς	A push configuration defining the push initiation pose of the robot and the duration of the pushing action.
$\Sigma_{\wp_O}$	The expected final pose covariance of the object.
$\Sigma_{\wp_{O_t^i}}$	Covariance of the observed object pose after reiterating push sequence i for a total of t times.
τ_i	Direction vector used for approximating the object's trajectory resulting from sequence i.
$\chi^2_k(p)$	The quantile function for probability p of the chi-squared distribution with k degrees of freedom.
ω_{max}	The threshold for the maximum allowed final orientation difference with the goal pose.
v_x	Translational velocity in x direction.
v_y	Translational velocity in y direction.
v_θ	Rotational velocity.

LIST OF ACRONYMS/ABBREVIATIONS

CBP	Case-based Planning
CBR	Case-based Reasoning
ERRT	Execution Extended Rapidly-exploring Random Tree
Exp-RRT	Experience-based Rapidly-exploring Random Tree
hRRT	Heuristically-guided Rapidly-exploring Random Tree
LfD	Learning from Demonstration
NAMO	Navigation Among Movable Obstacles
ODE	Open Dynamics Engine
OOI	Object of Interest
RRT	Rapidly-exploring Random Tree
TSR	Task Space Region

1. INTRODUCTION

Mobile manipulation constitutes a significant portion of the daily activities that humans perform in potentially unstructured, dynamic, and cluttered environments. As a result, the majority of the tasks that people expect their service robots to be able to handle involve some form of manipulation in environments with similar characteristics. There are several ways for a robot to manipulate the objects in its task environment. The requirements of the task and the constraints imposed by the physical properties of the environment, the object, and the robot determine which of the two main modalities of manipulation is more suitable for a given scenario:

- If the robot is equipped with a grasping mechanism and the object satisfies the payload constraints of the manipulator in terms of its weight and dimensions, *prehensile* manipulation could be the better option, as grasping the object first and unifying with it would provide more control over the object. The most common application of this type of manipulation is pick and place tasks.
- In cases where the object is too large or heavy, the robot is not equipped with a manipulator arm and/or a grasping unit, or it is more convenient to transport the object that way, *non-prehensile* manipulation, that is manipulating objects without grasping them, could be the better option. Examples of non-prehensile manipulation include sliding, rolling, throwing, and *pushing*.

If the robot is equipped with a grasping mechanism, the object needs to be approached in a certain and delicate way to obtain a reliable grasp so that the rest of the manipulation task can be executed successfully. In theory, there may exist an infinite number of such fine manipulation trajectories. However, in practice, we observe that humans repetitively utilize only a small set of these object-specific pre-grasp approach trajectories during everyday manipulation tasks [1]. Analogously, in robotic manipulation, these moves can be acquired only once either through demonstration or self-exploration and stored for future reuse instead of generating them from scratch every time the object needs to be manipulated. Considering that the majority of the

complexity in motion planning stems from trying to satisfy the tight constraints that define those delicate moves, storing and reusing them could help reduce the load on the generative motion planner, resulting in faster solutions and more reliable executions.

If, on the other hand, the robot is not equipped with a grasping mechanism, the objects in the task environment may still be manipulated through some other forms of interaction, such as pushing. In that case, a separate path should be planned and tracked for the object as it does not become a part of the robot during transportation. Planning such paths requires knowledge about how the object of interest moves in response to various pushes. The object's exact behaviors in response to these interactions are defined by several physical properties such as the object's mass distribution, its passive-mobility configuration, ground friction characteristics, and how and where the robot contacts the object during the push. While, in theory, it is possible to precisely measure all these properties and derive an analytical model of the effects of the pushes on the object, in practice, it is often non-trivial to obtain these measurements with enough accuracy to be able to deterministically predict the outcome of a particular push. Also, it is not feasible to provide such analytical models for every possible object, robot, and environment combination. Humans handle such problems flexibly due to their abilities to build observation-driven experimental interaction models and use them for predictive planning purposes. Robots can follow a similar approach to acquire that knowledge by observing the outcomes of various interactions with the objects of interest, and utilize those experimental models together with their associated uncertainties for push-manipulation planning and execution in their task environments.

Given these motivations, this thesis explores the following principal questions:

- Can we reuse fine pre-grasp trajectories from successful past attempts to improve manipulation planning and execution performance in mobile pick and place tasks?
- Can we reuse experimentally acquired observation-driven probabilistic interaction models as push primitives during the construction and execution of mobile whole-body push-manipulation plans?

1.1. Approach

Motivated by the aforementioned anecdotal evidence on how humans manipulate objects, this thesis contributes case-based methods to achieve practical and efficient mobile manipulation through the utilization of past experience, stored as object-specific, distinct, and potentially probabilistic *cases*. The presented concepts are independent of the actual setup; hence, can be applied to various prehensile and non-prehensile manipulation scenarios with even higher dimensional state spaces. However, in the scope of this thesis, we concentrate on mobile manipulation in indoor environments, such as offices and hospitals, where bulky objects need to be transported safely over long distances. For that purpose, we employ our CoBot [2] service robot as the whole-body manipulator in both real and simulated task environments, as shown in Figure 1.1.

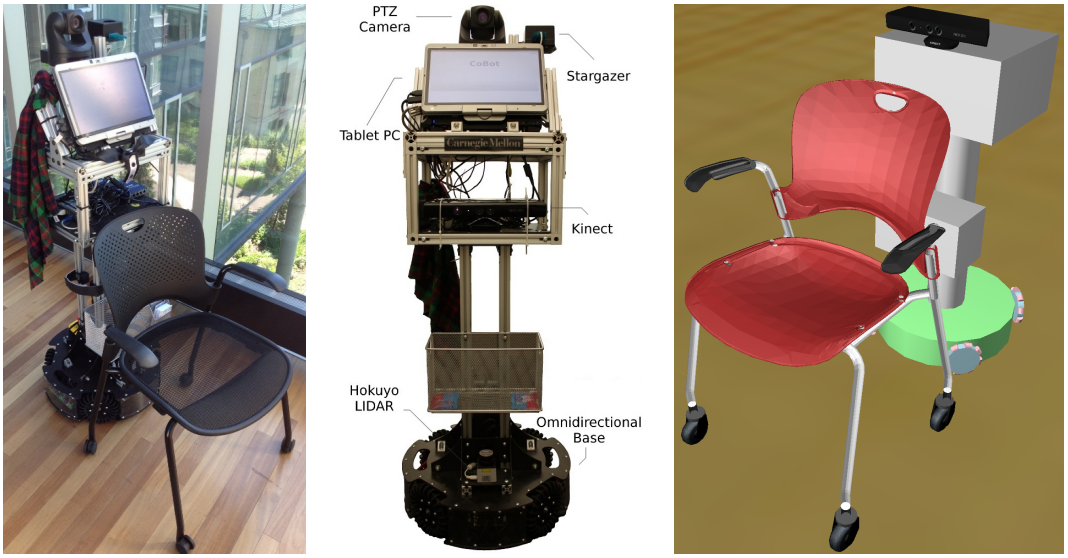


Figure 1.1. Carnegie Mellon's CoBot-2 robot and its simulated model.

By definition, case-based reasoning (CBR) and planning (CBP) systems handle new problems by reusing plans and solutions applied to similar cases in the past instead of planning from scratch [3, 4]. A traditional CBR/P cycle consists of four stages:

- Retrieve: Given a new problem, similar cases are retrieved from the memory.
- Reuse: Retrieved solution for the most similar case is adapted for the new case.
- Revise: Adapted solution is tested against the new case and revised as needed.
- Retain: The new solution is potentially added to the case base of solutions.

Getting inspiration from the *controllable state features* concept presented by Ros *et al.* [5], when applying the case-based paradigm to robotic manipulation, we acknowledge and utilize the fact that the robot can, to some extent, alter the state of the environment to fit its needs. For instance, it can reposition itself around the target of interest such that the available cases can be directly applied. This flexibility enables us to define the cases relative to the targets of interest (i.e. the object and/or the destination), making them invariant to the global poses of the robot and the targets.

1.1.1. Case-based Pick and Place Manipulation

In pick and place manipulation scenarios, we define a case as a successful fine approaching trajectory that the robot executes within close proximities of the object (for pick up) or the destination (for placement). This representation aligns very well with our observation of humans repeatedly using a finite number of alternatives for approaching a particular object and/or its destination, and allows us to store a small set of frequently reused local solutions. These stored cases serve mainly two purposes;

- implicitly forming a *critical manipulation region* around the target that can be used to guide the generative planning process to reach that region as roughly and directly as possible, hence alleviating the computational load of trying to satisfy fine reaching constraints, which is already handled by the stored cases.
- providing a way to reiterate previously experienced and known-to-be-successful fine approaching solutions, increasing the reliability of the execution.

Our extensive experimental analyses demonstrate that this approach reduces the overall computational demand for generative planning, hence improving efficiency compared to planning from scratch every time the objects of interest need to be manipulated. Also, reiterating the moves that have been executed in the past and are known-to-be-successful improves execution reliability.

1.1.2. Case-based Push Manipulation

In our push-manipulation scenarios, we define a case by the active push trajectory executed by the robot and the corresponding passive trajectory that the object follows. However, usually the passive object trajectories are not exactly repeatable; that is, even if the action performed by the robot remains the same, the object may trace a slightly different trajectory and end up at a different pose after every interaction, mainly due to the following reasons.

- Due to the complex 3D structures of the robot and the object, it is not easy to formulate a deterministic model for the outcomes of their potential interactions.
- Our objects move on passive caster wheels, which contribute to the motion uncertainty as the objects continue moving for some time even after the push is ceased and their trajectories are affected by the initial wheel orientations.

Therefore, it becomes critical to incorporate that uncertainty into the case definition. We address this problem by introducing the *probabilistic case* concept, where the outcome of a robot-object interaction is associated with a probability distribution. These distributions are formed incrementally as the robot interacts with the objects, through either human demonstration or self-exploration. They are mainly utilized for collision checking purposes to guarantee the safety and the *achievability* of the plan from both the robot's and the object's perspective.

An object can be push-manipulated through the utilization of these experimentally acquired probabilistic cases by following one of the strategies described below.

(i) A collision-free path is planned for the object without taking the actual interaction models into account, and the robot tries to track this path by reiterating the best locally-matching object trajectories until the goal is reached.

(ii) The push plan is constructed directly out of the experimentally acquired probabilistic interaction models such that its achievability and safety are guaranteed.

The first strategy follows the traditional case-based approach where the robot tries to solve the given (local) problem by picking the best matching case from its case base. However, since there are only a small number of available cases for the object, there is no guarantee that the path will be tracked successfully all the way to the goal. The second strategy, on the other hand, provides such guarantee as the cases themselves constitute the individual building blocks of the constructed push-plan. Our extensive experimental evaluation demonstrate the successful application of the second strategy in various challenging push-manipulation problems in simulated large scale, cluttered environments as well as in a real world setup.

The objects of interest in our manipulation scenarios are mainly office and hospital service objects and furnitures; therefore, it is very likely for their occupancy states to change from time to time by loading them with additional objects for transportation purposes. We demonstrate the advantage of having a purely interaction and observation driven case acquisition and tuning approach in adapting the past experience to the novel situations, such as changes in the dynamics of the manipulated objects due to loading/unloading, instead of having to go through the learning process all over for each such novelty. Our experimental results show that, after adaptation, the cases acquired for unloaded objects of interest could be successfully used for manipulating loaded ones, which have substantially different dynamics.

Finally, we investigate ways of utilizing the available manipulation experience for the objects other than the primary object of interest for relaxing the planning process, as the accumulation of push-manipulation experience for various objects enables the robot to modify and rearrange the task environment according to its needs. Every movable object that the robot has experience about can be treated as "permeable" during the construction of push plans for the primary object of interest. That way, the complexity of the constraints imposed by the task environments can be reduced considerably. Also, some problems that are unsolvable when all the obstacles in the environment are treated as static can be solved by repositioning the movable obstacles accordingly. Our experimental evaluation demonstrate the positive effects of this ability on the push-manipulation planning and execution performance.

1.2. Contributions

This thesis makes several contributions to the field of mobile manipulation:

- Formalization of mobile manipulation as a case-based planning problem, where the robot *retains* manipulation experiences as cases, *reuses* them for plan construction (non-prehensile) and guidance (prehensile), and *revises* them to match the slight differences observed in the manipulated object's behavior (loaded objects) to improve planning and execution performance.
- A mobile pick and place manipulation algorithm that defines target-specific fine reaching trajectories as cases and reuses them to guide the robot through manipulation plan construction and execution processes [6, 7].
- A mobile whole-body push-manipulation algorithm that incrementally acquires object-specific interaction models and stores them as distinct *probabilistic* cases to be reused as building blocks for achievable push planning and execution [8–10].
- A state lattice-based push-manipulation algorithm that is used for treating movable obstacles as "permeable" to reduce the complexity of planning during push-plan construction for the primary object of interest [11].
- An online algorithm that adapts the probabilistic cases to the changes in the object's behavior based on the observed deviation from the expected outcome [11].
- Extensive experimental evaluation of the presented case-based prehensile and non-prehensile mobile manipulation methods in simulated and real setups.

1.3. Evaluation

We extensively tested our contributed case-based mobile manipulation methods in large scale, cluttered task environments modeled in a realistic 3D simulator (APPENDIX A), where a variety of passively-mobile service objects needed to be safely transported to their desired poses either through pick and place or push-manipulation. Additionally, we verified the validity of the our experience-driven push-manipulation method through a set of preliminary tests in a real world setup with the physical CoBot robot in both static and movable obstacle scenarios (APPENDIX B).

1.4. Thesis Outline

The thesis is organized as follows:

- In Chapter 2, we describe our case-based approach to addressing some of the planning and execution challenges of mobile pick and place tasks in large scale, cluttered environments. We show, through extensive experimentation, the positive effects of guiding sampling-based planners through memorization and reuse of critical yet recurring object-specific fine reaching moves on the efficiency of the generated plans as well as the reliability of their executions.
- In Chapter 3, we present our approach to case-based push-manipulation, where the robot experimentally acquires discrete interaction models for the objects, stores them as probabilistic cases, and uses them as building blocks to construct safe and achievable push-manipulation plans. Our experiments in simulation and real world demonstrate successful application of the contributed method to various challenging push-manipulation problems in cluttered environments.
- In Chapter 4, we extend our push-manipulation method to equip the robot with the capability of adapting to the changes in object dynamics due to loading/unloading, so that the stored cases can still be utilized for manipulation. We show in our experiment results that, through a weighting mechanism, the robot can adapt the previously learned cases on the fly to the novel problem with different dynamics and use them to their full extent throughout the task execution.
- In Chapter 5, we demonstrate how the ability to rearrange the task environment through manipulating movable obstacles improves planning and execution performance when push-manipulating the primary object of interest.
- In Chapter 6, we summarize our major contributions and findings, and conclude the thesis with a discussion of potential directions for future work.

2. MOBILE PICK AND PLACE MANIPULATION

2.1. Motivation

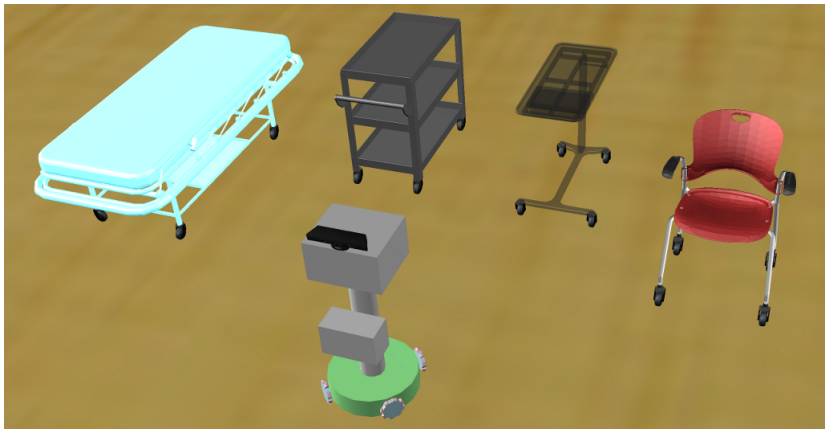
Majority of the manipulation activities that are performed in everyday living contexts are instances of pick and place tasks. This type of tasks usually require careful planning and delicate execution as the manipulator has to be finely aligned with the object of interest (OOI) during pick up and the destination during placement. 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 and robust executions.

Despite the intimidating complexity and delicacy of mobile manipulation, careful observation of recurring pick and place tasks performed in everyday scenarios helped us draw the following conclusions, which motivated the ideas behind our contribution:

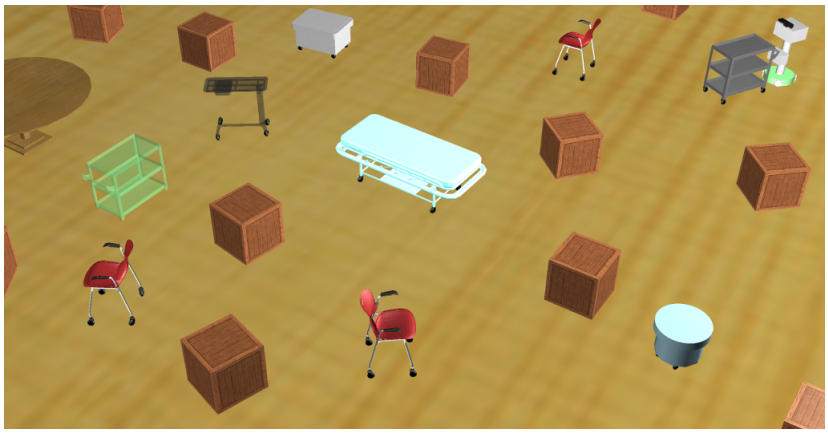
(i) Reaching for the OOI and/or the destination is usually done as directly, roughly, and quickly as possible. The most critical parts of a manipulation task are the moves performed within close proximities of the OOI prior to pick up or the destination prior to placement, and those need to be executed delicately.

(ii) 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 approaching a particular OOI and/or its destination. For example, a bottle is approached and 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.

Building on these observations and conclusions, we contribute a case/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, hence improving efficiency compared to planning



a) The simulated mobile whole-body manipulator modeled after our omni-directional mobile service robot CoBot [2] (Figure 1.1) and several manipulable hospital and office objects used in the experiments.



b) The robot (top right) is transporting the utility cart to its desired destination represented as a pale green ghost figure of the cart (middle left) in a cluttered task environment.

Figure 2.1. (a) The simulated mobile manipulator is capable of manipulating several office and hospital objects. (b) A random task environment cluttered with various manipulable objects, stationary obstacles, and furnitures.

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, various utility carts, and stretchers, to their desired destinations while avoiding collisions in cluttered task environments (Figure 2.1).

The rest of the chapter is organized as follows. Section 2.2 provides some brief background information on the Rapidly-exploring Random Tree (RRT) variant generative motion planning algorithms while Section 2.3 gives an overview of the related work. The contributed method is explained thoroughly in Section 2.4. The results of our extensive experimental evaluation are presented in Section 2.5. Finally, Section 2.6 summarizes and concludes the chapter while pointing out some potential future work.

2.2. Background

In this work, we utilize a set of RRT variant algorithms, namely the original RRT itself [12, 13], RRT-Connect [14], and RRT* [15], as the generative planner component of our proposed approach. In addition to their simplicity, practicality, and probabilistic completeness property, the sampling-based nature of these algorithms aligns very well with the way we define and store the robot's manipulation experiences.

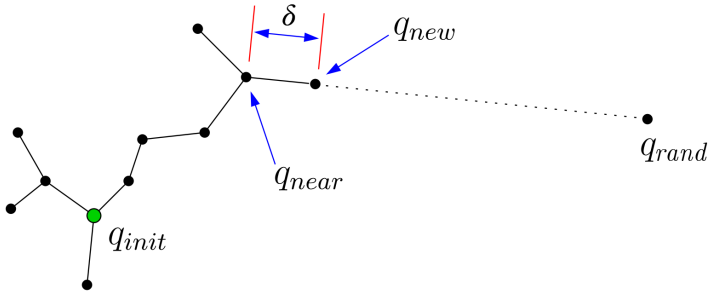


Figure 2.2. Tree construction process of the RRT algorithm [14].

The basic version of the RRT algorithm is given in Figure 2.3. 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. At 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 2.2). This process ends when the newly added node reaches the goal configuration within some similarity boundaries. 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 $1 - p$.

```
1: function BUILDRRT(q_init, q_goal)
2:     Tree ← q_init;                ▷ set the root of the tree as the initial configuration
3:     q_new ← q_init;
4:     while dist(q_new, q_goal) > THRESHOLD do ▷ while the goal is not reached
5:         q_rand ← Sample();                    ▷ sample a random configuration
6:         q_near ← Nearest(Tree, q_rand);       ▷ compute the nearest configuration
7:         q_new ← Extend(q_near, q_rand);       ▷ extend the tree towards the sample
8:         if CollisionFree(q_new) then          ▷ if the extension is not in collision
9:             Tree.add(q_new);                  ▷ add the new configuration to the tree
10:        end if
11:    end while
12:    return Trajectory(Tree, q_new);                                  ▷ return solution
13: end function
```

Figure 2.3. The basic RRT algorithm.

As the number of nodes in the tree increases, however, finding the closest node to the randomly sampled configuration takes longer as 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. Yershova and LaValle [16] presented significant speed ups in the nearest neighbor search by utilizing a kd-tree for partitioning the configuration space and performing the nearest neighbor search on the kd-tree, which is the approach we adopt in our implementation of the RRT variants. As a promising step towards reusing previously generated plans, Bruce and Veloso [17] 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 [18] 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.

In an attempt to move over larger distances during tree expansion, Kuffner and LaValle proposed the RRT-Connect [14] algorithm. 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. The solutions generated via the RRT-Connect algorithm can be quicker and shorter as the distances between the configurations are traversed more directly. This property of the algorithm aligns with our motivating idea (1) about reaching the critical manipulation region as quickly and directly as possible.

Even though RRT rapidly explores the configuration space and eventually reaches the goal, it does not guarantee the optimality of the solution. A recent study that focuses on the optimality of the paths generated by RRT was contributed by Karaman and Frazzoli [15], in which they present the RRT* algorithm. 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 optimality and directness property of the RRT*-generated solutions also align well with our motivating idea (1).

In Section 2.4.2, we elaborate on how we utilize these sampling-based planning algorithms as the generative planning component of our framework to be used for bridging gaps of various sizes and types between the robot and where it needs to be, and how we improve their performances via experience guidance and reuse.

2.3. Related Work

Reusing previously constructed paths, trajectories, or motion segments has been investigated in various forms in the literature. Here we review the most recent related studies, and compare them against the relevant components of our contribution.

Cohen *et al.* [19] proposed constructing a graph with predefined motion primitives and performing a search on that graph to plan a path while satisfying the desired constraints via the solutions provided by the primitives that are generated on the fly by various analytical solvers. 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. That, in a sense, resembles our approach, where the robot reaches the delicate manipulation region roughly and then performs the critical motions delicately within that region.

Berenson *et al.* [20] contributed a framework that utilizes stored end-to-end paths in addition to a generative planner while seeking a solution for a given manipulation 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 first solution that is returned by either process is used. The similarity of a stored path to the given situation is determined by comparing the start and end configurations. BiRRT is used to repair infeasible paths by filling in the gaps caused by the obstructions along the path. We also utilize the generative planner component of our contributed method to bridge the gaps along the blocked sequences in a similar manner.

Skoglund *et al.* [21] follow a Learning from Demonstration (LfD) [22] approach to demonstrate static trajectories to an industrial manipulator equipped with a vacuum gripper to pick and place objects with certain shape constraints in a tabletop manipulation scenario. Using a magnetic tracker, they capture the trajectories followed by the index finger of the human demonstrator, transform them to the robot's frame of reference, segment them to extract task primitives, and translate those primitives into robot-specific codes to be executed to replicate the demonstrated trajectory. Their approach may be suitable for factory environments, where the robot performs pick and place from/to the same object/destination location; however, it does not provide enough flexibility as it would require new demonstrations for each new task environment configuration that could potentially include static obstacles. Our method works independent of the global poses of the targets as the fine reaching trajectories are stored relative to the targets they are defined for.

Ye and Alterovitz [23] proposed a method that 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. Dynamic time warping [24] is used for aligning multiple demonstration sequences. Implicitly encoded constraints, such as keeping a full spoon 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 scenarios relatively similar to the ones used for learning. Generative planning is utilized to bridge the gaps when obstacles fragment the trajectories. As in the work of Berenson *et al.* [20], they also store full, end-to-end paths.

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, we store 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. Analogous to the endgame tablebases in board games like chess [25], these partial relative trajectories are also used for easing the load on the generative planner component, as we present in the following sections.

2.4. Case-Based Mobile Pick and Place

Pick and place tasks require planning and execution of delicate reaching moves. Especially 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. This kind of experience utilization potentially speeds up planning and improves execution robustness as those moves have been performed before and are known to be successful. Inspired by these observations, we develop the following components and bring them together harmoniously to achieve case-based prehensile mobile manipulation:

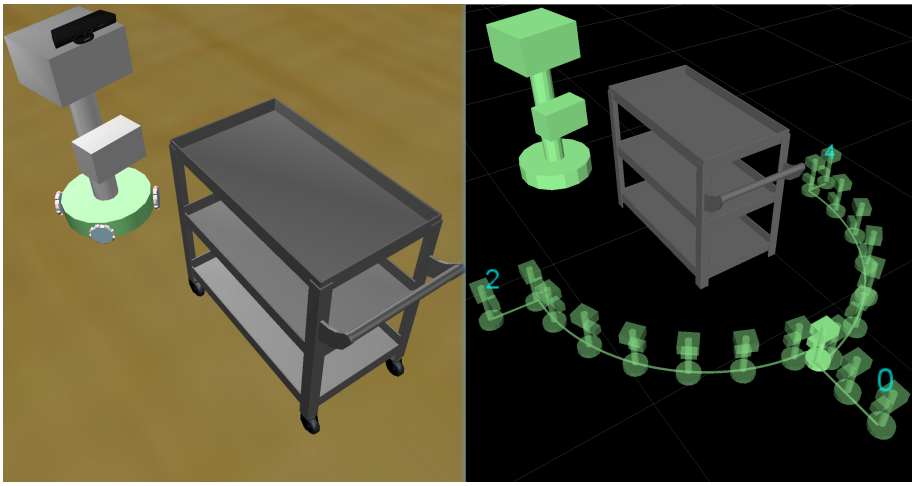
- A set of local reaching moves, represented as sequences of state-action pairs, that have been successfully performed in the past, hence can be treated as the "cases" that the robot knows how to handle.
- A generative planner to bridge the gaps at various stages of planning and execution, such as the initial reaching for the sequences and hopping among the fragmented ones due to obstruction.
- An execution monitoring system that ensures accurate reiteration of the memorized sequences by triggering corrective local motions when needed.

In the remainder of this section, we elaborate on how we define and acquire the target-specific local reaching moves (i.e. cases), how we ensure that these motion sequences are reiterated in such a way to re-obtain the previously experienced outcomes, and how we merge the solutions provided by the generative planner with the partial solutions provided by the available cases so that they complement each other effectively to yield faster and more reliable executions.

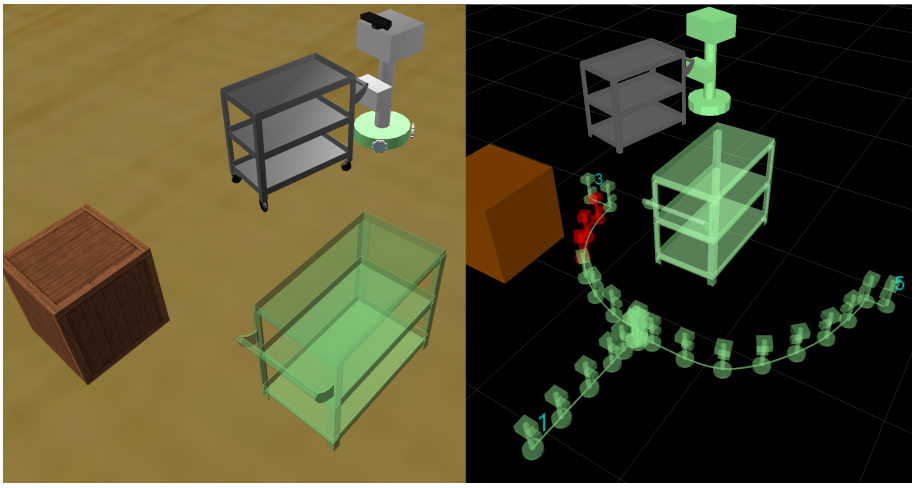
2.4.1. Cases for Delicate Reaching and Manipulation

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 and store them as *cases* that it knows how to handle. 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 come up with 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 memorizes



a) Three pick sequences obtained relative to the OOI.



b) Three place sequences obtained relative to the destination.

Figure 2.4. Visualization of (a) pick and (b) place sequences associated with the utility cart object and its potential destination.

those delicate moves. In other words, the robot needs to be storing its *trajectory* rather than simply the path that it needs to follow with an arbitrary velocity profile. 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 recording 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, we attach 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 $\wp_R$, denoted as $\langle x, y, \theta \rangle$, be φ_R w.r.t. φ_T. 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 "case" describing a local delicate reaching and fine manipulation move, represented as a *sequence* of state-action pairs of length n takes the form

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

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 M is total number of cases stored for a particular OOI. Figure 2.4 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 and the lengths of the stored sequences grow, processing efficiency and scalability suffers, especially if they are being recorded at each step of the robot's perception cycle, which, in our case, corresponds to 30Hz. To address this problem, we sparsify 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 only at the entry points. Figure 2.5 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.

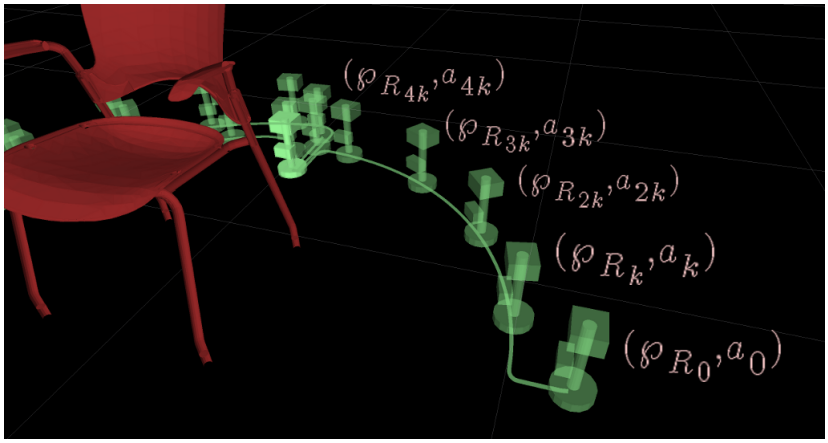


Figure 2.5. Visualization of the sequences around the chair object and their *entry points* depicted as scaled-down ghost robot figures.

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 within the task environment. One important role of the entry points is to provide a way to sparsely (hence quickly) check 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-developed 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 2.4b where an obstacle in the environment results in potential collisions although not directly obstructing the sequence.

Once reached (explained in Section 2.4.2) 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 (explained in Section 2.4.3) to detect divergences from the expected outcome and compensate for them. It also checks 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 portion of one of the sequences (explained in Section 2.4.2). 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 during potential re-planning. Figure 2.6 depicts the process of reiterating a collision-free sequence.

2.4.2. Generative Planner

As previously stated, in our proposed method, the sequences representing local fine manipulation solutions are used analogously to "cases" in a Case-Based Reasoning/Planning (CBR/P) system [26–28]; 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, we simply move the robot towards the cases that it can recall and apply directly. This is similar to what was done by Ros *et al.* [5] with the introduction of the *controllable state features* concept, which was used to transform the currently perceived state to a familiar one rather than trying to adapt the case to the current state.

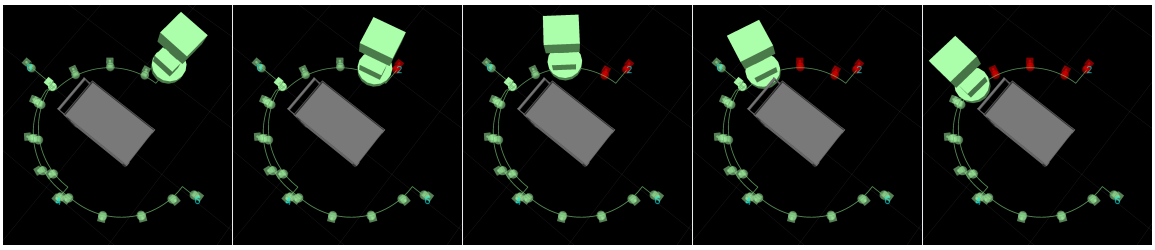


Figure 2.6. The robot reiterates a collision-free sequence simply by executing the motion commands associated with each frame in order. Traversed entry points are marked as *infeasible* (red) to prevent infinite loops in planning and execution.

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, we only need to create plans from scratch to navigate the robot along a collision-free path towards the sequences so that it can merge into one of them and reiterate it. 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 [12, 13], RRT-Connect [14], and RRT* [15], the working principles of which are provided in Section 2.2.

As mentioned in Section 2.2, it is possible to bias the growth of the RRT towards the goal by sampling the goal pose with probability p while sampling a random configuration 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. Our use of the set of feasible entry points to create a bigger region around the target to ease the task of the generative planner resembles the Task Space Regions (TSRs) concept presented by Berenson *et al.* [29], where a tolerance region around the target pose is created and sampled to be fed into the sampling based planner as acceptable goal poses to be reached. However, TSRs only represent the distribution of acceptable final configurations, whereas our representation encode a fine-tuned trajectory for reaching the target from its close proximity. Also, our approach addresses indirectly yet elegantly one of the major 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.

For that purpose, we define the following modes to utilize 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 are a small number of relatively sparse sequences, we 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, in this mode, we sample with probability p_s a sequence S_i from the set of sequences S associated with the target. Then we randomly sample an entry point from the set of feasible entry points E_{S_i} that belong to S_i. This gives us $(|S|p_s)$ as the total probability of sampling an entry point. Similar to the previous mode, a random configuration is sampled with probability $(1 - (|S|p_s) - p_g)$.
- *Sampling subgoals within goal probability*: In this mode, we combine the actual goal with the set of subgoals and obtain a total of $(|E| + 1)$ (sub)goals. After we decide with probability p_g to use *any* goal as a sample, we either pick the actual goal with probability $p_{g^*} = 1/(|E| + 1)$ or randomly pick one of the feasible subgoals 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. Considering that the feasible entry points provide a *region* of many subgoals to be reached as opposed to the single actual goal pose, it becomes likely to arrive at one of those subgoals first while trying to reach the actual goal pose.

Instead of just sampling random configurations as in the original RRT algorithm (Figure 2.3), our modified *Sample* function, described in Figure 2.7, utilizes the sequences when sampling robot configurations in the task environment. When these various guid-

```
1: function SAMPLE(SamplingMode)
2:     r ← rand();
3:     if SamplingMode == WITHIN_GOAL_PROBABILITY then
4:         if r ≤ p_g then                ▷ get a goal from the set of all (sub)goals
5:             r ← rand()
6:             if r ≤ 1.0/(|E| + 1) then
7:                 q_rand ← q_goal;                              ▷ get the actual goal
8:             else
9:                 q_rand ← GetRandomEntryPoint();               ▷ get an entry point
10:            end if
11:        else
12:            q_rand ← GetRandomConfiguration(); ▷ get a random configuration
13:        end if
14:    else
15:        if SamplingMode == INDIVIDUAL_SUBGOALS then
16:            p_total ← p_g + |E|p_ep;
17:        else if SamplingMode == 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;                          ▷ return sampled configuration
29: end function
```

Figure 2.7. Configuration sampling function that utilizes the sequences.

ance modes 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 2.5 as part of our extensive experimental evaluation.

Navigating the robot towards the sequences when they are far away is not the only purpose the generative planner is used for. When the robot is reiterating a sequence and the upcoming entry point is observed to be infeasible (i.e. obstructed), generative planning is used to help the robot hop onto another feasible entry point and proceed with the execution of the corresponding sequence. This resembles the methods used by Berenson *et al.* [20] and Ye and Alterovitz [23] to bridge the gaps along the stored trajectories. Figure 2.8 illustrates how the fragmented sequences are bridged via the use of the generative planner for maximizing experience reuse. As mentioned before and shown in the figure, infinite planning and execution loops are prevented by marking the visited entry points as *infeasible* (shown as red).

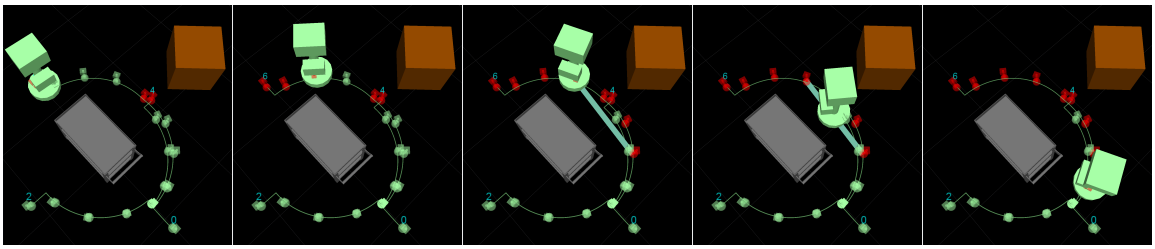


Figure 2.8. Reiteration of a sequence that is blocked by an obstacle. When the robot detects the upcoming entry point along the path to be *infeasible* (red), it uses the generative planner to hop to a feasible portion of either the same sequence or another one and proceed.

2.4.3. Execution Monitoring

Uncertainty is abundant in the world of a robot. Due to the uncertainty in sensing and actuation, there may be discrepancies between the expected outcome of a particular action and the actually observed one. Therefore, it is important for the robot to be actively monitoring itself and its task execution 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 [30].

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. Execution monitoring increases the chance of successfully completing the task as opposed to open-loop reiteration.

2.5. Experimental Evaluation

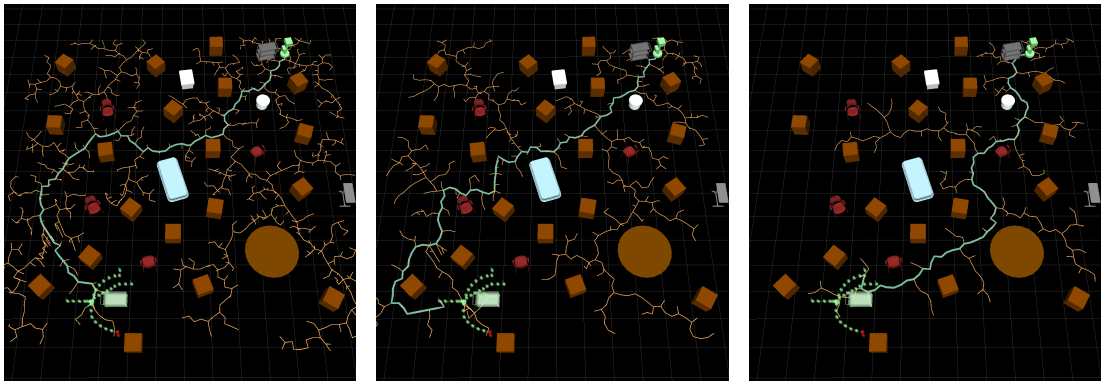
We conducted extensive experiments in the Webots [1] mobile robot simulation environment [31], where we modeled our omni-directional mobile manipulator robot and several passively-mobile, manipulable office and hospital objects shown in Figure 2.1 (described in APPENDIX A). Even though the physical CoBot lacks a manipulator arm and a gripper, in our experiments we assume a hypothetical gripper that makes the manipulated objects get "attached" to the robot when approached within certain relative pose limits. 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 2.2), which is set to be equal to the robot's radius, we placed a practical limit on the maximum number of nodes as 40000 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.

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

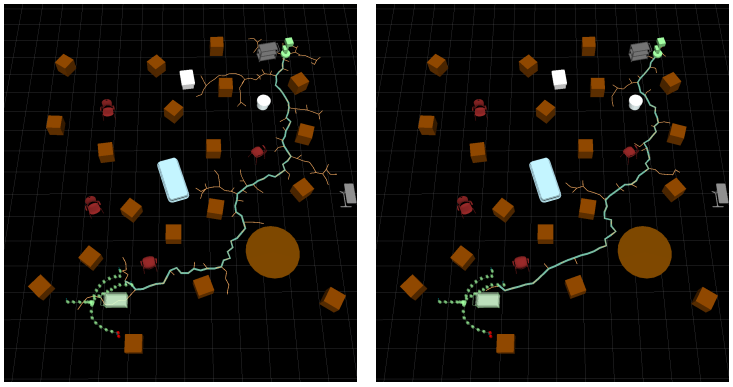
For the acquisition of the cases representing fine reaching moves, we followed a LfD approach to joystick the robot to demonstrate it how to pick up and place the manipulable objects available in the simulated environment, although it would also be possible for the robot to acquire that experience through self exploration and experimentation. Through these demonstrations, we provided the robot with four pick and four place sequences per OOI. Given the dimensions of the objects in our task environments and our motion precision requirements 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; hence, the exploratory nature of the sampling-based generative planners is maintained.

Figure 2.9 provides a visual overview of the effects of various subgoal utilization methods explained in Section 2.4.2 on the generated RRT branches (orange) as well as the solution paths (blue) for navigating the utility cart 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, hence the number of nodes, seem to decrease while the directness of the solution paths increase with the increased utilization of the sequences. This kind of effect is expected as the attraction of a critical 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



a) No subgoals used. b) Coincidental reach. c) Sampling within goal.



d) Sampling sequences. e) Sampling subgoals.

Figure 2.9. 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.

Table 2.1. RRT planning statistics for various subgoal utilization methods (corresponding to Figure 2.9). 38 out of 41 subgoals were feasible. There is an almost linear correlation between the number of nodes and planning time.

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

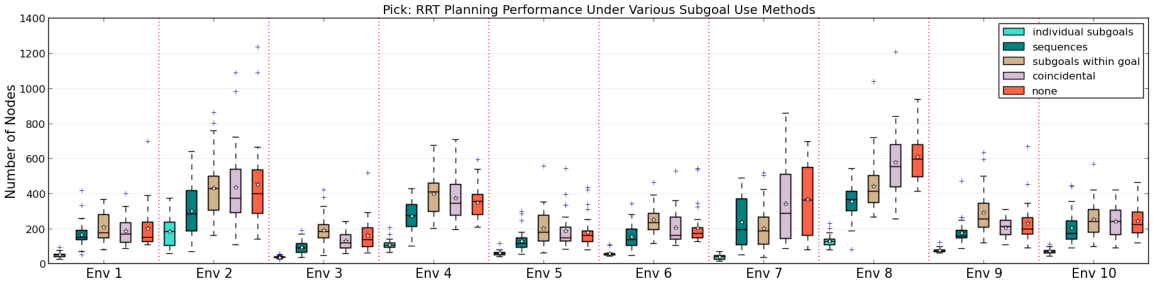
preliminary runs are provided in Table 2.1, where an almost linear correlation between the number of nodes and planning time can be observed.

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 10 randomly configured task environments with four manipulable objects; namely a chair, an overbed table, a utility cart, and a stretcher (shown in Figure 2.1a). 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) without taking the existence of the fine manipulation moves into account. [2]

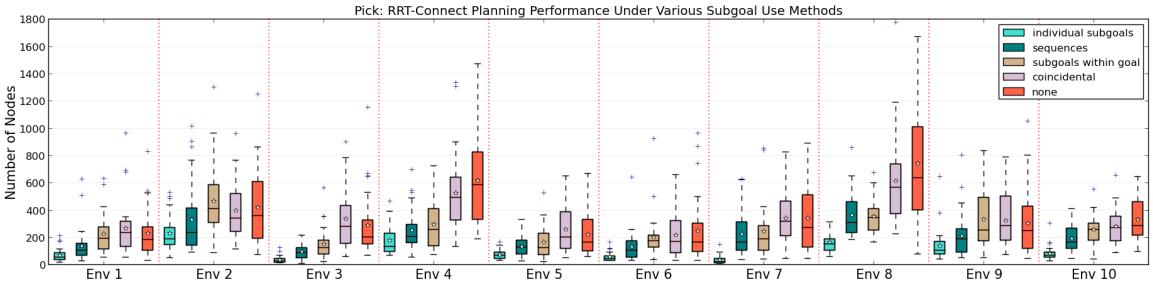
The box plots shown in Figure 2.10 and Figure 2.12 present the planning performance statistics, measured in number of nodes, obtained with each of these combinations for picking up and placing the chair object, respectively. The box plots shown in Figure 2.11 and Figure 2.13 provide the corresponding time statistics. Similar relative performances were observed in the experiments run with the other manipulable objects, the detailed results of which are given in APPENDIX C. 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 2.1.

Looking at the plots in Figure 2.10 and Figure 2.12, 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 method 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,

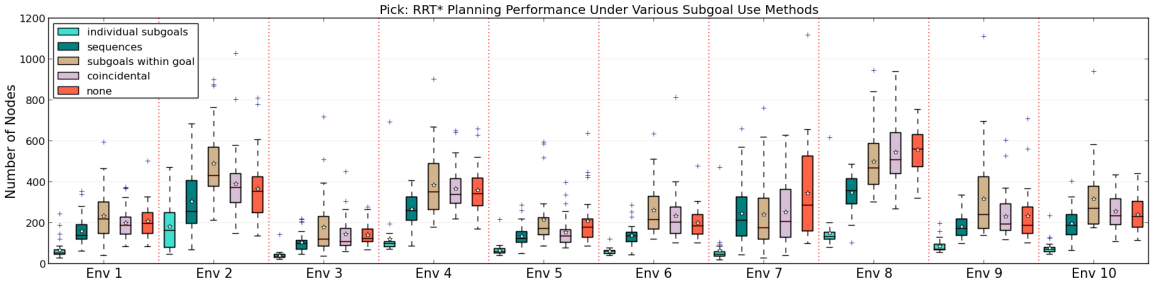
[2] Visualizations of the planning process with each planner and subgoal utilization method combination, where the shrinking effect of increased subgoal utilization on the spread of the generated trees and the solution paths can be seen in `http://youtu.be/ePZYq41uTrA`



a)



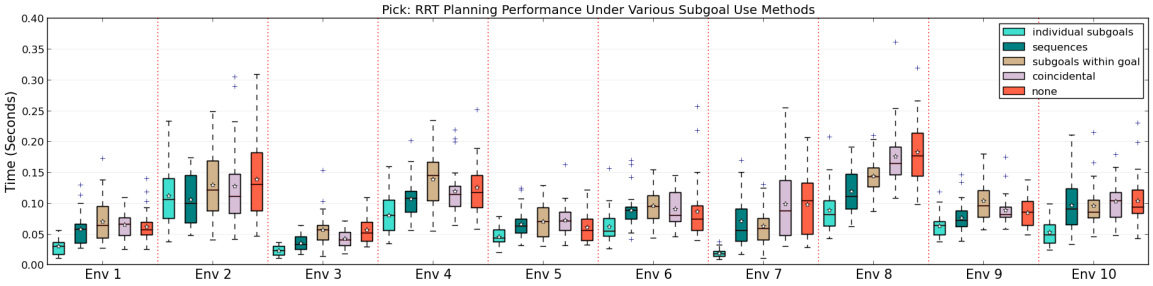
b)



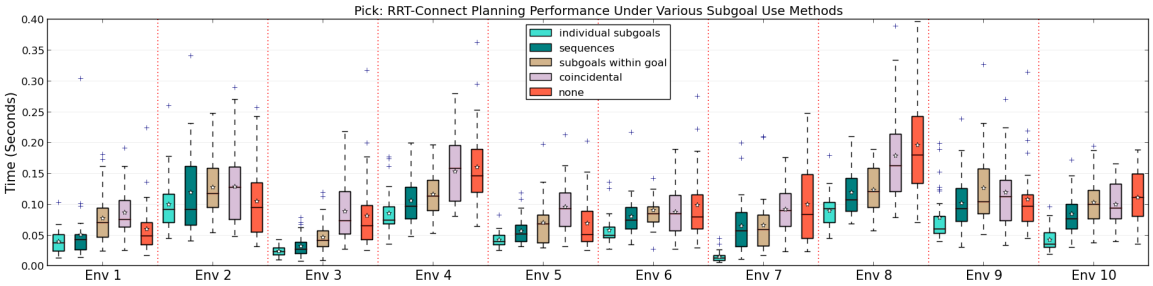
c)

Figure 2.10. The statistics for the number of generated nodes by (a) the RRT, (b) the RRT-Connect, and (c) the RRT* generative planners during planning for picking up the chair object in 10 different task environments.

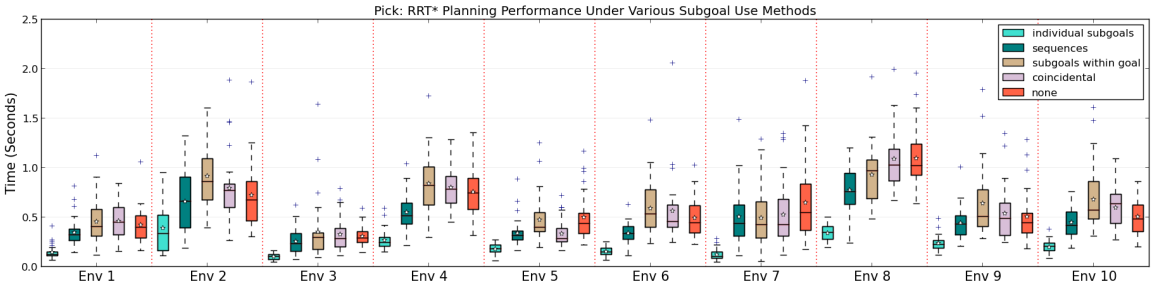
instead of consistently directing the planner towards a (sub)goal in each rare occasion of deciding to sample a goal, this method distracts the focus of the generative planner, diminishing the attractive properties of the fine manipulation region. In case the RRT-Connect planner is used, however, this method 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 working principles of the RRT-Connect algorithm, which is about direct extension towards the sample unless an obstacle is encountered. Therefore, once a (sub)goal is sampled, the algorithm tries to reach it along a straight



a)



b)

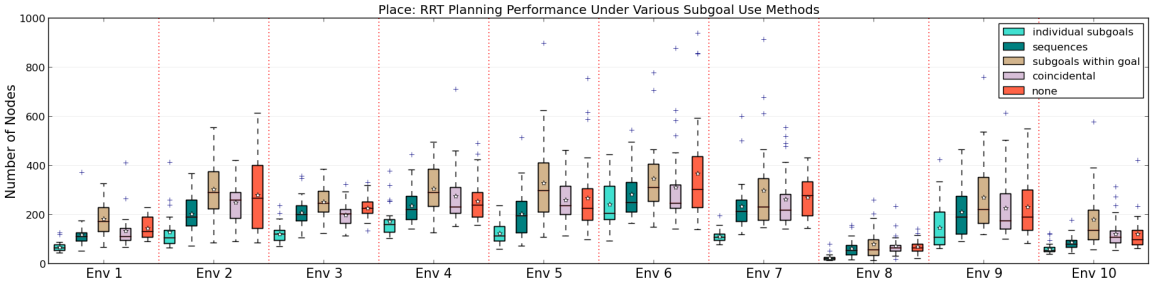


c)

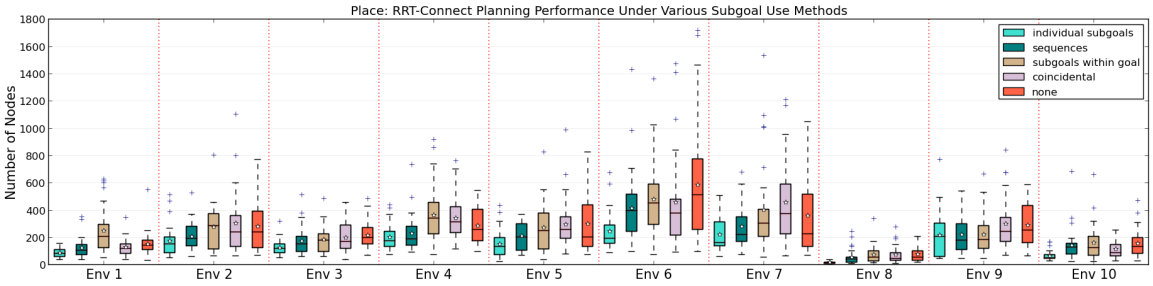
Figure 2.11. Time statistics for (a) the RRT, (b) the RRT-Connect, and (c) the RRT* planners to generate plans for picking up the chair object in 10 different task environments.

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 method are observed when the number of sequences and their densities are increased. The corresponding time statistics given in Figure 2.11 and Figure 2.13 reflect similar profiles, as the number of nodes and the planning time are correlated almost linearly as shown in Table 2.1.

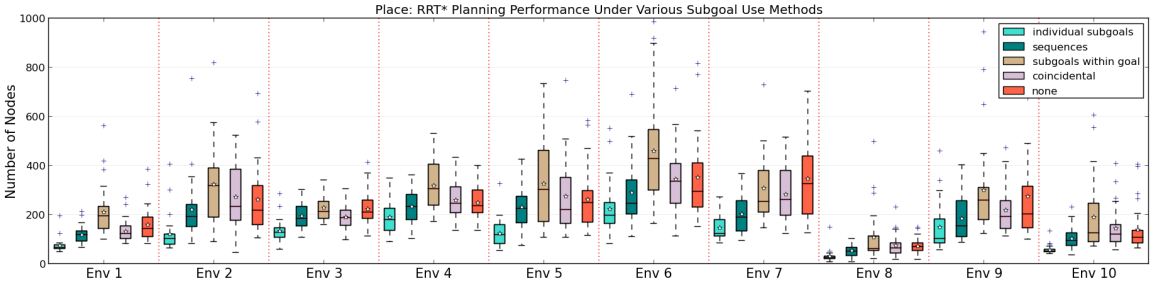
Table 2.2 and Table 2.3 provide sequence utilization statistics of the tests pre-



a)



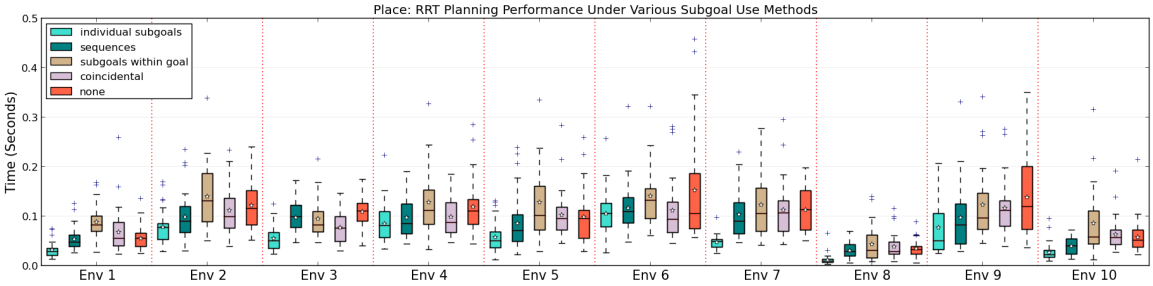
b)



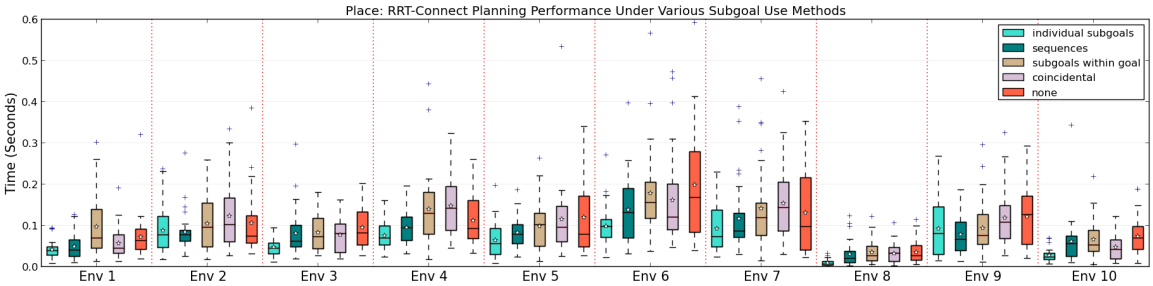
c)

Figure 2.12. The statistics for the number of generated nodes by (a) the RRT, (b) the RRT-Connect, and (c) the RRT* generative planners during planning for placing the chair object in 10 different task environments.

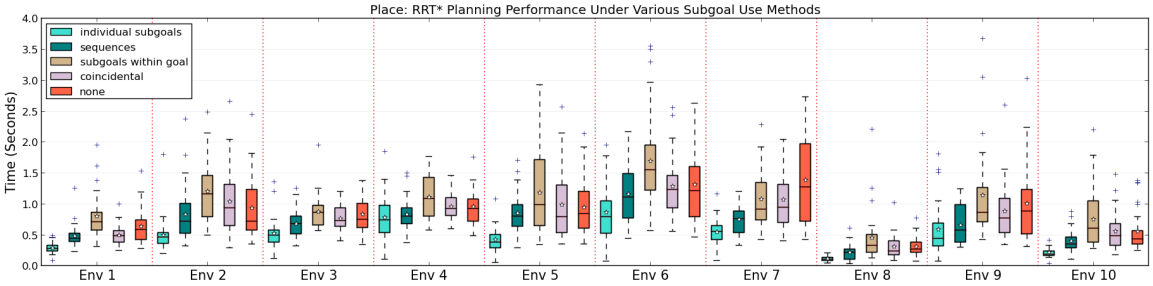
sented in Figure 2.10 and Figure 2.12, respectively. These numbers indicate in what percentage of the 30 planning trials with each combination the planner eventually ended up reaching an entry point. We see that sampling individual subgoals and sampling sequences have very high sequence utilization percentages, which is expected due to the increased total bias towards the delicate manipulation region. Even the coincidental termination method seems to be making decent use of the sequences, which helps reduce the load on the generative planner, as is reflected in the corresponding box plots. Although the percentages reported for the method of sampling subgoals within goal



a)



b)



c)

Figure 2.13. Time statistics for (a) the RRT, (b) the RRT-Connect, and (c) the RRT* planners to generate plans for placing the chair object in 10 different task environments.

probability seems to be unexpectedly high, this situation has a perfect explanation. Since this method 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 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

Table 2.2. Sequence utilization while planning and executing pick task for the chair object in ten different environments.

Planner	Subgoal Use Method	Sequence Utilization (%)									
		Pick									
		Env 1	Env 2	Env 3	Env 4	Env 5	Env 6	Env 7	Env 8	Env 9	Env 10
RRT	Coincidental	13.33	13.33	20.0	3.33	6.67	20.0	20.0	13.33	6.67	20.0
	Within goals	80.0	56.67	50.0	66.67	80.0	73.33	73.33	86.67	60.0	80.0
	Sequences	36.67	53.33	33.33	36.67	33.33	30.0	50.0	36.67	26.67	30.0
	Individual subgoals	80.0	66.67	90.0	93.33	83.33	50.0	70.0	96.67	53.33	83.33
RRT-Connect	Coincidental	30.0	33.33	36.67	13.33	33.33	6.67	20.0	20.0	16.67	16.67
	Withing goals	76.67	90.0	93.33	100.0	80.0	83.33	90.0	90.0	80.0	86.67
	Sequences	73.33	53.33	63.33	76.67	70.0	46.67	80.0	86.67	40.0	53.33
	Individual subgoals	93.33	90.0	96.67	96.67	96.67	56.67	96.67	90.0	66.67	93.33
RRT*	Coincidental	10.0	10.0	6.67	10.0	23.33	3.33	33.33	23.33	6.67	13.33
	Within goals	73.33	73.33	76.67	60.0	73.33	66.67	86.67	73.33	70.0	80.0
	Sequences	33.33	26.67	40.0	50.0	26.67	30.0	43.33	66.67	16.67	43.33
	Individual subgoals	83.33	70.0	83.33	90.0	86.67	46.67	73.33	93.33	43.33	90.0

Table 2.3. Sequence utilization while planning and executing place task for the chair object in ten different environments.

Planner	Subgoal Use Method	Sequence Utilization (%)									
		Place									
		Env 1	Env 2	Env 3	Env 4	Env 5	Env 6	Env 7	Env 8	Env 9	Env 10
RRT	Coincidental	26.67	16.67	10.0	6.67	23.33	10.0	13.33	10.0	6.67	13.33
	Within goals	90.0	80.0	76.67	80.0	86.67	90.0	86.67	93.33	86.67	90.0
	Sequences	20.0	23.33	36.67	26.67	50.0	20.0	20.0	23.33	20.0	26.67
	Individual subgoals	60.0	63.33	63.33	63.33	90.0	56.67	53.33	83.33	60.0	70.0
RRT-Connect	Coincidental	13.33	16.67	3.33	16.67	23.33	13.33	6.67	26.67	6.67	13.33
	Withing goals	90.0	93.33	86.67	86.67	93.33	90.0	93.33	93.33	100.0	93.33
	Sequences	46.67	33.33	33.33	43.33	73.33	46.67	60.0	50.0	43.33	56.67
	Individual subgoals	86.67	83.33	86.67	80.0	83.33	86.67	93.33	90.0	86.67	73.33
RRT*	Coincidental	20.0	20.0	13.33	3.33	20.0	20.0	16.67	3.33	13.33	6.67
	Within goals	83.33	86.67	80.0	80.0	83.33	86.67	80.0	90.0	83.33	66.67
	Sequences	33.33	23.33	33.33	30.0	50.0	26.67	16.67	36.67	23.33	43.33
	Individual subgoals	76.67	70.0	60.0	53.33	83.33	56.67	66.67	70.0	53.33	63.33

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 2.4 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 2.14 illustrates task completion time statistics obtained for both scenarios. Looking at the mean values,

Table 2.4. Number of entry points used when partial sequence segments are utilized versus when they are discarded.

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

marked as 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. When all feasible segments are utilized, due to the greater attraction of greater number of 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.

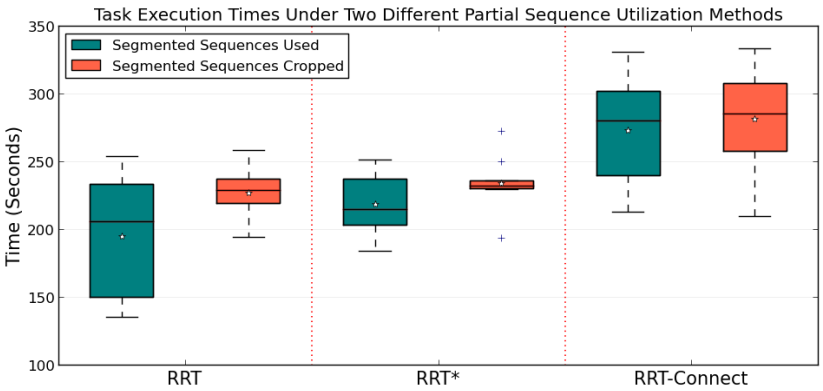


Figure 2.14. Pick and place task completion times when the feasible segments of the partially occluded sequences are utilized in planning and execution versus when the all fragments prior to the last collision-free segment are discarded.

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 2.15.

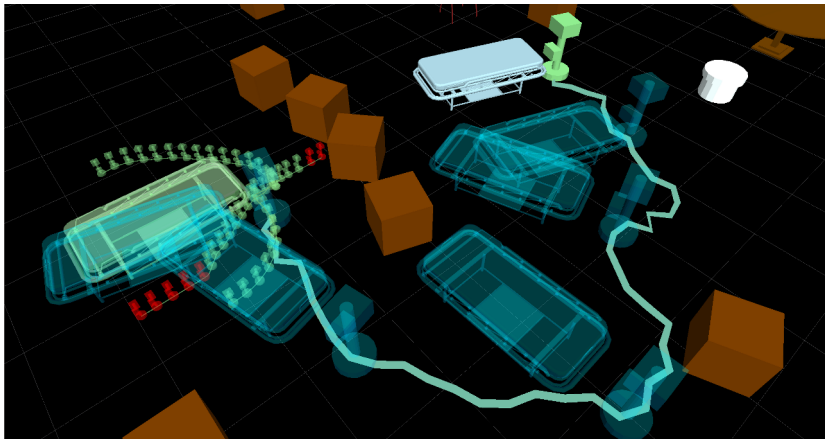


Figure 2.15. Visualization of the stretcher placement scenario where the direct path to the actual goal pose is blocked. 42 out of 48 entry points were feasible.

We experimented with each combination of the three generative planners and the four subgoal utilization methods 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 method 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 to $(42 \times 0.01 + 0.05 = 0.47)$; therefore, $p_g = 0.47$ for the base planning case. Figure 2.16 and Figure 2.17 illustrates the relative performances of each of the combinations measured in terms of the number of generated nodes and planning time, respectively. Even in this challenging scenario, all of our experience-guided methods performed better than the base case planning method, sampling individual subgoals resulting in the best performance. The entry points being distributed around the target helps the planner find its way around the obstacles quicker compared to a single goal with the same amount of total attraction.

Sampling subgoals within goal probability takes slightly longer than the base case with higher goal sampling bias for RRT* even though the former performs better in terms of the number of nodes metric. The potential reason is the rewiring of the tree for the goals on the opposing sides of the line of sight due to random sampling among

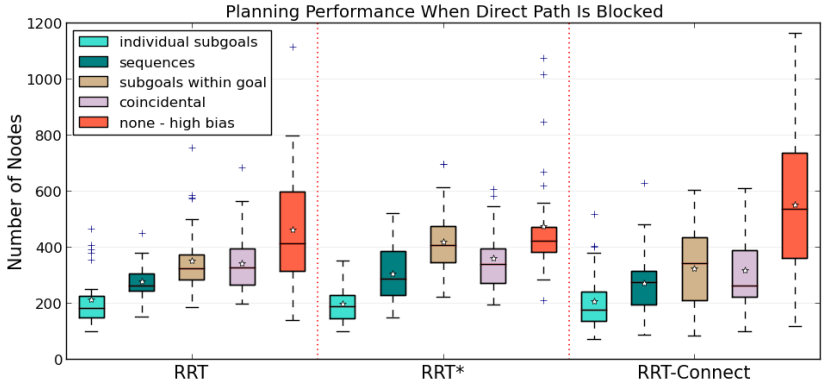


Figure 2.16. Statistics for the number of plan tree nodes when various subgoal utilization methods are used versus when the goal bias of the base planner is merely increased in the scenario where the direct path to the goal is blocked.

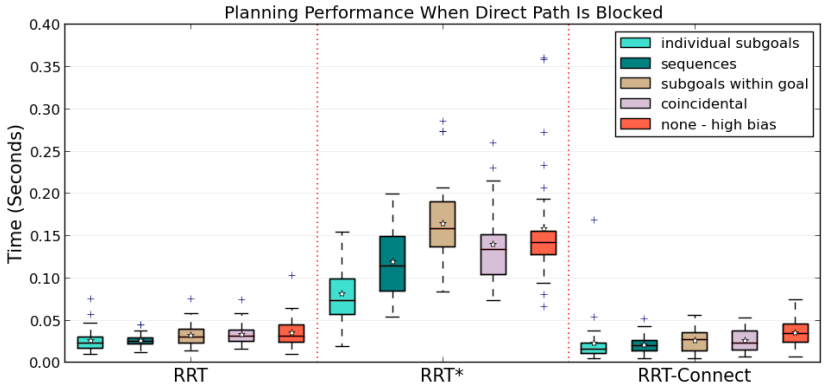


Figure 2.17. Time statistics of the analysis presented in Figure 2.16.

them, causing the tree to take longer to go around the barrier to reach the sequences. In general, since RRT* has to keep track of a set of near neighbors instead of the single nearest neighbor and rewire the tree accordingly during construction, which is a time consuming process, it results in an overall larger time footprint compared to the other planners, as can be seen in Figure 2.17. This effect can also be observed in all the time statistics plots of our pick and place experiments.

2.6. Discussion

Pick and place tasks, which constitute the majority of the manipulation activities that are performed in everyday living contexts, usually require careful planning and delicate execution. 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 and robust executions. On the other hand, careful observation of these tasks shows us that usually a small set of target-specific fine reaching moves are repetitively reused instead of planning for those parts from scratch.

In this chapter, we present an experience/case-based mobile prehensile manipulation method where the robot memorizes a few number of those 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 in a realistic 3D simulation environment 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 contributed approach harmoniously combines the already available partial plans and executions with the ones generated from scratch, yielding to fast, reliable, and repeatable solutions. [3]

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

3. MOBILE PUSH-MANIPULATION

3.1. Motivation

Pushing is one of the many modalities of *non-prehensile* manipulation [32], which may be the most suitable option depending on the requirements of the manipulation task and the constraints imposed by the physical properties of both the object and the robot. For instance, the object may be too large or heavy, the robot may not be equipped with a manipulator arm, or the utilization of some properties of the object may make its transportation more efficient and convenient that way. The objective of push-manipulation is to come up with and execute a sequence of pushing actions to maneuver an object incapable of moving by itself from an initial configuration to a goal configuration. In this study, we expect our omni-directional mobile robot CoBot [2] (Figure 1.1), which is not equipped with a manipulator arm and/or a grasping mechanism, to push-manipulate a set of passive mobile objects (Figure 3.1) in such a way to transport them to their desired poses while avoiding collisions in the task environment cluttered with both stationary and potentially movable obstacles. This is not a trivial problem and it gets even more challenging in our case due to the following facts:

- Our manipulable objects move on passive caster wheels, which introduce additional motion uncertainty as the objects continue moving for some time even after the push is ceased. Objects with these kinds of properties are inherently more difficult to push-manipulate compared to objects that slide quasi-statically on high-friction surfaces.
- Our objects have complex 3D structures. It is neither trivial nor feasible to write down analytical interaction and motion models for each and every one of such complex objects; hence, traditional model-based planning approaches will not solve the problem in a flexible way.

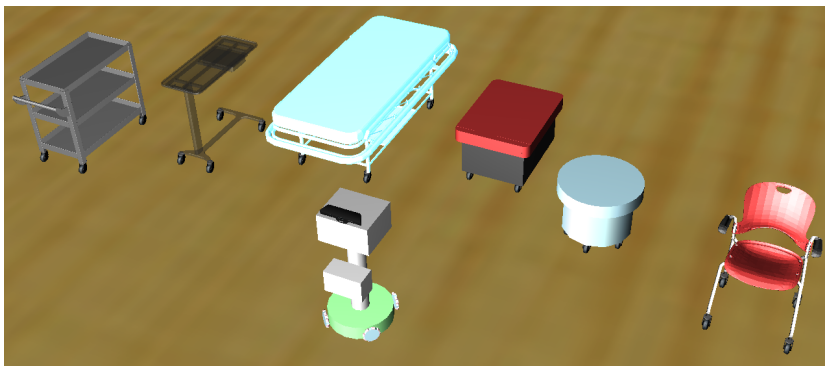


Figure 3.1. Our omni-directional mobile push-manipulator robot, modeled after CoBot, together with a set of realistically simulated passively-mobile objects.

As a promising solution to this problem, we develop a method that does not require any explicit analytical models for neither the objects nor the robot [8,9], rather, individual interactions with the objects are stored as *cases* similar to a case-based planning approach [26, 33, 34]. The robot builds object-specific *experimental motion models* by memorizing the observed effects of its pushing moves on various passively-mobile objects and iteratively tuning them. The acquired experimental models are then used for push-manipulating the object of interest (OOI) by following either a reactive strategy or an informed and more cautious one, as described below.

(i) The robot can plan a collision-free path for the OOI without taking into account whether the plan can actually be achieved with the available experimental models, and then tries to make the object track the path by reiterating the memorized pushing actions that result in the best locally-matching object trajectories until the goal is reached.

(ii) The robot can utilize the acquired experimental models as building blocks for constructing safe and *achievable* push-manipulation plans via Exp-RRT, a Rapidly-exploring Random Trees (RRT) variant planning algorithm we contribute [8, 9], and then execute the constructed plans by reiterating the corresponding pushing motions one after another while actively monitoring the execution.

The following sections elaborate on these two case-based push-manipulation approaches and present detailed experimental analyses.

3.2. Related Work

Pushing enables complex manipulation tasks to be performed with simple mechanics in cases where the object is too bulky or heavy to lift, or the robot simply lacks a manipulator arm. As a result of being one of the most interesting methods used within the non-prehensile manipulation domain [32, 35], push-manipulation has attracted several robotics researchers.

An early work by Salganicoff *et al.* [36] presents a very simple, 1-nearest neighbor based approximation method for the forward model of an object being pushed from a single rotational contact point in an obstacle-free environment by controlling only one degree of freedom. Their method is parallel to ours from the perspective of utilizing observations on the effects of past pushing actions rather than trying to estimate the parameters of the physical interactions between the object and the environment. However, they tackle the problem in a much lower dimensional space and utilize several assumptions about the setup, such as the object being attached to the robot's fingertip and sliding quasi-statically on the tabletop surface.

Agarwal *et al.* [37] propose an algorithm for computing a contact-preserving push plan for a point-sized pusher and a disk-shaped object. They use discrete angles at which the object can be pushed and a finite number of potential intermediate positions for the object. They assume that their pusher can place itself at any position around the object since it does not occupy any space; however, this approach cannot be used when real robots are considered as they have non-zero dimensions that can collide with the obstacles in the environment.

Nieuwenhuisen *et al.* [38,39] utilize compliance of the manipulated object against the obstacles rather than trying to avoid them, and make use of the obstacles with linear surfaces in the environment to guide the object's motion by allowing the object to slide along the boundaries. de Berg and Gerrits [40] computationally improve this approach and present both a contact preserving and an unrestricted push planning method in which the pusher can occasionally let go of the object. In our task environments, we

particularly avoid collisions since our passively-mobile objects could potentially damage the environment as well as themselves, both of which are undesired.

Similar to the potential field based motion planners [41], Igarashi *et al.* [42] propose a method that computes dipole-like vector fields around the object that guide the motion of the robot to get behind the object and push it towards the target. Relatively slow robot motions and high friction for the objects are assumed, and robots with circular bumpers are used to push circular and rectangular objects of various sizes in single and multi-robot scenarios.

As a promising step towards handling objects with more complex shapes, Lau *et al.* [43] achieve pushing of irregular-shaped objects with a circular robot by collecting hundreds of samples on how the object moves when pushed from different points in different directions, and using a non-parametric regression method to build the corresponding mapping, similar to the approach proposed by Walker and Salisbury [44]. Their approach resembles ours in the sense that they also utilize the observations of the object's motion in response to various pushing actions. Even though they use irregular-shaped objects in their experiments, those objects are flat ones with quasi-static properties and the final placement orientation is ignored in their experiments, which further simplifies the problem.

Zito *et al.* [45] present an algorithm that combines a global sampling-based planner with a local randomized push planner to explore various configurations of the manipulated object and come up with a series of manipulator actions that will move the object to the intermediate global plan states. Their experiment setup consists of a simulated model of a tabletop robot manipulator with a single rigid spherical fingertip and an L-shaped object (a polyflap) to be manipulated. The setup is obstacle-free and the state space is limited to the reach of the robot arm, which is relatively small, as they are using a stationary manipulator. The randomized local planner utilizes a realistic physics engine to predict the object's pose after a certain pushing action, which requires explicit object modeling.

Kopicki *et al.* [46] use the same problem setup and present an algorithm for learning through interaction the behavior of the manipulated object that moves quasi-statically in response to various pushes. However, the learned object behavior is not used for push planning in their work.

Scholz and Stilman [47] use the observed outcomes of a set of four linear and two rotational pushes for planning in a quasi-static tabletop setup where a manipulator arm with a spherical rigid finger push-manipulates objects with simple geometric shapes, ensuring single point of contact. Their use of the learned forward models to build sampling-based push-plans resembles our achievable push-manipulation approach, except that they handle the problem in a much lower dimensional space in terms of the interactions between the robot, the object, and the environment as well as the scale of their experiment setup.

Another recent study by Dogar and Srinivasa [48] uses push-manipulation in a tabletop scenario as a way to reduce uncertainty prior to grasping by utilizing the funneling effect of pushing instead of directly manipulating the OOI.

Along the lines of learning object kinematics and dynamics, Katz and Brock [49] propose an interactive perception approach, where the robot interacts with the objects in a tabletop scenario and identifies individual rigid bodies that compose the object by tracking how the corresponding visual feature points move relative to each other. Using that information, the robot is able to extract the kinematic properties of an articulated OOI to be later utilized for manipulating the OOIs potentially as tools.

According to our survey of the literature, the most common push-manipulation scenarios seem to involve pushing of objects with primitive geometric shapes using circular or point-sized robots, or rigid fingertips on a surface with relatively high friction that makes the object stop immediately when the pushing motion is ceased. Even then, relatively complex analytical models are used for contact modeling and motion estimation, or physics engines of simulators are utilized for these purposes. Our approach differs from many of these proposed ones in the sense that;

- we deal with real world objects of complex 3D structure that may contact the robot on various points (Figure 3.2),
- the manipulated objects do not move in a quasi-static manner; they continue moving freely for a while after the push, and their caster wheels contribute to their motion uncertainty,
- *mobile* manipulation is performed in a large-scale environment cluttered with obstacles, requiring construction and execution of safe and *achievable* plans,
- no explicit analytical model is used or learning based mapping is built; only the pushing motions performed in the past and their corresponding observed effects along with the associated variances are utilized for planning and execution.

a) b) c)

Figure 3.2. The object may contact (indicated by a red arrow) the robot at (a) its body, (b) basket, or (c) base, or a combination of these depending on its 3D structure and the pushing direction, making the interaction non-trivial to model explicitly.

3.3. Case-based Mobile Push-Manipulation

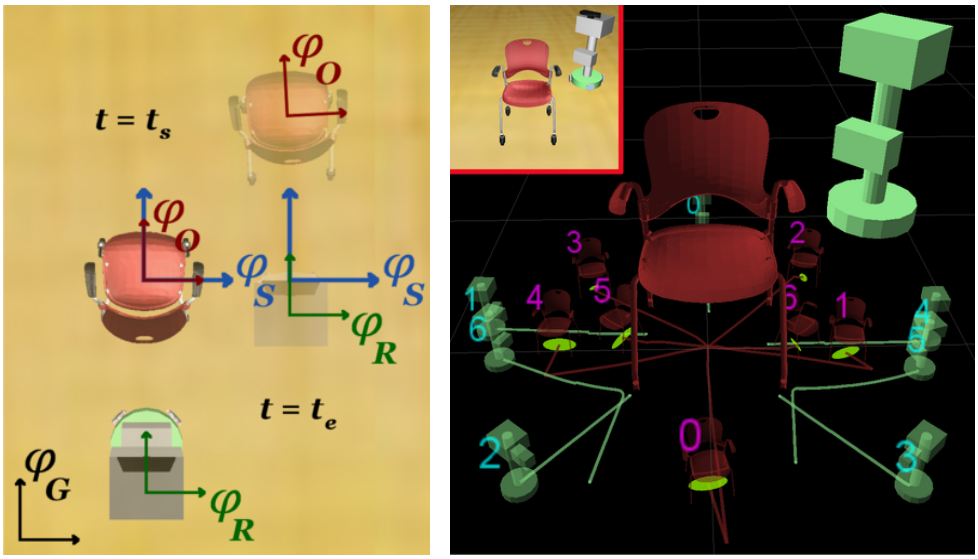
Humans learn and further sharpen their manipulation skills as well as their corresponding prediction-based planning abilities by interacting with their environments and observing the outcomes. Ideally, robots should also learn from their experiences as opposed to the unscalable and inefficient approach of providing them with detailed mathematical models of each and every object that they are expected to interact with, and utilizing physics engines to compute the outcomes of these interactions. In our case, due to the complexity of the potential interactions between the robot, the objects, and the floor surface as well as the resulting motion characteristics, it is neither trivial

nor efficient to try to define such models manually. For these reasons, we let our robot interact with the pushable objects either through self-exploration or demonstration via joysticking to observe how they move in response to various pushes. These observations are then turned into *experimental models* and stored as *cases*, analogous to a Case-Based Reasoning/Planning (CBR/P) system [26–28], to be used for planning and execution. When the robot is learning through self-exploration, the parameters of these interactions, such as the pushing locations, directions, and durations, are determined randomly. Our algorithm consists of the following components, which we explain in detail in the rest of this section:

- A set of object-specific *probabilistic cases* represented as *sequences* composed of the robot's motion commands, its resulting active trajectory, and the object's corresponding observed passive trajectory,
- A *generative planner* that makes use of these probabilistic cases as building blocks to construct achievable and collision-free push plans,
- An *execution monitoring module* to stop execution and trigger re-planning whenever there is a significant discrepancy between the expected and the actual motion of the object during plan execution.

3.3.1. Probabilistic Cases for Push-Manipulation

Each distinct interaction of the robot with a pushable object constitutes an object-specific *case* represented as a *sequence* of pose-action pairs for the robot and the corresponding poses for the object, representing their active and passive trajectories, respectively. These trajectories are defined with respect to various frames of reference. A static global frame of reference, φ_G, is attached to the environment. We also attach separate frames of reference to the robot and the object of interest, denoted as φ_R and φ_O, respectively, to define their poses within φ_G. In addition, we define an auxiliary frame of reference, φ_S, to indicate the last stationary pose of the object before it starts being pushed. Figure 3.3a illustrates these reference frames in a sample scenario where our mobile manipulator pushes a chair, causing it to get displaced.



a) b)

Figure 3.3. (a) Various reference frames used during case acquisition and reiteration depicted before ($t = t_s$) and after ($t = t_e$) a push. (b) Visualization that corresponds to the scene shown in the upper left corner of the image. The robot trajectory and the corresponding object trajectory components of 7 different sequences are illustrated.

Let $\wp_R$ be φ_R w.r.t. φ_O, and $\wp_O$ be φ_O w.r.t. φ_S, both of which are denoted as $\langle x, y, \theta \rangle$. Invariance to φ_O is achieved by recording $\wp_R$ together with the motion command at that moment and the corresponding $\wp_O$. Therefore, a sequence S_i of length n that is encapsulated by a case takes the form

$$S_{i \in [0,M)} : ((\wp_{R_0}, a_0, \wp_{O_0}), \ldots, (\wp_{R_{n-1}}, a_{n-1}, \wp_{O_{n-1}}))$$

where a_j is the action associated with $\wp_{Rj}$, denoted as $\langle v_x, v_y, v_\theta \rangle$ indicating the omni-directional motion command composed of the translational and rotational velocities of the robot, and M is the total number of cases. Figure 3.3b provides the visualization of the robot and object trajectories within the stored sequences. The transparent, scaled-down robot figures indicate the push initiation poses (i.e. the beginning of the sequences) whereas the scaled-down object figures indicate the mean observed poses of the object after the pushes (i.e. the end of the sequences). The robot trajectory (indicated by green curves) and the object trajectory (indicated by red curves) that

belong to the same sequence are marked with the same ID value. Final object pose uncertainty is depicted with the yellow ellipses drawn around the mean final poses. This is what makes our cases *probabilistic* ones as the output has uncertainty. The process of acquiring these experimental models is elaborated in Section 3.3.2.

As the number of the stored cases grow, processing efficiency and scalability starts to become a problem, as the corresponding sequences are recorded at each step of the robot's perception cycle, in our case at a frequency of 30Hz. In order to improve the efficiency of collision checking along the robot's and the object's trajectories, we define *keyframes* at every k^{th} frame of the sequence and perform collision checking only for the keyframes. The value of k can be adjusted according to the dimensions of the object being pushed; that is, the smaller the object, the better to check collisions more frequently along the trajectories.

3.3.2. Building Experimental Interaction Models

There are two challenges that the robot needs to handle while building experimental models that describe the outcomes of its interactions with the objects:

(i) The first challenge is the uncontrolled motion of the object after the push is ceased. As a result of moving on passively-rolling caster wheels, the pushable objects used in our experiments do not stop immediately after the robot stops pushing, and the exact poses of the objects after they come to rest vary even between the pushing attempts from the same direction for the same duration.

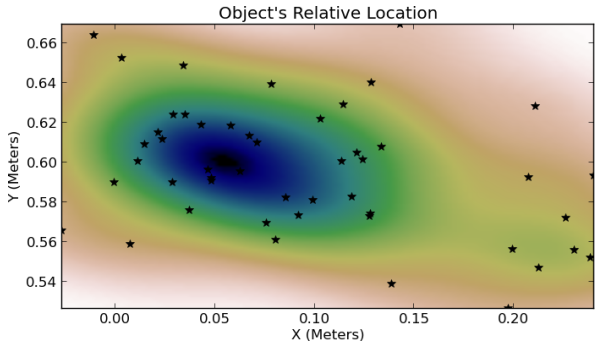
(ii) The second challenge is the effect of the initial stationary orientations of the object's caster wheels on the trajectory that the object follows while being pushed. The wheels do not immediately align with the pushing direction after the contact between the robot and the object is established, which introduces additional uncertainty to the motion of the object and its final observed pose.

We address these partly interrelated problems simultaneously by having the robot build its experience *incrementally* over several trials instead of relying on a single observation.

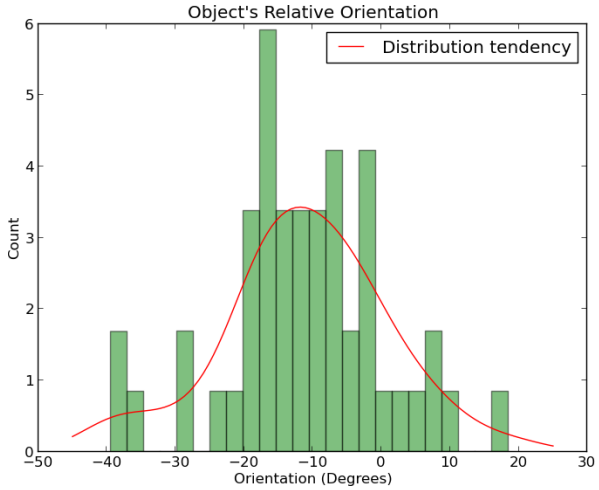
When the robot is asked to acquire experience about a given pushable object through self-exploration, it determines m random push initiation locations immediately around the object together with the corresponding random pushing durations ranging from 1 to 3 seconds. We name these tuples *push configurations*, $\varsigma = \{\varsigma_0, \ldots, \varsigma_{m-1}\}$, $\varsigma_i : (\wp_{R_0}, t)$, which are used to carry out the first push trials on the object. Each ς_i represents a simple linear push performed while moving with constant velocity. On the other hand, we can also demonstrate the robot more sophisticated and informative pushing motions for the given objects via the use of a joystick. Regardless of the method that the robot uses to acquire its object-specific experience, a new case is created and saved whenever the robot tries a particular push for the first time. The additional trials are merely reiterating the sequences of these newly acquired cases to update the parameters of the distributions associated with each of them that represent the uncertainty in the observed final pose of the relevant object after a push.

Figure 3.4 illustrates the visualization of the actual observed relative final object pose data recorded during the execution of one of the several memorized sequences for the chair object. As an attempt to capture this motion uncertainty caused by the caster wheels, the robot experiments with each S_i^{new} of the newly gathered set of sequences S^{new} for varying initial wheel orientations. If there are more than two newly acquired sequences, then the robot iterates over S^{new} by using a set of increments $\iota = \{\iota_0 = 1, \ldots, \iota_j = |S^{new}| - 1\}$ in a way similar to a hash collision resolution strategy. Starting with $\iota = \iota_0$, the robot alternates between S_i^{new} using $i = ((i + \iota) \mod |S^{new}|)$ until each of them are covered. Then it keeps picking other increments ι_l with $l = ((l+1) \mod (j+1))$ and continues its experimentation until n samples from each of the S_i^{new} are collected. If there are only one or two newly acquired sequences, then the robot either reiterates some of the other already existing sequences or executes random pushing motions to alter the initial orientations of the wheels.

Based on the visualizations of the actual data shown in Figure 3.4, we decided to use 3-dimensional Gaussians, for the sake of simplicity, to approximate the distributions of the 3 DoF final object poses represented as $\langle x, y, \theta \rangle$. During the collection of these n samples for each S_i^{new}, the corresponding distribution parameters are incrementally



a) Relative location



b) Relative orientation

Figure 3.4. Actual distribution of the object's relative final pose (location and orientation) when one of the sequences is reiterated several times. (a) Stars represent relative locations and colors represent distribution density. (b) Observed object orientation is discretized into bins to obtain a final relative orientation histogram.

updated according to Equation 3.1 and Equation 3.2, assuming that the observed final object poses will be normally distributed, as Figure 3.4 roughly suggests.

$$\bar{\wp}_{O_t^i} = \bar{\wp}_{O_{t-1}^i} + \frac{\wp_{O_t^i} - \bar{\wp}_{O_{t-1}^i}}{t} \tag{3.1}$$

$$\Sigma_{\wp_{O_t^i}} = \frac{(t-1)\Sigma_{\wp_{O_{t-1}^i}} + (\wp_{O_t^i} - \bar{\wp}_{O_t^i})(\wp_{O_t^i} - \bar{\wp}_{O_{t-1}^i})^T}{t} \tag{3.2}$$

In these equations, $\bar{\wp}_{O_t^i}$ denotes the mean of the observed final object pose after the t^{th} trial for a specific S_i^{new}, and $\Sigma_{\wp_{O_t^i}}$ is the corresponding covariance, which in our case is a 3×3 matrix as we are dealing with 3 DoF poses in the form of $\langle x, y, \theta \rangle$. This compact representation eliminates the need for storing all of the previously observed individual poses.

These distributions are also good indicators of how reliable and consistent individual push sequences are. Since the object moves in an uncontrolled manner after the pushing is ceased, we do not want it to end up in an unforeseen pose which may happen to collide with the obstacles or the other objects in the environment, or cause the next pushing motion in the plan to be unachievable due to the obstruction of the corresponding push initiation pose. Therefore, we eliminate the sequences with variances exceeding predefined thresholds to improve the safety and reliability of the plans generated using these sequences, preventing potential failures and reducing the number of re-plans needed along the way during plan execution.

Similar to incrementally building these distributions, it would be possible to update the object trajectories themselves by aligning them with the recently experienced ones using a Dynamic Time Warping (DTW) [24] approach, and computing the mean object trajectory accordingly. However, since the object trajectories are relatively short and the obstacles in the environment are large, it is plausible to assume that the trajectories observed at the very beginning of the case acquisition process are decent representatives of the ones to be observed in the future. This assumption simplifies the computational complexity of our contributed method while not degrading the overall performance of the system significantly.

3.4. Case-Based Reactive Push-Manipulation

One possible way of using the experimentally acquired interaction models (i.e. probabilistic cases) for push-manipulation is to plan an arbitrary collision-free guideline path for the object without worrying about whether it can actually be followed using the available sequences, and then reiterating the ones that result in the *best locally-*

matching non-colliding object trajectories until the goal is reached. First of all, the sequences with obstructed robot trajectories are filtered out as it would not be possible to reiterate them. As the final placement orientation is important in our problem, the next check is performed to see if the current orientation of the object differs from the goal orientation by an amount greater than the allowed tolerance. If that is the case and the local environment of the object has enough space for the required maneuver, then the sequence that will reduce the orientation difference the most is selected for execution. If, on the other hand, the orientation difference is still acceptable, this time *cosine similarities* between the directions of each of the non-colliding object trajectories (τ_i) and the direction of the next waypoint on the guideline path ($\mathcal{G}$) are computed as shown in Equation 3.3. The sequence that results in the object trajectory with the greatest similarity to the guideline is selected for execution.

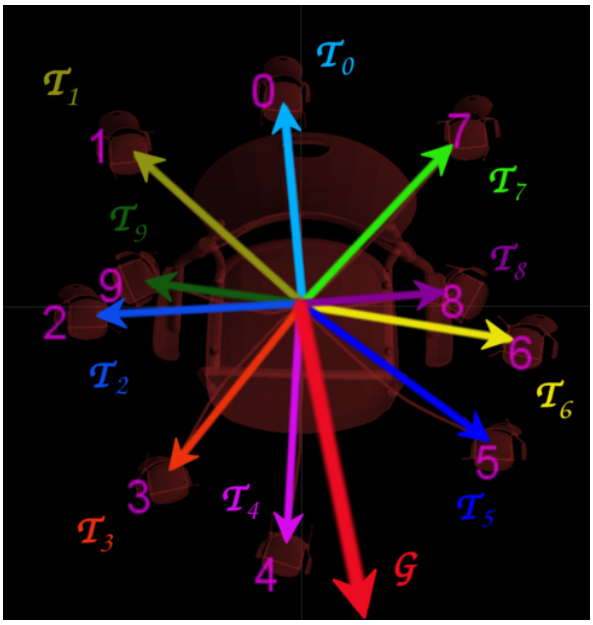


Figure 3.5. The anticipated object trajectories are approximated with vectors (marked with τ_i) and compared against the vector representing the desired moving direction (the red vector marked with $\mathcal{G}$) using cosine similarity. τ_4 is the most similar direction to the desired one in this particular figure.

$$sim(\tau_i, \mathcal{G}) = \frac{\tau_i \cdot \mathcal{G}}{||\tau_i||\,||\mathcal{G}||} \tag{3.3}$$

To better visualize the concept, Figure 3.5 shows the bird's-eye view of the object trajectory components of a set of sequences learned for a chair together with the direction of the next waypoint along the guideline path generated at the very beginning of the process. The direction vectors are depicted as arrows superimposed on the anticipated and desired trajectories. Trajectories with IDs 8 and 9 are the results of the rotational movements of the robot; hence, they are usually more suitable for reducing the orientation difference between current pose and the goal pose of the object.

The sequence selected based on its directional and rotational similarity to the desired intermediate state may result in collision of the object with the environment when executed all the way to the end. It is important to keep in mind that the elegance of manipulation by pushing comes with the potential danger of *irreversibility*; that is, the robot may push the object to such an inconvenient location that it may not be able to recover. For that reason, it becomes very important to be able to control the amount of movement so that the object is pushed only so much that it does not collide with anything, and ends up in a state that is as close to the desired one as possible. Therefore, in addition to the directional and rotational similarity checks, the robot also tries to find the best matching and collision-free projected pose (i.e. keyframe) of the object along the trajectory and stops pushing right at the corresponding moment.

3.4.1. Experimental Evaluation

The Webots simulation environment [31] enabled us to realistically simulate the passively-mobile real world objects and their motions on caster wheels (described in APPENDIX A). Even though we used identical caster wheels for all the pushable objects, their actual physical structures, weight distributions, and the varying wheel placements cause them to have distinct motion characteristics. For simulating the 2-axes rotation of the caster wheels, we set the Coulomb friction coefficient for the wheel axis as 0.1 and for the fork axis that rotates the wheel vertically as 1.0. Providing reasonable mass and friction parameters to Webots' physics simulator, Open Dynamics Engine (ODE), results in reasonably realistic object behaviors, although we are not particularly concerned with the accuracy of these parameter values.

In the simulated setup, instead of having the robot acquire it push-manipulation experience via random interactions, we demonstrated via joysticking a total of 10 pushing sequences to the robot; four linear pushes from the four main directions, four diagonal pushes, and two rotational pushes from either side of the object as shown in Figure 3.6d. The final placement of an object was considered successful if the distance to the desired goal location was below $0.15m$, and the orientation difference was below $\pi/6$ radians. In the experiments performed with the chair, the robot was able to place the object to its desired pose safely 6 out of 10 times. [4]

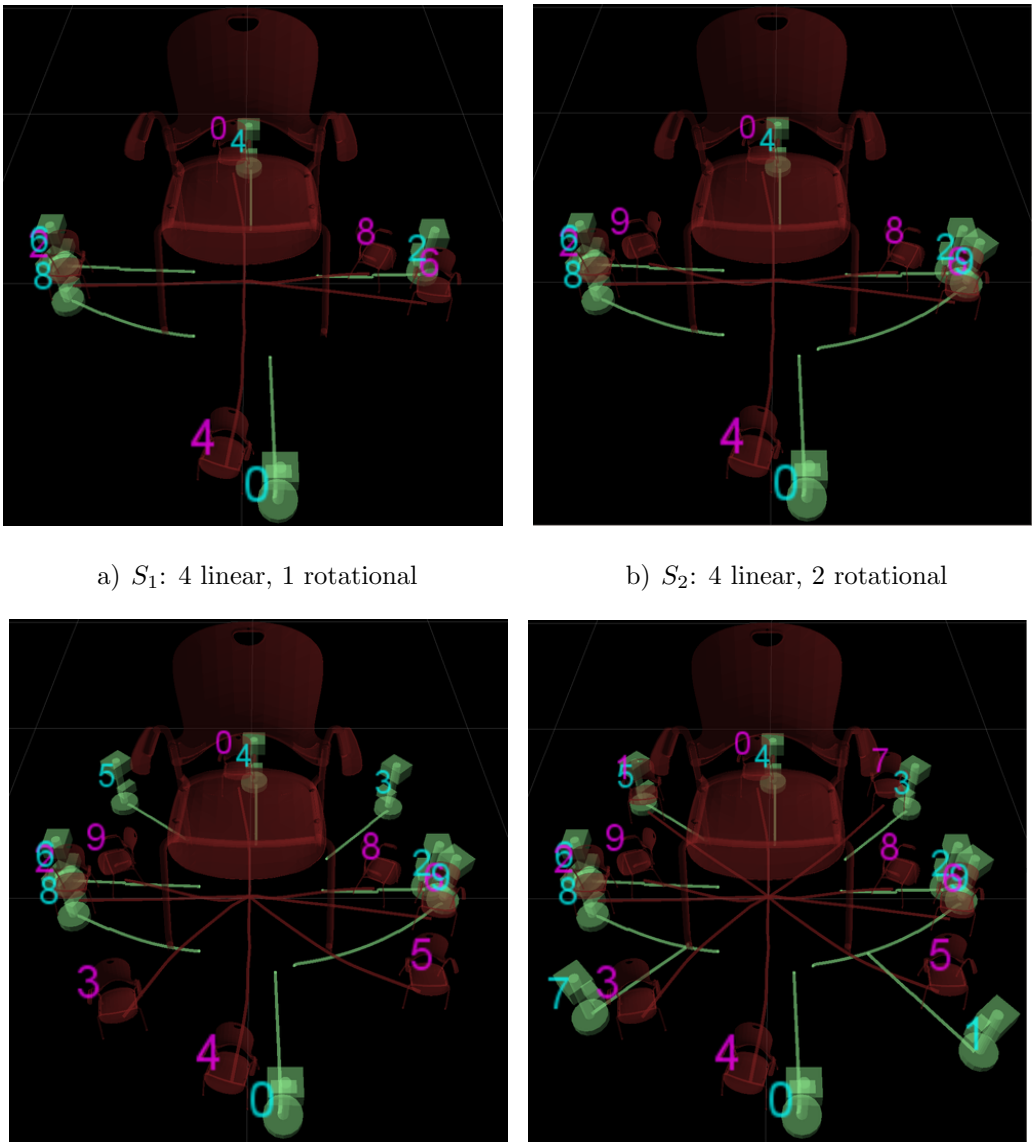
We also investigated the effects that the number and types of the available pushing sequences have on the overall success of the approach. Figure 3.6 shows the four different sets of the motion sequences that the robot was allowed to use. In a sample setup, we placed the chair rotated $\pi/2$ radians counterclockwise 3 meters away from the desired goal configuration so that the robot would need to perform rotational as well as translational manipulation moves in order to bring the chair to its desired pose. As we utilized a randomized generative planner to construct the guideline path, we repeated our experiments 10 times with each of the four sets of sequences. The average distances and orientation differences obtained with each of these sets as well as the number of pushes required to obtain these results are summarized in Table 3.1.

Table 3.1. Performances of different sets of motion sequences.

Sequence set	Push count	μ_{dist} (cm)	$\mu_{\triangle\theta}$ (deg)
S_1	25	14.11	26.12
S_2	8	14.82	14.06
S_3	7	14.87	23.52
S_4	7	14.86	20.49

Among these sets, S_1 only had a counterclockwise rotational pushing sequence in addition to four linear pushing sequences; therefore, it took a lot more pushes for the robot to bring the chair to the desired orientation by using the available sequences. Looking at Table 3.1, the general tendency seems to be towards a decreasing number of pushes as the translational and rotational variety of the available sequences grows.

[4] A video showing the robot performing case-based reactive push-manipulation can be seen here: `http://youtu.be/1Z8KW7fPGrA`



a) S_1: 4 linear, 1 rotational

b) S_2: 4 linear, 2 rotational

c) S_3: 4 linear, 2 diagonal, 2 rotational

d) S_4: 4 linear, 4 diagonal, 2 rotational

Figure 3.6. The four different sets of sequences used in the experiments. Among these sets, S_1 is the most challenging one as it has only one rotational pushing sequence, which makes it difficult to correct for clockwise orientation differences.

In some challenging setups, we observed insufficient variety of sequences resulting in situations where the robot got stuck due to either potential collisions, or having all the push initiation poses obstructed. Those were the main reasons of the robot failing to safely transport the object to its desired pose 4 out of 10 times in the aforementioned experiments. The success rate could be increased by increasing the number and the variety of sequences. However, the best results are obtained when achievability is taken into account during planning, which is explained in the following section.

3.5. Case-Based Achievable Push-Manipulation

The experimentally acquired probabilistic cases present primitives that can be used for building various kinds of graphs to search for the path to the requested goal pose in the task environment instead of trying to follow guideline paths that are arbitrarily generated in an uninformed manner. In order to maintain a small computational footprint, we take a sampling-based approach to planning the push-manipulation path for the primary OOI. We modify the original Rapidly-exploring Random Tree (RRT) algorithm [12, 13] and use the previously acquired probabilistic cases as building blocks for constructing the tree [9]. As opposed to the potentially linear δ extensions towards the random samples in the original RRT algorithm (Figure 2.3), our experience-based version, *Exp-RRT*, uses the previously observed object trajectories as building blocks for extending the tree towards the sample. Since the probabilistic cases encode the motion of both the robot and the OOI together with the associated uncertainty, using them to construct the push plan ensures *achievability* from both the robot's and the OOI's perspective in terms of motion feasibility and collision-safety. The pseudocode for Exp-RRT is given in Figure 3.7. At each iteration, we sample a random pose with probability p or use the goal as the sample with probability $1 - p$. The "nearest" node of the tree to the new sample is the one that gives the maximum similarity value according to the similarity function we define as in Equation 3.4,

$$sim(p_1, p_2) = \frac{d_{max}}{dist(p_1, p_2)} cos(p_1.\theta - p_2.\theta) \tag{3.4}$$

where d_{max} is the maximum possible distance that can be obtained in the task environment and $dist(p_1, p_2)$ is the Euclidean distance between the locations of the poses. Therefore, the closer the locations of the two poses and the smaller the angular difference between their orientations, the more similar they are. The nearest node is queried via the *MostSimilar* function in the algorithm. After the most similar node to the sample is determined, this time each of the collision-free final expected poses of the

1: **function** BUILDEXPRRT(ooi, q_{init}, q_{goal})
2: $Tree \leftarrow q_{init}$; ▷ set the root of the tree as the initial configuration
3: $q_{new} \leftarrow q_{init}$;
4: **while** $sim(q_{new}, q_{goal}) < THRESHOLD$ **do** ▷ while the goal is not reached
5: $q_{rand} \leftarrow Sample()$; ▷ sample a random configuration
6: $q_{most_similar} \leftarrow MostSimilar(Tree, q_{rand})$;
7: $q_{new} \leftarrow Extend(ooi, q_{most_similar}, q_{rand})$;
8: $Tree.add(q_{new})$; ▷ add the new configuration to the tree
9: **end while**
10: **return** $Trajectory(Tree, q_{new})$; ▷ return solution
11: **end function**
12: **function** MOSTSIMILAR($Tree, q_{target}$) ▷ find "most similar" configuration
13: **return** $\arg\max_{q \in Tree} sim(q, q_{target})$;
14: **end function**
15: **function** EXTEND($ooi, q_{source}, q_{target}$) ▷ extend using the best sequence
16: $S_{trans} \leftarrow \{Transform(S_i, q_{source})\} \forall S_i \in ooi.S$;
17: $S_{safe} \leftarrow \{S_{trans} \setminus \{Colliding(S_t)\}\} \forall S_t \in S_{trans}$;
18: **return** $\arg\max_{S_i.q^f \forall S_i \in S_{safe}} sim(S_i.q^f, q_{target})$;
19: **end function**

Figure 3.7. The Exp-RRT algorithm.

sequences originating from the pose of the most similar node are compared against the sample, again using the similarity function defined in Equation 3.4. The tree is extended towards the sample by using the final estimated object pose of the sequence that gives the highest similarity value and is collision-free for both the object and the robot, which is performed by the *Extend* function in the algorithm. This process is repeated until the pose of the newly added node falls within predefined distance and orientation difference limits to the goal pose. Figure 3.8 illustrates two steps of the tree construction process, assuming, for the sake of simplicity, that the goal itself is used as the sample to be reached. Object trajectories within the sequences are illustrated as dashed curves and the estimated final object poses are depicted as little squares.

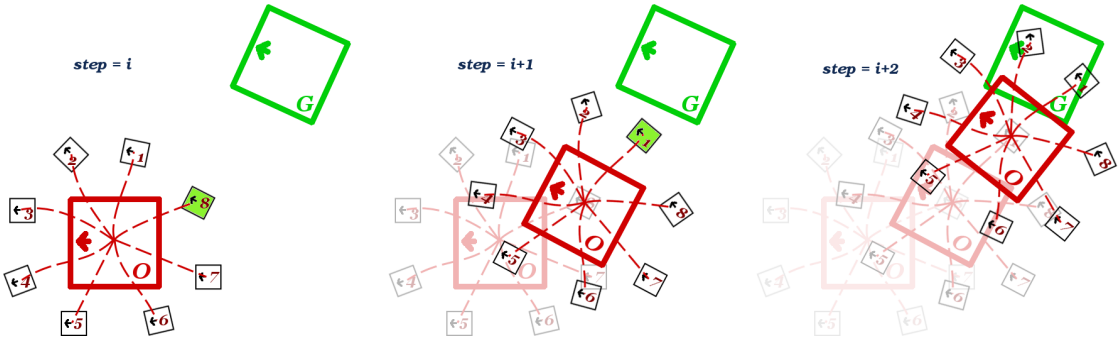


Figure 3.8. Illustration of the Exp-RRT construction process. The sequences resulting in the most similar poses are highlighted.

During tree construction, the suitability of a particular sequence for extending the tree is determined by evaluating its achievability in addition to its final pose similarity to the sample. The achievability of a sequence is determined by checking each keyframe along the robot and object trajectories within the sequence for potential collisions. Additionally, in order to incorporate the motion uncertainty of the object into the planning process, collision check for the final expected object pose is performed using the associated distribution rather than the mean observed pose only. For this purpose, we derive $2L + 1$ sigma points representing the extremes of the distribution from the mean and the covariance using Equations 3.5-3.7, where L is the dimensionality of the state space. In our case $L = 3$ as we are dealing with 3 DoF poses.

$$\chi_0 = \bar{\wp}_{O^f} \tag{3.5}$$

$$\chi_i = \bar{\wp}_{O^f} + \zeta(\sqrt{\Sigma_{\wp_{O_n^f}}})_i, i = 1, \ldots, L \tag{3.6}$$

$$\chi_i = \bar{\wp}_{O^f} - \zeta(\sqrt{\Sigma_{\wp_{O_n^f}}})_i, i = L+1, \ldots, 2L \tag{3.7}$$

In these equations, $\bar{\wp}_{O^f}$ is the mean of the final object poses observed so far for a particular sequence, $(\sqrt{\Sigma_{\wp_{O^f_n}}})_i$ is the i^{th} column of the matrix-square-root of the covariance matrix $\Sigma_{\wp_{O^f_n}}$, and ζ is the scalar scaling factor that determines the spread of the sigma points around $\bar{\wp}_{O^f}$. Increasing ζ increases the conservativeness of the planner. In our experiments, we used $\zeta = 3$. Each of these extreme poses are checked for collision and the corresponding probabilistic case is marked as *unachievable* and not used for extending the tree if any of these poses are in collision with the objects in the environment (Figure 3.9).

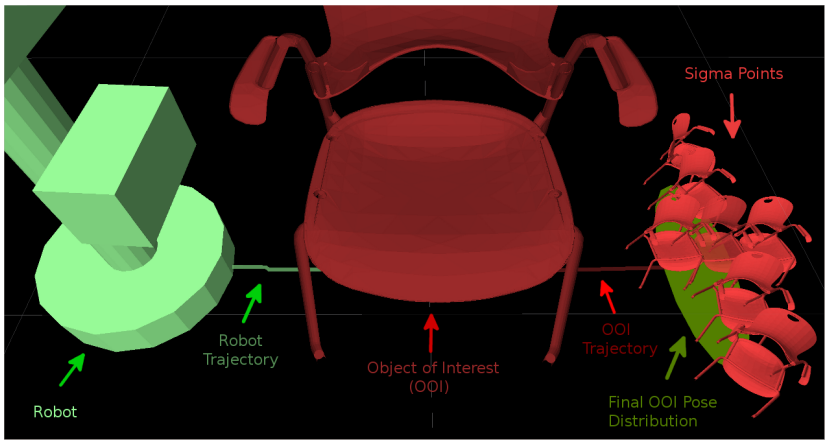


Figure 3.9. The *sigma points*, illustrated as scaled-down OOI figures, indicate the extreme poses (i.e. the extreme points on the corresponding distribution) that could be observed after a push. During plan construction, each of these poses are collision checked in order to be prepared for the worst case scenario during plan execution.

3.5.1. Execution Monitoring

Taking uncertainty into account during RRT planning has been studied in the literature [50–52], however, we do it for the manipulated objects instead of the robot itself in addition to performing it in a novel way. Instead of propagating uncertainty along the tree, we take a more optimistic approach, and extend branches from the means of the distributions. The constructed plan is executed by replaying one after another the robot trajectories of the chain of sequences that transports the object to the desired pose. Even though the plan is constructed by taking into account the uncertainties in the expected final object poses, the object inevitably digresses from its foreseen path, especially when it needs to be transported for a long distance. During

plan execution, re-planning may be triggered depending on whether the actual observed final pose of the object after a push falls within the *tolerance region* of the expected pose distribution, which is computed using Equation 3.8

$$(\wp_{O_o} - \bar{\wp}_O)^T \Sigma_{\wp O}^{-1} (\wp_{O_o} - \bar{\wp}_O) \leq \chi_k^2(p) \tag{3.8}$$

where $\wp_{O_o}$ is the observed final pose of the object, $\bar{\wp}_O$ is the expected final pose, $\Sigma_{\wp O}$ is the expected final pose covariance, and $\chi_k^2(p)$ is the quantile function for probability p of the chi-squared distribution with k degrees of freedom. In our case $k = 3$ and we use $p = 0.05$ to make sure that the observation is statistically significantly different from the expectation for the robot to trigger re-planning.

Additionally, in order to relax the planning process a bit, we design a heuristic that dynamically alters the desired final pose accuracy depending on the distance of the object from the goal, as defined in Equation 3.9 and Equation 3.10.

$$\delta = (dist(\wp_{O_o}, \wp_{O_g})/d_{max}) + \delta_{max} \tag{3.9}$$

$$\omega = \pi(dist(\wp_{O_o}, \wp_{O_g})/d_{max}) + \omega_{max} \tag{3.10}$$

where $\wp_{O_g}$ is the goal pose, δ and ω are the distance and orientation difference thresholds, respectively, and δ_{max} and ω_{max} are the maximum allowed final distance and orientation difference thresholds, respectively. This heuristic helps the robot plan a "rough" solution quickly when the object is far from the goal, and enforces more accurate planning at each re-planning attempt as the object gets closer to the goal.

3.5.2. Experimental Evaluation

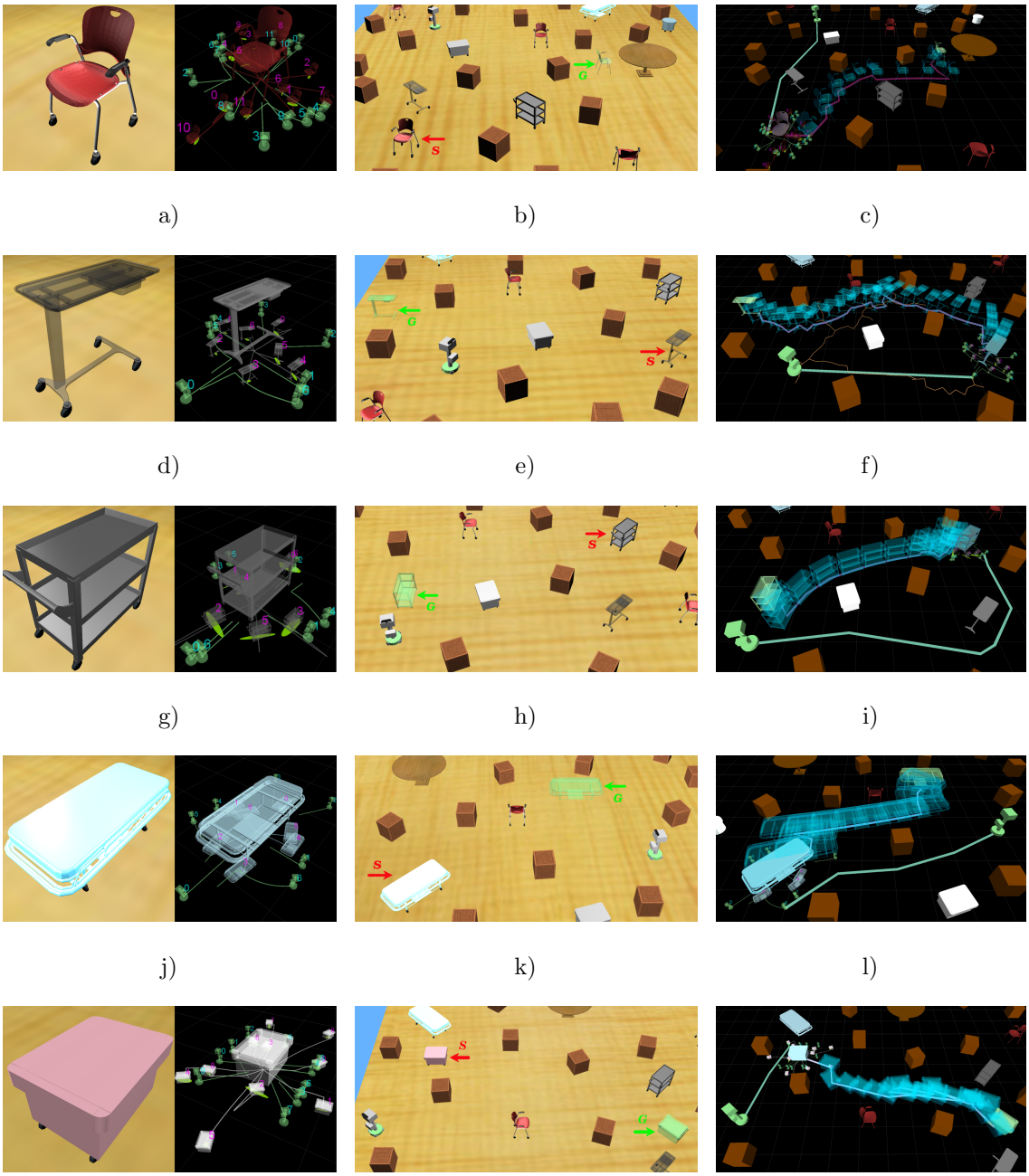
We performed majority of our case-based achievable push planning and manipulation experiments again in Webots. The final placement of an object was considered successful if the distance of the object to the desired goal was below $0.2m$ and the orientation difference was below $\pi/9$ radians. Considering the dimensions of the objects that our robot is expected to manipulate, such as a $0.8m \times 0.45m$ serving tray and a $1.9m \times 0.9m$ stretcher, these constraints are quite tight.

Separate sets of cases are acquired and stored for each of our pushable, passively-mobile objects. As briefly mentioned in Section 3.3.2, the first step before performing any evaluation is to select a *reliable* set of cases to be used for planning. We do that by eliminating the ones that cannot be reiterated consistently; that is, the ones that have high variance in the final observed object pose. We determined the maximum allowed position and orientation variances to have the same values as δ_{max} and ω_{max}.

Figure 3.10 visualizes the generated achievable and collision-free plans from initial (S) to goal (G) poses for five of our pushable objects, namely a chair, an overbed table, a serving tray, a stretcher, and a cart by using their corresponding probabilistic cases as building blocks. The cart is a particularly challenging one as only the two front wheels are casters and the rear ones are stationary, resembling the wheel configuration of a traditional shopping cart. This wheel configuration results in totally different motion characteristics. The purely observation-driven nature of our method enables the robot to acquire and make use of probabilistic cases that define the behavior of even these kinds of objects with different kinematic and dynamic properties.

As it can be seen from these screenshots, our experiment environment is much bigger and much more cluttered compared to the problem setups used in many of the related studies surveyed in Section 3.2. All of the generated plans were successfully executed in simulation without any failures. [5]

[5] A video showing the simulated CoBot successfully push-manipulating an overbed table can be seen here: `http://youtu.be/rN22PSjsniY`



a) b) c)

d) e) f)

g) h) i)

j) k) l)

m) n) o)

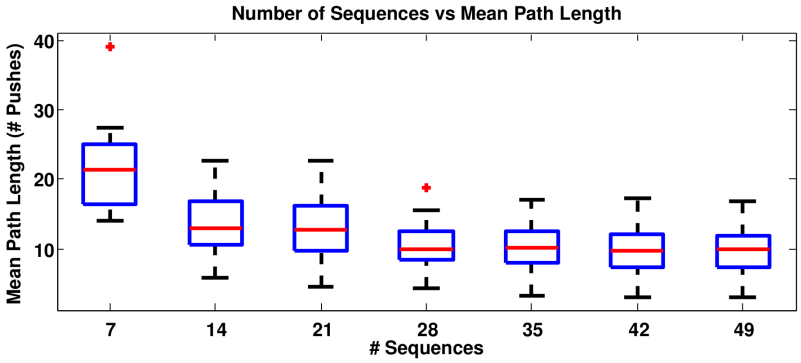
Figure 3.10. Collision-free and achievable push-plans generated (shown as blue ghost figures over the corresponding path) using the probabilistic cases stored for various pushable objects, namely a chair ((a), (b), and (c)), an overbed table ((d), (e), and (f)), a serving tray ((g), (h), and (i)), a stretcher ((j), (k), and (l)), and a cart ((m), (n), and (o)) in large-scale, cluttered environments.

In our first set of experiments, we evaluate the effect of the number and variety of object-specific probabilistic cases on the quality of the successfully generated solutions

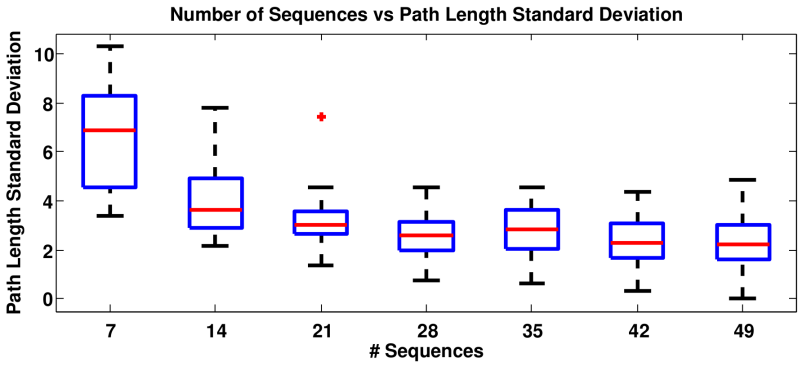
for randomly determined goal configurations. The solution quality is measured in terms of the path length (i.e. the lowest the number of pushes required to transport the object, the better the solution is) and having consistently similar path lengths in multiple trials. Starting with an empty one, we evaluate the proficiency of the available case-base in generating efficient solutions for each of the 20 randomly generated collision-free goal configurations in the task environment by adding new probabilistic cases to it one batch at a time. A batch size of $m = 7$ was used for this set of experiments. After the addition of each new batch to the case-base, 10 planning attempts were made for each of the goals in order to capture the corresponding statistics better as the Exp-RRT planner is a random sampling-based planning algorithm. During the evaluation and the actual planning processes, we consider a planning attempt unsuccessful if the total number of RRT nodes allowed is exceeded. In our experiments, we determined the maximum number of Exp-RRT nodes to be 50625 as we require 0.2m distance accuracy with at most $+/-\pi/9$ radians orientation difference in a $15m \times 15m$ environment.

Figure 3.11a shows how the mean path length computed over all 20 goals changes with the changing number and variety of the cases in the robot's case-base. It can easily be seen from the figure that the mean path length decreases with the increasing number of available cases for a while and then settles around a certain mean path length value. Figure 3.11b shows how the standard deviation of the mean path length changes with the increasing number of available cases, which is a measure of how consistent the solutions are in terms of path length. Similarly, we can interpret from the figure that the robot starts finding solutions that have consistently lower path lengths as the variety of the available cases increases. In case the robot is concerned about storing a minimal number of useful cases, these figures are good indicators for the robot to understand when it has learned enough variety of sequences to solve a decent number of push-manipulation problems for a specific object. The experiments with the other pushable objects in our task environment resulted in similar plots; therefore, we present only the results obtained for the chair object.

The aim of our second experiment was to understand how the number of practices per sequence (corresponding to the t in Equation 3.1 and Equation 3.2) affects the



a) Change of the mean path length computed over 20 goals with the increasing variety of sequences.



b) Change of the standard deviation of the mean path length computed over 20 goals with the increasing variety of sequences.

Figure 3.11. The effects of the growing variety of the available sequences for the chair object on the path length and its consistency.

robustness of the constructed push plan, measured in the number of re-plans performed during execution. We used the chair object as our primary OOI and utilized the set of 7 sequences illustrated in Figure 3.3b. The desired goal pose was approximately $4m$ away diagonally, which corresponds to an average of 12 pushes with the available set of cases. Starting from a minimum of 5 practices per sequence, with an increment of 5, we tried up to 20 practices per sequence, and performed 10 push-manipulation planning and execution experiments with each of these values. Figure 3.12 shows the results, where a general tendency (indicated by the green curve) of decreasing number of re-plans with the increasing number of practices per sequence can be observed. Given the average path length of 12, it can be inferred from the plot that almost 3 consecutive pushes can be achieved before re-planning in case of 20 practices per sequence, whereas that number is below 2 for the case of 5 practices per sequence.

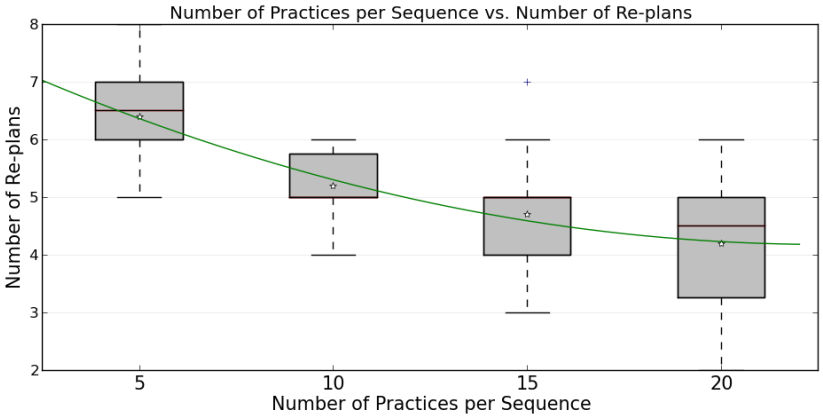


Figure 3.12. The number of re-plans during task execution decreases with the increasing number of practices per individual case. This tendency illustrated by the green curve is an indicator of the corresponding distributions becoming more stable and reliable in their predictive abilities.

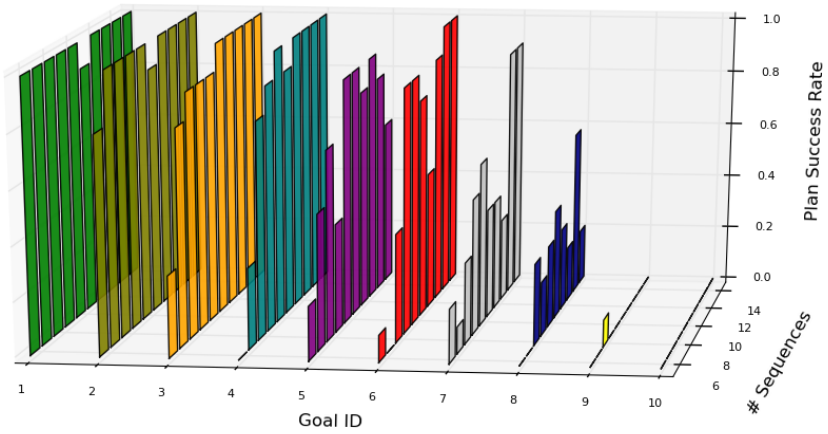


Figure 3.13. The number of cases versus plan success rate for 10 random goals. The general tendency is towards an increased plan success rate with the increasing number of cases. The goals in this plot are sorted based on the total plan success rate.

The last experiment we performed was to see the effect of the number and the variety of the cases in solving various planning problems. For that purpose, we created 10 random goal configurations scattered around the cluttered task environment. Again using the chair object as the primary OOI, we started from 5 cases (i.e. sequences) and went all the way up to 14 cases with increments of 1. Artificially limiting the maximum number of allowed Exp-RRT nodes to 2000, for each number of cases, we performed 10 planning experiments for each of the 10 random goals and measured whether the

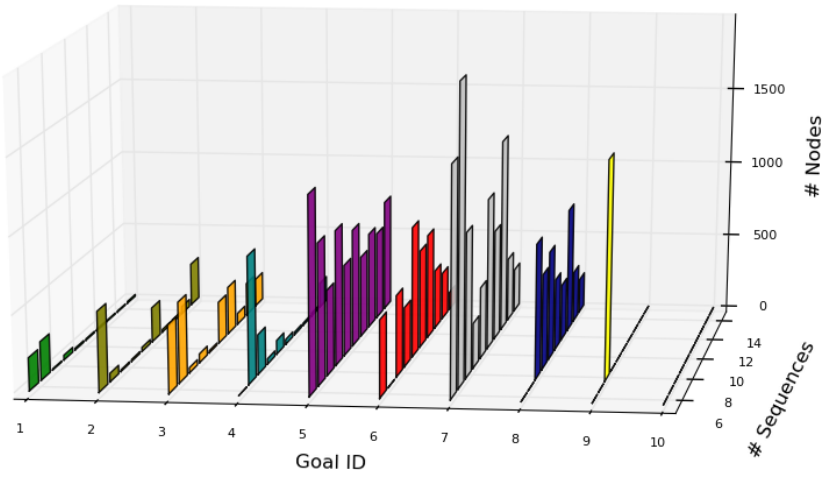


Figure 3.14. The number of cases versus the number of expanded Exp-RRT nodes. The general tendency is towards a reduced number of Exp-RRT nodes with the increasing number of cases.

planner succeeded (1 for success and 0 for failure) and if so how many nodes it had to expand before reaching the goal. We obtained a rough success rate percentage measure by counting the number of successes after each set of 10 experiments per goal and dividing the sum to the number of experiments (i.e. 10). Figure 3.13 illustrates how the plan success rate changes for each of the 10 goals with the increasing number of cases. The figure shows a general tendency towards an increasing success (i.e. reachability) rate for each goal as the number of cases increase. Out of the last two goals where the robot performed the poorest in terms of planning, one of them was at the opposite corner of the environment with many obstacles in between, and the other one was within close proximity of another obstacle, which made it difficult for the robot to generate a collision-free placement plan for these goal configurations. Figure 3.14 visualizes the second measurement in the same experiment scenario, which is the average number of expanded Exp-RRT nodes during the 10 experiments performed for each of the goals. It can be observed in the figure that, for each goal, the number of nodes tends to decrease with the increasing number of cases, which means that a solution can be found quicker with a richer set of cases.

It must be noted that the simulation environment is essentially a black box for the robot, as the real world would be, and the only information that the simulator provides

to the robot are the poses of the objects. It is a black box for us as well. In addition to the object meshes for visualization, we only provide empirically determined mass values and wheel friction coefficients for the ODE physics engine of the simulator to take care of the inter-object interactions to make them behave reasonably realistically. The only motivation behind using a realistic 3D simulator is to obtain a setup that "looks" an "behaves" reasonably realistically as we are not concerned with transferring any knowledge from the simulated environment to the real world or vice versa. In other words, both the internal and external parameters of the simulator are totally irrelevant to the operation of our method. Therefore, even if we had not set the physics parameters realistically, the robot would still be able to learn how to push-manipulate the objects under those circumstances, assuming that they would eventually come to a stop after each push. That makes our method totally independent of the robot, the object, and the environment, both in simulation and in real world.

3.5.3. Mobile Push-Manipulation in Real World

In addition to the detailed study we conducted in simulation, we also ran some preliminary tests in a physical setup to validate our contributed method in terms of its robot, object, and environment (i.e. simulated or real) independence. In this test, our physical CoBot robot [2] was asked to arrange a set of chairs in a predefined seating formation around a round table, some of which were already in place. Figure 3.15 shows a snapshot from one of the physical setups in which we tested our proposed method (described in detail in APPENDIX B).

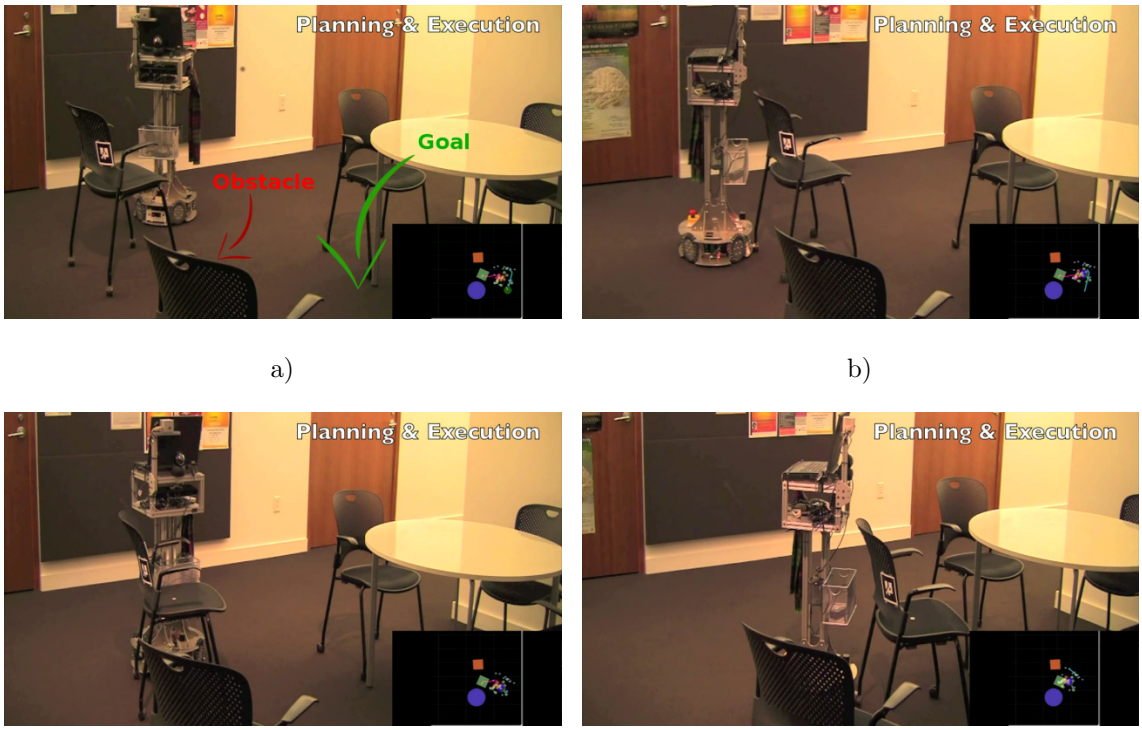
There are a number of challenges that need to be addressed when switching from the simulated environment to the physical one. The first challenge is the construction of a stable world model. In simulation, we get the global pose information of all the objects in the environment directly from the simulator. However, in the physical setup, the robot's global pose information comes from the localization module [53], which is noisier compared to the perfect information received in simulation. The pose of the chair is computed relative to the robot; hence, the calculated global pose of the chair is affected by the noise in the localization estimation of the robot. In order to make it

Figure 3.15. A snapshot from one of the real world tests, where the task of the robot is to arrange the chairs around the round table. Visualization of the setup in simulation is provided on the top left corner of the image.

easier to detect the chair visually, we placed Augmented Reality (AR) tags on both sides of the back of the chair (Figure 3.15), which are visible most of the time from almost all directions. However, perception is not perfect either; therefore, additional noise comes from the perception of the AR tags. The second challenge is the maintenance of the constructed world model at all times. Since the Kinect sensor that we use as the primary visual sensing device is placed at a certain location on the robot with a certain angle to satisfy multiple requirements, and the field of view of the camera is limited, the AR tags cannot be seen anymore when the robot gets very close to the object to push it. A good tracker needs to be employed so that the robot can still have an idea of where the object is even if it is not visible within the robot's field of view.

During our preliminary tests, the robot was, in general, able to construct a stable world model by combining its perception with its localization information to successfully generate and execute push-manipulation plans. Figure 3.16 shows some of the pushing moves that CoBot executed to transport the chair to its desired pose. [6] Even though we have not performed detailed experiments in this setup, we observed that there was an overall increase in the frequency of re-planning due to the increased uncertainty in both perception and action in real world.

[6] A video showing the physical CoBot acquiring and using push-manipulation cases in a real world setup can be seen here: `http://youtu.be/TORQdBPHJ3g`



a) b)

c) d)

Figure 3.16. Snapshots from the case-based push-manipulation planning and execution test in a real world setup. The robot was able to successfully transport the chair to its desired pose right next to the round table.

These preliminary real world tests run using the exact same code base verified the validity of our method and demonstrated its robot, object, and environment (simulated and real) independence as the only pieces of information needed were the robot's localization belief and the pose of the object inferred from the robot's own detection, processing, and transformation of the AR tags associated with the object of interest.

3.6. Discussion

Push-manipulation is one of the most interesting and challenging robotic manipulation modalities that has attracted many researchers. However, many of the proposed methods in the literature handle flat objects with primitive geometric shapes moving quasi-statically on high-friction surfaces, yet they usually make use of complex analytical models or utilize specialized physics engines to predict the outcomes of various interactions. In this chapter, we propose an observation-driven, case-based approach,

which does not require any explicit analytical model or the help of a physics engine. Our mobile robot simply experiments with pushable, passively-mobile complex 3D real world objects to observe and memorize their motion characteristics together with the associated uncertainties resulting from various pushing actions. It then uses this incrementally built experience either for trying to make the object of interest follow a guideline path by reiterating in a reactive manner the sequences that result in the best locally-matching outcomes, or as building blocks of a sampling based planner we contribute, the Exp-RRT, to construct push plans that are safe and achievable. In contrast to the proposed approaches in the literature, in our contribution;

- we handle real world objects with complex 3D structures that may contact the robot on more than one point,
- the manipulated objects move on passively-rolling caster wheels, which introduce two additional sources of motion uncertainty; the effect of the initial orientations of the wheels on the object's trajectory, and the object's continued motion even after the push is ceased,
- the experiment environment is cluttered with obstacles; hence, both collision-free and achievable plans should be constructed and manipulation should be performed delicately,
- we do not use any explicit analytical robot-object interaction models or learn a mapping between the trajectories of the robot and the object; we only utilize the experimented and observed effects of the past pushing motions as discrete probabilistic cases to anticipate the future, plan, and act accordingly.

We extensively tested our method in a realistic 3D simulation environment, where a variety of passively-mobile pushable objects with caster wheels needed to be safely navigated among obstacles to reach their desired final poses. We also performed some preliminary tests in a physical setup to verify the validity of our method. Our experiments demonstrate safe transportation and successful placement of several pushable objects in simulation and promising results for push-manipulation tasks in real world, such as arranging chairs in predefined seating formations in a study area.

4. ADAPTING PAST EXPERIENCE TO NOVEL SITUATIONS

4.1. Motivation

After the initial learning period in a case-based push-manipulation scenario, the acquired probabilistic cases are usually reliable enough to push-manipulate the known OOIs within the environment successfully, requiring re-planning every once in a while during task execution. However, especially considering that all of our potential OOIs are instances of office, hospital, and service furniture, their occupation state, and hence dynamics, may change from time to time as they are mainly intended for transporting various loads. Figure 4.1 depicts such a scenario, where the overbed table is loaded with several objects of unknown masses. This addition to the object changes its dynamics and it starts behaving differently than expected, as illustrated in the figure. Even in such cases where the OOIs are loaded with other objects, we would still like to be able to make as much use of the past experience as possible to solve the novel problem without having the go through the learning process all over again.

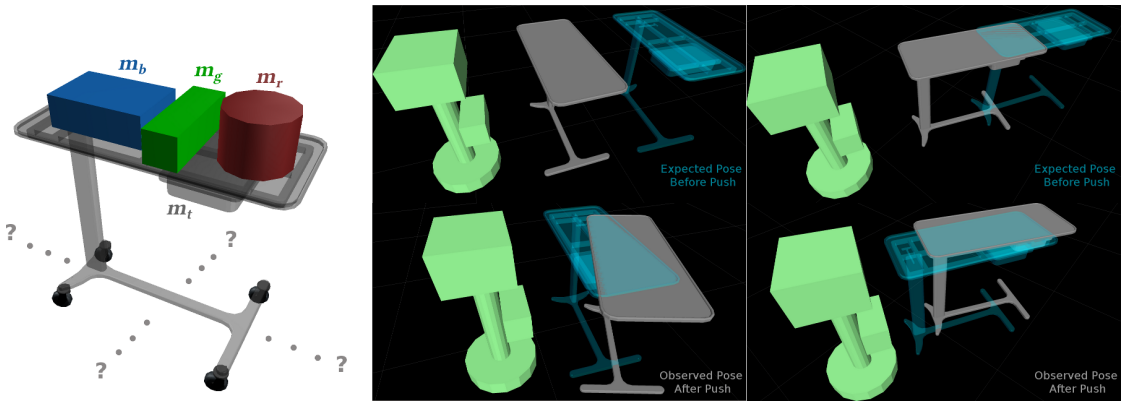


Figure 4.1. A large difference is observed between the expected pose and the realized one when the loaded OOI is tried to be push-manipulated using the past experience obtained from the unloaded OOI. However, the robot can incrementally adapt the corresponding cases to these new observations instead of trying to learn a completely new set of cases for the loaded OOI.

4.2. Related Work

This problem can be viewed as an instance of lifelong robot learning [54]. Along these lines, Sturm *et al.* [55,56] contributes a method that allows the robot to learn and maintain a generative model of its own physical body through self-observation, which helps in detecting and adapting to mismatches between the model prediction and self observation in case there is a change in the robot's physiology. In our approach, the robot observes the changes in the behavior of the object and tries to adapt the experimental models (i.e. cases) pertaining to that particular object in such a way to establish a match between the model-based prediction and observation.

4.3. Approach

To achieve that, we seek a way of modifying the available distributions to quickly accommodate for the difference between the expected and the observed object behavior. However, since the parameters of the distributions associated with individual cases are updated incrementally after each trial during case acquisition, the latest observations start becoming less and less effective in shaping the distributions as the number of trials per case increases. We overcome this problem by introducing a weighting factor to determine how much relative weight the distributions representing the past experience should have over the newly acquired experience. This weight coefficient is computed by evaluating the difference between the expected and the observed pose of the object via the pose similarity function defined in Equation 3.4 and multiplying the obtained value with a scaling factor K, as defined in Equation 4.1. The resulting weight is used as a scaling factor to adjust the influence of the past distribution defined by $\bar{\wp}_{O^f_{t-1}}$ and $\Sigma_{\wp_{O^f_{t-1}}}$, as shown in Equation 4.2 and Equation 4.3. That is, if the difference between the expected pose and the observed one is significant, then their similarity value will be small, resulting in a decrease in the weight of the past distribution parameters. That way, the past experience is treated as if it were formed out of a smaller number of samples in the first place, allowing the distributions to quickly shift towards the new observation. Otherwise, these updates will have a similar effect as the ones performed by the original equations given in Equation 3.1 and Equation 3.2.

$$W = sim(p_{expected}, p_{observed})K \tag{4.1}$$

$$\bar{\wp}^{W}_{O^f_{t-1}} = W\bar{\wp}_{O^f_{t-1}} \tag{4.2}$$

$$\Sigma^{W}_{\wp_{O^f_{t-1}}} = W\Sigma_{\wp_{O^f_{t-1}}} \tag{4.3}$$

4.4. Experimental Evaluation

In our experiment, we investigate how the ability to continuously adapt to the changes regarding the primary OOI (e.g. loaded versus unloaded) affects robust plan execution measured in the number of consecutive pushes before a re-plan is needed. The overbed table was used as the primary OOI and the desired goal pose was placed approximately $10m$ away diagonally, which corresponds to an average of 25 pushes with the available set of cases. The cases were acquired on the unloaded overbed table, and the robot was expected to adapt and utilize them for push-manipulating the loaded one, which has significantly different dynamics as shown in Figure 4.1. Adaptation was performed on the fly during plan execution with the hope that the execution would become more robust after each re-planning attempt.

Figure 4.2 clearly shows that continuous adaptation enables the cases learned on an unloaded OOI to be adapted and used on a loaded one (the left part of the plot) as well as improving the performance during plan execution in the original unloaded OOI manipulation scenario (the right part of the plot). The first box on the left-hand side of the plot shows that the robot pretty much had to re-plan after each push when it tried manipulating a loaded overbed table using the cases acquired on the unloaded one. The second box depicts the performance improvement in the case of continuous adaptation, where an average of more than 2 pushes could be achieved by adapting

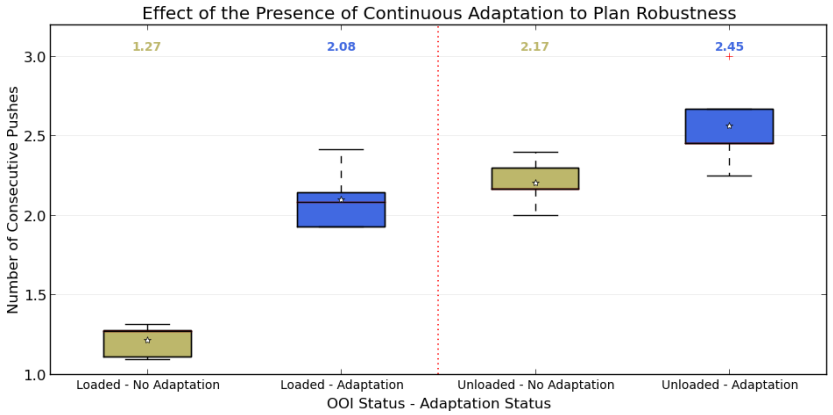


Figure 4.2. Continuous adaptation to the observed changes contributes significantly to the robust execution of the generated plans, both under occupation state changes (left) and during regular operation (right). Median number of consecutive pushes are given at the top.

the cases to the novel situation. This is noteworthy considering that the robot had to adapt to the new dynamics of the OOI on the fly in the course of an average of just 25 pushes in total.

In addition to handling drastic changes in the OOI's dynamics, it is also possible to utilize the flexibility of our incremental learning approach to enable the robot to continuously learn and adapt by updating the parameters of the corresponding distributions after each push during regular task execution. The positive effects of continuous adaptation in such scenarios can be seen on the right-hand side of the plot in Figure 4.2. The first box indicates the push-manipulation execution robustness in the default case, where the cases learned on an unloaded overbed table are being used on the same object without any adaptation. The second box shows that a considerable amount of performance improvement in the predictive abilities of the final object pose distributions (hence increased number of consecutive pushes) can be achieved through continuous adaptation even when the occupation status of the OOI remains the same.

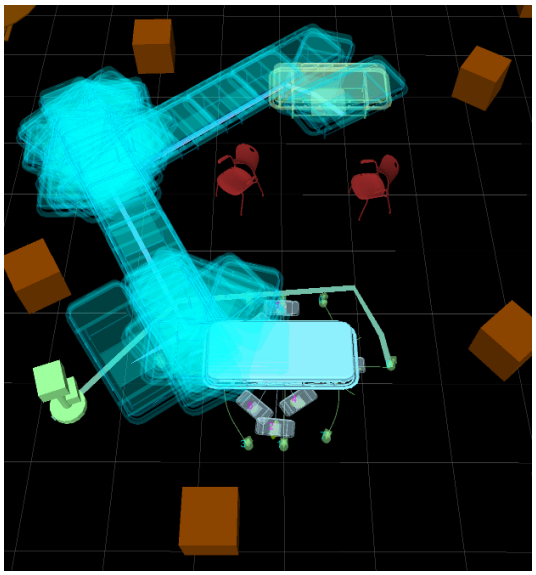
4.5. Discussion

In this chapter, we demonstrate the advantage of having a purely interaction and observation driven case acquisition and tuning approach in adapting the past experience to the novel situations, such as changes in the dynamics of the manipulated objects due to loading/unloading, instead of having to go through the learning process all over for each such novelty. Our experimental results indicate that, after adaptation, the cases acquired for unloaded OOIs could be successfully used for manipulating loaded ones, which have significantly different dynamics. Additionally, we show that the same paradigm can be applied to obtain continuous performance improvement in terms of robust plan execution during regular operation.

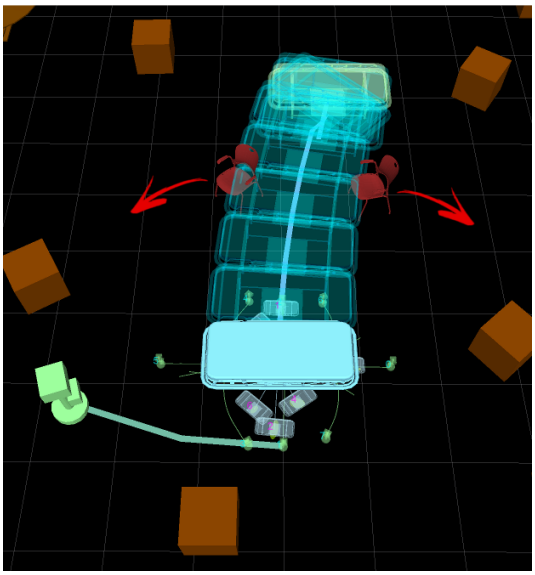
5. MOBILE PUSH-MANIPULATION AMONG MOVABLE OBSTACLES

5.1. Motivation

Having acquired experience about how several objects in its task environment move in response to various pushes enables the robot to modify and rearrange the task environment according to its needs whenever necessary. When the robot knows that it can move some of the obstacles to make way for the primary OOI, the complexity of the task environments can be reduced considerably by treating every movable obstacle as "permeable" while constructing push plans for the primary OOI (Figure 5.1).



a) No utilization of movable obstacles: The robot plans around all obstacles regardless of them being movable or not.



b) Utilizing movable obstacles: The robot treats movable obstacles as "permeable", knowing that it can push them out of the way.

Figure 5.1. Effect of movable obstacle utilization during planning. (a) When all obstacles are treated as static, planning takes longer and results in sophisticated solutions. (b) Planning becomes much quicker and the resulting solutions get simpler and more direct when the movable obstacles are utilized.

5.2. Related Work

In handling movable obstacles, Stilman *et al.* [57–59] contribute practical algorithms for Navigation Among Movable Obstacles (NAMO), which globally reason about free space connectivity and identify which objects to move as well as where to move them starting from the very beginning of the planning project. Rigid grasps are used for pushing or pulling the movable obstacles out of the way. Dogar and Srinivasa [48] use the same framework they developed to perform pre-grasp push-manipulation in a tabletop scenario for rearranging the clutter via single-step linear pushes to be able to reach and grasp the target object.

5.3. Push-Manipulation Among Movable Obstacles

The robot may take advantage of its experience about the other pushable objects in the environment to alleviate the computational load of manipulation planning and execution for the primary OOI. As the robot would know how to push-manipulate them, all the known movable objects in the task environment can essentially be treated as *permeable* during push plan construction for the OOI, allowing the solution paths to pass through them. Before executing each push along the constructed push plan for the primary OOI, the robot checks whether the action would potentially result in a collision with any of the movable obstacles. In case a collision is anticipated, a state lattice graph [60, 61] is expanded for each of the involved movable obstacles. Since we do not know where to push the obstacle ahead of time, full graph construction until candidate collision-free configurations are found is a more suitable approach than using Exp-RRT, which would require a particular goal to be specified. Full planning graphs are expanded by using the relevant probabilistic cases as building blocks until collision-free poses for the movable obstacles are reached. Figure 5.2 illustrates a single branch expansion in the process of state lattice construction for the overbed table object. As opposed to various applications of the state lattices in the literature [62,63], we have no concerns regarding the continuity of the successive motion primitives as our primitives represent discrete achievable motions of the OOI, not the robot. Therefore, none of the cases are eliminated during graph construction, except for the ones that are in

collision with the environment. Among the set of candidate paths that first reach collision-free configurations, the one that pushes the obstacle farthest from the OOI's path as well as all the other obstacles is selected for execution. The push plan for the OOI is ceased until all of the immediately blocking movable obstacles are pushed out of the way according to the generated plan, and it is resumed when the path of the primary OOI is clear. Figure 5.3 shows the stages of movable obstacle clearance during push-manipulation of a stretcher. [7] Even though we cannot claim that our approach to handling push-manipulation among movable obstacles produces optimal plans and results, our experiments demonstrate its practicality as we never observed a failure in our test scenarios.

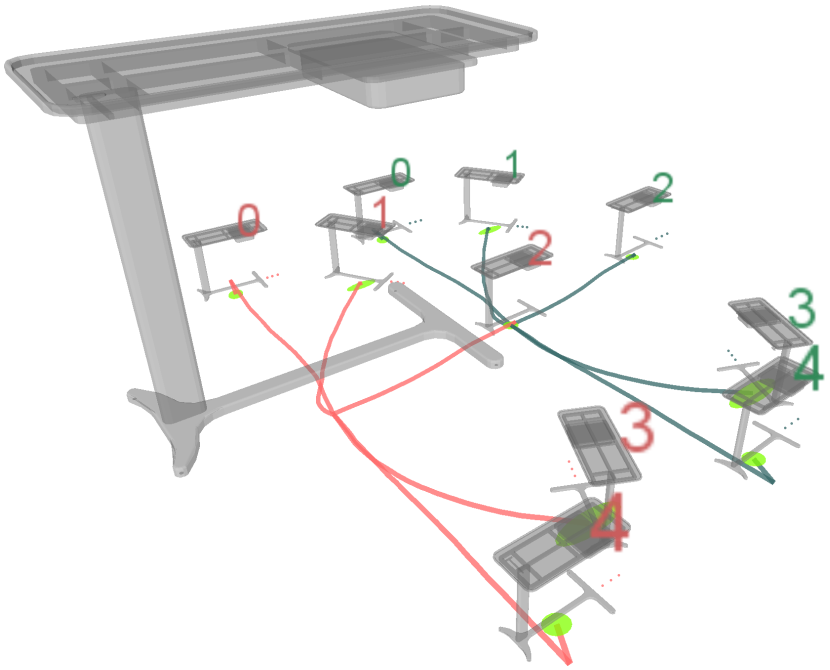
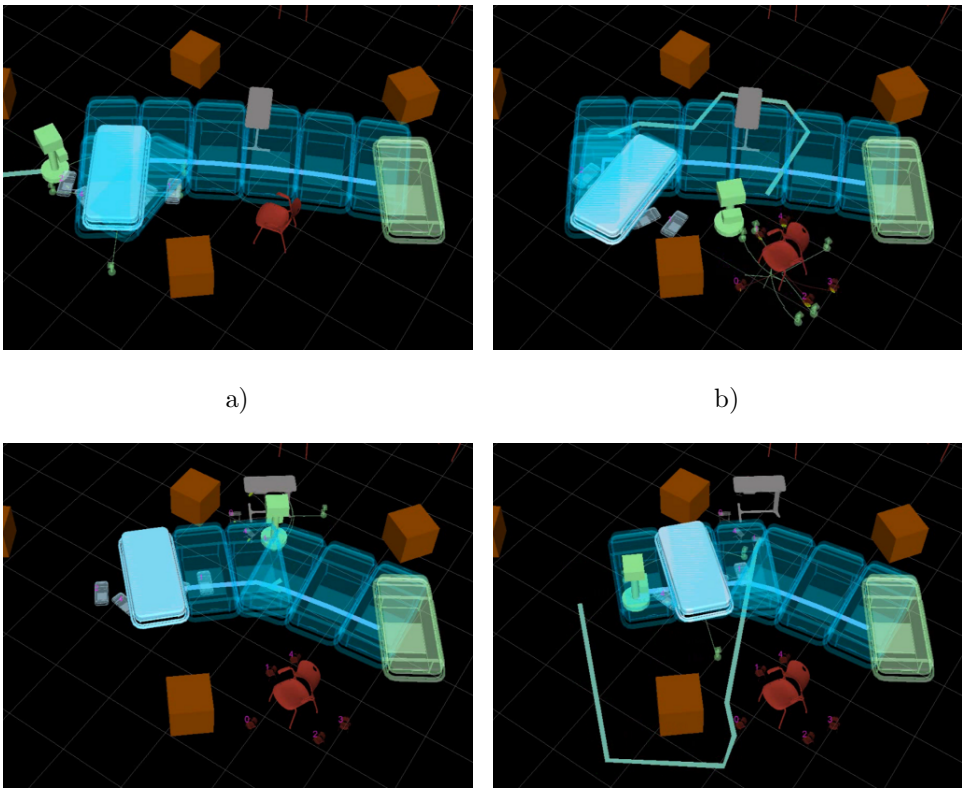


Figure 5.2. The construction of the state lattice is achieved by repeating the set of probabilistic cases at the end of every individual case in a breadth-first manner. Following an optimistic tree construction strategy, each set of branches is originated from the mean values of the final object pose distributions of the previous depth level.

5.4. Experimental Evaluation

We ran two sets of experiments to evaluate the benefits of the method we propose for handling and utilizing the movable obstacles in the task environment.

[7] A video showing experience-based push-manipulation among movable obstacles can be seen here: `http://youtu.be/cvI6YipRCns`

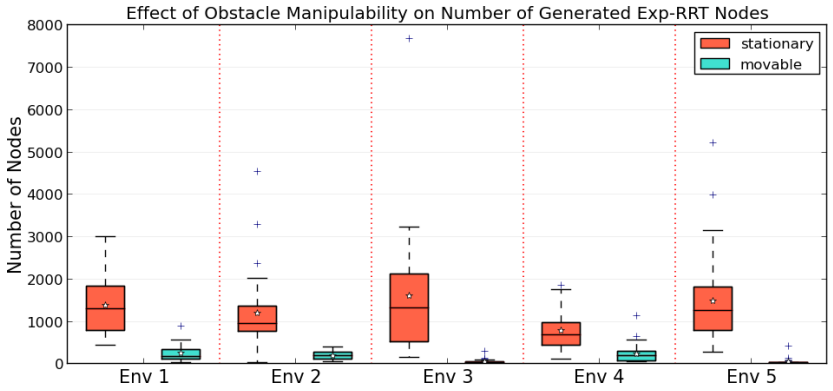


a) b)

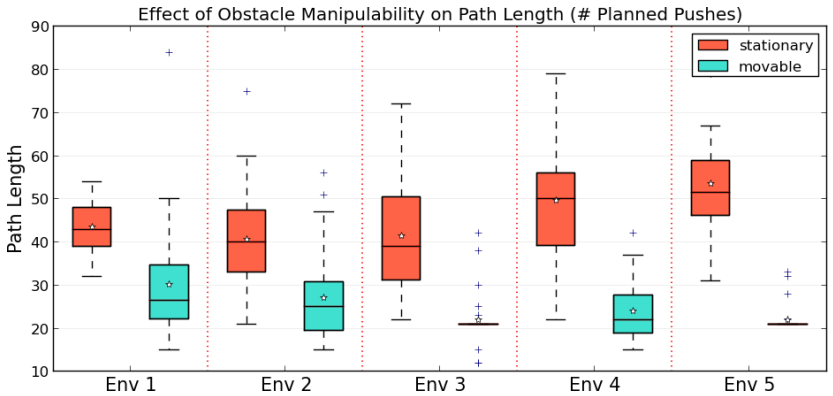
c) d)

Figure 5.3. Clearing movable obstacles along the path of the stretcher. (a) First the stretcher is push-manipulated until a collision is anticipated. (b) The chair is pushed out of the way. (c) The overbed table is cleared after resuming the original push plan and navigating the stretcher a little further. (d) The original push plan for the stretcher is resumed.

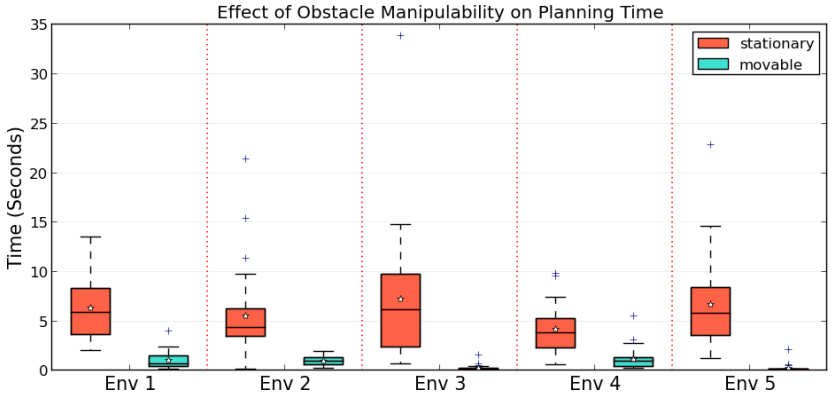
Initially excluding the execution part, we first tried to understand how such capability influences planning performance. We created 5 random test environments and ran 30 planning trials in two different scenarios. In Scenario 1, all the obstacles were treated as stationary, whereas in Scenario 2, some of the obstacles were considered as movable. Planning time is improved dramatically in Scenario 2 as a result of eliminating many of the planning constraints. As Figure 5.4 shows, both the means and the variances of the various performance metrics obtained in Scenario 2 (cyan) are considerably lower than those obtained in Scenario 1 (red). The results are intuitive since the planner needs more effort to find potentially more tortuous solution paths that avoid all the obstacles in Scenario 1, which explains longer planning times and path lengths.



a) Total number of Exp-RRT nodes.



b) The length of the solution path.



c) Planning time.

Figure 5.4. Planning performance in the presence of movable obstacles measured in terms of (a) the total number of Exp-RRT nodes, (b) the solution path length, and (c) planning time.

In the second set of experiments, we additionally tested the task completion times (i.e. both planning and execution) in a random test environment by running 10 trials for each of the two scenarios. Table 5.1 shows a significant difference between the

Table 5.1. Task completion statistics for stationary and movable obstacle configurations.

Obstacle configuration	μ (sec)	σ (sec)	# replans
Stationary (Scenario 1)	187.5	47.33	8
Movable (Scenario 2)	92.05	17.29	5

average task completion times as well as the number of re-plans during the navigation of the primary OOI, which, in this case, was an overbed table.

An interesting application of the movable obstacle manipulation capability would be to transport the primary OOI to its desired destination by "minimally invading" the environment. That is, the robot would try to clean up after itself by trying to push the movable obstacles back as close to their original poses as possible.

5.5. Discussion

The robot can utilize the probabilistic cases acquired for the objects other than the primary OOI to manipulate and re-arrange its task environment to fit its needs. In this chapter, we present an auxiliary algorithm that helps reduce the complexity of the task environment by treating every known movable obstacle as "permeable" during push-plan construction for the primary OOI. Whenever the next push of the OOI is anticipated to result in a potential collision, the robot constructs state lattices out of their corresponding probabilistic cases for each of the movable obstacles that are involved in the anticipated collision until collision-free configurations are reached for all of them. The results of our experiments performed in various scenarios demonstrate the advantage of being able to manipulate the task environment in reducing the complexity of the primary push plans measured in number of Exp-RRT nodes, path length, and planning time.

6. CONCLUSIONS AND FUTURE WORK

We present a case-based approach to achieving practical and efficient mobile manipulation through the utilization of past experience, stored as object-specific, distinct, and potentially probabilistic *cases*. This chapter summarizes and concludes the thesis while reviewing its major scientific contributions and discussing several promising directions for future research.

6.1. Case-Based Mobile Pick and Place

The delicacy and precision that pick and place activities performed in everyday living contexts require 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.

6.2. Case-Based Mobile Push-Manipulation

Push-manipulation is one of the most interesting and challenging robotic manipulation modalities that has attracted many researchers. However, many of the proposed methods handle flat objects with primitive geometric shapes moving quasi-statically on high-friction surfaces, yet they usually make use of complex analytical models or utilize specialized physics engines to predict the outcomes of various interactions. On the other hand, we propose an experience-based approach, which does not require any explicit analytical model or the help of a physics engine. Our mobile robot simply experiments with pushable complex 3D real world objects to observe and memorize their motion characteristics together with the associated motion uncertainties resulting from varying initial caster wheel orientations and potential contacts between the robot and the object. It then uses this incrementally built experience either for trying to make the object of interest follow a guideline path by reiterating in a reactive manner the sequences that result in the best locally-matching outcomes, or as building blocks of a sampling based planner we contribute, the Exp-RRT, to construct push plans that are safe and achievable. In contrast to the proposed approaches in the literature, in our contribution;

- we handle real world objects with complex 3D structures that may contact the robot on more than one point,
- the manipulated objects move on passively-rolling caster wheels and do not stop immediately after the pushing is ceased,
- the experiment environment is cluttered with obstacles; hence, both collision-free and achievable plans should be constructed and manipulation should be performed delicately,
- we do not use any explicit analytical models or learn a mapping between the trajectories of the robot and the object; we only utilize the experimented and observed effects of the past pushing motions to anticipate the future, plan, and act accordingly.

We extensively tested our method in a realistic 3D simulation environment where a variety of pushable objects with passively-rolling caster wheels needed to be navigated among obstacles to reach their desired final poses. We also performed some preliminary tests in a physical setup to verify the validity of our method. Our experiments demonstrate safe transportation and successful placement of several pushable objects in simulation and promising results for push-manipulation tasks in real world, such as arranging chairs in predefined seating formations in a study area. Additionally, we show that the incremental acquisition and tuning of the probabilistic cases allows the robot to adapt to the changes in the dynamics of the objects when continually used after the initial case acquisition phase. Based on our experimental results, our method proves to be robot, object, and environment (real or simulated) independent due to its purely interaction and observation driven nature.

6.3. Contributions

This thesis makes the following major contributions to the field of mobile robotic manipulation:

- Formalization of mobile manipulation as a case-based planning problem, where the robot *retains* manipulation experiences as cases, *reuses* them for plan construction (non-prehensile) and guidance (prehensile), and *revises* them to match the slight differences observed in the manipulated object's behavior (loaded objects) to improve planning and execution performance.
- A mobile pick and place manipulation algorithm that defines target-specific fine reaching trajectories as cases and reuses them to guide the robot through manipulation plan construction and execution processes.
- A mobile whole-body push-manipulation algorithm that incrementally acquires object-specific interaction models and stores them as distinct *probabilistic* cases to be reused as building blocks for achievable push planning and execution.
- A state lattice-based push-manipulation algorithm that is used for treating movable obstacles as "permeable" to reduce the complexity of planning during push-plan construction for the primary object of interest.

- An online algorithm that adapts the probabilistic cases to the changes in the object's behavior based on the observed deviation from the expected outcome.
- Extensive experimental evaluation of the presented case-based prehensile and non-prehensile mobile manipulation methods in simulated and real setups.

6.4. Future Research Directions

There are several promising future research directions that this thesis could lead to, some of which are highlighted below:

- Evaluating the presented pick-and-place and push-manipulation methods through extensive testing and detailed experimentation in a physical setup,
- Active learning of the push-manipulation cases as needed through proactive asking by the robot for additional demonstrations in an informed manner,
- Incorporating dynamic time warping based alignment and adjustment of the successively observed object trajectories during push-manipulation learning to track variance not only over the final object pose but also along the trajectories,
- Repairing only the problematic parts of the generated push plans and reusing them whenever possible instead of complete re-planning,
- Investigating the potential benefits of propagating uncertainty through the plan graph over our current optimistic approach of keeping the distributions of consecutive push results independent from each other,
- Exploring the construction of deterministically optimal paths for the primary object of interest through the utilization of a state lattice planner and comparing the performance against the Exp-RRT,
- Transferring acquired prehensile and non-prehensile manipulation cases among objects with similar properties.

APPENDIX A: BUILDING REALISTIC SIMULATION ENVIRONMENTS

In this section, we elaborate on how we build the simulated models of the office and hospital objects used in our experiments as well as our CoBot robot (Figure 1.1) in Webots [31], and how we control the simulated CoBot to make it move as desired in its task environment. [8]

A.1. Simulated Robot and Object Models

Being an autonomous mobile service robot, CoBot is expected to manipulate a variety of office and hospital service objects, such as office chairs, utility carts, overbed tables, and stretchers. Figure A.1 illustrates some of the objects that we used in our mobile manipulation experiments and the realistic simulation models we prepared for each of them. As it can be seen in the figure, the bodies of our simulated objects are exactly the same as the physical counterparts. We achieve this by simply importing the SolidWorks CAD models of those objects into Webots as meshes. We then define bounding surfaces, ideally with the same forms as the objects themselves, that the simulator can use to compute inter-body collisions. Finally, we augment those simulated object models with caster wheels so that our robot can push-manipulate them in the task environment. Webots provides a standard caster wheel model, as shown in Figure A.2. For each of the objects of interest, we provide reasonable Coulomb friction coefficients for the wheel axis and the fork axis to obtain a decent caster wheel behavior. In our experiments, a friction coefficient of 0.1 for the wheel axis and 1.0 for the fork axis seemed to work fine. Providing reasonable mass and friction parameters to Webots' physics simulator, Open Dynamics Engine (ODE), results in reasonably realistic object behaviors, although it must be noted that our contributed methods are only concerned about the observed poses of the objects in the task environment.

[8] The Webots world files and the controller software we developed during this thesis work can be obtained from `http://tekin.mericli.com/share/thesis_webots.zip`

a) b)

c) d)

e) f)

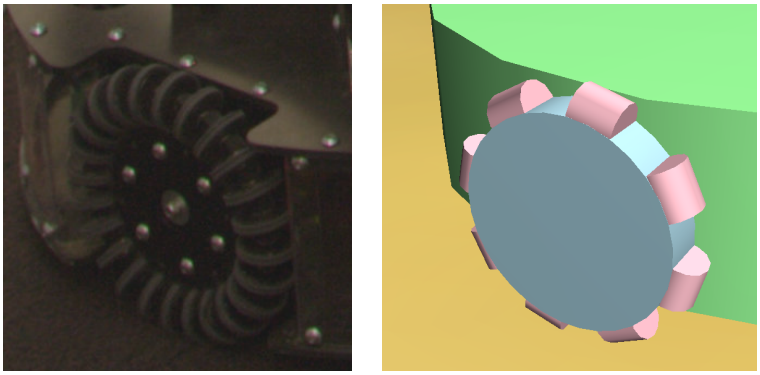
Figure A.1. Realistically simulated models of some common office and service objects, such as a chair, an overbed table, and a utility cart.

a) b)

Figure A.2. A standard generic caster wheel and its simulated counterpart.

A.2. Robot Kinematics and Control

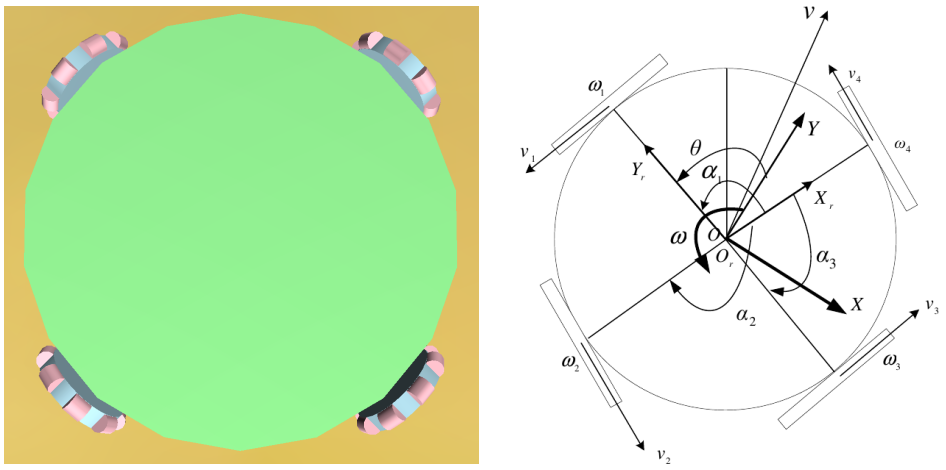
It is important to have a good grasp of the engineering principles behind the robot's design to be able to write kinematics equations and define various controllers to make it move as intended. Our CoBot robot has a holonomic base; that is, the controllable degrees of freedom are greater than or equal to the total physical degrees of freedom. This holonomic property of the robot is achieved by the use of four omni-directional wheels (Figure A.3) placed around the round body at every 90 degrees, as shown in Figure A.4b. As long as the robot stays within the safe velocity limits, this configuration allows it to move in any arbitrary combination of $\langle v_x, v_y, \omega \rangle$, which represents the desired translational (forward-backward and sideways) and rotational (around the center of rotation) velocity components of the robot's body. We approximated the many small rollers on the omni-directional wheels of the physical CoBot with larger rollers sparsely placed on the simulated wheel and obtained similar motion characteristics to the physical platform while potentially being less computationally demanding on the simulator side. The simulated CoBot achieves its motion through the interaction of these small rollers with the floor surface of the task environment, just like how the physical robot achieves its motion.



a) b)

Figure A.3. Simulated model of the omni-directional wheel with a lower resolution of rollers compared to the wheels of the physical platform.

The robot can be moved around by continuously providing it with non-zero omni-directional motion commands in $\langle v_x, v_y, \omega \rangle$ form. The source of these motion commands could be the ROS-based joystick interface implemented for tele-operating CoBot and



a) Simulated CoBot base. b) Robot kinematics. [64]

Figure A.4. Simulated four-wheeled omni-directional CoBot base and its kinematics.

extended for the purposes of this thesis, or the path tracker and motion controller modules during autonomous operation. Through a set of kinematics equations, these commands in the body coordinate frame can be translated into angular velocities for the individual wheels. In addition to the requested motion command, these equations make use of the information about the total number of wheels and their placements around the body, the radii of the wheels, and the distances of the wheels from the center of the robot's body.

In Figure A.4b, the coordinate frame $X_rO_rY_r$ is attached to the robot and XOY is the world frame. The radii of the wheels are assumed to be equal and denoted by r, and the distance between a wheel's center and the center of the robot is denoted by b, assuming that all wheels have equal b values. $\alpha_i, i \in \{1, ..., 4\}$ represent the angles between the axles of the wheels. The angular velocity of each wheel is denoted as $\omega_i, i \in \{1, ..., 4\}$ and the direction of the linear velocity of the center of each wheel with respect to $X_rO_rY_r$ is denoted as $v_i, i \in \{1, ..., 4\}$. The angle between the reference frames $X_rO_rY_r$ and XOY is θ. The linear and angular velocities of the robot are $v = [v_x v_y]^T$ and ω, respectively. Assuming that the axis X_r aligns with the axle of the fourth wheel, we can write the following kinematics equations.

$$r\omega_i = b\omega + v_r^T v_i, i \in \{1, ..., 4\} \tag{A.1}$$

$$v_i = [-sin(\alpha_i)\ cos(\alpha_i)]^T, i \in \{1, ..., 4\} \tag{A.2}$$

$$v_{rx} = v_x cos(\theta) + v_y sin(\theta) \tag{A.3}$$

$$v_{ry} = -v_x sin(\theta) + v_y cos(\theta) \tag{A.4}$$

$$v_r = [v_{rx}\ v_{ry}]^T \tag{A.5}$$

Hence the angular velocities for each individual wheel can be computed as follows.

$$\omega_i = r^{-1}(b\omega + v_{ry} cos(\alpha_i) - v_{rx} sin(\alpha_i)), i \in \{1, ..., 4\} \tag{A.6}$$

APPENDIX B: REAL WORLD TEST SETUP

The aim for our preliminary tests in the physical world was to see if we could utilize our proposed case-based push-manipulation algorithm to get our CoBot robot (Figure 1.1) to organize a set of office chairs around a round table in one of the study areas of the 9^{th} floor of the Gates Hillman Center at Carnegie Mellon University, as seen in Figure 3.15. There are a number of challenges that need to be addressed when switching from the simulated environment to the physical one. In simulation, global poses of all of the objects in the environment are directly reported by the simulator. However, in the physical setup, the robot has to detect and infer the poses of the objects and the obstacles through its on-board sensors. In order to make the visual detection of the chair easier, we placed Augmented Reality (AR) tags on both sides of the back of the chair (Figure B.1), which are visible most of the time from almost all directions. The visual detection and tracking of the tags are handled by using the AR Track Alvar package, which is available in the ROS repositories. [9]

Figure B.1. AR Tag tracking visualization. The simulated chair moves in the world model of the robot as the physical chair moves in the real world.

[9] `http://wiki.ros.org/ar_track_alvar`

AR Track Alvar publishes the 6 DoF poses of the detected tags in the reference frame of the camera; however, we would like to know where the geometric center of the object with respect to the base of the robot, none of which are directly provided as a result of the detection process. However, since we place the tag on the object, we can measure its pose relative to the geometric center of the object, and publish that information as a static transform to be a part of the `tf` tree, which is a mechanism in ROS that lets the user keep track of multiple reference frames on the robot and in the environment over time. [10] Similarly the static transform between the optical frame of the camera and the base of the robot is also known and published as a part of the `tf` tree. Therefore, given that we know where the object's center (φ_{OOI}) is with respect to the tag (φ_{Tag}), where the tag is with respect to the camera (φ_{Camera}), and where the camera is with respect to the robot's base (φ_{Robot}), we can compute where the object's center with respect to the robot's base, as seen in Figure B.2.

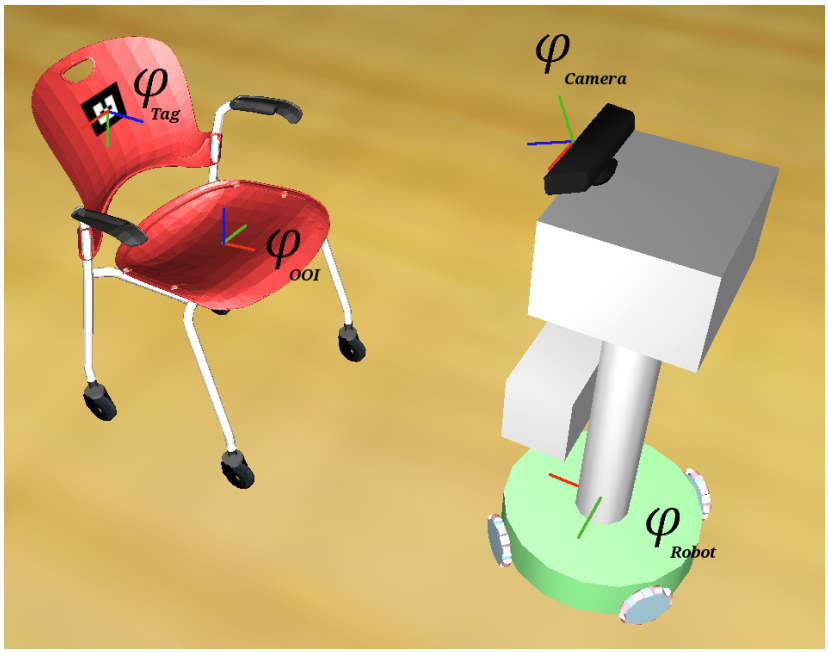


Figure B.2. Various reference frames attached to the AR tag (φ_{Tag}), the object's geometric center (φ_{OOI}), the camera (φ_{Camera}), and the robot's base (φ_{Robot}). These coordinate frames are utilized for computing the location of the object's center with respect to the robot.

[10] `http://wiki.ros.org/tf`

In addition to detecting the objects, the robot also has to localize itself in the environment so that it can navigate safely and have an idea about how to move to deliver the objects to their desired locations requested in global map coordinates. The general assumption that the robot makes is that the large vertical planar surfaces would correspond to the interior walls of the building. The robot uses these features detected through processing the depth images provided by the on-board Kinect sensor, which is also the primary camera, to localize itself in the environment [53]. The visuals depicting this process are shown in Figure B.3. The obtained pose information is noisier compared to the perfect pose information received in simulation.

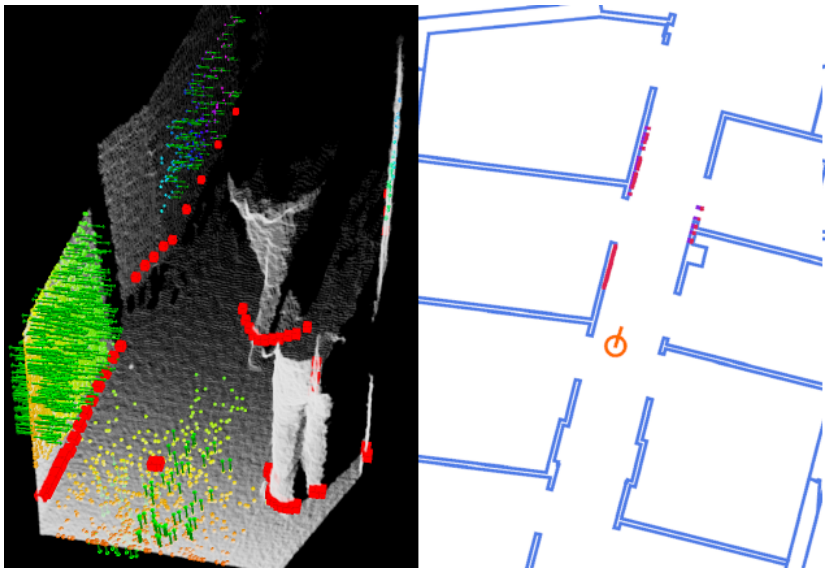


Figure B.3. Screenshot from the operation of the Fast Sampling Plane Filtering based planar feature detection and Corrective Gradient Refinement based localization algorithms [53, 65].

After the objects' and the robot's poses in the global map are computed, they can be plugged into our robot and environment agnostic codes, just like the pose values reported by the simulator, to acquire push-manipulation cases for the object and use them for planning collision-free and achievable push plans for the OOI.

APPENDIX C: ADDITIONAL EXPERIMENTAL RESULTS

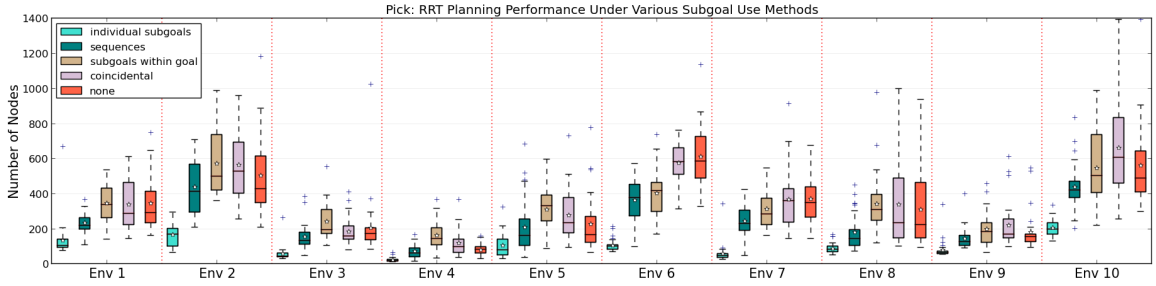
C.1. Experience-based Mobile Pick and Place

In this section, we present additional experimental results obtained with various office and service objects, namely an overbed table, a utility cart, and a stretcher, in the prehensile mobile manipulation domain for each combination of the three generative planners (RRT, RRT-Connect, and RRT*) and the four subgoal utilization methods (sampling individual subgoals, sampling sequences, sampling subgoals within goal probability, and coincidental termination) as well as the base case of planning only for reaching the actual goal.

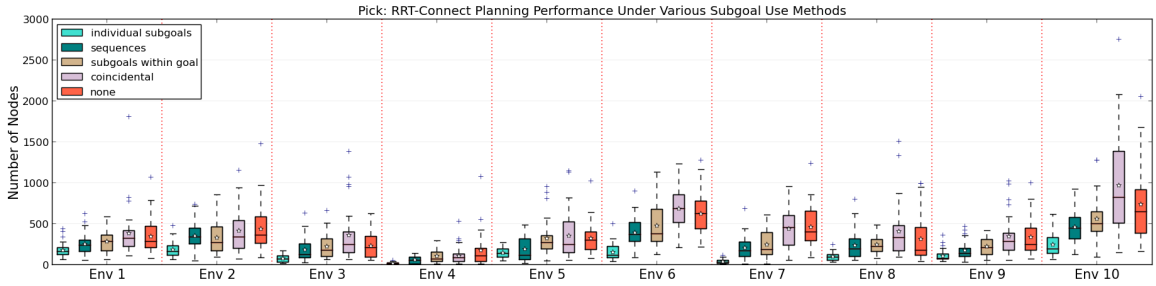
The box plots shown in Figure C.1 and Figure C.3 present the planning performance statistics, measured in number of nodes, obtained with each of these combinations for picking up and placing the overbed table object, respectively. The box plots shown in Figure C.2 and Figure C.4 provide the corresponding time statistics.

Table C.1 and Table C.2 provide sequence utilization statistics of the tests presented in Figure C.1 and Figure C.3, respectively. These numbers indicate in what percentage of the 30 planning trials with each combination the planner eventually ended up reaching an entry point. We see that sampling individual subgoals and sampling subgoals within goal probability have very high sequence utilization percentages. In the case of sampling individual subgoals, the total bias towards the delicate manipulation region increases, hence the increased sequence utilization percentages. Since the method of of sampling subgoals within goal probability combines the actual goal with the set of subgoals, it becomes very likely for a subgoal to be sampled instead of the actual goal whenever a goal is decided to be sampled.

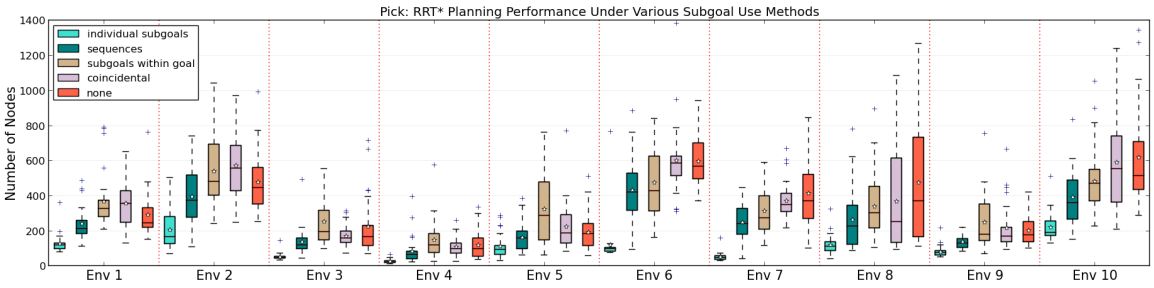
The box plots shown in Figure C.5 and Figure C.7 present the planning performance statistics, measured in number of nodes, obtained with each of these combina-



a)



b)



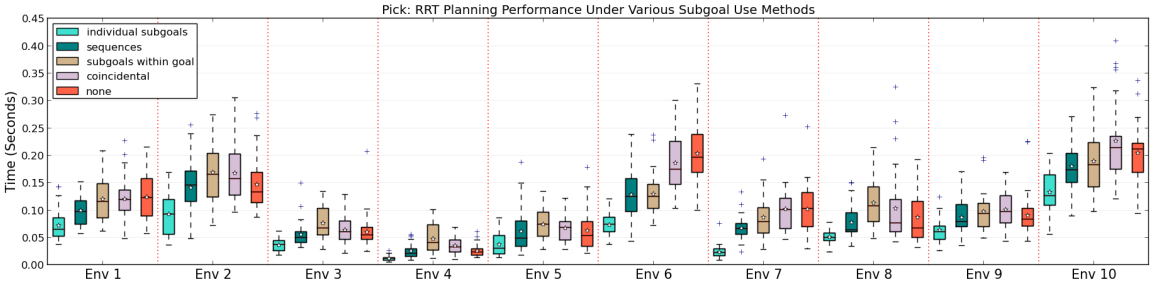
c)

Figure C.1. The statistics for the number of generated nodes by (a) the RRT, (b) the RRT-Connect, and (c) the RRT* generative planners during planning for picking up the overbed table object in 10 different task environments.

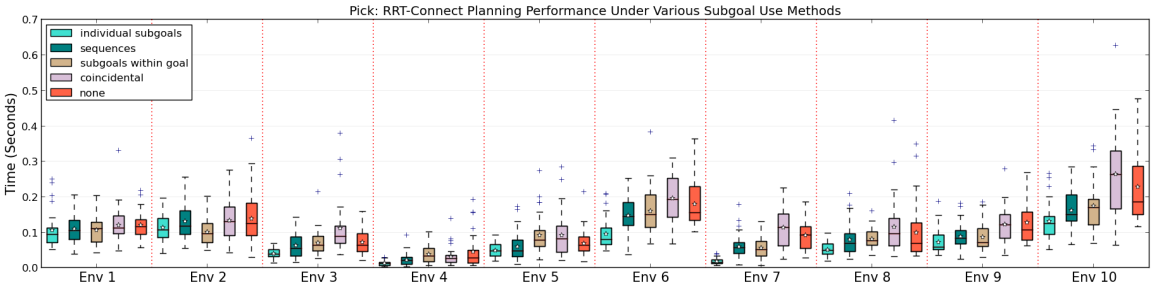
tions for picking up and placing the utility cart object, respectively. The box plots shown in Figure C.6 and Figure C.8 provide the corresponding time statistics.

Table C.3 and Table C.4 provide sequence utilization statistics of the tests presented in Figure C.5 and Figure C.7, respectively.

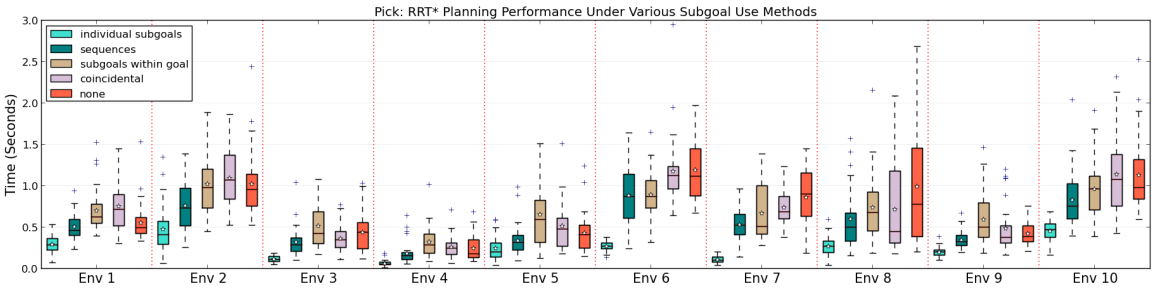
The box plots shown in Figure C.9 and Figure C.11 present the planning performance statistics, measured in number of nodes, obtained with each of these combinations for picking up and placing the stretcher object, respectively. The box plots



a)



b)

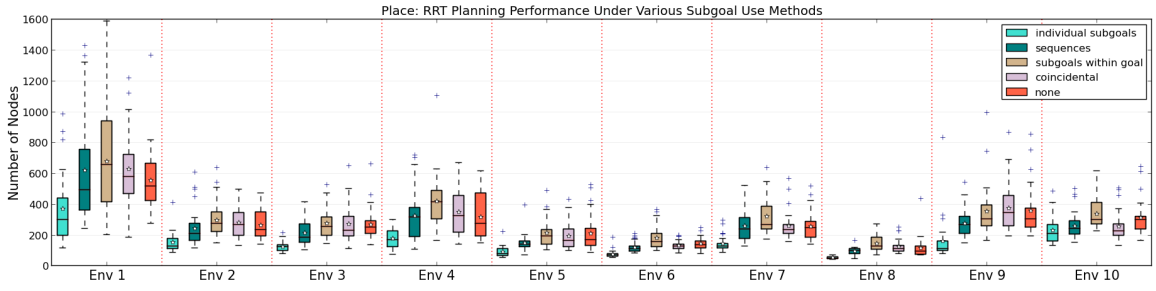


c)

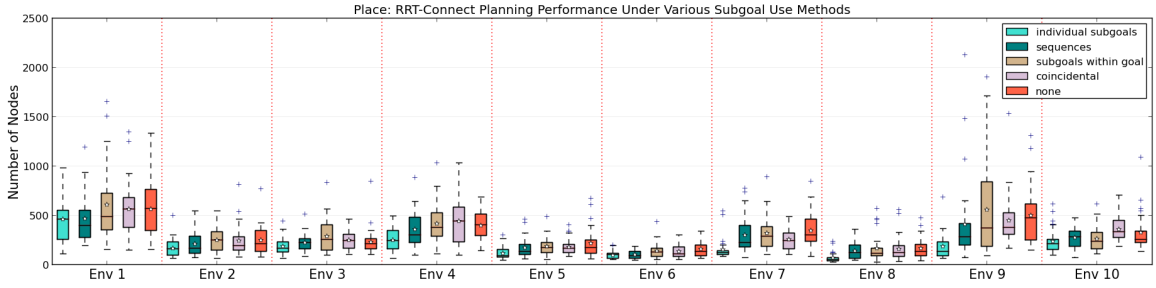
Figure C.2. Time statistics for (a) the RRT, (b) the RRT-Connect, and (c) the RRT* planners to generate plans for picking up the overbed table object in 10 different task environments.

shown in Figure C.10 and Figure C.12 provide the corresponding time statistics.

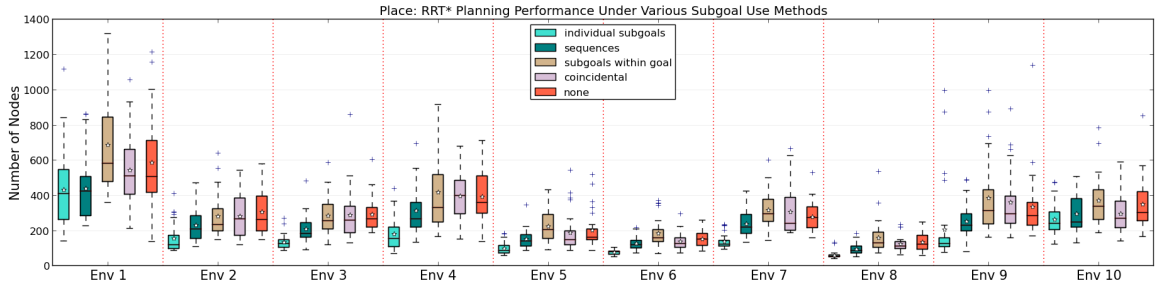
Table C.5 and Table C.6 provide sequence utilization statistics of the tests presented in Figure C.9 and Figure C.11, respectively.



a)

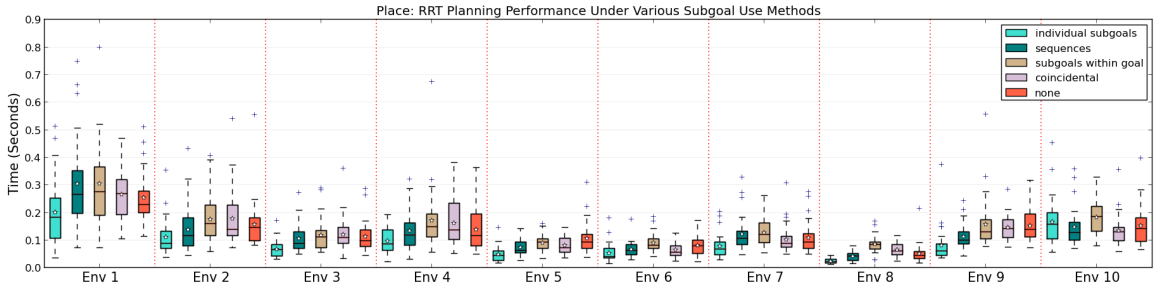


b)

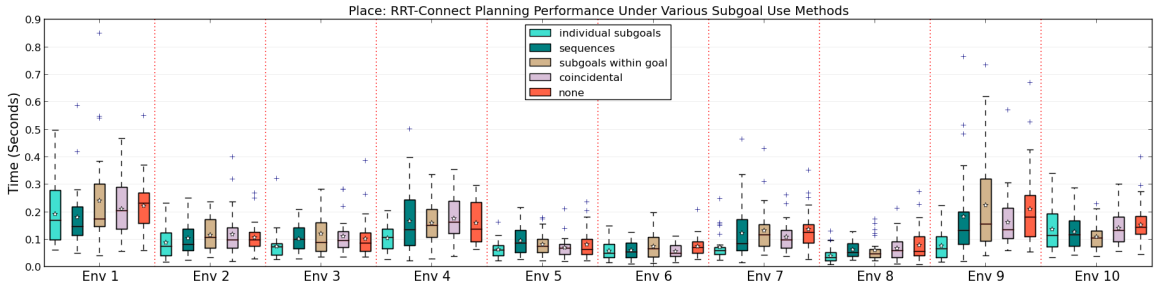


c)

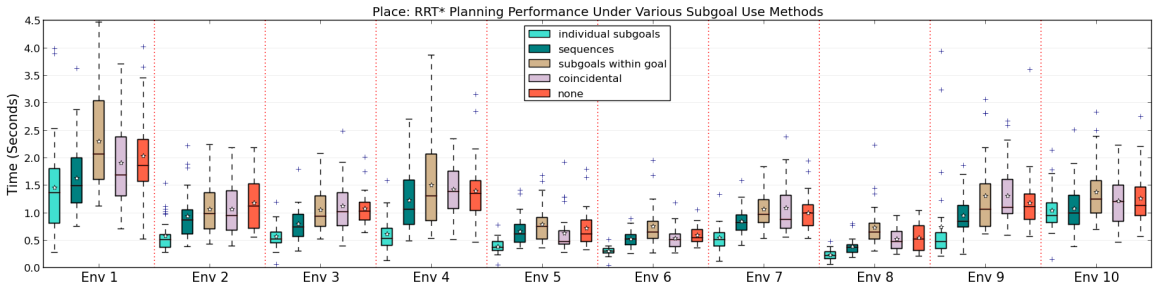
Figure C.3. The statistics for the number of generated nodes by (a) the RRT, (b) the RRT-Connect, and (c) the RRT* generative planners during planning for placing the overbed table object in 10 different task environments.



a)



b)



c)

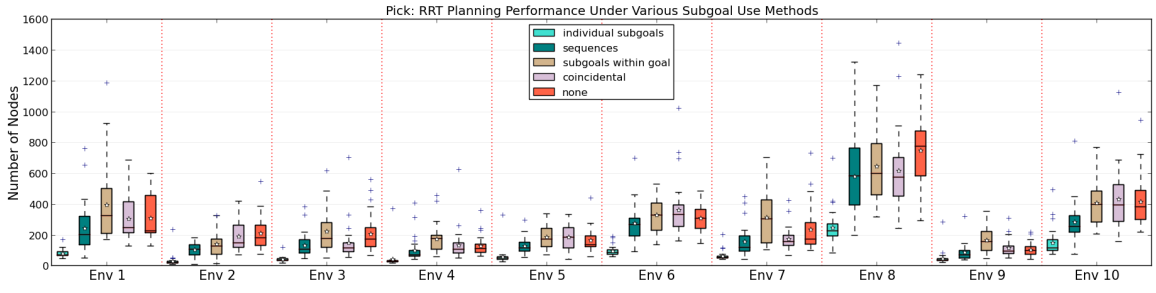
Figure C.4. Time statistics for (a) the RRT, (b) the RRT-Connect, and (c) the RRT* planners to generate plans for placing the overbed table object in 10 different task environments.

Table C.1. Sequence utilization while planning and executing pick task for the overbed table object in ten different environments.

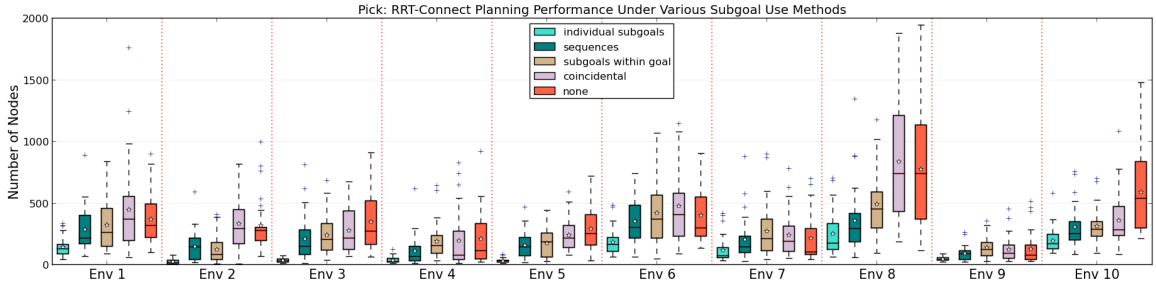
Planner	Subgoal Use Method	Sequence Utilization (%)									
		Pick									
		Env 1	Env 2	Env 3	Env 4	Env 5	Env 6	Env 7	Env 8	Env 9	Env 10
RRT	Coincidental	6.67	3.33	0.0	3.33	3.33	13.33	13.33	30.0	16.67	3.33
	Within goals	93.33	73.33	86.67	93.33	73.33	93.33	86.67	93.33	86.67	86.67
	Sequences	26.67	43.33	13.33	33.33	20.0	76.67	50.0	46.67	43.33	30.0
	Individual subgoals	76.67	83.33	56.67	76.67	50.0	90.0	93.33	83.33	93.33	90.0
RRT-Connect	Coincidental	23.33	20.0	6.67	10.0	13.33	3.33	3.33	33.33	16.67	10.0
	Withing goals	93.33	83.33	90.0	90.0	90.0	86.67	93.33	96.67	93.33	93.33
	Sequences	40.0	60.0	73.33	60.0	63.33	66.67	66.67	63.33	83.33	73.33
	Individual subgoals	83.33	80.0	86.67	80.0	96.67	90.0	96.67	86.67	86.67	83.33
RRT*	Coincidental	13.33	3.33	6.67	3.33	0.0	10.0	13.33	30.0	23.33	6.67
	Within goals	90.0	93.33	73.33	80.0	80.0	86.67	96.67	96.67	76.67	90.0
	Sequences	30.0	30.0	33.33	26.67	13.33	76.67	56.67	53.33	40.0	53.33
	Individual subgoals	86.67	90.0	56.67	80.0	46.67	93.33	96.67	90.0	80.0	83.33

Table C.2. Sequence utilization while planning and executing place task for the overbed table object in ten different environments.

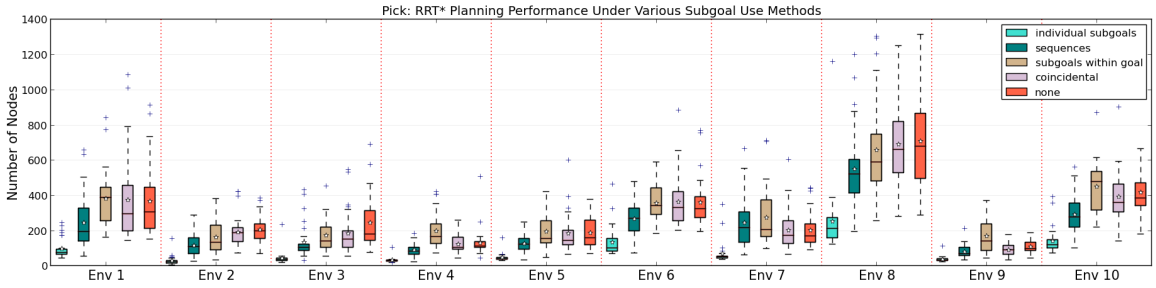
Planner	Subgoal Use Method	Sequence Utilization (%)									
		Place									
		Env 1	Env 2	Env 3	Env 4	Env 5	Env 6	Env 7	Env 8	Env 9	Env 10
RRT	Coincidental	26.67	6.67	30.0	20.0	30.0	20.0	3.33	13.33	23.33	16.67
	Within goals	93.33	100.0	93.33	96.67	90.0	83.33	83.33	93.33	90.0	93.33
	Sequences	56.67	23.33	70.0	33.33	56.67	26.67	30.0	36.67	26.67	36.67
	Individual subgoals	80.0	76.67	86.67	83.33	93.33	76.67	56.67	76.67	70.0	73.33
RRT-Connect	Coincidental	33.33	16.67	23.33	16.67	26.67	16.67	13.33	23.33	33.33	16.67
	Withing goals	93.33	100.0	93.33	100.0	93.33	90.0	90.0	96.67	90.0	96.67
	Sequences	70.0	66.67	70.0	43.33	53.33	50.0	53.33	56.67	80.0	66.67
	Individual subgoals	90.0	86.67	90.0	86.67	80.0	76.67	93.33	76.67	90.0	86.67
RRT*	Coincidental	26.67	23.33	36.67	13.33	43.33	13.33	3.33	3.33	10.0	23.33
	Within goals	96.67	90.0	93.33	93.33	96.67	90.0	96.67	90.0	80.0	93.33
	Sequences	46.67	26.67	46.67	13.33	56.67	40.0	10.0	33.33	53.33	30.0
	Individual subgoals	90.0	70.0	93.33	83.33	96.67	83.33	70.0	86.67	66.67	70.0



a)

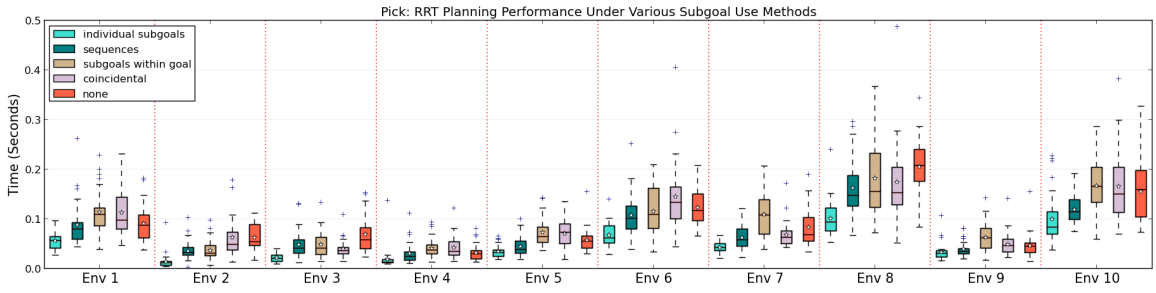


b)

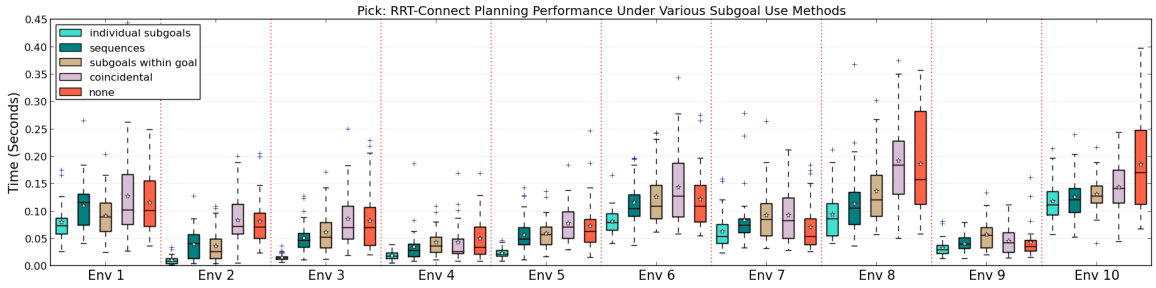


c)

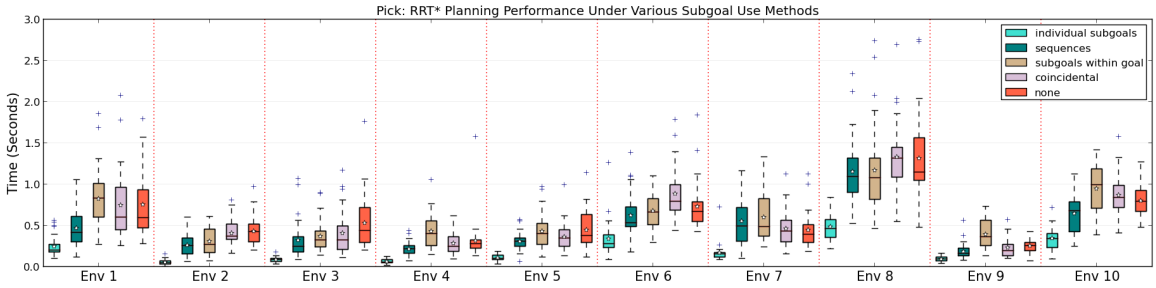
Figure C.5. The statistics for the number of generated nodes by (a) the RRT, (b) the RRT-Connect, and (c) the RRT* generative planners during planning for picking up the utility cart object in 10 different task environments.



a)

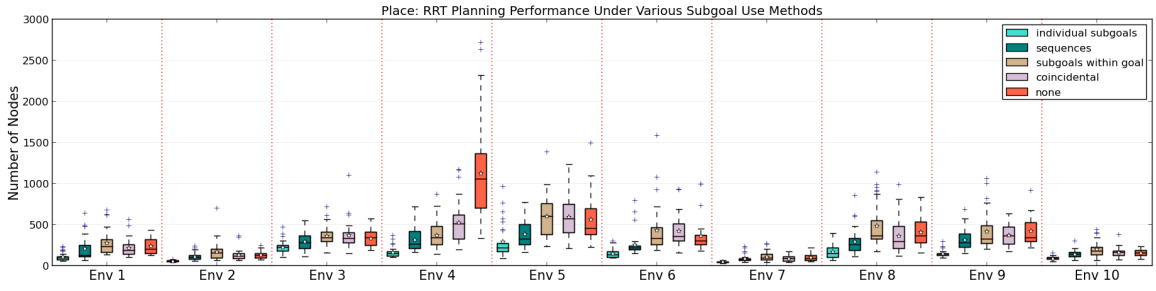


b)

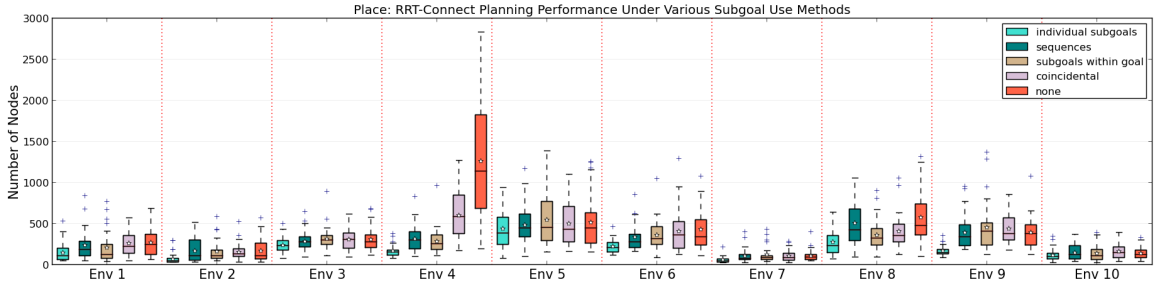


c)

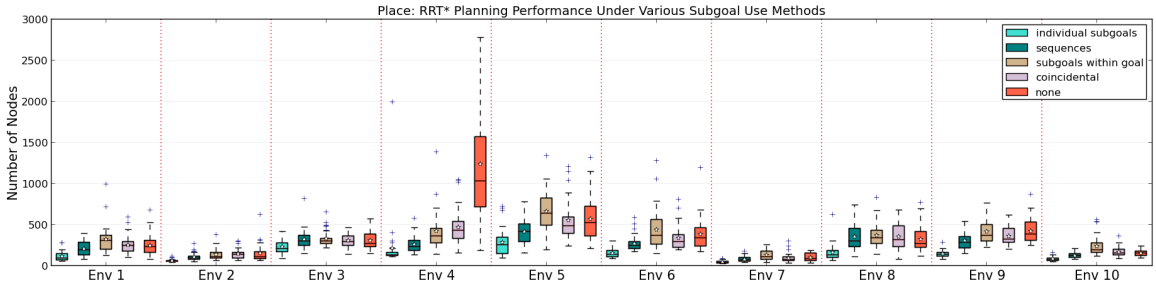
Figure C.6. Time statistics for (a) the RRT, (b) the RRT-Connect, and (c) the RRT* planners to generate plans for picking up the utility cart object in 10 different task environments.



a)

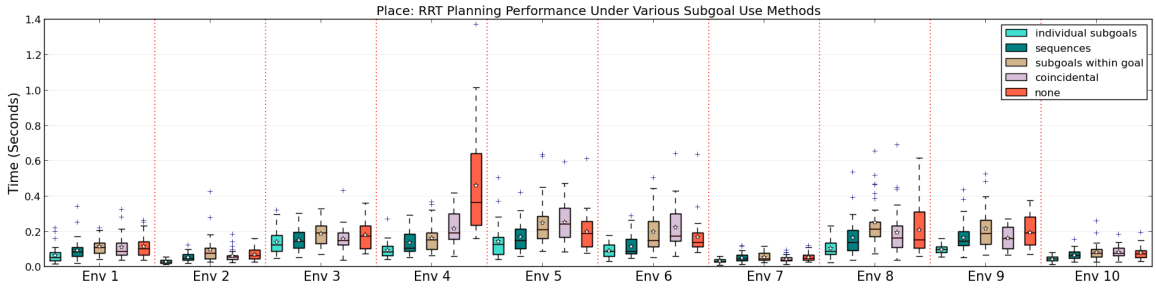


b)

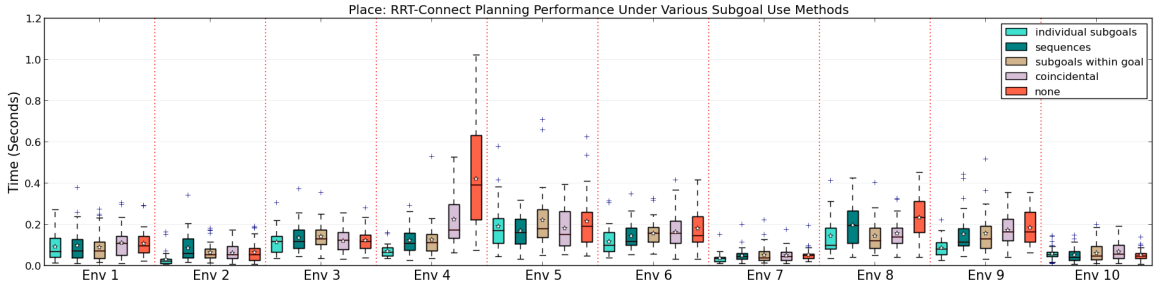


c)

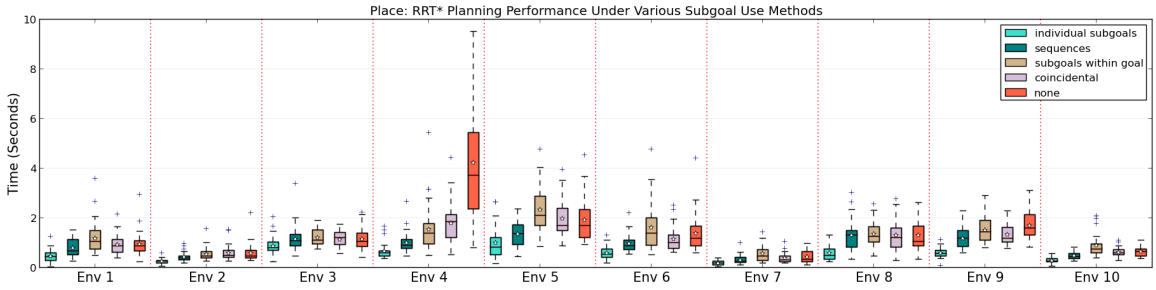
Figure C.7. The statistics for the number of generated nodes by (a) the RRT, (b) the RRT-Connect, and (c) the RRT* generative planners during planning for placing the utility cart object in 10 different task environments.



a)



b)



c)

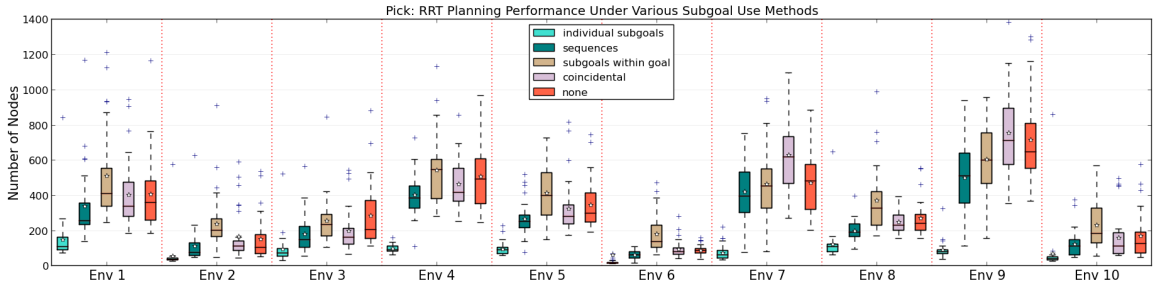
Figure C.8. Time statistics for (a) the RRT, (b) the RRT-Connect, and (c) the RRT* planners to generate plans for placing the utility cart object in 10 different task environments.

Table C.3. Sequence utilization while planning and executing pick task for the utility cart object in ten different environments.

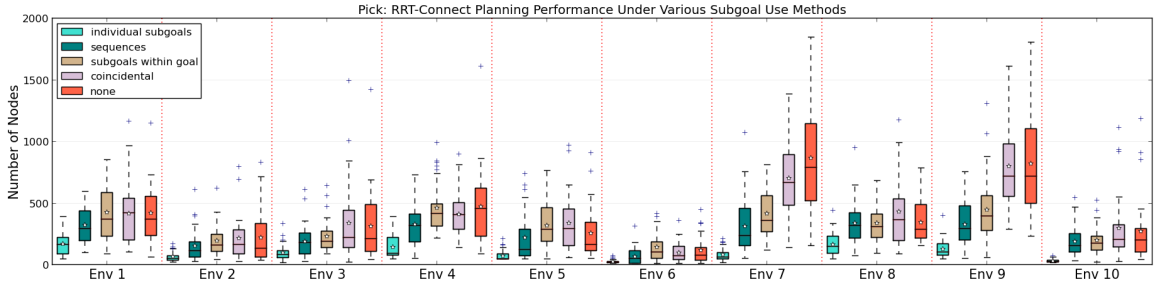
Planner	Subgoal Use Method	Sequence Utilization (%)									
		Pick									
		Env 1	Env 2	Env 3	Env 4	Env 5	Env 6	Env 7	Env 8	Env 9	Env 10
RRT	Coincidental	30.0	16.67	13.33	40.0	6.67	16.67	20.0	10.0	3.33	6.67
	Within goals	83.33	93.33	76.67	76.67	83.33	86.67	73.33	76.67	83.33	73.33
	Sequences	36.67	40.0	23.33	23.33	43.33	46.67	23.33	60.0	30.0	53.33
	Individual subgoals	73.33	96.67	80.0	66.67	93.33	90.0	53.33	90.0	46.67	90.0
RRT-Connect	Coincidental	26.67	13.33	13.33	30.0	16.67	30.0	40.0	6.67	16.67	16.67
	Withing goals	90.0	100.0	100.0	86.67	96.67	96.67	80.0	86.67	90.0	96.67
	Sequences	50.0	86.67	60.0	40.0	63.33	76.67	50.0	66.67	43.33	86.67
	Individual subgoals	90.0	96.67	96.67	70.0	96.67	90.0	70.0	96.67	76.67	93.33
RRT*	Coincidental	33.33	23.33	10.0	26.67	10.0	3.33	26.67	6.67	26.67	16.67
	Within goals	83.33	93.33	83.33	76.67	83.33	76.67	73.33	83.33	86.67	73.33
	Sequences	36.67	63.33	36.67	16.67	33.33	46.67	40.0	53.33	30.0	36.67
	Individual subgoals	86.67	96.67	83.33	60.0	96.67	86.67	50.0	96.67	50.0	86.67

Table C.4. Sequence utilization while planning and executing place task for the utility cart object in ten different environments.

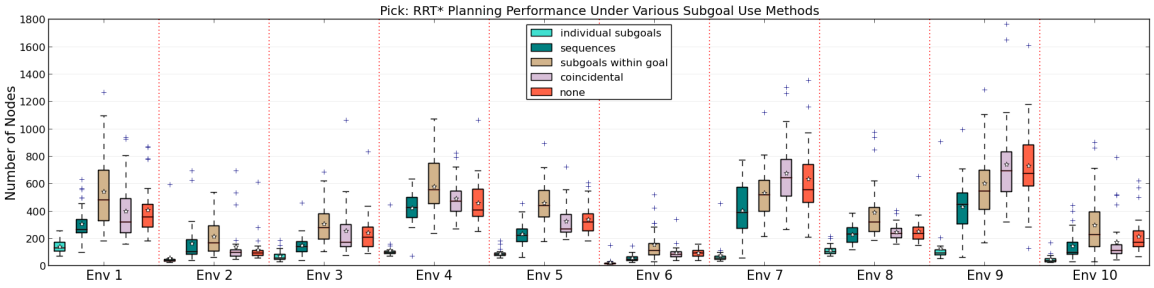
Planner	Subgoal Use Method	Sequence Utilization (%)									
		Place									
		Env 1	Env 2	Env 3	Env 4	Env 5	Env 6	Env 7	Env 8	Env 9	Env 10
RRT	Coincidental	16.67	13.33	16.67	70.0	30.0	16.67	6.67	16.67	50.0	6.67
	Within goals	80.0	90.0	90.0	93.33	83.33	80.0	73.33	80.0	93.33	76.67
	Sequences	33.33	53.33	50.0	80.0	43.33	33.33	36.67	20.0	60.0	23.33
	Individual subgoals	73.33	63.33	56.67	83.33	63.33	73.33	70.0	56.67	73.33	60.0
RRT-Connect	Coincidental	23.33	6.67	36.67	80.0	36.67	20.0	10.0	16.67	26.67	6.67
	Withing goals	100.0	80.0	83.33	90.0	90.0	90.0	83.33	90.0	90.0	93.33
	Sequences	50.0	30.0	46.67	96.67	56.67	66.67	46.67	56.67	83.33	46.67
	Individual subgoals	80.0	63.33	73.33	83.33	73.33	86.67	76.67	90.0	83.33	80.0
RRT*	Coincidental	10.0	10.0	33.33	73.33	6.67	13.33	3.33	3.33	50.0	16.67
	Within goals	76.67	80.0	80.0	90.0	86.67	86.67	80.0	76.67	76.67	83.33
	Sequences	16.67	36.67	36.67	63.33	46.67	33.33	33.33	33.33	73.33	30.0
	Individual subgoals	53.33	60.0	66.67	83.33	56.67	76.67	60.0	66.67	80.0	60.0



a)

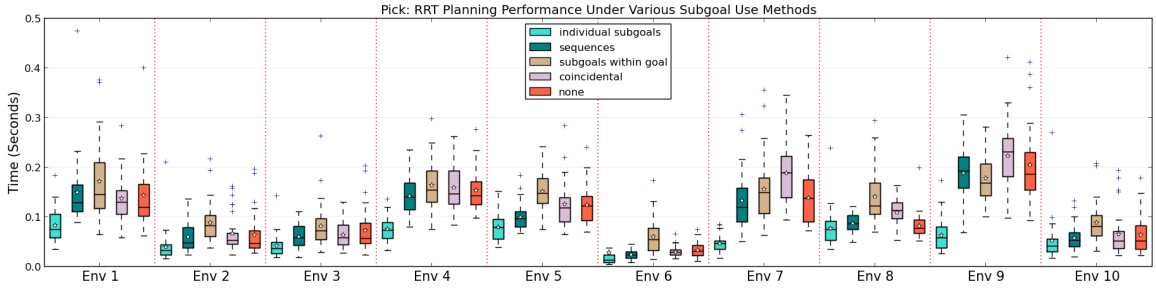


b)

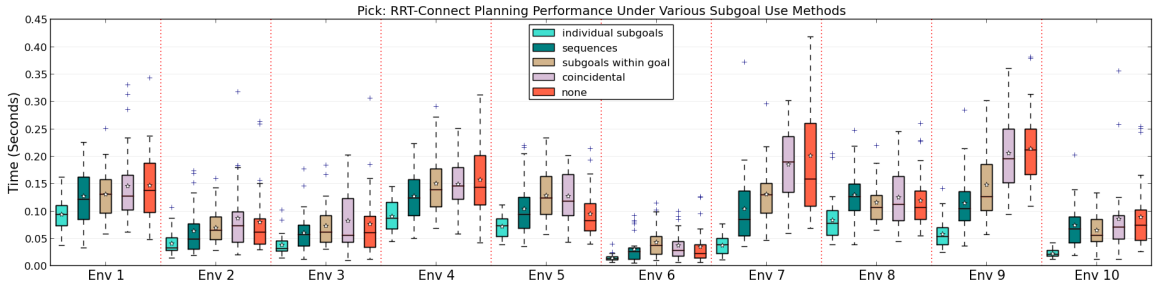


c)

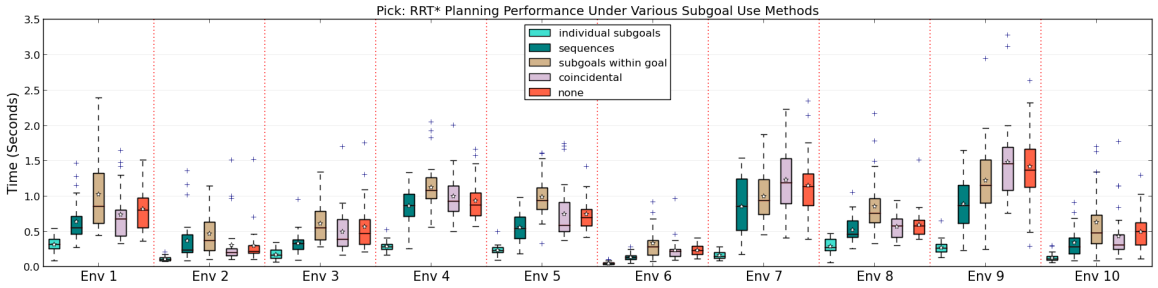
Figure C.9. The statistics for the number of generated nodes by (a) the RRT, (b) the RRT-Connect, and (c) the RRT* generative planners during planning for picking up the stretcher object in 10 different task environments.



a)

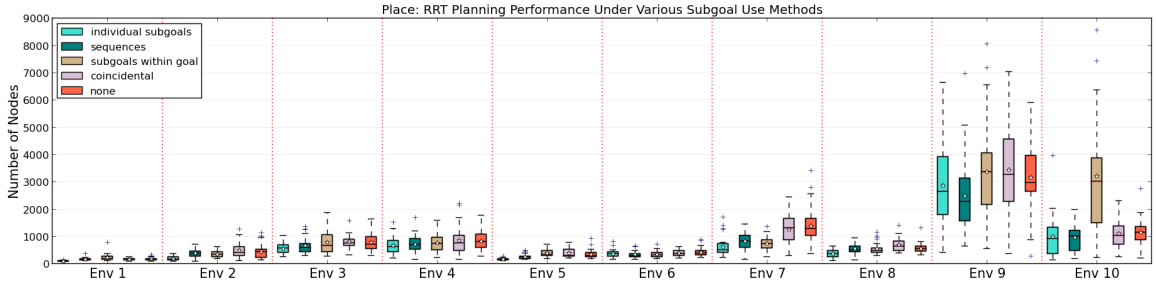


b)

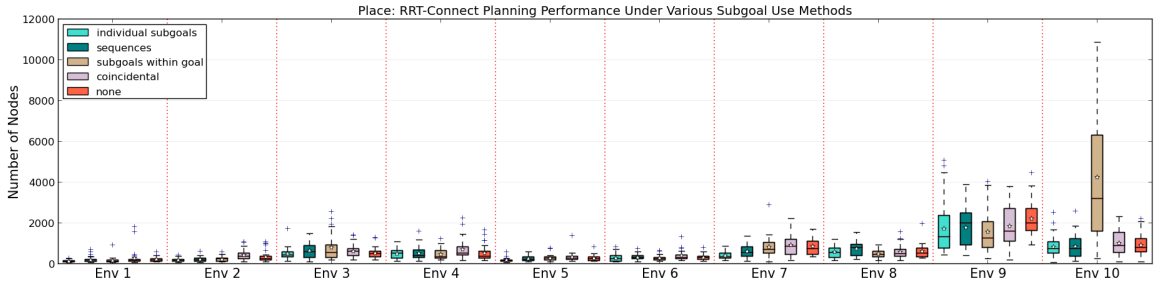


c)

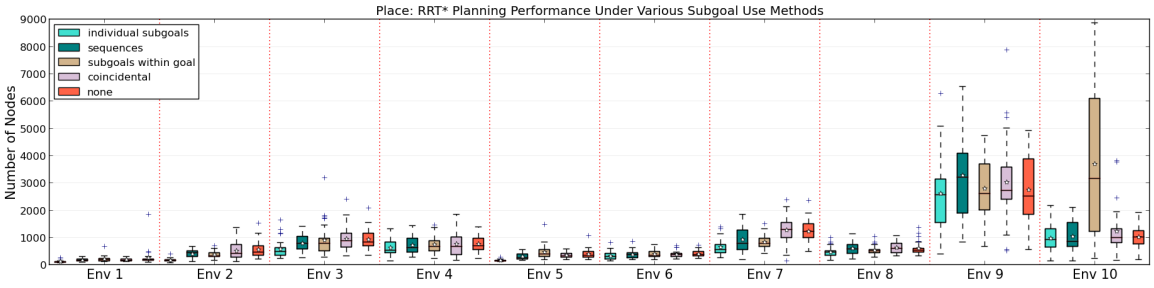
Figure C.10. Time statistics for (a) the RRT, (b) the RRT-Connect, and (c) the RRT* planners to generate plans for picking up the stretcher object in 10 different task environments.



a)

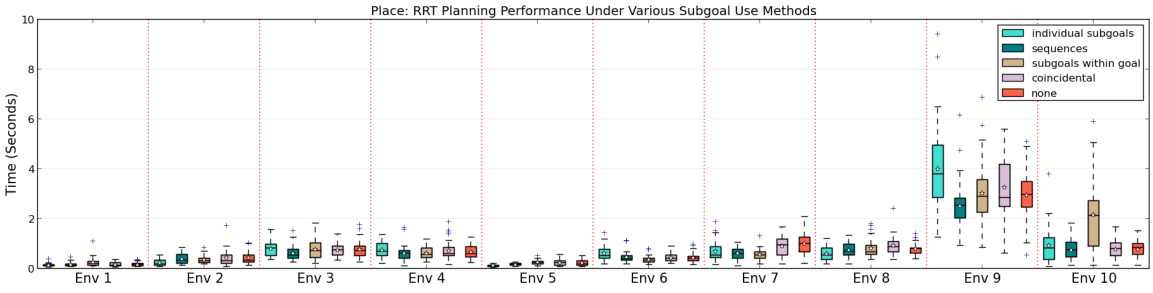


b)

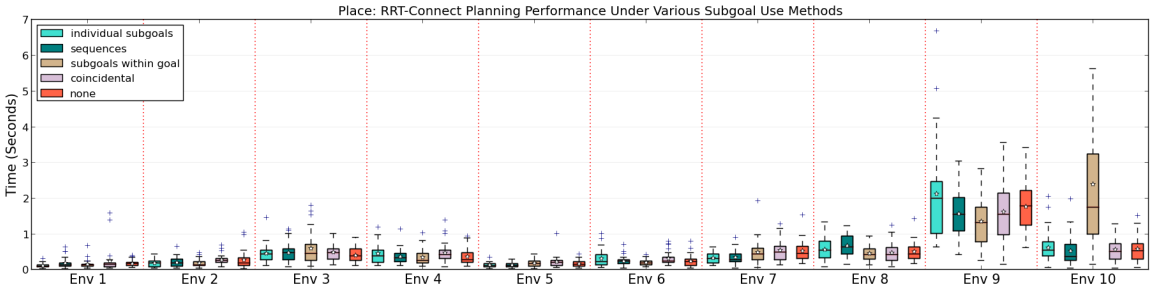


c)

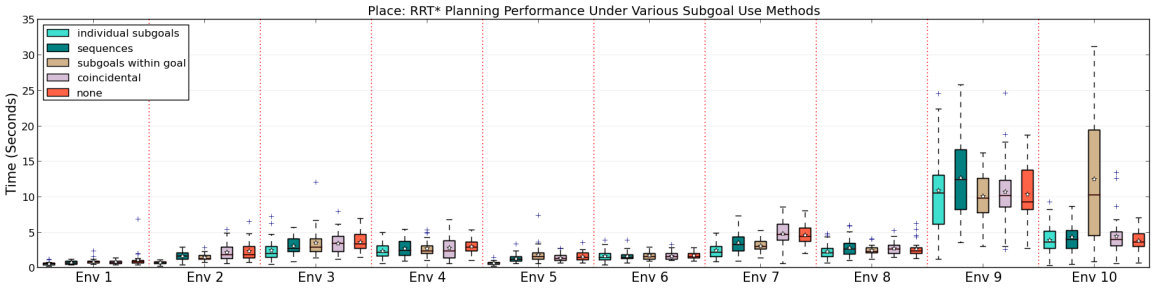
Figure C.11. The statistics for the number of generated nodes by (a) the RRT, (b) the RRT-Connect, and (c) the RRT* generative planners during planning for placing the stretcher object in 10 different task environments.



a)



b)



c)

Figure C.12. Time statistics for (a) the RRT, (b) the RRT-Connect, and (c) the RRT* planners to generate plans for placing the stretcher object in 10 different task environments.

Table C.5. Sequence utilization while planning and executing pick task for the stretcher object in ten different environments.

Planner	Subgoal Use Method	Sequence Utilization (%)									
		Pick									
		Env 1	Env 2	Env 3	Env 4	Env 5	Env 6	Env 7	Env 8	Env 9	Env 10
RRT	Coincidental	43.33	13.33	33.33	6.67	6.67	20.0	0.0	16.67	6.67	26.67
	Within goals	60.0	56.67	70.0	66.67	50.0	46.67	80.0	50.0	76.67	66.67
	Sequences	43.33	40.0	50.0	26.67	26.67	26.67	70.0	23.33	66.67	20.0
	Individual subgoals	73.33	46.67	80.0	93.33	56.67	43.33	93.33	50.0	96.67	73.33
RRT-Connect	Coincidental	36.67	23.33	23.33	10.0	10.0	10.0	3.33	26.67	16.67	26.67
	Withing goals	73.33	43.33	76.67	76.67	73.33	70.0	96.67	80.0	93.33	83.33
	Sequences	66.67	60.0	60.0	63.33	53.33	40.0	100.0	73.33	90.0	46.67
	Individual subgoals	86.67	66.67	86.67	90.0	46.67	50.0	96.67	80.0	96.67	93.33
RRT*	Coincidental	46.67	6.67	16.67	3.33	13.33	16.67	23.33	20.0	13.33	30.0
	Within goals	40.0	56.67	63.33	76.67	40.0	66.67	90.0	63.33	93.33	73.33
	Sequences	36.67	30.0	43.33	33.33	20.0	36.67	66.67	30.0	76.67	26.67
	Individual subgoals	76.67	66.67	86.67	86.67	40.0	50.0	90.0	56.67	96.67	76.67

Table C.6. Sequence utilization while planning and executing place task for the stretcher object in ten different environments.

Planner	Subgoal Use Method	Sequence Utilization (%)									
		Place									
		Env 1	Env 2	Env 3	Env 4	Env 5	Env 6	Env 7	Env 8	Env 9	Env 10
RRT	Coincidental	33.33	20.0	46.67	10.0	40.0	86.67	56.67	60.0	30.0	33.33
	Within goals	90.0	100.0	96.67	70.0	93.33	100.0	86.67	93.33	100.0	96.67
	Sequences	60.0	93.33	60.0	26.67	73.33	83.33	50.0	46.67	60.0	43.33
	Individual subgoals	73.33	90.0	86.67	76.67	93.33	93.33	80.0	93.33	90.0	93.33
RRT-Connect	Coincidental	26.67	26.67	23.33	33.33	43.33	56.67	26.67	46.67	23.33	36.67
	Withing goals	86.67	93.33	93.33	96.67	93.33	100.0	96.67	86.67	100.0	96.67
	Sequences	60.0	73.33	50.0	60.0	66.67	86.67	76.67	70.0	73.33	43.33
	Individual subgoals	96.67	96.67	90.0	93.33	93.33	96.67	90.0	80.0	96.67	53.33
RRT*	Coincidental	26.67	30.0	33.33	13.33	33.33	86.67	30.0	50.0	33.33	20.0
	Within goals	83.33	93.33	93.33	90.0	86.67	100.0	100.0	93.33	100.0	93.33
	Sequences	40.0	66.67	70.0	26.67	60.0	80.0	60.0	76.67	56.67	46.67
	Individual subgoals	86.67	90.0	86.67	76.67	86.67	96.67	93.33	90.0	96.67	96.67

REFERENCES

1. Mason, C. R., J. E. Gomez and T. J. Ebner, "Hand Synergies During Reach-to-Grasp", *Journal of Neurophysiology*, Vol. 86, No. 6, pp. 2896–2910, 2001.

2. Rosenthal, S., J. Biswas and M. Veloso, "An Effective Personal Mobile Robot Agent Through Symbiotic Human-Robot Interaction", *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2010.

3. Aamodt, A. and E. Plaza, "Case-Based Reasoning; Foundational Issues, Methodological Variations, and System Approaches", *AI Communications*, Vol. 7, No. 1, pp. 39–59, 1994.

4. Veloso, M. M., H. Munoz-Avila and R. Bergmann, "General-Purpose Case-Based Planning: Methods and Systems", *AI Communications*, Vol. 9, No. 3, pp. 128–137, 1996.

5. Ros, R., J. L. Arcos, R. L. de Mantaras and M. Veloso, "A Case-based Approach for Coordinated Action Selection in Robot Soccer", *Artificial Intelligence*, Vol. 173, pp. 1014–1039, 2009.

6. Meriçli, T., M. Veloso and H. L. Akın, "Experience Guided Mobile Manipulation Planning", *Proceedings of the 8th International Cognitive Robotics Workshop at AAAI'12*, Toronto, Canada, 2012.

7. Meriçli, T., M. Veloso and H. L. Akın, "Improving Prehensile Mobile Manipulation Performance through Experience Reuse", *International Journal of Advanced Robotic Systems*, 2014.

8. Meriçli, T., M. Veloso and H. L. Akın, "Experience Guided Achievable Push Plan Generation for Passive Mobile Objects", *Beyond Robot Grasping - Modern Approaches for Dynamic Manipulation at IROS'12*, Algarve, Portugal, 2012.

9. Meriçli, T., M. Veloso and H. L. Akın, "Achievable Push-Manipulation for Complex Passive Mobile Objects using Past Experience", *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, Saint Paul, Minnesota, USA, 2013.

10. Meriçli, T., M. Veloso and H. L. Akın, "Push-Manipulation of Complex Passive Mobile Objects using Experimentally Acquired Motion Models", *Autonomous Robots*, pp. 1–13, 2014.

11. Meriçli, T., M. Veloso and H. L. Akın, "Case-Based Mobile Push-Manipulation: Framework and Applications", *Journal of Intelligent and Robotic Systems*, 2014.

12. LaValle, S. M., *Rapidly-Exploring Random Trees: A New Tool for Path Planning*, Tech. Rep. TR 98-11, Department of Computer Science. Iowa State University, 1998.

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

14. Kuffner, J. J. and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning", *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 995–1001, 2000.

15. Karaman, S. and E. Frazzoli, "Incremental Sampling-based Algorithms for Optimal Motion Planning", *Proceedings of the Robotics: Science and Systems Conference (RSS)*, Zaragoza, Spain, 2010.

16. Yershova, A. and S. M. LaValle, "Improving Motion-Planning Algorithms by Efficient Nearest-Neighbor Searching", *IEEE Transactions on Robotics*, Vol. 23, No. 1, pp. 151–157, 2007.

17. Bruce, J. and M. M. Veloso, "Real-time Randomized Path Planning for Robot Navigation", *Lecture Notes in Computer Science*, pp. 288–295, 2003.

18. Urmson, C. and R. Simmons, "Approaches for Heuristically Biasing RRT Growth", *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Vol. 2, 2003.

19. Cohen, B. J., G. Subramania, S. Chitta and M. Likhachev, "Planning for Manipulation with Adaptive Motion Primitives", *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5478–5485, 2011.

20. Berenson, D., P. Abbeel and K. Goldberg, "A Robot Path Planning Framework that Learns from Experience", *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3671–3678, 2012.

21. Skoglund, A., B. Iliev, B. Kadmiry and R. Palm, "Programming by Demonstration of Pick-and-Place Tasks for Industrial Manipulators using Task Primitives", *International Symposium on Computational Intelligence in Robotics and Automation (CIRA)*, pp. 368–373, 2007.

22. Argall, B. D., S. Chernova, M. Veloso and B. Browning, "A Survey of Robot Learning from Demonstration", *Robotics and Automation Systems*, Vol. 57, No. 5, pp. 469–483, 2009.

23. Ye, G. and R. Alterovitz, "Demonstration-guided Motion Planning", *Proceedings of the International Symposium on Robotics Research (ISRR)*, 2011.

24. Sakoe, H. and S. Chiba, "Dynamic Programming Algorithm Optimization for Spoken Word Recognition", *Acoustics, Speech and Signal Processing, IEEE Transactions on*, Vol. 26, No. 1, pp. 43–49, 1978.

25. Samadi, M., F. Asr, J. Schaeffer and Z. Azimifar, "Extending the Applicability of Pattern and Endgame Databases", *Computational Intelligence and AI in Games, IEEE Transactions on*, Vol. 1, No. 1, pp. 28–38, 2009.

26. Veloso, M. M., *Planning and Learning by Analogical Reasoning*, Springer-Verlag

New York, Inc., Secaucus, NJ, USA, 1994.

27. Bartsch-Spörl, B., M. Lenz and A. Hübner, "Case-Based Reasoning - Survey and Future Directions", *Proceedings of the 5th German Biennial Conference on Knowledge-Based Systems*, pp. 67–89, Springer Verlag, 1999.

28. Spalazzi, L., "A Survey on Case-Based Planning", *Artificial Intelligence Review*, Vol. 16, pp. 3–36, 2001.

29. Berenson, D., S. Srinivasa and J. Kuffner, "Task Space Regions: A Framework for Pose-Constrained Manipulation Planning", *International Journal of Robotics Research (IJRR)*, Vol. 30, No. 12, pp. 1435 – 1460, 2011.

30. Pettersson, O., "Execution Monitoring in Robotics: A Survey", *Robotics and Autonomous Systems*, Vol. 53, pp. 73–88, 2005.

31. Michel, O., "Webots: Professional Mobile Robot Simulation", *International Journal of Advanced Robotics Systems*, Vol. 1, No. 1, pp. 39–42, 2004.

32. Lynch, K. M., *Nonprehensile Robotic Manipulation: Controlability and Planning*, Ph.D. Thesis, Robotics Institute, Carnegie Mellon University, 1996.

33. Veloso, M. M., "Flexible Strategy Learning: Analogical Replay of Problem Solving Episodes", *Proceedings of the Twelfth National Conference on Artificial Intelligence (Vol. 1)*, AAAI '94, pp. 595–600, American Association for Artificial Intelligence, Menlo Park, CA, USA, 1994.

34. Veloso, M. M., "Merge strategies for multiple case plan replay", D. Leake and E. Plaza (Editors), *Case-Based Reasoning Research and Development*, Vol. 1266 of *Lecture Notes in Computer Science*, pp. 413–424, Springer Berlin Heidelberg, 1997.

35. K. M. Lynch and M. T. Mason, "Dynamic Nonprehensile Manipulation: Controllability, Planning, and Experiments", *International Journal of Robotics Research*,

Vol. 18, pp. 64–92, 1997.

36. Salganicoff, M., G. Metta, A. Oddera and G. Sandini, "A Vision-Based Learning Method for Pushing Manipulation", *Proceedings of the AAAI Fall Symposium on Machine Learning in Vision: What Why and How?*, 1993.

37. Agarwal, P. K., J. Latombe, R. Motwani and P. Raghavan, "Nonholonomic Path Planning for Pushing a Disk Among Obstacles", *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 1997.

38. Nieuwenhuisen, D., A. van der Stappen and M. Overmars, "Path Planning for Pushing a Disk using Compliance", *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2005.

39. Nieuwenhuisen, D., A. van der Stappen and M. H. Overmars, "Pushing Using Compliance", *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2006.

40. de Berg, M. and D. Gerrits, "Computing Push Plans for Disk-Shaped Robots", *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2010.

41. Khatib, O., "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots", *The International Journal of Robotics Research*, Vol. 5, No. 1, pp. 90–98, 1986.

42. Igarashi, T., Y. Kamiyama and M. Inami, "A Dipole Field for Object Delivery by Pushing on a Flat Surface", *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2010.

43. Lau, M., J. Mitani and T. Igarashi, "Automatic Learning of Pushing Strategy for Delivery of Irregular-Shaped Objects", *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2011.

44. Walker, S. and J. K. Salisbury, "Pushing Using Learned Manipulation Maps",

Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 2008.

45. Zito, C., R. Stolkin, M. Kopicki and J. Wyatt, "Two-level RRT Planning for Robotic Push Manipulation", *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012.

46. Kopicki, M., S. Zurek, R. Stolkin, T. Mörwald and J. Wyatt, "Learning to Predict How Rigid Objects Behave Under Simple Manipulation", *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2011.

47. Scholz, J. and M. Stilman, "Combining Motion Planning and Optimization for Flexible Robot Manipulation", *Proceedings of the IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pp. 80–85, 2010.

48. Dogar, M. and S. Srinivasa, "A Planning Framework for Non-Prehensile Manipulation under Clutter and Uncertainty", *Autonomous Robots*, Vol. 33, No. 3, pp. 217–236, 2012.

49. Katz, D. and O. Brock, "Manipulating Articulated Objects with Interactive Perception", *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 272–277, Pasadena, CA, 2008.

50. Melchior, N. and R. Simmons, "Particle RRT for Path Planning with Uncertainty", *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1617–1624, 2007.

51. Berg, J. V. D., P. Abbeel and K. Goldberg, "LQG-MP: Optimized Path Planning for Robots with Motion Uncertainty and Imperfect State Information", *Proceedings of the Robotics: Science and Systems Conference (RSS)*, Zaragoza, Spain, 2010.

52. Bry, A. and N. Roy, "Rapidly-exploring Random Belief Trees for Motion Planning Under Uncertainty", *Proceedings of the IEEE International Conference on Robotics*

and Automation (ICRA), pp. 723–730, 2011.

53. Biswas, J., B. Coltin and M. Veloso, "Corrective Gradient Refinement for Mobile Robot Localization", *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2011.

54. Thrun, S. and T. M. Mitchell, "Lifelong Robot Learning", L. Steels (Editor), *The Biology and Technology of Intelligent Autonomous Agents*, Vol. 144 of *NATO ASI Series*, pp. 165–196, Springer Berlin Heidelberg, 1995.

55. Sturm, J., C. Plagemann and W. Burgard, "Body Scheme Learning and Life-Long Adaptation for Robotic Manipulation", *Proceedings of the Workshop on Robot Manipulation at the Robotics: Science and Systems Conference (RSS)*, Zurich, Switzerland, 2008.

56. Sturm, J., C. Plagemann and W. Burgard, "Adaptive Body Scheme Models for Robust Robotic Manipulation", *Proceedings of the Robotics: Science and Systems Conference (RSS)*, Zurich, Switzerland, 2008.

57. Stilman, M. and J. J. Kuffner, "Navigation Among Movable Obstacles: Real-time Reasoning in Complex Environments", *International Journal of Humanoid Robotics*, Vol. 2, No. 04, pp. 479–503, 2005.

58. Stilman, M., K. Nishiwaki, S. Kagami and J. Kuffner, "Planning and Executing Navigation Among Movable Obstacles", *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 820–826, 2006.

59. Stilman, M., *Navigation Among Movable Obstacles*, Ph.D. Thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, 2007.

60. Pivtoraiko, M. and A. Kelly, "Constrained Motion Planning in Discrete State Spaces", *Field and Service Robotics*, pp. 269–280, 2005.

61. Pivtoraiko, M. and A. Kelly, "Efficient Constrained Path Planning via Search

in State Lattices", *Proceedings of the 8th International Symposium on Artificial Intelligence, Robotics and Automation in Space*, 2005.

62. Howard, T. and A. Kelly, "Optimal Rough Terrain Trajectory Generation for Wheeled Mobile Robots", *International Journal of Robotics Research*, Vol. 26, No. 2, pp. 141–166, 2007.

63. McNaughton, M., C. Urmson, J. Dolan and J.-W. Lee, "Motion Planning for Autonomous Driving with a Conformal Spatiotemporal Lattice", *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4889–4895, 2011.

64. Huang, L., Y. S. Lim, D. Li and C. E. L. Teoh, "Design and Analysis of a Four-wheel Omnidirectional Mobile Robot", *Proceedings of the International Conference on Autonomous Robots and Agents*, 2004.

65. Biswas, J. and M. Veloso, "Depth Camera Based Indoor Mobile Robot Localization and Navigation", *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1697–1702, 2012.