# Safe Motion Planning under Uncertainty for Mobile Manipulators in Unknown Environments

by

**Vinay Kumar Pilania**

B.Tech., and M.Tech. Dual Degree, Indian Institute of Technology Kharagpur, 2009

Dissertation Submitted in Partial Fulfillment of the
Requirements for the Degree of
Doctor of Philosophy

in the
School of Engineering Science
Faculty of Applied Sciences


**SIMON FRASER UNIVERSITY**
**Fall 2015**

# Approval

| | |
|---|---|
| **Name:** | **Vinay Kumar Pilania** |
| **Degree:** | **Doctor of Philosophy** |
| **Title:** | ***Safe Motion Planning under Uncertainty for Mobile Manipulators in Unknown Environments*** |
| **Examining Committee:** | **Chair:** Dr. Rodney Vaughan<br>Professor |

**Dr. Kamal K. Gupta**
Senior Supervisor
Professor _____

**Dr. Parvaneh Saeedi**
Supervisor
Associate Professor _____

**Dr. Carlo Menon**
Supervisor
Associate Professor _____

**Dr. Ahmad Rad**
Internal Examiner
Professor, School of Mechatronic
Systems Engineering _____

**Dr. Siddhartha Srinivasa**
External Examiner
Associate Professor, RI, CMU
Pittsburgh, USA _____

**Date Defended:** December 8, 2015

# Abstract

For a mobile manipulator to operate and perform useful tasks in human-centered environments, it is important to work toward the realization of robust motion planners that incorporate uncertainty inherent in robot's control and sensing and provide safe motion plans for reliable robot operation. Designing such planners pose a significant challenge because of computational complexity associated with mobile manipulator planning and planning under uncertainty. Current planning approaches for mobile manipulation are often conservative in nature and the uncertainty is largely ignored. In this thesis, we propose sampling-based efficient and robust mobile manipulator planners that use smart strategies to deal with computational complexity and incorporate uncertainty to generate safer plans. The first part of the research addresses the design of an efficient planner for deterministic case, where robot state is fully known, and then subsequent extension to incorporate base pose uncertainty. In the first part, we propose a Hierarchical and Adaptive Mobile Manipulator Planner (HAMP) that plans both for the base and the arm in a judicious manner - allowing the manipulator to change its configuration autonomously when needed if the current arm configuration is in collision with the environment as the mobile manipulator moves along the planned path. We show that HAMP is probabilistically complete. We then propose an extension of HAMP (HAMP-U) to account for localization uncertainty associated with the mobile base position. The advantages of our planners are illustrated and discussed. The second part of the research deals with the computational complexity involved in planning under uncertainty. For that, we propose localization aware sampling and connection strategies that help to reduce the planning time significantly with little compromise on the quality of path. In the third part, we learnt from the shortcomings of HAMP-U and took advantage of our smart strategies developed to combat the computational complexity. We propose an efficient and robust mobile manipulator planner (HAMP-BAU) that plans judiciously and considers the base pose uncertainty and the effects of this uncertainty on manipulator motions. It uses our localization aware sampling and connection strategies to consider only those nodes and edges which contribute toward better localization. This helps to find the same quality of path in shorter time. We also extend HAMP-BAU to incorporate task space constraints (HAMP-BAU-TC). Finally, in the last part of the work, we incorporate our planners (HAMP-BAU and HAMP-BAU-TC) within an integrated and

fully autonomous system for mobile pick-and-place tasks in unknown static environments. We demonstrate our system both in simulation and real experiments on SFU mobile manipulator.

**Keywords:** Planning under uncertainty; Autonomous mobile manipulation; Sampling and Connection strategies; unknown environment; 3D exploration; mobile pick-and-place tasks

# Dedication

*To my parents, my brothers (Vivek, Vikas) and my wife (Varsha).*

# Acknowledgements

First, I would like to acknowledge my senior supervisor, Dr Kamal K. Gupta, for all the support that I received during the course of my PhD studies. I would like to thank him for educating me in writing good research papers and being patient during all the discussions we had.

I would like to thank my fellow graduate students for their constant support during this academic exploration. Thanks also go to my friends at the MENRVA lab, for helping me in constructing my final experiment setup and many more.

Finally, I would like to acknowledge the tremendous sacrifices that my mother *S. Pilania* and my father *R.S. Pilania* made to ensure that I get an excellent education. I would like to thank my younger brothers, *Vivek* and *Vikas*, for their incredible patience. It has been painful staying away from you all. I would like to thank my wife *Varsha* for her unconditional support during my studies. Whenever the PhD memories will be remembered, the episode of my final experiments, when my wife stood with me shoulder to shoulder for those continuous 48 hours experimental work (we hardly took rest for less than 5 hours), will be remembered first.

# Table of Contents

# List of Tables

# List of Figures

# Acronyms

| | |
|---|---|
| **ATACE** | Alternate Task-space And C-space Exploration |
| **BiRRT** | Bi-directional Rapidly-exploring Random Tree |
| **BRM** | Belief Roadmap |
| **BRM-LAS** | Belief Roadmap with Localization Aware Sampling |
| **CBiRRT** | Constrained Bi-directional Rapidly-exploring Random Tree |
| **EKF** | Extended Kalman Filter |
| **HAMP** | Hierarchical and Adaptive Mobile manipulator Planner |
| **HAMP-BAU** | HAMP with Base uncertainty propagated to Arm motion Uncertainty |
| **HAMP-BAU-TC** | HAMP-BAU with Task Constraints |
| **HAMP-RRT** | HAMP with Rapidly-exploring Random Tree as underlying sub-planners |
| **HAMP-BiRRT** | HAMP with Bi-directional Rapidly-exploring Random Tree as underlying sub-planners |
| **HAMP-U** | HAMP with base pose Uncertainty |
| **HERB** | Home Exploring Robot Butler |
| **IK** | Inverse Kinematics |
| **LAS** | Localization Aware Sampling |
| **LAC** | Localization Aware Connection |
| **Lazy-CPC-PRM** | Lazy Collision Probability Constrained Probabilistic Roadmap |
| **LQG** | Linear Quadratic Gaussian |
| **MCL** | Monte Carlo Localization |

| | |
|---|---|
| **MPV** | Maximize Physical Volume |
| **NBV-A** | Next Best View of Arm |
| **NBV-B** | Next Best View of Base |
| **OMPL** | Open Motion Planning Library |
| **POMDP** | Partial Observable Markov Decision Process |
| **PRM** | Probabilistic Roadmap |
| **ROS** | Robot Operating System |
| **RNG** | Random Number Generator |
| **RRBT** | Rapidly-exploring Random Belief Tree |
| **RRBT-TF** | Rapidly-exploring Random Belief Tree Type Framework |
| **RRBT-LAC** | Rapidly-exploring Random Belief Tree with Localization Aware Connection |
| **RRBT-LAS** | Rapidly-exploring Random Belief Tree with Localization Aware Sampling |
| **RRBT-LASC** | Rapidly-exploring Random Belief Tree with Localization Aware Sampling and Connection |
| **RRG** | Rapidly-exploring Random Graph |
| **RRT** | Rapidly-exploring Random Tree |
| **SLAM** | Simultaneous Localization And Mapping |
| **UAV** | Unmanned Aerial Vehicle |

# Chapter 1

# Introduction

## 1.1 Introduction

Robots that use mobility and manipulation capabilities, such as wheeled mobile manipulators or humanoid robots, can be seen as an attempt by robotics fraternity to design a machine to imitate various capabilities - mobility, manipulation, etc. - of a human being. The utility of such robots, generally called as mobile manipulators, has been demonstrated in a range of indoor and outdoor applications, for example, to assist elder people [1], helper in household chores [2], [3], for search and rescue operations [4], [5], and planetary exploration [6]. The Intel HERB mobile manipulation platform has demonstrated impressive capabilities ranging from pick-and-place objects (collecting the objects and loading them into a dishwasher rack) [2], [7] to push-based manipulation on tabletop environments [8]. One of the key components that imparts intelligence to mobile manipulators is motion planning where a planner plans a collision-free path from start to goal (base poses and manipulator configurations) by minimizing a cost metric, for example, path length. The mobile manipulator motion planning is the main focus of our research work. Even in motion planning, there can be two broad categories, one, the planners for deterministic case where robot state is fully known and second, the planners for stochastic case where robot state is partially known (or uncertainty in robot state). The second category is also popularly known as motion planning under uncertainty. Here, the word stochastic is used in context of uncertainty associated with the mobile base position. In this thesis, we contribute to both the categories. Our contributions are described in four phases and evaluated in context to wheeled mobile manipulators, however, at a certain level of abstraction, they can be extended to humanoid robots as well. First, we consider the mobile manipulator planners for deterministic case. Most work [9, 10, 11] usually takes a very conservative approach, which is, to fold the arm to some safe "home" configuration and then plan for a 2D projected footprint of the mobile manipulator in a projected 2D representation of the world from start base pose to goal base pose. Clearly, this approach has two main limitations: (i) the

projection of the mobile manipulator with extended arm may have a large footprint, and may be in collision with 2D projected map, while the mobile manipulator is collision-free in 3D map, and more fundamentally (ii) it may not always possible to change the arm to a predefined home configuration at base's start pose because of physical constraints or there may be task constraints that prevent the arm being folded, e.g., if the robot is carrying a glass of liquid which needs to be kept vertical to avoid spillage. Another example is where the mobile manipulator is carrying a long payload, say a pole and it needs to continuously move the arm (and thereby the pole to avoid the pole colliding with walls and other objects in the environment) to navigate through the doors and hallways. In such scenarios, mobile manipulator with arm in start configuration can not reach the goal unless it changes the arm configuration several times along the path.

One possible solution to this motion planning problem is to use sampling based planners [12, 13] in full configuration (C-space) of the mobile manipulator. However, besides being somewhat computationally expensive, the computed path for the mobile manipulator may result into undesired and excessive motions for the manipulator. This is primarily because of the randomness associated with sampling based planners and persists even after applying a post processing smoothing filter. In most scenarios, there is no need to move manipulator except at certain base poses - the undesired arm motion (post smoothing) refers to this extraneous manipulator motion while the base is moving. We would like to avoid such undesired manipulator motions. Furthermore, it is generally difficult to ensure tight error bounds on the mobile base that are comparable to those for the arm and hence synchronizing controllers between the two can be difficult. Therefore, it is quite reasonable to execute arm and base motions sequentially, and within this overall paradigm, our proposed planner, as outlined below, is quite reasonable.

The first part of the research addresses the design of an efficient mobile manipulator planner for deterministic case. We propose a Hierarchical and Adaptive Mobile Manipulator Planner (HAMP) that plans both for the base and the arm in a judicious manner - allowing the manipulator to change its configuration autonomously when needed if the current arm configuration is in collision with the environment as the mobile manipulator moves along the planned path. Our planner first constructs a base roadmap (using Probabilistic Roadmap (PRM) in the base configuration space) and then for each node in the roadmap it checks for collision status of current manipulator configuration along the edges formed with adjacent nodes, if the current manipulator configuration is in collision, the manipulator C-space is searched for a new reachable configuration such that it is collision-free as the mobile manipulator moves along the edge and a path from current configuration to the new reachable configuration is computed. If no such manipulator configuration is found, then a new edge will be searched for in the base roadmap, and the process repeats.

Summarizing, HAMP searches in two sub-spaces (base sub-space and manipulator sub-space) in a novel way and on a "need to" basis, i.e., the search in manipulator space is

invoked only for those points in the base-space where it is needed. Hence, HAMP searches a much smaller size of space, as a result, it computes paths in shorter time with higher success rate than a search in the full configuration space of the mobile manipulator, and more importantly, it also avoids unnecessary motion of the arm, as is the case for the full search. Both these key points are validated in our experiments. Our choice to use PRM as underlying core sub-planner (for base and for the manipulator) within the HAMP framework is primarily because when we incorporate base uncertainty (as explained later in this section) in the mobile manipulator paths, it allows us to optimize the paths with respect to the base uncertainty (at the goal). This would not be the case if we were to use tree versions of sampling based planners (such as RRT [12]) as core sub-planners within HAMP. However, RRT (in the absence of uncertainty) is generally more efficient in terms of planning time than PRM, especially the Bi-directional RRT [14]. Therefore, we also evaluated the tree versions of HAMP with RRT and Bi-directional RRT as the core underlying sub-planners. Moreover, we provide a mathematical proof to show that HAMP is probabilistically complete.

Safe execution of motion plans is of critical importance for many robotic tasks. As a result of uncertainty associated with a robot's motion and its sensory readings, the true robot state is not available. Deterministic planners (including HAMP) assume deterministic motion and leave the issues of uncertainty to the control phase in which the path is executed with a feedback controller. However, a mobile base inherently has localization uncertainty due to wheel slippage and other unmodeled errors. Therefore, a planning method must account for these uncertainties for safe and collision-free execution of motion plans. Partially observable Markov decision process (POMDP) [15] is a general framework to deal with motion and sensing uncertainty, however due to its significant complexity, solving realistic problems with large state spaces remains a challenge, even though progress has been made on the efficiency issues of these approaches [16, 17, 18, 19]. A class of methods that carries robot state and associated uncertainty is an approximation to POMDP. Among them, a sub-class [20, 21, 22] assumes the presence of landmark regions in the environment where accumulated motion uncertainty can be "reset". Another sub-class [23, 24, 25, 26, 27, 28, 29] uses sampling-based methods (graph-based and tree-based) where uncertainty is propagated from start to goal. We call this sub-class as sampling-based stochastic motion planners.

In this thesis, we address the sampling-based stochastic motion planners for mobile manipulators. Motion planning under uncertainty has made considerable progress over the past few years for mobile robots and unmanned aerial vehicles (UAVs) but still largely ignored for mobile manipulators. One key reason might be because most work usually take conservative approach, which is, to fold the arm to some "safe home configuration" and treat mobile manipulator essentially as a mobile robot where planners designed for them can be used to deal with uncertainty. Although, we assume that the motion of manipulator, in and of itself, is quite accurate (a reasonable assumption, give the joint

encoders are quite precise), however, the uncertainty in mobile base position causes some repercussions on manipulator (robotic arm) motion and grasping task (robotic hand). We group the base pose uncertainty and its effects in three levels: Level 1 - the base pose uncertainty is not translated to manipulator motion and grasping task; Level 2 - the base pose uncertainty is translated to manipulator motion but not to grasping task; Level 3 - in addition to Level 2, the base pose uncertainty is also translated to grasping task. Please note that even for a fixed mobile base, there is uncertainty associated with grasping tasks [30], which arises mainly from errors in object localization and perceived shape. Most of the existing grasping approaches deal with uncertainty in the execution stage using reactive grasp execution controller where feedback from range or image sensors and tactile sensors is used to guide the gripper. In this thesis, we consider only Level 1 and Level 2 and leave the grasping uncertainty (Level 3) for future work.

The first part of the research also addresses the extension of HAMP to incorporate Level 1 base pose uncertainty. We propose a mobile manipulator planner, HAMP-U, that uses belief space planning to account for localization uncertainty associated with the mobile base position and ensures that the resultant path for the mobile manipulator has low uncertainty at the goal. A probability distribution over all possible states is referred as belief and the set of all possible distributions is called belief space. Our experimental results show that the paths generated by HAMP-U are less likely to result in collision and are safer to execute than those generated by HAMP (without incorporating uncertainty), thereby showing the importance of incorporating base pose uncertainty in our overall HAMP algorithm.

Although HAMP-U helped in generating paths which are less likely to result in collision, still these paths are not completely safe for execution. There are two main reasons for this: a) HAMP-U does not consider the effect of base pose uncertainty on manipulator motions, b) HAMP-U assumes the robot is at the mean position (of belief, represented by uncertainty ellipse) and checks collision from mean position to mean position. In both cases if the mobile base slightly deviates from its intended path then the arm (which was in some collision-free configuration along intended path but may not necessarily hold along the deviated path) could collide with the surrounding obstacles. Below, we mention the problems involved in addressing above mentioned issues, followed by the solutions.

It is important to note that addressing the shortcomings of HAMP-U requires involvement of uncertainty related techniques at different stages of path planning. This in turn requires costly operation of 3D collision checks, another possible reason why planning under uncertainty for mobile manipulators may be largely ignored. Designing a reliable mobile manipulator planner is difficult to realise for real time applications unless one addresses the issue of computational complexity involved with mobile manipulator planning. [31] deals with it by using a multi-layered 2D representation of both the robot and the environment. However, since the planning is still carried out only for the base and not the manipulator, their approach will fail in the scenarios where arm configuration needs to be changed while

navigating from start to goal. Our strategy in HAMP helps to avoid unnecessary 3D collision checks without being overtly conservative. It first checks the 2D projected footprint of the base against the 2D representation of the world (obtained from projecting 3D range date up to certain height), and if it is collision-free then a 3D collision check is performed on the manipulator. Although, our strategy helps to reduce the planning time (from collision checks perspective), we need to find additional efficiencies for mobile manipulator planning under uncertainty as incorporating uncertainty further increases the computational time.

One way is to look at next level, i.e., after doing collision checks for the sampled point, do we really need to retain all the points (nodes) or the local paths (edges) connecting two points. A good decision at this level (before connecting the sampled point to the graph or tree) could help us improve the run time associated with mobile manipulator planning. Note that expensive 3D collision checking is not the only factor involved in computational complexity, there is another facet to it other than the high-dimensional full configuration space (which we handle in HAMP by planning in two different sub-spaces). It is important to understand why incorporation of uncertainty makes the mobile manipulator planning computationally expensive.

The sampling-based stochastic motion planners can be implemented either in an incremental (graph-based [28] or tree-based [24, 25, 27]) or in a non-incremental way (graph-based [23, 26, 27, 24]). These planners are computationally demanding as compared to their counterparts that do not consider uncertainty (deterministic motion planners). This is because they do not follow the "optimal substructure" property of paths, i.e., the incurred costs on different edges depend on each other. To compute the cost of an edge emanating from a node, the full knowledge of belief (robot pose and associated uncertainty) at the node is required, this in turn requires full knowledge of the history of observations and actions leading up to the node. [29] is an exception in the sense that the incurred costs on different edges do not depend on each other. This comes at the cost of some simplifying assumptions including holonomic robot and Gaussian belief for robot states with trivial dynamics. The computational cost further increases if an edge cost in these planners uses collision probability [25, 27, 29], computation of which depends on the beliefs along that edge. Furthermore, this cost will go up drastically if collision checks are carried out in 3D (for example, for mobile manipulators). Since the time consuming step in stochastic motion planners arises from the uncertainty propagation along the edges, incremental stochastic planners can be computationally more demanding as compared to non-incremental ones where search mechanism is carried out only once while in former, search mechanism is repeated every time a new sample is added to the roadmap. For real time applications, for instance to facilitate anytime planning [32], it is important to reduce this run time. At least part of this run time reduction can be achieved by "smart" sampling and connection strategies. Current stochastic motion planners [23, 24, 25, 26, 27, 28, 29] use traditional sampling and connection strategies which are designed for deterministic motion planners and address the issue

of uncertainty at path search phase. These strategies add unnecessary nodes and edges that do not contribute to better localization. This leads to a dense roadmap which in turn increases the computational cost.

In the second part of our research, we propose efficient localization aware sampling and connection strategies to bring down the computational cost for sampling-based stochastic motion planners. The localization aware sampling strategy avoids putting large number of samples by considering the "localization ability" of a new sample relative to its neighbouring nodes. It puts more samples in regions where sensor data is able to achieve higher uncertainty reduction while maintaining an adequate number of samples in regions where uncertainty reduction is poor. This leads to a less dense roadmap that results in significant time savings in the path search phase. Note that localization of a robot at a point depends on 1) the path taken to reach the point and 2) on the update based on sensor model. However, at the sampling stage the path taken to a node is not available. We develop a new measure of "localization ability of a sample" that "extracts" how well a sensor observation at a sample point reduces uncertainty without explicitly knowing the path leading to it and use this measure to design a localization aware sampling strategy.

A key reason we use reduction in uncertainty as a measure is that higher uncertainty is more detrimental and hence has higher cost for many tasks. Nevertheless, one possible consequence of our sampling technique is that path quality (we use true localization uncertainty along the path as a quality metric) may suffer, if the path passes through regions where uncertainty reduction is poor. Via simulation results, we show that, at least empirically, there is little compromise in path quality. Furthermore, note that since at the sampling stage, true localization uncertainty is not available, a cost function metric using it can not be computed, hence can not be used. The best one can do is to use the uncertainty reduction ability of the sensor at the sample point, as we do. Note that in the search phase (where edges are added and uncertainty is propagate along the path), appropriate cost function is still minimized.

The localization aware connection strategy first connects the new sample to a nearest node (chosen based on an uncertainty metric and not on distance metric) and then to other neighbouring nodes. Connection from new sample to a neighbouring node is made only if the new path to that node reduces the uncertainty. Our efficient connection strategy eliminates the inefficient edges that would be created in current connection schemes but do not contribute toward better localization. As a result, it also reduces the number of search queue iterations needed to update the paths. This helps to find a well-localized path in shorter time with no compromise on the quality of path. Note that our strategy is applicable to graph-based incremental stochastic planners that maintain a single belief at a node and is not applicable to planners with multiple beliefs. Multiple beliefs at a node are needed for planners that optimize multiple objective functions since multiple paths to a node can not be completely ordered (as is the case for a single objective function, which can

be a weighted sum of multiple costs) and need to be kept so as not to prematurely prune an optimal one, although domination criteria can be used to do some pruning (see [28, 33], the later is more specific to manipulator planning for fixed base). Of course, tree-based methods, by definition, have single belief since they have a unique path to any given node. Our simulation results showed that by using these smart strategies, the planning time can be reduced significantly with little compromise on the quality of path. Our simulations involved mobile robot (2D planning), therefore, it is expected that the savings in planning time will be even higher for mobile manipulator (as we show in our third part of research). The probabilistic completeness issues with our approaches are also discussed.

In the third part of our research, we integrated HAMP with the smart sampling and connection strategies to implement an efficient and robust mobile manipulator planner (HAMP-BAU) that plans judiciously and considers the base pose uncertainty and the effects of this uncertainty on manipulator motions (Level 2). It uses our localization aware sampling and connection strategies to consider only those nodes and edges which contribute toward better localization. Moreover, it respects the collision probability threshold along the path and uncertainty threshold at goal. We also propose an extension of HAMP-BAU to incorporate task space constraints (we call the resultant planner as HAMP-BAU-TC). We evaluated both the planners and show that our planners find a safer path as compared to other variants where uncertainty is not considered at different levels, for example, not incorporating base uncertainty on manipulator plans, not respecting collision probability threshold along the edges. We also show that the variants of these planners that do not use our localization aware sampling and connection strategies will take longer to find the same quality of path.

Finally, in the last part of work, we incorporate our planners (HAMP-BAU and HAMP-BAU-TC) within an integrated and fully autonomous system for mobile[1] pick-and-place tasks in unknown static environments. A key aspect of our integrated system is that the planner works in tandem with base and arm exploration (view planning) modules that explore the unknown environment. Note that we assume unknown areas of environment as obstacles and not free. The task of base exploration is to take the mobile manipulator (mainly mobile base) to next best view of the base (NBV-B), take a scan using a 3D sensor (Kinect, mounted on the mobile base) and then invoke arm exploration which scans the local region surrounding the manipulator using a 2D sensor (Hokuyo, mounted at end-effector acting as eye-in-hand) by reaching to next best views of arm (NBVs-A). From 2D sensor (or a 2D scan) we mean line scan while from 3D sensor (or a 3D scan) we mean area scan. Also note that since the eye-in-hand sensor can provide only line scans, therefore, at NBV-A, the sensor rotates to make an area scan by collecting all the line scans during rotation. Scans from both Kinect and Hokuyo sensors are inserted into a global Octomap [34]. The base exploration module works on a 2D occupancy grid map to compute a NBV-B and uses

[1]The world "mobile" emphasizes that the mobile manipulator is required to move from one location to another.

HAMP-BAU to plan a path for it. This 2D occupancy grid map is obtained by the fusion of two 2D maps - one is from down projection of global Octomap upto a certain height and other is from SLAM (simultaneous localization and mapping). There is a third sensor (LMS100, mounted at base bottom) which feed SLAM. Please see Figure 5.1 for system components (mobile manipulator, sensors and their mounting locations) and Figure 5.6 for how the sensor information is used and different maps are obtained. On the other hand, the arm exploration module works on a local Voxelmap (obtained from global Octomap) to compute NBVs-A and uses a manipulator planner (that considers base pose uncertainty) to reach there. After each scan, the arm exploration module updates the Voxelmap and repeat the procedure until the Voxelmap is fully explored. It is important to note that since sensor scans are not directly inserted into Voxelmap, therefore, the status (occupied, free, unknown) of each voxel cell in the Voxelmap is updated by communicating with global Octomap. We also want to state that in addition to scans taken by respective sensors at NBV-B and NBV-A, the scans collected during the mobile manipulator motion to reach NBV-B or the manipulator (end-effector) motion to reach NBV-A are also incorporated into the global Octomap. Our system is implemented both in simulation and on the actual SFU mobile manipulator. Please note that the pick and place modules that we use in our integrated system are very quick and an ad hoc attempt to be able to show the system for pick-and-place task. In future, it will be replaced by a systematic approach.

## 1.2 Related Work

In this section, we review the related work and place our research work into context. We consider the work concerning mobile manipulator motion planning, planning under uncertainty, sampling and connection strategies, mobile manipulator based autonomous systems in unknown environment.

### 1.2.1 Mobile Manipulator Motion Planning

Most of the previous work [35, 36, 37, 38] on mobile manipulation mainly deals with the coordination of the mobile base and the manipulator motion for following a given end effector trajectory. In motion planning related work, [10] and [11] use a compact 3D representation of the environment, but path planning is accomplished in a projected 2D environment representation with a 2D footprint of the mobile manipulator. Such an approach will fail, for example, where the mobile manipulator is required to push and store a cart under a table. [31] improved upon [10, 11] using a multi-layered 2D representation of both the robot and the environment. However, since the planning is still carried out only for the base and not the manipulator, their approach will fail in the scenarios where arm configuration needs to be changed while navigating from start to goal. [39] proposed an adaptive approach for efficient humanoid robot navigation, which allows for finding solutions for foot-step planning

where planning based on a 2D grid fails. Our approach (HAMP) has a similar adaptive flavour, but it is in the context of mobile manipulation and not foot step planning. In the context of mobile manipulators, hierarchical strategies have been used to estimate reachable workspace [40]. An interesting use of adaptive dimensionality has recently been introduced in [41]. Their approach uses deterministic search (A* over discretized C-space) in a low dimensional end-effector C-space interleaved with tracking in the full mobile manipulator C-space. It is shown that the resulting planner outperforms a full dimensional RRT in a class of tasks where the end-effector is carrying a large payload. One could characterize this approach toward the "greedy" end of the spectrum since the search is, in effect, guided by a path for the end-effector. While this approach could be used in a relatively small region near the goal, as shown in the example tasks in the above mentioned paper, a key problem is that due to its deterministic search, it is not applicable to relatively large areas as is the case in our examples. Finally a genetic optimization based planner for a mobile manipulator that plans motions in real time in dynamic environments is presented in [42]. The planner takes advantage of redundancy in optimizing overall motion via randomly invoking a "stop" genetic operator that allows for either the base or the manipulator to remain stationary during a portion of the trajectory. Note that the performance of genetic optimization relies on maintaining a diverse population of trajectories that belong to different homotopic groups which is a significant challenge.

### 1.2.2 Sampling Based Planning Under Uncertainty

While standard motion planning algorithms often assume that a mobile base can track its position reliably during path execution stage (as is the case with HAMP), in reality, there is always some uncertainty associated with mobile base position. The uncertainty typically originates from three sources: (i) motion uncertainty - uncertainty in a robot's motion often caused by factors such as wheel slippage, (ii) sensor uncertainty - uncertainty in its sensory readings, and (iii) map uncertainty - uncertainty in the environment map or imperfect locations of features (information sources) in the environment.

Planning under uncertainty has made considerable progress over the past few years for mobile robots. For example, approaches in [26, 25, 24, 43, 23, 44], essentially add an uncertainty dimension(s) to the robot state and each belief state then is a combination of robot state and the associated uncertainty. An attractive aspect of Belief Roadmap (BRM) [26] is that, while it explicitly simulates measurements along candidate paths and then chooses the path with minimal uncertainty at the goal, it uses covariance factorization techniques to significantly reduce the computation burden of this process but with the assumption of maximum likelihood observation, i.e., the controller is capable of driving the state estimate back to the desired path. More recent approaches have also accounted for the controller in the planning stage, e.g., [27, 28, 29], however, there is significant increase in the computational cost. These planners only consider the motion and sensing uncertainty.

Approaches in [45, 46, 47, 48] consider the mapping uncertainty about the environment but not the motion and sensing uncertainty.

While mobile robotics literature (mobile base only) has extensively considered uncertainty (world is 2-dimensional in most of these cases, although some recent work has considered 3D world, but for SLAM and not planning), to the best of our knowledge, this uncertainty is largely ignored in mobile manipulation. [33] considered this, but for the case of fixed mobile base only.

### 1.2.3 Sampling Strategies

A large number of sampling schemes have been used with the standard (without uncertainty) sampling based planners (RRT or PRM) such as, sample around and near the obstacles, or in narrow corridors, medial axis sampling to sample far away from the obstacles, use visibility to reduce the number of samples, adaptive strategies such as restrict sampling to size-varying balls around nodes, entropy guided approaches, etc. [49] and [50] provide a survey of recent work in non-uniform sampling for PRMs. Above mentioned sampling approaches do not consider the uncertainty associated with robot and its sensors.

[45] proposed an approach where the sampling strategy incorporates mapping uncertainty (they do not consider localization uncertainty that we consider in this paper) in which the decision to accept or reject a sample is based on its collision probability (computed using each of the possible world model). However, the issue of "how good a sample would be in localizing the robot?", which we explicitly consider does not arise in their problem context. As mentioned earlier, computing the collision probability in the presence of localization uncertainty of a sample right at the sampling stage, i.e., before connecting it to the roadmap is not possible. Note that at sampling stage we consider only sensing uncertainty while for path search (where uncertainty is propagated from start) we consider both motion and sensing uncertainty. To the best of our knowledge, we are not aware of any other sampling approach that considers uncertainty. All sampling-based stochastic motion planners [23, 24, 25, 26, 27, 28, 29] use one of the sampling techniques from deterministic motion planners and address the motion and sensing uncertainty at path search phase by propagating uncertainty from start to goal.

Although not directly related to motion planning (or sampling techniques), the notion of uncertainty has been used in the past to select the best sample (the next best goal of robot) for search and exploration. For example: [51] first plans for each of the possible goal candidates and selects the one (as next best goal) which in addition to information maximization (unknown region), also has good localization along the path.

### 1.2.4 Connection Strategies

Connection strategies used in sampling-based deterministic motion planners simply connect the new sample to the neighbouring nodes within in a fixed size ball or size varying ball. A thorough discussion on these strategies can be found in [52, 53] while for more recent updates we refer to [49, 54]. These approaches do not account for uncertainty associated with robot and its sensors.

All [23, 24, 25, 26, 27, 28, 29] of the sampling-based stochastic motion planners that consider uncertainty inherit the connection strategy from deterministic motion planners. Among incremental planners, [28] is obliged to use traditional connection strategy as they optimized multiple objective functions, hence are required to maintain multiple paths to (hence multiple beliefs at) a node in order to guarantee not to prune an optimal path, although some pruning can be done via domination criteria. To the best of our knowledge, the work of [28] is the only roadmap (graph) based stochastic motion planner that works in an incremental fashion. Although it is designed for a set of beliefs, the same strategy also works for the case of single belief at a node. We call their algorithm (RRBT) with single belief as RRBT type framework (RRBT-TF). It minimizes the uncertainty at goal while respecting the chance-constraints (threshold on uncertainty) along the path. Planners in [23, 24, 25, 26, 27, 29] also use single belief at a node but they do not incrementally construct the roadmap.

The problem with the use of traditional connection strategy for incremental stochastic planners is that it even considers those edges which do not contribute toward better localization. With the inclusion of such edges, the planning time increases, however, the same quality of path can be find in lesser time if we eliminate these edges. This is exactly what our localization aware connection strategy does. It eliminates those edges which do not contribute toward better localization.

Similar to graph-based incremental stochastic planners, current tree-based stochastic motion planners [24, 25, 27] also inherit the connection strategy from tree-based deterministic motion planners. There the EXTEND step simply connects the sample to nearest node (distance based) and then propagate the uncertainty to it. However, this does not provide the least uncertain path to the sample. Instead, our connection strategy will connect the sample to a neighbouring node (within a ball) the uncertainty propagation from which gives minimal uncertainty at the sample. We use the additional “rewiring" notion of RRT* [54], albeit with uncertainty metric, to rewire the connections to the neighbouring nodes.

### 1.2.5 Mobile manipulator based autonomous systems in unknown environment

Please note that in this section we do not talk about individual exploration (view planning algorithms) techniques for either base or the arm. There is huge literature on that

and a good review can be found in [55]. But we review the related work on integrated and autonomous systems that use mobile manipulator for some application in unknown environment. Note that we consider unknown regions of the environment as obstacles and not collision-free regions (which can be detrimental) as the assumption in [56]. Neither the work in [56] uses view planning to explore the environment. It just incorporates the sensor readings (from a 3D sensor mounted on the base and not acting as eye-in-hand) as the mobile manipulator moves along the path that follows end-effector trajectory. [57] searches for an object in the unknown environment using a planar range sensor mounted at end-effector but mobile base was fixed in their experiments. A system proposed by Lila Torabi [55, 58, 59] (an earlier Ph.D. Thesis work from our Robotic Algorithms and Motion Planning (RAMP) Lab) autonomously builds a 3D model of an object placed in unknown environment. The work considers decoupled approach for mobile manipulator planning and moreover, uncertainty is not considered. There is lot of work related to environment exploration and mapping both in 2D and 3D either using mobile base [51] or UAVs [60]. However, we have not come across any work where mobile manipulator is used to explore the unknown environment and achieve some tasks (for example, mobile pick-and-place). We believe that our system is first of its kind in many ways: a) it is the first integrated application that explores the unknown environment, picks the object (once the object is deemed to be in the known region) and then further explores the environment with object in hand and places it at target location only after the place location is deemed to be in the known region, b) it combines two different exploration schemes into one - uses frontier based exploration for the base and information gain maximization (in workspace) based exploration technique for the arm, and finally c) how the scans from multiple sensors are integrated and then used for base and arm view planning.

## 1.3 Contributions

There are two broad visions associated with this thesis. The first is to design an efficient and reliable mobile manipulator planner that plans judiciously both for the base and the arm and considers the base pose uncertainty and the effects of this uncertainty on manipulator plans. The second is to integrate such planner into a system that autonomously explores the unknown environment to complete an assigned task, for example, mobile pick-and-place task in our case. The key contributions of the thesis are listed below:

- We designed a novel mobile manipulator planner (HAMP) for deterministic case that plans both for the base and the arm in a judicious manner. We also evaluated the tree versions of HAMP with RRT and Bi-directional RRT as the core underlying sub-planners. A mathematical proof is also provided to show that HAMP is probabilistically complete. Furthermore, we extended HAMP to design a new planner (HAMP-U)

that incorporates localization uncertainty associated with the mobile base position. These works have been published in [61] and [62].

- We designed novel efficient localization aware sampling and connection strategies for sampling based motion planning under uncertainty. For sampling, we developed a new measure of "localization ability of a sample" that "extracts" how well a sensor observation at a sample point reduces uncertainty without explicitly knowing the path leading to it. A mathematical proof is also provided to show the probabilistic completeness of our sampling strategy under some reasonable conditions on parameters. The sampling work has been published in [63], while the connection work has been published in [64].

- Integrating the above two components, we designed an efficient and robust mobile manipulator planner (HAMP-BAU) that plans judiciously and incorporates the base pose uncertainty and the effects of this uncertainty on manipulator plans. We also extended HAMP-BAU to incorporate task space constraints (HAMP-BAU-TC). This work is reported in [65].

- We incorporated HAMP-BAU and HAMP-BAU-TC in an integrated and fully autonomous system for mobile pick-and-place tasks in unknown static environments. The system is demonstrated both in simulations and real experiments on SFU mobile manipulator. This work is also reported in [65].

## 1.4 Outline of Thesis

The rest of the thesis is organised as follows. In Chapter 2 we present a mobile manipulator planner (HAMP) for deterministic case, its probabilistic completeness proof and extension to incorporate base pose uncertainty (HAMP-U). In Chapter 3 we present localization aware sampling and connection strategies for sampling based motion planning under uncertainty. In Chapter 4 we present a more advanced and safer mobile manipulator planner (HAMP-BAU) and its extension (HAMP-BAU-TC) to incorporate task space constraints. In Chapter 5 we describe a mobile manipulator based autonomous system for mobile pick-and-place tasks in unknown environment and demonstrate the system in simulations and real experiments on SFU mobile manipulator. We conclude in Chapter 6.

# Chapter 2

# Mobile Manipulator Planning I

In this chapter, we first present a mobile manipulator Planner (HAMP) for deterministic case, evaluate it and its tree variants extensively in different scenarios, and provide a mathematical proof to show the probabilistic completeness of HAMP. We then present an extension of HAMP to incorporate base pose uncertainty (HAMP-U). We show that the paths generated by HAMP-U are less likely to result in collision and are safer to execute than those generated by HAMP. Recall that (from Section 1.1) HAMP-U falls in Level 1 category where only base pose uncertainty is considered and the effects of this uncertainty are not considered on manipulator motions. A more advanced version (HAMP-BAU) which falls in Level 2 category is presented in Chapter 4.

## 2.1 Overview of HAMP and HAMP-U

We propose a Hierarchical and Adaptive Mobile Manipulator Planner (HAMP) and a schematic illustrating the planned mobile manipulator path is given in Fig 2.1. The HAMP algorithm is a two stage process: in the first stage it constructs a base roadmap (using PRM in the base configuration space) where it connects the start and goal base poses (with manipulator remaining in a fixed home[1] configuration). In the second stage, the algorithm reconfigures or "adapts" the manipulator configuration to a new configuration along the edges in the base roadmap constructed earlier by checking for the manipulator collisions along them from start to goal. This two stage process iterates until a collision-free mobile manipulator path is found or the time limit is over. The second stage works as follows: for each node in the base roadmap, the current manipulator configuration is checked for collisions along the edges corresponding to the adjacent nodes. If it is in collision along an edge in the base roadmap, then the manipulator is reconfigured (while base is stationary at the base node) by moving it to a new configuration such that the new configuration is

[1]Note that there are other options here, e.g., one could simply construct the base roadmap for the base only, however, this could lead to several nodes/edges being invalidated in the subsequent stage.

collision-free if the mobile manipulator with manipulator in new configuration were to move along the edge in the base roadmap. This reconfiguration step is carried out via motion planning for the manipulator in the manipulator's C-space constructed at the given base node. If no such manipulator configuration is found, then a new edge will be searched for in the base roadmap, and the process repeats.

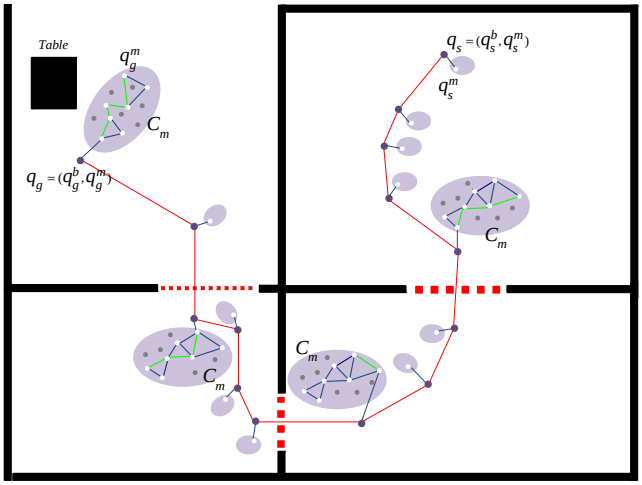



Figure 2.1: A schematic illustrating the planned mobile manipulator path $\Pi^{bm}$ given by HAMP algorithm. Please see text for explanation.

Fig 2.1 schematically illustrates HAMP algorithm. In the figure, blue dots correspond to base pose nodes, the red segments are the base edges, and light purple ellipses (small and big) corresponding to each blue dot is the manipulator C-space. Small purple ellipses with one white dot indicate that the manipulator configuration, corresponding to the white dot, is free along the base edge (to the next base node) and no manipulator planning was required. Three red color dash lines denote the physical gates (overhead view). The big ellipses show where manipulator planning was done, with the manipulator roadmap shown with its nodes and edges inside each ellipse. For the first three ellipses, the manipulator configuration at each base node just before the gate was in collision along the edge (as the mobile manipulator moves through the gates) and hence the roadmap was built and searched for a path and the sequence of light green edges shows the path. The manipulator moves along this path to the end configuration, which is, by construction (as explained in the detailed algorithm in Sec 2.3), collision free as the mobile manipulator moves across the gate to the next base node. The fourth big ellipse (at base goal pose) shows a reconfiguration step to the goal configuration of the manipulator.

We extend our HAMP approach - we call it HAMP-U - to account for localization uncertainty associated with the mobile base position and a schematic illustrating the planned mobile manipulator path is given in Fig 2.2. Blue dots correspond to mean base pose nodes, and the uncertainty in position is shown by ellipses (green color). In HAMP-U, in the first stage, the base roadmap is substituted by a belief roadmap (BRM) in the belief-space of mo-

bile base, using the approach proposed by [26] with additional modifications explained later in Sec 2.5. In the second stage, the search algorithm searches for the mobile manipulator path in the BRM by propagating base pose uncertainty from start to goal, in a manner that minimizes the goal uncertainty, as in [26], again, with some modifications. Note that, in Fig 2.2, the mobile base path detours from the shortest path (Fig 2.1) through sensing-rich environment to remain well-localized, a direct and well known consequence of standard BRM. Due to this low uncertainty in base position, the planned mobile manipulator motions are also less likely to result in collision as validated in our experiments.

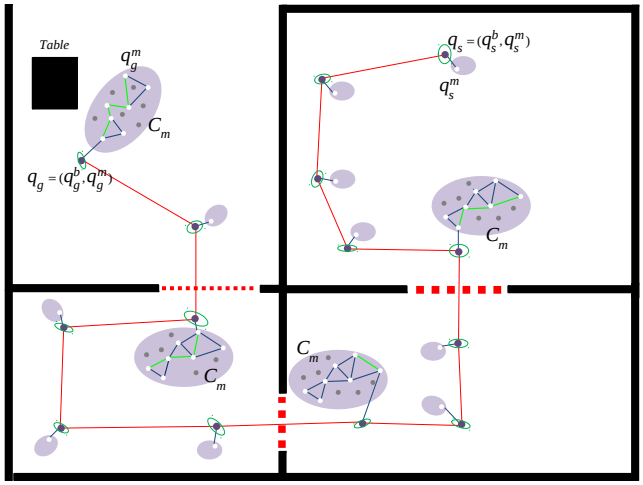



Figure 2.2: A schematic illustrating the planned mobile manipulator path $\Pi^{bm}$ given by HAMP-U algorithm. Please see text for explanation.

## 2.2 Problem Formulation

We use $q_i = (q_i^b, q_i^m)$ in $C_{bm}$, the C-space of the mobile manipulator, to represent $i^{th}$ mobile manipulator configuration, where $q_i^b = [x, y, \theta] \in C_b$, the C-space of the mobile base, is the base configuration (also called base pose) and $q_i^m = [\theta_1, \theta_2, ...., \theta_d] \in C_m$, the C-space of the $d$ degree of freedom manipulator, is the manipulator configuration. $C_{b_{free}}$ is the set of all collision-free base poses and $C_{b_{obs}}$ is the set of poses resulting in collision with obstacles. For a given base pose, $q_i^b$, $C_{m_{free}}$ denotes the set of free manipulator configurations (for simplicity we omit the reference to the corresponding base node $q_i^b$ in the notation) and $C_{m_{obs}}$ denotes the set of manipulator configurations that are in collision with obstacles. We use $q_H^m$ to denote the home configuration of the manipulator, a compact and folded configuration of the arm, specified by the user. For simplicity of explanation, the 3D environment is assumed to be known or acquired by previous sensing, but our framework is extended to the simultaneous sensing and planning by incorporating a view planner similar to [66, 67].

Given a 3D map, the start $q_s = (q_s^b, q_s^m)$ and goal $q_g = (q_g^b, q_g^m)$ configurations of the mobile manipulator, the objective of our HAMP algorithm is to find a collision-free path.

Because of the hierarchical and adaptive approach, the nature of mobile manipulator path will have a specific structure as shown in Fig 2.1 and can be expressed as

$$\Pi^{bm} = \{(q_s^b, \pi_s^m), (q_{r_2}^b, \pi_2^m), ..., (q_{r_{n-1}}^b, \pi_{n-1}^m), (q_g^b, \pi_g^m)\}$$

We call this type of specific mobile manipulator path as an H-path (short for HAMP-path). It consists of a sequence of poses $q_{r_i}^b$ at which the manipulator reconfiguration step takes place (subscript $r$ denotes reconfiguration), i.e., the base remains stationary and the manipulator moves along path $\pi_i^m$ to a new configuration, the end point of $\pi_i^m$. It is implicit in the notation that the mobile manipulator motion from a node, say $(q_{r_i}^b, \pi_i^m)$ to $(q_{r_{i+1}}^b, \pi_{i+1}^m)$ consists of three steps: (i) at $q_{r_i}^b$, the manipulator will reconfigure by moving along $\pi_i^m$ to the last configuration in $\pi_i^m$, (ii) with this new fixed manipulator configuration, the base moves along base path segment $\pi_i^b$ to $q_{r_{i+1}}^b$, and (iii) manipulator again reconfigures by moving along $\pi_{i+1}^m$ to the last configuration in $\pi_{i+1}^m$.

## 2.3 The HAMP Algorithm

We now describe the HAMP algorithm in detail explained in Algorithm 1.

In the first stage, ConstructBaseRoadmap() routine is invoked to build a base roadmap in $C_{b_{free}}$ by randomly sampling base poses (with manipulator in a fixed home configuration) and connecting them as in BasicPRM routine in [49], and the pseudo-code for it is given in Algorithm 2. It constructs a base roadmap in an incremental manner until start and goal nodes are connected. Please note that while the sampling is in $C_b$, the collision checks are done for the entire mobile manipulator with manipulator in fixed home configuration. One could simply construct the base roadmap for base only, however, this could lead to several nodes/edges being invalidated in the subsequent stage. Algorithm 2 returns a connected base roadmap for start and goal base poses with manipulator in home configuration.

For second stage, we augment the node structure such that each node $n$, in addition to a base pose $n[q^b]$, and manipulator configuration $n[q^m]$, now has a best path field $n[p]$, a cost $n[c]$ (Euclidean metric in $C_b$) and a set of reconfiguration path fields $n[\text{RPATHS}^{[n_{adj}]}]$, one path for each adjacent node, $n_{adj}$.

The second stage described in Lines 5-19 of Algorithm 1 is a search mechanism that searches the base roadmap using a variant of Dijkstra's algorithm and SearchManipPath() routine in an intertwined manner. First, we change the manipulator configuration at start node, $n_s$, in the base roadmap from home $q_H^m$ to a given configuration $q_s^m$ and insert it in the search queue (Line 4). This is needed because the base roadmap was constructed with $q_H^m$ which is different from $q_s^m$.

**Algorithm 1:** $\Pi^{bm} = \text{HAMP}(q_s, q_g, q_H^m)$

**Input**: $q_s := (q_s^b, q_s^m)$, $q_g := (q_g^b, q_g^m)$ and $q_H^m$
**Output**: H-path $\Pi^{bm}$ from $q_s$ to $q_g$

1 $G^b :=$ ConstructBaseRoadmap($q_s^b$, $q_g^b$, $q_H^m$)
2 Augment node structure with best path $p := \emptyset$, cost $c := 0$ and reconfiguration paths $\text{RPATHS}^{[n_{adj}]} = \emptyset$ such that $n_i := \{q^b, q_H^m, p, c, \text{RPATHS}^{[\cdot]}\}$
3 **while** ! TimeUp **do**
4  $Q \leftarrow n_s := \{q_s^b, q_s^m, \emptyset, 0, \emptyset\}$
5  **while** $Q \neq \emptyset$ ***and*** ! TimeUp **do**
6  $n :=$Pop($Q$)
7  **if** $n[q^b] = n_g[q^b]$ **then**
8  $\pi_{n_g}^m =$SearchManipPathAtBaseGoal($\hat{q}_g^m, q_g^m$)
9  **if** $\pi_{n_g}^m = \emptyset$ **then**
10  Continue (go to step 5)
11  Exit (go to step 33)
12  **for** ***all*** $n'$ of adj$[n]$ ***and*** $n' \notin n[p]$ **do**
13  **if** $n[c]+cost[e_{nn'}] < n'[c]$ **then**
14  $(q_{new}^m, \pi_n^m) :=$SearchManipPath($n, n'$)
15  **if** $\pi_n^m \neq \emptyset$ **then**
16  $n'[c] := n[c]+\text{cost}[e_{nn'}]$
17  $n[\text{RPATHS}^{[n']}] := \pi_n^m$
18  $n' = \{-, q_{new}^m, n[p] \cup \{n'\}, n'[c], -\}$
19  $Q \leftarrow Q \cup \{n'\}$
20  **if** ! TimeUp **then**
21  **for** ***each*** *node* $n_i$ *in* $G^b$ **do**
22  $n_i := \{n_i[q^b], q_H^m, \emptyset, 0, \emptyset\}$
23  ExpandBaseRoadmap()
24 **return** $\Pi^{bm}$

**Algorithm 2:** $G^b$=ConstructBaseRoadmap($q_s^b$,$q_g^b$,$q_H^m$)

1 $n_s :=$AddNode($q_s^b, q_H^m$); $n_g :=$AddNode($q_g^b, q_H^m$)
2 Create edge if CollisionFree($n_s, n_g$)
3 **while** ! SameComponent($n_s, n_g$) ***and*** ! TimeUp **do**
4  Sample base poses $q_i^b$ from $C_{b_{free}}$ using a standard PRM sampling strategy to build base roadmap node set $\{n_i\}$ such that $n_i := (q_i^b, q_H^m)$
5  Create edge set $\{e_{ij}\}$ between nodes $(n_i, n_j)$ if CollisionFree($n_i, n_j$)
6 **return** $G^b = \{\{n_i\}, \{e_{ij}\}\}$

At each iteration of the while loop (Line 5), a node $n$ is popped out from search queue and if the base component is not the base goal node then the manipulator configuration

**Algorithm 3:** $(q^m_{new}, \pi^m_n)$=SearchManipPath$(n, n')$

**Input**: base roadmap nodes $n, n'$ along an edge $e_{n,n'}$
**Output**: Manipulator path $\pi^m_n$ at node $n$ with $q^m_n$ as start and $q^m_{new}$ as goal such that $q^m_{new}$ is collision-free along an edge $e_{n,n'}$

1 $n'' := (q^b_{n'}, q^m_n)$
2 **if** CollisionFree$(n, n'')$ **then**
3 $\quad q^m_{new} \leftarrow q^m_n$ and $\pi^m_n \leftarrow \{q^m_n\}$
4 **else**
5 $\quad$ goal$^m$ := ComputeArmGoals$(n, n',$ K$_{\text{Goals}}$)
6 $\quad n^m_s$ :=AddNode$(q^m_n)$; $n^m_{g_i}$ :=AddNode(goal$^m[i]$)
7 $\quad$ **while** ! ArmPlanningTimeUp ***and*** ! TimeUp **do**
8 $\quad\quad$ Sample $q^m_i$ from $C_{m_{free}}$ using a standard PRM sampling strategy to build arm roadmap and search for a path $\pi^m_n$ from $n^m_s$ to $n^m_{g_i}$
9 $\quad\quad q^m_{new} \leftarrow n^m_{g_i}$
10 **return** $(q^m_{new}, \pi^m_n)$

**Algorithm 4:** goal$^m$=ComputeArmGoals$(n, n',$K$_{\text{Goals}})$

**Input**: base roadmap nodes $n, n'$ and number of arm configurations (K$_{\text{Goals}}$) to be computed
**Output**: a set of arm configurations as goals

1 **while** $i <$ K$_{\text{Goals}}$ ***and*** ! ArmGoalsTimeUp **do**
2 $\quad$ sample $q^m_i$ from $C_{m_{free}}$
3 $\quad u \leftarrow \{q^b_n, q^m_i\}$ and $v \leftarrow \{q^b_{n'}, q^m_i\}$
4 $\quad$ **if** CollisionFree$(u, v)$ **then**
5 $\quad\quad$ goal$^m$ := goal$^m \cup \{q^m_i\}$; $i := i + 1$
6 **return** goal$^m$

corresponding to node $n$ is checked for collisions along each edge formed with adjacent nodes $n'$ and in case a collision is detected, a reconfiguration path is searched for the manipulator. This is done by invoking a routine SearchManipPath(). If routine SearchManipPath() returns success as shown by the check on Line 15, then we insert adjacent node $n'$ into the search queue and update the member variables at nodes $n$ and $n'$ (Lines 16-19). At node $n$, we update the reconfiguration path corresponding to adjacent node $n'$, while at node $n'$, we change the manipulator configuration with the last configuration $q^m_{new}$ in the reconfiguration path and also update the new path and the corresponding cost. If the base component of popped out node $(n[q^b])$ from search queue is the base goal node $(n_g[q^b])$, then simply a manipulator reconfiguration path $\pi^m_{n_g}$ is searched from achieved manipulator configuration $\hat{q}^m_g$ to the desired manipulator configuration $q^m_g$ at base goal pose (Lines 7-11). The final path is computed using $n_g[p]$ and $n_i[\text{R}_{\text{PATHS}}{}^{[n_{adj}]}]$ such that $n_i, n_{adj} \in n_g[p]$.

If the search mechanism (stage 2) fails to find a path in the base roadmap constructed during stage 1, then we go back to stage 1 to further expand the base roadmap and the process repeats. The base roadmap expansion step is carried out from Lines 20-23 by invoking a routine ExpandBaseRoadmap() which randomly samples additional base poses and adds them to the base roadmap, as well as expands the base nodes in narrow regions. We have implemented the random-bounce walks strategy of standard PRM [13]. One could also use other strategies [49]. Please note that this expansion of the base roadmap (possibly repeated multiple times), allows HAMP to deal with narrow passages as well (theoretically, we show HAMP is probabilistically complete). Let's say we pop out a node in the base roadmap near the entrance to a narrow passage. It is not possible to get into the narrow passage without reconfiguring the arm. However, we are so close to the entrance that the arm can not be reconfigured into the proper configuration for entry (it would hit the walls). In this case, HAMP will pop out the next node from the search queue and will try to enter the narrow passage through different base paths. For instance, it might try a base node away from the entrance, reconfigure the arm such at this node, that might allow it to enter the narrow passage (e.g., see Scenario E, Figure 2.7 (c) in Section 2.6).

Now we explain the SearchManipPath() routine which works as follows: it first checks if the manipulator configuration at base node $n$ is collision-free along the edge formed with adjacent base node $n'$ (Lines 1-3, Algorithm 3). If it is, then the returned manipulator path is that single configuration. This corresponds to the small purple ellipses with one white dot in Fig 2.1, which indicates that the manipulator configuration, corresponding to the white dot, is collision free along the edge and no manipulator planning was required. Otherwise, a set of manipulator configurations $(\text{goal}^m)$[2] are randomly sampled at base node $n$ using a routine ComputeArmGoals(), described in Algorithm 4, such that each of the configuration in the $\text{goal}^m$ is collision-free along the edge formed with an adjacent base node $n'$. The manipulator planning is carried out at base node $n$, by constructing an arm roadmap using an incremental PRM and a manipulator path is searched from manipulator configuration at base node $n$ to any of the goal configurations in $\text{goal}^m$. To make the distinction between roadmap nodes in $C_b$ and $C_m$ , we use superscript $m$ for the arm roadmap nodes in $C_m$.

We can divide the failures to solve the overall problem within permitted time in three types:

a) Type 1 - ConstructBaseRoadmap() fails to connect the start and goal configurations of the mobile manipulator with manipulator in home configuration for the entire base roadmap. This implies that the algorithm failed even before starting the path search phase as the start and goal are not in the same connected components.

[2]We are using multiple possible goal configurations because empirically it was faster than searching for a single goal configuration. Most likely, it is because the likelihood of multiple goal configurations being difficult to reach would be significantly lower.

b) Type 2 - SearchManipPath() fails to search for a manipulator path, mainly because, either ComputeArmGoals() fails to find manipulator goal configurations within permitted time or SearchManipPath() fails to connect the start manipulator configuration at node $n$ to any of the goal configurations reported by routine ComputeArmGoals() within ArmPlanningTimeUp. This failure implies that the algorithm failed in the middle of the path search phase because the algorithm could not reconfigure the arm in a very narrow or cluttered region.

c) Type 3 - path search reaches the goal base node in the base roadmap but routine SearchManipPathAtBaseGoal() fails to compute a path from achieved manipulator configuration to the desired one. This failure implies that the algorithm failed at the end of the path search phase.

## 2.4 Probabilistic completeness proof

This section deals with probabilistic completeness proof of HAMP. Suppose $q_s$, $q_g \in C_{bm_{free}}$ are two mobile manipulator configurations that can be connected by an H-path in $C_{bm_{free}}$. HAMP is considered to be probabilistic complete, if for any given $(q_s, q_g)$

$$\lim_{\substack{n\to\infty \\ m\to\infty}} Pr[(q_s, q_g)FAILURE] = 0 \tag{2.1}$$

where $Pr[(q_s, q_g)FAILURE]$ denotes the probability that HAMP fails to answer the query $(q_s, q_g)$ after a base roadmap in $C_{b_{free}}$ with $n$ samples and manipulator roadmaps each with $m$ samples in $C_{m_{free}}$ have been constructed. The outline of the probabilistic completeness proof is as follows: First we assume that an H-path $\Pi^{bm}$ from $q_s$ to $q_g$ exists - recall that an H-path consists of a sequence of sub-paths where the base moves with fixed manipulator configuration (base path segments), followed by a reconfiguration step where base is fixed but manipulator moves to a different configuration (re-configuration path). We then tile the base path as well as all the manipulator paths with a set of carefully chosen balls such that generating a sample in each ball ensures that these samples can be connected with appropriate collision-free edges and hence a collision-free H-path, $\hat{\Pi}^{bm}$ between $q_s$ and $q_g$ will be found by HAMP and the probability of generating such samples approaches 1 with increasing $m$ and $n$.

Assume an H-path $\Pi^{bm}$ from $q_s$ to $q_g$ with $k$ (finite but can be arbitrarily large) manipulator reconfiguration steps exists - the path is composed of $k$ base path segments as shown in Fig 2.3. For each base path segment $\pi_i^b$, the mobile manipulator moves with a fixed manipulator configuration; at the end of the segment, denoted by base pose $q_{r_i}^b$, a manipulator reconfiguration step is executed (with base stationary), and the mobile manipulator now moves along the next base path segment $\pi_{i+1}^b$ with the new fixed manipulator configuration. Let $Q^b = \{q_{r_i}^b\}$ denote the set of base poses along $\pi^b$ at which reconfiguration steps take

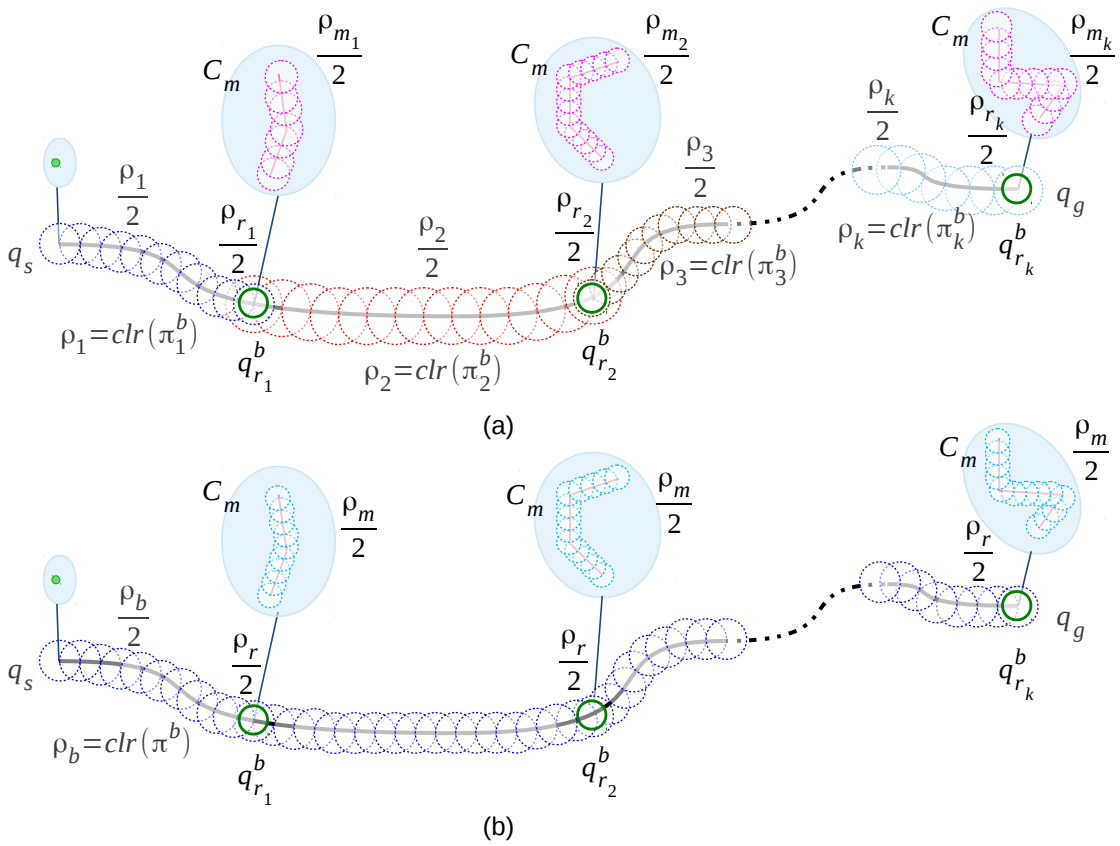



Figure 2.3: A schematic illustrating a known mobile manipulator path and the tiling with carefully chosen balls. (a) clearance is not uniform through out the base path and for all the manipulator paths, for e.g., for a clearance of $\rho_i$ the corresponding base path segment is tiled with balls each of radius $\frac{\rho_i}{2}$; (b) clearance is the minimum of all balls in (a), we tile the base path $\pi_b$ with balls each of radius $\frac{\rho_b}{2}$, base poses $q^b_{r_i} \in Q^b$ with balls of radius $\frac{\rho_r}{2}$, and the manipulator reconfiguration paths with balls of radius $\frac{\rho_m}{2}$.

place. Let the length of the entire base path be $L^b$ and the length of manipulator path for $i^{th}$ reconfiguration step be $L^m_{r_i}$. Lastly, let $d_b$, $d_m$ denote the dimensions of $C_b$ and $C_m$, respectively.

We now define three clearances. The clearance of $\pi^b_i$, denoted $\rho_i = clr(\pi^b_i)$, is the farthest distance (in $C_b$) away from the path segment at which a given base pose with manipulator in the fixed configuration (provided by $\Pi^{bm}$) can be guaranteed to be collision-free. If $\pi^b_i$ lies in $C_{b_{free}}$, then $clr(\pi^b_i) > 0$. Let $\rho_b = clr(\pi^b) = min_i(\rho_i)$ be the clearance along the entire base path $\pi^b$. The clearance of the base pose $q^b_{r_i} \in Q^b$, denoted $\rho_{r_i} = clr(q^b_{r_i})$, is the farthest distance (in $C_b$) from $q^b_{r_i}$ at which the manipulator reconfiguration path (provided by $\Pi^{bm}$) can be executed collision-free. Again, let $\rho_r = min_i(\rho_{r_i})$. The clearance of a manipulator path corresponding to a base pose $q^b_{r_i}$ (tiled with a ball of radius $\frac{\rho_{r_i}}{2}$), denoted $\rho_{m_i}$, is defined (in $C_m$) as the minimum of all the clearances that can be obtained for a manipulator path corresponding to any sampled base pose from the ball. Let $\rho_m = min_i(\rho_{m_i})$.

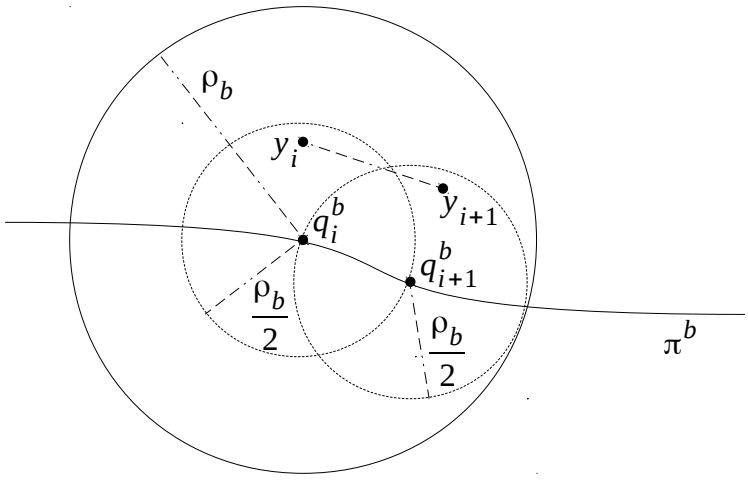



Figure 2.4: A closer look at base path tiling with balls of radius $\frac{\rho_b}{2}$. Points $y_i$ and $y_{i+1}$ are inside the balls and therefore, the line segment $\overline{y_i y_{i+1}}$ must lie inside $C_{b_{free}}$ since both endpoints lie in the ball $B_{\rho_b}(q_i^b)$.

The measure $\mu$ denotes the volume of a region of space, e.g., $\mu(B_\epsilon(x))$ measures the volume of an open ball $B_\epsilon(x)$ of radius $\epsilon$ centered at $x$. If $A \subset C_{b_{free}}$ is a measurable subset and $x$ is a random point chosen from $C_{b_{free}}$ by sampling strategy of standard PRM, then

$$Pr(x \in A) = \frac{\mu(A)}{\mu(C_{b_{free}})} \tag{2.2}$$

We now tile the base path $\pi_b$ with balls each of radius $\frac{\rho_b}{2}$, base poses $q_{r_i}^b \in Q^b$ with balls of radius $\frac{\rho_r}{2}$ (green circles), and the manipulator reconfiguration paths with balls of radius $\frac{\rho_m}{2}$ as shown in Fig 2.3 (b). Let $p^b = \lceil \frac{2L^b}{\rho_b} \rceil$ and observe that there are $p^b$ points (centers of balls) on the path $\pi^b$ such that $dist^b(q_i^b, q_{i+1}^b) < \rho_b$, where $dist^b$ is a Euclidean metric on $\mathbb{R}^{\mathrm{d_b}}$. Out of these $p^b$ points, there are $k$ points ($Q^b$) where a manipulator reconfiguration step is needed. Let $\neg Q^b$ denote the set of remaining $(p^b - k)$ points along $\pi^b$. Let $y_i \in B_{\rho_b/2}(q_i^b)$ and $y_{i+1} \in B_{\rho_b/2}(q_{i+1}^b)$. Then the line segment $\overline{y_i y_{i+1}}$ must lie inside $C_{b_{free}}$ since both endpoints lie in the ball $B_{\rho_b}(q_i^b)$ as shown in Fig 2.4. Let $p_{r_i}^m = \lceil \frac{2L_{r_i}^m}{\rho_m} \rceil$ and observe that there are $p_{r_i}^m$ points on the $i^{th}$ manipulator reconfiguration path such that $dist^m(q_i^m, q_{i+1}^m) < \rho_m$, where $dist^m$ is a Euclidean metric on $\mathbb{R}^{\mathrm{d_m}}$.

Let $V^b \subset C_{b_{free}}$ be a set of $n$ base poses generated uniformly at random by HAMP for the construction of base roadmap. Similarly, $V_{r_i}^m \subset C_{m_{free}}$ be a set of $m$ manipulator configurations generated uniformly at random by HAMP for the construction of $i^{th}$ manipulator roadmap. If the following conditions hold then an H-path from $q_s$ to $q_g$ will be found: (a) each ball along the base path $\pi^b$ corresponding to $q_i^b \in \neg Q^b$ gets atleast one sample, i.e., there is a subset $\{y_i\} \subset V^b$ of $(p^b - k)$ base poses such that $y_i \in B_{\rho_b/2}(q_i^b)$; (b) remaining balls along $\pi^b$ corresponding to $q_{r_i}^b \in Q^b$ get atleast one sample each, i.e., subset

$\{y'_i\} \subset V^b$ of $k$ base poses such that $y'_i \in B_{\rho_r/2}(q^b_{r_i})$; (c) for all the manipulator paths, the corresponding balls also get atleast one sample each, i.e., there is a subset $\{y^r_i\} \subset V^m_{r_i}$ of $p^m_{r_i}$ manipulator configurations such that $y^r_i \in B_{\rho_m/2}(q^m_i)$. As mentioned earlier, these conditions ensure at least one sample in each ball such that these samples can be connected with appropriate collision-free edges and hence a collision-free H-path will be found by HAMP.

To formalize it mathematically, let $I_1, ..., I_{p^b}$ be a set of indicator variables such that each $I_j$ (excluding those $k$ indicators where manipulator reconfiguration is needed) witness the event that there is a $y \in V^b$ and $y \in B_{\rho_b/2}(q^b_i)$ while remaining $k$ indicator variables witness the event that there is a $y' \in V^b$ and $y' \in B_{\rho_r/2}(q^b_{r_i})$. Let $I_{r_1}, ..., I_{r_k}$ be a set of indicator variables such that each $I_{r_i}$ witness the event that all balls along the $i^{th}$ manipulator reconfiguration path get at least one sample each. Let $I^m_1, ..., I^m_{p^m_{r_i}}$ be a set of indicator variables for each $I_{r_i}$ such that $I_{r_i} = I^m_1 \wedge I^m_2 \wedge, ....., \wedge I^m_{p^m_{r_i}}$ and each $I^m_t$ witness the event that there is a $y^r \in V^m_{r_i}$ and $y^r \in B_{\rho_m/2}(q^m_i)$. It follows that HAMP succeeds in answering the query $(q_s, q_g)$ if $I_j = 1$ for all $1 \leq j \leq p^b$ and $I_{r_i} = 1$ for all $1 \leq i \leq k$. If at least one of the indicator variables $(I_j, I_{r_i})$ is 0 then HAMP would fail. Therefore, the probability of failure (Equation 2.1) then can be written as

$$Pr[(q_s, q_g)FAILURE] \leq Pr((\bigvee_{j=1}^{p^b} I_j = 0) \vee (\bigvee_{i=1}^{k} I_{r_i} = 0)) \tag{2.3}$$

$$\leq \sum_{j=1}^{p^b} Pr[I_j = 0] + \sum_{i=1}^{k} Pr[I_{r_i} = 0] \tag{2.4}$$

where the last inequality follows from the union bound. We further breakdown the first term into two components: (i) $(p_b - k)$ balls along $\pi^b$ where manipulator reconfiguration is not needed, and (ii) remaining $k$ balls each with a radius of $\frac{\rho_r}{2}$ where reconfiguration is needed. Therefore, RHS of last inequality (Equation 2.4) can be written as

$$\leq \sum_{j \in \neg Q^b} Pr[I_j = 0] + \sum_{j \in Q^b} Pr[I_j = 0] + \sum_{i=1}^{k} Pr[I_{r_i} = 0] \tag{2.5}$$

The probability of a given $I_j = 0$ is computed by observing that none of the $n$ randomly generated independent samples falls in $B_{\rho_b/2}(q^b_j)$, therefore for $j \in \neg Q^b$,

$$Pr[I_j = 0] = \left(1 - \frac{\mu(B_{\rho_b/2}(q^b_j))}{\mu(C_{b_{free}})}\right)^n \tag{2.6}$$

However,

$$\frac{\mu(B_{\rho_b/2}(\cdot))}{\mu(C_{b_{free}})} = \frac{(\frac{\rho_b}{2})^{d_b}\mu(B_1(\cdot))}{\mu(C_{b_{free}})} = \sigma_1 \rho_b^{d_b} \tag{2.7}$$

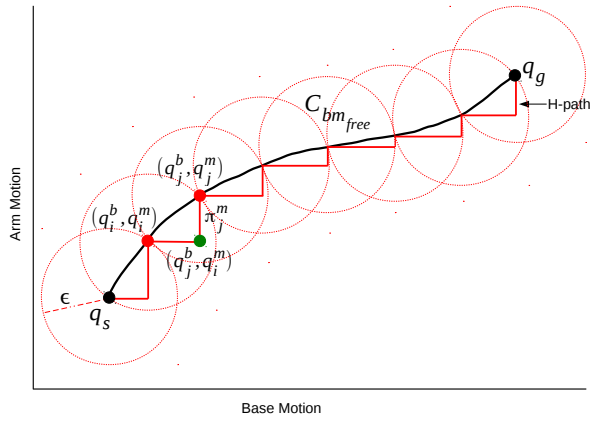



Figure 2.5: This figure shows that any path in $C_{bm_{free}}$ (assuming it is an open set) connecting the start ($q_s$) and the goal ($q_g$) configurations can be approximated by an H-path, also lying in $C_{bm_{free}}$. The black curve denotes an arbitrary path that lies in $C_{bm_{free}}$. Let $\epsilon$ be minimum clearance (from C-obstacles) along the path (it must exist since $C_{bm_{free}}$ is open). The path is then tiled with balls of radius $\epsilon$. The H-path approximation is shown in red and by construction, it is guaranteed to be collision-free.

where $B_1(\cdot)$ is the unit ball in $\mathbb{R}^{\mathrm{d_b}}$ and $\sigma_1 = \mu(B_1(\cdot))/2^{d_b}\mu(C_{b_{free}})$. Hence RHS of Equation 2.5 becomes

$$\leq (p^b - k)(1 - \sigma_1 \rho_b^{d_b})^n + k(1 - \sigma_1 \rho_r^{d_b})^n + \sum_{i=1}^{k} Pr[I_{r_i} = 0] \tag{2.8}$$

$I_{r_i} = 0$ only if $\vee_{t=1}^{p_{r_i}^m} I_t^m = 0$. Therefore, $Pr[I_{r_i} = 0]$ is

$$Pr[I_{r_i} = 0] \leq Pr[\bigvee_{t=1}^{p_{r_i}^m} I_t^m = 0] \leq \sum_{t=1}^{p_{r_i}^m} Pr[I_t^m = 0] \tag{2.9}$$

$$\sum_{t=1}^{p_{r_i}^m} Pr[I_t^m = 0] \leq p_{r_i}^m \left(1 - \frac{\mu(B_{\rho_m/2}(q_i^m))}{\mu(C_{m_{free}})}\right)^m \tag{2.10}$$

The RHS of last inequality (Equation 2.8) can now be written as

$$\leq (p^b - k)(1 - \sigma_1 \rho_b^{d_b})^n + k(1 - \sigma_1 \rho_r^{d_b})^n + \sum_{i=1}^{k} p_{r_i}^m (1 - \sigma_2 \rho_m^{d_m})^m \tag{2.11}$$

where $\sigma_2 = \mu(B_2(\cdot))/2^{d_m}\mu(C_{m_{free}})$ and $B_2(\cdot)$ is the unit ball in $\mathbb{R}^{\mathrm{d_m}}$. Using the relation $(1 - \beta)^n \leq e^{-\beta n}$ for $0 \leq \beta \leq 1$, the above inequality then finally can be written as

$$\leq c_1 e^{-\sigma_1 \rho_b^{d_b} n} + c_2 e^{-\sigma_1 \rho_r^{d_b} n} + c_3 e^{-\sigma_2 \rho_m^{d_m} m} \tag{2.12}$$

where $c_1 = (p^b - k)$, $c_2 = k$, and $c_3 = \sum_{i=1}^{k} p_{r_i}^m$. The expression above converges exponentially to 0 as $n \to \infty$ and $m \to \infty$, hence showing the completeness of the HAMP algorithm with respect to a class of paths, i.e., H-path. However, given any arbitrary path in open $C_{bm_{free}}$, we can always create an H-path that lies in open $C_{bm_{free}}$ as shown in Fig 2.5. Hence HAMP is probabilistically complete.

## 2.5 The HAMP-U Algorithm

Now we extend our HAMP algorithm to account for localization uncertainty associated with the mobile base pose. With the motion and sensor uncertainty, the state of mobile base is not precisely known and is represented by Gaussian belief. We assume that the motion of manipulator is quite accurate (a reasonable assumption, give the joint encoders are quite precise). The objective of HAMP-U is to find a collision-free path with low belief covariance at the goal. Like HAMP, the HAMP-U algorithm is a two stage process described in Algorithm 5. For brevity, we omit the expand roadmap portion in our pseudo code.

In the first stage (Line 1 of Algorithm 5), we first create a belief roadmap (instead of the standard probabilistic roadmap in the base pose space for the basic HAMP) for the mobile base with manipulator remaining in a fixed home configuration. The belief roadmap formation is explained in Algorithm 6. A naive approach to build the belief roadmap is to sample beliefs directly from belief space $(\mu, \Sigma)$. However, the biggest challenge is to ensure that the nodes are reachable. Therefore, the planner first samples a set of mean poses $\{\mu_i\}$ from $C_{b_{free}}$ using the standard sampling step in PRM algorithm [13]. In our case, a sample mean pose is collision-free if mobile manipulator with mobile base positioned at $\mu_i$ and manipulator in home configuration $q_H^m$ is collision-free. We add an edge $e_{ij}$ between pairs $(\mu_i, \mu_j)$ if a sequence of controls exists to move the mobile manipulator without collisions along the straight line between poses. We then simulate a sequence of controls and measurements along each edge. To achieve this, a motion model of the mobile base and a sensor model of the sensor is needed. The motion model, sensor model and belief estimation along edges (using one-step transfer function) are explained in Appendix A. Our notation follows that of [26]. The effect of these models are essentially given by matrices $G_t$, $V_t$, $H_t$, the Jacobians corresponding to the motion model (w.r.t state variable and w.r.t control variable) and the sensor model (w.r.t state variable); and matrices $W_t$, $Q_t$, the noise covariance matrices for motion and sensing, respectively. The matrices $G_t$, $R_t = V_t W_t V_t^T$ and $M_t = H_t^T Q_t^{-1} H_t$, for each step along the edge $e_{ij}$ are computed. BRM uses maximum likelihood observations and a covariance factorization to combine multiple updates along an edge $e_{n,n'}$ into a single transfer function, represented by a descriptor matrix $S_{1:T_{n,n'}}$. The matrix encodes the uncertainty along the edge.

The second stage (Lines 2-19 of Algorithm 5) is a search mechanism that searches the belief roadmap using a variant of standard breadth first search algorithm and SearchManipPath()

**Algorithm 5:** $\Pi^{bm} = \text{HAMP-U}(q_s, q_g, q_H^m)$

**Input**: The start $q_s = ((\mu_s, \Sigma_s), q_s^m) = (q_s^b, q_s^m)$ and goal $q_g = ((\mu_g, -), q_g^m) = (q_g^b, q_g^m)$ configurations, 3D Map and $q_H^m$

**Output**: Path $\Pi^{bm}$ from $q_s$ to $q_g$ with low goal covariance $\Sigma_g$

1 Roadmap $G_{belief}^b :=$ ConstBeliefRoadmap$(q_H^m)$

2 Append $G_{belief}^b$ with nodes $\{n_s :=(\mu_s, q_H^m)$, $n_g :=(\mu_g, q_H^m)\}$, edges $\{\{e_{s,j}\}, \{e_{i,g}\}\}$, and *one* descriptor matrices $\{\{S_{1:T_{s,j}}\}, \{S_{1:T_{i,g}}\}\}$

3 Augment node structure with best path $p := \emptyset$, covariance $\Sigma := \emptyset$, $\text{R}_{\text{PATHS}}^{[n_{adj}]} = \emptyset$ such that $n_i := \{(\mu, \Sigma), q_H^m, p, \text{R}_{\text{PATHS}}^{[\cdot]}\}$

4 $Q \leftarrow n_s := \{(\mu_s, \Sigma_s), q_s^m, \emptyset, \emptyset\}$

5 **while** $Q \neq \emptyset$ **do**

6 $\quad n :=$Pop$(Q)$

7 $\quad$ **if** $n[q^b] = n_g[q^b]$ **then**

8 $\quad\quad$ Continue

9 $\quad$ **for** ***all*** $n'$ *of* $adj[n]$ ***and*** $n' \notin n[p]$ **do**

10 $\quad\quad$ Compute one-step update $\Psi' = \Psi \star S_{1:T_{n,n'}}$ where $\Psi = \begin{bmatrix} I & n[\Sigma] \\ 0 & I \end{bmatrix}$

11 $\quad\quad \Sigma' \leftarrow \Psi'_{12}$

12 $\quad\quad$ **if** $tr(\Sigma') < tr(n'[\Sigma])$ ***or*** $tr(n'[\Sigma]) = \phi$ **then**

13 $\quad\quad\quad (q_{new}^m, \pi_n^m) :=$SearchManipPath$(n, n')$

14 $\quad\quad\quad$ **if** $\pi_n^m \neq \emptyset$ **then**

15 $\quad\quad\quad\quad n[\text{R}_{\text{PATHS}}^{[n']}] := \pi_n^m$

16 $\quad\quad\quad\quad n' := \{(-, \Sigma'), q_{new}^m, n[p] \cup \{n'\}, -\}$

17 $\quad\quad\quad\quad Q \leftarrow Q \cup \{n'\}$

18 $\pi_{n_g}^m =$SearchManipPathAtBaseGoal$(\hat{q}_g^m, q_g^m)$

19 $\Pi^{bm} \leftarrow$ traceback using $n_g[p]$ and $n_i[\text{R}_{\text{PATHS}}^{[n_{adj}]}]$ such that $n_i, n_{adj} \in n_g[p]$

20 **return** $\Pi^{bm}$

**Algorithm 6:** $G_{belief}^b =$ConstBeliefRoadmap$(q_H^m)$

**Input**: $q_H^m$

**Output**: Belief roadmap $G_{belief}^b$

1 Sample mean poses $\{\mu_i\}$ from $C_{b_{free}}$ using a standard PRM sampling strategy to build roadmap node set $\{n_i\}$ such that $n_i = q_i = (\mu_i, q_H^m) = (q_i^b, q_H^m)$

2 Create edge set $\{e_{ij}\}$ between nodes $(n_i, n_j)$ if CollisionFree$(n_i, n_j)$

3 Build *one* descriptor matrices $\{S_{1:T}\} \forall e_{ij} \in \{e_{ij}\}$

4 **return** $G_{belief}^b = \{\{n_i\}, \{e_{ij}\}, \{S_{1:T_{ij}}\}\}$

routine in an intertwined manner, similar to HAMP except that the metric used is uncertainty along the path rather than the length of path. The search process uses a queue function for the expansion of $(\mu, \Sigma)$ nodes in a first-in, first-out order. Line 9 prevents cycling problems, where an adjacent node $n'$ is only considered if it is not already in the

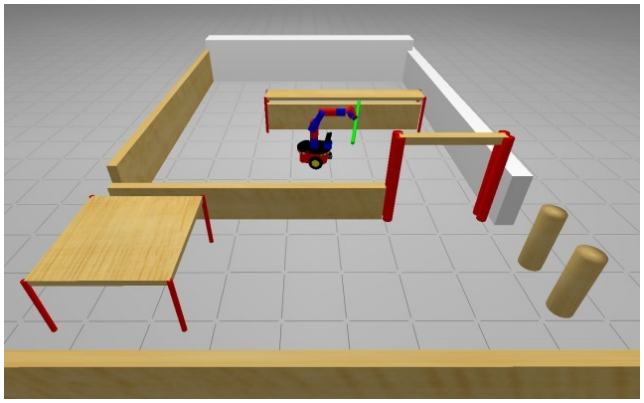


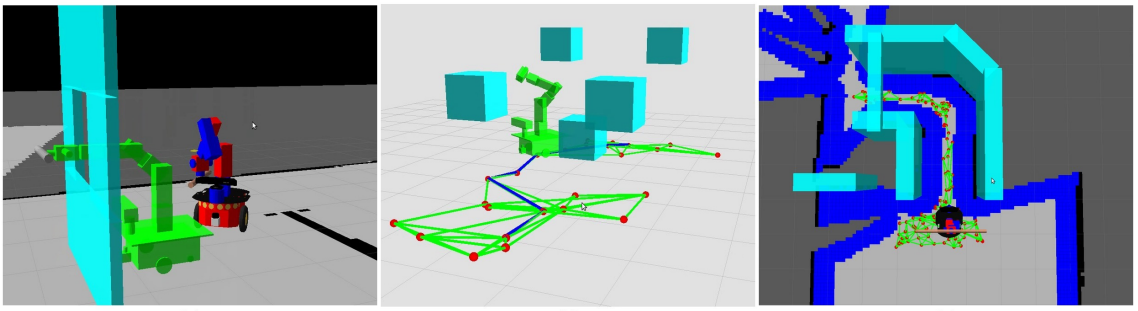
Figure 2.6: This example (simulation environment for scenarios A and B) shows the mobile manipulator carrying a long payload (120 cm long stick) and it needs to pass through a doorway to reach the goal on the other side.

mobile base path $n[p]$. Note that, in Lines 12-17, we only expand nodes if search algorithm has found a new posterior covariance $\Sigma'$ such that some measure of uncertainty (we use the trace) for it is less than that for existing posterior covariance $n'[\Sigma]$ and there is a path for the mobile manipulator along an edge $e_{n,n'}$. It is also assumed that a node $n'$ replaces any current queue member $n'$ when pushed onto the queue in line 17.

## 2.6 Results

We performed a series of simulations and real experiments on the SFU mobile manipulator roaming around in our lab to evaluate HAMP and HAMP-U. Our evaluation consisted of two main objectives: (a) to demonstrate the usefulness of the hierarchical search (HAMP) and its comparison with a full 9D PRM based on a set of performance criteria; (b) to show that by taking into account the uncertainty in the planning process, i.e., using HAMP-U, the paths generated are less likely to result in collision as compared to the basic HAMP.

The SFU mobile manipulator consists of a powerbot mobile base with a 6DOF Schunk powercube arm mounted on it. The world representation is computed offline by manually moving the robot around and using the two on-board sensors, an LMS100, a planar laser rangefinder that provides a 240° field-of-view at 30 Hz and an effective range of 18m; and a Kinect that provides 3D range data at 30 Hz and an effective range of 0.7-6m. In simulations also, we used the same mobile manipulator model with the corresponding sensors. We run our tests under linux (Ubuntu 10.10) on a Pentium dual core 2.5 Ghz computer with 4GB memory. We made use of portions of publicly available ROS [68] and OMPL [69] code as needed for implementation of our HAMP and HAMP-U algorithms.



(a) (b) (c)

Figure 2.7: Simulation environments for scenarios C, D & E: (a) in this simulation the task is to pass a stick of 50 cm through a window of size 40cm×50cm, this figure shows the mobile manipulator with stick in start and goal configurations; (b) the mobile manipulator needs to navigate through 5 cuboid obstacles to reach the goal; (c) shows a narrow corridor (overhead view) of width 80cm with a tight round turn; the manipulator is required to move in the narrow corridor and negotiate the turn while carrying a 100 cm long stick.

Table 2.1: Experimental results in 5 scenarios for mobile manipulator planning (HAMP vs. FULL 9D PRM).

| | Permitted Time (s) | Planner | # Base Nodes | | Total Arm Nodes | | Time (s) | | # Coll. Checks | | #Reconfig /#Armchks. | B. Path Len. (m) | #Succ. /#Runs |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | mean | s. d. | mean | s. d. | mean | s. d. | mean | s. d. | | | |
| A | 40 | HAMP | 87 | 108 | 63 | 104 | 5.7 | 6.1 | 23k | 25k | 4/95 | 4.5 | 30/30 |
| | | PRM | 489 | 326 | N/A | N/A | 12.5 | 10.7 | 80k | 52k | N/A | 6.1 | 24/30 |
| B | 40 | HAMP | 115 | 152 | 180 | 269 | 12.9 | 11.2 | 39k | 36k | 6/116 | 4.6 | 29/30 |
| | | PRM | 823 | 267 | N/A | N/A | 19.5 | 14.0 | 130k | 41k | N/A | 7.8 | 7/30 |
| C | 120 | HAMP | 2 | 1 | 1958 | 920 | 36.9 | 19.4 | 173k | 81k | 1/1 | 0.8 | 30/30 |
| | | PRM | 1638 | 265 | N/A | N/A | 84.2 | 21.3 | 297k | 97k | N/A | 4.9 | 12/30 |
| D | 70 | HAMP | 29 | 10 | 736 | 796 | 27.8 | 14.6 | 114k | 76k | 15/34 | 3.2 | 28/30 |
| | | PRM | 945 | 491 | N/A | N/A | 38.2 | 18.2 | 192k | 82k | N/A | 4.3 | 21/30 |
| E | 300 | HAMP | 96 | 76 | 2897 | 1874 | 77.8 | 42.5 | 205k | 134k | 80/278 | 3.0 | 29/30 |
| | | PRM | 10154 | 2469 | N/A | N/A | 193.5 | 48.9 | 786k | 187k | N/A | 4.7 | 23/30 |

### 2.6.1 World Representation and Collision Checks

We use two types of map representation for collision check for efficiency reasons: a 2D costmap (inflates costs based on 2D occupancy grid and user specified inflation radius) and a 3D collision map (a set of occupied voxels) derived from an incrementally built global octree [34]. For efficiency reasons, collision detection for the whole mobile manipulator is accomplished in a two-stage process as follows. During initial construction of the roadmap (in routines ConstructBaseRoadmap() as well as ConstBeliefRoadmap()), the 2D projected footprint of the base is checked against the 2D costmap, and if it is collision-free then a 3D collision check is performed on the manipulator. We use height threshold to project 3D range data (from Kinect) to get a 2D costmap. During search (routines ComputeArmGoals() and SearchManipPath()), since the path is already collison-free with respect to the base and home configuration of the arm, only 3D collision checks are done for the arm. This strategy helps us to avoid unnecessary 3D collision checks (which can be expensive) without being overtly conservative.

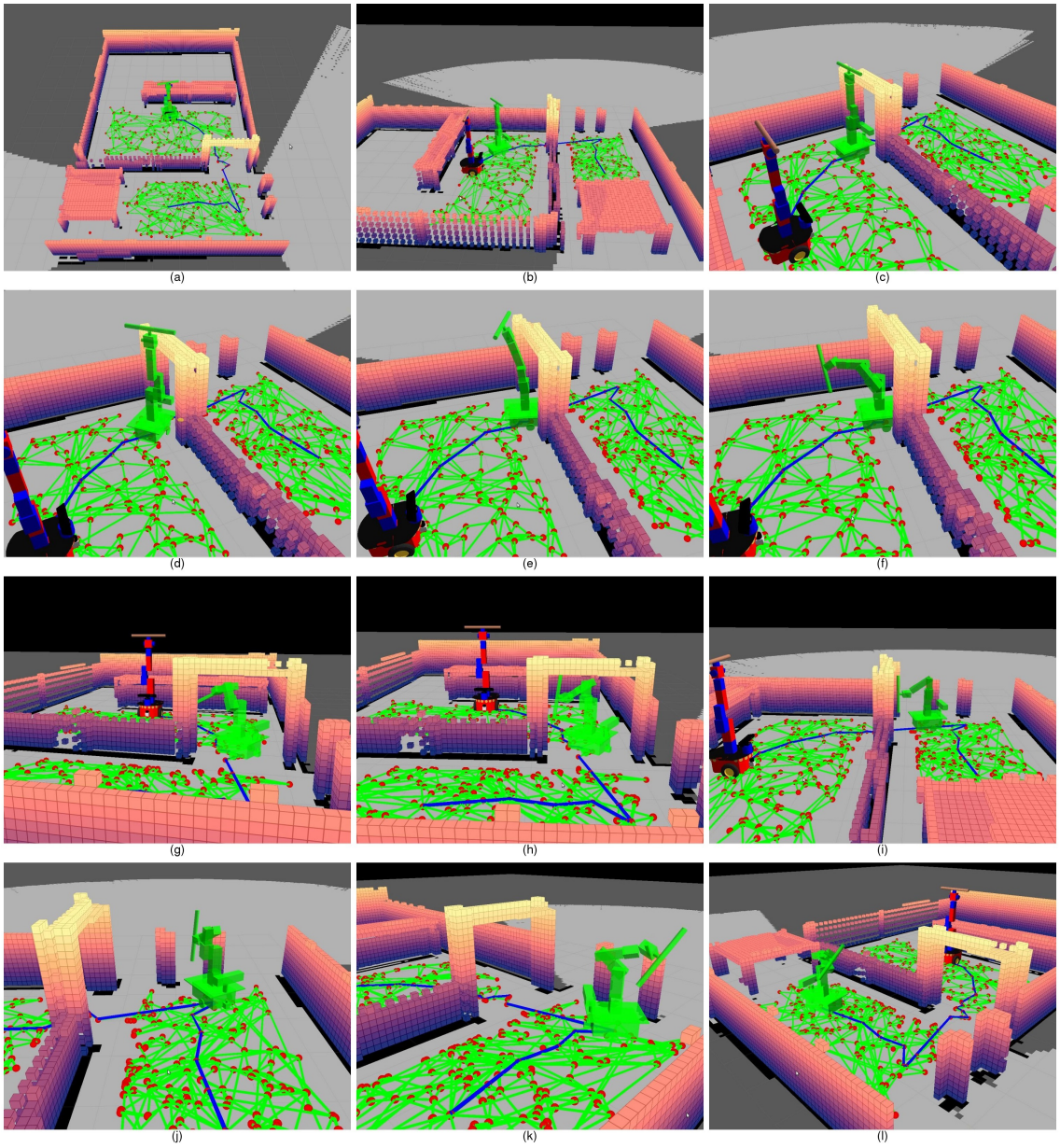



Figure 2.8: HAMP simulation test for scenario B: This simulation shows the mobile manipulator carrying a 50 cm stick as payload and navigating from one side of the door to the other side. The corresponding base roadmap and base path (blue color) of H-path are shown as well. (a) mobile manipulator's start configuration; (b), (c), (d) show a sequence of snapshots of mobile manipulator motions along the path with the same arm configuration; (e), (f) show the first reconfiguration step, in order to move along the path, arm configuration needs to be changed, as mobile manipulator can not cross the doorway due to arm and long payload collision with gate; (g), (h) show the second arm reconfiguration step; (i) the mobile manipulator advances along the path with arm in final configuration of the last reconfiguration step; (j), (k) show the third reconfiguration step; (l) mobile base has reached the goal but not the arm, this snapshot shows the mobile manipulator before the final reconfiguration step at goal.

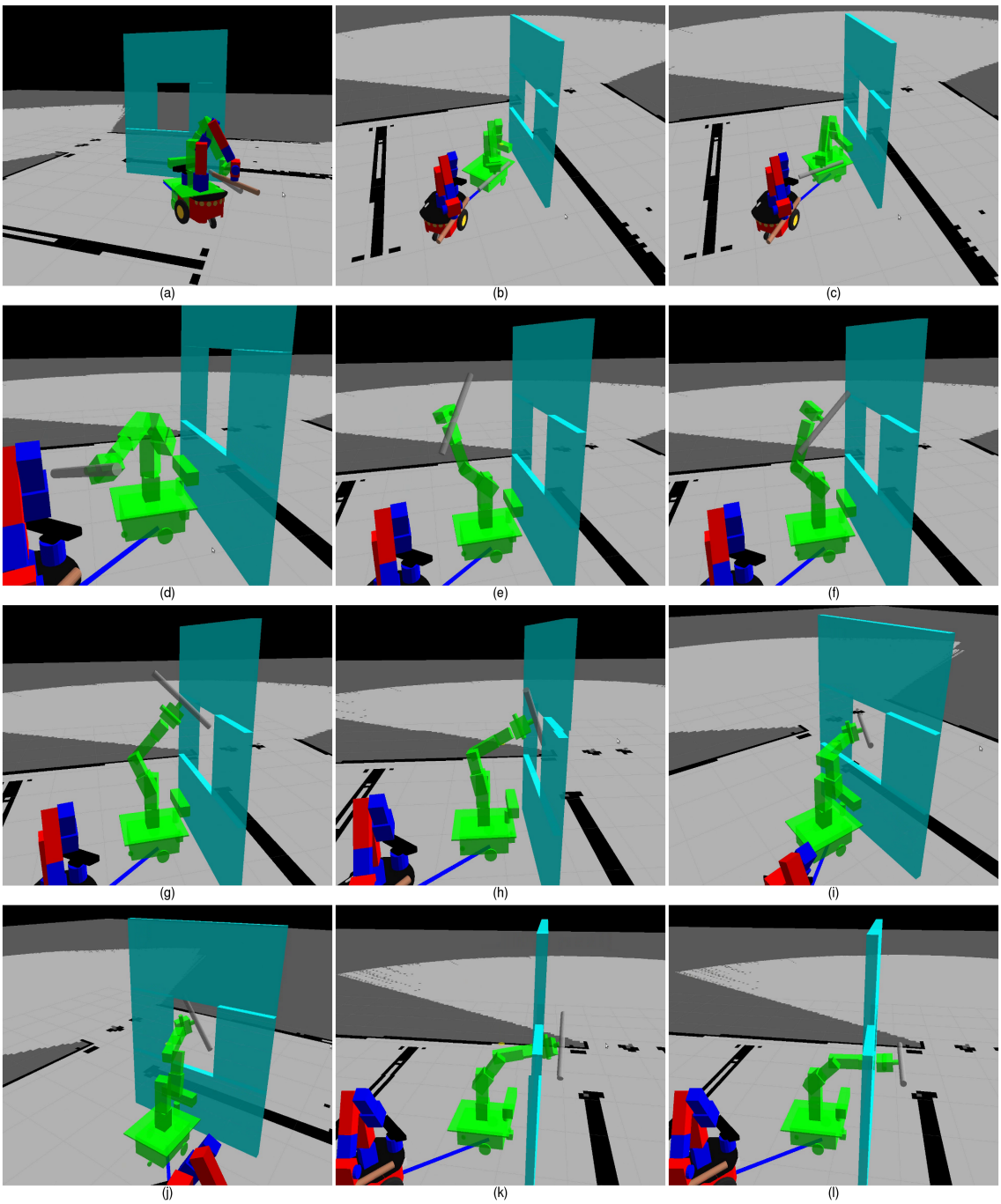



Figure 2.9: HAMP simulation test for scenario C: This simulation shows the mobile manipulator passing a 50 cm stick through a window of 40cm×50cm (a) mobile manipulator's start configuration, here the arm is in compact state (folded) and the payload is held parallel along the side of the robot; (b), (c) mobile manipulator with the same arm configuration; (d), (e), (f), (g), (h), (i), (j), (k) show snapshots for arm reconfiguration step at goal, (l) shows the mobile manipulator in goal configuration.

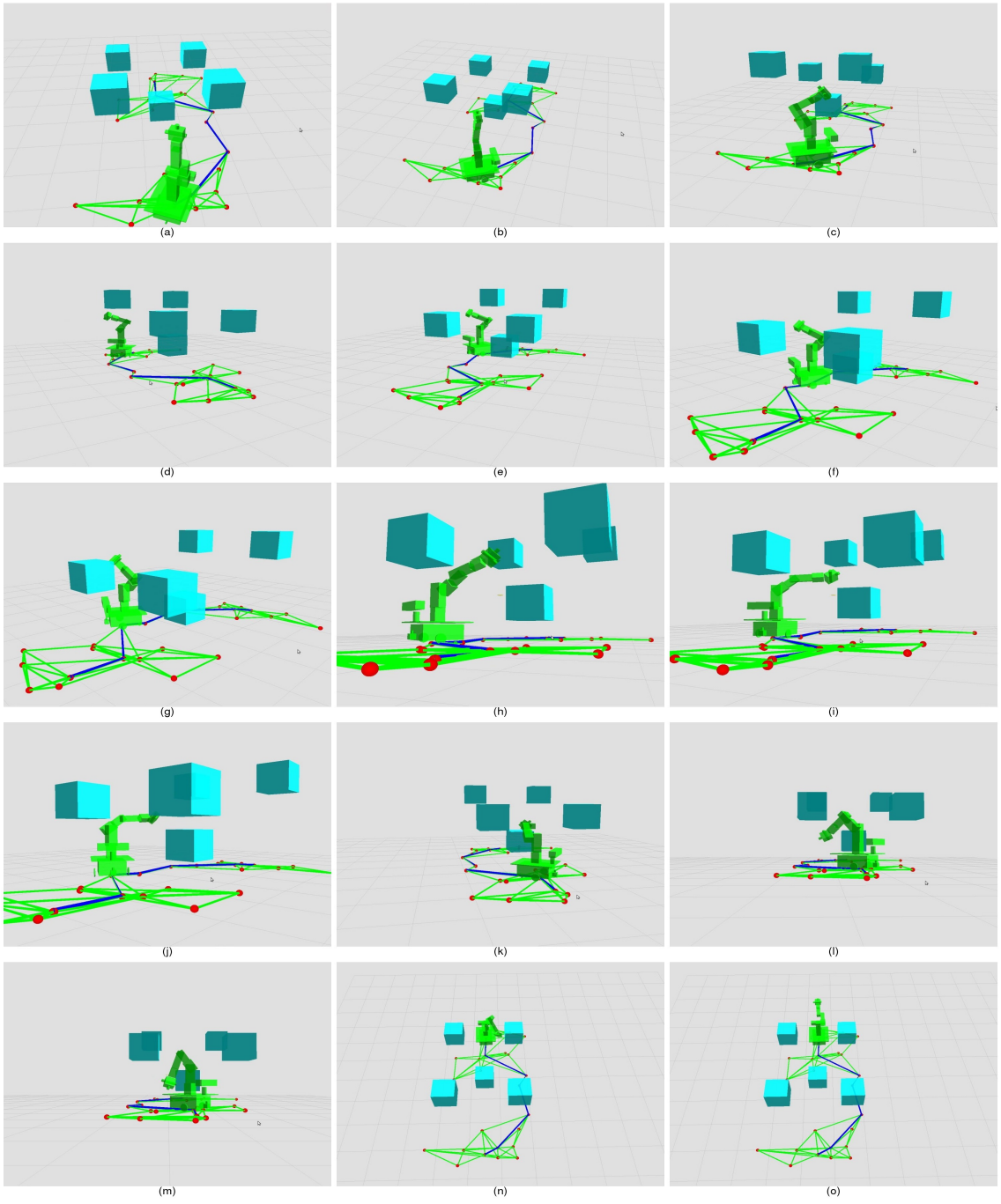



Figure 2.10: HAMP simulation test for scenario D: This simulation shows the mobile manipulator navigating through 5 randomly placed cuboid obstacles (a) mobile manipulator's start configuration, here the arm is in vertically extended configuration; (b), (c) show the first reconfiguration step; (d), (e), (f), (g) mobile manipulator advances with the same arm configuration; (h), (i) show the second reconfiguration step; (j), (k) mobile manipulator advances with the same arm configuration; (l), (m) show the third reconfiguration step, (n) mobile base has reached the goal but not the arm, (o) shows the mobile manipulator's goal configuration, after the final reconfiguration step.

### 2.6.2 Simulation Results for HAMP

We ran HAMP and a full 9D PRM on 5 different scenarios (A to E) with varying levels of complexity in simulation and compared the outcomes. We used OMPL [69] to implement full 9D PRM in an incremental way (single-query) along with random-bounce walk expansion strategy [13] to connect narrow passages. The key parameters we used in HAMP are as follows: $K_{Goals} = 3$, ArmGoalsTimeUp = 2 seconds, and ArmPlanningTimeUp = 6 seconds. For HAMP and full 9D PRM, we used 5 nearest neighbours to connect the new sample to the neighbouring nodes. Simulation environment corresponding to scenarios A and B is shown in Fig 2.6, while for scenarios C, D and E, the corresponding simulation environments are shown in Fig 2.7. The size of mobile base is 63 cm in width and 83 cm in length, while the manipulator can fully stretch to 76 cm. In scenarios A and B, the mobile manipulator with arm in vertically extended configuration was required to navigate from one side of the door (of width 70 cm) to the other side. In A, the manipulator has no payload, whereas in B, we increased the difficulty of task by adding a payload - a stick of 50cm length to the manipulator. In C, the task required passing a 50 cm long stick through a window of 40cm×50cm. D is a different type of environment where the mobile manipulator needed to navigate through five cuboid obstacles (each of size 40 cm and roughly placed 50 cm apart in x-y and 40 cm apart in z direction) in order to reach the goal. In scenario E (we feel the most difficult task among the five scenarios), the mobile manipulator carrying a stick of 100 cm, enters from an open area into a very narrow corridor of width 80 cm, navigates through the corridor and makes a turn through a very tight round corner and then finally exits into an open area, requiring frequent reconfiguration steps along the way.

Figures 2.8, 2.9 and 2.10 show snapshots of the simulation tests for scenarios B, C and D, respectively. In simulation tests corresponding to Fig 2.8 and Fig 2.10, the manipulator changes its configuration 3 times along the path and 1 time at the goal, while in Fig 2.9, the manipulator changes its configuration only at the goal. Videos corresponding to the simulation tests for scenarios A to E are available online at `http://www.sfu.ca/~vpilania/research.html` and also attached to this thesis (Multimedia Resources 1 to 4).

We compared HAMP and full 9D PRM on the basis of three criteria: (a) planner run time, (b) percentage of successful attempts, (c) base path length, and the results are presented in Table 2.1. We observe from the table that for each of the scenarios A to E, full 9D PRM is outperformed by HAMP. HAMP solved the problems in less time (and less number of collision checks) with a higher success rate and shorter base path lengths as compared to full 9D PRM paths.

HAMP's failures for the scenarios B, D, E are of Type 2. In this case, the base roadmap gets denser and denser in an attempt to connect the narrow regions, and hence requires large number of calls to SearchManipPath(). These failures happened because permitted time was over before the completion of calls to SearchManipPath() for all the edges in the

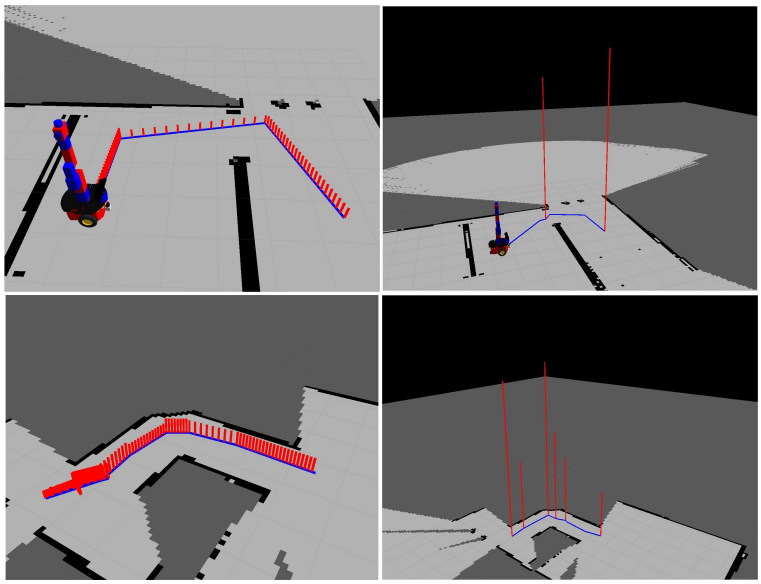


Figure 2.11: This figure shows a comparison of manipulator motion (the height of red lines corresponds to the length of arm motion (in $C_m$)) along the base path (blue line) between full 9D PRM (left) and HAMP (right) for scenarios B (top) and E (bottom). Please see text for explanation.

base roadmap. In E, sometimes (11 out of 30 runs) HAMP failed to find an H-path in the first iteration (lines 1-25, Algorithm 1) even though the start and goal base poses were in the same connected component of the base roadmap. This is mainly because of the failure of manipulator planning (Type 2 and 3 failures). In that case, HAMP expands the base roadmap by adding more samples (lines 26-31, Algorithm 1) and successfully found the path in the subsequent iterations. On average, HAMP took 3 "expand roadmap" iterations to find a path for scenario E. In practice, it also attests to probabilistic completeness of HAMP, i.e., if it can not find a path in the first iteration then it adds more samples and repeats the search mechanism until a path is find or the permitted time to solve the problem is over. In addition, it is noteworthy that manipulator reconfiguration ("#Reconfig") is needed rather infrequently as compared to the number of times manipulator configurations were checked for collisions ("#Armchks") along the edges in the base roadmap as illustrated in the column "#Reconfig/#Armchks" in Table 2.1. This supports our claim that HAMP moves the manipulator on a "need to" basis. Indeed the ratio is higher for more cluttered scenarios, but even for E, it is less than 30% (80/278).

Lastly, we illustrate the undesired motion of the arm in full 9D PRM vis a vis HAMP via a graphical representation, as shown in Fig 2.11, It shows a typical manipulator motion for HAMP and for full 9D PRM for scenarios B and E. Note that these motions were generated after a postprocessing step (path shortening) [70] . The figure shows the manipulator motion essentially as a histogram (red vertical lines) along the base path, shown in blue. For full 9D PRM (left figures), the height of each red line denotes the length of arm motion (in $C_m$)

between two consecutive poses along the discretized base path. For HAMP, the manipulator motion takes place at fixed base configurations, hence the length is zero except at some base poses, where the height of red line denotes the length of the manipulator re-configuration path (right figures). It clearly supports our claim that HAMP avoids undesired arm motions as the base moves along a path which is not the case with full 9D PRM.

### 2.6.3 Simulation Results for Tree Versions of HAMP

We also evaluated the tree versions of HAMP with RRT and Bi-directional RRT (BiRRT) as the core sub-planners for searching for both the base and the manipulator, respectively called HAMP-RRT and HAMP-BiRRT. In tree versions, it is not necessary to construct the entire tree first, hence stage 1 and 2 are essentially merged, i.e., for every new potential node to be added to the tree, SearchManipPath() is invoked from the nearest node. If it returns success then the new node is added to the tree along with corresponding updates at that node, else the node is rejected.

We compared HAMP-RRT with full 9D RRT and HAMP-BiRRT with full 9D BiRRT for scenarios A to E (as mentioned in Sec 2.6.2) and the corresponding results are presented in Tables 2.2 and 2.3. From the tables, we observe that for each of the scenarios A to E, full 9D RRT and full 9D BiRRT are outperformed by the corresponding tree versions of HAMP. Although the planning time difference is not that significant between HAMP-BiRRT and full 9D BiRRT. Similar to full 9D PRM, the computed path for the mobile manipulator using full 9D RRT or full 9D BiRRT also result into undesired and excessive motions for the manipulator even after applying a post processing smoothing filter. It is also important to note that as compared to HAMP, HAMP-RRT takes less time to plan a path and the planning time is further reduced in HAMP-BiRRT. This also holds true in case of full 9D PRM, full 9D RRT and full 9D BiRRT. In scenario C, full 9D RRT was only able to find a path in 8 trials out of 30. This is because there the goal was located in a very narrow region. In such cases BiRRT helps to find a path quickly as evident from the Table 2.3. Clearly, for the case with no uncertainty, HAMP-BiRRT is the planner of choice, because not only it is fast, it also avoids unnecessary motions of the arm.

However, for planning with uncertainty, tree versions of HAMP are not suitable, because there is no optimization possible with respect to uncertainty (note that RRT gives a unique path). One could get multiple paths via multiple runs of HAMP-RRT, however, this is likely to be computationally more expensive than the PRM version of HAMP [27]. HAMP-BiRRT is, of course, not applicable because the uncertainty at the goal is not known in advance (it in fact depends on the path taken to the goal), hence one can not grow a tree from the goal, as needed in HAMP-BiRRT.

Table 2.2: Experimental results in 5 scenarios for mobile manipulator planning (HAMP-RRT vs. FULL 9D RRT).

| | Permitted Time (s) | Planner | # Base Nodes | | Total Arm Nodes | | Time (s) | | # Coll. Checks | | #Reconfig /#Armchks. | B. Path Len. (m) | #Succ. /#Runs |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | mean | s. d. | mean | s. d. | mean | s. d. | mean | s. d. | | | |
| A | 40 | HAMP-RRT | 33 | 12 | 477 | 484 | 3.1 | 2.6 | 18k | 15k | 3/32 | 4.3 | 30/30 |
| | | RRT | 1677 | 1379 | N/A | N/A | 7.7 | 6.9 | 45k | 16k | N/A | 5.9 | 30/30 |
| B | 40 | HAMP-RRT | 34 | 28 | 771 | 730 | 5.7 | 4.1 | 31k | 21k | 5/33 | 4.5 | 30/30 |
| | | RRT | 2511 | 1208 | N/A | N/A | 13.8 | 7.3 | 68k | 37k | N/A | 6.5 | 30/30 |
| C | 120 | HAMP-RRT | 13 | 4 | 4338 | 2675 | 16.4 | 7.2 | 92k | 43k | 4/13 | 1.1 | 30/30 |
| | | RRT | 5924 | 634 | N/A | N/A | 52.6 | 8.7 | 233k | 56k | N/A | 4.6 | 8/30 |
| D | 70 | HAMP-RRT | 27 | 13 | 989 | 778 | 9.0 | 4.3 | 53k | 36k | 12/26 | 3.2 | 30/30 |
| | | RRT | 3218 | 756 | N/A | N/A | 18.2 | 8.1 | 96k | 41k | N/A | 4.2 | 30/30 |
| E | 300 | HAMP-RRT | 36 | 25 | 1379 | 1097 | 6.5 | 5.0 | 44k | 33k | 10/35 | 3.0 | 30/30 |
| | | RRT | 3814 | 1097 | N/A | N/A | 23.4 | 9.3 | 127k | 36k | N/A | 4.7 | 30/30 |

Table 2.3: Experimental results in 5 scenarios for mobile manipulator planning (HAMP-BiRRT vs. FULL 9D BiRRT).

| | Permitted Time (s) | Planner | # Base Nodes | | Total Arm Nodes | | Time (s) | | # Coll. Checks | | #Reconfig /#Armchks. | B. Path Len. (m) | #Succ. /#Runs |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | mean | s. d. | mean | s. d. | mean | s. d. | mean | s. d. | | | |
| A | 40 | HAMP-BiRRT | 21 | 11 | 309 | 336 | 2.4 | 1.2 | 15k | 6k | 2/20 | 4.4 | 30/30 |
| | | BiRRT | 602 | 244 | N/A | N/A | 2.6 | 1.3 | 17k | 7k | N/A | 5.9 | 30/30 |
| B | 40 | HAMP-BiRRT | 23 | 19 | 663 | 528 | 4.5 | 2.7 | 24k | 11k | 4/23 | 4.5 | 30/30 |
| | | BiRRT | 1447 | 591 | N/A | N/A | 8.1 | 3.4 | 42k | 17k | N/A | 6.6 | 30/30 |
| C | 120 | HAMP-BiRRT | 5 | 3 | 739 | 156 | 1.8 | 1.1 | 11k | 5k | 2/5 | 1.0 | 30/30 |
| | | BiRRT | 346 | 185 | N/A | N/A | 2.2 | 1.1 | 13k | 7k | N/A | 4.6 | 30/30 |
| D | 70 | HAMP-BiRRT | 19 | 16 | 617 | 499 | 5.9 | 1.8 | 39k | 12k | 7/19 | 3.3 | 30/30 |
| | | BiRRT | 1083 | 369 | N/A | N/A | 6.1 | 2.1 | 40k | 22k | N/A | 4.3 | 30/30 |
| E | 300 | HAMP-BiRRT | 30 | 23 | 1235 | 756 | 6.2 | 4.4 | 41k | 27k | 9/33 | 3.0 | 30/30 |
| | | BiRRT | 1371 | 481 | N/A | N/A | 6.6 | 2.5 | 50k | 38k | N/A | 4.2 | 30/30 |

### 2.6.4 Simulation Results for HAMP-U

We tested HAMP-U on 6 different scenarios in simulation and compared the outcomes with HAMP. Our main objective is to show that as a result of embedding BRM in our mobile manipulator planner (HAMP), the paths generated by HAMP-U are less likely to result in collision and are safer to execute as compared to HAMP. Scenarios A to E are same as explained in Sec 2.6.2, the only difference is instead of 50 cm stick we used 120 cm stick for scenario B. Fig 2.12 shows a sequence of snapshots for one of the simulation tests for scenario B, where the mobile manipulator needed to navigate with a stick of 120 cm. It shows that even with arm folded in most compact state (home configuration) at the start with the payload held parallel to the side of the base, the arm configuration still needs to be changed in order to reach the goal (as shown in Fig 2.12 (c), the mobile manipulator can not make a turn due to long payload collision with the wall or the gate). In scenario F, the mobile manipulator needed to navigate through 3 doorways of varying heights and widths to reach the goal. The corresponding simulation videos are available online at `http://www.sfu.ca/~vpilania/research.html`

A comparison between HAMP and HAMP-U for scenarios A to F is presented in Table 2.4. We generated 30 successful runs with HAMP and HAMP-U. Each path is then executed (using a feedback controller) in simulation with artificially generated process and measurement noise, and the number of executions resulted in collision were counted. The percentage times a path generated by HAMP or HAMP-U results in collision with the obstacles upon execution is given in Table 2.4. We can see that the paths generated by HAMP-U are less likely to result in collision as compared to HAMP (which does not take into account the un-

Table 2.4: Results for HAMP vs. HAMP-U (30 Runs).

| | % times a given path results in coll. | |
|---|---|---|
| | HAMP | HAMP-U |
| A | 16.6 % | 3.3 % |
| B | 46.6 % | 10.0 % |
| C | 63.3 % | 50.0 % |
| D | 56.6 % | 16.6 % |
| E | 73.3 % | 43.3 % |
| F | 33.3 % | 6.6 % |

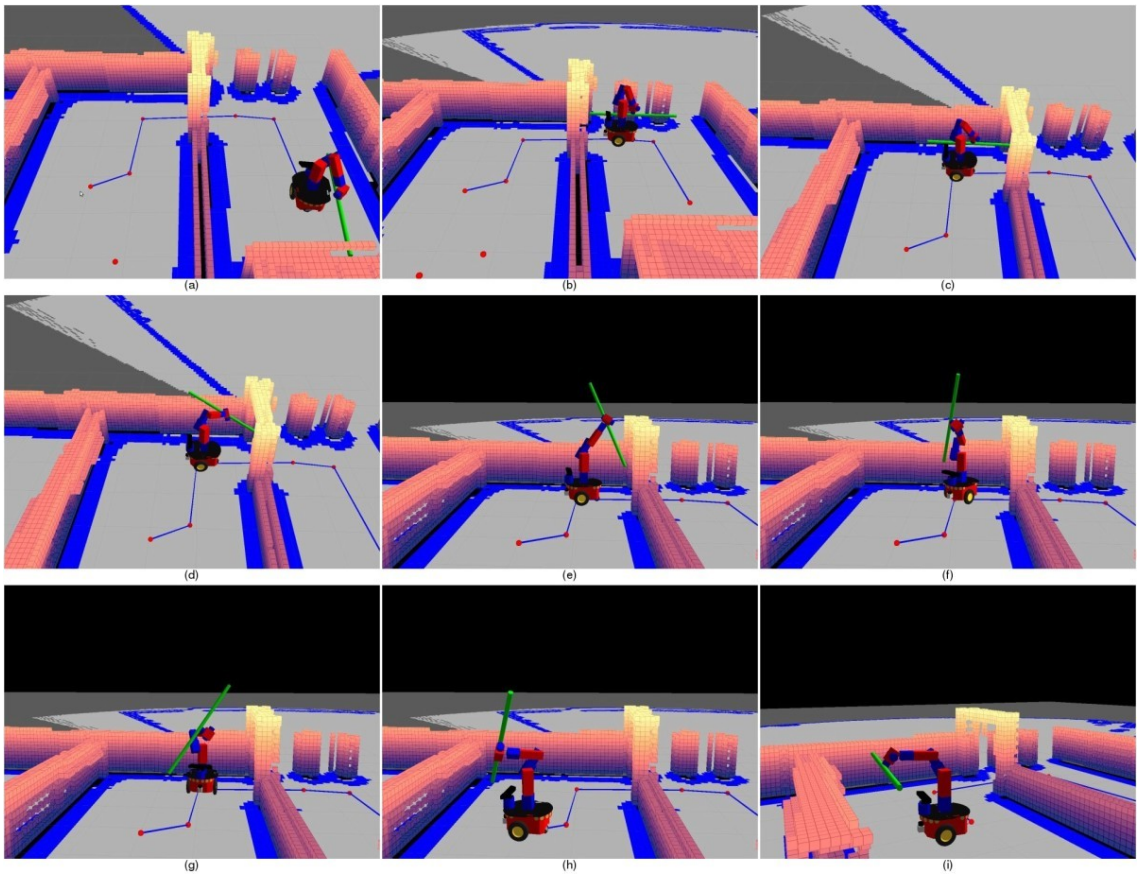



Figure 2.12: HAMP-U simulation test for scenario B: The mobile manipulator carrying a 120 cm stick as payload. This simulation shows that even if you put the arm in compact state (folded), the arm configuration may still need to be changed in order to reach the goal (a) mobile manipulator's start configuration, here the arm is in compact state (folded) and the payload is held parallel along the side of the robot, (b) mobile manipulator with the same arm configuration, (c) in order to move along the path, arm configuration needs to be changed, as mobile manipulator can not make a turn due to long payload collision with wall or gate; (d), (e), (f), (g) show the reconfiguration step, (h) mobile base has reached the goal but not the arm, (i) shows the mobile manipulator's goal configuration, after the final reconfiguration step.

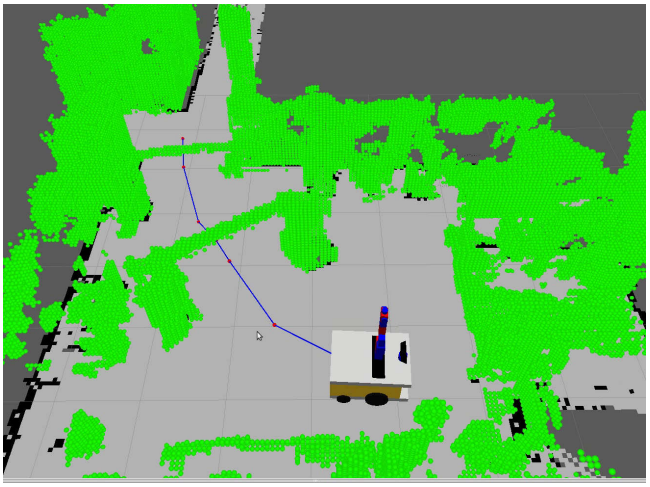


Figure 2.13: The 3D world (collision map) is formed by pre-sensing the environment with kinect and is shown in green above. The 2D map is also formed by pre-sensing with LMS100. The experimental set-up consists of two gates of different height.

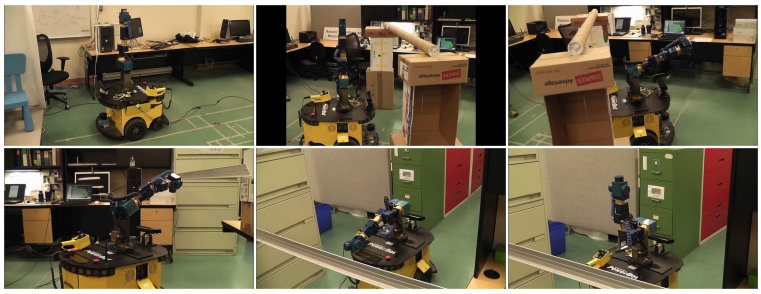


Figure 2.14: Experiment on the real mobile manipulator. Photos are arranged from left to right in each row and then top to bottom. The mobile manipulator moves through two gates of varying heights and the manipulator reconfigures to a new configuration before crossing each gate.

certainties). However, in HAMP-U for scenarios C and E, the collisions are still significant. This is mainly because the corresponding environments are more cluttered, hence, even a minor deviation of the base pose from the planned one leads to collisions. These collisions in HAMP-U can be mitigated by incorporating the effects of base uncertainty (for instance by explicitly using collision probabilities) on the planned manipulator motions. Note that the feedback controller for base, that we used both in simulation and real experiments (in this chapter and throughout thesis) for path execution, computes control commands for the next waypoint along the planned path in such a way that it almost tries to follow the planned path as can be seen in the respective videos. For example, for a path segment, it computes the sequence of actions as the rotation required to orient the robot along that segment, followed by the translation required to reach the other end of the segment.

### 2.6.5 Experiments on SFU mobile manipulator

Figures 2.13 and 2.14 show the 3D map of our lab formed by pre-sensing the environment with kinect and snapshots of one of the several experiments performed with the SFU mobile manipulator in our lab. The same example is shown in the experimental video attached to the thesis (Multimedia Resource 5) and also available online at `http://www.sfu.ca/~vpilania/research.html`. Our experimental set-up consists of two gates, 2nd gate is just 10 cm above the minimum height that can be attained by the manipulator. We used the extended arm configuration as initial configuration to illustrate the reconfiguration step. The planner runtime for the experiment, for which selected screenshots are shown in Fig 2.14 is 5 seconds. We carried out 18 experiments on SFU mobile manipulator with the experimental setup, the minimum and maximum planner runtime was noted to be 2.43 seconds and 7.8 seconds, respectively. In real experiments, for now, we were not able to demonstrate the pole example due to lack of a gripper in SFU mobile manipulator.

# Chapter 3

# Localization Aware Sampling and Connection Strategies

In this chapter, we present our efficient Localization Aware Sampling (LAS) and Localization Aware Connection (LAC) strategies for motion planning under uncertainty. We develop a new measure of "localization ability of a sample" and use this measure to design a localization aware sampling strategy. We evaluate both strategies in the context of a mobile robot. These efficient strategies will save even more computational time for mobile manipulators where 3D collision checks are required (as we show in Chapter 4). We provide simulation results that show that a) our localization aware sampling strategy places less samples and find a well-localized path in shorter time with little compromise on the quality of path as compared to existing sampling techniques, b) our localization aware connection strategy finds a well-localized path in shorter time with no compromise on the quality of path as compared to existing connection techniques, and finally c) combined use of our sampling and connection strategies further reduces the planner run time. The probabilistic completeness and optimality issues with our approaches are also discussed. We show that a stochastic planner that uses our sampling strategy is probabilistically complete under some reasonable conditions on parameters. Our connection strategy is also trivially complete.

## 3.1 Overview of LAS and LAC

The localization aware sampling strategy avoids putting large number of samples by considering the "localization ability" of a new sample relative to its neighbouring nodes in the roadmap. It puts more samples in regions where sensor data is able to achieve higher uncertainty reduction while maintaining an adequate number of samples in regions where uncertainty reduction is poor. This leads to a less dense roadmap that results in significant time savings in the path search phase. Note that localization of a robot at a point depends on 1) the path taken to reach the point and 2) on the update based on sensor model. How-

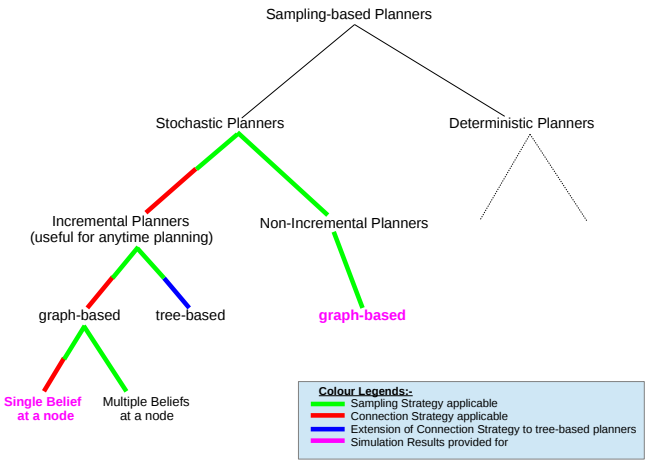



Figure 3.1: It shows the class of planners for which our sampling and connection strategies can be used. For example, while going down the line from incremental planners to graph-based - our both approaches are applicable. Similarly, from incremental planners to tree-based - our sampling strategy and extension of connection strategy to tree-based planners are applicable. It also shows the type of planners for which we provide simulation results.

ever, at the sampling stage the path taken to a node is not available. We develop a new measure of "localization ability of a sample" that "extracts" how well a sensor observation at a sample point reduces uncertainty without explicitly knowing the path leading to it and use this measure to design a localization aware sampling strategy.

A key reason we use reduction in uncertainty as a measure is that higher uncertainty is more detrimental and hence has higher cost for many tasks. Nevertheless, one possible consequence of our sampling technique is that path quality (we use true localization uncertainty along the path as a quality metric) may suffer, if the path passes through regions where uncertainty reduction is poor. Via simulation results, we show that, at least empirically, there is little compromise in path quality. Furthermore, note that since at the sampling stage, true localization uncertainty is not available, a cost function metric using it can not be computed, hence can not be used. The best one can do is to use the uncertainty reduction ability of the sensor at the sample point, as we do. Note that in the search phase (where edges are added and uncertainty is propagate along the path), appropriate cost function is still minimized.

The localization aware connection strategy first connects the new sample to a nearest node (chosen based on an uncertainty metric and not on distance metric) and then to other neighbouring nodes. Connection from new sample to a neighbouring node is made only if the new path to that node reduces the uncertainty. Our efficient connection strategy eliminates the inefficient edges that would be created in current connection schemes but do not contribute toward better localization. As a result, it also reduces the number of search queue iterations needed to update the paths. This helps to find a well-localized path in

shorter time with no compromise on the quality of path. Note that our strategy applies to graph-based incremental stochastic planners that maintain a single belief at a node and is not applicable to planners with multiple beliefs. Multiple beliefs at a node are needed for planners that optimize multiple objective functions since multiple paths to a node can not be completely ordered (as is the case for a single objective function, which can be a weighted sum of multiple costs) and need to be kept so as not to prematurely prune an optimal one, although domination criteria can be used to do some pruning (see [28]). Of course, tree-based methods, by definition, have single belief since they have a unique path to any given node. Fig 3.1 clearly shows the class of planners for which we provide simulation results and where our contributions are applicable.

Please note that our main objective here is to come up with more efficient sampling and connection strategies that are applicable to classes of planners as stated in Fig 3.1. We use an existing planner, RRBT basically as a tool (i.e., as a base planner) to demonstrate the efficiency gained by our strategies for incremental stochastic planners. The objectives in the respective planners we choose to implement are those of the original planners, i.e., minimize the uncertainty at goal while respecting the chance-constraints (threshold on uncertainty) along the path.

## 3.2 Localization Ability of a Sample

In this section, we describe how to compute the localization ability of a sample. For this, we first briefly explain the extended Kalman filter (EKF) [26] and then develop an expression for the localization ability of a sample.

Applying a control input $u_t$ at time $t$ brings the robot from state $x_t$ at time $t$ to state $x_{t+1}$ at time $t+1$ according to a given stochastic dynamics model:

$$x_t = f(x_{t-1}, u_{t-1}, w_t), \qquad w_t \sim \mathcal{N}(0, W_t) \tag{3.1}$$

where $w_t$ is the process noise at time $t$ drawn from a zero-mean Gaussian distribution with variance $W_t$ that models the motion uncertainty. After each motion, the robot receives noisy sensor readings $z_t$ at time $t$ that provide us with partial information about the state according to a given stochastic observation model:

$$z_t = h(x_t, q_t), \qquad q_t \sim \mathcal{N}(0, Q_t) \tag{3.2}$$

where $q_t$ is the measurement noise drawn from a zero-mean Gaussian distribution with variance $Q_t$ that models the sensor uncertainty.

We assume that the robot state is represented by Gaussian $(\mu, \Sigma)$ - $\mu$ being the mean and $\Sigma$ being the covariance. The systems in our case are generally non-linear, therefore, the EKF linearizes $f$ and $h$ functions at each step. The EKF estimates the state at time $t$

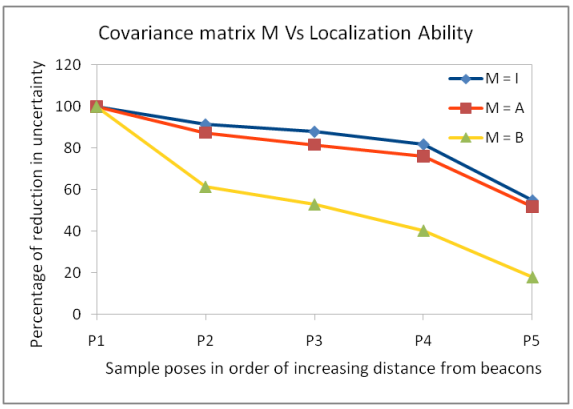



Figure 3.2: Covariance matrix ($M$) Vs Localization ability ($L_n$) for five sample points (P1 to P5). I = [1, 1, 1], A = [1, 0.15, 1], B = [0.15, 0.15, 1].

from the estimate at time $t-1$ in two separate steps: process step to propagate the applied control input $u_{t-1}$, and a measurement step to incorporate the obtained measurements $z_t$. The process step follows as:

$$\overline{\mu}_t = f(\mu_{t-1}, u_{t-1}) \tag{3.3}$$

$$\overline{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + V_t W_t V_t^T \tag{3.4}$$

where $G_t$ and $V_t$ are the Jacobian matrices of $f$ with respect to $x$ and $w$. Similarly, the measurement step follows as:

$$\mu_t = \overline{\mu}_t + K_t(h(\overline{\mu}_t) - z_t) \tag{3.5}$$

$$\Sigma_t = \overline{\Sigma}_t - K_t H_t \overline{\Sigma}_t \tag{3.6}$$

where $H_t$ is the Jacobian of $h$ with respect to $x$ and $K_t$ is known as the Kalman gain,

$$K_t = \overline{\Sigma}_t H_t^T (H_t \overline{\Sigma}_t H_t^T + Q_t)^{-1} \tag{3.7}$$

Equation 3.4 propagates the uncertainty in robot state from $\Sigma_{t-1}$ (at time $t-1$) to $\overline{\Sigma}_t$ (at time $t$) after incorporating control input and associated motion noise. Equation 3.6 further propagates it from $\overline{\Sigma}_t$ to $\Sigma_t$ after incorporating sensor measurements and the associated sensor noise. It is Equation 3.6 that reduces the uncertainty with the help of meaningful measurements and it is this reduction in uncertainty, from $\overline{\Sigma}_t$ to $\Sigma_t$, that we are interested in capturing.

This can be achieved if we assume an a priori uncertainty at each sample point, say a covariance matrix M. Therefore Equations 3.6 and 3.7 will change to:

$$\Sigma_n = M - K_n H_n M \tag{3.8}$$

$$K_n = MH_n^T(H_nMH_n^T + Q_n)^{-1} \tag{3.9}$$

where subscript $n$ stands for newly sampled robot pose. The localization ability of a sample $n$ is then given by $L_n = \frac{tr(M)-tr(\Sigma_n)}{tr(M)} \times 100$. Since we are using trace[1], therefore, we use a diagonal matrix for M, in fact an identity matrix.

Clearly, $L_n$ in general depends on $M$. However, Fig 3.2 empirically shows that the $L_n$ monotonically reduces irrespective of specific $M$ (we chose three arbitrary $M$'s) as the distance of samples from beacons increases, i.e., ability of sensor data to reduce uncertainty is reduced for all three different $M$. It is this trend that is important.

As mentioned in Sec 3.1, $L_n$ reflects just the sensor's ability to gather accurate information and not the actual localization uncertainty at the sample. The latter also depends on the path chosen and the accuracy of process model and can not be computed at sampling stage. Once the sample is connected to the roadmap, the true belief will be computed by search mechanism (uncertainty propagation from start). $L_n$ is just a measure to accept or reject a sample.

## 3.3 Rapidly-exploring Random Belief Tree with Localization Aware Sampling Strategy

In this section, we provide a Rapidly-exploring Random Belief Tree with Localization Aware Sampling Strategy (RRBT-LAS) algorithm where we replace uniform sampling of RRBT [28] with our localization aware sampling strategy. Please note that we did not consider linear-quadratic Gaussian (LQG) controller in RRBT-LAS simply because our focus is on showing efficiency of our sampling scheme and if RRBT-LAS is efficient (without LQG) then it will be more efficient after incorporating LQG that requires additional computation along the edges of the roadmap.

The algorithm operates on a set of nodes $V$ and edges $E$, that define a roadmap in state space. Each node $v \in V$ has a state $v.x$, state estimate covariance $v.\Sigma$, a parent node $v.parent$, and localization ability $v.\text{loc}$. The state covariance prediction and chance-constraint checking [28] is implemented by a PROPAGATE$(e, v_{start})$ routine that takes as arguments an edge and a starting node for that edge, and returns a covariance matrix at the ending node for that edge. If the chance-constraint is violated by the uncertainty at ending node, the function returns no covariance matrix. The comparison of partial paths at a node $v$ is implemented by UPDATEBELIEF$(v, \Sigma)$ routine that updates the covariance matrix and parent node at $v$ if the new path is less uncertain. We also require the following routines: SAMPLE() returns i.i.d. uniform samples, NEAREST$(V, v_{new})$ takes the current set of nodes as an argument and returns the node in $V$ that minimizes euclidean distance to

[1]Note that other works have commonly used trace as a measure, however, there are other options as well, for example, one can use determinant.

**Algorithm 7:** RRBT-LAS Algorithm

```
1  v.x := x_start; v.Σ := Σ_0; v.parent := NULL
2  v.loc := tr(M)
3  V := {v}; E := {}
4  while i < P do
5      (x_rand, loc_ability) := LocalizationBiasedSample()
6      v_nearest := Nearest(V, x_rand)
7      e_nearest := Connect(v_nearest.x, x_rand)
8      if Propagate(e_nearest, v_nearest.Σ) then
9          v_rand.loc := loc_ability; v_rand.x := x_rand
10         V := V ∪ v_rand
11         E := E ∪ e_nearest
12         Q := Q ∪ v_nearest
13         V_near := Near(V, v_rand)
14         for all v_near ∈ V_near do
15             E := E ∪ Connect(v_near.x, x_rand)
16             Q := Q ∪ v_near
17         while Q ≠ ∅ do
18             u := Pop(Q)
19             for all v_neighbor of u do
20                 Σ' :=Propagate(e_neighbor, u.Σ)
21                 if UpdateBelief(v_neighbor, Σ') then
22                     Q := Q ∪ v_neighbor
23     i := i + 1
```

**Algorithm 8:** LocalizationBiasedSample()

```
1  x_rand := Sample()
2  L_{x_rand} :=ComputeLocalizationAbility(x_rand)
3  if L_{x_rand} < LocAbilityTH then
4      V_neighbor := Neighbor(V, x_rand, DistTH)
5      for all v_neighbor ∈ V_neighbor do
6          if v_neighbor.loc > L_{x_rand} then
7              reject sample, go to step 1
8  return (x_rand, L_{x_rand})
```

$v_{new}$, and $\text{NEAR}(V, v_{new})$ returns every node within some ball centered at $v_{new}$ of radius $\rho \propto (\log(n)/n)^{1/d}$ where $n$ is the number of nodes and $d$ is the state dimension (See [54]).

### 3.3.1 RRBT-LAS Algorithm Description

The RRBT-LAS algorithm is described in Algorithm 7. The roadmap is initialized with a single node with state $x_{start}$, covariance $\Sigma_0$ and its localization ability $tr(M)$ (trace of

matrix $M$) from lines 1-3. At each iteration of the while loop, the roadmap is updated by sampling a new state using our localization aware sampling strategy (line 5), described in Sec 3.3.2, and then adding edges to the nearest and near nodes as in the RRG algorithm [54]. Whenever an edge is added from an existing node to the new node, the existing node is added to the queue (lines 12 and 16). It should be noted that the new node is only added to the roadmap (along with the appropriate edges) if the chance-constraint can be satisfied by propagating an existing belief at the nearest node to the new sampled node as shown on line 8. After all the edges have been added, the queue is exhaustively searched from lines 17-22 using UpdateBelief() routine. Note that the true belief of a node is computed during search mechanism (uncertainty propagation) from lines 19-22 which is different from its localization ability (line 5).

### 3.3.2 Localization Aware Sampling Strategy

Our localization aware sampling strategy puts more samples in regions where sensor data is able to achieve higher uncertainty reduction while maintaining adequate samples in regions where uncertainty reduction is poor. The regions are decided based on a threshold "LocAbilityTH", i.e., if the localization ability of a new uniformly sampled point is above this threshold, then the sample lies in regions with high uncertainty reduction and is simply added as a node. If the localization ability of a sample is below the threshold, the sample lies in regions with low uncertainty reduction, and the decision to accept or reject is governed by the localization ability of neighbouring nodes. If any neighbouring node within a ball of radius "DistTH" centered at the new sample has a localization ability above that of the new sample, the new sample is simply rejected, otherwise it is accepted as a node. This localization aware sampling strategy is implemented by routine LocalizationBiasedSample() as described in Algorithm 8.

A main motive behind accepting all the samples which lie within regions with high uncertainty reduction is to favour paths through such regions because they will likely result in high path quality. In the worst case, i.e., when the only way for a robot is to pass through regions of low uncertainty reduction (as explained in Fig 3.15 in Sec 3.6), path quality will be compromised since there are less samples in these regions (as validated in simulations, compromise is small), but at the same time we gain significant savings in planning time. Do note that if we decrease "DistTH" or "LocAbilityTH", our localization aware sampling strategy will converge to uniform sampling. Correspondingly, a higher DistTH results in faster run time, however, to retain probabilistic completeness, there is an upper bound (see Sec 3.3.3).

Furthermore, as we have mentioned before, localization uncertainty also depends on process noise at the sample point, information that is not available at sampling stage since it requires knowledge of path to the sample point. Assuming that the process noise is similar within the neighbourhood of a sample, accepting or rejecting a sample based on $L_n$


Figure 3.3: Regions with high uncertainty reduction (better localization ability of sample points) also have high process noise. However, the path search phase mitigates this when the cost function is minimized. (a) a stochastic planner knows only one Gaussian process noise model, i.e., it does not know that Region 1 has high process noise and Region 2 has low process noise, therefore, a path is computed which passes through Region 1 (near beacons), (b) planner knows two Gaussian process noise models, one corresponding to each Region, the path search phase that minimizes uncertainty finds a path that passes through Region 2.

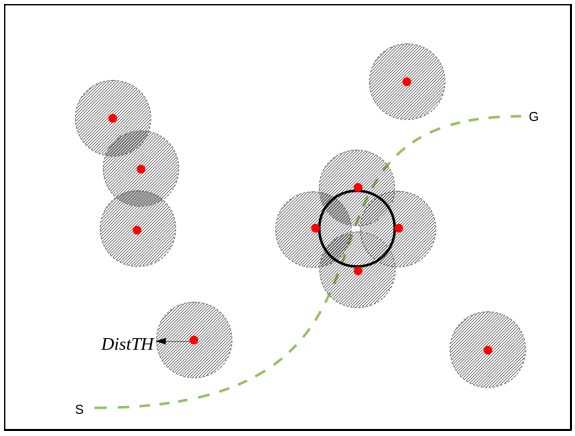



Figure 3.4: Black color (bold) circle denotes one of the balls of radius $r$ that is used to tile a path, red color dots represent randomly placed samples, and hatched region (of radius $DistTH = r$) around each sample denotes the restricted region where samples can not be placed according to heuristic used in localization aware sampling. This figure shows the situation where none of the sample has yet been placed inside the black ball, but some samples are placed at a distance $d$ from its center such that $r < d < r + \epsilon$. Even in this worst case scenario, the probability of generating a sample, given by the ratio of volume of white region inside the black ball (after excluding the hatched region) and total volume of white region, is greater than 0. This ratio approaches one as more and more samples are placed outside the black ball.

(even though we note that localization depends on both steps of EKF) is defensible. It is indeed possible that regions with high uncertainty reduction ability of sensor may also have high process noise, hence the overall uncertainty may still be high. This is mitigated by the search phase of the planner where actual uncertainty is computed and the cost function (based on uncertainty) is minimized. See Fig 3.3 for such an example. In the figure, Region 1 has high process noise but our localization aware sampling strategy puts more samples because the localization ability of sample points is higher in that region as compared to Region 2 (which gets lesser samples). In (a) a stochastic planner does not know that Region 1 has high process noise and Region 2 has very low process noise, it just knows one Gaussian process noise model from which it samples the process noise while propagating the uncertainty from start to goal at path search phase. This is the case with all the stochastic planners. Therefore, a path is computed which passes through Region 1. However, it is not difficult to adapt the planner to different process noise models. In (b) the planner knows two Gaussian process noise models and which one to apply for a region at the path search phase. Therefore, it mitigates the overall uncertainty and finds a path which passes through Region 2.

### 3.3.3 Probabilistic Completeness and Optimality

First, we explain why we show probabilistic completeness for RRBT-LAS as generally the notion of completeness is useful for deterministic planners. The class of sampling-based stochastic planners that we are concerned with in this paper (including RRBT and thereby RRBT-LAS) first requires the construction of roadmap (or tree) in configuration space (C-space) and then followed by an uncertainty propagation step. This class has underlying collision checks with respect to nominal paths. Therefore, if a sampling strategy (as in our case for some value of DistTH) does not allow to connect the different regions of the C-space then the next step of uncertainty propagation can not be done and the planner will not find a path. That is why it is important to discuss the completeness of sampling strategy in exploring the C-space prior to uncertainty propagation step.

The probabilistic completeness is along the lines of [71] and is essentially proved by assuming that a collision free path with clearance $\rho \geq 2r$ exists (where $2r$ is the radius of the largest inscribed circle within the robot), and then tiling it with a set of carefully chosen balls of radius $r$ such that generating a sample in each ball ensures that these samples can be connected with collision-free edges and therefore, a collision-free path will be found. As shown in Fig 3.4, we can show that for $DistTH \leq r$ the probability of generating such samples approaches 1 as the number of samples increases. We show that RRBT-LAS is probabilistically complete under this reasonable restriction of DistTH. The complete proof is available in Appendix B.

As a result of the sampling strategy, RRBT-LAS, in general, will not be optimal. However, in specific class of scenarios, one may be able to show that asymptotic optimality is retained. we are currently looking into it and is part of our future work.

## 3.4 Localization Aware Connection Strategy

RRBT-TF (see Sec 1.2.4) inherits the connection strategy from the Rapidly-exploring Random Graph (RRG) [54] which is a deterministic motion planner. Fig 3.5 demonstrates how RRBT-TF uses this connection strategy for incremental stochastic motion planners. For a newly sampled point $x_{new}$, RRBT-TF propagates the uncertainty from nearest node (selected based on distance metric) to the new sample and if the uncertainty obtained satisfies the chance-constraints then the new sample and corresponding edge ($e_{nearest}$) are added to the roadmap as shown in Fig 3.5 (b). Note that the nearest node is now inserted into search queue $Q$ , however, there is no belief update at $x_{new}$. Only after successful connection of $x_{new}$ to $v_{nearest}$, it proceeds further to connect $x_{new}$ to other neighbouring nodes and insert them into $Q$ as shown in Fig 3.5 (c). RRBT-TF then uses $Q$ to iterate through inserted nodes and update the paths (uncertainty propagation).

Now we explain our localization aware connection strategy which is demonstrated in Fig 3.6. For a newly sampled point $x_{new}$, instead of connecting it to the distance metric based

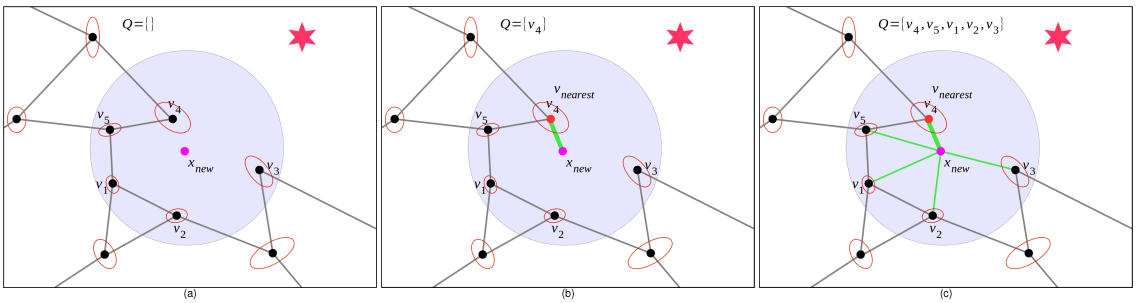



Figure 3.5: These snapshots show the connection strategy of RRBT-TF. Red color star denotes beacon, red color ellipses denote belief nodes and $Q$ denotes search queue. Furthermore, red color node denotes $v_{nearest}$, thick green line denotes $e_{nearest}$, and thin green lines denote $e_{near}$. (a) newly sampled point $x_{new}$; (b) $v_{nearest}$ (based on distance metric) is connected to $x_{new}$ only if chance-constraints are satisfied and then inserted into $Q$, so far no belief update at $x_{new}$; (c) connecting $x_{new}$ to all other neighbouring nodes and inserting them into $Q$, again no belief update at $x_{new}$. RRBT-TF now uses the search queue $Q$ to iterate through the inserted nodes and update the beliefs.

nearest node ($v_4$ in case of RRBT-TF, Fig 3.5), we connect it to one of the neighbouring nodes such that the uncertainty propagation from that node to $x_{new}$ gives minimal uncertainty at $x_{new}$. We call that neighbouring node as our nearest node. Fig 3.6 (b) and (c) demonstrate it: (b) shows the uncertainty propagation from neighbouring nodes to $x_{new}$; (c) shows that $v_1$ is selected as nearest node and it is connected to $x_{new}$ (i.e., $x_{new}$ and $e_{nearest}$ are added to the roadmap given that uncertainty obtained at $x_{new}$ satisfies the chance-constraints). In contrast to RRBT-TF, we update belief at $x_{new}$ and do not insert $v_1$ into $Q$. This is because a least uncertain path to $x_{new}$ is already computed. Only after successful connection of $x_{new}$ to $v_{nearest}$ ($v_1$), we proceed further to connect $x_{new}$ to other neighbouring nodes ($v_2$, $v_3$, $v_4$, $v_5$). Different (and efficient) from RRBT-TF, which simply connect $x_{new}$ to all other neighbouring nodes as in deterministic planner RRG, we connect $x_{new}$ to only those nodes for which the new path (through $x_{new}$) reduces the uncertainty. Fig 3.6 (d) explains it better. In the figure, the uncertainty propagation from $x_{new}$ to $v_2$ increases the uncertainty at $v_2$ (green ellipse is bigger than red ellipse), therefore $x_{new}$ can not be connected to $v_2$. Similarly, in Fig 3.6 (e), the uncertainty propagation from $x_{new}$ to $v_3$ and $v_4$ reduces the uncertainty at $v_3$ and $v_4$, therefore, $x_{new}$ can be connected to these nodes. Fig 3.6 (f) shows the edges that are finally added to the roadmap along with the belief updates at corresponding nodes (shown by green ellipse). It also shows the nodes that are inserted into $Q$. Similar to RRBT-TF, the planner then uses $Q$ to iterate through inserted nodes and update the paths.

In summary, our localization aware connection strategy does not add those edges along which localization is poor and also inserts less number of nodes into search queue. The combined effect of these two factors reduces the run time in our case with no impact on path quality. Also, the roadmap obtained using our connection strategy is a sub-roadmap

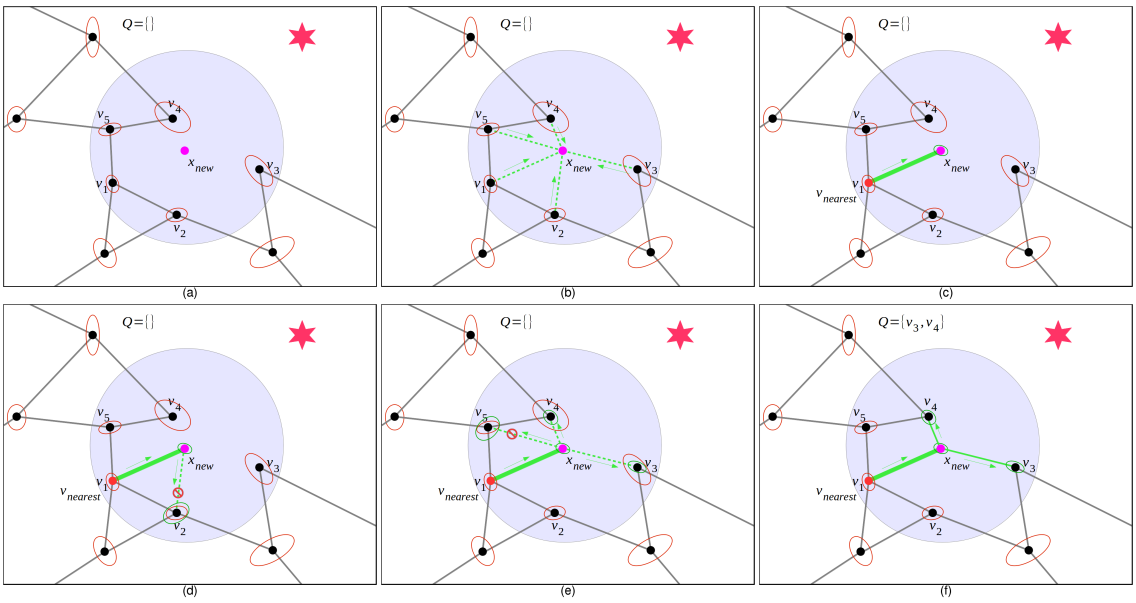



Figure 3.6: The sequence of snapshots show our localization aware connection strategy. All the notations stated in Fig 3.5 also hold true in this figure. In addition, green color dashed lines denote the uncertainty propagation without actually adding those edges. The direction of uncertainty propagation is shown by green arrow. (a) newly sampled point $x_{new}$; (b) to search for $v_{nearest}$, uncertainty is propagated from neighbouring nodes to $x_{new}$; (c) $v_{nearest}$ is found (different from RRBT-TF), if chance-constraints are satisfied then $e_{nearest}$ is added and belief at $x_{new}$ is updated (shown by green ellipse); (d) uncertainty propagation from $x_{new}$ to $v_2$, new path to $v_2$ has more uncertainty, therefore, no update of belief at $v_2$ (also shown by red color cross mark); (e) similar attempts to connect $x_{new}$ to other neighbouring nodes $v_3$, $v_4$, $v_5$; (f) successful edges (to nodes $v_3$, $v_4$) that reduce uncertainty are added along with belief updates at corresponding nodes. Note that only nodes $v_3$, $v_4$ are inserted into $Q$.

of the RRBT-TF roadmap. These points can be verified by comparing Fig 3.6 (f) with Fig 3.5 (c).

### 3.4.1 Rapidly-exploring Random Belief Tree with Localization Aware Connection Strategy

We provide a Rapidly-exploring Random Belief Tree with Localization Aware Connection Strategy (RRBT-LAC) algorithm where we replace connection strategy of RRBT-TF with our localization aware connection strategy.

The RRBT-LAC algorithm is described in Algorithm 9. The roadmap is initialized with a single node with state $x_{start}$ and covariance $\Sigma_0$ from lines 1-2. At each iteration of the while loop (line 3), the roadmap is updated by sampling a new state and then adding edges to the neighbouring nodes using our localization aware connection strategy. Lines 6-10 correspond to Fig 3.6 (b) where we propagate the uncertainty from neighbouring nodes to

**Algorithm 9:** RRBT-LAC Algorithm

```
1  v.x := x_start; v.Σ := Σ_0; v.parent := NULL
2  V := {v}; E := {}
3  while i < P do
4      x_rand := SAMPLE(); Σ_rand := ∅
5      V_near := {NEAR(V, x_rand)}
6      for all v_near ∈ V_near do
7          e_near := CONNECT(v_near.x, x_rand)
8          Σ' :=PROPAGATE(e_near, v_near.Σ)
9          if tr(Σ') < tr(Σ_rand) or tr(Σ_rand) = 0 then
10             v_nearest=v_near; e_nearest=e_near; Σ_rand=Σ'
11     if tr(Σ_rand) ≠ 0 then
12         V := V ∪ v_rand(x_rand, Σ_rand, v_nearest)
13         E := E ∪ e_nearest
14         for all v_near ∈ V_near \ v_nearest do
15             e_near := CONNECT(v_rand.x, v_near.x)
16             Σ' :=PROPAGATE(e_near, v_rand.Σ)
17             if UPDATEBELIEF(v_near, Σ') then
18                 E := E ∪ e_near
19                 Q := Q ∪ v_near
20         while Q ≠ ∅ do
21             u := POP(Q)
22             for all v_neighbor of u do
23                 Σ' :=PROPAGATE(e_neighbor, u.Σ)
24                 if UPDATEBELIEF(v_neighbor, Σ') then
25                     Q := Q ∪ v_neighbor
26     i := i + 1
```

the new sample in order to search for the nearest node. In lines 12-13 we actually add the new sample and the corresponding edge (connecting new sample to the nearest node) to the roadmap which corresponds to Fig 3.6 (c). Then from lines 14-19 we try to connect the new sample to the other neighbouring nodes and adding them to the search queue $Q$ only if the corresponding connections reduce the uncertainty at these nodes as shown by the check on line 17. This corresponds to Fig 3.6 (d), (e) and (f). After all the successful edges have been added, the search queue $Q$ is exhaustively searched from lines 20-25 where the queue iterates through all the inserted nodes and updates the paths.

### 3.4.2 Probabilistic Completeness

The planner (RRBT-LAC) that uses our localization aware connection strategy uniformly sample a new state. Moreover, the connection (to a node) is avoided only if the new path

does not contribute to better localization at that node as compared to its old path. Basically, the first connection to a node (whenever possible, i.e., collision-free) is always established. Therefore, RRBT-LAC is probabilistically complete and the proof is same as mentioned in [71].

### 3.4.3 Asymptotic Optimality

Unlike sampling strategy, our connection strategy does not compromise on the connection part. It eliminates only those partial paths for which there exists a better substitution, i.e. less uncertain path. Therefore, RRBT-LAC is asymptotic optimal on the similar lines as mentioned in RRBT.

## 3.5 Extending Our Localization Aware Connection Strategy to Tree-based Planners

The tree version of our localization aware connection strategy uses the uncertainty metric to connect to the "nearest" node and then "rewires" (borrowing the "rewiring" notion of RRT* [54]). More formally, we propose the following key modifications in RRT* to handle stochasticity associated with a robot's motion and its sensory readings:

a) Instead of connecting the new sample to the distance metric based nearest node, we propose to search for a nearest node (and connect it to the new sample) as we demonstrate in Fig 3.6 (b) and (c) for incremental roadmap. This ensures a least uncertain path for the new sample.

b) To connect the new sample to other neighbouring nodes, we propose to use the similar strategy as we demonstrate in Fig 3.6 (e), however, it should use the "rewiring" notion in order to maintain tree structure. For example: after connecting the new sample to the nearest node (as described above), the uncertainty is propagated from the new sample to a neighbouring node, if it reduces the uncertainty at that node then the edge connecting that neighbouring node to its parent node is removed and the new edge connecting the new sample to that neighbouring node is retained. This ensures a least uncertain path from the root of the tree to a node.

## 3.6 Results

In this section, we provide simulation results for RRBT-LAS, RRBT-LAC and RRBT-LASC (RRBT with our localization aware sampling and connection strategies).

We used the motion model and sensor model from [26] for all the planners. We experimented with two different sensor models: RangeModel 1 - where beacons have a limited

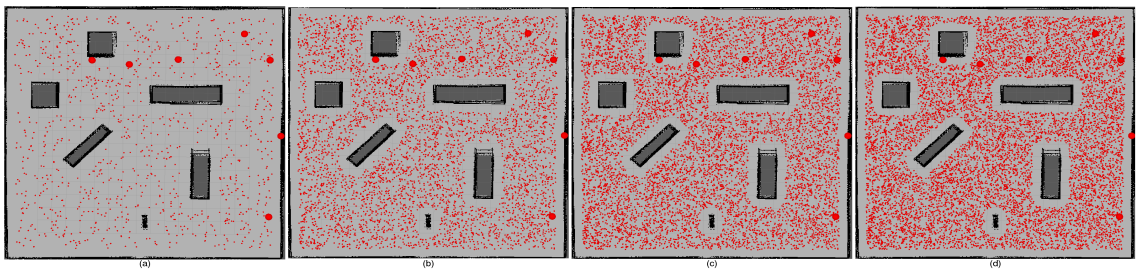


Figure 3.7: RRBT-TF [uniform sampling]. # input samples from a seed [# actual nodes in the roadmap] : (a) 1000 [1000], (b) 5000 [5000], (c) 8000 [8000], (d) 10000 [10000]

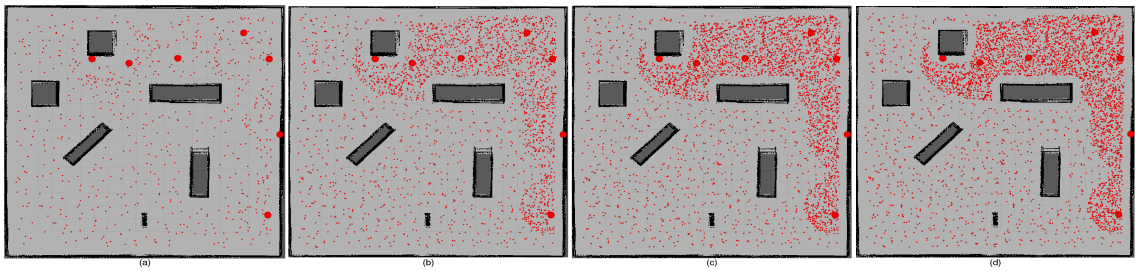


Figure 3.8: RRBT-LAS [localization aware sampling] using RangeModel 2. # input samples from a seed [# actual nodes in the roadmap] : (a) 1000 [680], (b) 5000 [2331], (c) 8000 [3347], (d) 10000 [4060] . Here we used DistTH = 30 cm and LocAbilityTH = 90% (reduction in uncertainty).

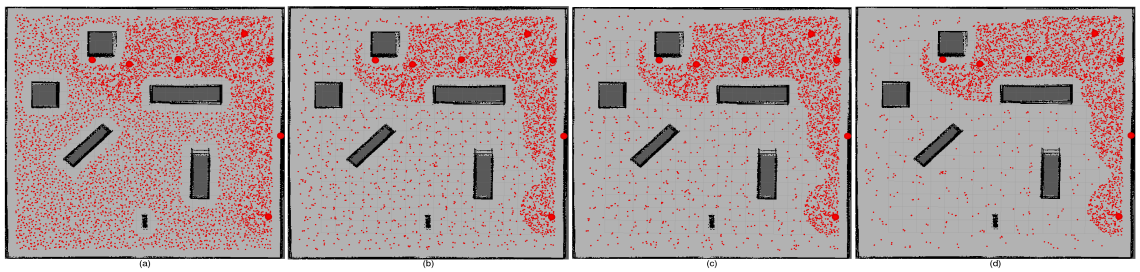


Figure 3.9: Effect of varying DistTH (while keeping LocAbilityTH = 90% and the number of input samples from a seed as 10000) in RRBT-LAS using RangeModel 2. DistTH in (a) 10 cm, (b) 30 cm, (c) 40 cm, (d) 60 cm.

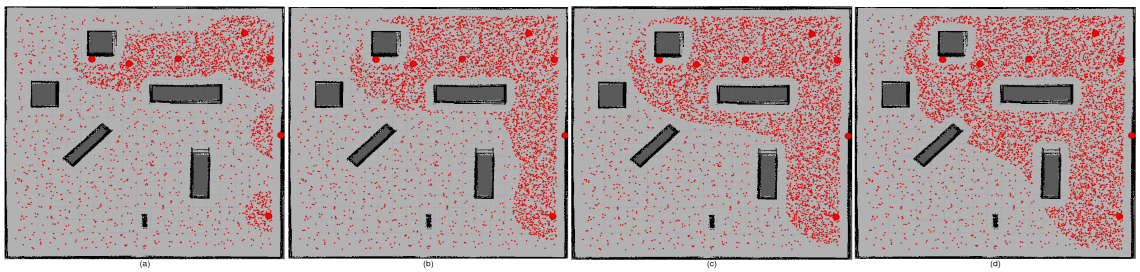


Figure 3.10: Effect of varying LocAbilityTH (while keeping DistTH = 30 cm and the number of input samples from a seed as 10000) in RRBT-LAS using RangeModel 2. LocAbilityTH in (a) 93.3%, (b) 86.6%, (c) 83.3%, (d) 76.6%.

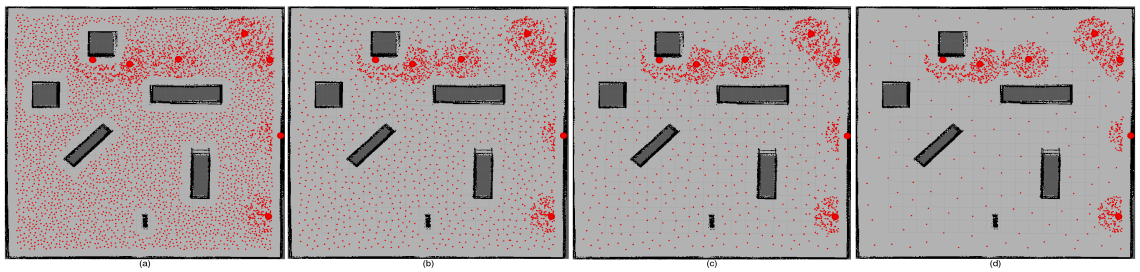


Figure 3.11: Effect of varying DistTH (while keeping LocAbilityTH = 86.6% and the number of input samples from a seed as 10000) in RRBT-LAS using RangeModel 1. DistTH in (a) 20 cm, (b) 40 cm, (c) 50 cm, (d) 70 cm.

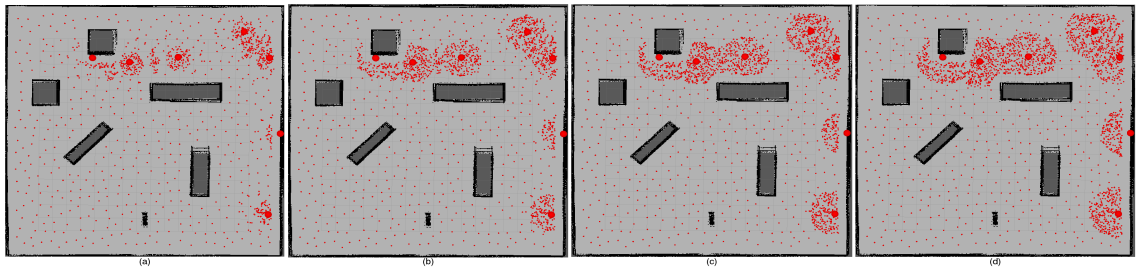


Figure 3.12: Effect of varying LocAbilityTH (while keeping DistTH = 30 cm and the number of input samples from a seed as 10000) in RRBT-LAS using RangeModel 1. LocAbilityTH in (a) 93.3%, (b) 86.6%, (c) 83.3%, (d) 76.6%.

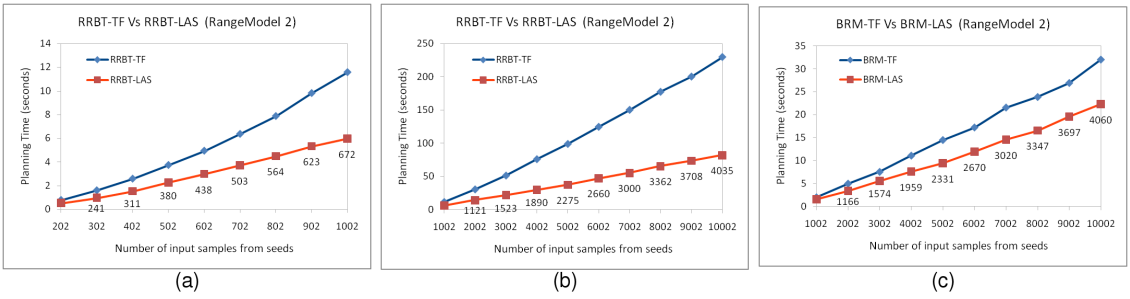



Figure 3.13: Plots show the comparison of RRBT-TF Vs RRBT-LAS for incremental motion planning (a, b) and of BRM-TF Vs BRM-LAS for non-incremental motion planning (c). Data labels for each data point along red curves in RRBT-LAS and BRM-LAS show the actual number of nodes in the roadmap. For RRBT-TF and BRM-TF, actual number of nodes and number of input samples from seeds are the same, therefore, data labels are not shown along their corresponding curves. For the plots we used DistTH = 30 cm and LocAbilityTH = 86.6%. Also note that the saving in planning time is for $DistTH \leq r$ (where $2r$ is the inscribed radius of robot).

range (we used 2 meters) and RangeModel 2 - where beacons have range that spans the entire map. For both sensor models, the sensor data has a distance varying Gaussian noise. First we report our results with RangeModel 2. Please note that our localization aware sampling and connection strategies also hold for complex measurement models, for example: range sensors.

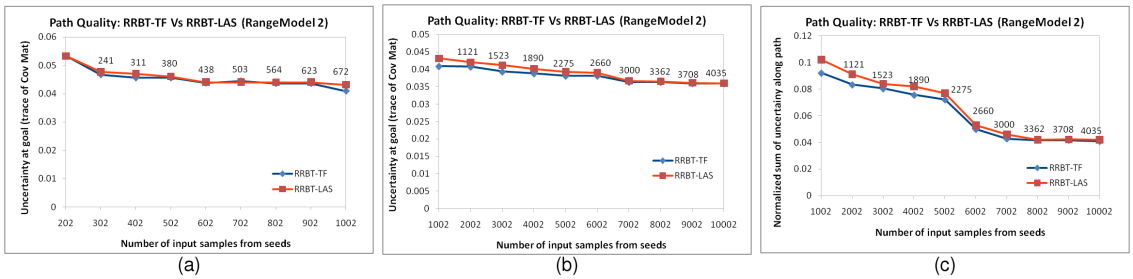



Figure 3.14: Comparison of path quality between RRBT-TF and RRBT-LAS for a scenario where the only path to goal passes through regions with low uncertainty reduction, essentially a worst case scenario for path quality for our sampling scheme (see Fig 3.15). Y-axis denotes the trace of covariance matrix at goal in plots (a, b) and normalized sum of trace of covariance matrices along path in plot (c).

We used 30 different seeds, each seed generating a set of 10000 pseudo random collision-free input samples. We ran all the planners on each set by varying the number of input samples from 100 to 10000 in incremental manner and provided our simulation results by averaging the outcome over 30 sets. Our implementation is in C++ under linux and runs on a Pentium dual core 2.5 Ghz computer with 4GB memory.

### 3.6.1 Simulation Results for RRBT-LAS

We compared RRBT-LAS algorithm with RRBT-TF. We demonstrate our approach in two ways: (i) through visualization in Fig 3.7 - 3.10, we show the efficacy of our localization aware sampling strategy in judiciously placing the samples, and (ii) we use plots in Fig 3.13 and 3.14 to show that our localization aware sampling leads to saving in planning time with little compromise on the quality of path.

Fig 3.7 shows the placement of nodes in the roadmap (edges are not shown) for uniform sampling as we increase the number of input samples from a seed. The uniform sampling strategy is not aware of sensor model, therefore, the actual number of nodes in the roadmap are equal to the number of input samples added from a seed. Big red color balls (7 of them) in the snapshots denote the beacons which were used for localization. Compared to uniform sampling (Fig 3.7), the actual number of nodes in our localization aware sampling strategy are significantly reduced as shown in Fig 3.8. From the figures, we observe that our localization aware sampling strategy places more samples in regions with high uncertainty reduction (near beacons) and eliminates unnecessary samples from regions with low uncertainty reduction. We also show the effect of varying two thresholds (DistTH and LocAbilityTH) in our localization aware sampling strategy. In Fig 3.9, we varied only DistTH and observed that the sparsity of nodes in regions with low uncertainty reduction increases with the increase of DistTH. However, the nodes in regions with high uncertainty reduction remain unchanged with the variation of DistTH. Similarly, in Fig 3.10, we varied only Lo-

cAbilityTH and observed that the area under regions deemed as high uncertainty (higher than the threshold) reduction increases with the decrease of LocAbilityTH. In Fig 3.9 and 3.10, we kept the number of input samples from a seed as 10000, therefore, the comparison should be done with Fig 3.7(d).

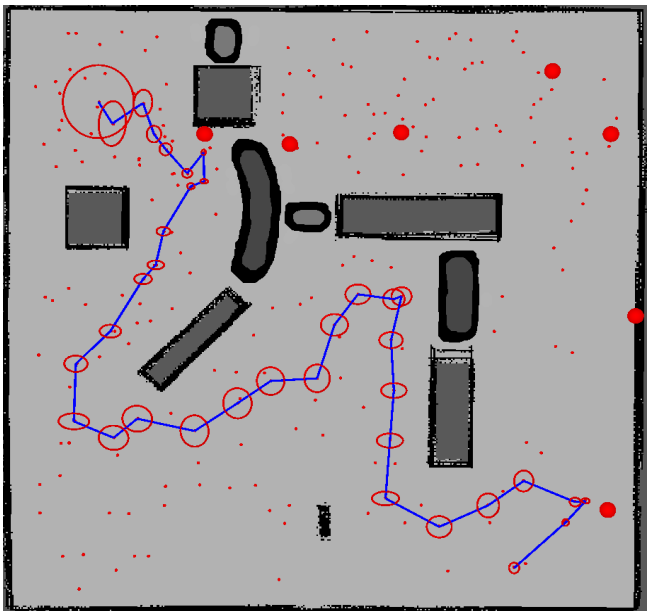


Figure 3.15: It demonstrates the importance of maintaining an adequate number of samples in regions with low uncertainty reduction. In the figure, the regions with high uncertainty reduction are obstructed by obstacles, therefore a path was found which most of the time passes through regions with low uncertainty reduction (away from beacons). The red color ellipses show the uncertainty at waypoints along the path.

In Fig 3.13 (a and b), we observe that RRBT-LAS reduces the planning time significantly as a result of our localization aware sampling strategy. This can be seen from the graphs as we move from 200 input samples to 10000 input samples, split over two sub-plots due to range of horizontal scales. We also observed that the run time savings increase supra linearly with the number of input samples. We talk about plot in Fig 3.13 (c) at the end of this section where we discuss the utility of our sampling strategy for non-incremental stochastic planners.

Furthermore, we compared the quality of paths generated by RRBT-TF and RRBT-LAS. We used two comparison metrics: (a) trace of covariance matrix at goal, (b) normalized sum of trace of covariance matrices along path. For worst case scenario, where the only way for the robot is to pass through regions with low uncertainty reduction (as shown in Fig 3.15), we observed little compromise on the quality of paths generated by RRBT-LAS as compared to uniform sampling of RRBT-TF. Plots in Fig 3.14 show the comparison. At 1000 input samples, RRBT-LAS has 10% more uncertainty along path and 5% more uncertainty at goal. This is the highest degradation in path quality that we observed as we

move from 200 input samples to 10000 input samples. Note that the path quality saturates at about 7000 samples for the considered case. The compromise (even if it is relatively small) on path quality comes from the fact that, for this scenario, the entire path passes through regions with low uncertainty reduction where we reduce the number of samples. It is reasonable to expect that for a large majority of scenarios, only portions of a path will pass through regions with low uncertainty reduction, hence the path quality compromise would be even smaller. This is the key reason that our localization aware sampling strategy simply accepts all the samples within regions with high uncertainty reduction (as shown in Fig 3.8 to 3.10).

We also implemented a non-incremental planner, Belief Roadmap (BRM) [26] with our Localization Aware Sampling - we call the resulting planner BRM-LAS. However, our BRM implementation used the uncertainty propagation approach of [27] to propagate uncertainty along the edges (instead of using one-step transfer function approach of BRM). This is because the transfer function approach, although more efficient, assumes maximum likelihood observations along the path and therefore, can not infer the true a-priori probability distributions along the path. Since it is a straight forward modification of BRM, the corresponding pseudo code is not provided in the paper. We compared BRM-LAS algorithm with BRM-TF (original BRM with uniform sampling and uncertainty propagation approach of [27]). Fig 3.13 (c) shows the simulation results. From the plot, we observed that our localization aware sampling strategy reduces the planner run time for non-incremental planners as well (although the saving is not as large as for incremental planners).

### 3.6.2 Simulation Results for RRBT-LAC

We carried out simulations to compare our RRBT-LAC algorithm with RRBT-TF. We demonstrate our approach in two ways: (i) through visualization in Fig 3.16, we show the efficacy of our localization aware connection strategy in reducing the number of edges in the roadmap, and (ii) we use plots in Fig 3.17 to show that our localization aware connection strategy leads to saving in planning time.

Fig 3.16 shows the roadmap and planned path for RRBT-LAC and RRBT-TF as we increase the number of input samples from a seed. The number of edges in the roadmap constructed by RRBT-LAC are significantly reduced as compared to the roadmap of RRBT-TF. This can be observed by comparing the roadmaps in (a), (d) and (b), (e) and (c), (f). From the figure, it is also important to note that the nature of paths planned by RRBT-LAC and RRBT-TF remains same, i.e., our localization aware connection strategy used in RRBT-LAC does not compromise on the quality of path. This is because the edges, which are present in the roadmap of RRBT-TF but not in the roadmap of RRBT-LAC , do not help to reduce the uncertainty along those partial paths and therefore, are removed from the roadmap of RRBT-LAC. Fig 3.17 shows that RRBT-LAC reduces the planning time as a result of our localization aware connection strategy.

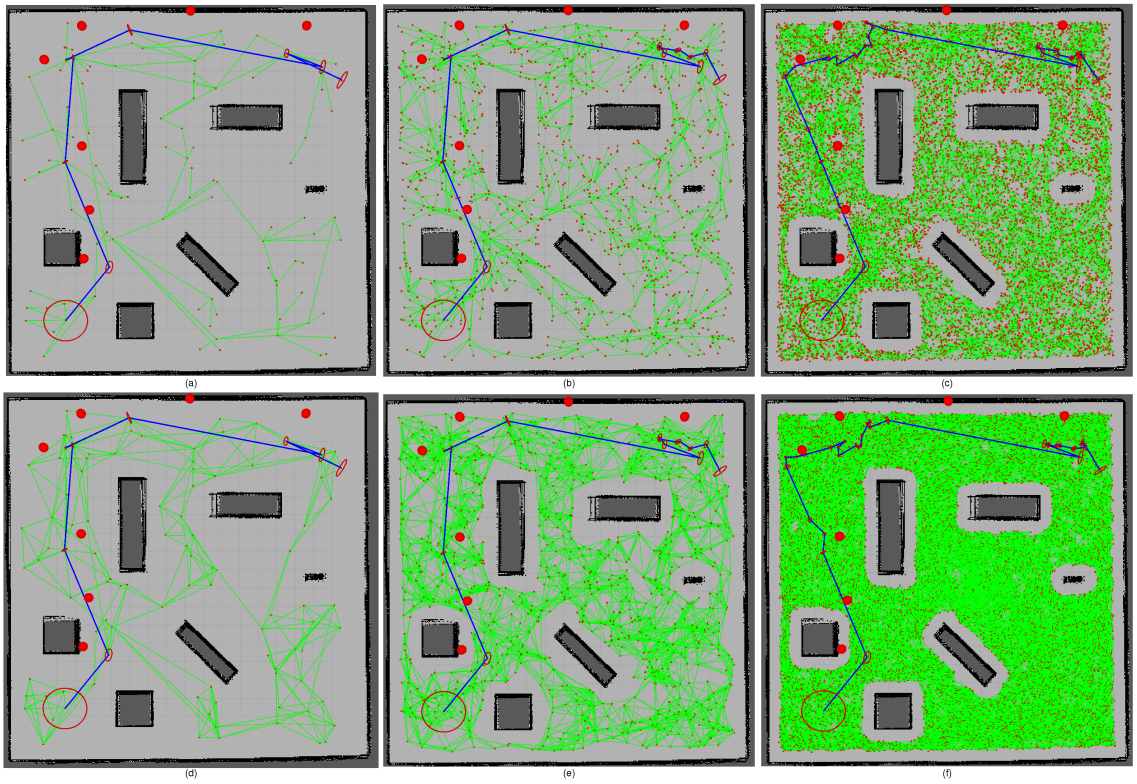


Figure 3.16: RRBT-LAC Vs RRBT-TF. Big red color balls of circular shape represent the beacons. Top row (a, b, c) shows the roadmap and planned path (blue color) for RRBT-LAC where the # [nodes] and # [edges] are: (a) [100] and [121], (b) [1000] and [1338], (c) [10000] and [13373]. The bottom row (d, e, f) is for RRBT-TF where the # [nodes] and # [edges] are: (d) [100] and [308], (e) [1000] and [5007], (f) [10000] and [67405]. Note that the nature of well localized path remains same as we reduce the number of edges (which do not contribute toward better localization) in our RRBT-LAC.

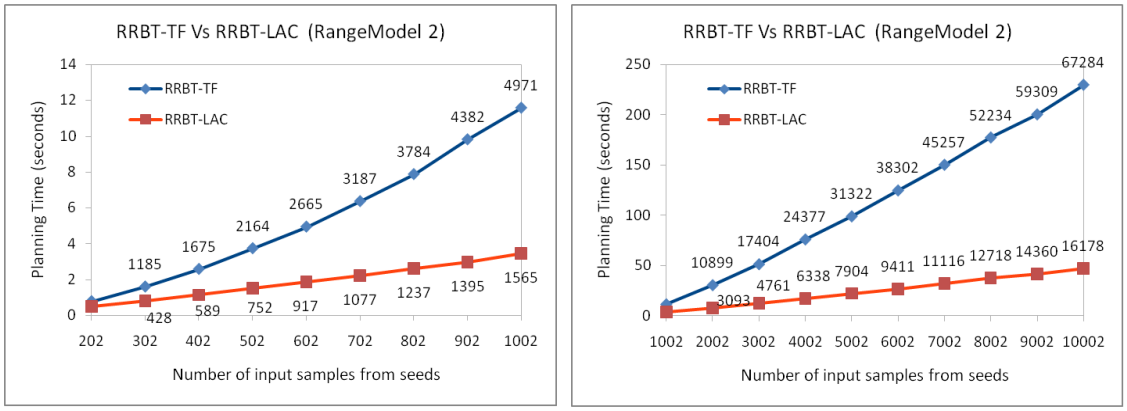



Figure 3.17: Comparison of planning time for RRBT-LAC Vs RRBT-TF. Data labels for each data point along red curves in RRBT-LAC and blue curves in RRBT-TF show the number of edges in the roadmap. Please take a note of y-axis scale while comparing different graphs.

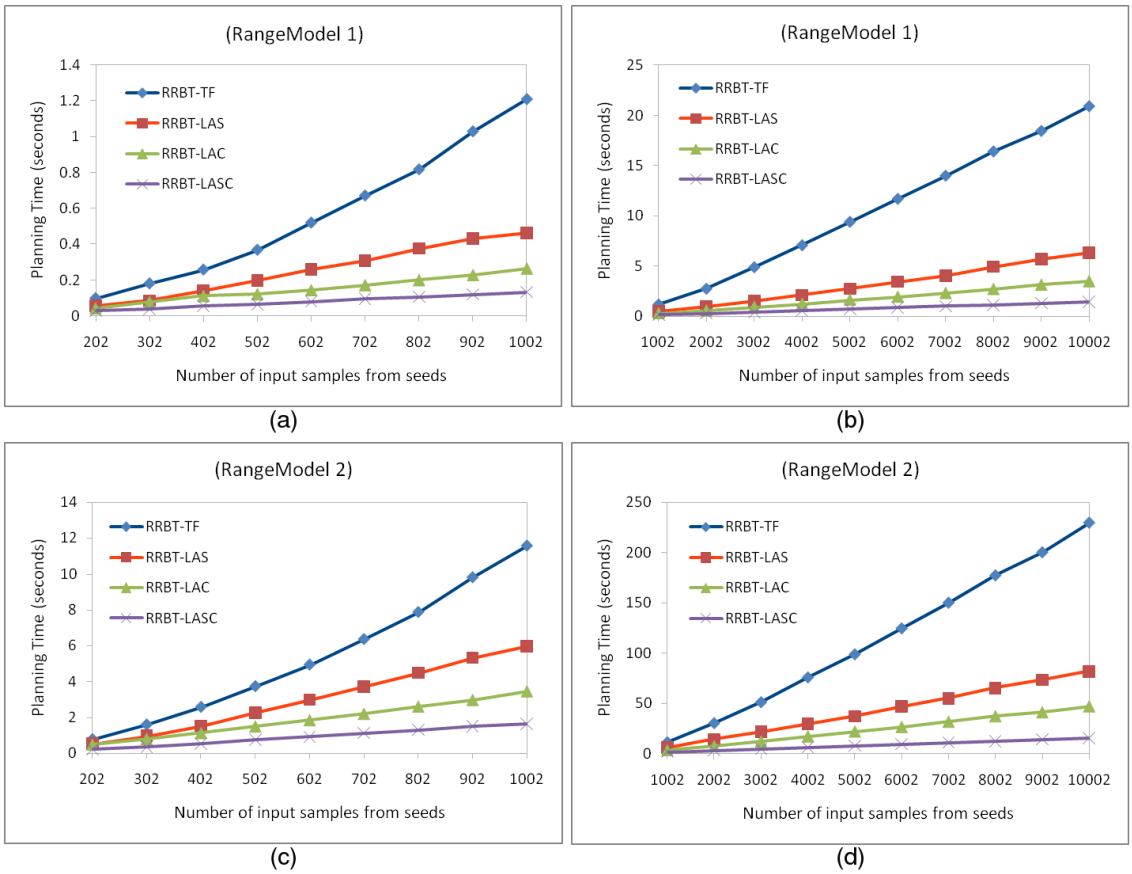



Figure 3.18: Comparison of planning time for RRBT-TF Vs RRBT-LAS Vs RRBT-LAC Vs RRBT-LASC. Please take a note of y-axis scale while comparing different graphs.

### 3.6.3 Simulation Results for RRBT-LASC

Fig 3.18 (c), (d) show the simulation results for the combined effects of the sampling and connection strategies where we replace the sampling and connection strategies of RRBT-TF with our localization aware sampling and connection strategies and call it RRBT-LASC. We observe that the run time savings increase supra linearly (from RRBT-LAS to RRBT-LASC) with the number of input samples.

### 3.6.4 Simulation Results with Different Sensor Model

Additionally, we also evaluated all the planners with a different sensor model (RangeModel 1) where beacons have a limited range (we used 2 meters). We observed similar behaviour as with RangeModel 2. Fig 3.11 and 3.12 show the effect of varying two thresholds (DistTH and LocAbilityTH) in our localization aware sampling strategy. The run time saving from RRBT-LAS to RRBT-LASC is shown in Fig 3.18 (a), (b).

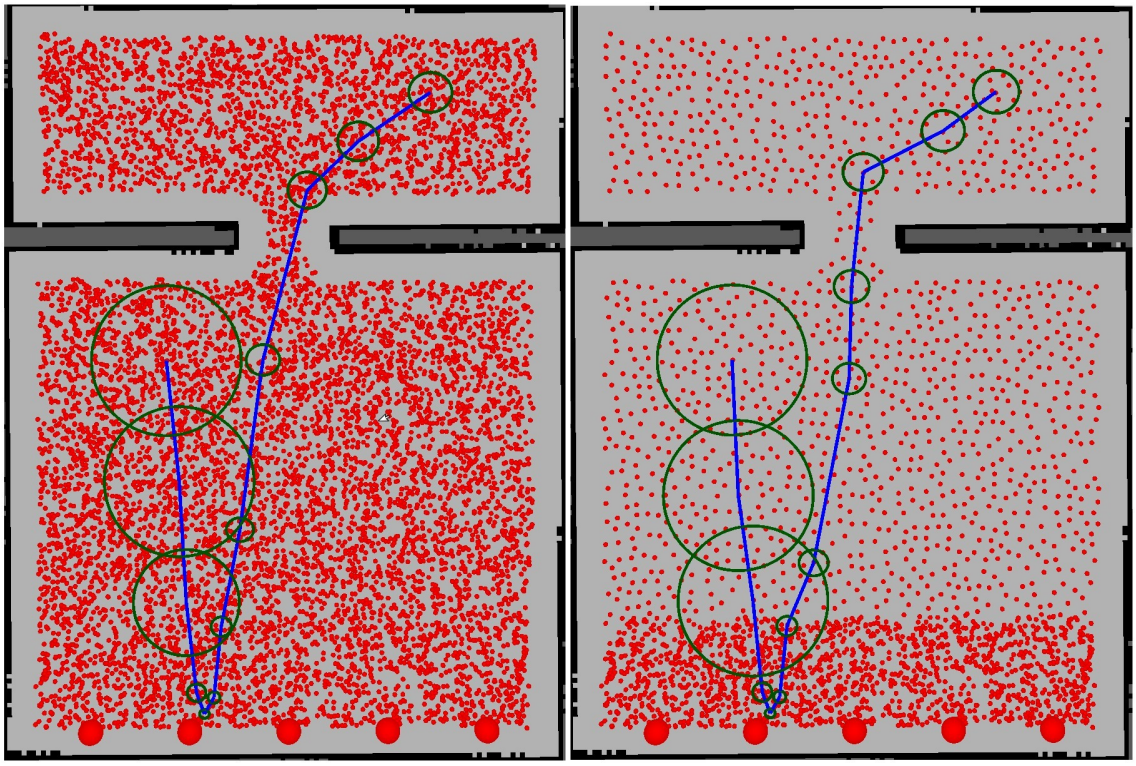


Figure 3.19: These figures show the path planned by RRBT-TF (left) and RRBT-LAS (right) with number of input samples from a seed as 10000. For RRBT-LAS, we used DistTH as 10 cm and LocAbilityTH as 76.6%. Note that sensor measurements are available only within a beacon's range (big red color balls, 5 of them), which is 2 meters in this example.

### 3.6.5 Simulation Results with Replicated Environment

To demonstrate our sampling and connection strategies in a different scenario, we replicated a simple environment used in [28] as shown in Fig 3.19. In the example, the goal is protected by two obstacles, with a narrow gap in between. The initial uncertainty in robot's pose is such that a direct path to the goal may result in a collision. It is therefore necessary for the robot to drive into the well-localized region near beacons to gain sensor measurements. This will reduce the uncertainty in its own position which in turn will decrease the chance of collision while passing through the narrow gap. We provide our simulation results for this scenario in Fig 3.19 to Fig 3.21.

Fig 3.19 shows one of the trials of RRBT-TF and RRBT-LAS after 10000 number of input samples from a seed. The actual number of samples in case of RRBT-TF remains same (10000) while in case of RRBT-LAS, it was reduced to 1893. It is important to note that the nature of the path remains same where sensor data is able to achieve higher uncertainty reduction (near to beacons). Plots in Fig 3.20 show the difference in quality of paths generated by RRBT-TF and RRBT-LAS. The quality slightly degrades in case

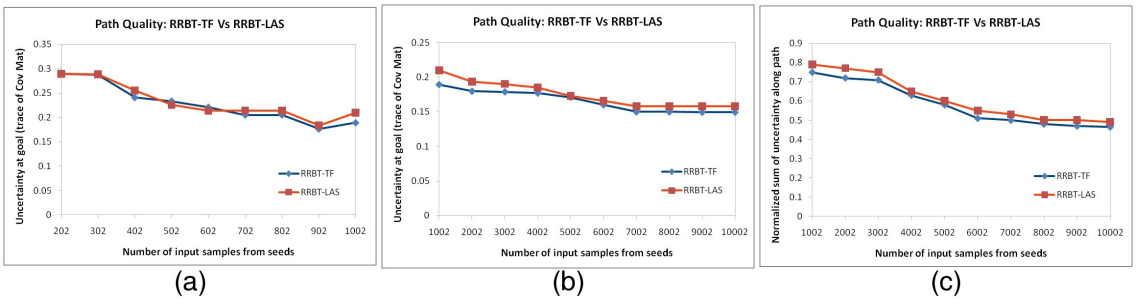



Figure 3.20: Comparison of path quality between RRBT-TF and RRBT-LAS for a scenario shown in Fig 3.19. Y-axis denotes the trace of covariance matrix at goal in plots (a, b) and normalized sum of trace of covariance matrices along path in plot (c).

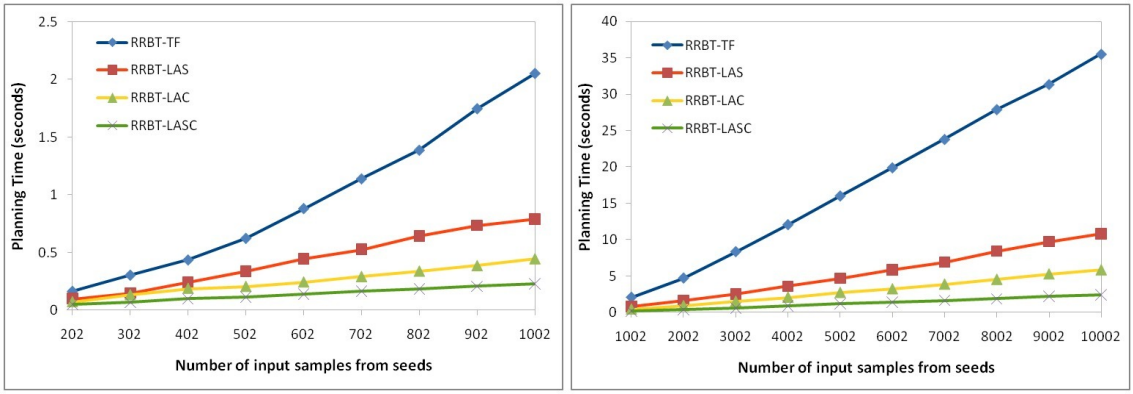



Figure 3.21: Comparison of planning time for RRBT-TF Vs RRBT-LAS Vs RRBT-LAC Vs RRBT-LASC for a scenario shown in Fig 3.19.

of RRBT-LAS mainly because most of the time the path passes through regions where there are no sensor measurements. Therefore, the sampling strategy used in RRBT-LAS limits the number of samples to only one with in DistTH as in no measurement zone the localization ability of two samples remains same. In such scenarios, if we decrease DistTH then the quality will improve but at the cost of computational time. The comparison of planning time is shown in Fig 3.21. We observe that the run time savings increase supra linearly (from RRBT-LAS to RRBT-LASC) with the number of input samples.

We also investigated how effective the computed paths are in getting the robot to reach the goal region reliably. For that, we executed (see Section 2.6.4 for controller detail) each computed path ten times by varying the motion and sensor noises and found that the paths generated by planners that use our smart strategies are as good as that of RRBT-TF, i.e., in all the execution trials the robot successfully reached the goal as shown in Fig 3.22.

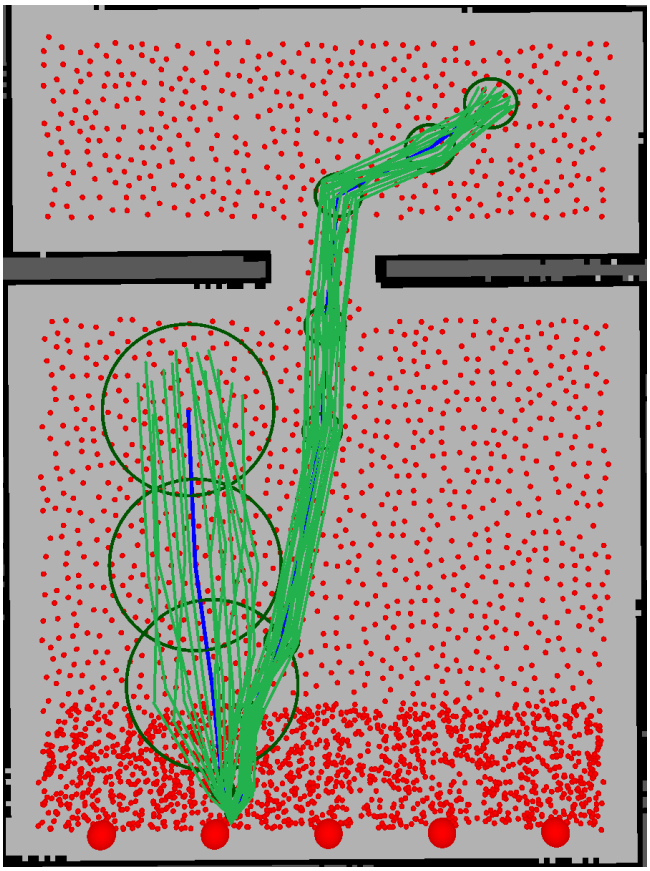


Figure 3.22: Light green color shows the trails of path execution. In this particular example, we executed the planned path 20 times.

# Chapter 4

# Mobile Manipulator Planning II

In this chapter, we learnt from the shortcomings of HAMP-U (as mentioned in Section 1.1) and took advantage of our smart strategies developed to combat the computational complexity involved in motion planning under uncertainty.

We present a sampling-based mobile manipulator planner that considers the base pose uncertainty and the effects of this uncertainty on manipulator motions. The overall planner has three distinct and novel features: i) it uses the Hierarchical and Adaptive Mobile Manipulator Planner (HAMP) that plans for both the base and the arm in a judicious manner, ii) it uses localization aware sampling and connection strategies to consider only those nodes and edges which contribute toward better localization. This helps to reduce the planning time significantly since 3D collision checks - a relatively expensive computation - are carried out along the edges in subsequent stages for the mobile manipulator. iii) it incorporates base pose uncertainty along the edges (where arm remains static) and the effects of this uncertainty are considered on arm motion at node. We call this overall planner HAMP-BAU, where BAU stands for Base Uncertainty and its propagation to Arm motions. We then present an extension of HAMP-BAU to incorporate task space constraint, called HAMP-BAU-TC. We evaluate both planners in known environment and show that our planners find a safer path as compared to other variants where uncertainty is not considered at different levels, for example, not incorporating base uncertainty on manipulator plans, not respecting collision probability threshold along the edges. We also show that variants of these planners that do not use our localization aware sampling and connection strategies will take longer to find the same quality of path.

## 4.1 Overview of HAMP-BAU and HAMP-BAU-TC

The planner constructs a tree by accepting only those nodes and edges which contribute toward better localization. Moreover, it respects the collision probability threshold along the path and uncertainty threshold at goal. The tree construction procedure is as follows:

HAMP-BAU first connects the new sample (sampled using our localization aware sampling strategy) to a nearest node (chosen based on an uncertainty metric and not on distance metric) provided that the edge connecting nearest node to new sample satisfies two conditions - a) collision probability along the edge is below a given threshold, and b) current manipulator configuration at nearest node is collision-free as the mobile manipulator moves along the edge, if not, then there should exists such a reachable reconfiguration. If this connection is successful then only HAMP-BAU tries to rewire the connections to other neighbouring nodes via new sample. The connection from a new sample to a neighbouring node is then rewired if new path to a neighbouring node helps to reduce the uncertainty. Finally, the goal sample is connected to the tree only if above two conditions are met and the uncertainty at goal is below the threshold. We then propose an extension of HAMP-BAU to incorporate task space constraints (we call it HAMP-BAU-TC). In HAMP-BAU-TC, the manipulator reconfiguration paths are computed such that they follow the task space constraints (also called end-effector constraints).

## 4.2 The HAMP-BAU Algorithm

### 4.2.1 Problem Statement

We use $q_i = (q_i^b, q_i^m)$ in $C_{bm}$, the C-space of the mobile manipulator, to represent $i^{th}$ mobile manipulator configuration, where $q_i^b = [x, y, \theta] \in C_b$, the C-space of the mobile base, is the base configuration (also called base pose) and $q_i^m = [\theta_1, \theta_2, ...., \theta_d] \in C_m$, the C-space of the $d$ degree of freedom manipulator, is the manipulator configuration. $C_{b_{free}}$ is the set of all collision-free base poses and $C_{b_{obs}}$ is the set of poses resulting in collision with obstacles. For a given base pose, $q_i^b$, $C_{m_{free}}$ denotes the set of free manipulator configurations (for simplicity we omit the reference to the corresponding base node $q_i^b$ in the notation) and $C_{m_{obs}}$ denotes the set of manipulator configurations that are in collision with obstacles.

Given the start $q_s = (q_s^b, q_s^m)$ and goal $q_g = (q_g^b, q_g^m)$ configurations of the mobile manipulator, the objective of our HAMP-BAU algorithm is to find a collision-free H-path $\Pi^{bm}$ that respects collision probability threshold (CollProbTH) along the path and base pose uncertainty threshold (GoalUnctTH) at goal. Recall that H-path comprises sequential motions of base and arm, i.e., base moves with manipulator in static configuration, followed by a reconfiguration step where manipulator moves while base remains static.

### 4.2.2 General Information

The algorithm operates on a set of nodes $V$ and edges $E$, that define a tree in $C_{bm}$. Each node $v \in V$ has a base pose $v.q^b$, a manipulator configuration $v.q^m$, base pose estimate covariance $v.\Sigma$, a parent node $v.parent$, localization ability $v.\text{loc}$ and reconfiguration path $v.\text{RPATHS}^{[v_{adj}]}$ corresponding to child node $v_{adj}$. The base pose covariance prediction is im-

**Algorithm 10:** HAMP-BAU Algorithm

$v.q^b := q_s^b$; $v.q^m := q_s^m$; $v.\Sigma := \Sigma_s$; $v$.parent $:= \emptyset$; $v$.loc $:= tr(M)$; $v.\text{RPATHS}^{[v_{adj}]} := \emptyset$
$V := \{v\}$; $E := \{\}$
**while** ! TIMEUP **do**
  **if** RNG$(0,1) <$ GOALBIAS **then**
    $(q_{rand}^b, -) :=$ SAMPLEBASEGOAL$(q_g^b)$
  **else**
    $(q_{rand}^b, la) :=$ LOCALIZATIONBIASEDSAMPLE()
  $V_{near} := \{\text{NEAR}(V, q_{rand}^b)\}$; $\Sigma_{rand} := \emptyset$
  **for all** $v_{near} \in V_{near}$ **do**
    $e_{near} :=$ CONNECT$(v_{near}.q^b, q_{rand}^b)$
    **if** COLLISIONFREE2D$(e_{near})$ **then**
      $\Sigma' :=$PROPAGATE$(e_{near}, v_{near}.\Sigma)$
      **if** $tr(\Sigma') < tr(\Sigma_{rand})$ ***or*** $tr(\Sigma_{rand})$=0 **then**
        $v_{nearest} := v_{near}$; $e_{nearest} := e_{near}$
        $\Sigma_{rand} := \Sigma'$
  **if** $tr(\Sigma_{rand}) \neq 0$ **then**
    $v' := \{q_{rand}^b, -, \Sigma_{rand}, v_{nearest}, la, -\}$
    $(q_{new}^m, \pi^m) :=$SEARCHRPATH$(v_{nearest}, v')$
    **if** $\pi^m \neq \emptyset$ **then**
      $cp :=$ COMPUTECP$(v_{nearest}, e_{nearest}, q_{new}^m)$
      **if** $cp <$ COLLPROBTH **then**
        **if** *goal sample* **then**
          **if** $tr(\Sigma_{rand}) <$ GOALUNCTTH **then**
            **if** RPATHATGOAL$(q_{new}^m, q_g^m)$ **then**
              **return** *H-path* $\Pi^{bm}$
            Continue (go to step 4)
          Continue (go to step 4)
        $v'.q^m := q_{new}^m$; $v_{nearest}.\text{RPATHS}^{[v']} := \pi^m$
        $V := V \cup v'$; $E := E \cup e_{nearest}$
  **if** *node* $v'$ *is added* **then**
    **for all** $v_{near} \in V_{near} \setminus v_{nearest}$ **do**
      $e_{near} :=$ CONNECT$(v'.q^b, v_{near}.q^b)$
      **if** COLLISIONFREE2D$(e_{near})$ **then**
        $\Sigma' :=$PROPAGATE$(e_{near}, v'.\Sigma)$
        **if** $tr(\Sigma') < tr(v_{near}.\Sigma)$ **then**
          $(q_{new}^m, \pi^m) :=$SEARCHRPATH$(v', v_{near})$
          **if** $\pi^m \neq \emptyset$ **then**
            $cp =$ COMPUTECP$(v', e_{near}, q_{new}^m)$
            **if** $cp <$ COLLPROBTH **then**
              $v_{parent} :=$Parent$(v_{near})$
              $v_{near}$=$\{-, q_{new}^m, \Sigma', v', -, -\}$
              $v'.\text{RPATHS}^{[v_{near}]} := \pi^m$
              $E$=$E \setminus (v_{parent}, v_{near}) \cup e_{near}$
              UPDATETREEBRANCH$(v_{near})$

**Algorithm 11:** $(q^m_{new}, \pi^m_n) = \text{SEARCHRPATH}(n, n')$

**Input**: nodes $n, n'$ along an edge $e_{n,n'}$
**Output**: reconfiguration path $\pi^m_n$ at node $n$ with $n.q^m$ as start and $q^m_{new}$ as goal such that $q^m_{new}$ is collision-free along an edge $e_{n,n'}$

```
n'' := (n'.q^b, n.q^m)
if CollisionFree3D(e_{n,n''}) then
    q^m_new ← n.q^m and π^m_n ← {n.q^m}
else
    goal^m := ComputeArmGoals(n, n', K_Goals)
    while ! ArmPlanningTimeUp and ! TimeUp do
        Search for a manipulator path from n.q^m to one of the goal configurations in goal^m using a planner that considers base pose uncertainty.
        π^m_n :=LazyCPC-PRM(n.q^b, n.Σ, n.q^m,goal^m[i])
        q^m_new ←goal^m[i]
return (q^m_new, π^m_n)
```

**Algorithm 12:** $cp = \text{COMPUTECP}(n, e, q^m_{new})$

```
Uniformly sample k particles from Gaussian distribution with mean n.q^b and
covariance n.Σ such that weight (probability) of i^th particle is w_i
while i < k do
    Compute collision status c_i by simulating the actions along e starting at i^th
    particle instead of n.q^b with manipulator in configuration q^m_new. Note: collision
    free (c_i = 0) or in collision (c_i = 1).
    cp := cp + (c_i × w_i)
```

plemented by a $\text{PROPAGATE}(e, \Sigma)$ routine that takes as arguments an edge and a covariance matrix at starting node for that edge, and returns a covariance matrix at the ending node. Routine LOCALIZATIONBIASEDSAMPLE() outputs a random sample and its "localization ability". Localization ability of a sample reflects just the sensor ability to gather accurate information and not the actual localization uncertainty at the sample. The trajectory between two states can be computed by routine CONNECT(). In our case, both simulation and real experiments assume holonomic robot to demonstrate our planners. However, if the system dynamics requires nominal trajectory and stabilizing controllers (which is beyond the scope of this thesis) then it can be accommodated in this routine.

We also require the following routines: $\text{NEAR}(V, v.q^b)$ returns every node within some ball centered at $v.q^b$ of radius $\rho \propto (\log(n)/n)^{1/d^b}$ where $n$ is the number of nodes and $d^b$ is the dimension of $C_b$ (See [54]). Note that NEAR() uses Euclidean metric in $C_b$. RNG() is a random number generator that generates a number distributed uniformly over an interval.

### 4.2.3 Algorithm Description

The HAMP-BAU algorithm is described in Algorithm 10. The tree is initialized with a single node with base pose $q_s^b$, manipulator configuration $q_s^m$, covariance $\Sigma_s$ and its localization ability $tr(M)$ (trace of matrix $M$) from lines 1-2. Note that the localization ability of a sample is stored at the node and this information will be used by our localization aware sampling strategy and not anywhere else in the algorithm. To know how it is used, we refer the reader to Section 3.3.2.

At each iteration of the while loop, the tree is updated by adding a new sample. If the bias is not toward goal (line 6), a new base pose $q_{rand}^b$ is sampled using our localization aware sampling strategy (line 7) and then connected to the nearest node. To connect the new sample to a nearest node and other neighbouring nodes (rewiring), we use our localization aware connection strategy. The nearest node is chosen based on uncertainty metric and not distant metric as described from lines 8-15. For each neighbouring node within a ball, the uncertainty is propagated from it to the new sample given that the corresponding path is collision-free. At this stage only 2D collision checks are performed, i.e. base footprint is checked with 2D representation of the environment. The neighbouring node which gives minimal uncertainty at the new sample is selected as nearest node.

The connection from nearest node to the new node $v'$ is made only if the local base path ($e_{nearest}$) connecting nearest node and new node satisfies two conditions: a) there exists a reconfiguration path at nearest node such that the resultant manipulator configuration is collision-free as the mobile manipulator moves along the local base path, b) the collision probability along the local base path is below the given threshold. This is explained from lines 16-29 (excluding lines 22-27 which are for goal sample and explained later). The reconfiguration path condition is checked in lines 18-19 where a routine SearchRPath(), described in Algorithm 11, searches for a reconfiguration path that considers base pose uncertainty. While the second condition is checked in lines 20-21 where a routine ComputeCP(), described in Algorithm 12, computes the collision probability of the mobile manipulator motion along the local base path with manipulator in last configuration of the reconfiguration path. If both the conditions are satisfied then the information is updated at nearest and new nodes (line 28) and the new node $v'$ and corresponding edge are added to the tree (line 29).

If the new node $v'$ is successfully added then the HAMP-BAU algorithm rewires the connection to other neighbouring nodes, i.e. if the new path to a neighbouring node via new node gives less uncertainty then the new path is retained (edge connecting new node to a neighbouring node is added) while the edge connecting a neighbouring node to its parent node (in the old path) is deleted. The rewire connection from a new node to a neighbouring node is made only if following conditions are satisfied: a) new path is less uncertain, b) there exists a reconfiguration path, and c) the collision probability is below the threshold. The

rewire connection procedure is explained from lines 30-44. The first condition is checked in lines 34-35 while the other two conditions are checked in lines 36-37 and lines 38-39, respectively. If all the conditions are satisfied then rewiring is done from lines 40-43. After rewiring a neighbouring node, the tree branch from that node onwards is updated using a routine UpdateTreeBranch(). This includes reconfiguration paths, collision probability checks and uncertainty propagation (line 44).

If the bias is toward goal (line 4), then HAMP-BAU tries to connect the base goal pose to the tree. The chance of this happening in 5% as we use GoalBias = 0.05. Same treatment is carried out to this goal sample up to line 21, i.e., collision checks are performed, reconfiguration path is searched and collision probability is checked. Thereafter, HAMP-BAU ensures that the uncertainty achieved at base goal pose is below the threshold (line 23). Note that because of reconfiguration steps along the path, the achieved manipulator configuration at base goal pose is different from the desired one $q_g^m$. As a result of that routine RPathAtGoal() is used to reconfigure the manipulator to $q_g^m$. If above conditions are satisfied then H-path is returned else the while loop continues till the permitted time to compute the path is over.

### 4.2.4 Reconfiguration Path

The reconfiguration path search is implemented in SearchRPath() which is described in Algorithm 11. This routine is similar to the one that we defined for HAMP in Section 2.3 except with one modification, i.e., instead of using a traditional manipulator planner, we use a Lazy-CPC-PRM [33] that considers base pose uncertainty. In brief, if the current manipulator configuration is collision-free along the edge then reconfiguration step is not required (lines 1-4), else collision-free (along edge) manipulator configurations are searched using a routine ComputeArmGoals(). Then Lazy-CPC-PRM is used to plan a path from current manipulator configuration to any of the configurations computed by ComputeArmGoals().

In routine RPathAtGoal(), we do not need to search a manipulator configuration to plan a path for as we already know that the manipulator needs to be reconfigured to $q_g^m$. Therefore, Lazy-CPC-PRM is used to plan a path from current manipulator configuration to $q_g^m$.

### 4.2.5 Collision Tests

3D collision checks are performed in routines SearchRPath(), RPathAtGoal() and routine ComputeArmGoals(). Our way of performing 3D collision checks is different: first we check the 2D footprint of the mobile base with 2D representation of the world, if that is collision-free then only the manipulator model is checked with 3D representation of the world (not the mobile manipulator model). While in CollisionFree2D(), only 2D collision

checks are performed. The specifics of collision checks and world representation is to be described in later section.

### 4.2.6 Collision Probability

Collision probability is an important metric that tells about the vulnerability (to collisions if robot deviates) of a path, which in turn helps to select a safer path. This is even more important for a mobile manipulator planner where manipulator keeps changing its configuration as the mobile manipulator moves along a path. A slight deviation can lead to collision even in less cluttered environment.

There are a few ways to compute the collision probability (say along an edge with uncertainty ellipses at start and end vertices): (i) [25] samples particles from both the ellipses and perform collision checks along local paths obtained from "all such pairwise paths". If $n$ particles are sampled from each ellipse then there will be $n^2$ paths to check. Therefore the complexity is of the order of $O(n^2)$, (ii) [27] uses ellipse transformation approach and it was for the simple case of a disc robot, (iii) monte-carlo approach - where the edge is discretized and for each point along the edge the corresponding uncertainty ellipse is obtained. For each ellipse, particles are sampled and then collision status is checked at each particle. That gives the collision probability of one ellipse. For an edge, all the collision probabilities along the discretized edge are multiplied (hence complexity of $O(kn)$ where $k$ is the number of discretized points along the edge, and $n$ the number of particles as above), (iv) [29] uses action simulation approach, i.e., the same sequence of actions (used to travel along an edge) is simulated at each particle obtained from uncertainty ellipse at start vertex. The collision probability is then sum of weights corresponding to the particles that result in collision. Recall that, in our case, we compute the sequence of actions as the rotation required to orient the robot along the edge, followed by the translation required to reach the other end of the edge. However, in general, there could be a non-straight line path between two nodes, in that case the corresponding sequence of actions should be considered. The complexity of this approach is of the order of $O(n)$.

Since first and third approaches are computationally more expensive, as indicated by their respective complexity as mentioned above, and especially for a mobile manipulator because of 3D collision checks this may result in quite long computation times. The second approach was applied to the simple case of a disc robot. Therefore, we use the fourth approach. The collision probability computation using this approach is described in Algorithm 12.

## 4.3 The HAMP-BAU-TC Algorithm

In this section, we extend our HAMP-BAU algorithm to incorporate task space constraints. We call the resultant planner as HAMP-BAU-TC.

**Algorithm 13:** HAMP-BAU-TC Algorithm

$v.q^b := q_s^b$; $v.p^e := p_s^e$; $v.\Sigma := \Sigma_s$; $v$.parent $:= \emptyset$; $v$.loc $:= tr(M)$; $v.\text{RPATHSTS}^{[v_{adj}]} := \emptyset$
$V := \{v\}$; $E := \{\}$
**while** ! TIMEUP **do**
  **if** RNG$(0,1) <$ GOALBIAS **then**
    $(q_{rand}^b, -) :=$ SAMPLEBASEGOAL$(q_g^b)$
  **else**
    $(q_{rand}^b, la) :=$ LOCALIZATIONBIASEDSAMPLE()
  $V_{near} := \{\text{NEAR}(V, q_{rand}^b)\}$; $\Sigma_{rand} := \emptyset$
  **for all** $v_{near} \in V_{near}$ **do**
    $e_{near} :=$ CONNECT$(v_{near}.q^b, q_{rand}^b)$
    **if** COLLISIONFREE2D$(e_{near})$ **then**
      $\Sigma' :=$PROPAGATE$(e_{near}, v_{near}.\Sigma)$
      **if** $tr(\Sigma') < tr(\Sigma_{rand})$ ***or*** $tr(\Sigma_{rand})$=0 **then**
        $v_{nearest} := v_{near}$; $e_{nearest} := e_{near}$
        $\Sigma_{rand} := \Sigma'$
  **if** $tr(\Sigma_{rand}) \neq 0$ **then**
    $v' := \{q_{rand}^b, -, \Sigma_{rand}, v_{nearest}, la, -\}$
    $(p_{new}^e, \pi^{ts}) :=$SEARCHRPATHTS$(v_{nearest}, v')$
    **if** $\pi^{ts} \neq \emptyset$ **then**
      $cp=$COMPUTECPTS$(v_{nearest}, e_{nearest}, p_{new}^e)$
      **if** $cp <$ COLLPROBTH **then**
        **if** *goal sample* **then**
          **if** $tr(\Sigma_{rand}) <$ GOALUNCTTH **then**
            **if** RPATHTSATGOAL$(p_{new}^e, p_g^e)$ **then**
              **return** *H-path* $\Pi^{bm}$
            Continue (go to step 4)
          Continue (go to step 4)
        $v'.p^e = p_{new}^e$; $v_{nearest}.\text{RPATHSTS}^{[v']} = \pi^{ts}$
        $V := V \cup v'$; $E := E \cup e_{nearest}$
  **if** *node* $v'$ *is added* **then**
    **for all** $v_{near} \in V_{near} \setminus v_{nearest}$ **do**
      $e_{near} :=$ CONNECT$(v'.q^b, v_{near}.q^b)$
      **if** COLLISIONFREE2D$(e_{near})$ **then**
        $\Sigma' :=$PROPAGATE$(e_{near}, v'.\Sigma)$
        **if** $tr(\Sigma') < tr(v_{near}.\Sigma)$ **then**
          $(p_{new}^e, \pi^{ts})=$SEARCHRPATHTS$(v', v_{near})$
          **if** $\pi^{ts} \neq \emptyset$ **then**
            $cp=$COMPUTECPTS$(v', e_{near}, p_{new}^e)$
            **if** $cp <$ COLLPROBTH **then**
              $v_{parent} :=$Parent$(v_{near})$
              $v_{near}=\{-, p_{new}^e, \Sigma', v', -, -\}$
              $v'.\text{RPATHSTS}^{[v_{near}]} := \pi^{ts}$
              $E=E \setminus (v_{parent}, v_{near}) \cup e_{near}$
              UPDATETREEBRANCHTS$(v_{near})$

**Algorithm 14:** $(p^e_{new}, \pi^{ts}_n) = \text{SearchRPathTs}(n, n')$

**Input**: nodes $n, n'$ along an edge $e_{n,n'}$

**Output**: reconfiguration path $\pi^{ts}_n$ in task space at node $n$ with $n.p^e$ as start and $p^e_{new}$ as goal such that IK($p^e_{new}$) is collision-free along an edge $e_{n,n'}$

```
n'' := (n'.q^b, n.p^e)
if CollisionFree3D(e_{n,n''}) then
    p^e_new ← n.p^e and π^ts_n ← {n.p^e}
else
    goal^e := ComputeArmGoalsTs(n, n', K_Goals)
    while ! ArmPlanningTimeUp and ! TimeUp do
        Search for a manipulator path in task space from n.p^e to one of the goal
        configurations in goal^e using a planner that considers base pose uncertainty.
        π^ts_n :=LazyCPC-PRMts(n.q^b, n.Σ, n.p^e,goal^e[i])
        p^e_new ←goal^e[i]
return (p^e_new, π^ts_n)
```

**Algorithm 15:** $cp = \text{ComputeCPts}(n, e, p^e_{new})$

```
Uniformly sample k particles from Gaussian distribution with mean n.q^b and
covariance n.Σ such that weight (probability) of i^th particle is w_i
while i < k do
    Compute collision status c_i by simulating the actions along e starting at i^th
    particle instead of n.q^b with manipulator in configuration IK(p^e_new). Note:
    collision free (c_i = 0) or in collision (c_i = 1).
    cp := cp + (c_i × w_i)
```

### 4.3.1 Problem Statement

We use $p_i = (q^b_i, p^e_i)$ in $C \times T$, the combined C-space and T-space (task space), to represent $i^{th}$ sample point, where $q^b_i = [x, y, \theta] \in C_b$, the C-space of the mobile base, is the base pose and $p^e_i = [x, y, z, \alpha, \beta, \gamma] \in T_e$, the T-space of the end-effector, is the end-effector pose.

Given the start $p_s = (q^b_s, p^e_s)$ and goal $p_g = (q^b_g, p^e_g)$ poses of the mobile base and end-effector, the objective of our HAMP-BAU-TC algorithm is to find a collision-free H-path $\Pi^{bm}$ in $C_{bm}$ that respects the task space constraints, collision probability threshold along the path and base pose uncertainty threshold at goal. We define a task constraint as a constraint on the motion of manipulator end-effector.

### 4.3.2 General Information

The algorithm operates on a set of nodes $V$ and edges $E$, that define a tree in $C \times T$. Each node $v \in V$ has a base pose $v.q^b$, end-effector pose $v.p^e$, base pose estimate covariance $v.\Sigma$,

parent node $v.parent$, localization ability $v$.loc and reconfiguration path $v.\text{RPATHS}\text{T}\text{S}^{[v_{adj}]}$ in T-space.

We also require a new routine: IK($p_i^e$) that inputs end-effector pose as argument and return its inverse kinematic solution. While IK($p_i^e, q_{seed}^m$) returns inverse kinematic solution closest to a seed (manipulator configuration).

### 4.3.3 Algorithm Description

The HAMP-BAU-TC algorithm is described in Algorithm 13. The tree is initialized with a single node with base pose $q_s^b$, end-effector pose $p_s^e$, covariance $\Sigma_s$ and its localization ability from lines 1-2. The algorithm operates in a similar fashion as HAMP-BAU but with one key modification. In contrast to HAMP-BAU, where reconfiguration paths are searched in C-space of the manipulator, the HAMP-BAU-TC searches for the reconfiguration paths in T-space of the end-effector. This modification helps to maintain the task space constraints as the mobile manipulator moves along the planned path. To accommodate this we modify our following routines: SearchRPathTs(), RPathTsAtGoal() and ComputeCPts(). First two routines are related to computation of reconfiguration paths and explained in Section 4.3.4. While ComputeCPts() computes the collision probability along an edge and is implemented in Algorithm 15. This is different from ComputeCP() in the sense that its third argument is an end-effector pose and not a manipulator configuration. Therefore, IK() routine is used to compute the inverse kinematic solution and then collision probability is computed for that manipulator configuration (solution). We also rename routine UpdateTreeBranchTs() to ensure that the update of tree branch after rewiring step requires reconfiguration steps in task space rather than manipulator C-space. Once the goal base pose is connected to the tree (line 24, Algorithm 13) then using backtracking the H-path can be computed. However, that path will be in $C \times T$ and not in $C_{bm}$ as reconfiguration steps were carried out in task space. Therefore, first these reconfiguration paths are converted into C-space of manipulator using interpolated inverse kinematic and then an H-path in $C_{bm}$ is returned in line 25. Note that, we also store the seed information (manipulator configuration) at each node in the tree that we do not explicitly mention in the pseudo-code of HAMP-BAU-TC algorithm. This helps to find a IK solution closest to previous manipulator configuration along the edges of the tree and avoids sudden jump in manipulator motion.

### 4.3.4 Reconfiguration Path in Task Space

The reconfiguration path search in task space is implemented in routine SearchRPathTs() which is described in Algorithm 14. This routine works as follows: if the inverse kinematic solution of current end-effector pose is collision-free along the edge then reconfiguration step is not required (lines 1-4), else end-effector poses are searched using a rou-

tine ComputeArmGoalsTs() such that their inverse kinematic solution is collision-free (along edge). Then Lazy-CPC-PRMts, a variant of Lazy-CPC-PRM, is used to plan a path in task space from current end-effector pose to any of the poses computed by ComputeArmGoalsTs(). Note that in Lazy-CPC-PRMts, the technique of Lazy-CPC-PRM [33] (to incorporate base pose uncertainty) can be embedded with ATACE [72] or CBiRRT [73] or [74].

In routine RPathTsAtGoal(), we do not need to search an end-effector pose to plan a path for as we already know such a pose $p_g^e$. Therefore, Lazy-CPC-PRMts is used to plan a path from current end-effector pose to $p_g^e$.

## 4.4 Simulation Results for HAMP-BAU & HAMP-BAU-TC

In this section, we evaluate both planners in known environments and provide simulation results. Our implementation is in C++ under linux and runs on a Pentium dual core 2.5 Ghz computer with 4GB memory. From the simulations, we want to demonstrate two things: (i) our planners compute safer plans for mobile manipulator albeit at the cost of computational time, and (ii) consideration of our localization aware sampling and connection strategies in both planners helps to find the same quality of path in lesser time.

To prove the first objective, we compared HAMP-BAU and HAMP-BAU-TC (both belong to Level 2) with their variants where uncertainty is not considered at different levels. Note that these variants still make use of our localization aware sampling and connection strategies, however, differ in the consideration of uncertainty along the edges and on manipulator motions. Below, we briefly describe these variants.

1) HAMP-BAU$_0$: This variant does not compute the collision probability along the edges and therefore, the corresponding threshold (CollProbTH) is not maintained. Also, the effects of base pose uncertainty on manipulator motions (reconfiguration paths) are not considered, i.e., instead of Lazy-CPC-PRM, a standard manipulator planner is used as in HAMP. However, the uncertainty is still propagated along the edges and the uncertainty threshold at goal is still maintained. In short, it considers Level 1 uncertainty.

2) HAMP-BAU$_1$: This variant maintains the collision probability threshold along the edges and uncertainty threshold at goal, however, the effects of base pose uncertainty are not considered on manipulator motions. It also considers Level 1 uncertainty but with an additional feature of collision probability.

Similar to the variants of HAMP-BAU, we do have variants for HAMP-BAU-TC. In HAMP-BAU-TC$_0$, both collision probability threshold and effects of base pose uncertainty on manipulator motions are not considered. While in HAMP-BAU-TC$_1$, the former is considered

Table 4.1: Comparison of HAMP-BAU and its variants (600 Runs).

| | HAMP-BAU$_0$ | | HAMP-BAU$_1$ | | HAMP-BAU | |
|---|---|---|---|---|---|---|
| | T. (sec) | P.C. | T. (sec) | P.C. | T. (sec) | P.C. |
| B | 8.2 | 15.0% | 11.5 | 6.6% | 31.1 | 0.8% |
| C | 26.4 | 53.3% | 28.0 | 45.0% | 52.8 | 6.6% |
| E | 58.5 | 48.0% | 65.0 | 32.0% | 168.0 | 5.2% |

Table 4.2: Comparison of HAMP-BAU-TC and its variants (600 Runs).

| | HAMP-BAU-TC$_0$ | | HAMP-BAU-TC$_1$ | | HAMP-BAU-TC | |
|---|---|---|---|---|---|---|
| | T. (sec) | P.C. | T. (sec) | P.C. | T. (sec) | P.C. |
| B$'$ | 9.4 | 13.2% | 13.0 | 5.1% | 37.4 | 0.3% |
| C$'$ | 34.6 | 50.0% | 37.2 | 43.3% | 68.5 | 5.8% |
| E$'$ | 79.2 | 44.3% | 89.0 | 29.5% | 216.8 | 4.6% |

but not the later. A key reason to compare HAMP-BAU (and HAMP-BAU-TC) with its two variants, instead of just HAMP-BAU$_0$, is also to show how the planner runtime increases as we incorporate the uncertainty at different levels and which component of the uncertainty consideration contributes what amount of increase in runtime.

We evaluated HAMP-BAU and its variants in 3 different scenarios with varying level of complexity in simulation and compared the outcomes. Simulation environments corresponding to scenarios B, C, E are shown in Fig 2.7. The HAMP-BAU-TC and its variants are also evaluated on these scenarios but with a different task at hand. Instead of carrying a long stick as payload (as in B, C, E), the mobile manipulator in scenarios B$'$, C$'$, E$'$ was required to carry a bottle while maintaining a task constraint, i.e., keeping the bottle upright within a threshold of 10 degrees rotation along x and y axes as the mobile manipulator moves along a planned path.

For each scenario, we ran a planner 30 times to get 30 different paths and measured two properties out of them. One is the average planner runtime and second is the quality (in term of safety) of the computed path, i.e. if this path is executed then what is the probability that the mobile manipulator may collide with the obstacles. To compute the second property, we further executed each path 20 times by artificially generating the noise (10% and 15%) in control commands and sensor measurements. Therefore, for a planner (in a scenario), we monitored the number of times a collision has occured among 600 runs (30×20). These two properties are recorded for HAMP-BAU, HAMP-BAU-TC (and their variants) in Tables 4.1 and 4.2, respectively. Acronym “P.C." in the tables denotes percentage collision which is our second property. It tells the % of times a path results in collision.

From the Table 4.1, we observe that HAMP-BAU$_0$ takes less time to plan a path but at the cost of sacrificing the path safety. Paths generated by HAMP-BAU$_0$ are highly prone to collisions. This is because at planning stage it does not respect the collision probability threshold along the edges and also the effects of base pose uncertainty are not considered on

manipulator motions. As compared to scenario B, which is relatively a simple environment, the scenarios in C and E are complex and therefore, require highly safer paths for collision-free executions. That is why P.C. of paths generated for these scenarios is even higher. As we move toward HAMP-BAU$_1$, which does consider collision probability along the edges, the planning time increases by small amount along with the improvements in collision status, i.e., paths generated by HAMP-BAU$_1$ are less prone to collisions as compared to HAMP-BAU$_0$. Note that collision probability computation along the edges adds to planner runtime but then also gives safer plans. For scenario B, paths are 50% safer (as P.C. decreased from 15.0 to 6.6) but that is not the case with scenarios C and E. For C, not much can be done with collision probability alone as the important task (of passing a long stick through the window) is achieved at base goal pose through reconfiguration path and HAMP-BAU$_1$ does not consider uncertainty on manipulator motions. While for E, P.C. is reduced a lot while the rest can be achieved by consideration of uncertainty on reconfiguration paths. Finally, as we move toward HAMP-BAU, we can observe that the planner runtime is increased by 2 to 4 times while path safety is significantly improved. The increase in runtime by that amount is mainly due to Lazy-CPC-PRM, a manipulator planner that computes reconfiguration paths by considering base pose uncertainty. Separately, it is known that Lazy-CPC-PRM takes 2 to 20 seconds to compute a manipulator path depending on a threshold used there and the complexity of the environment. And for scenarios B, C and E, the average number of reconfiguration steps required were 5, 4, 20, respectively. That explains the increase in runtime. Similarly, the Table 4.2 can be analyzed. In addition, it is important to note two points. One is the complexity involved in task space planning that increases the planner runtime. And second is as compared to B, C, E, where tasks involved carrying a 50cm long stick (thus increases the chance of collision if base slightly deviates), the tasks in scenarios B′, C′, E′ involve carrying a bottle, therefore, P.C. in Table 4.2 is reduced as compared to P.C. in Table 4.1.

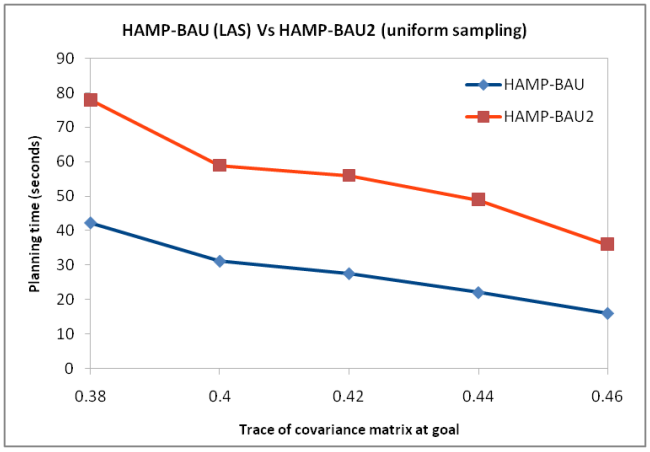



Figure 4.1: HAMP-BAU Vs HAMP-BAU$_2$ by varying GoalUnctTH while CollProbTH remains as 0.08 (for scenario B).

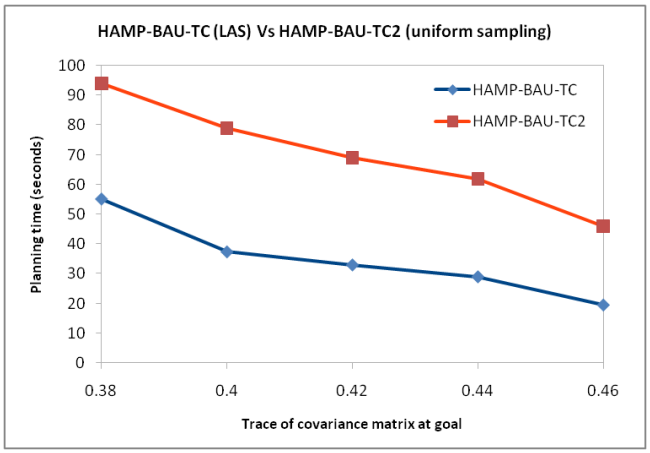



Figure 4.2: HAMP-BAU-TC Vs HAMP-BAU-TC$_2$ by varying GoalUnctTH while CollProbTH remains as 0.08 (for scenario B′).

To prove our second objective, we compared HAMP-BAU and HAMP-BAU-TC with another variants where localization aware sampling strategy and a part of localization aware connection strategy were not used. In HAMP-BAU$_2$ and HAMP-BAU-TC$_2$, a uniform sampling is used and the new sample is connected to the nearest node based on Euclidean distance and not on uncertainty based metric as in HAMP-BAU and HAMP-BAU-TC. Note that both these variants still make use of rewiring notion. For our simulations, we kept the collision probability threshold fixed (0.08) while varying the uncertainty threshold at goal. We ran each planner 30 times (for each GoalUnctTH) and averaged the planner runtime. The results are provided in Figures 4.1 and 4.2 for scenarios B and B′, respectively. From the plots, it can be observed that HAMP-BAU and HAMP-BAU-TC take less time to plan the same quality of path as a result of our smart strategies. On the other hand, their variants were also successful in finding a path that respects corresponding thresholds but take longer to reach there. It is also important to note that as we decrease the GoalUnctTH (looking for more safer paths), the runtime difference between original planner and its variants increases. Therefore, our localization aware sampling and connection strategies help to reach toward well-localized path in shorter time by picking the right choice of samples and their connections (edges) which is highly useful and needed for mobile manipulator planning as it involves 3D collision checks.

### 4.4.1 World Representation and Collision Checks

We use two types of map representation for collision check for efficiency reasons: a 2D world model (costmap) and a global 3D world model (collision map). The 2D world model is obtained by projecting the 3D map up to a certain height. At this stage, for evaluation of HAMP-BAU and HAMP-BAU-TC in known environment, it can be assumed that the 2D and 3D maps are known (provided). Later on, when we discuss about pick-and-place

tasks in unknown environment (in Chapter 5), we explain in detail how these maps are constructed and the nitty-gritty involved. For efficiency reasons, collision detection for the whole mobile manipulator is accomplished in a two-stage process as follows. First, the 2D projected footprint of the base is checked against the 2D map, and if it is collision-free then a 3D collision check is performed on the manipulator only. This strategy helps us avoid unnecessary 3D collision checks (which can be expensive) without being overtly conservative.

Table 4.3: Parameters used in HAMP-BAU and HAMP-BAU-TC.

| Parameter name | HAMP-BAU | HAMP-BAU-TC |
|---|---|---|
| $K_{Goals}$ | 3 | 3 |
| ArmGoalsTimeUp | 2 seconds | 1 second |
| ArmPlanningTimeUp | 6 seconds | 8 seconds |
| GoalUnctTH | 0.4 | 0.4 |
| CollProbTH | 0.08 | 0.08 |
| DistTH | 0.35 m | 0.35 m |
| LocAbilityTH | 83.3% | 83.3% |
| $\delta$ (Lazy-CPC-PRM) | 0.2 | 0.25 |

### 4.4.2 Parameters and Thresholds Values

Table 4.3 shows the key parameters and their corresponding values that we used in HAMP-BAU, HAMP-BAU-TC and their variants. Few of these parameters are visible in the pseudo-code of the algorithms while rest of them are hidden inside few routines and previously developed approaches. For example: ArmGoalsTimeUp is a part of routine ComputeArmGoals(), precise detail of the pseudo-code can be found in Section 2.3. DistTH and LocAbilityTH are two thresholds used in routine LocalizationBiasedSample(), our localization aware sampling strategy. Effects of these two thresholds are well studied in Chapter 3. Lazy-CPC-PRM [33] also uses a threshold ($\delta$) to compute a manipulator path for fixed uncertain base pose. Note that GoalUnctTH and CollProbTH are two different entities, so please do not correlate their values. Parameters CollProbTH and $\delta$, both being collision probability, differ in their values. This is because the former is applied along the edges in the base roadmap (or tree) while the latter is applied to reconfiguration paths in manipulator C-space. These values are empirically chosen. We observed that it is faster to search for an end-effector pose as goal (for reconfiguration path in task space) as compared to a search for a manipulator configuration (for reconfiguration path in manipulator C-space). While on the other hand, it is difficult to find a reconfiguration path in task space as compared to a path in C-space. Therefore, we adjusted ArmGoalsTimeUp and ArmPlanningTimeUp accordingly. One example of GoalUnctTH of 0.4 would be 10 cm (0.1 m) uncertainty each along x and y axes and 11 degree (0.2 radians) along base rotation (.1+.1+.2).

# Chapter 5

# Integrated & Autonomous System for mobile pick-and-place Tasks in Unknown Environments

In this chapter, we incorporate HAMP-BAU and HAMP-BAU-TC in an integrated and autonomous system for mobile pick-and-place tasks in unknown static environments. We provide simulation results as well as real experiments on SFU mobile manipulator. First, we describe our mobile manipulator model and the locations of different sensors, and thereafter, provide an overview, followed by a concise problem statement and then the description of the system will follow. Please note that exploration/next best view is a large area and we basically integrate that in the overall framework. So it is more of a "systems/framework/implementation" in this chapter.

## 5.1 System Components

The SFU mobile manipulator model, both in simulation and physical environment, consists of a powerbot mobile base, 6 DOF schunk powercube arm mounted on the base, 2-finger schunk gripper, LMS100 2D range sensor placed in front of mobile base (used for SLAM), Kinect 3D depth and image sensor mounted on the base, and a light weight hokuyo 2D range sensor mounted on the gripper (to act as eye-in-hand) as shown in Figure 5.1.

### 5.1.1 Why we use 3 different sensors ?

One can argue that the unknown environment can be explored using a single sensor then why the author use 3 sensors in their system. In this section we discuss those possibilities and their shortcomings. The problem with a single 3D sensor, such as Kinect mounted on the base, is that it can not explore regions occluded by objects due to the fixed nature of the sensor as it is mounted on the base. An eye-in-hand sensor (Hokuyo in our case) is

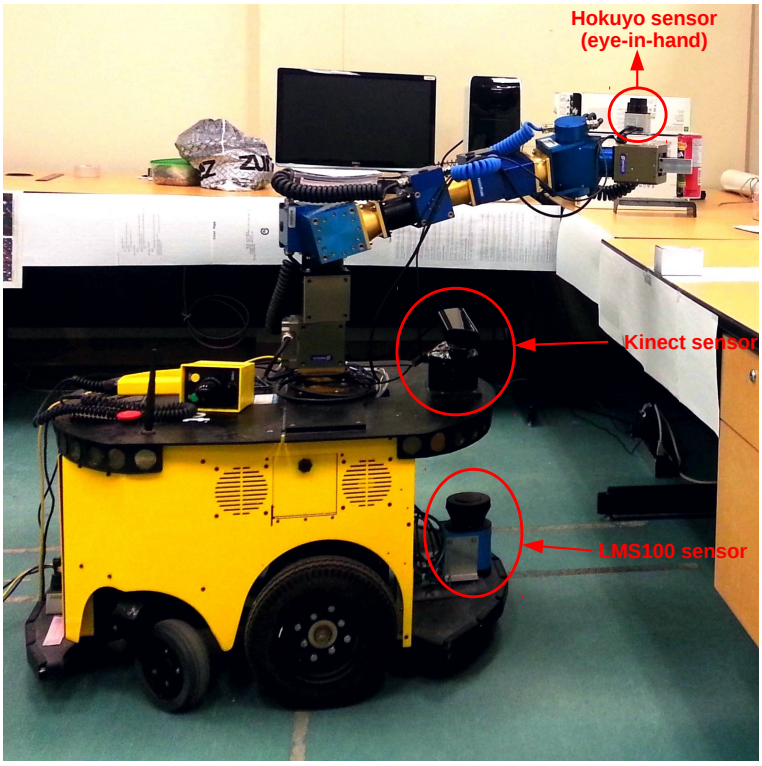



Figure 5.1: The SFU mobile manipulator and mounting locations of different sensors.

better able to explore the environment since it gives more reachability and manoeuvrability to the sensor. Since the eye-in-hand sensor can provide only line scans, therefore, at NBV-A (next best view of arm), the sensor rotates to make an area scan by collecting all the line scans during rotation. The second sensor (Kinect) is added for online monitoring of path execution, however, once it is there it also acts as an additional sensor for exploring the environment. This is also required because Hokuyo does not work well with black surfaces and most of the surfaces in real environment (RAMP Lab) are black. Kinect as eye-in-hand sensor (instead of Hokuyo in order to speed up the exploration) can not sense upto 1 meter distance (near clipping) and therefore, can not scan nearby regions. The third sensor, LMS 100 mounted at the front bottom of the base, is used to localize the mobile base. We could also have used Kinect to localize, but that is computationally quite expensive, hence we use an additional sensor for 2D SLAM.

## 5.2 Overview

A key aspect of our integrated system is that the planner works in tandem with base and arm exploration (view planning) modules that explore the unknown environment. Note that we assume unknown areas of environment as obstacles and not free. The task of base exploration is to take the mobile manipulator (mainly mobile base) to next best view of

the base (NBV-B), take a scan using the Kinect and then invoke arm exploration which scans the local region surrounding the manipulator using the eye-in-hand Hokuyo at next best views of arm (NBVs-A). Scans from both Kinect and Hokuyo sensors are inserted into a global Octomap. The base exploration module works on a 2D occupancy grid map to compute a NBV-B and uses HAMP-BAU to plan a path for it. This 2D occupancy grid map is obtained by the fusion of two 2D maps - one is from down projection of global Octomap upto a certain height and other is from SLAM (simultaneous localization and mapping). The third sensor (LMS100, mounted at base bottom) feeds SLAM. Please see Figure 5.1 for system components and Figure 5.6 for how the sensor information is used and different maps are obtained. On the other hand, the arm exploration module works on a local Voxelmap (obtained from global Octomap) to compute NBVs-A and uses a manipulator planner (that considers base pose uncertainty) to reach there. After each scan, the arm exploration module updates the Voxelmap and repeat the procedure until the Voxelmap is fully explored. It is important to note that since sensor scans are not directly inserted into Voxelmap, therefore, the status (occupied, free, unknown) of each voxel cell in the Voxelmap is updated by communicating with global Octomap. We also want to state that in addition to scans taken by respective sensors at NBV-B and NBV-A, the scans collected during the mobile manipulator motion to reach NBV-B or the manipulator (end-effector) motion to reach NBV-A are also incorporated into the global Octomap.

## 5.3 Objective

Given global pick and place end-effector poses, the task of an integrated and autonomous system is to explore the environment, pick the object once it is in the known region and then explore the remaining environment (if needed) with object in hand, while maintaining task space constraints if any, to place it at the target location (place pose). We assume that a very small region around mobile manipulator is known in the beginning but other than that the entire environment is unknown. In future the grasp would be decided by the system too but for now we give the grasp pose.

## 5.4 System Architecture Description

Our integrated and autonomous system architecture is described in Figure 5.2. The only inputs to the system are global (w.r.t. world frame) pick and place end-effector poses. For each given pose, the corresponding possible base poses and manipulator configurations are computed. Note that at current stage the collision status of these base poses and manipulator configurations can not be verified as the environment is unknown. The method to compute a valid base pose and manipulator configuration corresponding to an end-effector pose is described in [55] and is briefly as follows. First, a base pose is randomly sampled from

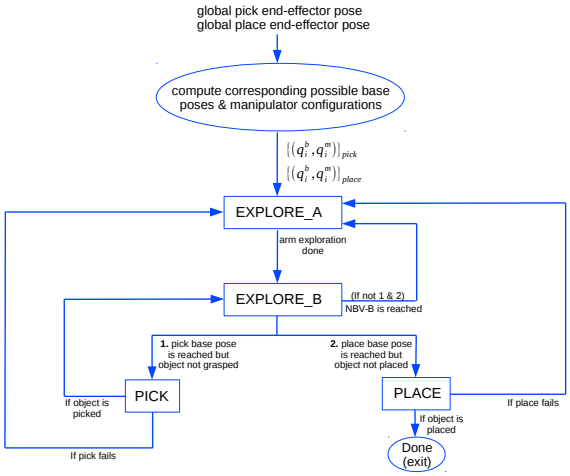



Figure 5.2: An integrated and autonomous system for pick-and-place task in unknown environments.

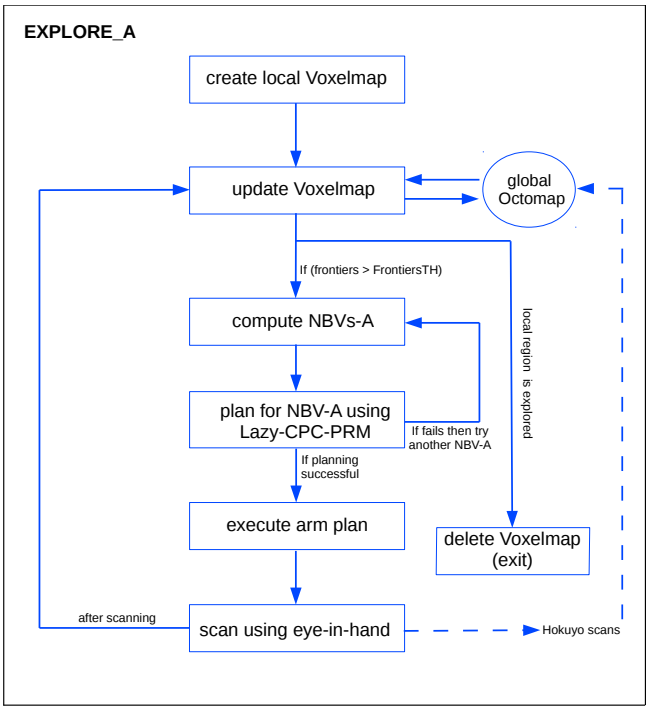



Figure 5.3: EXPLORE_A module of the system.

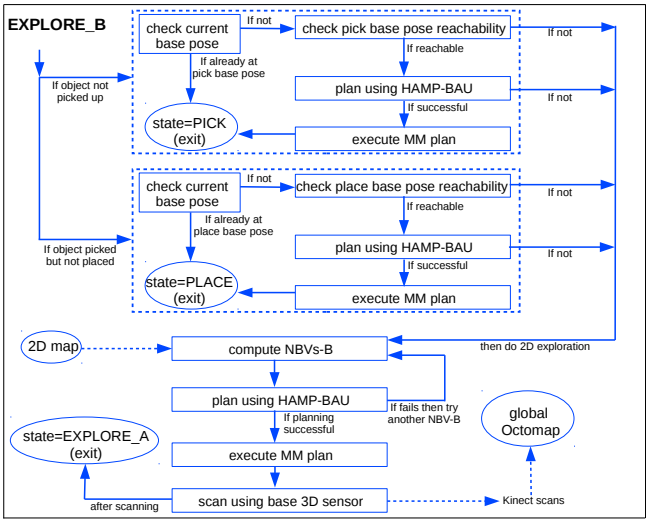



Figure 5.4: EXPLORE_B module of the system.

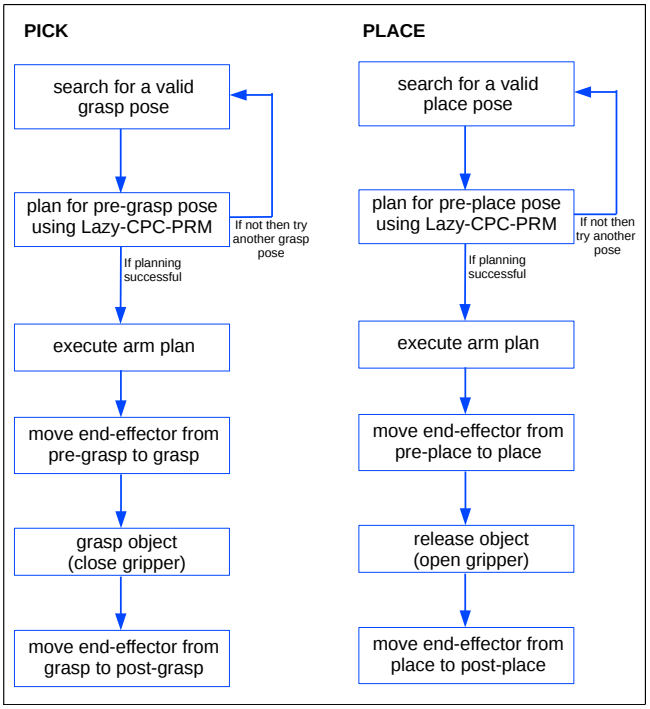



Figure 5.5: Pick and place modules of the system.

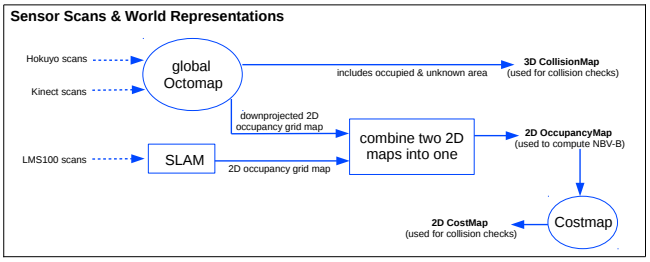



Figure 5.6: Sensor scans and world representations (2D and 3D).

a disk region (bounded by the arm reachability) and then a given end-effector pose (pick or place) is checked for reachability from the sampled base pose using inverse kinematics.

At any instant of time the system can be in one of the four states: EXPLORE_A, EXPLORE_B, PICK and PLACE. In the beginning the system starts in EXPLORE_A. This module uses eye-in-hand sensor to explore the local region around the mobile manipulator (with in the reachability of the end-effector). Once the local region is fully explored, the system changes its state to EXPLORE_B. Broadly, one task of this module is to check for the reachability of pick and place base poses. If any of them is reachable then a path is planned to move the mobile manipulator to that base pose. Depending on success, the state is changed to PICK or PLACE. If none of these pick or place base poses is deemed reachable, then the EXPLORE_B module explores the environment by reaching to a NBV-B and state is then changed to EXPLORE_A. If at some point of time, the state is changed to PICK, i.e., pick base pose is reached, then the PICK module plans for pick end-effector pose and grasps the object. If grasping is successful then the state is switched back to EXPLORE_B to explore the remaining environment to complete the other part of objective, i.e., to place the object. As the system explores more and more environment (using EXPLORE_B and EXPLORE_A), the place base pose will be reachable at some point of time. Once the place base pose is reached then PLACE module is invoked to place the object at target location. The details of these modules are provided from Figures 5.3 to 5.5. We now describe these modules.

EXPLORE_A (see Figure 5.3) creates a local Voxelmap, a 3D version of occupancy grid map, at fixed base pose. Note that the boundaries of Voxelmap should be with in the reachability of end-effector. The status (occupied, free, unknown) of each voxel cell in the Voxelmap is updated by communicating with global Octomap (3D world representation). Then the frontiers (free cells next to unknown) are computed. If the number of frontiers are below a given threshold then the arm exploration is aborted which states that the local region around mobile manipulator is fully explored. If not then arm view planning is invoked to compute NBVs-A which is basically a set of end-effector poses arranged in the priority order of information gain. The end-effector is then moved to arm next best view (A-NBV) to take the scans using eye-in-hand Hokuyo sensor. Note that scans are inserted into the global Octomap and not the local Voxelmap. Lazy-CPC-PRM is used to plan a manipulator path for the IK solution of A-NBV as goal. After scanning, the local Voxelmap is updated and the procedure continues until the Voxelmap is explored or the maximum number of iterations are reached. We use MPV (Maximize Physical Volume) based algorithm for arm view planning as alluded to in Section 1.2.5. The procedure to compute NBVs-A is as follow: arm view planning samples valid (IK exists and the solution is collision-free as well with in joint limits) end-effector poses and computes the information gain at each sensor pose by simulating the sensor model and then returns the set of these poses in the order of high information gain at the top.

EXPLORE_B (see Figure 5.4) first checks, with in the known region of the environment, if it is possible to move the mobile manipulator to pick base pose or place base pose. If the object is not picked yet then the collision status of previously computed pick base poses is checked. If any of the base pose is found to be collision-free then HAMP-BAU is used to plan a path. If the pick base pose is reached (planning is successful) or the mobile base is already at pick base pose then the EXPLORE_B module aborts by switching the state to PICK. Similarly, if the object is grasped but not placed, the reachability of a collision-free place base pose is checked and if successful, the EXPLORE_B module aborts by switching the state to PLACE. However, if the planning fails or pick and place base poses are not collision-free with in the explored region then a base next best view (NBV-B) is reached to explore the environment using sensor (Kinect) mounted on the base. Again, the sensor scans are inserted into the global Octomap. After taking scans, the state is switched to EXPLORE_A to explore the local region, this time from a different base pose. To compute NBV-B, we invoke base view planning where we use frontier-based exploration [75]. The base view planning uses a 2D occupancy grid map to compute NBV-B and this map comes from a series of steps as shown in Figure 5.6.

We now explain PICK and PLACE modules (see Figure 5.5). PICK module is invoked once the mobile manipulator has reached a base pose from where object can be grasped. Previously computed information (base poses and manipulator configurations) corresponding to pick end-effector pose (grasp pose) may not be useful any longer because due to the uncertainty in mobile base position, the mobile manipulator (basically mobile base) does not exactly reach the intended pick base pose. Therefore, either a new manipulator configuration needs to be searched from the reached base pose or a new valid grasp pose (and thereby the corresponding reachable manipulator configuration), with in the close proximity of already given grasp pose, needs to be computed. In case of latter (grasp adjustment), we describe our simplistic approach as follows: Recall that a grasp pose is denoted as $p = [x, y, z, \alpha, \beta, \gamma]$, pose of end-effector frame (located in the center of gripper jaws) with respect to a fixed frame, for example, base frame of the manipulator. In simulation and real experiment examples, the new valid grasp pose is computed by varying $\gamma$ while keeping other 5 parameters same. Varying $\alpha$, $\beta$ will increase the search space. As mentioned in Section 1.1, this is a very quick and an ad hoc attempt to show the system. For a valid grasp pose (collision-free IK solution exists), we also test if its pre-grasp and post-grasp poses are valid ones as well, which are typically 10 cm offset (back and up, respectively) from the intended grasp pose, see [76] for detail. Moreover, the end-effector motions from pre-grasp to grasp and grasp to post-grasp should be feasible. Once a valid grasp is found, a path is computed using Lazy-CPC-PRM to reach to the IK solution of pre-grasp pose and then followed by a straight line motion of the end-effector from pre-grasp to grasp pose. For more precise grasping, one can use reactive behaviour while moving the gripper from pre-grasp to grasp pose or a force-regulating controller that helps to close the gripper which

takes feedback from tactile sensor in order to safely grasp an object [77, 76]. However, in our implementation, we skip this due to the lack of tactile sensor in our gripper. PLACE module also follows the same steps but in reverse order as mentioned in Figure 5.5. In the future, our PICK module will be replaced by a systematic approach. For example, for cases where the object is known and can be visually tracked (by putting some markers), visual servoing with end-effector cameras can ensure the robust execution of grasps by incrementally correcting the position of the object relative to the gripper. An excellent survey on this subject can be found in [78]. For cases where the object and gripper can not be visually tracked, methods such as [77], [79], [80] can be incorporated. While in the presence of object pose uncertainty and high clutter, push-grasping approach in [8] can be helpful.

## 5.5 HAMP-BAU and HAMP-BAU-TC for NBV-B

Note that HAMP-BAU and HAMP-BAU-TC are designed to plan a path from start to goal (base poses and manipulator configurations). However, to plan for NBV-B, we just have the goal base pose but the goal manipulator configuration is not known (neither required). Still, with minor modification, both the planners can be used to plan a path for the mobile manipulator with NBV-B as a goal. For the modification, we skip line 24 in Algorithm 10 for HAMP-BAU and in Algorithm 13 for HAMP-BAU-TC, i.e., the reconfiguration step at goal base pose (NBV-B) is not needed.

## 5.6 Changes in System in Case of Task-constraints

In case of task constraints, the system will replace (post-grasping) HAMP-BAU with HAMP-BAU-TC and Lazy-CPC-PRM with Lazy-CPC-PRM$_{\text{TS}}$. Everything else remains same.

## 5.7 Sensor Scans and World Representations

Figure 5.6 describes how the scans from different sensors are inserted and two (2D and 3D) world representations are formed. Hokuyo and Kinect scans are inserted into global Octomap [34], a 3D world representation that maintains occupied, free and unknown regions. From the Octomap, we get a 3D collision map (a set of occupied and unknown voxels, all of the same size) and a down projected (up to a certain height) 2D occupancy grid map. This occupancy map is further fused with another 2D map obtained from SLAM module [81] (which uses LMS100 scans for localization with map resolution of 0.05 m) to get a single 2D map. The fused 2D map, which shares information from all the sensors, is then used by base view planning to compute NBVs-B. We further input this map to costmap module (inflates costs based on 2D occupancy grid and user specified inflation radius) to get a 2D

costmap. The 2D costmap and 3D collision map are then used to perform collision checks as mentioned in Section 4.4.1.

## 5.8 Details of System Implementation

In this section, we mention about local Voxelmap and key parameters used in arm view planning and then describe how our system manages the data from both Kinect and Hokuyo sensors published at high rate given the limited capability of octomap in term of scans insertion.

In NBV-A algorithm, we use three parameters. Two of them are: the maximum number of iterations to try before declaring that arm view planning is over even if it is not (we use 15), the maximum number of sensor views to be searched to find the next best view (we use 50). While the third parameter (FrontiersTH as mentioned in Figure 5.3) tells whether arm view planning is needed or not (at fixed base pose). If the number of frontiers in the local Voxelmap are below FrontiersTH (we use 10), then EXPLORE_A module declares that local region around mobile manipulator is already explored and arm view planning is not required at current base pose. The end-effector in our SFU mobile manipulator as shown in Figure 5.1 can maximally extend up to 0.76 m. Therefore, in EXPLORE_A, we construct a Voxelmap that contains 20700 ($30 \times 30 \times 23$) voxel cells, each cell is a square of size 0.05 m. There are 30 cells each along $x$, $y$ and 23 cells along z axis (20 along +z and 3 along -z). Figure 5.7 shows the visualization of a Voxelmap (light magenta) where yellow colour denotes the frontiers as shown separately in Figure 5.8 (top view). Initially, we assume that the region (cylinder of radius 0.5 m) surrounding mobile manipulator is known free that is why the center region of Voxelmap is empty.

Now we mention about issues with real Hokuyo sensor. In simulation, scans from Hokuyo sensor are published at the rate of 10 Hz and each scan consists of 683 points. While Hokuyo on real robot also publishes at the same rate (with sensing range of 4.0 m) but each scan consists of roughly 383 points. However, given the sensor view ranging from -1.57 to 1.56 at angle increment of 0.0061 radians (slightly greater than simulation which is 0.0046) there should be 514 points. That implies around 130 points are not reported by sensor as the reading of corresponding rays is zero. We found out that lot of surfaces in our real environment (our lab) are black and for that Hokuyo does not work well [82]. As a result of this, the voxels covered by the rays that we are missing in case of real Hokuyo will not be cleared, therefore, the exploration in real environment takes longer as compared to simulation.

For our experiments, we used Fuerte version of ROS in Ubuntu 12.04. Therefore, the efficiency issues of octomap[1] (scans insertion with octree resolution of 0.05 m) that we mention here are related to that version. For Kinect, we throttled the point cloud data to

[1]We believe that octomap in recent ROS version might have been improved to some extent at least.

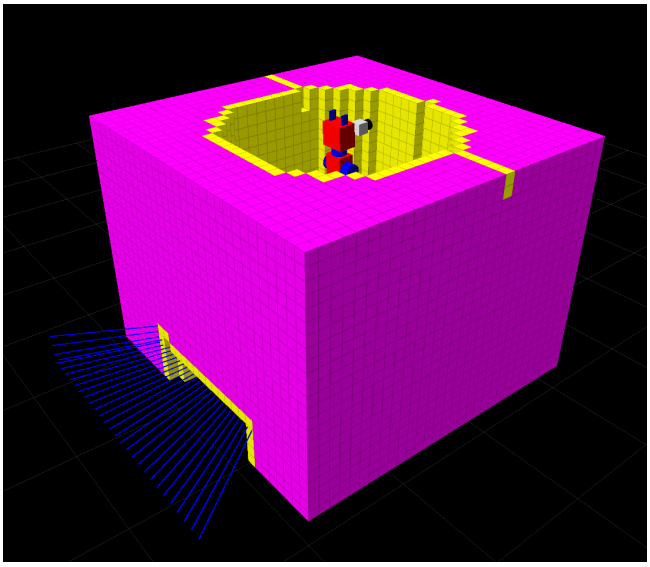


Figure 5.7: Voxelmap with known region in the center. Voxel cells (unknown) are shown in light magenta and frontiers in yellow colour.

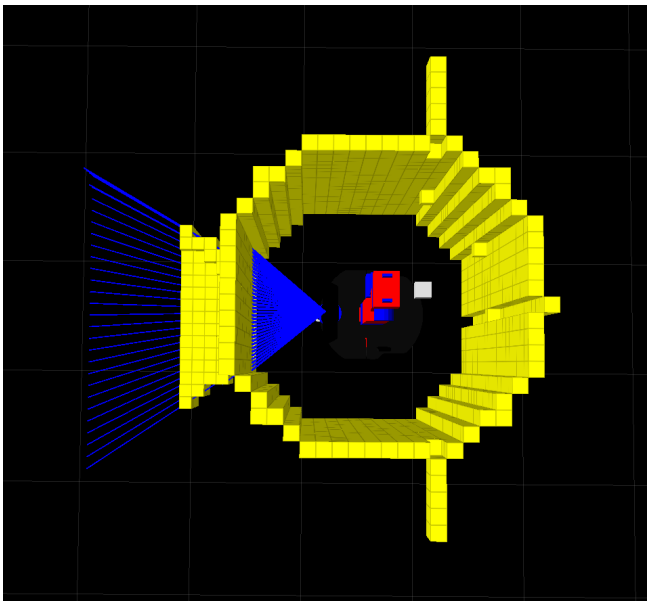


Figure 5.8: Top view of Voxelmap, only frontiers are shown.

get scans at the reduced rate of 2 Hz (original was 30 Hz) and then downsampled such that each scan consists of around 11189 points (17500 without downsample). We used QQVGA setting both for image and depth mode that gives the resolution of 160 $\times$ 120. Recall that the job of global octomap is to insert scans coming from Hokuyo and Kinect and also to publish 3D collision map as well as downprojected 2D map upto a certain height (we use 1 meter) as shown in Figure 5.6. Insertion of a Kinect scan takes 0.1 to 0.2 second and of a Hokuyo scan takes 0.01 to 0.02 second. While the publishing of maps (both occupancy

and collision) takes approximately 1 second. Therefore, it is obvious that some Hokuyo and Kinect scans will be missed given the rate at which data is published and the rate at which global octomap incorporates them. Our observation from experiments tells that Kinect contribution is only 25% in the exploration. Hokuyo is the main sensor that explores most of the environment. Therefore, one way to deal with missing scan issue is to reduce the speed at which Hokuyo takes the scans (we move arm with maximum speed of 0.05 radians/seconds).

## 5.9 Results

First, we separately evaluate EXPLORE_A module which is a key component of the system. If this module does not do its job properly then the system may not be able to complete the pick-and-place task. Thereafter, we provide full-fledged simulation and real experiment results for mobile pick-and-place task in unknown environment.

We evaluated EXPLORE_A for two different experiments, one without any task constraint while the other with task constraint, i.e., to keep the object in hand in upright position. The task was to explore the unknown region within the boundaries of a local Voxelmap. We carried out 40 trials and for each trial (with and without task constraint) we used different scenarios ranging from simple surrounding environment (no obstacles in the Voxelmap region) to cluttered ones like table with objects on it along with walls on other two sides. For experiments without any task constraint, we set 10 seconds as maximum permitted time for Lazy-CPC-PRM to plan a path while for experiments with task constraint, we set 60 seconds for Lazy-CPC-PRMTS. For both the experiments, we monitored (at each iteration) the number of frontiers, voxelmap update time, time to compute NBVs-A and planner runtime. To give a clear picture of how much time is taken by each component of EXPLORE_A as the number of iterations approach to the maximum limit, we provide one trial result for the cluttered scenario in Table 5.1 (without constraint) and for the simplest scenario (no obstacles in surrounding) in Table 5.2 (with constraint). From Table 5.1, we can observe that the number of frontiers decreases below threshold with the increase of iterations. It is also important to note that the time to compute NBV-A increases drastically as the number of frontiers dropped to small numbers. This is because to find NBV-A for a small unknown region requires large number of samples which in turn requires many simulations of sensor model (ray-tracing) which is a time consuming step. In our 40 trials, EXPLORE_A without task space constraint succeeded in exploring the entire Voxelmap region all the time with in 15 iterations. On an average it took 11 to 15 iterations to explore the local region.

On the other hand, EXPLORE_A with task space constraint failed to explore the Voxelmap in any of the 40 trials (15 iterations per trial) as shown in Table 5.2. Figure 5.9 shows the region remained unexplored even after 15 iterations. Seeing this, we increased

Table 5.1: EXPLORE_A module test in cluttered environment without task constraints.

| | # frontiers | Voxelmap update T. (sec) | NBV-A T. (sec) | Planning T. (sec) |
|---|---|---|---|---|
| 1 | 1498 | 0.0421 | 3.15 | 6.76 |
| 2 | 1649 | 0.0016 | 4.93 | 3.63 |
| 3 | 1562 | 0.6911 | 3.97 | 3.50 |
| 4 | 1395 | 0.2380 | 4.90 | 4.21 |
| 5 | 1380 | 0.5071 | 4.34 | 3.27 |
| 6 | 1226 | 0.0448 | 3.97 | 7.34 |
| 7 | 1191 | 0.0714 | 4.79 | 2.66 |
| 8 | 894 | 0.7044 | 4.58 | 2.20 |
| 9 | 728 | 0.0098 | 4.28 | 6.49 |
| 10 | 491 | 0.0756 | 8.74 | 2.39 |
| 11 | 151 | 0.4243 | 14.5 | 1.28 |
| 12 | 52 | 0.0427 | 29.6 | 1.35 |
| 13 | 24 | 0.0938 | 66.1 | 3.18 |
| 14 | 4 | 0.1619 | - | - |

Table 5.2: EXPLORE_A module test in simplest environment with task constraints.

| | # frontiers | Voxelmap update T. (sec) | NBV-A T. (sec) | Planning T. (sec) |
|---|---|---|---|---|
| 1 | 1498 | 0.3418 | 2.91 | 60 (failed) |
| | - | - | - | 16.18 |
| 2 | 1754 | 0.6897 | 3.32 | 60 (failed) |
| | - | - | - | 37.78 |
| 3 | 1696 | 0.0376 | 3.18 | 28.35 |
| 4 | 1581 | 0.0078 | 3.04 | 12.17 |
| 5 | 1568 | 0.0321 | 3.99 | 60 (failed) |
| . | . | . | . | . |
| . | . | . | . | . |
| . | . | . | . | . |
| 15 | 1490 | 0.3784 | 4.81 | 60 (failed) |

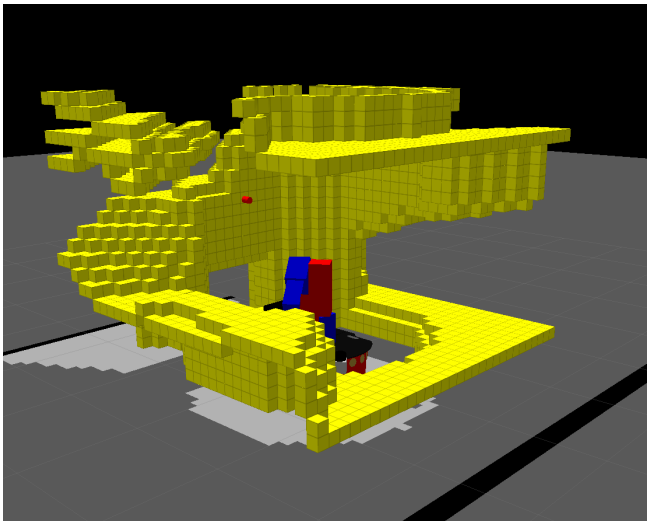


Figure 5.9: EXPLORE_A with task space constraint failed to explore complete region of local Voxelmap. Only frontiers are shown in yellow colour, voxel cells are not shown in this screenshot. Red colour dot shows NBV-A but Lazy-CPC-PRMTS failed to find a path for it.

the permitted time to plan a path to 180 seconds and maximum number of iterations to 25. Still the behaviour of EXPLORE_A with task space constraint remains same. We attribute the failure to two things: crucial one is the task constraints put on end-effector, i.e., maintain vertical orientation, as a result the eye-in-hand sensor can't really get an area scan, hence is unable to sense much of the environment as compared to when there are no constraints while the less significant factor is the Lazy-CPC-PRMTS as 60% of the times it fails to find a path to reach a NBV-A (Figure 5.9 shows one such example). One can not do much in this situation, the best that can be done is to put down the object and explore before picking the object or use other alternatives.

Because of ineffectiveness of EXPLORE_A with task constraint, we now demonstrate our planners in the following way. For HAMP-BAU, we demonstrate it as planned, i.e., for mobile pick-and-place tasks in unknown environment. We then save the global octomap (explored environment) obtained from the demonstration and use it to demonstrate HAMP-BAU-TC for mobile pick-and-place tasks but in known environment.

### 5.9.1 Simulations

We now present simulation results for mobile pick-and-place task in unknown environment to demonstrate HAMP-BAU. The simulation environment is shown in Figure 5.10. We ran our integrated and autonomous system in this environment and the outcomes are provided in Table 5.3. On an average our system took 2 hours to completely explore the environment and completes the pick-and-place task. It is important to note that only 17% of total time is the computational time while the remaining is motion execution time (100 minutes)

that includes physically moving the mobile manipulator or its parts, for example, executing manipulator plans to reach NBVs-A and then scanning using eye-in-hand sensor, executing mobile manipulator plans to reach NBVs-B or pick and place base poses, etc. In our simulation trials, we actually moved the arm with maximum speed of 0.4 radians/seconds (not 0.05 as we mentioned above and only used for real experiments), however, the arm appeared to move much slower commensurate with maximum speed of 0.03 radians/seconds. We believe that CPU is consumed by multiple tasks like simulation platform (Gazebo, ROS), visualization tool (Rviz, ROS) and hence slows down the execution. On an average, the system invoked EXPLORE_A 3 times and EXPLORE_B 4 times for a total of 18 NBV-A and 2 NBV-B were reached. Our EXPLORE_B calls also include call to reach pick base pose or place base pose.

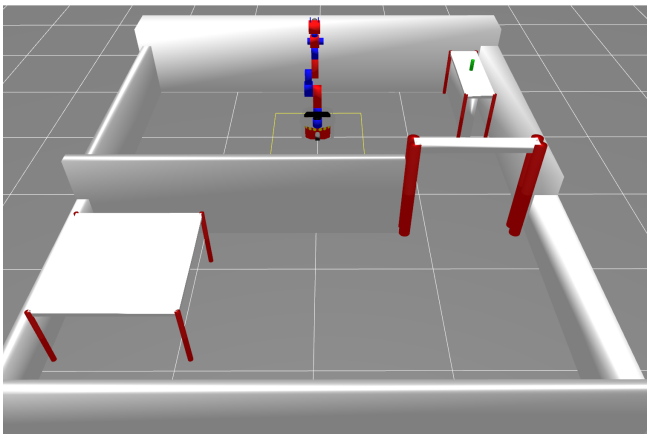


Figure 5.10: Simulation environment for pick-and-place task. The task is to explore the environment, pick the object (green colour) and place it on table located on the other side of the door.

Table 5.3: Comparison of simulations and real experiments for pick-and-place task in unknown environment.

| Detail | Simulations | Real Experiments |
|---|---|---|
| Total trials | 8 | 2 |
| Total time taken (avg.) | 120 minutes | 150 minutes |
| Total execution time (avg.) | 100 minutes | 116 minutes |
| Total computational time (avg.) | 20 minutes | 34 minutes |
| Total Hokuyo scan time (avg.) | 49 minutes | 63 minutes |
| Num. of EXPLORE_A calls | 3 | 4 |
| Num. of NBV-A reached | 18 | 23 |
| Num. of EXPLORE_B calls | 4 | 8 |
| Num. of NBV-B reached | 2 | 3 |

One of the simulation trials is depicted from Figures 5.11 to 5.13 and also available in the attached video (Multimedia Resource 6). In the figures, cyan colour represents the

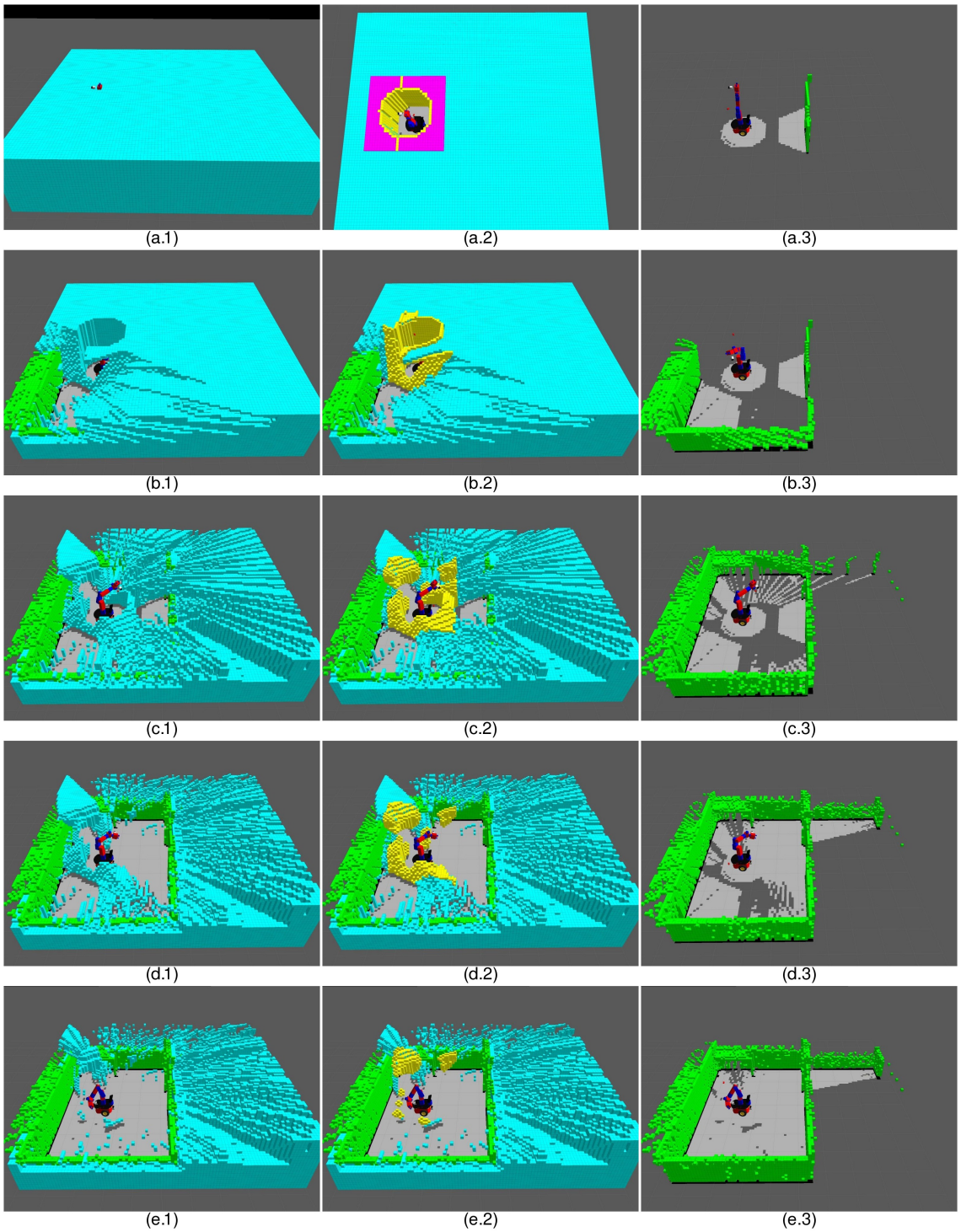



Figure 5.11: Continued...

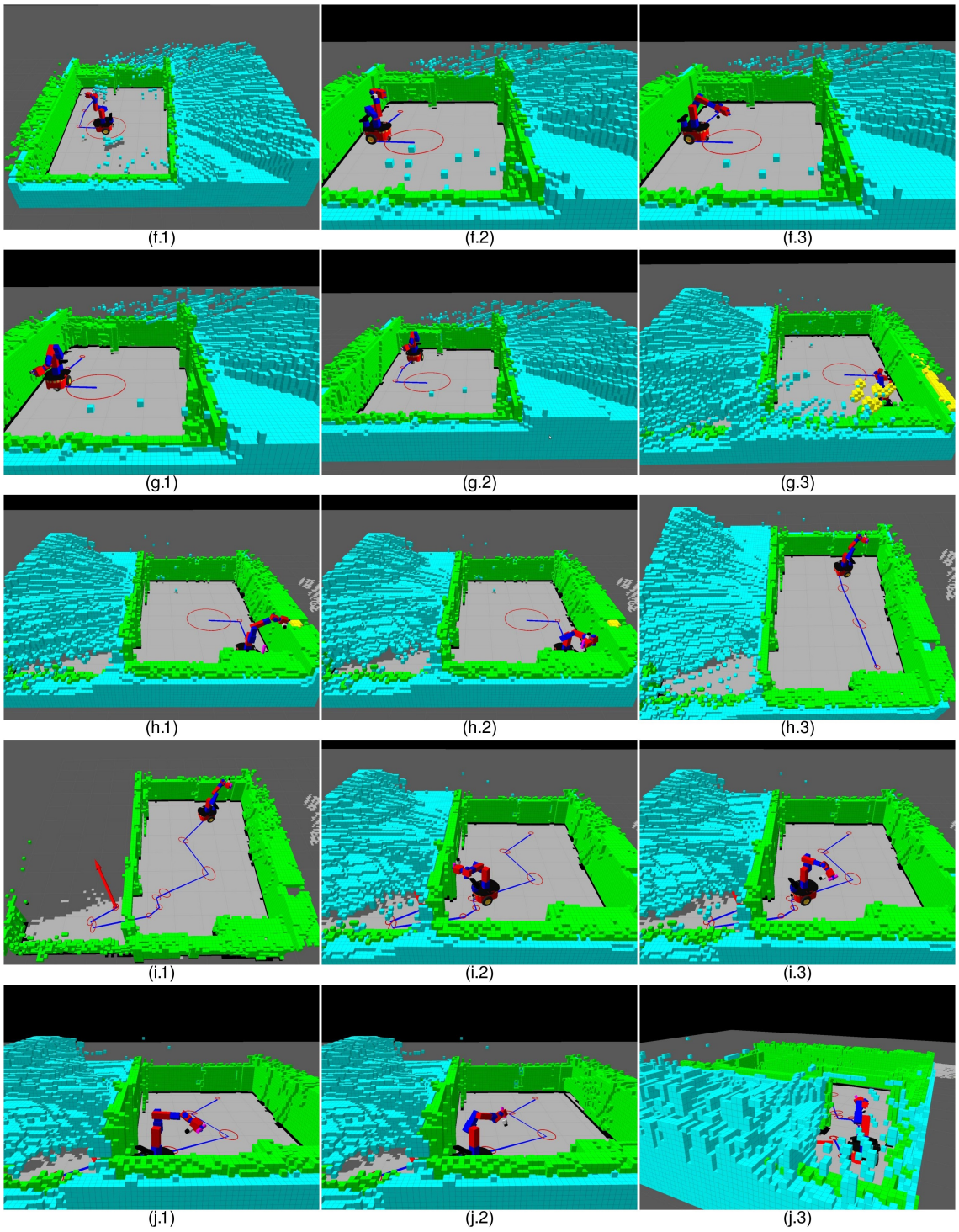



Figure 5.12: Continued...

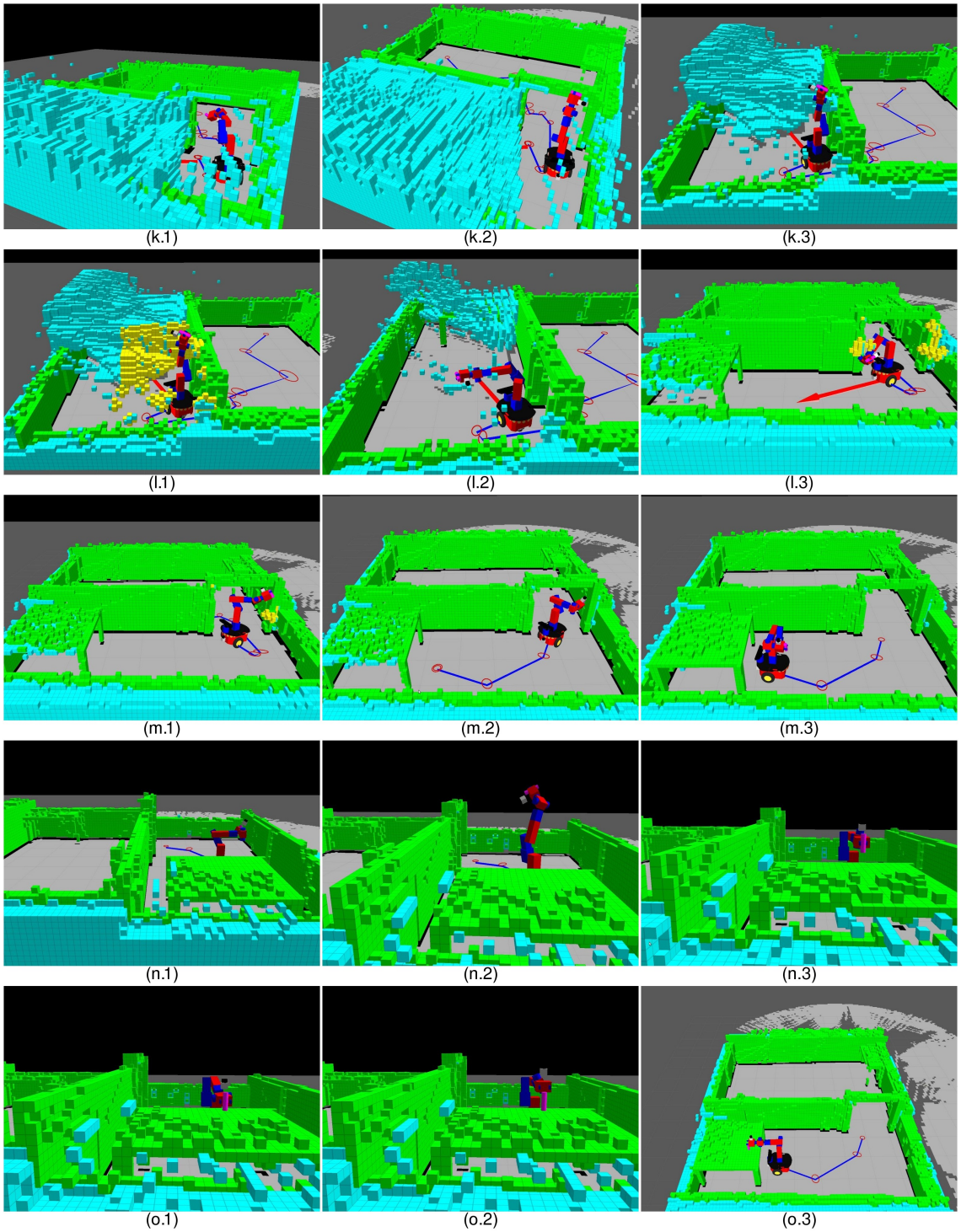



Figure 5.13: Simulation test for pick-and-place task in unknown environment to demonstrate HAMP-BAU. (a.1) shows the initial unknown environment and (o.3) shows the environment after exploration along with object placement. Please see text for description.

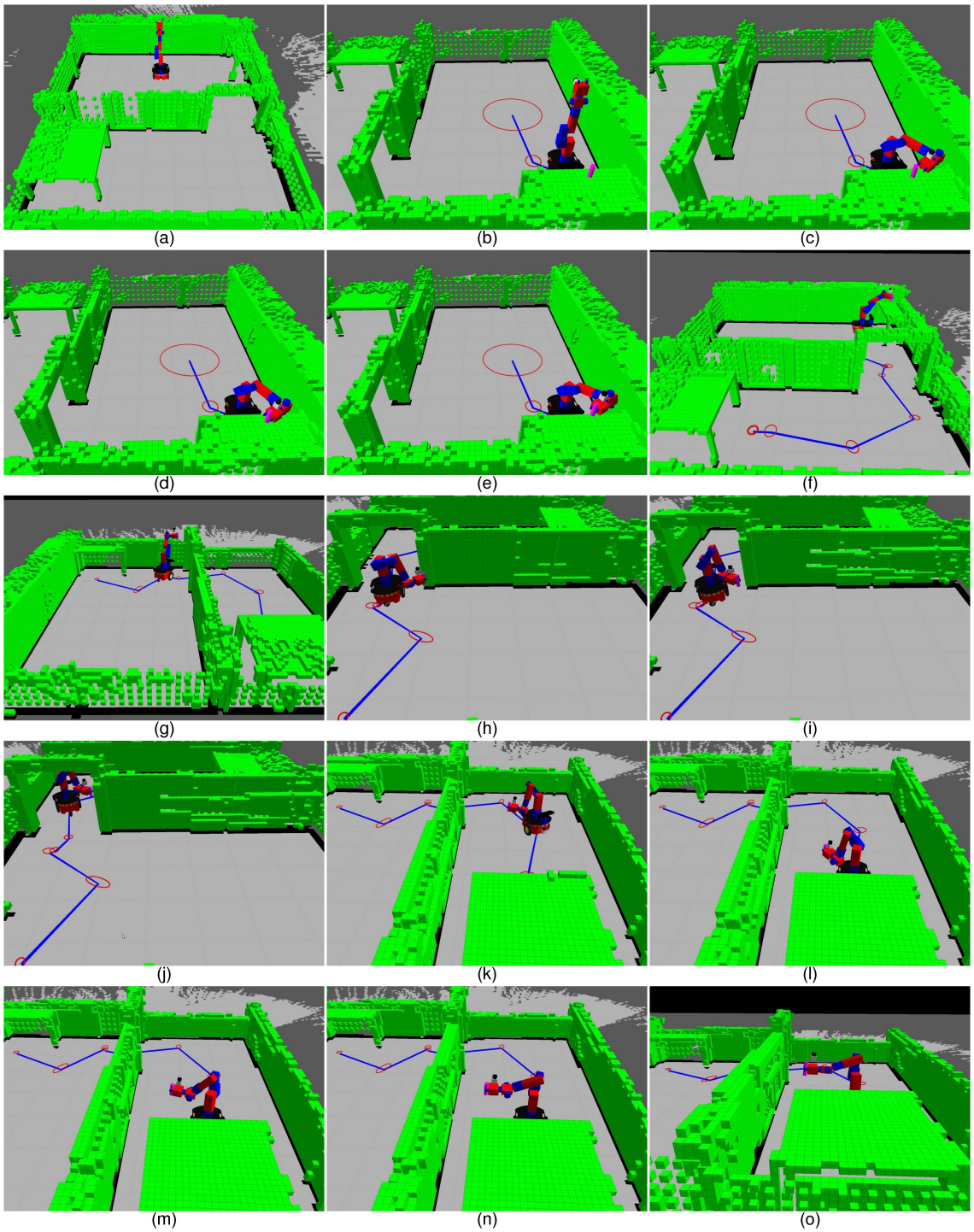



Figure 5.14: Continued...

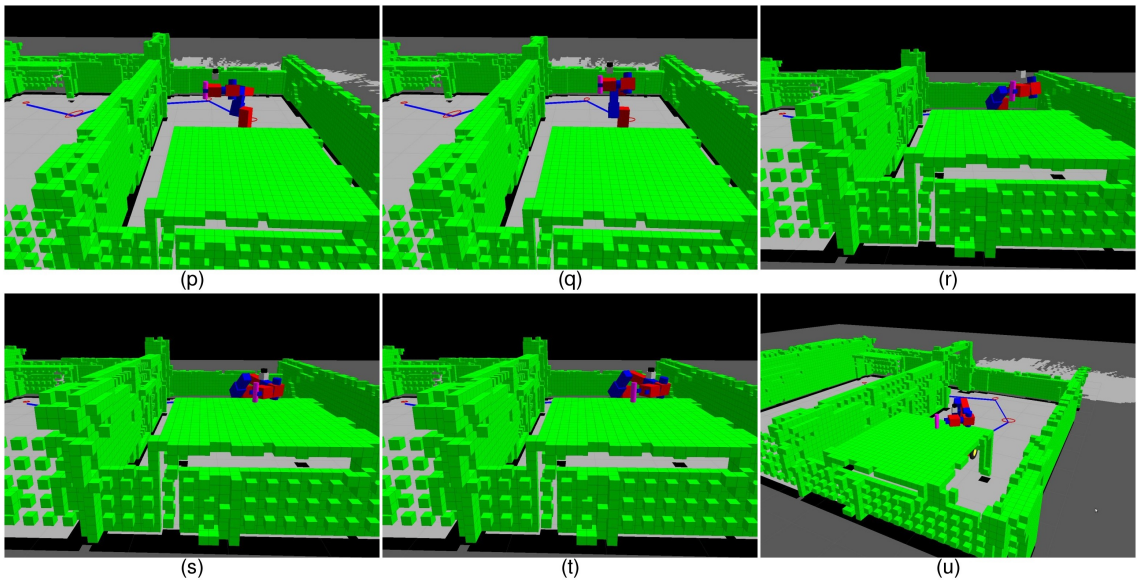



Figure 5.15: Simulation test for pick-and-place task in known environment to demonstrate HAMP-BAU-TC. Please see text for description.

unknown regions, green colour represents the obstacles, magenta as Voxelmap and frontiers are represented by yellow colour. Screenshots in (a) show the unknown region (6m × 4m × 2m) in the beginning and the initial known region assumed around the mobile manipulator. The Voxelmap and frontiers at start are shown in (a.2). In the beginning, EXPLORE_A module was invoked to explore the local region and the screenshots from (b) to (e) show the 2D and 3D environment explored at different iterations of arm view planning. In total, it took 10 iterations, i.e., 10 NBVs-A were reached, and the region explored at the end is shown in (e.1) and (e.3). Within the known region, the pick base pose was reachable, therefore, a path was planned to reach as shown in (f.1). Screenshots from (f) to (g) show the execution of mobile manipulator path. Along the path, the arm reconfigured once as shown in (f.2) and (f.3). After reaching to pick base pose, the grasping was not possible due to unexplored region around the vicinity of object, therefore, arm view planning was invoked to clear the unknown region. (g.3) and (h.1) show the frontiers before and after arm view planning. The object is grasped in (h.2). Post-grasping, the mobile manipulator moved toward already explored region (h.3) as there were few unknown voxels left. From there, a new NBV-B was searched and a path was planned (i.1). Screenshots from (i) to (k) show the path execution. Along the path, arm reconfigured 4 times, also shown in the screenshots. After reaching to NBV-B, arm view planning was invoked to explore local region. Frontiers at different iterations are shown from (i.1) to (m.1) (not all steps are shown). After local exploration (took 7 iterations), a path to reach place base pose was found and the object was placed as shown from (m.2) to (o.2). Finally explored environment is shown in (o.3).

Figure 5.16: Real environment for pick-and-place task. The mobile manipulator start configuration and object (bottle) are shown in the top figure while the bottom figure shows the table on the other side of the door where object should be placed.

The explored environment (global octomap) was saved and then used to demonstrate HAMP-BAU-TC for pick-and-place task in known environment. The demonstration is shown in Figures 5.14 and 5.15 and also available in the attached video (Multimedia Resource 7). HAMP-BAU was used to plan a path to reach pick base pose, shown in (b), as there was no end-effector constraints up to that point. Post-grasping (e), the path was computed using HAMP-BAU-TC. Screenshots from (f) to (u) show the path execution while maintaining task space constraints, i.e., to keep the object upright. The arm reconfigured twice along the path, as shown in (h) and (i). The placement of object is shown from (r) to (u).

### 5.9.2 Real Experiments on SFU Mobile Manipulator

For real experiments on SFU mobile manipulator, we used the environment shown in Figure 5.16 to demonstrate our integrated and autonomous system for pick-and-place task in unknown environment. We carried out two trials and the outcomes are provided in Table 5.3. As compared to simulations where the Hokuyo sensor was not sensitive to black surfaces,

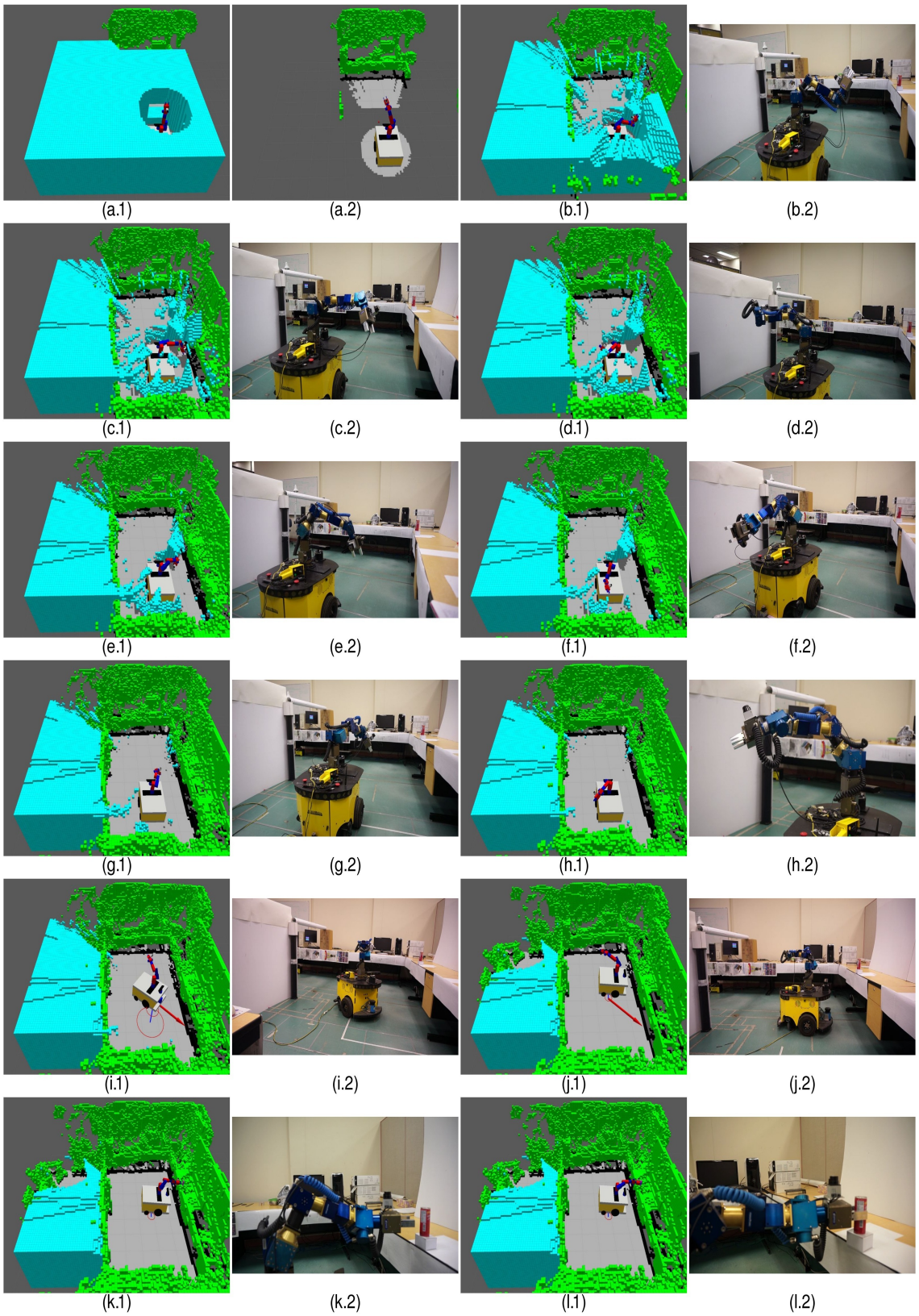



Figure 5.17: Continued...

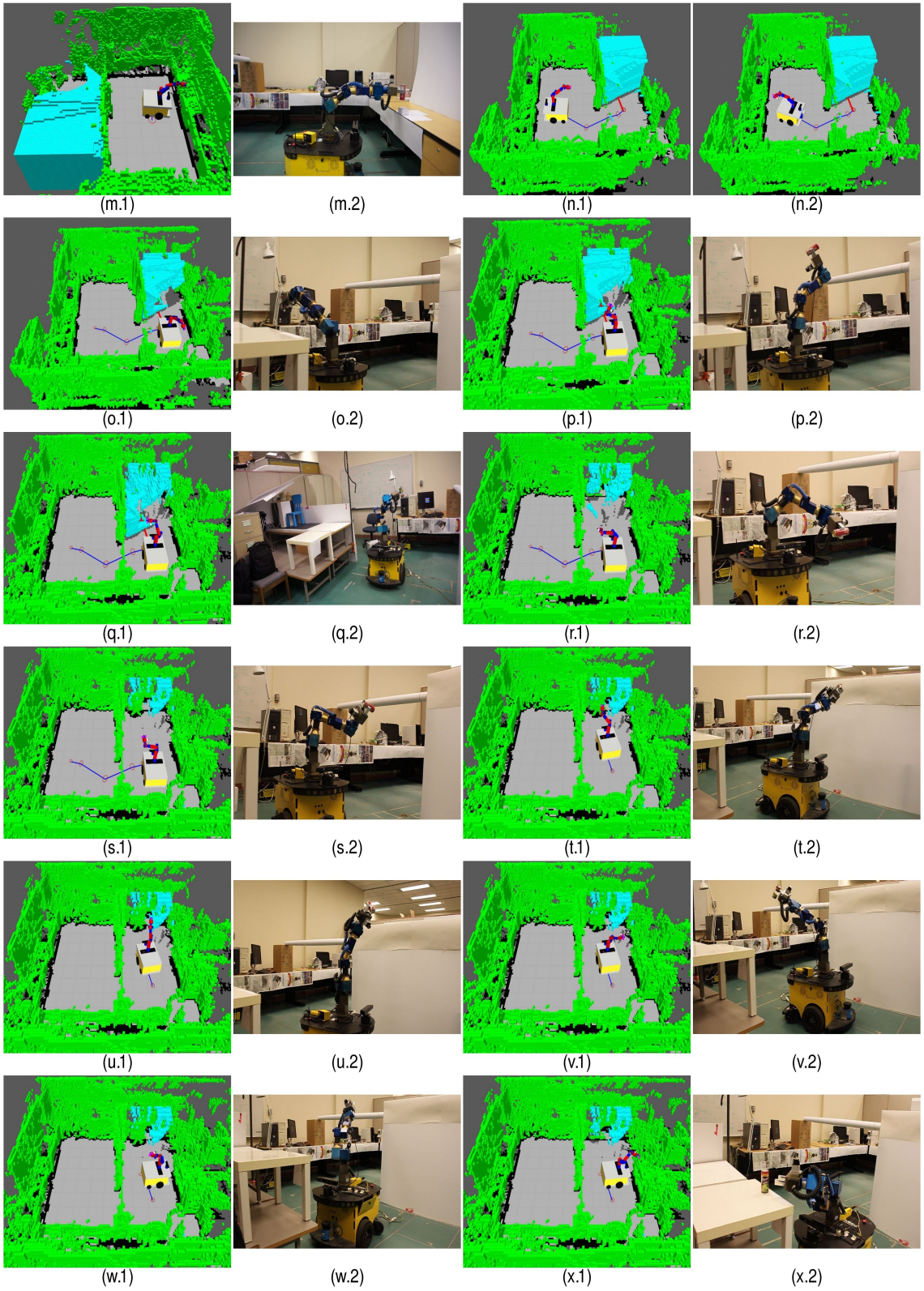



Figure 5.18: Real experiment for pick-and-place task in unknown environment to demonstrate HAMP-BAU. (a.1) shows the initial unknown environment and (x.1) shows the environment after exploration. Please see text for description.

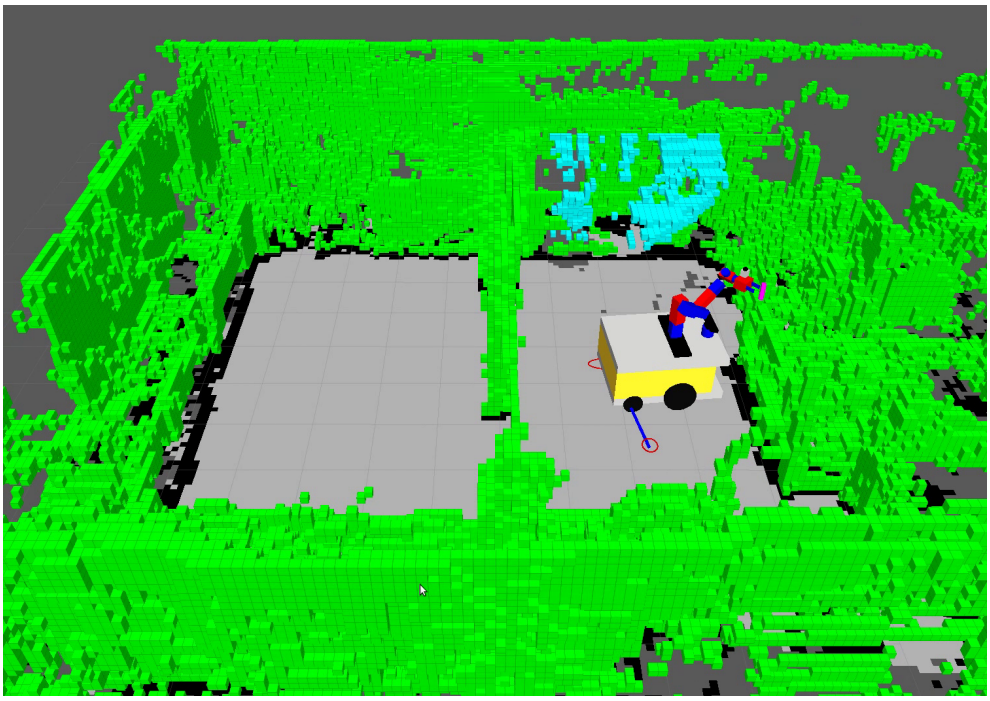


Figure 5.19: [Real experiment trial 1]: less than 1% of the environment remained unexplored (in cyan colour) as the system was able to complete the pick-and-place task within the known region.

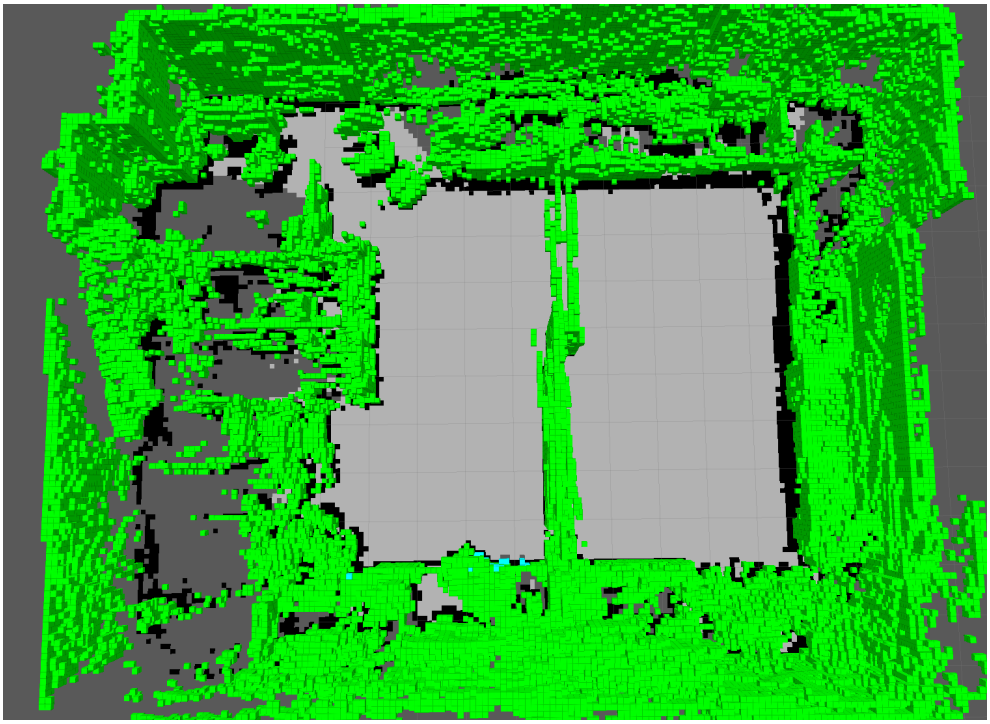


Figure 5.20: [Real experiment trial 2]: fully explored environment.

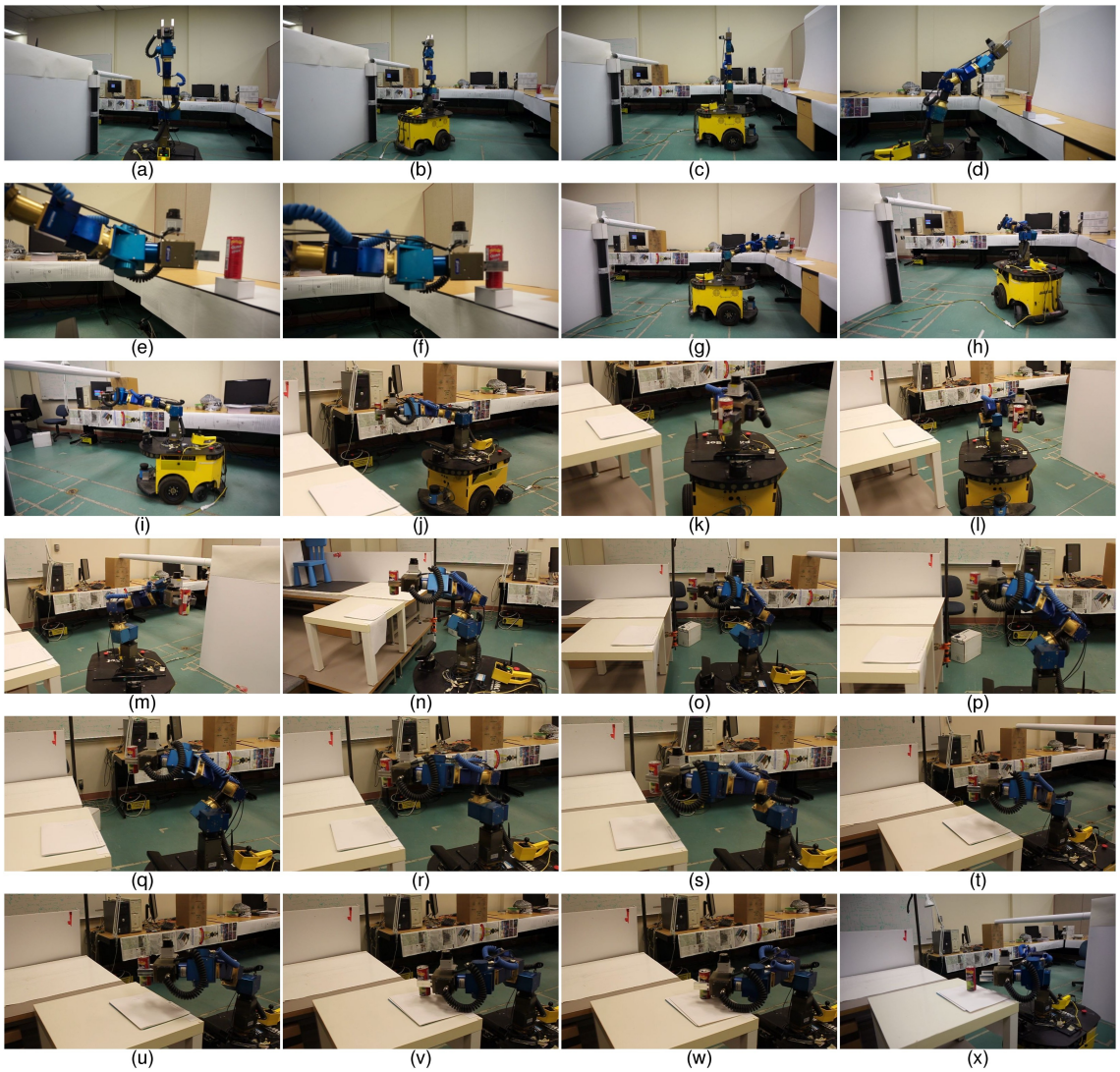



Figure 5.21: Real experiment for pick-and-place task in known environment to demonstrate HAMP-BAU-TC. Please see text for description.

the system for real experiments took longer (150 minutes) to explore the environment, pick the object and place it at target location. Note that we hid 30% of the black surfaces by covering with papers as can be seen in the screenshots of the environment. Due to issues with eye-in-hand sensor (Hokuyo), the system took more number of iterations in a EXPLORE_A call, i.e., in total 23 NBVs-A were reached. Also, the system took 8 EXPLORE_B calls that include few of the failed attempts (3), for example, pick or place base pose was collision-free but the system failed to find a path with in the permitted time. This also shows that our system is robust to failure of individual modules as it tries to revisit the same problem next time in the loop.

One of the trials for real experiment is demonstrated from Figures 5.17 to 5.18. Screenshots in (a) show the unknown and known region in the beginning. The arm view planning

was invoked at start base pose to explore the local region surrounding the mobile manipulator and screenshots from (b) to (h) show the eye-in-hand sensor at different NBVs-A and the environment left unexplored after each scanning from a NBV-A. This EXPLORE_A call took 15 iterations, i.e., 15 NBV-A were reached and the environment cleared at the end is shown in (h.1). Compared to simulation, the EXPLORE_A module in real experiment roughly takes 30% more time to explore the same amount of space. With in the explored region, the pick base pose was not reachable, therefore, a path was planned using HAMP-BAU to reach NBV-B shown in (i). Thereafter, the pick base pose was reached and the object was grasped as shown in (j) and (k)-(m), respectively. Screenshots from (n) to (o) show the mobile manipulator path execution to reach NBV-B to explore the unknown region on the other side of the door. From the reached NBV-B, the EXPLORE_A module was invoked that took 8 iterations to explore the local region as shown from (p) to (s). Note that, post arm exploration, there was some unexplored region left (s.1) but that was outside the local Voxelmap (not shown here) and on the other hand the place base pose was reachable with in the explored region. Therefore, a path was planned and figures from (t) to (v) show the arm reconfiguration step along the path. In (w) and (x), the mobile manipulator reached to place base pose and the object was placed at target location. Figure 5.19 shows the explored environment after completing the task. This real experiment trial is also shown in the video attached to this paper (Multimedia Resource 8). Figure 5.20 shows the final outcome of our second trial where the environment was fully explored.

The saved environment (global octomap) from second trial was then used to demonstrate the system for pick-and-place task in known environment where HAMP-BAU-TC was used (post-grasping) to plan mobile manipulator paths that maintain task-space constraints. Figure 5.21 shows the screenshots of a real experiment where the SFU mobile manipulator reached to pick base pose (a)-(c), grasped the object (d)-(g), and then reached to place base pose (h)-(n) to put the object at target location (o)-(x). Note that the arm reconfigured along the mobile manipulator path planned from pick base pose (post-grasping) to place base pose and the reconfiguration step maintained task space constraints as shown in (l) and (m). This experiment is also available in the attached video (Multimedia Resource 9).

# Chapter 6

# Conclusions and Future Work

## 6.1 Conclusions

In this thesis, we proposed sampling-based efficient and robust mobile manipulator planners that use efficient and smart strategies to deal with computational complexity and incorporate uncertainty to generate safer plans. In the first part of research, we addressed the design of an efficient mobile manipulator planner for deterministic case, where robot state is fully known. For that, we proposed a Hierarchical and Adaptive Mobile Manipulator Planner (HAMP) that plans both for the base and the arm in a judicious manner - allowing the manipulator to change its configuration autonomously when needed if the current arm configuration is in collision with the environment as the mobile manipulator moves along the planned path. We showed that HAMP is probabilistically complete. We extensively evaluated HAMP in different scenarios with varying levels of complexity. We also evaluated the tree versions of HAMP with RRT and Bi-directional RRT (BiRRT) as the core sub-planners for searching for both the base and the manipulator, respectively called HAMP-RRT and HAMP-BiRRT.

In the second part, we proposed localization aware sampling and connection strategies (LAS and LAC, respectively) to consider only those nodes and edges which contribute toward better localization. Our novel sampling strategy judiciously places the samples using a new notion of "localization ability of a sample", i.e., it puts more samples in regions where sensor data is able to achieve higher uncertainty reduction while maintaining adequate samples in regions where uncertainty reduction is poor. Our simulation results showed that these strategies help to reduce the planning time significantly with little compromise on the quality of path. We also discussed probabilistic completeness and optimality issues associated with our strategies.

Third, integrating the above two components (LAS and LAC with in HAMP), we designed an efficient and robust mobile manipulator planner (HAMP-BAU) that plans judiciously and incorporates the base pose uncertainty and the effects of this uncertainty on

manipulator plans. We also extended HAMP-BAU to incorporate task space constraints (HAMP-BAU-TC). We evaluated both planners in known environment and showed that our planners find a safer path as compared to other variants where uncertainty is not considered at different levels.

Finally, in the last part of the work, we incorporated our planners (HAMP-BAU and HAMP-BAU-TC) within an integrated and fully autonomous system for mobile pick-and-place tasks in unknown static environments. A key aspect of our integrated system is that the planner works in tandem with base and arm exploration modules that explore the unknown environment. We demonstrated our system both in simulation and real experiments on SFU mobile manipulator.

## 6.2 Future Work

Below, we suggest few directions to further extend our work:-

- HAMP can be enhanced in term of reduction in planning time. For example, one key enhancement could be a reachable manipulator configuration, serve as a goal for reconfiguration path, should be searched in task space instead of C-space. This was noticed during evaluation of HAMP-BAU-TC. Our experience tells that it is one of the time consuming steps. Furthermore, there is a scope to improve the path search phase.

- HAMP basically decomposes the full space into two sub-spaces, i.e., base sub-space and manipulator sub-space. In this particular example, the decomposition is quite natural and is motivated by the fact that in a majority of day to day indoor environments, it is often the case that when the base moves the arm does not need to move to avoid collisions except at few configurations where arm reconfiguration can take place while the base is stationary. In principle, the HAMP framework can be applied to any robotic system by decomposing it into two (or more) sub-spaces and then searching them in HAMP type manner. This approach may not always lead to efficient planners and an interesting question to explore would be under what conditions such a decomposition would lead to more efficient planners rather than searching the full space ?

- We believe that $L_n$ can be extended to the multimodal distribution using Monte Carlo localization (MCL) [83] that uses a particle filter to represent the distribution of likely states, with each particle representing a possible robot state. The MCL algorithm works in two stages. First, it uses the motion model to shift the particles to predict its new state after the motion and the likelihood (weight) of each new particle is computed using sensor measurements. In the second stage, the particles

are resampled based on how well the actual sensed data correlate with the predicted state.

To extend our $L_n$ measure to multimodal distribution, we bypass the prediction of new particles based on the robot motion as we do not know the control commands at the sampling stage. The procedure to compute the localization ability of a sample is then as follows. We could assume a fixed distribution of particles around a sampled point with each particle assigned the same weight. This set of particles essentially serves the same role as $M$ for the Gaussian case. The same distribution is used for all samples by appropriately transforming corresponding to the co-ordinates of the sample points. Sensor measurement step is then used to assign new weights to each particle followed by a resampling step as in standard MCL. This new set of particles essentially serves the same role as $\Sigma_n$ for the Gaussian case. Kullback-Leibler divergence [84] that measures the information gain between two probability distributions can then be used as localization ability of a sample.

- There is ample scope of improvement in HAMP-BAU, especially the computation of reconfiguration paths by considering the base pose uncertainty. Presently, we use Lazy-CPC-PRM and this manipulator planner fails to find a path even in simple environment if base pose uncertainty is high (and also depending on the threshold used there). One possible solution is that the planner should give the best scenario path even if it fails to find one that satisfies collision probability threshold.

- Our integrated and autonomous system (described in Chapter 5) does not consider a systematic approach for grasping task, it assumes a given grasp pose. Therefore, first an appropriate grasp planner should be integrated with in the system by replacing the PICK module. Thereafter, the respective grasp planner can be extended to incorporate uncertainty associated with grasping tasks. So, there is a possibility of extending our work in that direction.

# Bibliography


[1] A. Jain and C. Kemp, "EL-E: An assistive mobile manipulator that autonomously fetches objects from flat surfaces," *Autonomous Robots*, vol. 28, no. 1, pp. 45–64, 2010.

[2] S. Srinivasa, D. Ferguson, C. Helfrich, D. Berenson, A. Collet, R. Diankov, G. Gallagher, G. Hollinger, J. Kuffner, and M. VandeWeghe, "HERB: A home exploring robotic butler," *Autonomous Robots*, vol. 28, no. 1, pp. 5–20, 2010.

[3] M. Beetz, U. Klank, I. Kresse, A. Maldonado, L. Mosenlechner, D. Pangercic, T. Ruhr, and M. Tenorth, "Robotic roommates making pancakes," in *Proc.of the 11th IEEE-RAS International Conference on Humanoid Robots*, Bled, Slovenia, 2011, pp. 529–536.

[4] J. Stuckler, M. Schwarz, M. Schadler, A. Topalidou-Kyniazopoulou, and S. Behnke, "NimbRo Explorer: Semi-autonomous exploration and mobile manipulation in rough terrain," *to appear in Journal of Field Robotics*, 2015, (accessed on November 20, 2015). [Online]. Available: http://www.ais.uni-bonn.de/papers/JFR_2015_NimbRo_Explorer.pdf

[5] A. S. et al, "Chimp, the cmu highly intelligent mobile platform," *Journal of Field Robotics*, vol. 32, no. 2, pp. 209–228, 2015.

[6] M. Maimone, Y. Cheng, and L. Matthies, "Two years of visual odometry on the mars exploration rovers," *Journal of Field Robotics*, vol. 24, 2007.

[7] S. Srinivasa, D. Ferguson, J. M. Vandeweghe, R. Diankov, D. Berenson, C. Helfrich, and H. Strasdat, "The robotic busboy: Steps towards developing a mobile robotic home assistant," in *International Conference on Intelligent Autonomous Systems*, 2008, pp. 129–136.

[8] M. Dogar and S. S. Srinivasa., "Push-grasping with dexterous hands: Mechanics and a method," in *Proc.of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2010, pp. 2123–2130.

[9] D. Berenson, J. Kuffner, and H. Choset, "An optimization approach to planning for mobile manipulation," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, May 2008, pp. 1187–1192.

[10] E. Marder-Eppstein, E. Berger, T. Foote, B. P. Gerkey, and K. Konolige, "The office marathon: Robust navigation in an indoor office environment," in *Proc.of the IEEE International Conference on Robotics and Automation (ICRA)*, May 2010, pp. 300–307.

[11] J. Scholz, S. Chitta, B. Marthi, and M. Likhachev, "Cart pushing with a mobile manipulation system: Towards navigation with moveable objects," in *Proc.of the IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 2011, pp. 6115–6120.

[12] S. M. Lavalle, "Rapidly-exploring random trees: A new tool for path planning," Computer Science Dept., Iowa State University, Tech. Rep. 98-11, 1998.

[13] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

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

[15] L. Kaelbling, M. Littman, and A. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence*, vol. 101, pp. 99–134, 1998.

[16] J. Pineau, G. Gordon, and S. Thrunn, "Point-based value iteration: An anytime algorithm for POMDPs," in *International Joint Conferences on Artificial Intelligence*, August 2003, pp. 1025–1032.

[17] H. Kurniawati, Y. Du, D. Hsu, and W. S. Lee, "Motion planning under uncertainty for robotic tasks with long time horizons," *International Journal of Robotics Research*, vol. 30, no. 3, pp. 308–323, 2011.

[18] H. Kurniawati, T. Bandyopadhyay, and N. Patrikalakis, "Global motion planning under uncertain motion, sensing, and environment map," *Autonomous Robots*, vol. 33, no. 3, pp. 255–272, 2012.

[19] H. Bai, D. Hsu, and W. S. Lee, "Integrated perception and planning in the continuous space: A POMDP approach," *The International Journal of Robotics Research*, vol. 33, no. 9, pp. 1288–1302, June 2014.

[20] B. Bouilly, T. Simeon, and R. Alami, "A numerical technique for planning motion strategies of a mobile robot in presence of uncertainty," in *Proc.of the IEEE International Conference on Robotics and Automation (ICRA)*, May 1995, pp. 1327–1332.

[21] A. Lazanas and J.-C. Latombe, "Motion planning with uncertainty: a landmark approach," *Artificial Intelligence*, vol. 76, no. 1-2, pp. 287–317, 1995.

[22] T. Fraichard and R. Mermond, "Path planning with uncertainty for car-like robots," in *Proc.of the IEEE International Conference on Robotics and Automation (ICRA)*, May 1998, pp. 27–32.

[23] A. Lambert and D. Gruyer, "Safe path planning in an uncertain-configuration space," in *Proc.of the IEEE International Conference on Robotics and Automation (ICRA)*, Roma, Italy, September 2003, pp. 4185–4190.

[24] N. A. Melchior and R. Simmons, "Particle RRT for path planning with uncertainty," in *Proc.of the IEEE International Conference on Robotics and Automation (ICRA)*, Roma, Italy, April 2007, pp. 1617–1624.

[25] Y. Huang and K. Gupta, "RRT-SLAM for motion planning with motion and map uncertainty for robot exploration," in *Proc.of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, September 2008, pp. 22–26.

[26] S. Prentice and N. Roy, "The belief roadmap: Efficient planning in belief space by factoring the covariance," *The International Journal of Robotics Research*, vol. 28, no. 11-12, pp. 1448–1465, July 2009.

[27] J. van den Berg, P. Abbeel, and K. Goldberg, "LQG-MP: Optimized path planning for robots with motion uncertainty and imperfect state information," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 895–913, 2011.

[28] A. Bry and N. Roy, "Rapidly-exploring random belief trees for motion planning under uncertainty," in *Proc.of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, May 2011, pp. 723–730.

[29] A. Agha-mohammadi, S. Chakravorty, and N. Amato, "FIRM: Sampling-based feedback motion planning under motion uncertainty and imperfect measurements," *The International Journal of Robotics Research*, vol. 33, no. 2, pp. 268–304, February 2014.

[30] R. Glashan, K. Hsiao, L. P. Kaelbling, and T. Lozano-Pérez, "Grasping POMDPs: Theory and experiments," *Robotics Science and Systems Manipulation Workshop: Sensing and Adapting to the Real World*, 2007, (accessed on November 20, 2015). [Online]. Available: http://www.willowgarage.com/sites/default/files/paper__grasping_pomdps_theory_and_experiments___glashan.pdf

[31] A. Hornung, M. Phillips, E. G. Jones, M. Bennewitz, M. Likhachev, and S. Chitta, "Navigation in three-dimensional cluttered environments for mobile manipulation," in *Proc.of the IEEE International Conference on Robotics and Automation (ICRA)*, May 2012, pp. 423–429.

[32] S. Karaman, M. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the RRT*," in *Proc.of the IEEE International Conference on Robotics and Automation (ICRA)*, 2011, pp. 1478–1483.

[33] Y. Huang and K. Gupta, "Collision-probability constrained PRM for a manipulator with base pose uncertainty," in *Proc.of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2009, pp. 1426–1432.

[34] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Autonomous Robots*, vol. 34, no. 3, pp. 189–206, April 2013.

[35] H. G. Tanner and K. J. Kyriakopoulos, "Nonholonomic motion planning for mobile manipulators," in *Proc.of the IEEE International Conference on Robotics and Automation (ICRA)*, April 2000, pp. 1233–1238.

[36] J. Tan and N. Xi, "Unified model approach for planning and control of mobile manipulators," in *Proc.of the IEEE International Conference on Robotics and Automation (ICRA)*, May 2001, pp. 3145–3152.

[37] Y. Yamamoto and X. Yun, "Coordinating locomotion and manipulation of a mobile manipulator," *IEEE Trans. Autom. Control*, vol. 39, no. 6, pp. 1326–1332, 1994.

[38] Y. Yang and O. Brock, "Elastic roadmaps - motion generation for autonomous mobile manipulation," *Autonomous Robot*, vol. 28, no. 1, pp. 113–130, 2010.

[39] A. Hornung and M. Bennewitz, "Adaptive level-of-detail planning for efficient humanoid navigation," in *Proc.of the IEEE International Conference on Robotics and Automation (ICRA)*, May 2012, pp. 997–1002.

[40] J. Yang, P. Dymond, and M. Jenkin, "Exploiting hierarchical probabilistic motion planning for robot reachable workspace estimation," in *Informatics in Control Automation and Robotics, LNEE85*. Springer-Verlag, 2011, pp. 229–241.

[41] K. Gochev, A. Safonova, and M. Likhachev, "Planning with adaptive dimensionality for mobile manipulation," in *Proc.of the IEEE International Conference on Robotics and Automation (ICRA)*, May 2012, pp. 2944–2951.

[42] J. Vannoy and J. Xiao, "Real-time adaptive motion planning (RAMP) of mobile manipulators in dynamic environments with unforeseen changes," *IEEE Transctions on Robotics*, vol. 24, no. 5, pp. 1199–1212, 2008.

[43] J. Gonzalez and A. Stentz, "Planning with uncertainty in position: An optimal and efficient planner," in *Proc.of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, Edmenton, Canada, May 2005, pp. 1327–1332.

[44] N. Roy and S. Thrun, "Coastal navigation with mobile robots," in *Advances in Neural Processing Systems 12*, 1999, pp. 1043–1049.

[45] P. Missiuro and N. Roy, "Adapting probabilistic roadmaps to handle uncertain maps," in *Proc.of the IEEE International Conference on Robotics and Automation (ICRA)*, May 2006, pp. 1261–1267.

[46] B. Burns and O. Brock, "Sampling-based motion planning with sensing uncertainty," in *Proc.of the IEEE International Conference on Robotics and Automation (ICRA)*, 2007, pp. 3313–3318.

[47] A. Nakhaei and F. Lamiraux, "A framework for planning motions in stochastic maps," in *Proc.of the International Conference on Control, Automation, Robotics and Vision (ICARCV)*, 2008, pp. 1959–1964.

[48] L. Guibas, D. Hsu, H. Kurniawati, and E. Rehman, "Bounded uncertainty roadmaps for path planning," in *Workshop on Algorithmic Foundations of Robotics (WAFR)*, ser. Springer Tracts in Advanced Robotics, vol. 57. Springer, 2008, pp. 199–215.

[49] D. Hsu, J. C. Latombe, and H. Kurniawati, "On the probabilistic foundations of probabilistic roadmap planning," *International Journal of Robotics Research (IJRR)*, vol. 25, no. 7, pp. 627–643, 2006.

[50] R. A. Knepper and M. T. Mason, "Real-time informed path sampling for motion planning search," *International Journal of Robotics Research (IJRR)*, vol. 31, no. 11, pp. 1231–1250, 2012.

[51] C. Stachniss, G. Grisetti, and W. Burgard, "Information gain-based exploration using rao-blackwellized particle filters," in *Proc.of the Robotics: Science and Systems (RSS)*, 2005, pp. 65–72.

[52] S. M. LaValle, *Planning Algorithms*. Cambridge, UK: Cambridge University Press, 2006.

[53] H. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. Cambridge, MA: MIT Press, 2005.

[54] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *International Journal of Robotics Research (IJRR)*, vol. 30, no. 7, pp. 846–894, 2011.

[55] L. Torabi, "Integrated view and path planning for a fully autonomous mobile-manipulator system for 3d object modeling," Ph.D. thesis, Simon Fraser University, 2011.

[56] P. Lehner, A. Sieverling, and O. Brock, "Incremental, sensor-based motion generation for mobile manipulators in unknown, dynamic environments," in *Proc.of the IEEE International Conference on Robotics and Automation (ICRA)*, May 2015, pp. 4761–4767.

[57] C. Dornhege and A. Kleiner, "A frontier-void-based approach for autonomous exploration in 3d," in *Proc.of the IEEE International Symposium on Safety, Security, and Rescue Robotics*, November 2011, pp. 351–356.

[58] L. Torabi and K. Gupta, "An autonomous six-dof eye-in-hand system for in-situ 3d object modeling," *International Journal of Robotics Research (IJRR)*, vol. 31, no. 1, pp. 82–100, January 2012.

[59] L. Torabi and K. Gupta, "An autonomous 9-dof mobile-manipulator system for in situ 3d object modeling," in *video Proc.of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, Portugal, 2012, pp. 4540–4541.

[60] S. Shen, N. Michael, and V. Kumar, "Autonomous indoor 3d exploration with a micro-aerial vehicle," in *Proc.of the IEEE International Conference on Robotics and Automation (ICRA)*, 2012, pp. 9–15.

[61] V. Pilania and K. Gupta, "A hierarchical and adaptive mobile manipulator planner," in *Proc. of the IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, Madrid, Spain, November 2014, pp. 45–51.

[62] V. Pilania and K. Gupta, "A hierarchical and adaptive mobile manipulator planner with base pose uncertainty," *Autonomous Robots*, vol. 39, no. 1, pp. 65–85, June 2015.

[63] V. Pilania and K. Gupta, "A localization aware sampling strategy for motion planning under uncertainty," in *Proc.of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015, (accepted, to appear).

[64] V. Pilania and K. Gupta, "Localization aware sampling and connection strategies for incremental motion planning under uncertainty," *Autonomous Robots*, 2015, (accepted: December 10, 2015).

[65] V. Pilania and K. Gupta, "Mobile manipulator planning under uncertainty in unknown environments," *International Journal of Robotics Research (IJRR)*, 2015, (under review).

[66] Y. Yu and K. Gupta, "C-space entropy: A measure for view planning and exploration for general robot-sensor systems in unknown environments," *International Journal of Robotics Research*, vol. 23, no. 12, pp. 1197–1223, 2004.

[67] L. Torabi, M. Kazemi, and K. Gupta, "Configuration space based efficient view planning and exploration with occupancy grids," in *Proc.of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2007, pp. 2827–2832.

[68] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Y. Ng, "ROS: an open-source robot operating system," in *Proc. Open-Source Software workshop of the International Conference on Robotics and Automation (ICRA)*, 2009.

[69] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, 2012. [Online]. Available: http://ompl.kavrakilab.org

[70] H. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. Cambridge, MA: MIT Press, 2005, ch. 7, pp. 212–214.

[71] H. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. Cambridge, MA: MIT Press, 2005, ch. 7, pp. 242–246.

[72] Z. Yao and K. Gupta, "Path planning with general end-effector constraints," *Robotics and Autonomous Systems*, vol. 55, no. 4, pp. 316–327, 2007.

[73] D. Berenson, S. Srinivasa, D. Ferguson , and J. Kuffner, "Manipulation planning on constraint manifolds," in *Proc.of the IEEE International Conference on Robotics and Automation (ICRA)*, 2009, pp. 625–632.

[74] M. Stilman, "Task constrained motion planning in robot joint space," in *Proc.of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2007, pp. 3074–3081.

[75] B. Yamauchi, "A frontier-based approach for autonomous exploration," in *Proc.of the Computational Intelligence in Robotics and Automation*, 1997, pp. 146–151.

[76] A. Leeper, K. Hsiao, E. Chu, and J. K. Salisbury, "Using near-field stereo vision for robotic grasping in cluttered environments," in *Intl. Symposium on Experimental Robotics*, 2010.

[77] M. Ciocarlie, K. Hsiao, E. G. Jones, S. Chitta, R. B. Rusu, and I. A. Sucan, "Towards reliable grasping and manipulation in household environments," in *Intl. Symposium on Experimental Robotics*, ser. Springer Tracts in Advanced Robotics, vol. 79. Springer, 2010, pp. 241–252.

[78] D. Kragic and H. Christensen, "Robust visual servoing," *International Journal of Robotics Research*, vol. 22, pp. 923–939, 2003.

[79] S. Chitta, E. G. Jones, M. Ciocarlie, and K. Hsiao, "Mobile manipulation in unstructured environments," *IEEE Robotics and Automation Magazine*, vol. 19, no. 2, pp. 58–71, 2012.

[80] A. Collet, M. Martinez, and S. S. Srinivasa, "The MOPED framework: Object recognition and pose estimation for manipulation," *International Journal of Robotics Research*, vol. 30, no. 10, pp. 1284–1306, 2011.

[81] B. Gerkey, "SLAM gmapping," http://wiki.ros.org/gmapping, 2015, [Online; accessed June 21, 2015].

[82] L. Kneip, F. TÃćche, G. Caprari, and R. Siegwart, "Characterization of the compact hokuyo URG-04LX 2D laser range scanner," in *Proc.of the IEEE International Conference on Robotics and Automation (ICRA)*, 2009, pp. 1447–1454.

[83] D. Fox, W. Burgard, F. Dellaert, and S. Thrun, "Monte carlo localization: Efficient position estimation for mobile robots," in *Proc.of the National Conference on Artificial Intelligence*, 1999, pp. 343–349.

[84] S. Kullback, *Information Theory and Statistics*. New York: Wiley, 1959.

[85] N. Roy and S. Thrun, "Online self-calibration for mobile robots," in *Proc.of the IEEE International Conference on Robotics and Automation (ICRA)*, 1999, pp. 2292–2297.

[86] T. N. Y. Jr. and C. R. Shelton, "Simultaneous learning of motion and sensor model parameters for mobile robots," in *Proc.of the IEEE International Conference on Robotics and Automation (ICRA)*, 2008, pp. 2091–2097.

# Appendix A

# Mobile Base Belief Estimation using BRM Approach

This background information is included primarily for completeness and is largely taken from [26]. We use extended Kalman filter (EKF) to estimate the state of mobile base, in which the state distribution is assumed to be Gaussian. The next state $s_t$ and observation $z_t$ are given by the following equations,

$$s_t = g(s_{t-1}, u_t, w_t), \qquad w_t \sim N(0, W_t) \tag{A.1}$$

$$z_t = h(s_t, q_t), \qquad q_t \sim N(0, Q_t) \tag{A.2}$$

where $u_t$ is a control action, $w_t$ and $q_t$ are random, unobservable noise variables. The EKF computes the state distribution at time $t$ in two steps: a process step and a measurement step. The process step follows as

$$\bar{\mu_t} = g(\mu_{t-1}, u_t), \qquad \bar{\Sigma_t} = G_t \Sigma_{t-1} G_t^T + V_t W_t V_t^T \tag{A.3}$$

where $G_t$ is the Jacobian of $g$ with respect to $s$ and $V_t$ is the Jacobian of $g$ with respect to $w$. For convenience, we denote $R_t = V_t W_t V_t^T$. Similarly, the measurement step follows as:

$$\mu_t = \bar{\mu_t} + K_t(H_t \bar{\mu_t} - z_t), \qquad \Sigma_t = (I - K_t H_t)\bar{\Sigma_t} \tag{A.4}$$

where $H_t$ is the Jacobian of $h$ with respect to $s$ and $K_t$ is known as the Kalman gain, given by

$$K_t = \bar{\Sigma_t} H_t^T (H_t \bar{\Sigma_t} H_t^T + Q_t)^{-1} \tag{A.5}$$

we denote $M_t = H_t^T Q_t^{-1} H_t$.

## A.1 Belief Updating as a One-Step Operation

Briefly, *one* descriptor matrix $S_{1:T}$ to compose the filter updates for $T$ time steps (a sequence of controls and measurements) along an edge between two nodes $i$ and $j$ can be calculated using the operator Redheffer star product (denoted with a '$\star$') as,

$$S_{1:T} = \begin{bmatrix} G_{1:T} & R_{1:T} \\ -M_{1:T} & G_{1:T}^T \end{bmatrix} = S_1 \star S_2 \star ... \star S_T. \tag{A.6}$$

where $S_t$ is given by the star product of control update scattering matrix $S_t^C$ and measurement update scattering matrix $S_t^M$

$$S_t^C = \begin{bmatrix} G & R \\ 0 & G^T \end{bmatrix}_t \qquad S_t^M = \begin{bmatrix} I & 0 \\ -M & I \end{bmatrix}_t \tag{A.7}$$

$$S_t = S_t^C \star S_t^M = \begin{bmatrix} G & R \\ -M & G^T \end{bmatrix}_t \tag{A.8}$$

We can then compute posterior covariance $\Sigma_T$ from initial condition $\Sigma_0$ using

$$\begin{bmatrix} \cdot & \Sigma_T \\ \cdot & \cdot \end{bmatrix} = \begin{bmatrix} I & \Sigma_0 \\ 0 & I \end{bmatrix} \star S_{1:T} \tag{A.9}$$

(The matrix element $\cdot$ are irrelevant to the final solution for the covariance).

## A.2 Motion Model and Sensor Model

Below we present the linearized version of motion and sensor models for use in EKF. Note that, for readability, we omit time index subscripts; however, all matrices derived are time-varying quantities.

We use the following non-linear probabilistic motion model with the assumption that the drive and turn commands are independent [85],

$$g_x = x + D\ cos(\theta + R) \tag{A.10}$$

$$g_y = y + D\ sin(\theta + R) \tag{A.11}$$

$$g_\theta = (\theta + R)\ \ mod\ 2\pi \tag{A.12}$$

where $g_x$, $g_y$ and $g_\theta$ are the components of $g$ corresponding to each state variable, and the control variable $u_t$ is given by $u_t = [D\ R]^T$ where $D$ and $R$ denote the robot's translation and rotation, respectively.

In the EKF, the state transition matrix $G$ is the Jacobian of the motion model with respect to the state, and is computed by linearizing the state transition function $g$ about the mean state $\mu$.

$$G = \begin{bmatrix} 1 & 0 & -D\ sin(\mu_\theta + R) \\ 0 & 1 & D\ cos(\mu_\theta + R) \\ 0 & 0 & 1 \end{bmatrix} \tag{A.13}$$

The linearized process noise in state space is computed as $R = VWV^T$ where $W$ is the covariance matrix of the noise in control space

$$W = \begin{bmatrix} \sigma_D^2 & 0 \\ 0 & \sigma_R^2 \end{bmatrix} \tag{A.14}$$

and $V$ is the motion noise matrix mapped from control to state space, computed as the Jacobian of the motion model with respect to the control space components

$$V = \begin{bmatrix} cos(\mu_\theta + R) & -D\ sin(\mu_\theta + R) \\ sin(\mu_\theta + R) & D\ cos(\mu_\theta + R) \\ 0 & 1 \end{bmatrix} \tag{A.15}$$

Now we explain the sensor model. The sensor model for 2D range sensor used in our experiments is given by $z^i = h^i(x) + N^i(0; Q)$, superscript $i$ denotes $i^{th}$ ray in the range scan

$$r^i = \sqrt{(x - x_{r^i})^2 + (y - y_{r^i})^2} + N^i(0; Q) \tag{A.16}$$

where $(x, y, \theta)$ is the robot pose, $x_{r^i}, y_{r^i}$ is the obstacle location where $i^{th}$ ray hit the object, $r^i$ is the range of $i^{th}$ ray. The linearized transformation from measurement space to state space is computed as the measurement Jacobian $H^i$, computed as the partial derivatives of the measurement function $h^i(x)$ with respect to each component of the state. The measurement Jacobian at time $t$ is given by

$$H_t^i = \begin{bmatrix} \frac{(x - x_{r^i})}{\sqrt{(x-x_{r^i})^2 + (y-y_{r^i})^2}} & \frac{(y - y_{r^i})}{\sqrt{(x-x_{r^i})^2 + (y-y_{r^i})^2}} & 0 \end{bmatrix} \tag{A.17}$$

Note that $(x - x_{r^i}) = r^i cos(\theta_m)$ and $(y - y_{r^i}) = r^i sin(\theta_m)$, where $\theta_m = atan2(y - y_{r^i}, x - x_{r^i})$ is the angle of measurement relative to the robot pose. Therefore, Equation A.17 becomes

$$H_t^i = \begin{bmatrix} cos(\theta_m) & sin(\theta_m) & 0 \end{bmatrix} \tag{A.18}$$

The measurement noise covariance $Q$ at time $t$ for $i^{th}$ sensor ray is

$$Q_t^i = \begin{bmatrix} \sigma_{r^i}^2 \end{bmatrix} \tag{A.19}$$

Note that in Chapter 3, where beacons are used to demonstrate our LAS and LAC strategies, we use a different sensor model as mentioned in [26]. Also for learning of motion model and sensor model parameters, we refer the reader to [85] and [86].

# Appendix B

# Probabilistic Completeness Proof for RRBT-LAS

In this section, we provide a formal proof that a planner with our localization aware sampling strategy is probabilistically complete for DistTH less than or equal to half of the inscribed radius of the robot.

The worst case situation that leads to probabilistic completeness issues with our approach is using RangeModel 1 where the sensors (beacons in our case) have limited range. In that case the heuristic used in our sampling strategy will limit the samples to only one (within ball of radius DistTH) for regions with low uncertainty reduction. This is where the completeness issue arises. If the value of DistTH is large then the planner that uses our sampling strategy may not be able to find a path. We show that if we keep DistTH below half of the inscribed radius of the robot (a reasonable assumption) then if there exists a collision-free path, a planner that uses our sampling strategy will also find one. For the proof we assume that the entire path passes through regions with low uncertainty reduction (a worst case scenario for our sampling strategy). Also note that our proof builds along the lines of [71], therefore, we follow most of their notations.

Suppose $q_s,\ q_g \in C_{free}$ (free region of C-space) are two robot configurations that can be connected by a path in $C_{free}$. RRBT-LAS is considered to be probabilistically complete, if for any given $(q_s, q_g)$

$$\lim_{n\to\infty} Pr[(q_s, q_g)FAILURE] = 0 \tag{B.1}$$

where $Pr[(q_s, q_g)FAILURE]$ denotes the probability that RRBT-LAS fails to answer the query $(q_s, q_g)$ after a roadmap in $C_{free}$ with $n$ samples has been constructed. The outline of the probabilistic completeness proof is as follows: First we assume that a path $\pi$ from $q_s$ to $q_g$ exists. We then tile the path with a set of carefully chosen balls such that generating a sample in each ball ensures that these samples can be connected with appropriate collision-free edges and hence a collision-free path, $\hat{\pi}$ between $q_s$ and $q_g$ will be found by RRBT-LAS and the probability of generating such samples approaches 1 with increasing $n$.

Assume a path $\pi$ (of length $L$) from $q_s$ to $q_g$ exists in $d$ dimensional C-space. The clearance of $\pi$, denoted $\rho = clr(\pi)$, is the farthest distance away from the path at which a given point

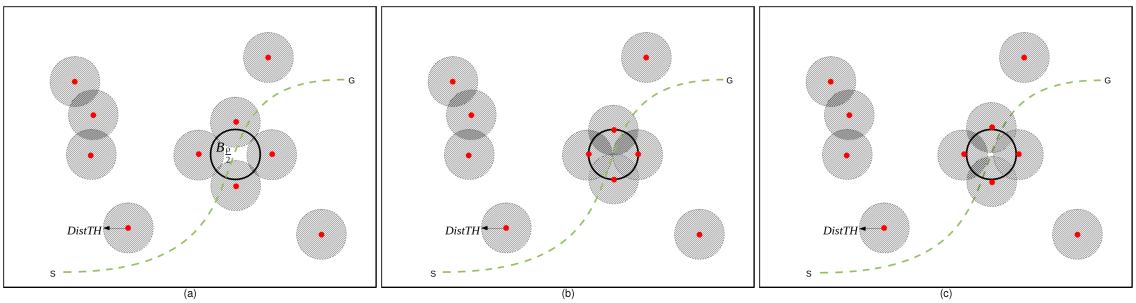



Figure B.1: [Case $DistTH \leq \frac{\rho}{2}$] - black colour (bold) circle denotes one of the balls $B_{\frac{\rho}{2}}(q_i)$ that is used to tile a path, red colour dots represent randomly placed samples, and hatched region (of radius DistTH=$\frac{\rho}{2}$) around each sample denotes the restricted region where samples can not be placed according to heuristic used in localization aware sampling. This figure shows the situation (excluding b) where none of the samples has yet been placed inside the black ball. (a) neighbouring samples around $B_{\frac{\rho}{2}}(\cdot)$ restrict some region (hatched areas inside the black ball) inside the black ball where samples can not be placed, (b) neighbouring samples totally covered $B_{\frac{\rho}{2}}(\cdot)$ but in that case samples lie on the periphery (closed set), (c) samples are placed at a distance $d$ such that $\frac{\rho}{2} < d < \frac{\rho}{2} + \epsilon$, even in this worst case scenario the probability of generating a sample in $B_{\frac{\rho}{2}}(\cdot)$ is greater than 0 (see text for explanation).

can be guaranteed to be collision-free. Note that $\rho \geq 2r$, where $r$ is the inscribed radius of the robot. The measure $\mu$ denotes the volume of a region of space, e.g, $\mu(B_\epsilon(x))$ measures the volume of an open ball $B_\epsilon(x)$ of radius $\epsilon$ centered at $x$. If $A \subset C_{free}$ is a measurable subset and $x$ is a random point chosen from $C_{free}$, then

$$Pr(x \in A) = \frac{\mu(A)}{\mu(C_{free})} \tag{B.2}$$

We now tile the path $\pi$ with balls each of radius $\frac{\rho}{2}$. Let $m = \lceil \frac{2L}{\rho} \rceil$ and observe that there are $m$ points (centers of balls) on the path such that $dist(q_i, q_{i+1}) < \frac{\rho}{2}$, where $dist$ is a Euclidean metric on $\mathbb{R}^d$. Let $y_i \in B_{\rho/2}(q_i)$ and $y_{i+1} \in B_{\rho/2}(q_{i+1})$. Then the line segment $\overline{y_i y_{i+1}}$ must lie inside $C_{free}$ since both endpoints lie in the ball $B_\rho(q_i)$. An illustration of this basic fact is given in Figure 7.17 of [71]. Let $V \subset C_{free}$ be a set of $n$ configurations generated by our localization aware sampling strategy. If there is a subset of configurations $\{y_1, ..., y_m\} \subset V$ such that $y_i \in B_{\rho/2}(q_i)$, then each ball will get a sample and a path from $q_s$ to $q_g$ will be found. Let $I_1, ..., I_m$ be a set of indicator variables such that each $I_i$ witnesses the event that there is a $y \in V$ and $y \in B_{\rho/2}(q_i)$. It follows that RRBT-LAS succeeds in answering the query $(q_s, q_g)$ if $I_i = 1$ for all $1 \leq i \leq m$. If at least one of the indicator variables is 0 then RRBT-LAS would fail. Therefore, the probability of failure (Equation B.1) then can be written as

$$Pr[(q_s, q_g) FAILURE] \leq Pr\left( \bigvee_{i=1}^{m} I_i = 0 \right) \tag{B.3}$$

$$\leq \sum_{i=1}^{m} Pr[I_i = 0] \tag{B.4}$$

where the last inequality follows from the union bound. We now mainly focus on the computation of $Pr[I_i = 0]$ for $i^{th}$ ball, i.e., after placing $n$ samples by our localization aware sampling strategy what is the probability that none of these samples lie in a ball $B_{\rho/2}(q_i)$.

For RangeModel 1, in regions outside the sensor range where there is no sensor information, hence no uncertainty reduction, our localization aware sampling strategy does not allow another sample within the vicinity (DistTH) of an already placed sample point (see Fig B.1). Therefore, the probability of failure to generate a second sample in a ball $B_{\rho/2}(q_i)$ depends on where the first sample was placed and so on. Let $I_i^1, ..., I_i^n$ be a set of indicator variables for the $i^{th}$ ball such that each $I_i^k$, for all $1 \leq k \leq n$, witnesses the event that the $k^{th}$ sample does not lie in ball $B_{\rho/2}(q_i)$. These events are dependent on each other. Below we provide the expressions to compute $Pr[I_i^k = 0]$ that will lead us to the computation of $Pr[I_i = 0]$. For the first two samples, the probability of failure to generate a sample inside ball $B_{\rho/2}(q_i)$ can be written as

$$Pr[I_i^1 = 0] = \left\{1 - \frac{\mu(B_{\rho/2}(q_i))}{\mu(C_{free})}\right\} \tag{B.5}$$

$$Pr[I_i^2 = 0] = \int Pr(I_i^2 = 0 \mid x^1) Pr(x^1) dx^1 \tag{B.6}$$

In above expression $x^1$ denotes the position of first sample. Above expression is just the marginalization over the position of first sample. Similarly, the expression for the third sample is

$$Pr[I_i^3 = 0] = \iint Pr(I_i^3 = 0 \mid x^1, x^2) Pr(x^2 \mid x^1) Pr(x^1) dx^2 dx^1 \tag{B.7}$$

and for $n^{th}$ sample the expression (for $Pr[I_i^n = 0]$) is

$$= \int \cdots \int Pr(I_i^n = 0 \mid x^1, ..., x^{n-1}), ..., Pr(x^2 \mid x^1) Pr(x^1) dx^{n-1} dx^{n-2}, ..., dx^2 dx^1 \tag{B.8}$$

Clearly, parameters $\frac{\rho}{2}$ (radius of ball $B$) and DistTH (restricted region around a sample) are embedded in above expressions. Using Equations B.5-B.8, Equation B.4 can now be written as

$$Pr[(q_s, q_g) FAILURE] \leq \left\lceil \frac{2L}{\rho} \right\rceil \left( \prod_{k=1}^{n} Pr[I_i^k = 0] \right) \tag{B.9}$$

Note that $Pr(I_i^n = 1 \mid x^1, ..., x^{n-1})$ denotes the probability of generating the $n^{th}$ sample inside ball $B_{\rho/2}(q_i)$ given that $n - 1$ samples have been placed. This is nothing but the ratio of volume of white region inside the black ball (after excluding the hatched region) and total volume of white region (with reference to Fig B.1). In general, this can be written as

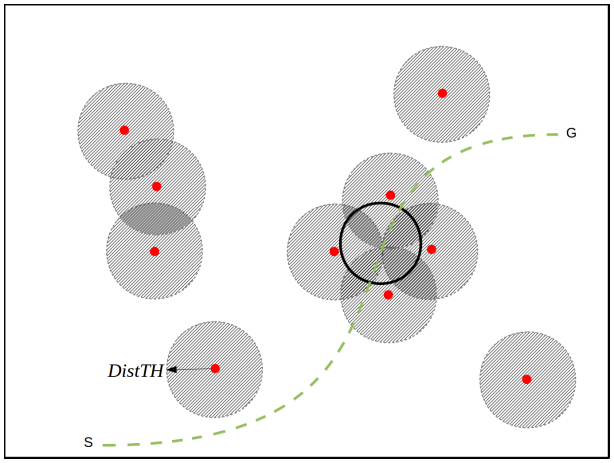



Figure B.2: [Case $DistTH > \frac{\rho}{2}$] - This figure shows that if $DistTH > \frac{\rho}{2}$ then the restricted regions of neighbouring samples may completely block the ball $B_{\rho/2}(q_i)$ and will prevent generation of a sample inside it. This will lead to the failure of a planner that uses our localization aware sampling strategy.

$$Pr(I_i^n = 1 \mid x^1, ..., x^{n-1}) = \frac{\mu({}^B C_{free}^R)}{\mu(C_{free}^R)} \tag{B.10}$$

where $C_{free}^R$ denotes the $C_{free}$ left after excluding the restricted regions around already placed samples and ${}^B C_{free}^R$ denotes the same but inside ball $B_{\rho/2}(q_i)$. This ratio approaches one as more and more samples are placed outside the black ball. That implies that $Pr(I_i^n = 0 \mid x^1, ..., x^{n-1})$ approaches zero. The expression in Equation B.8 is one of the product terms in RHS of inequality B.9. Convergence of $Pr[I_i^n = 0]$ (to 0) will lead to the convergence of RHS of inequality B.9. Therefore, $Pr[(q_s, q_g)FAILURE]$ converges to 0 as the number of samples increases, hence showing the completeness of RRBT-LAS. The same completeness can not be guaranteed for $DistTH > \frac{\rho}{2}$ (see Fig B.2).

# Appendix C

# Index to Multimedia Resources

Table C.1: Table of Multimedia Extensions

| Extension | Type | Description |
|---|---|---|
| 1 | Video | HAMP demonstration corresponding to scenario B (simulation) |
| 2 | Video | HAMP demonstration corresponding to scenario C (simulation) |
| 3 | Video | HAMP demonstration corresponding to scenario D (simulation) |
| 4 | Video | HAMP demonstration corresponding to scenario E (simulation) |
| 5 | Video | HAMP-U demonstration on SFU mobile manipulator |
| 6 | Video | HAMP-BAU demonstration using autonomous system for mobile pick-and-place task in unknown environment (simulation) |
| 7 | Video | HAMP-BAU-TC demonstration using autonomous system for mobile pick-and-place task in known environment (simulation) |
| 8 | Video | HAMP-BAU demonstration using autonomous system for mobile pick-and-place task in unknown environment (real experiment on SFU Mobile Manipulator) |
| 9 | Video | HAMP-BAU-TC demonstration using autonomous system for mobile pick-and-place task in known environment (real experiment on SFU Mobile Manipulator) |