12-1-2015

High-Dimensional Motion Planning and Learning Under Uncertain Conditions

Nick D. Malone

Nick Malone

Candidate

Computer Science

Department

This dissertation is approved, and it is acceptable in quality and form for publication:

Approved by the Dissertation Committee:

Dr. Lydia Tapia, *Chairperson*

Dr. John Wood *co-chair*

Dr. Deepak Kapur

Dr. Melanie Moses

Dr. Meeko Oishi

High-Dimensional Motion Planning and Learning Under Uncertain Conditions

by

Nick Malone

B.S. Computer Science, University of Tulsa, 2007
M.S. Computer Science, University of Tulsa, 2009

DISSERTATION

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Doctor of Philosophy
Computer Science

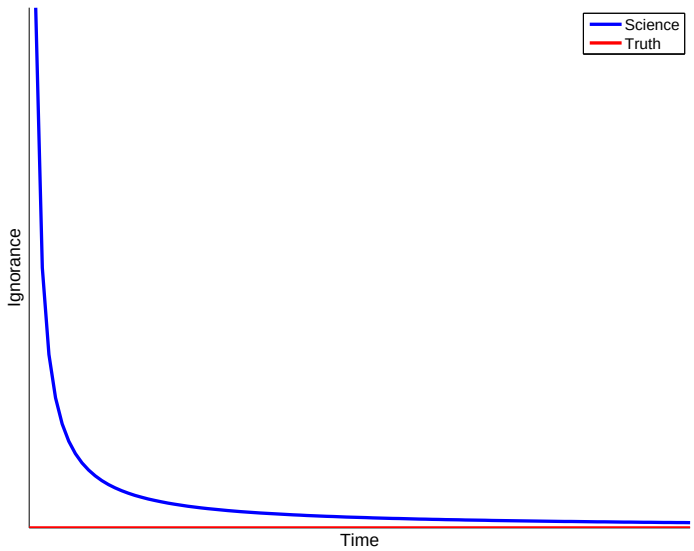
The University of New Mexico

Albuquerque, New Mexico

December, 2015

Dedication

- To everyone who has shown me the paths blazed before me.



- Dr. James Aldridge

Acknowledgments

I would like to thank my advisors, Dr. Lydia Tapia and Dr. John Wood. Their guidance and support has made everything done here possible. I would also like to thank my committee members Dr. Deepak Kapur, Dr. Melanie Moses and Dr. Meeko Oishi. I would like to particularly thank Dr. Meeko Oishi for ensuring that we did everything as mathematically correctly as possible.

I would also like to thank my contributors on my works, Dr. Kendra Lesser, Kasra Manavi, Lewis Chiang, Dr. Aleksandra Faust, Dr. Brandon Rohrer, and Prof. Ron Lumia. Their combined efforts have turned our rough ideas into something truly useful for the robotics community. A special thanks to Kendra for working with us path planners and providing us with support on the control theory code base.

Finally, I would like to thank my family and friends for putting up with all the madness and last minute plan changes needed during my PhD program. First, to my wife for understanding when I had to spend all night making last changes on papers. Also, to my parents for teaching how to think and for passing on their knowledge and understanding of the world. I have only achieved what I have due to standing on the shoulders of those who came before me.

BECCA development was supported by the Laboratory Directed Research and Development program at Sandia National Laboratories for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. Support was provided by contract SNL PO-1074659, titled "Learning for WAMs", under the direction of Dr. Brandon Rohrer. Prof. L. Tapia was supported in part by the National Institutes of Health (NIH) Grant P20RR018754 to the Center for Evolutionary and Theoretical Immunology. Partial support was provided by contract SNL-3 PO #1110220, titles Automated Systems Research, issued to the UNM Manufacturing Engineering Program. This work was also funded in part by IIS-1528047.

High-Dimensional Motion Planning and Learning Under Uncertain Conditions

by

Nick Malone

B.S. Computer Science, University of Tulsa, 2007

M.S. Computer Science, University of Tulsa, 2009

Ph.D., Computer Science, University of New Mexico, 2015

Abstract

Many existing path planning methods do not adequately account for uncertainty. Without uncertainty these existing techniques work well, but in real world environments they struggle due to inaccurate sensor models, arbitrarily moving obstacles, and uncertain action consequences. For example, picking up and storing children's toys is a simple task for humans. Yet, for a robotic household robot the task can be daunting. The room must be modeled with sensors, which may or may not detect all the strewn toys. The robot must be able to detect and avoid the child who may be moving the very toys that the robot is tasked with cleaning. Finally, if the robot missteps and places a foot on a toy it must be able to compensate for the unexpected consequences of its actions. This example demonstrates that even simple human tasks are fraught with uncertainties that must be accounted for in robotic path planning algorithms. This work presents the first steps towards migrating sampling-based path planning methods to real world environments by addressing three different types of uncertainty: (1) model uncertainty, (2) spatio-temporal obstacle uncertainty (moving obstacles) and (3) action consequence uncertainty. Uncertainty is encoded directly into path planning through a data structure in order to successfully and efficiently identify safe robot paths in

sensed environments with noise. This encoding produces comparable clearance paths to other planning methods which are a known for high clearance, but at an order of magnitude less computational cost. It also shows that formal control theory methods combined with path planning provides a technique that has a 95% collision-free navigation rate with 300 moving obstacles. Finally, it demonstrates that reinforcement learning can be combined with planning data structures to autonomously learn motion controls of a seven degree of freedom robot arm at a low computational cost despite the number of dimensions.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Motivation

Uncertainty is an inevitable part of all physical robot systems. No system or task can be completely known. However, uncertainty is such a difficult problem that many autonomous methods for planning and learning are designed explicitly to exclude or substantially limit it. While limiting uncertainty works for small tasks and simple systems, it is inapplicable on autonomous robotic systems meant to operate in real world environments outside of a carefully controlled lab setting. The primary contribution of this work is a set of new motion planning algorithms which explicitly incorporate uncertainty into the planning process. These algorithms provide a stepping stone for migrating fully autonomous robotics from carefully controlled laboratory environments to the uncontrollable and unpredictable real world.

In recent years, fully autonomous robots have transitioned from the laboratory and controlled industrial settings to uncontrolled environments such as households and hospitals. For example, the Roomba has reached increased mar-

ket penetration into households across the world [29]. The military is seeking robots to lighten soldiers' loads with assistive robotics such as Boston Dynamics Big Dog [85], and with an ageing population in several countries, hospital robots like the Robot for Interactive Body Assistance (RIBA) [76] will become critical to health care. However, these technologies are still limited because of the difficulties caused by uncertainty. For example, the Roomba has circumvented the uncertainty problem by using a random search pattern, which can result in lost performance due to repeatedly cleaning the same section of floor [29]. This floor cleaning task contains some uncertainty but it does not require precision to perform at satisfactory levels, thus a random pattern is sufficient. Unfortunately, this approach is not applicable to many robotic applications such as dish washing or laundry folding because these tasks contain uncertainty and require precision.

These precise but more useful robotic applications will require more sophisticated motion planning methodologies. At an abstract level Motion planning is the task of finding a collision free path from some start state to some goal state. However, in order for motion planning to be helpful and useful, it will need to model the environments, consider situations arising from moving obstacles such as humans, and be able to adapt to the kinodynamic changes that were not or could not be accounted for in the original plan. Each of these tasks are subject to uncertainty.

Uncertainty arises from many sources including sensors, moving obstacles, and wear. This work is primarily concerned with the impact uncertainty has on the motion planning problem and methods to compensate for various types of uncertainty. As such, there are three primary sources of motion planning uncertainty in many robotic systems: 1) model uncertainty, 2) spatio-temporal obstacle uncertainty and 3) transition function uncertainty.

Model uncertainty arises from the process of using a sensor to reconstruct a

model of an environment. Many methods exist to *solve* the reconstruction task such as Simultaneous Localization and Mapping (SLAM) [6] or Simultaneous Planning and Mapping (SPAM) [106], but the solutions invariably produce some inaccuracies where the model does not correctly match the true environment. There are several sources of this error, but the primary error is due to sensor noise, with secondary factors due to the reconstruction algorithm [6]. Reconstruction is often done with vision sensors such as stero-vision, time of flight or structure light techniques. In the field of mapping and model reconstruction, there has always been an implication that sensor error will continue to decrease with the advent of new technologies. Even though sensors may eventually become near perfect, there will always be uncertainty in the reconstruction due to occlusion and perception. Even the human visual system cannot know the shape of occluded objects, and even fails on specific degenerate cases colloquially known as optical illusions. Thus, model uncertainty needs to be accounted for in the motion planning algorithm itself.

In dynamic environments with moving obstacles, another type of uncertainty emerges, namely spatio-temporal uncertainty [105]. This creates a whole new type of uncertainty that no improvement in sensor technology can mitigate. The primary problem is that a valid path planned at the current state of the environment could later be invalidated when an obstacle moves across the path. Many methods exist to attempt to solve this problem [84, 13, 109, 41, 86, 93], but most are restricted to a small number of simple moving obstacles because they 1) use expensive methods to predict obstacle motions or 2) do not consider stochastic uncertainty.

Finally, the last form of uncertainty considered applies when a robot takes an action. In motion-based robot learning, often a mapping from the current robot state (e.g. input from sensors) to the appropriate output (e.g. an action or mo-

tion) is unknown. This mapping from inputs to outputs is referred to as the transition function [56]. Transition function uncertainty occurs when the transition function is unknown or probabilistic. Slippage, wear, and momentum are the primary causes of transition function uncertainty for most designed robot systems [104]. However, the transition function can be completely unknown and must be learned via exploration. This type of transition function uncertainty is an active area of research and can be approached with a variety of techniques, including reinforcement learning (RL). Most RL methods need a human in the loop to tune the parameterization [88], which can be time consuming and difficult. In order to overcome this limitation, a technique called BECCA, a Brain Emulating Command and Control Architecture, is explored which generates its own features and tuning [88]. Each of these uncertainty types impacts the motion planning problem differently, but this work shows that uncertainty can be handled through direct encoding into the motion planning data structures.

Here, methods are presented that directly couple uncertainty handling with planning methods. Motion planning in perfectly known environments with perfectly characterized robots is itself a challenging problem, and has been shown to be P-Space hard [39, 40]. Any type of uncertainty merely complicates the problem further and each type of uncertainty creates a unique challenge. Sampling-based methods were initially developed because of the infeasibility of complete planning for moderately complex robots and environments. They trade optimality for efficiency by only operating on a small subset of all possible robot configurations. This subset is produced by sampling configurations. In many cases, this subset is sufficient to solve the motion planning problem quickly. However, in some difficult cases, these methods may take an indefinite amount of time to find a solution. It has been shown that many sampling-based methods are probabilistically complete, meaning that if a solution exists, the sampling method will eventually find it [52]. These methods traditionally work in environments without uncertainty

and have been shown to be highly successful at finding valid motion plans from a start state to a goal state, but suffer when uncertainty is present.

Specifically, three classes of methods are presented: (1) Safety-PRM for model uncertainty, (2) Stochastic Reachable (SR) sets combined with motion planning for moving obstacles, and (3) Reinforcment Learning (RL) combined with a Probabilistic Roadmap Method (PRM) for transition function uncertainty. Figure 1.1 shows a breakdown of the uncertainty sources and how they interact. A solution which can handle all three source of uncertainty simultaneously would provide the foundation for a robot to reason about motion-based uncertainty.

A PRM is a sampling-based method that probabilistically samples configurations to form vertices of a roadmap. These vertices are then connected by a local planning method and graph search is used to find a path from some start vetex to some goal vertex. Safety-PRM builds upon PRM by incorporating a probability of collision directly into the roadmap. This allows Safety-PRM to find paths with high clearance from the modeled obstacles and thus compensate for inaccurately modeled environments. For moving obstacles, a formal verification method, SR sets, is used to produce a likelihood estimate of collision for a moving obstacle. Then SR sets are combined with roadmap methods and Artificial Potential Field (APF) methods, which provides the planning methods with a more accurate model of the obstacle motion and allows for more informed paths to be constructed that successfully navigate environment cluttered with many moving obstacles. APF methods simply produce repulsion fields around obstacles and make local path planning decisions by following the local repulsion gradient. Finally, the BECCA RL is combined with PRM methods to create a method which learns how to navigate a roadmap for high degree of freedom robots. This combination provides a framework which allows the RL to converge efficiently and learn how to operate a robot without any *a priori* knowledge of the transition function.

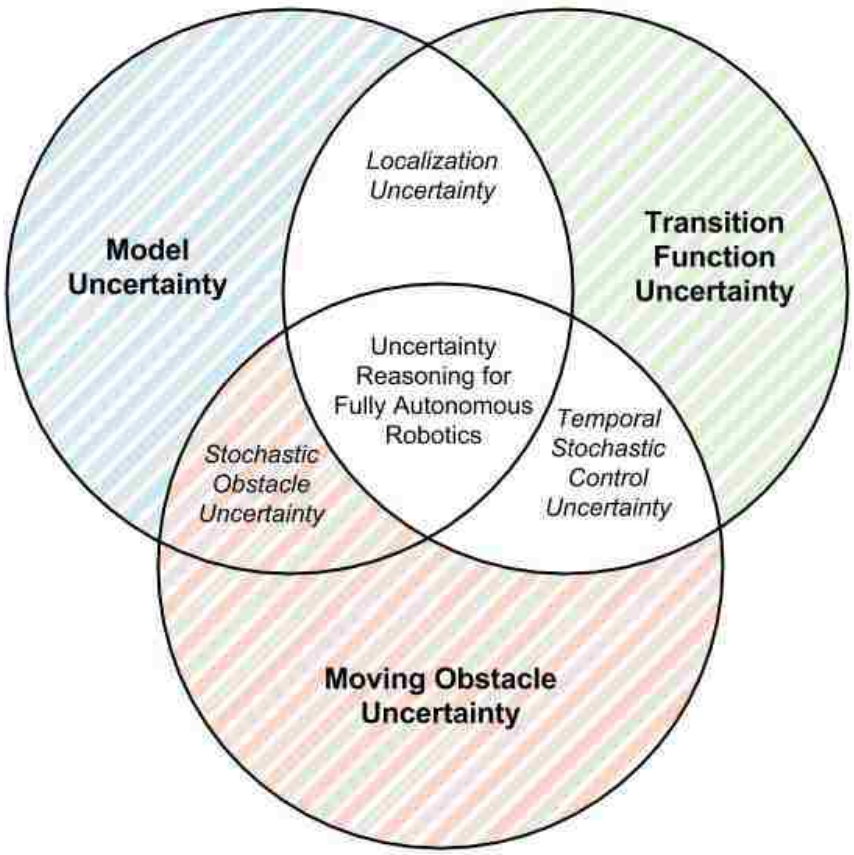


Figure 1.1: The three sources of robotic uncertainty (bold text) and the related problems (italic text). Highlighted areas are problems addressed in this thesis.

These three methods provide techniques, which allow a robot to path plan under varying types of uncertainty.

1.2 Contributions

The research presented here provides solutions for some of the most common types of uncertainty experienced during path planning methods. The body of this research is based on the following publications:

Journal Publications

- Nick Malone, Aleksandra Faust, Brandon Rohrer, Ron Lumia, John Wood, Lydia Tapia, "Efficient Motion-based Task Learning for a Serial Link Manipulator" Transactions on Control and Mechanical Systems, Vol. 3, Num. 1, Janaury 2014.

Conference Publications

- Hao-Tien Chiang, Nick Malone, Kendra Lesser, Meeko Oishi, Lydia Tapia, "Path-Guided Artificial Potential Fields with Stochastic Reachable Sets for Motion Planning in Highly Dynamic Environments" In IEEE International Conference on Robotics and Automation (ICRA), Seattle, Washington May 2015.
- Aleksandra Faust, Nick Malone, Lydia Tapia, "Preference-Balancing Motion Planning under Stochastic Disturbances," In IEEE International Conference on Robotics and Automation (ICRA), Seattle, Washington May 2015.
- Hao-Tien Chiang, Nick Malone, Kendra Lesser, Meeko Oishi, Lydia Tapia, "Aggressive Moving Obstacle Avoidance Using a Stochastic Reachable Set Based Potential Field," In International Workshop on the Algorithmic Foundations of Robotics (WAFR), Istanbul, Turkey, 3-5 August 2014.
- Nick Malone, Kendra Lesser, Meeko Oishi and Lydia Tapia, "Stochastic Reachability Based Motion Planning for Multiple Moving Obstacle Avoidance" In Proc. International Conference on Hybrid Systems: Computation and Control (HSCC), Berlin, Germany, April 2014.
- Nick Malone, Kasra Manavi, John Wood, Lydia Tapia, "Construction and Use of Roadmaps that Incorporate Workspace Modeling Errors," In

Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 1264-1271, Tokyo, Japan, November 2013.

- Nick Malone, Brandon Rohrer, Lydia Tapia, Ron Lumia, John Wood, "Implementation of an Embodied General Reinforcement Learner on a Serial Link Manipulator," IEEE International Conference on Robotics and Automation (ICRA), St. Paul, Minnesota, May 2012

Workshop Publications

- Nick Malone, Aleksandra Faust, Brandon Rohrer, John Wood, Lydia Tapia, "Efficient Motion-based Task Learning," Robot Motion Planning Workshop, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vilamoura, Portugal, October 2013.

- Aleksandra Faust, Nick Malone, Lydia Tapia, "Planning Preference-balancing Motions with Stochastic Disturbances", Machine Learning in Planning and Control of Robot Motion Workshop at IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Chicago, IL, September 2014.

Combined together this research describes, 1) Safety-PRM in chapter 3, 2) stochastic reachability set based obstacle avoidance in chapter 4, and 3) BECCA combined with PRMs in chapter 5. As such, it presents a set of methods for:

- Encoding of uncertainty into roadmaps for handling of model uncertainty (Safety-PRM)

- Validation of Safety-PRM on various error models

- Safety-PRM implemented on real robotic hardware
- Robot velocity modulation based on perceived likelihood of collision
- Path planning with moving obstacles using SR sets with several planning methods (PRM, APF, tree-based)
- Path planning with hundreds of moving obstacles
- Planning data structures used to learn motion-control task for high Degree of Freedom robots
- Transfer of learning from high fidelity simulation to a physical robot

Chapter 2

Related Work

Before discussing the specifics of uncertainty handling methods, the motion planning problem background and existing methodologies must be discussed. Section 2.1 describes the basic motion planning problem and some common techniques that have been developed and how this thesis work compares to the existing literature.

2.1 Motion Planning

Motion planning is a difficult problem. Many techniques utilize the concepts of configuration space, which maps the n degrees of freedom (DoF) of a robot to a point in an n dimensional space, and the concept of the workspace, which is the physical space in which the robot and environment exists (typically 2D or 3D). Configuration space (C-*space*) is a set consisting of all possible configurations of the robot [7]. C-*space* has two distinct subsets, C-*free* and C-*collision*. C-*free* is the set of all configurations which are not in collision with an obstacle or in self collision. C-*collision* is the inverse set. Unfortunately, C-*space* is often intractable to map

and path planning methods typically avoid directly mapping it. There are four primary categories of path planning methods: grid-based [12, 91, 92], geometric [8, 9], potential fields [33, 15, 54, 57, 18], and sampling-based [64, 109, 86, 46, 58, 77]. Here the focus is on potential field and sampling-based methods, as these two methods are known for low runtime cost and effectiveness in high degree of freedom problems respectively. The work presented in this thesis expands on potential field methods and sampling-based methods to handle various types of uncertainty.

Artificial Potential Field (APF) methods are a local path planning method. They operate by only considering the next step. These methods produce attraction forces towards the goal and repulsion forces away from obstacles [33, 15, 54, 57]. At each step these forces are summed together, and the robot moves along the force gradient. This computation is often done locally and thus is inexpensive to compute, which makes APF methods ideal for real time applications.

In contrast sampling-based techniques utilize global knowledge. Sampling-based techniques attempt to approximate the topology of the collision-free C-*space*. One class of sampling-based techniques works by building a graph (roadmap) in collision-free C-*space* through sampling collision-free robot configurations (*vertex generation*), connecting neighboring vertices with weighted edges if a collision-free transition exists (*vertex connection*), and then querying the resulting graph (roadmap) by finding a path to a goal configuration (*roadmap query*). Vertex generation can be done via several different methods, e.g., using a cell decomposition of the space [59], a uniform random distribution [51], obstacle boundaries [115], or visibility [94]. The utility of these various vertex generation methods varies with problem complexity. For example, cell decomposition methods are powerful, but their utility degrades with complex obstacle boundaries and in high-dimensional planning problems. On the other hand, uniform random place-

ment, Uniform PRM, works well with high-dimensional problems, but has difficulty with obstacles that form tight narrow passages. Collision-free tests are often performed with static obstacles, and edge weights can be determined by several metrics of interest, e.g., distance [51], clearance from obstacles [66] [74], or other problem-specific measures [95].

2.2 Validation

Degenerate tests are done by producing small test cases which the results can be determined via hand calculations. These test cases are then evaluated on the research code to determine if the results match the expected output. This validation technique is referred to as white-box testing, which tests the internal structures of algorithms instead of the functionality. Research code is constantly evolving and the results of the final output do not meet a specification. However, the internal components of the research code do have expected results. Thus, testing was done at a white-box level via degenerate test cases and fault injection.

2.3 Motion Planning with Uncertainty

Motion planning with uncertainty is the primary contribution of this work. To that end the existing methodologies must be discussed. There are many classes of sampling-based methods such as Probabilistic Roadmap Methods (PRMS) [64, 109, 86, 46], Rapidly Exploring Random Trees (RRTs)[58, 77], and Grid based method [59] to name a few. However, few methods were natively designed to solve the motion planning problem with uncertainty. A multitude of techniques have been proposed to compensate for uncertainty including clearance-

informed PRM methods, modifiable roadmaps and explicit planning with uncertainty. However, the first stage in utilizing these methods in the real world requires the environment to be modeled with sensors.

2.3.1 Modeling Environments With Sensors

Path planning algorithms for physical robots often plan using a model of the actual environment. The modeling process is one of the most challenging tasks in robotics [102]. Some common technologies used for modeling include: GPS, radar, laser, sonar and cameras. However, every technology is subject to error, measurement noise, which is not statistically independent and thus modeling is subject to systematic correlated errors [102].

Two commonly used techniques are RGB-D mapping and Laser range finders. RGB-D mapping uses a RGB camera with distance values for every pixel in the image [38, 43, 26, 97]. Either active stereo [55] or time of flight sensing is used [13]. The measurement noise from models created with RGB-D mapping varies depending on the method used [38]. Laser range finders use lasers to scan and map an environment. For example, [103] and [50] proposes a SLAM solution for mapping and a probabilistic method for localization but maps are subject to cumulative error.

2.3.2 Clearance-informed Roadmaps

The first set of existing methods which can be used to handle uncertainty are the clearance informed roadmap methods. Clearance-informed methods utilize obstacle information to create high quality paths, more efficient sampling or to handle moving obstacles. Obstacle surfaces are critical in methods such as Obstacle

Based PRM (OBPRM) [5] and Medial Axis PRM (MAPRM) [114]. In OBPRM, configurations are placed near the obstacle surfaces in order to traverse narrow passages easier. In MAPRM, configurations are placed on the medial axis of C_{free} to increase clearance and visibility. Here, random samples are generated and retracted towards the medial axis. However, in both methods a complete and accurate model of the environment is needed. Earlier methods related to MAPRM such as Generalized Voronoi Diagram and Hierarchical Generalized Voronoi Graph, were restricted to workspace clearance [30] [19] [21].

PRMs have also been adapted to handle changes in the environment due to moving obstacles. The work in [41] expands PRMs to work under both kinodynamic constraints and with moving obstacles. However, uncertainty is not built into the roadmaps, directly. Sensing errors are handled by growing the obstacles. The work in [86] also utilizes PRMs with moving obstacles. In this method, a first-stage approximate dynamic global roadmap about the connectivity is maintained and a second-stage path is extracted from the dynamic global roadmap to locally plan.

Another method, Toggle PRM [22], maps both C-*free* and C-*obstacle* (a subset of C-*collision* of space occupied by obstacles). Toggle PRM uses C-*free* and C-*obstacle* to aid sample efficiency in narrow passages.

All these methods can handle certain classes of uncertainty, but the methods do not directly encode and plan for the uncertainty. The nature of clearance informed methods allow them to be used for model uncertainty but they typically suffer from high computation costs and only work with certain classes of uncertainty as a byproduct of high clearance. The methods proposed in this thesis, however, directly consider the uncertainty in the planning process and produce comparable results at less computational cost.

2.3.3 Modifiable Roadmap Methods

The second set of sampling based methods which can be used with uncertainty are modifiable roadmap methods. Modifying a roadmap is a means to construct tunable roadmap paths, handle invalid paths and to accommodate moving obstacles. One type of modifiable roadmap, [95], constructs a coarse roadmap which is refined in the areas of interest relative to a query. The approach generates an approximate roadmap, postponing detailed validation until query time where query preferences are applied to customize the roadmap. For example, [34] takes an initial roadmap and query solution and adds vertices and edges to improve the query solution.

Deformable roadmaps such as [116] replan online paths by using deformation to fix invalid parts of a path. If a portion of a path is found to be in collision, the midpoint of the invalid portion is pushed a specified distance away from the obstacle. A similar approach in [48] looks at the path homotopy class, which relies on the notion of path deformability. This method only looks at homotopy classification and the possibility of deforming a given path to fit another.

The approaches of [84] and [13] address real-time obstacle avoidance in dynamic environments. These methods start with an initial path that is collision free and then incrementally modify the path to maintain a smooth, collision free path. These methods rely on workspace clearance by using *protective bubbles* to deform the path.

Again these methods can be used with uncertainty such as moving obstacles, however, they often suffer from high runtime computation costs. Instead of directly planning for the uncertainty these methods add a modification step to adjust existing roadmaps generated by existing techniques. In contrast, the methods presented in this thesis shift the computation cost to a pre-processing step that can

be done offline, which allows for minimal computation cost during path construction and traversal. This allows the presented methods to be highly reactive to the changing environment.

2.4 Moving Obstacles

Planning with both static and dynamic obstacles is complicated by the need for constant adjustments of plans to account for moving obstacles, yet critical in applications such as flight coordination and autonomous vehicles. In these dynamic environments, it is important to produce trajectories that avoid both static and dynamic obstacles with high success rates in a computationally efficient manner.

Common approaches to solving the motion planning problem for dynamic obstacles include Artificial Potential Field (APF) methods [33, 15, 54, 57, 18], tree based planners [58, 77], Probabilistic Roadmap Methods (PRMs) [64, 109, 86, 46], and several variants which use heuristics [3, 11].

APF methods create a potential landscape and use gradient descent for navigation, plan locally, and can be dynamically reactive to unexpected obstacles. These methods generate an artificial potential in the robot's workspace, which repels the robot from obstacles and attracts the robot to the goal [53]. They are applicable to several robotic problems, including unmanned aerial vehicles [15, 54], robot soccer [113], and mobile robots [33, 25, 107, 96]. For example, a recent APF method assigns non uniform repulsive bubbles around moving human obstacles to prevent robots from moving in front of a walking human [57].

Recent work has extended the APF method to account for cases in which the goal is not reachable due to obstacle proximity [33], and navigation in narrow passages is required [25]. Other recent work has focused on modification of the

computation of the potential field through fuzzy [96] and evolutionary [107] APFs. Another branch of work on APFs utilizes the repulsive and attractive concepts of APFs but also integrates another path planning method [47, 81]. For example, [47] uses a user defined costmap to influence vertex placement in a RRT algorithm. The costmap dictates a repulsiveness or attractiveness factor for every region. Similarly, Navigation Fields [81] assign a gradient which agents follow and is used for crowd modeling.

Roadmap-based techniques, including PRM variants, have been developed to address planning in spaces with moving obstacles [84, 13, 109, 41, 86, 93]. Generally, these approaches adapt to moving obstacles using one of two approaches. The first category generates a roadmap with little obstacle information, and later filters paths at runtime with local obstacle information [84], [13]. These methods have low precomputation costs, but generally prove expensive during path selection. They start with an initial path that is collision free and incrementally modify the path to maintain a smooth, collision free path. These methods only rely on physical obstacle clearance by using protective bubbles to deform the path.

The second category approximates the environment and is cheap at runtime. These methods create an approximate roadmap and then use a heuristic approach to produce locally valid paths to avoid moving obstacles. These methods decrease runtime costs at the expense of path accuracy [86], [46]. In [86], a first stage constructs a dynamic roadmap that considers some obstacles and is shared across multiple moving robots. Then, in a second stage, a path is extracted by a single robot that is locally modified to account for neighboring robots (moving obstacles). Similarly, [109] repairs the existing roadmap when an obstacle makes an edge or group of edges invalid. The authors of [116] use a roadmap, but deform the edges around moving obstacles. The work in [3] trades off distance from the goal and the dynamic obstacles to path plan. Approaches in [112] and [79] utilize

roadmap methods with heuristics to manage the moving obstacles, while [111] attempts to optimize the roadmap for moving obstacles under motion constraints. These existing moving obstacles methods suffer from either expensive runtime costs or do not consider obstacle motion in a rigorous manner. In contrast, this thesis demonstrates a method with low runtime costs which utilizes control theory as a foundation for the obstacle motion prediction. This allows the method to make more informed planning decisions.

2.4.1 Planning With Uncertainty

The two main types of uncertainty are model uncertainty and localization error. [14] extends PRMs to work while building a workspace model and is used to guide exploration to areas that have not been sensed, but it does not deal with measurement noise. The work in [44] and [82] is concerned with localization error of the robot. In contrast, [74] is concerned with model error, but it uses a probability of collision for rejection sampling of vertices in the roadmap. Furthermore the method is tailored to 2D environments where the model noise is quantifiable. In [16], the general PRM and RRT method is followed, however, the cost of connecting two vertices is evaluated through Monte Carlo simulations to deal with uncertainty. The work in [4] instead samples local motions at each state to estimate the state transition probability for each possible action. A roadmap and the state transition probabilities are then used to formulate a Markov Decision Process (MDP) which is then solved using Infinite Horizon Dynamic Programming.

Localization error can also be handled by working in belief space. In [2] the authors chose to model a 2D motion planning problem as a Partially Observable Markov Decision Process (POMDP). Belief space and POMDPs are also used to solve the uncertainty problem in [83]. Here the belief space is used to approximate

the solution to the POMDP on a 2D motion planning problem. Similarly, [56] handles restricted moving obstacles with a POMDP in real time.

Planning with uncertainty often is done with POMDP and Belief space methods. While these methods are theoretically sound they typically are impractical for large state-spaces or complex robots due to the high computational costs [83]. The primary drawback is that these methods tend to be exponential in the size of the state-space [83, 56]. In contrast, this work provides methods to plan with uncertainty at a lower computational cost than POMDP or belief space methods by directly considering uncertainty in the planning process. Directly incorporating uncertainty into the planning algorithms themselves provides superior solutions in terms of clearance and success rates compared to methods which do not consider uncertainty.

2.5 Stochastic Reachability

Another method of handling moving obstacles is to incorporate a formal method from control theory. Stochastic reachability (SR) analysis provides offline verification of dynamical systems, to assess whether the state of the system will, with a certain likelihood, remain within a desired subset of the state-space for some finite time, or avoid an undesired subset of the state-space [1]. To solve problems in collision avoidance, the region in the relative state-space, which constitutes collision is defined as the set of states the system should avoid [98, 49]. SR sets provide a formally grounded estimate of the probability of collision between a robot and a particular obstacle. This probability of collision can be combined with ad-hoc path planning methods such as Probabilistic Roadmap Methodsand Artificial Potential Fields to produce predictive path planning methods.

SR is based upon the concepts of reachability calculations, which determines control inputs to avoid collisions with deterministic obstacle dynamics. A Hamilton-Jacobi-Bellman (HJB) formulation [75] allows for both a control input and a disturbance input to model collision-avoidance scenarios [69], [35] for motion planning. The result of the HJB reachability calculations is a maximal set of states within which collision between two objects is guaranteed (in the worst-case scenario), also known as the reachable set. The set which assures collision avoidance is the complement of the reachable set. In [100], reachable sets are calculated to assure a robot safely reaches a target while avoiding a single obstacle, whose motion is chosen to maximize collision, and the robot cannot modify its movements based on subsequent observations. In [24], a similar approach is taken, but reachable sets are computed iteratively so that the robot can modify its actions. In [60], multiple obstacles that act as bounded, worst-case disturbances are avoided online, based on precomputed invariant sets.

An alternative approach is to calculate a SR set that allows for obstacles whose dynamics include stochastic processes. Discrete-time SR generates probabilistic reachable sets [1] based on stochastic system dynamics. In [98], the desired target set is known, but the undesired sets that the robot should avoid are random and must be propagated over time. In [49], a two-player stochastic dynamical game is applied to a target tracking application in which the target acts in opposition to the tracker.

Chapter 3

Modeling Uncertainty: Inaccurate Workspaces

This chapter presents a method for explicitly planning with modeling uncertainty. The work presented here is based on [67] and [65].

The motion planning problem consists of finding a valid (collision-free) path from a start state to a goal state. One solution to this problem is to capture the topology of the collision-free portion of the configuration space. Probabilistic Roadmap Methods (PRMs) provide a solution by constructing a roadmap of randomly sampled robot configurations [51]. Collision free configurations are kept, while collision configurations are rejected. Connections are made between two configuration samples when a collision-free transition can be made. These samples (vertices) and connections (edges) define a graph, referred to as a roadmap, that the robot can safely traverse. Recently, PRMs have been extended to be adapt-

able [84] [13]. These new methods can deform paths [48], update roadmaps due to moving obstacles [41] [86], map both collision and collision-free states [22], and deal with uncertainty in the motion model [16] [4] [2] [83]. However, despite all these advances, roadmap construction in PRMs require that the model of the problem must be accurate, e.g., there must be a clear delineation between collision and collision-free states. This workspace models must be accurate because collision detection is done by mapping a sampled configuration back into the workspace, and then checking to see if the robot is in collision with any obstacles in the workspace. Inaccurate collision detection due to an inaccurate work space model can lead to erroneous roadmaps which produce feasible paths in the modelled environment but lead to collisions in the actual world.

Figure 3.1: Whole Arm Manipulator (WAM) touching an obstacle boundary.

Distinguishing between collision and collision-free configurations requires a model of the planning space. These models are often manually constructed, have well-defined obstacle boundaries, and can be easily tested for robot-obstacle collision. Advancing technologies are producing 3D environment models at lower costs than ever before [102]. These models are constructed using technology such as sensors [102] and cameras [38]. However, all these technologies are prone to

modeling error. Therefore, unlike the manually modelled environment, obstacle boundaries can be fuzzy or approximated thus making collision tests error-prone.

Safety-PRM, a planning method that accounts for modeling uncertainty in the roadmap, was introduced in [67]. This method calculates and incorporates a probability of collision during roadmap construction. These probabilities reflect the amount of certainty in the collision-state of a vertex or edge in the roadmap, thus allowing the robot to have an expectation of safety from an obstacle's surface. The certainty can also be used to weigh roadmap edges, thus allowing robots to easily transition between being safer (higher expected clearance) or take shorter paths (lower regard to expected clearance). The focus of the previous work was on validating Safety-PRM against other planning methods. To that end simulated noise was used to construct the roadmap. However, the method was designed specifically for the problem of sensor noise. Here, Safety-PRM is demonstrated in a much more challenging setting with noisily modelled environments and environments reconstructed from sensor data.

The applicability of this method is demonstrated on a series of environments with rigid body and articulated linkage robots. Of particular interest is how these methods are affected by different error models and error amounts. To evaluate this, a true model is constructed in simulation, and then a series of deformed models is generated from that true model to represent a sensor reconstructing the true environment. Path planning is then conducted in the deformed (*sensed*) environments and the paths are evaluated in the true environment. In most rigid body cases and in all Linkage cases, Safety-PRM can generate roadmaps with less computational cost than basic PRM and MAPRM (methods known for low computational cost and clearance maximization, respectively).

Safety-PRM also is particularly relevant to experimental robot systems. In this chapter, Safety-PRM is demonstrated on a Barrett Whole Arm Manipulator

(WAM). First, Safety-PRM is compared to Uniform PRM and MAPRM on an environment model generated by the Kinect sensor [73]. A mapping of expected safety to robot speed is developed, so that the robot can safely test the validity of configurations using torque estimation.

3.1 Safety-PRM Method

In order to handle environment models with inaccuracies, the basic PRM method must be modified. In the basic PRM, collision checking of the robot to obstacles in the environment is done as a binary check (either free or in collision). Often this is done because sensing technologies are constantly improving and will hypothetically produce models approaching 100% accuracy. However, current sensing technology still produces noisy data. In order to handle an environment with noise, Safety-PRM stores a probability of collision with each configuration. This probability is a function of the clearance or penetration to the nearest obstacle to the configuration. These probabilities are also used to guide connection and to find feasible paths. Instead of using raw collision probability, a weighing function between collision probability and distance is used to allow path tuning. Therefore, Safety-PRM provides flexible methods for tuning between planning goals (expected clearance and path length), works on many robot types (rigid bodies and linkages), and is inexpensive to compute.

3.1.1 Vertex Generation

The first step in PRM methods is generating a set of samples that approximates C_{free}. In an environment modelled with noise, the boundary between C_{free} and $C_{obstacle}$ is fuzzy. Thus, unlike a standard PRM method, in collision vertices are not

discarded. Rather, a probability of validity is associated with each vertex, which is dependent on its expected distance from the obstacle surface. This ensures that not all vertices are weighted with an equal measure of quality.

A vertex is generated by sampling a configuration, X. Here uniform random sampling is used but other sampling techniques can also be utilized. Then a probability of collision is stored with each vertex based on the amount of perceived clearance (negative clearance is penetration). The clearance probability, P_v(X), of configuration X is calculated in Equation 3.1.

$$P_v(X) = \frac{-1 * atan(D(X) - 1) + \frac{\pi}{2}}{\pi} \tag{3.1}$$

Where, $D(X)$ is the distance of configuration X to the nearest obstacle surface in the noisy model in units. This is just one example to produce a collision probability due to noisy obstacle boundaries, which can be tuned to expected sensor error. The $\frac{\pi}{2}$ shifts the equation so that being close to the obstacle, but not necessarily in collision, has a high collision probability (e.g., $P(0) = 0.75$). This provides an extra buffer around obstacle surfaces and causes the algorithm to favor higher clearance vertices. Figure 3.2 shows a plot of the probability function defined in Equation 3.1. Equation 3.1 is a fabricated functions that merely demonstrates the algorithm. This function was found empirically to work well with the error model used in the experiments. However, this function can be tuned to match the actual error function of the sensors used to model the the workspace.

Since the introduction of the first PRM method [51], there have been many PRM variants introduced including [5] and [114]. Since a probability of collision is associated to all samples, configurations are generated using a uniform random distribution. This method is able to produce many samples quickly at a low computational cost. However, many PRM sampling variants could be used for sample

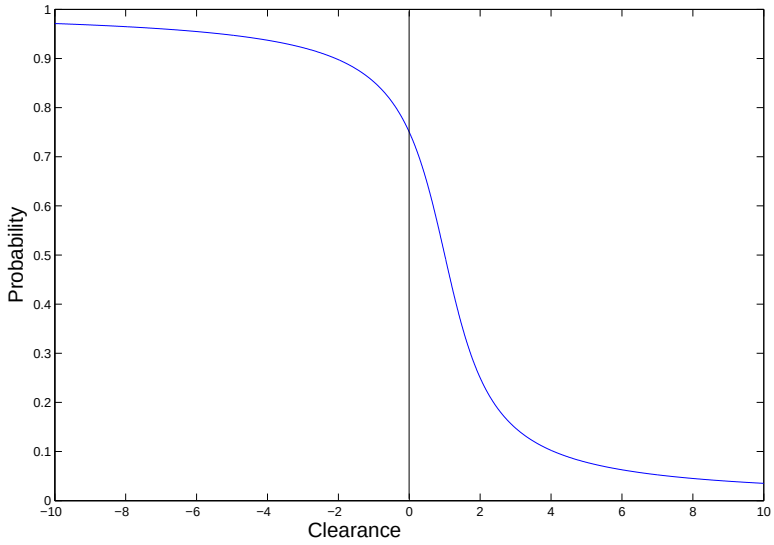


Figure 3.2: Probability of a configuration being in collision based on the clearance of the configuration X, $(D(X))$. Based on Equation 3.1.

generation.

3.1.2 Vertex Connection

For each vertex in the roadmap, an attempt is made to connect it with its k nearest neighbors with a local planner. However, since the collision status of the configuration is only partially known, the definition of nearest neighbor considers the probability of collision. Thus, a function (equation on X) is defined that combines distance and probability of collision. This causes the best candidate neighbors to likely be free and proximate. Note, that an edge between neighbors consists of a sequence of configurations. The equation, $(d(c_i, c_j))$, provides a calculation of the distance between two configurations, c_i and c_j. In this equation λ is a weighting term, c_i is the vertex, $P_v(c_j)$ is the probability that c_j is in collision, and $dist(c_i, c_j)$ is the Euclidean distance of c_j from vertex c_i.

$$d(c_i, c_j) = (\lambda)P_v(c_j) + (1 - \lambda)dist(c_i, c_j) \tag{3.2}$$

This metric will provide smaller scores to neighbors which are close and have a low probability of being in collision. The k closest neighbors are then chosen for connection. $P_v(c_j)$ is evaluated by using Equation 3.1 where the input is the clearance of configuration c_j.

In the results shown, edges are computed between neighbors c_i and c_j by using a straight-line in C_{space}. The weight for the edge e_{ij} (such that e_{ij} is the edge between configurations c_i,c_j) is a function of the probability of collision, $P_e(e_{ij})$, and the length of an edge. The probability of collision of an edge, $P_e(e_{ij})$, is a function of the intermediate configurations along the edge $c_i = c_0, c_1, c_2, ..., c_{n-1}, c_n = c_j$, where the number of intermediate configurations depends on the resolution, a parameter of the method. Here, the probability of collision of an edge e, P_e, is the maximum probability of any configuration along that edge, e.g. $P_e(e_{ij}) = max(P_v(c_0), P_v(c_1), P_v(c_2), ..., P_v(c_{n-1}), P_v(c_n))$, . The probability of a collision for each configuration along the edge, $P_v(c_i)$, is computed using Equation 3.1.

Equation 3.3 shows how the weight is calculated for an edge between configuration c_i and configuration c_j. This edge is denoted e_{ij}. The parameter γ allows for customizable scaling between clearance and edge length. This is particularly important because modeling errors can be highly variable.

$$Weight(e_{ij}) = (\gamma)P_e(e_{ij}) + (1 - \gamma)nlen(e_{ij}) \tag{3.3}$$

Where $nlen(e_{ij})$ is a normalized length of an edge. In the results shown, these values are normalized by the maximum edge length in order to provide intuitive scaling between probabilities of collision and edge length.

Queries are then done using the standard Dijkstra's algorithm on the weighted roadmap [23]. Since the roadmap edge weights capture the uncertainty, Dijkstra's algorithm will choose paths with the least uncertainty to the goal.

3.2 Experiments

Safety-PRM is evaluted under three different experimental environments. The first set of experiments (Section 3.3) compare Safety-PRM to Uniform PRM [51] and MAPRM [114] with a simulated noise model. MAPRM is known for creating high clearance paths because configurations are placed on the medial axis of C_{free}, and it does this by retracting generated random samples towards the medial axis. Uniform PRM is the traditional rejection sampling method under a uniform random sampling distribution. The second set of experiments (Section 3.4) evaluates the three methods under more realistic conditions and with varying error model types and amounts. These conditions are created by building a *true* environment and then applying an error model to produce a deformed model which represents a *sensed* environment. Roadmap construction and path planning are then done on the *sensed* environment, while paths are validated on the *true* environment for collision. Finally, the three methods are evaluated on a physical robot system (Section 3.5). Here, a model of the environment is constructed with a Kinect sensor, roadmap construction and path planning are performed on the reconstructed model, and then the path is evaluated on the physical robot in the real environment for collisions.

3.3 Simulated Noise with Rigid Bodies and Linkages

This set of experiments constructs a roadmap, path plans, and validates with simulated noise. This simple setup demonstrates the affect of the parameters on the Safety-PRM method without other confounding effects. It also show that Safety-PRM produces path of similar quality to MAPRM but with fewer collision detection calls.

The value of γ in Equation 3.3 determines the trade-off between short paths and paths with high expected clearance. Thus, experiments with varying γ values are shown. Each value of γ is shown for 10 runs with 10 different random seeds. The parameter used to identify neighbors is fixed at $\lambda = 0.75$. This value was empirically found to make well-connected roadmaps at $k = 5$, k being the number of neighbors each vertex has. Safety-PRM was implemented within the Parasol Motion Planning Library (PMPL) developed at Texas A&M University. Experiments were run on a single core of an Intel 3.40 GHz CORE i7-2600 CPU and 8 GB of RAM.

3.3.1 Environments

Safety-PRM is explored with rigid body and articulated linkage robots in two environments. Figure 3.3 depicts the environments. In each environment the query is designed to show the trade-off in paths with high collision-free probability (clearance) versus path length by varying γ in the edge weighting function.

- **Narrow:** consists of an elongated environment with three boxes dividing the space. The first and second boxes produce a narrow corridor while the second and third box produce a wide corridor. The query is built so that the narrow corridor has a shorter path to the goal but higher probability of

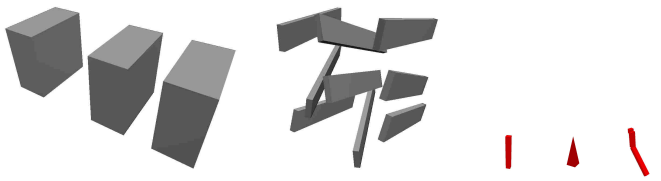


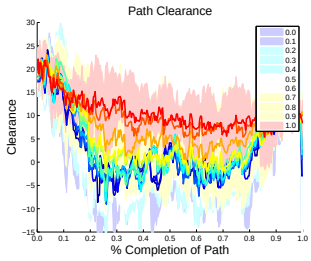
Figure 3.3: Environments: (a) Narrow and (b) Plank and Robots (c) Stick, (d) Big Arrow, and (e) Linkage.

being in collision, while the wide corridor has a low probability of being in collision but a longer path length.

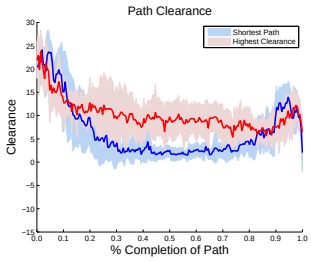
- **Plank:** The Plank environment has several long planks running the same direction but with minor offsets in their angles. This produces several narrow corridors through which the robot must navigate length wise and transverse across.

Each environment was run with three different robots. These robots were selected to demonstrate the robustness of Safety-PRM under different planning conditions. However, graphs are only shown for a single robot in each environment because Safety-PRM's quality was unaffected by the robot type.

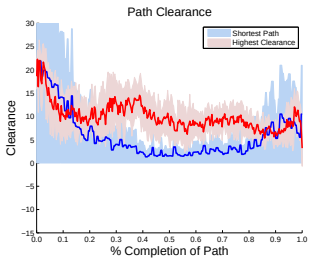
- **Stick:** is a long thin rigid body object. Its long side is too long to fit through most of the paths, but when oriented correctly it can pass through most of the passages with ease.
- **Big Arrow:** The Big Arrow is a rigid body pyramid and will just narrowly fit through many of the passages.
- **Linkage:** The Linkage is a serial three link robot mounted on a pivot which can be in any orientation and location in 3-space. Each joint can be moved independently for a total of nine degrees of freedom.



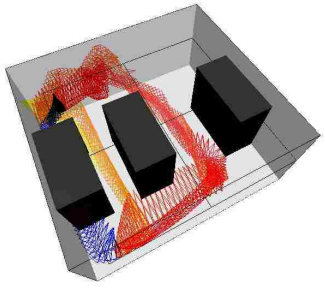
(a) Safety-PRM Clearance



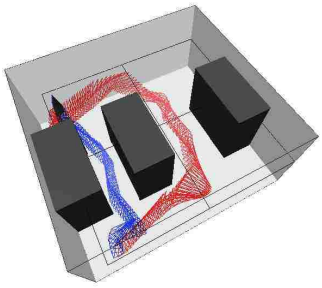
(b) MAPRM Clearance



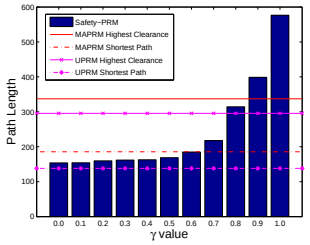
(c) Uniform PRM Clearance



(d) Safety-PRM Clearance Paths



(e) MAPRM Clearance Paths



(f) Path Lengths

Figure 3.4: Narrow Environment with the Big Arrow robot and 500 vertices. The paths have been normalized to the range of $[0, 1]$ so that multiple runs can be compared. 0 is the start of the query while 1 is the goal configuration of the query. The shaded regions indicate the standard deviation over 10 runs for each experiment. For 3.4d and 3.4e Each color indicates a different path based on clearance. Red is the highest clearance path and blue is the shortest path. For MAPRM the shortest and highest clearance paths are shown.

In order to demonstrate modeling error, a simple error model is used in the simulations. Error is introduced into every collision detection test. The error is modelled as $\pm$ 5% of the maximum reach of the robot to scale the problem with the robot. The error model used most closely matches a sensor that would produce uniform errors. While this simplified model does not exactly match the error one would see from sensed environments, it approximates the error enough to demonstrate this approach. However, the goal of Safety-PRM is to create a roadmap which can compensate for many types of sensor error models.

3.3.2 Rigid Bodies

Path quality and performance are the two metrics used to evaluate Safety-PRM. Path quality is determined by path length and clearance. Unfortunately, in certain planning problems these two parameters can be at odds with each other. Paths with high clearance can have longer path lengths and short paths can have lower clearance because of the obstacles in the environment. Performance is determined primarily by execution time which is directly related to the number of vertices, edges and collision detection calls. Here path clearance is measured as the distance of the robot (at a certain configuration) to the nearest obstacle surface, with penetration being negative. The algorithm performances are primarily impacted by the number of collision detection calls, as it is an expensive atomic operation during roadmap construction. Collision detection is done by mapping a configuration of a robot back into the workspace and then checking, in the workspace, if the robot intersects any of the obstacles (or itself in the case of linkages).

Figure 3.4a shows the path clearances for $\gamma = [0, 1.0]$ with a step size of 0.1 on a roadmap of 500 vertices. Figure 3.4f shows the path lengths for the corresponding γ values. The graphs indicate the γ values under 0.7 produce very poor paths. However, γ values above 0.7 produce significantly different successful paths (paths which reach the goal without collision). Figure 3.4d shows the Safety-PRM paths for $\gamma = 0.0, 0.6, 0.7, 0.8, 0.9, 1.0$. Figures 3.4a and 3.4d directly show the tunability as $\gamma = 0.7$ goes through the shortest path and as the γ value increases beyond 0.7 the paths have slightly higher clearance. Note that for $\gamma > 0.7$ all the paths go through the higher clearance section of the environment. These results show that γ provides a mechanism to tune the algorithm for short but low clearance path or for longer but higher clearance paths.

However, the tunability is not the primary contribution of this work. Primarily,

Safety-PRM is concerned with an efficient way to produce high clearance paths in an inaccurate model. 3.4e shows the shortest path and the highest clearance path for a roadmap of 500 vertices built with MAPRM. These paths are comparable to the paths produced by Safety-PRM in figure 3.4d. In this particular experiment, Safety-PRM produces slightly higher clearance paths but more importantly Safety-PRM often produces these paths at lower cost, i.e. with fewer Collision Detection (CD) Calls. In sampling methods the number of Collision Detection calls is a primitive operation that reflects the computational time spent by the algorithm. Table 3.1 for the Big Arrow robot at 500 vertices shows that Safety-PRM makes 72% fewer CD calls than MAPRM. For the Big Arrow in the narrow environment Safety-PRM makes at worst 45% fewer CD calls and at best 75% fewer CD calls. In this environment Safety-PRM is able to produce a high clearance path compared to MAPRM but at a cheaper cost. MAPRM run to completion is also shown in Table 3.1. The goal is to produce a cheap reusable roadmap which well approximates the C_{space} of the inaccurately modelled environment, so high expected clearance paths can be found. Running MAPRM to completion produces roadmaps which solve the query but are not necessarily general. For example, Table 3.1 shows MAPRM producing at most 33 vertices for any robot environment combination. This is due to the way MAPRM pushes vertices to the medial axis, thus allowing it to find a path with few vertices.

For the slightly more complex Plank environment, Safety-PRM performs slightly worse than on the narrow environment. In the Plank environment, Table 3.1 shows that Safety-PRM requires 14% more CD calls than MAPRM for a roadmap with 100 vertices. Note that the number of vertices impacts how well the roadmap approximates the topology of C-*space*. However, for more than 100 vertices Safety-PRM does better than MAPRM. For more than 100 vertices Safety-PRM does at worst 18% fewer CD calls than MAPRM (500 vertices) and at best 40% fewer CD calls (2000 vertices). For the one instance where Safety-PRM does

worse than MAPRM, MAPRM on rigid bodies is able to use workspace clearance and the fact that on 100 vertices MAPRM has 40% fewer edges than Safety-PRM while for the other vertex values MAPRM has between 24% to 14% fewer edges. Furthermore, since MAPRM is using this workspace clearance it is able to push vertices to the medial axis in a single step and this environment essentially has three hallways thus favoring MAPRM's medial axis solution as the solution path lies along the hallways. Also, this is not using inflated obstacles. If these were sensed obstacles, the standard way of using MAPRM would be to inflate the obstacle boundaries, however, doing so could potentially lose the narrow passageways. It is important to note that for rigid bodies and roadmaps greater than 500 vertices, Safety-PRM is always cheaper than Uniform PRM. Safety-PRM is cheaper than Uniform PRM because it does not have a rejection step for in-collision vertices. Uniform PRM must resample configurations which are in-collision and thus incur more computational overhead.

3.3.3 Linkages

Now the difficulty of the problem is increased by using Linkage robots. In order for MAPRM to accurately push configurations to the medial axis it must use an approximate ray casting solution [114]. The proposed method does not need to use such a technique as a probabilistic encoding of the clearance ($P_v(X)$) is used.

In this experiment, a 3 link planar robot mounted on a pivot in 3 space is used. Each link is connected via parallel revolute joints, to form a simple planar linkage. These joints are then mounted on a free floating pivot which can be oriented and located freely in 3D. Thus, the robot has 9 Degrees of Freedom, 3 position, 3 orientation and 3 joint angles. Orientation and position is relative to the first joint in the linkage. The Linkage experiments are run in the Narrow and Plank

environments with the same starting position and goal position as the rigid body experiments. The only exception is that the joints are set to be slightly offset from fully extended.

Similar to the rigid body experiments, Safety-PRM produces comparable if not higher clearance paths than MAPRM. Figure 3.5a shows the clearances for Safety-PRM and 3.5b shows the clearances for MAPRM for a roadmap of 500 vertices. These two graphs show that Safety-PRM produces paths that are comparable to MAPRM. However, Table 3.1 shows that Safety-PRM makes between 95% to 98% fewer CD calls than MAPRM. Similarly, to demonstrate tunability, Figure 3.5g shows the path lengths for varying γ values. Figure 3.5a shows that as γ increases the path clearance become higher.

Figures 3.5d, 3.5e and 3.5f show the paths for Safety-PRM, MAPRM and Uniform PRM, respectively. The MAPRM and Uniform graphs show both the shortest path and the highest clearance path in the roadmap, while Safety-PRM shows paths for $\gamma = 0.6, 0.7, 0.8, 0.9, 1.0$. These graphs show that Uniform PRM produces less smooth paths than either MAPRM or Safety-PRM. Similarly, it shows that the paths produced by Safety-PRM are comparable to the paths produced by MAPRM. It is important to note that while results for MAPRM to completion are shown, the task is not simply trying to solve one query. The goal is to produce an inexpensive roadmap with high expected clearance, which approximates the C_{space} of the environment in order to solve multiple queries in an inaccurate workspace model.

These experiments show that the computational cost of Safety-PRM is comparable to MAPRM with rigid body robots and significantly less expensive than MAPRM in the Linkage environments. They also demonstrate that Safety-PRM produces comparable quality paths in terms of clearance to MAPRM. These two advantages are due to Safety-PRM being built to take inaccurate models into ac-

count. This means that given an inaccurate model, Safety-PRM will produce relatively high expected clearance paths with less computational cost than MAPRM. MAPRM is capable of producing high clearance path in the inaccurate workspace models because it pushes configuration to the medial axis, but it does not directly take the uncertainty into account. Similarly, Uniform PRM does not consider the uncertainty but with these small amounts of error can still sometimes find solutions. In general, Safety-PRM can produce high-clearance path with inaccurate workspace models without added computation cost.

		Safety-PRM			MAPRM			Uniform PRM		
Robot	Env	Vertices	Edges	CD's	Vertices	Edges	CD's	Vertices	Edges	CD's
Big Arrow	Narrow	Comp	N/A	N/A	33	122.0	8,385.7	25	100.4	2,437.4
		100	1,173.4	15,255.5	100	435.6	27,758.7	100	570.6	12,171.5
		500	4,725.2	52,470.2	500	2,250.2	184,380.1	500	3,666.0	65,983.9
		1000	8,703.0	90,103.4	1000	4,764.6	306,870.6	1000	7,854.8	128,984.7
		1500	13,170.0	130,688.2	1500	7,381.0	508,205.1	1500	12,178.4	188,379.4
		2000	17,688.4	170,725.3	2000	10,010.6	652,091.8	2000	16,594.2	245,356.5
Stick	Plank	Comp	N/A	N/A	14	44.0	1,872.1	11	37.6	1,542.3
		100	904.0	20,702.4	100	551.79	18,158.8	100	539.2	20,082.9
		500	4,551.2	81,534.0	500	3,472.4	99,259.1	500	3,367.4	101,648.8
		1000	9,152.6	149,258.1	1000	7,472.0	208,658.0	1000	7,261.4	201,228.4
		1500	13,767.0	213,742.3	1500	11,621.8	329,380.4	1500	11,256.2	300,961.3
		2000	18,420.0	277,172.4	2000	15,858.6	458,286.9	2000	15,409.2	401,185.2
Linkage	Narrow	Comp	N/A	N/A	Comp	61.8	266,273.7	Comp	66.6	7,099.5
		100	1,516.4	101,187.6	100	478.6	1,934,083.5	100	1,649.2	139,243.9
		500	7,709.2	427,025.7	500	2,565.0	10,285,723.8	500	5,455.0	432,761.8
		1000	15,568.8	807,674.6	1000	5,244.0	20,001,180.5	1000	8,634.1	655,001.1
		1500	23,413.2	1,167,438.8	1500	7,959.0	30,300,016.4	1500	11,667.7	854,130.0
		2000	31,313.2	1,517,071.4	2000	10,676.4	39,538,284.2	2000	15,292.2	1,077,846.8
Linkage	Plank	Comp	N/A	N/A	Comp	40.2	303,423.2	Comp	49.0	3,221.4
		100	1,627.3	68,258.7	100	353.8	2,485,202.3	100	382.0	25,191.7
		500	8,332.6	271,513.3	500	2,238.0	10,958,301.2	500	2,331.2	149,392.9
		1000	16,695.6	491,709.8	1000	4,739.8	20,904,573.9	1000	4,993.6	330,395.5
		1500	25,191.2	705,611.7	1500	7,320.2	31,503,340.8	1500	7,781.0	533,506.3
		2000	33,653.8	905,726.3	2000	9,975.6	41,851,387.7	2000	10,635.8	747,872.1

Table 3.1: Rigid Body and Linkage Experiments for different environments and robots with the number of neighbors for each vertex set to 5. Vertices and edges indicate the number of vertices and edges in the roadmap. CD's are the number of collision detection calls made during roadmap construction.

3.4 Varying Level of Noise in the Environment Model

A model of the physical environment is often extracted through sensing, which introduces modeling error. Path planning is done using the sensed model, but, because of the modeling error, these paths could potentially cause collision.

In these simulation experiments, the effect of different error models and error amounts are evaluated on Safety-PRM Uniform PRM, and MAPRM. To perform this evaluation, a known ground truth environment is produced in simulation. This ground truth environment is then distorted via a specific error model. The distortion is parameterized by type (Spherical, Gaussian, or Log-Normal) and amount of distortion. The distorted environment simulates the error introduced by sensing an environment. Path planning is then done on the distorted environments. The path is then migrated to the ground truth environment and evaluated for clearance and collisions. If the path does not have any collisions it is considered valid. All of these experiments are done in simulation. The ground truth is a simulated environment and the sensed environment is built by modifying the simulated ground truth environment with an error model.

Experiments were conducted on three different error models in simulation. Experiments were performed on the Narrow and Plank environments with all three robot types (Stick, Big Arrow and Linkage robots). However, since the results for all three robots and both environments were similar and all showed the same trends, for brevity, data is only shown from the Narrow environment and the Big Arrow robot. The previous section showed the results of all three robot types planning and executing in the same environment. Here, similar trends for planning in a *sensed* environment but executing in *true* environments are demonstrated. Of primary concern is the success rate for the different methods, however runtime costs were also compared.

3.4.1 Error Models

An obstacle model in the workspace is made up of triangles. The triangles have vertices and edges which define a location and orientation in space. To create a

simulated *sensed* obstacle model the vertices which make up the triangles of the model are distorted using the error models described below.

Spherical Error: The spherical error model distorts each vertex in the true model by a uniformly random distance capped by the radius of the sphere. This model is parameterized by the radius r. However, this type of distortion is unrealistic as adjacent vertices are uniformly randomly distorted. To adjust for this the model is then smoothed with a simple Laplacian smoothing function. The Laplacian smoothing function is parameterized by n, the number of passes. Figure 3.6b shows an example of a true model distorted by the Spherical error model.

Gaussian Error: The Gaussian error model also distorts each vertex in the true model however, the distortion is done by randomly sampling a Gaussian function. The model is parameterized by μ and σ, and is sampled in three dimensions. Figure 3.6c shows an example of Gaussian distortion. This is the traditional error modelled assumed in most work on noise as many sensors and processes can be modelled via Gaussian error.

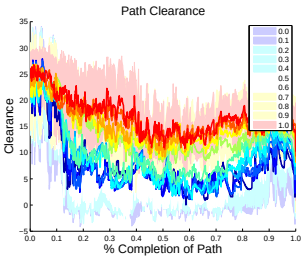
Log-Normal Error: Log-normal error is similar to the Gaussian error, except it has a heavy tail. Again each vertex is distorted by sampling a Log-Normal function. It is parameterized by μ and σ and is also sampled in three dimensions. Figure 3.6d shows an example of the log-normal distortion.

In general, the spherical error causes the most sharp angles in the distorted models, while the Log-Normal model causes the fewest sharp angles. This is due to the distribution probabilities. The Spherical model is uniform random while the Log-Normal model has a very high probability of sampling a point near the mean and low probability of sampling points far from the mean.

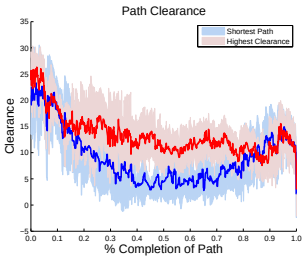
3.4.2 Environments

Figure 3.3 depicts the *true* environments while Figure 3.6 shows an example of *sensed* (distorted) environments. In each environment the query is designed to show the trade-off in paths with high collision-free probability (clearance) versus path length by varying γ in the edge weighting function. The true environments are distorted using the various error models to produce a facsimile of a *sensed* environment. The true environments and robots are explained in detail in Section 3.3.1.

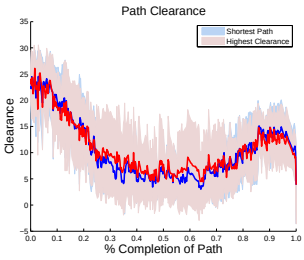
In the following experiments Safety-PRM is compared to MAPRM and uniform PRM on varying amounts of error. Specifically $r = 2, 6, 10$ for the spherical error model and $\sigma = 2, 6, 10$ for the Gaussian and log-normal error models. All vertices are distorted by offsetting from the true vertex location. The paths produced in the distorted model are then tested in the known ground truth. If the path is collision free it is considered valid, otherwise it is considered invalid. Each environment is run with different roadmap sizes, (500, 1000, 2000). For each roadmap, γ is tested from 0 to 1 with an increment of 0.1. Finally, for each test, 10 runs are performed with different random seeds to produce different roadmaps. This minimizes the influence on the success rate due to roadmap differences.



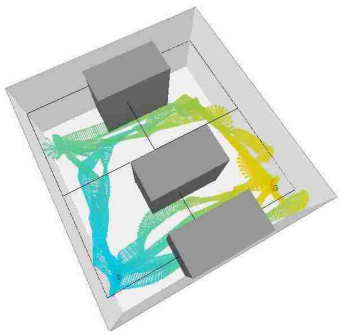
(a) Safety-PRM Clearance



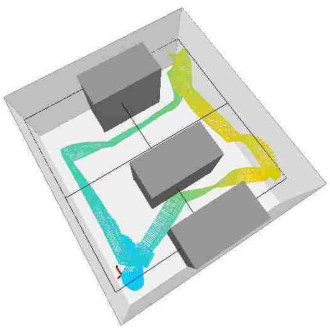
(b) MAPRM Clearance



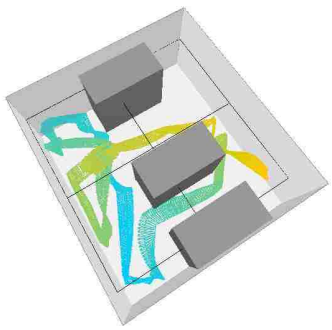
(c) U-PRM Clearance



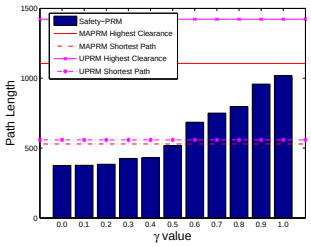
(d) Safety-PRM Paths for varying γ values



(e) MAPRM shortest and highest clearance paths

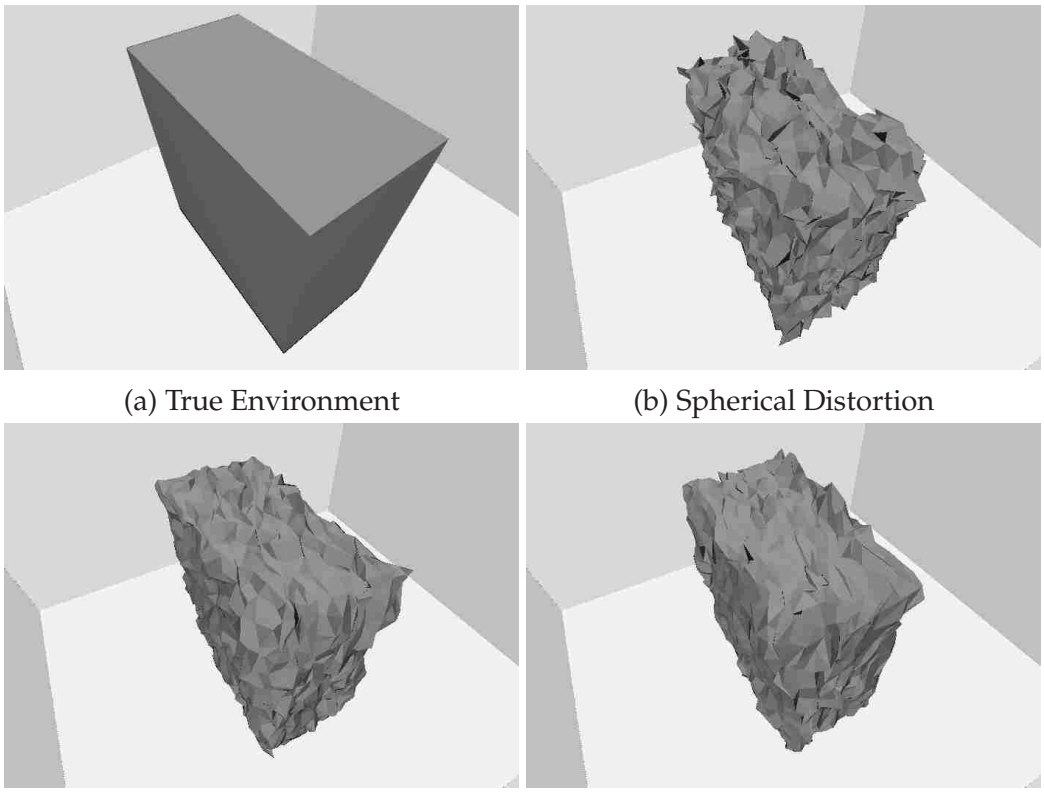


(f) Uniform PRM shortest and highest clearance path



(g) Path Lengths

Figure 3.5: Link Robot in Narrow Environment with 500 vertices. 3.5d 3.5e 3.5f Show the Linkage Paths on the Narrow Environment. The colors of each path indicate the start (blue) and the end (yellow) of the paths. The paths have been normalized to the range of $[0,1]$ so that multiple runs can be compared. 0 is the start of the query while 1 is the goal configuration of the query. The shaded regions indicate the standard deviation over 10 runs for each experiment. Ideally, the algorithm will produce paths which maximize the clearance between obstacles along the paths.



(a) True Environment (b) Spherical Distortion

(c) Gaussian Distortion (d) Log Normal Distortion

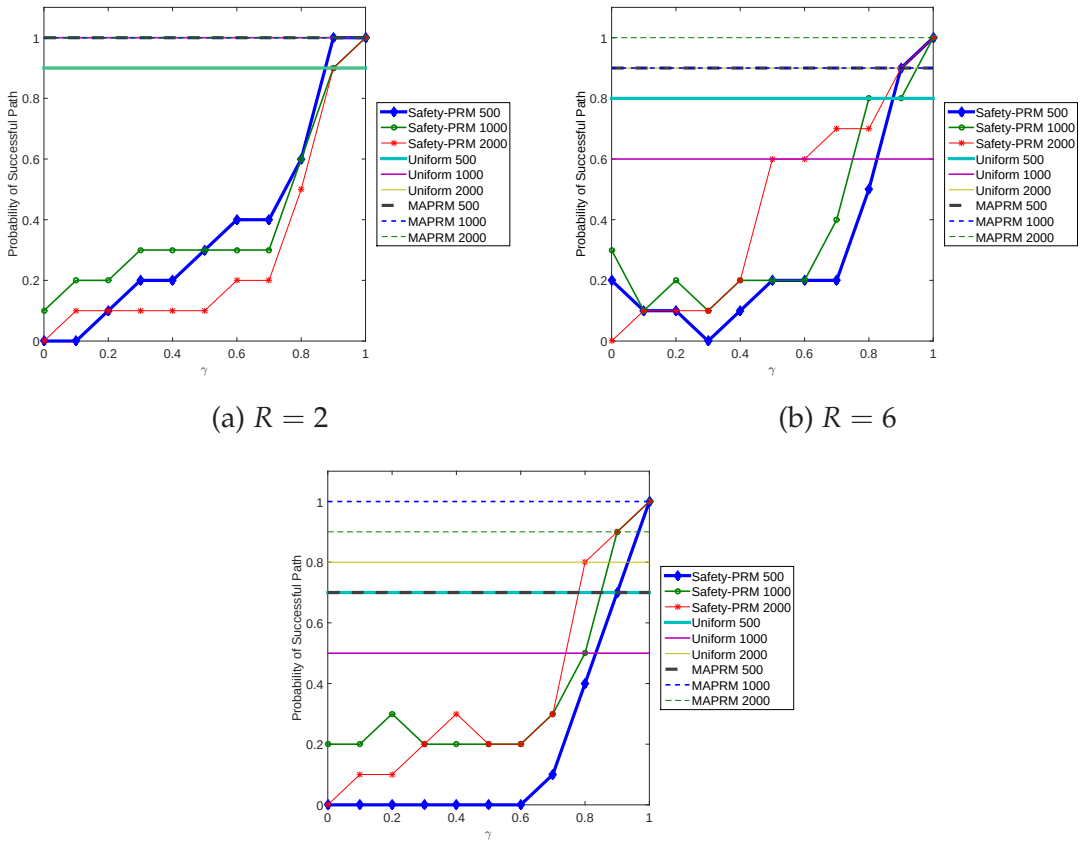
Figure 3.6: Examples of the distortion methods on an example box in an environment. (a) shows the true environment. (b) shows an example of the true environment (a) distorted with spherical error $r = 2$. (c) shows an example of Gaussian distortion of (a) with $\sigma = 2$. (d) shows an example of Log Normal distortion of (a) with $\sigma = 2$.

3.4.3 Spherical Error Model

A Spherical error model is the simplest error model. As discussed in the setup section, the spherical error model distorts each vertex of the model by a uniform random amount bounded by the radius of the sphere.

As the amount of error in the model increases, the problem becomes harder. Figure 3.7 shows the results for Safety-PRM with varying spherical error amounts. These graphs show the percentage of runs which have valid paths in the true environment. Interestingly, Safety-PRM is mostly agnostic to the amount of error as each graph maintains approximately the same shape. This can be seen since all runs have a probability of success greater than 0.9 regardless of the amount of error, for $\gamma > 0.9$ and roadmap size greater than 100. The value of γ is significant because it is the trade off parameter between clearance and path length and the affect of γ is impacted by the amount of error. In contrast, MAPRM (Figure 3.7), while still producing good results, begins to suffer as the error increases. For instance, once the error amount is at $r = 6$ (a maximum expected error for a Kinect sensor), all roadmap sizes less than 2000 have a maximum success rate of 0.9. Similarly for an error of $r = 10$, a 1500 vertex roadmap has a success rate of 0.7. It is important to note that MAPRM is typically intended to run to completion which means the roadmaps produced will be significantly smaller than the maps tested here. This is done because MAPRM is expensive to run on linkages due to the need for ray casting to approximate the medial axis. However, these results indicate that smaller roadmaps will produce poorer success rates for MAPRM, Thus, the quality of results possible with MAPRM are comparable to the results with using Safety-PRM, however the cost of MAPRM is higher (Table 3.2).

Finally, Figure 3.7 shows the results for running the spherical error models with Uniform PRM. As expected the results for Uniform PRM on this task are



(a) $R = 2$

(b) $R = 6$

(c) $R = 10$

Figure 3.7: : Success rates for Safety-PRM, MAPRM and Uniform PRM for spherical error with distortion radius R = 2, 6 and 10.

poor because Uniform PRM does not take clearances into account. However, it is important to note that Safety-PRM has approximately the same running time as Uniform PRM but produces results comparable to MAPRM.

3.4.4 Gaussian Error Model

The Gaussian error model is a standard error model present in many sensors. This error model distorts each vertex in the true model by a random amount drawn

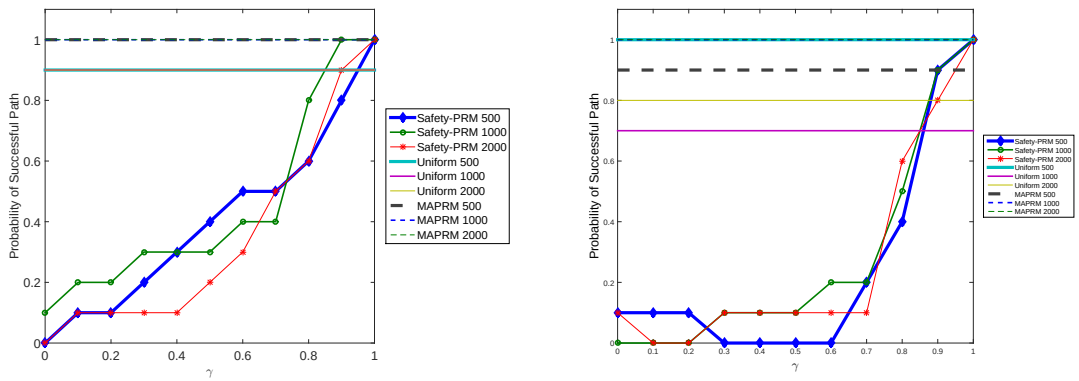
from a Gaussian distribution.

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2 / 2\sigma^2} \tag{3.4}$$

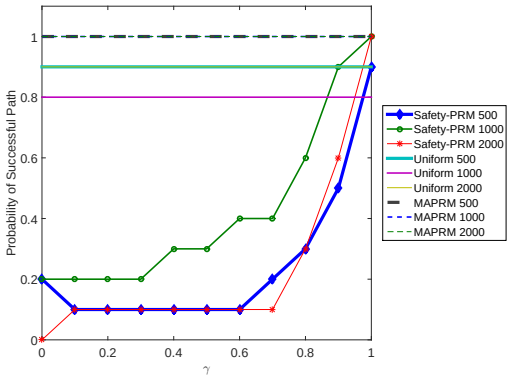
Where μ is the vertex in the model location and σ is the standard deviation.

Similar to the spherical error model the problem becomes harder as the error increases. However the Gaussian error model produces less overall distortion than the Spherical error model. The models distorted by the Gaussian model hold more of their original shape than when distorted by the spherical model. As such it is expected that MAPRM and Uniform will perform better on this task. Figure 3.8 shows just such a trend, with only the smallest MAPRM roadmap falling to 90% success rate at $\sigma = 6$. Similarly, Uniform PRM does not suffer as much in this case either.

Both the comparison methods and Safety-PRM do better under the Gaussian error model because the original obstacle's shape is retained better than the spherical error model. It is important to note that Safety-PRM retains the same shape in this error model as in the spherical error model. This indicates that Safety-PRM is mostly unaffected by the error model type. Safety-PRM is able to compensate for the increasing error in Figure 3.8 but as the error increase the success curve is pushed further down and requires higher γ values to reach high success rates. Again, it is apparent that Safety-PRM produces comparable success rates to MAPRM but at lower cost.



(a) $\sigma = 2$ (b) $\sigma = 6$



(c) $\sigma = 10$

Figure 3.8: Success rates for Safety-PRM, MAPRM and Uniform PRM for Gaussian error with distortion σ = 2, 6 and 10.

3.4.5 Log-Normal Error Model

Here, the *sensed* environment is produced by distorting the vertices with a log-normal error model. This model typically produces nearby distortions similar to the Gaussian model but has a very low probability of producing a very large distortion. The probability density function for the log-normal model is:

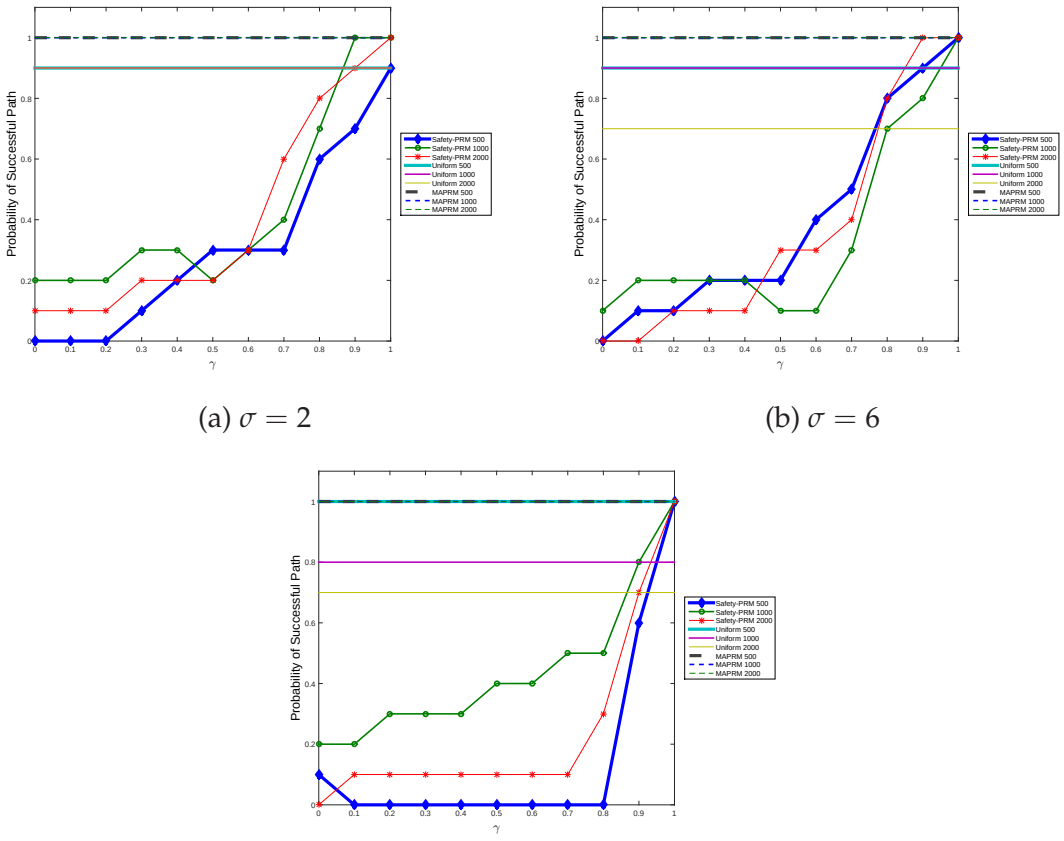
$$f_x(x;\mu,\sigma) = \frac{1}{x\sigma\sqrt{2\pi}} e^{\frac{-(lnx-\mu)^2}{2\sigma^2}}, x > 0 \tag{3.5}$$

Figure 3.9 shows the results of the same experimental setup as the other two models. As expected this error model benefits the MAPRM method as it achieves 100% success for all parameters. Interestingly, Uniform PRM does not benefit from the distorted models much. However, despite this method producing something of a worst case situation for Safety-PRM it remains mostly unaffected by the large protrusions. At the largest σ this error model produces sporadic large protrusions which produces an area of high collision probability in the *sensed* environment. These protrusions practically block the passageway between obstacles. Thus, this error model causes Safety-PRM to produce paths that are closer to the true obstacle surfaces as it attempts to avoid these large spikes.

Despite the challenges of the log-normal error model for Safety-PRM it still retains a similar shaped curve to the two error model experiments. Thus, Safety-PRM is able to successfully produce paths in a variety of error model types and amounts. At the highest clearance/path length trade-off value of γ Safety-PRM always produces comparable paths and success rates to MAPRM, but at cost comparable to Uniform PRM, i.e. less expensive than MAPRM.

3.5 Kinect Reconstructed Environment with Physical Robot Validation

In this final set of experiments, motion planning is done on a physical robot, the Barrett Whole Arm Manipulator (WAM), in an environment modelled with sensors. First, a Kinect sensor is used to "construct" a model of the physical envi-



(a) $\sigma = 2$

(b) $\sigma = 6$

(c) $\sigma = 10$

Figure 3.9: : Success rates for Safety-PRM, MAPRM and Uniform PRM for Log-normal error with distortion σ = 2, 6 and 10.

ronment. Then, a roadmap is built in this environmental model and a path is extracted. This path is finally validated in the physical environment with the WAM robot.

3.5.1 Whole Arm Manipulator

The Barrett Whole Arm Manipulator (WAM) [45] platform used in the real robot experiments is a seven degree of freedom (DoF) robotic arm as seen in Figure 3.1.

		Safety-PRM			MAPRM			Uniform Random PRM		
Robot	Env	Vertices	Edges	CD's	Vertices	Edges	CD's	Vertices	Edges	CD's
Big Arrow	$R = 2$	100	800.0	58,141.8	100	647.0	82,874.0	100	562.2	75,442.7
		500	4,070.6	242,629.0	500	3,170.2	315,425.1	500	3,000.0	332,446.8
		1000	8,231.2	453,367.7	1000	6,302.8	560,822.6	1000	6,110.2	637,774.6
		1500	12,368.0	645,746.8	1500	9,466.4	787,097.9	1500	9,267.8	940,927.5
		2000	16,543.2	830,164.1	2000	12,590.2	996,711.1	2000	12,431.0	1,226,638.0
Big Arrow	$R = 6$	100	809.6	59,632.5	100	645.6	82,112.8	100	534.6	68,345.5
		500	4,101.2	248,299.2	500	3,176.0	315,411.8	500	2,886.8	311,483.8
		1000	8,296.2	464,736.7	1000	6,301.0	560,560.3	1000	5,901.4	602,357.4
		1500	12,473.2	663,507.5	1500	9,445.0	785,060.5	1500	8,870.6	888,447.6
		2000	16,680.8	854,503.5	2000	12,570.6	994,518.2	2000	11,923.8	1,161,099.5
Big Arrow	$R = 10$	100	817.8	61,042.5	100	648.4	83,172.6	100	476.6	61,080.7
		500	4,167.2	258,892.2	500	3,167.6	314,143.9	500	2,624.4	280,824.0
		1000	8,415.2	485,088.4	1000	6,302.2	558,991.8	1000	5,406.8	536,050.4
		1500	12,669.2	696,219.5	1500	9,435.8	782,933.5	1500	8,212.0	782,880.3
		2000	16,928.8	899,171.5	2000	12,587.2	995,477.6	2000	11,071.2	1,023,579.5

Table 3.2: Edges and Collision Detection (CD) calls for the simulation experiments on the spherical environment with varying error amounts.

It is a cable-driven system controlled with position encoders and torque estimation sensors. For the experiments in this chapter, the WAM has been connected to a GE Intelligent Platforms reflective memory network in a spoke design that allows multiple computers to share memory at speeds ranging from 43 MB/s to 170 MB/s. The reflective memory networks allows remote computers to handle the planning processing, while leaving a small and fast computer on-board the WAM to handle simple motion control.

The WAM is connected to an xPC Target Kernel running Matlab Simulink 7.7.0 R2008b [72]. The controller for the WAM is written in Simulink and interfaces with remote computers via the reflective memory network. The Simulink code responsible for directly issuing commands to the WAM, hence the WAM controller, receives a command vector by reading a specific block of reflective memory. The command vector is a length-seven vector containing the desired joint angles in radians of each for the seven WAM joints.

The WAM controller, upon receiving a command vector, places the command vector into a buffer, which only stores one move until completion of the move. The command vector is first sanitized so that each entry is within the WAM's joint limits. If the WAM is not executing a move, it compares its current loca-

tion to the command vector buffer. If the command vector buffer is sufficiently different from the current location, the WAM controller computes a straight line interpolation between the two points and executes the path within the allowable WAM workspace. In the current architecture a move cannot be interrupted except by a collision. The velocity of the straight line interpolation follows a fifth-order smooth polynomial as seen in Fig 3.10, is used both for safety and for mimicking biological motion [28]. These slow beginnings and endings to the motion events provides safer joint torques than a simple uniform motion event.

For additional safety, the clearance probability is also used to determine the speed of the move. The speed of a move is proportional to $P(Edge)$, the probability that an edge is in collision. Any function can be used to determine the speed. This modulation means that the robot moves slower when it expects that the arm is near an obstacle, thus if an impact occurs it will be at a slower velocity.

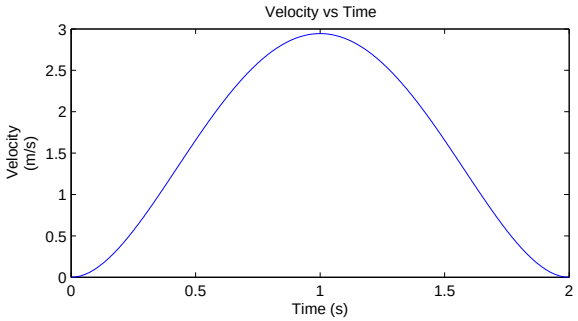


Figure 3.10: Velocity Profile of the WAM Controller

Finally, torque estimation is used to determine if the arm has collided with an obstacle. Torque is approximated by the on board position encoders in each WAM joint and is computed via a proprietary algorithm. This produces noisy torque estimates which slightly lag behind the actual torque experienced by the arm. Furthermore, a simple threshold is used to determine if the torque is too high for the arm. It is important to note that the torque estimation is sensitive to

the speed of the WAM. If the arm is moving too fast the torque estimation will not be able to detect a collision and stop the arm before damaging the arm. Thus the velocity modulation is critical to keeping the arm's velocity low near potential obstacle collisions. To further reduce the possibility of damaging the arm during collisions, the joint stiffness is reduced by a uniform amount across all joints by the onboard controller. This allows the arm to flex during collisions.

3.5.2 Environment

The pipline for the physical experiments is to, 1) build a 3D model of the environment from sensed data, 2) Build a roadmap using the Parasol Motion Planning Library (with Safety-PRM, MAPRM and Uniform PRM), 3) Extract a path using Dijkstra Algorithm. In the case for Safety-PRM the weights are assigned via equation 3.3, 4) Using the extracted path instruct the WAM arm to navigate between the path vertices, 5) In the case of collision, stop the arm to prevent damage. Here, the path plans generated by the planning algorithms are evaluated. If a plan leads to collision, the method is considered to have failed on the test.

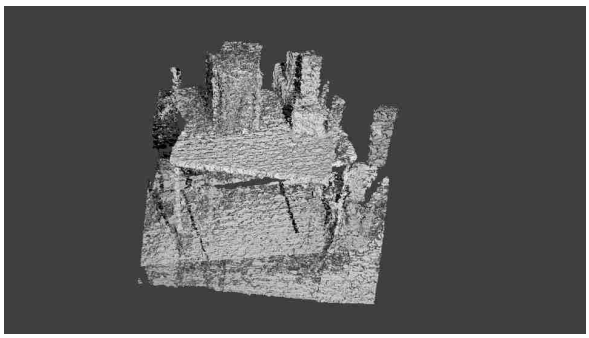
For this set of experiments a Microsoft Kinect sensor was used to collect 3D point cloud data of an environment to reconstruct a 3D model of the workspace (Figure 3.11). Eight point clouds were collected at even intervals over a 120 degree rotation around the workspace. These clouds were all collected from the same height with the Kinect pointed towards the center of the workspace at each sample point. This produced a series of point clouds which represented the workspace. These clouds where then registered using the Point Cloud Library 1.6.0 (PCL) [31] to produce a noisy point cloud representation. This combined point cloud was then triangulated, again using PCL, to produce a mesh representation. Finally, the PCL segmentation and convex hull method were applied to the triangle mesh

to produce the final model used for path planning [31].

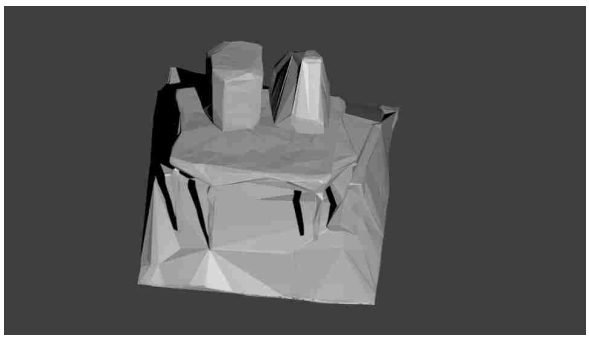
There are two environments for this set of experiments. The first environment is used for the WAM validation experiments and the second environment is used for the dynamic replanning experiments. The first environment is a table set up in front of the WAM with two boxes separated by a gap approximately 2 times the diameter of the WAM hand (10 inches apart). The task is for WAM to reach between the boxes and curl around one of the obstacles without collision. Figure 3.11 shows an image of the actual table and the steps used to produce the final 3D model used for path planning. Figure 3.11a shows a photograph of the environment, 3.11b shows the raw point clouds after registration and 3.11c shows the 3D model used for path planning after segmentation and convex hull application. These figures demonstrate the errors introduced by the Kinect sensor in the final model. Note that the error is different from the three simulation error models and has been shown to increase quadratically with distance [78].

The second environment is a single box setup on a table that the WAM must reach over. This task is designed to demonstrate dynamic replanning. Figure 3.14 shows the environment. This box is placed such that the WAM must reach up and over. However, for low values of γ the WAM will reach into the box and produce a collision, triggering a replanning event. Section 3.5.4 discusses replanning in more detail.

(a) Photograph of the physical environment.



(b) Point cloud collected by the Kinect sensor data after registration and triangulation.



(c) Final 3D model of the environment after segmentation and convex hull applications.

Figure 3.11: WAM Physical Environments

3.5.3 WAM Validation

Figure 3.12 shows the results of running path planning with Safety-PRM, MAPRM, and Uniform PRM on the sensed data and the WAM robot for the task show in Figure 3.13. As the previous experiments showed that $\gamma < 0.70$ produces poor results in most runs, these experiments are limited to Safety-PRM runs of $\gamma \in [0.70, 0.80, 0.90, 1.0]$. Similarly, to provide the robot with the most safety, the comparison methods were run until a solution was found instead of limiting the vertices to match those of Safety-PRM. Each experiment was run 5 times with different roadmaps and averaged together. Figure 3.13 shows an example run for the task completed by Safety-PRM. The goal is to reach around the colored blocks without collision under no time constraints.

Figure 3.12 shows that all methods were able to complete the task without collision. However, for MAPRM and Uniform PRM the sensed data set had to be manually cleaned up as the error in the model placed the edge of the table inside the starting configuration. The comparison methods completed the task because the error induced by the Kinect favors expanding the obstacles more than reducing them. Thus, the Kinect's error model acts like a buffer. However, Table 3.3 shows that Safety-PRM was able to solve the task with fewer CD calls than Uniform PRM and MAPRM. Thus, Safety-PRM performs the physical task as well as the comparison methods but at a lower cost. This, combined with the previous experimental results, shows that Safety-PRM successfully encodes the collision probabilities into the roadmap and is able to generate paths with a similar quality to MAPRM but with fewer CD calls.

		Safety-PRM			Uniform Random PRM			MAPRM		
Robot	Env	γ	Edges	CD's	γ	Edges	CD's	γ	Edges	CD's
WAM	Table	0.7 0.8 0.9 1.0	54,582.6 77,919.3 77,749.3 77,750.3	295,842.6 293,095.3 288,513.0 288,513.1	N/A	110,207.3	359,457.6	N/A	30,544.6	34,612,483.0

Table 3.3: Edges and Collision Detection (CD) calls for the WAM experiments.

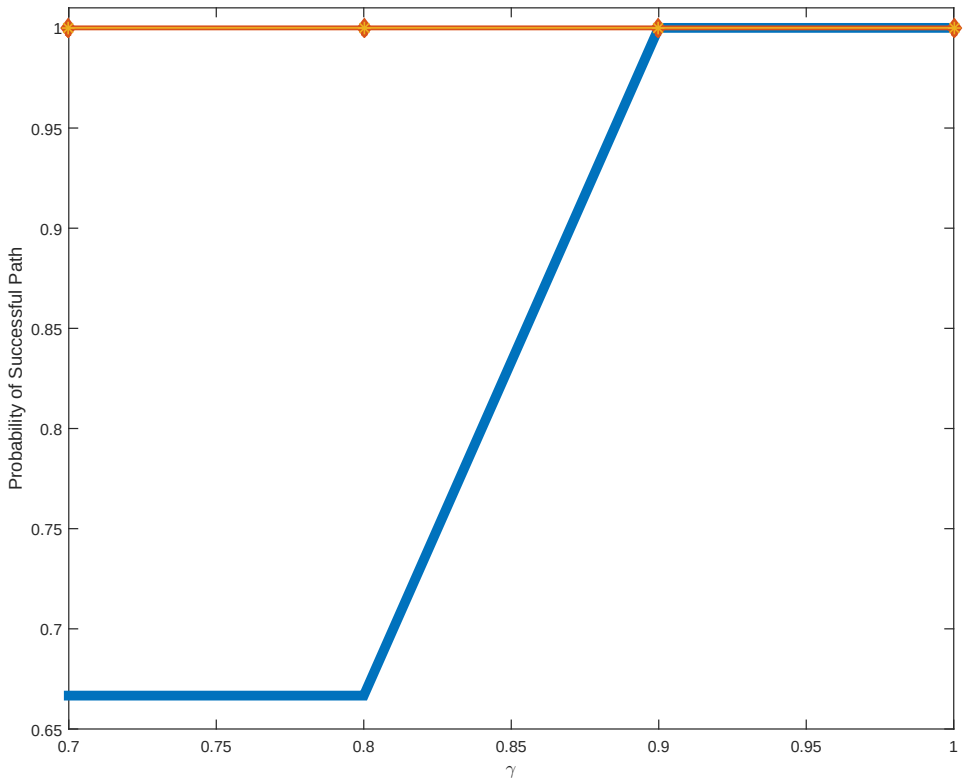


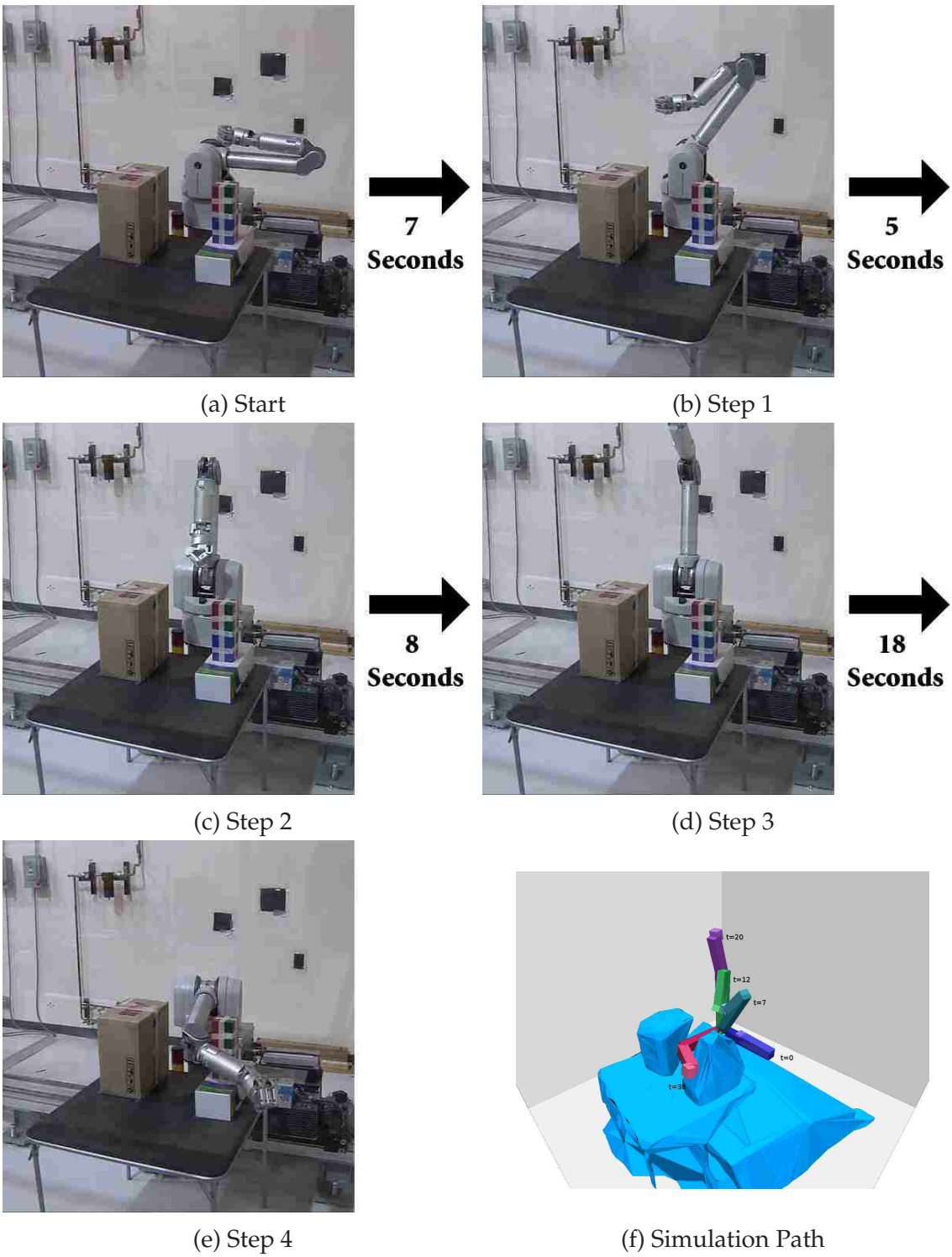
Figure 3.12: MAPRM (red line), Uniform PRM (yellow line), and Safety-PRM (blue line) success rates for the physical experiment.

3.5.4 Dynamic Replanning

Another advantage of Safety-PRM is that it builds a roadmap to a specific size and then plans a path based on the safety trade-off parameter. This is particularly useful for real robot applications with noisy sensors. Often, path length is an important consideration and shorter paths are preferred as such γ will be set less than the maximum. Even with γ set to 1, it is still possible for collision to occur as sensor error could produce poor or erroneous models of the environment. Thus, dynamic path replanning is a very useful capability for a path planning algorithm.

Since Safety-PRM builds specific sized roadmaps, multiple paths to the goal will likely exist in the roadmap and dynamic path replanning is possible. To demonstrate path replanning, experiments are performed on an environment consisting of a table with a box that the WAM must reach over without collision. Figure 3.14 show an example of the environment and the task.

Dynamic replanning is based on detecting collisions. Torque estimation is used to detect collisions and to refine the roadmap. The simple implementation utilizes a torque threshold on the WAM motors to determine if a collision has occurred. If the arm collides with an obstacle, it sends a stop signal. Then the edge that is being traversed is determined to be in collision. This information is then used to remove the in-collision edge from the roadmap. The WAM is backed up to the last known safe vertex in the roadmap and the path is replanned in the pruned roadmap. This allows for intelligent refinement of the roadmap given the expected clearance of the Safety-PRM method. Figure 3.14 shows a sequence of this process. In this experiment, the γ value is set to 0.5, so that the planner will choose a short path which collides with the obstacle. Figure 3.14c is when the WAM collides with the obstacle and Figure 3.14d shows the WAM backing up. The remaining figures show the replanned route to the goal.

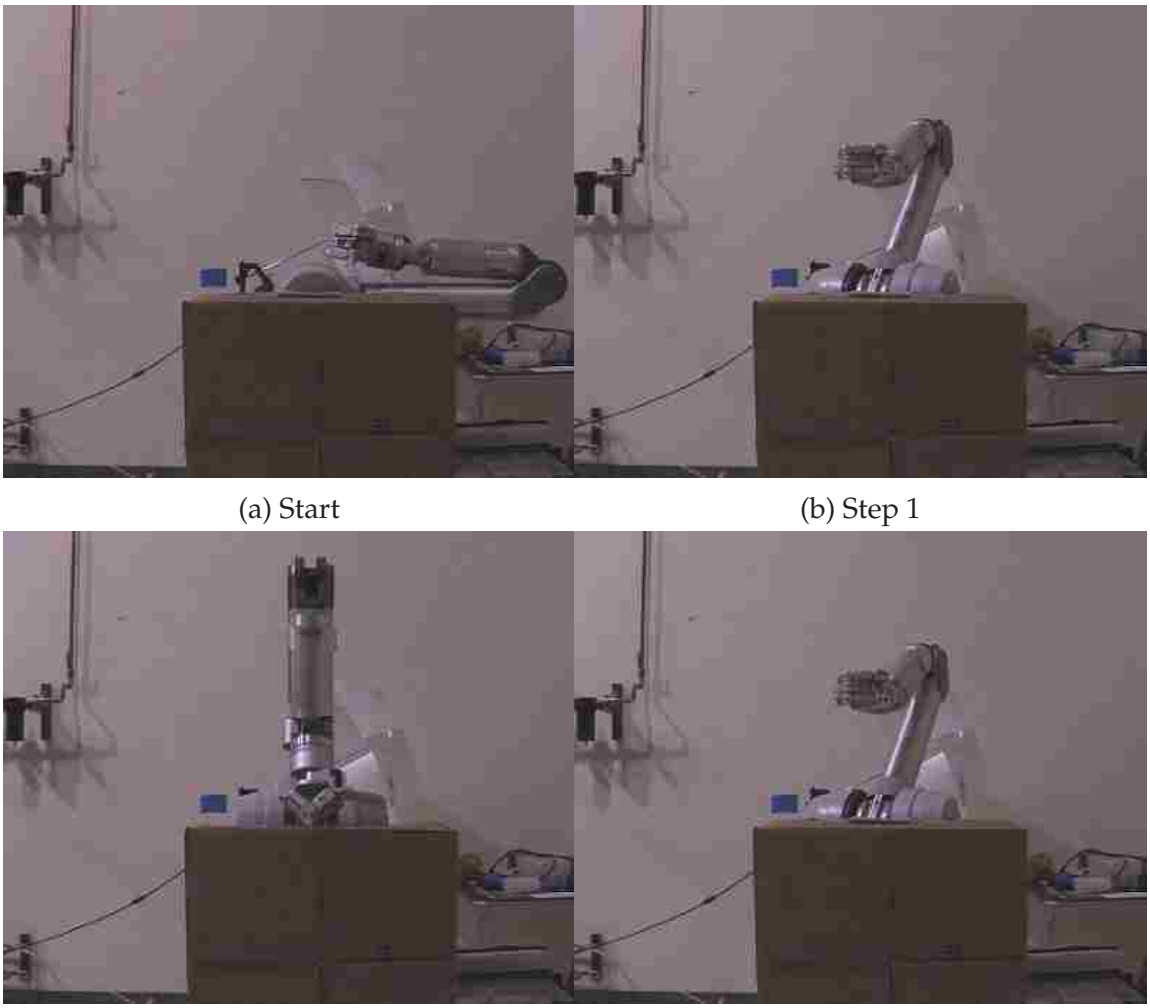


(a) Start (b) Step 1

(c) Step 2 (d) Step 3

(e) Step 4 (f) Simulation Path

Figure 3.13: Sequence of the WAM Path, The move time is determined by $2 + (1 + P(Edge))^2$ where $P(Edge)$ is the probability of an edge being in collision.



(a) Start (b) Step 1

(c) Step 2 (d) Step 3

Figure 3.14: Sequence of WAM Path using torque sensing, 3.14c is where the WAM collides with the obstacle and replans the path.

3.6 Conclusions

This chapter has shown that the Safety-PRM roadmap offers several advantages over basic PRM roadmaps for real robots. Using Safety-PRM allowed for cheap, tunable roadmaps to be produced for complex robots and environments. First, Safety-PRM was shown computationally cheaper than MAPRM. Second, it allowed for inaccurate workspace models without the need for the traditional method of enlarging obstacles. However, it provided high expected clearance paths. These advantages of Safety-PRM allow it to be used on real robotic hardware as demonstrated by the WAM applications.

This chapter demonstrated Safety-PRM on several environment with several robots. These environment and robots were chosen in such a way as to demonstrate the generalization of the Safety-PRM method. Experiments were done on a simple environment, a medium complexity environment, and finally on a complex and Cluttered environment. The results obtained for all of these environment were similar. This indicates that the algorithm is mostly agnostic to the environment types and robot types. It is instead mostly impacted by the type and amount of uncertainty in the roadmap. Furthermore, the Safety-PRM method was shown with a uniform random sampler as the underlying sampling function. However, that is not required. Any of the multitude of sampling based methods could be used in conjunction with the Safety-PRM scheme thus making the method more general.

Chapter 4

Spatio-Temporal Uncertainty: Moving Obstacle Avoidance

Dynamic and changing environments due to moving obstacles is another major source of uncertainty in motion planning. Moving obstacles can invalidate a current plan or force the robot to avoid an imminent collision. This chapter proposes solutions for the moving obstacle problem from [64] and [61].

Consider, self-driving vehicles. Pedestrians, other drivers, and unexpected animal crossing all create situations that the path planning algorithm used by the vehicle must consider. Due to moving obstacles, the current plan must be revisited and possibly adjusted.Unfortunately, simply recalculating a full plan every time a path is invalidated is ineffective in many situations. Thus, the planning algorithm must make more informed decisions that consider possible obstacle dynamics.

In this chapter, two methods are presented to account for moving obstacles.

The first method is SR-Query, which combines roadmap methods with formal control theory to weight roadmap edges in an informed manner. The second method is APF-SR, which uses formal control theory as a heuristic to fit a potential field around obstacles that matches their expected motion. APF-SR was developed to provide a motion planning algorithm which is more reactive than SR-Query. To demonstrate this reactivity, APF-SR is shown navigating environments with up to 900 moving obstacles which can switch between line and arc trajectories. These two methods consider the obstacle motions and are thus able to make informed decisions. It is important to note that the methods do not know the obstacle's future trajectories. Rather, they only have a model of the obstacle motion which provides probabilities of collisions in the short term.

4.1 SR-Query

One of the many challenges in designing autonomy for operation in uncertain and dynamic environments is the planning of collision-free paths. In applications such as search and rescue, coordinated sensing, collaborative monitoring, or automated manufacturing environments, a robot must traverse from a known start state to a goal state, in an environment that could contain many moving obstacles with stochastic dynamics. While theoretical solutions may be available via stochastic reachability, the high computational expense limits such an approach to a very small number of dynamic obstacles, depending on the model complexity of the robot and obstacle dynamics. Motion planning techniques provide a more computationally feasible alternative, depending on degrees of freedom of the robot, the nature of the environment, and the planning constraints. However, there is strong evidence that any complete planner will require exponential time in the number of DOFs of the robot [51], [42], [20].

This chapter presents a novel, stochastic reachability based method to create probabilistic roadmaps that accommodate many moving obstacles that travel stochastically along straight line or arc trajectories. The likelihood of collision with a given object (computed *a priori* via stochastic reachability (SR)) is used to inform the likelihood of collision along a given path. This method is demonstrated computationally on scenarios with up to 50 obstacles with stochastic velocities.

Stochastic reachability analysis provides offline verification of dynamical systems, to assess whether the state of the system will, with a certain likelihood, remain within a desired subset of the state-space for some finite time, or avoid an undesired subset of the state-space [1]. To solve problems in collision avoidance, the region in the relative state-space which constitutes collision is defined as the set of states for the system to avoid [98, 49]. Unfortunately, the computation time for stochastic reachable sets (SR sets) is exponential in the dimension of the continuous state, making the assessment of collision probabilities with many simultaneously moving obstacles next to impossible (once the dynamics of each obstacle are incorporated into the state). However, while expensive, SR sets can be computed offline and the result queried online.

This method combines multiple SR sets (computed pairwise between the robot and each dynamic obstacle Equation (4.7)), to generate appropriate weights associated with the edges in the roadmap. The SR sets are generated offline, computed individually for relative dynamics associated with each obstacle, and the results combined and queried at runtime by the method. In an environment with multiple obstacles, the intersection of multiple SR sets clearly cannot provide a strict assurance of safety, since the reachable set is computed for one dynamic obstacle in isolation. However, such an approach can significantly improve the ability of the roadmap to reflect obstacle dynamics. Further, in simulation, it is found that the SR - weighted roadmap is able to intelligently navigate in the presence

of stochastic dynamic obstacles significantly more often than standard roadmap methods.

The proposed method has several advantages over existing moving obstacle solutions. First, the method is fast since it does not have to make expensive collision detection calls and instead just queries the precomputed SR set. Second, it scales well with many obstacles. Furthermore, it provides a framework in which multiple SR sets can be combined to generate approximate collision avoidance probabilities with many moving obstacles, which would otherwise be impossible using a single SR set that accounts for all obstacles simultaneously. Finally, by using SR for the underlying collision probability calculation, the method provides an upper bound on the probability of collision avoidance, which can be used comparatively to select the best path.

Section 4.1.1 presents the robot and obstacle dynamics, and known techniques for roadmap construction. Section 4.1.2 presents the computed stochastic reachable sets for collision avoidance with two types of stochastic dynamic obstacles, as well as the algorithm for roadmap construction that queries the stochastic reachable set. Section 4.1.4 describes the computational experiments, with two moving obstacles, and finally with 50 moving obstacles. Lastly, conclusions are offered in Section 4.1.5.

4.1.1 Preliminaries

Obstacle Dynamics

Two representative types of dynamic obstacles, which have known trajectories but stochastic velocities, are considered. In particular, two-dimensional point mass obstacles with straight-line and constant-arc trajectories are used. The obstacle

dynamics are of the form $\dot{\overline{x}}^o = f(w,t)$, with obstacle state $\overline{x}^o = (x^o, y^o)$, and with the velocity, w a discrete random variable that takes on values in $\mathcal{W}$ with probability distribution $p(w)$. A discrete random variable is considered here for computational simplicity, although a continuous random variable could be introduced. The discretized obstacle dynamics (via an Euler approximation with time step Δ) are

$$\begin{aligned} x^o_{n+1} &= x^o_n + \Delta w_{n+1} \\ y^o_{n+1} &= \alpha x^o_{n+1} \end{aligned} \tag{4.1}$$

for straight-line movement, with speed $w \in \mathcal{W}$ and slope $\alpha \in \mathbb{R}$, and

$$\begin{aligned} x^o_{n+1} &= x^o_n + \Delta r \left(\cos(w_{n+1}(n+1)) - \cos(w_{n+1}n)\right) \\ y^o_{n+1} &= y^o_n + \Delta r \left(\sin(w_{n+1}(n+1)) - \sin(w_{n+1}n)\right) \end{aligned} \tag{4.2}$$

for constant-arc movement, with angular speed $w \in \mathcal{W}$.

Relative robot-obstacle dynamics

A two-dimensional point-mass model for the robot is presumed. The robot has state $\overline{x}^r = (x^r, y^r)$ and dynamics in Cartesian coordinates

$$\begin{aligned} \dot{x}^r &= u_x \\ \dot{y}^r &= u_y \end{aligned} \tag{4.3}$$

with two-dimensional control input $u = (u_x, u_y)$ that is the velocity of the robot in both directions. While the obstacle is not trying to actively collide with the robot, its dynamics (4.1), (4.2) contain a stochastic component, which can be considered a disturbance that affects the robot's behavior. Discretizing the dynamics (4.3) using an Euler approximation with time step Δ results in

$$\overline{x}^r_{n+1} = \overline{x}^r_n + \Delta \cdot u. \tag{4.4}$$

A collision between the robot and the obstacle occurs when $|\overline{x}_n^r - \overline{x}_n^o| \leq \epsilon$ for some n and ϵ small. A relative coordinate space is constructed such that it is fixed to the obstacle, with the relative state defined as $\tilde{x} = \overline{x}^r - \overline{x}^o$. Hence the dynamics of the robot *relative* to the obstacle are

$$\tilde{x}_{n+1} = \tilde{x}_n + \Delta u_n - \Delta f(w_n, t_n) \tag{4.5}$$

with $f(\cdot)$ as in (4.1) and (4.2), and a *collision* is defined as

$$|\tilde{x}_n| \leq \epsilon. \tag{4.6}$$

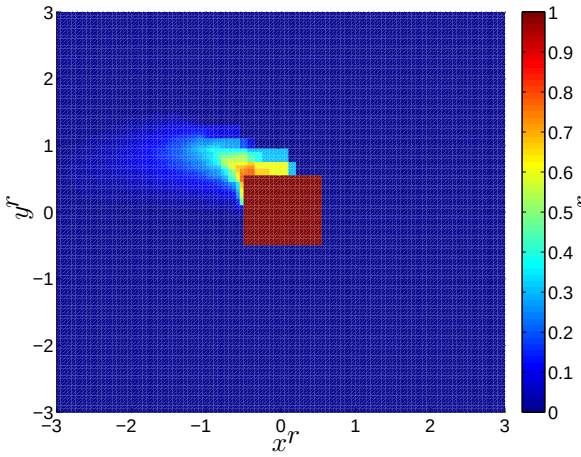
Using Equation (4.5), a dynamical system is defined with state $\tilde{x} \in \mathcal{X}$, control input $u \in \mathcal{U}$ that is bounded, and stochastic disturbance w. Because $\tilde{x}_{n+1}$ is a function of a random variable, it is also a random variable. Its transitions are governed by a stochastic transition kernel, $\tau(\tilde{x}_{n+1} \mid \tilde{x}_n, u_n, n)$, that represents the probability distribution of $\tilde{x}_{n+1}$ conditioned on the known values $\tilde{x}_n$, u_n and time step n.

Roadmap Construction

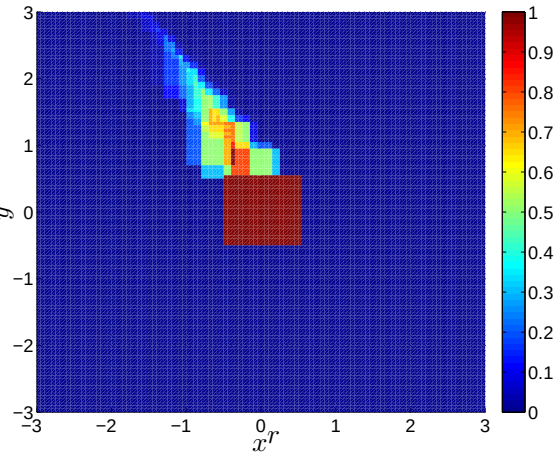
The proposed method combines SR sets with roadmap base techniques. Here Uniform PRM [51], and cell decompotion [59] are used to construct the underlying roadmap. Chapter 2 discusses these methods in more detail.

4.1.2 Methods

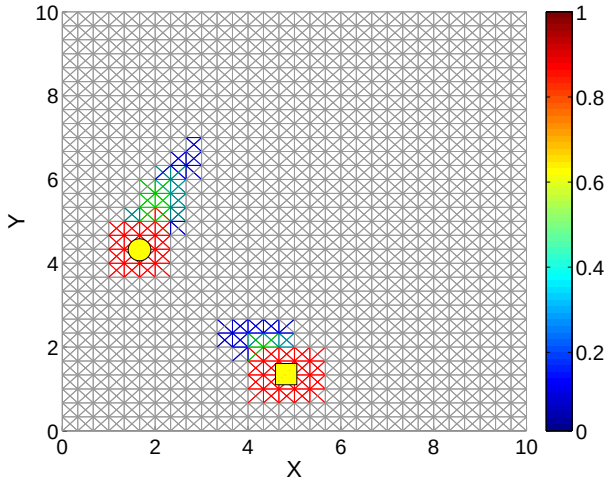
In this section, the novel methods for integrating SR sets with roadmap path extraction are presented. First, the SR problem for collision avoidance with the straight-line and constant-arc dynamic obstacles is formulated. Then, the method for using SR to help build roadmaps that select a path that avoids multiple moving obstacles is shown.



(a) SR set for arc obstacle.



(b) SR set for line obstacle.



(c) Roadmap with edges weighted by SR sets

Figure 4.1: This figure shows how the SR sets for constant-arc (a) and straight-line (b) obstacles are incorporation into a roadmap (c). (a) Stochastic reachable set that shows the likelihood of collision between the robot and an arc obstacle (in relative coordinates). (b) Stochastic reachable set for a straight-line obstacle with $\alpha = 1$. (c) A grid based roadmap (in Cartesian coordinates) with likelihood of collision with an obstacle indicated by edge color. More redish color indicates a higher probability of collision The yellow circle (square) shows the line (arc) obstacle locations.

4.1.3 SR for Collision Avoidance

The SR problem can be formulated in the context of collision avoidance, where the probability of avoiding collisions within some finite time horizon is determined. The set $\overline{K}$ is defined as the set of states in which a collision is said to occur (4.6). To avoid collision with the obstacle, the robot should remain within K, the complement of $\overline{K}$. The probability that the robot will remain within K over N time steps, with initial relative position $\tilde{x}_0$, is given by

$$A_{\tilde{x}_0}^{\overline{u},N}(K) = \mathbb{P}\left[\tilde{x}_0, \ldots, \tilde{x}_N \in K \mid \tilde{x}_0, \overline{u}\right] \tag{4.7}$$

with $\mathbb{P}$ denoting probability and input sequence
$\overline{u} = [u_0, u_1, \cdots, u_{N-1}]^T$.

Since $\mathbb{P}[x \in K] = \mathbb{E}[\mathbf{1}_K(x)]$, with $\mathbb{E}$ denoting expected value and $\mathbf{1}_K(x)$ denoting the indicator function defined as $\mathbf{1}_K(x) = 1$ for $x \in K$, and 0 otherwise, Equation (4.7) can be rewritten as (see [1])

$$A_{\tilde{x}_0}^{\overline{u},N}(K) = \mathbb{E}\left[\prod_{n=0}^{N} \mathbf{1}_K(\tilde{x}_n) \mid \tilde{x}_0, \overline{u}\right], \tag{4.8}$$

since $\prod_{n=0}^{N} \mathbf{1}_K(\tilde{x}_n) = 1$ if $\tilde{x}_0, \ldots, \tilde{x}_N \in K$, and 0 otherwise.

Finally, instead of assuming a predetermined set of control inputs $\overline{u}$, a state-feedback control input is constructed to maximize the likelihood of avoiding collision and to facilitate real-time control selection for motion planning. Equation (4.8) can then be reformulated as a stochastic optimal control problem.

$$A_{\tilde{x}_0}^{N}(K) = \max_{\pi \in \Pi} \mathbb{E}\left[\prod_{n=0}^{N} \mathbf{1}_K(\tilde{x}_n) \mid \tilde{x}_0\right] \tag{4.9}$$

Hence, we define a policy $\pi = (\pi_0, \ldots, \pi_{N-1})$ with $\pi_n : \mathcal{X} \to \mathcal{U}$ and optimize Equation (4.9) over all possible policies Π of this form. The resulting optimal policy π^* provides an upper bound on the probability of avoiding collision.

A dynamic programming recursion [10] is implemented, first introduced for the reachability problem in [1], to estimate the collision avoidance probability.

$$V_N(\tilde{x}) = \mathbf{1}_K(\tilde{x}) \tag{4.10}$$

$$V_n(\tilde{x}) = \mathbf{1}_K(\tilde{x}) \int_{\mathcal{X}} V_{n+1}(\tilde{x}')\tau(\tilde{x}' \mid \tilde{x}, u, n)\, d\tilde{x}' \tag{4.11}$$

Iterating Equations (4.10), and (4.11) backwards, the value function at time 0 provides the probability of avoiding collision,

$$V_0(\tilde{x}_0) = A^N_{\tilde{x}_0}(K). \tag{4.12}$$

The optimal control is determined by evaluating

$$V^*_n(\tilde{x}) = \sup_{u\in\mathcal{U}} \left\{ \mathbf{1}_K(\tilde{x}) \int_{\mathcal{X}} V^*_{n+1}(\tilde{x}')\tau(\tilde{x}' \mid \tilde{x}, u, n)\, d\tilde{x}' \right\} \tag{4.13}$$

which also returns the optimal policy π^*, with

$$\pi^*_n(\tilde{x}) = u_n = \arg\sup_{u\in\mathcal{U}} V^*_n(\tilde{x}). \tag{4.14}$$

Equation (4.13) can be simplified to

$$V^*_n(\tilde{x}) = \max_{u\in\mathcal{U}} \left\{ \mathbf{1}_K(\tilde{x}) \sum_{w\in\mathcal{W}} V^*_{n+1}\left(\tilde{x} + \Delta u - \Delta f(w,n)\right) p(w) \right\}. \tag{4.15}$$

Figure 4.1a shows the SR set for a constant-arc obstacle with radius $r = 5$, and probabilities $p(w) = \{0.2, 0.2, 0.3, 0.3\}$ associated with angular speeds $w \in \mathcal{W} = \left\{\frac{.4}{2\pi}, \frac{.6}{2\pi}, \frac{.9}{2\pi}, \frac{1.2}{2\pi}\right\}$. The slight curvature seen in the probability peaks corresponds to the obstacle trajectory. Similarly, Figure 4.1b shows the SR set for a straight-line obstacle with probabilities $p(w) = \{0.3, 0.4, 0.3\}$ associated with speeds $w \in \mathcal{W} = \{0.5, 0.7, 0.9\}$, and slope $\alpha = -1$. The peaks show higher probability of collision

with the obstacle when the robot is in line with the obstacle trajectory. Intuitively, the closer the robot is to the obstacle, the higher the probability of collision.

On a single core of an Intel 3.40 GHz CORE i7-2600 CPU with 8 GB of RAM, Figure 4.1a took 1727.25 seconds to compute, over a horizon of $N = 30$ steps and a time step of length $\Delta = 1$. Figure 4.1b took 1751.87 seconds to compute, again with $N = 30$ and $\Delta = 1$. In both cases, convergence is observed in the stochastic reachable sets for $N > 5$ since the robot and obstacle traveled sufficiently far apart within this time frame.

With a single obstacle, $V_0^*(\tilde{x}_0)$ in Equation (4.15) is the maximum probability of avoiding a collision, and hence a tight upper bound. For two obstacles with separately calculated avoidance probabilities $V_0^{*,1}(\tilde{x}_0^1)$, $V_0^{*,2}(\tilde{x}_0^2)$ (with relative position $\tilde{x}_0^i$ with respect to obstacle i), the probability of avoiding collision with *both* obstacles is

$$\begin{aligned}
\mathbb{P}[B_1 \cap B_2] &= \mathbb{P}[B_1] + \mathbb{P}[B_2] - \mathbb{P}[B_1 \cup B_2] \\
&\leq \min\{\mathbb{P}[B_1], \mathbb{P}[B_2]\} \\
\mathbb{P}[B_1 \cap B_2] &\leq \min\{V_0^{*,1}(\tilde{x}_0^1), V_0^{*,2}(\tilde{x}_0^2)\} \qquad (4.16)
\end{aligned}$$

where B_i corresponds to the event that the robot avoids collision with obstacle i. An upper bound on the collision avoidance probability is obtained for two obstacles by taking the minimum of the individual avoidance probabilities. The same holds similarly for m obstacles. The minimum of the m individual avoidance probabilities provides an upper bound on the probability of avoiding collision with all m obstacles.

Lastly, note that because the collision avoidance probabilities are used to determine routing choices on a roadmap, the true probabilities are of less interest than the relative probabilities. By generating an upper bound on the probability of avoiding collision with several moving obstacles, the robot can identify

and travel along the path with the greatest upper bound. Furthermore, in the case where avoiding the nearest obstacle is not interfered with by other obstacles, then this upper bound is tight. Thus, the robot can accurately identify the safest route through the roadmap. However, if the obstacle density is too high, then this bound is not tight. This is due to only calculating the SR set for the robot and a single obstacle, but planning in an environment with multiple obstacles.

SR Query

SR sets (4.12) for straight-line and constant-arc obstacles are now integrated into a pre-computed roadmap using techniques developed for static obstacles [59, 51]. Given a roadmap and an SR set for each moving obstacle, paths that are likely to be free are identified.

Algorithm 4.1 describes integration of the stochastic reachable sets into an existing roadmap via the roadmap query process. Although the SR calculation is performed offline, Algorithm 4.1 is intended to run in real time, using the information currently available to the robot (i.e. obstacle locations). Paths are extracted using Dijkstra's algorithm [23]. However, to find paths of combined shortest distance and lowest probability of collision, the SR computation must be integrated into the roadmap edge weights. First, since the robot knows the location of each obstacle at the current time, the positions of the obstacles are updated to reflect their current locations. Second, each vertex in the roadmap is considered to be a waypoint. Updates of the roadmap weights are then performed at waypoints (see Algorithm 4.1, line 7). Updates consist of reweighting all edges (line 9), finding the path of lowest edge weight (line 11), querying the SR optimal control Equation (4.14) to determine the robot's speed and resulting trajectory (lines 13 and 14), and traversing along that edge with the determined robot speed for the allotted time (line 15). If the robot is not at a waypoint, then it continues along the

predetermined roadmap edge.

Two elements that are critical to the success of Algorithm 4.1 and atypical for probabilistic road maps are 1) updating of the obstacles, and 2) the subsequent effect on edge weights.

Regarding the first element, the likelihood of avoiding collision Equation (4.12) and the optimal control Equation (4.14) are evaluated over a discretized set of states, and are stored for use during run time for path planning. The algorithm propagates the location of the obstacles according to each obstacle's stochastic dynamics (4.1), (4.2). The stochastically determined obstacle speeds are chosen as per the randomization in Algorithm 4.2. The relative states are computed for every robot-obstacle pair in the environment.

Regarding the second element, edges define a transition between two configurations (see Section 4.1.1). These edges can be subdivided (often uniformly) into sets of discrete points defining the transition between configurations in the roadmap, and each point corresponds to a new intermediate configuration. The weight for a single edge is updated as in Algorithm 4.3.

For each intermediate configuration associated with an edge, the relative distance to each obstacle is calculated and the probability of collision avoidance at the current relative distance is queried for each obstacle. The minimum of all avoidance probabilities is taken as the weight for that configuration. This calculation is fast in comparison to standard collision detection methods whose computational complexity is defined by the number of polygons in the planning problem. The assigned edge weight is then the lowest probability of collision avoidance amongst all intermediate configurations for that edge, inverted for use in Dijkstra's algorithm (which finds minimum cost paths for graphs with nonnegative edge weights). Note that it is presumed that the time horizon N and time step Δ

in Algorithm 4.1 are the same as those in the reachability calculations.

4.1.4 Experiments

The method is evaluated via successful navigation in environments with several moving obstacles. Successful navigation is defined as the ability to find a path from a start state to goal state, without any collisions and within a specified time horizon. The stochastic reachable sets were computed in Matlab, and the SR Query was added to the Parasol Motion Planning Library (PMPL) from Texas A&M University. PMPL was also used to generate the initial roadmaps. Experiments were run on a single core of an Intel 3.40 GHz CORE i7-2600 CPU with 8 GB of RAM.

SR Query is compared to a Lazy-based method (Lazy) for moving obstacle avoidance [46]. The Lazy method updates the roadmap as obstacles move, by invalidating edges and vertices that are found to be in collision with the new position of the moving obstacles. This comparison shows the accuracy gained by considering the probabilities of collision instead of just the obstacles' current locations.

Furthermore, the flexibility of the method is shown by running experiments with vertex generation done with a uniform random distribution (PRM) [51] and with a regular cell decomposition (Grid) [59]. While cell decompositions can be ideal solutions, they are often infeasible for planning problems with several or complex static obstacles or of high dimensionality. In those cases, PRMs are often preferred. Since both types of roadmaps are treated the same way by the algorithm, the impact of the different roadmap toplogies is investigated. In the Grid roadmaps, every vertex is connected with up to 8 adjacent neighbors. PRM roadmaps are constructed with uniform random sampling and each vertex is con-

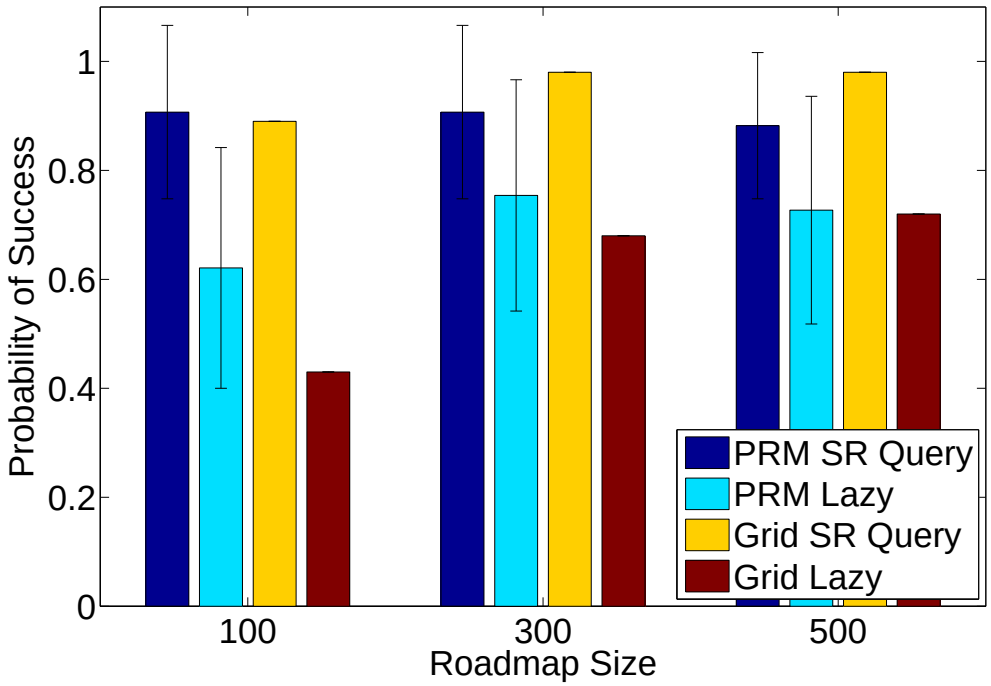
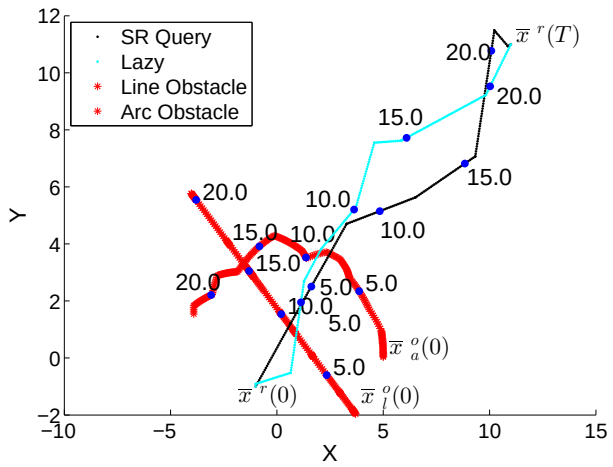


Figure 4.2: Comparison of SR Query and Lazy methods for the two dynamic obstacle experiment. Averaged likelihood of successfully traversing a collision-free path within the allotted time horizon for a given roadmap size, for Grid-based maps and PRM roadmaps. Note: Grid runs do not have error bars since there is only a single cell decomposition for a given roadmap size.

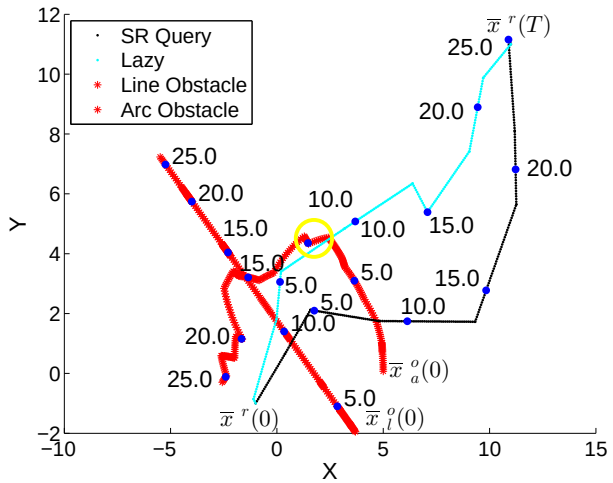
nected to its five closest neighbors.

Two Moving Obstacles

In this experiment, the robot navigates across a planning space while avoiding two dynamic obstacles: one follows straight-line dynamics (4.1) and the other follows constant-arc dynamics (4.2) from initial conditions $\overline{x}_l^o(0)$, $\overline{x}_a^o(0)$. The robot's start state and goal state are at the opposite corners of a 20×20 planning space (Figure 4.3a). The obstacle trajectories are chosen to generate sufficient opportunities for conflict with the robot, and obstacles may exit the planning space.



(a) Similar paths generated by SR Query and Lazy methods for two dynamic obstacle scenario.



(b) Different paths generated by SR Query and Lazy methods for two dynamic obstacle scenario.

Figure 4.3: Sample trajectories found by SR Query (black line) and Lazy (blue line) methods on a PRM roadmap with two obstacles (red lines) over $T = N\Delta = 30$ seconds. (a) Both methods found qualitatively similar paths, due to little obstacle interference. (b) SR Query and Lazy found very different paths, likely due to a near miss with one of the dynamic obstacles (yellow circle).

In order to evaluate the performance of the algorithm, roadmaps of $|\mathcal{N}| = 100$, 300, and 500 vertices are constructed using the standard PRM method. For each

map size, 10 random seeds were used to create 10 different PRM roadmaps. Grid roadmaps of size $\lfloor\sqrt{|\mathcal{N}|}\rfloor^2$ vertices were also produced, where $\mathcal{N}$ is the number of vertices in the corresponding PRM roadmap, to account for their square and unformly spaced vertex structure. One hundred obstacle pair trajectories were simulated, resulting in $10 \times 100 = 1,000$ simulations for each map size. The success of the algorithm was measured by collision-free path completion (the robot reaching the target) within the given time horizon. To be conservative, instances in which the robot did not find a collision-free path within the allotted time horizon were declared as unsuccessful. However, time horizons are only applicable for the SR Query method since the Lazy method is allowed to run until a path is found, no path exists, or a collision occurs. Each simulation was run for $T = 30$ seconds with a sampling interval of $\Delta = 1$ seconds ($N = 30$ time steps).

Figure 4.2 shows the effect of map size as well as the relative effectiveness of the two methods on PRM and Grid roadmaps in terms of the mean percentage of success. For the PRM roadmaps, SR Query was able to find successful paths 88% to 91% of the time, for roadmaps with 100 and 500 vertices, respectively. The error bars show how the randomized roadmap structures impact the success rate. In comparison, the Lazy method found successful paths 63% to 75% of the time. Unsuccessful runs of Lazy were due either to pruned vertices and edges that made traversal to the goal impossible, or direct collision with a moving obstacle.Error bars are not included for Grid due to the static map structure of a cell decomposition. In comparing Grid-based maps to PRM roadmaps, one finds that the Grid-based maps produce better results for SR Query with larger map sizes, but poorer results for Lazy (for all map sizes). This is consistent with evidence that Grids perform as well or better than randomized roadmaps in environments without static obstacles [59]. In all cases (Grid-based or PRM roadmaps), the SR Query method performs between 15% and 45% better than the Lazy method.

Furthermore, the paths selected by the two algorithms were examined. In Figure 4.3a, the path generated via the SR Query method (black line) is fairly similar to the path generated via the Lazy method. The moving obstacles are shown in red, and time is indicated as labeled waypoints along each path. Both Lazy and SR Query methods follow the same path initially, but at around $t = 10$ seconds, the SR Query method identifies an incoming obstacle and moves the robot away from the obstacle. However, the Lazy method does not anticipate a possible collision, and so it does not change its path. In this case, the Lazy method allows the robot to barely pass in front of the obstacle. A similar near collision for the Lazy method is shown in Figure 4.3b. In this example, the paths for SR Query and Lazy are the same for the first 10 seconds. Again, SR Query anticipates an incoming obstacle and changes its path to avert a possible collision. The Lazy method generates a path for the robot that passes in front of the obstacle with very little clearance. This near miss is highlighted in Figure 4.3b inside the yellow circle.

Algorithm 4.1 SR Query

Input: Obstacles O, *Roadmap*, Max time $T = N \cdot \Delta$

Output: boolean *Success*

1: $nextNode = start$
2: $previousNode = start$
3: **for** $t_n = 0; t_n < T; n = n + 1$ **do**
4: **for** Obstacle $o \in O$ **do**
5: $updateObstacle(o)$
6: **end for**
7: **if** at(robot, nextNode) **then**
8: **for** each edge $e \in Roadmap$ **do**
9: $EdgeWeight = updateEdgeWeights(e, O)$
10: **end for**
11: $Path = Dijkstras(previousNode, GoalNode)$
12: $nextNode = Path.next$
13: $x^r_{n+1} = Path.next.getXVelocity()$
14: $y^r_{n+1} = Path.next.getYVelocity()$
15: **end if**
16: $\overline{x}^r_{n+1} = interp(previousNode, nextNode, t_n)$
17: **end for**

Algorithm 4.2 updateObstacle

Input: time $t_n = n \cdot \Delta$, obstacle o, velocities $w \in \mathcal{W} = \{w_1, w_2, ..., w_{n_W}\}$, probabilities $p(w)$

1: **if** $mod(t_n, 1) == 0$ **then**
2: $s = rand(0, 1)$
3: **for** $index = 0; index < n_W; index{+}{+}$ **do**
4: **if** $s \leq p(w)[index]$ **then**
5: $o.w = w[index]$
6: break
7: **end if**
8: **end for**
9: **end if**
10: $\overline{x}^o_{n+1} = \overline{x}^o_n + \Delta \cdot f(o.w, t_n)$

Algorithm 4.3 updateEdgeWeight

Input: Edge e, Obstacles O

1: $EdgeWeight = 0$
2: **for** Configuration $c \in e$ **do**
3: $\quad PROB = 1$
4: $\quad$ **for** Obstacle $o \in O$ **do**
5: $\quad\quad \tilde{x} = c - o$
6: $\quad\quad PROB = \min\{PROB, o.V_0^*(\tilde{x})\}$
7: $\quad$ **end for**
8: $\quad$ **if** $PROB < EdgeWeight$ **then**
9: $\quad\quad EdgeWeight = PROB$
10: $\quad$ **end if**
11: **end for**
12: $e.Weight = \frac{1}{EdgeWeight}$

Fifty Moving Obstacles

In this experiment, a robot navigates across a 60×60 planning space while avoiding 50 dynamic obstacles, O_i, $i \in \{1, \cdots, 50\}$. Twenty-five of the obstacles have straight-line dynamics (4.1), five each traveling along lines with $\alpha \in \{-1.5, -1, -0.5, +0.5, 1.0\}$, respectively. The other 25 dynamic obstacles have constant-arc dynamics (4.2), 10 each with radius $r = 50$, 10 with $r = 40$, and five with $r = 30$. The speeds and associated probabilities for each obstacle are as described in Section 4.1.3.

Ten of the three roadmap sizes were constructed, as in Section 4.1.4. Again 100 obstacle trajectories were generated, resulting in 1000 total simulations for each map size. Since obtaining a feasible path is more difficult with so many more obstacles, the time horizon was increased to $T = 100$ seconds.

Figure 4.4 shows the effect of map size as well as the average success rate of the two methods on PRM and Grid roadmaps. As this is a significantly harder problem, the percentages of success are lower as compared to the two obstacle scenario in Figure 4.2. However, in all cases the SR Query method is at least 20% better than the Lazy method. Interestingly, the Grid-based solution is significantly more successful than the PRM-based method. This is likely due to the regular spacing of the roadmap vertices, which prevents long edges and allows the algorithm to make quicker replanning decisions. However, this advantage would not likely exist in more complex environments. As in Section 4.1.4, the error bars in Figure 4.4 indicate the significant impact of randomization in the PRM roadmap.

The 50 obstacle test in Figure 4.4 has lower success rates than the two obstacle test in Figure 4.2 because of two factors. First, finding a collision-free path is significantly harder with 50 obstacles as opposed to merely two. Second, the roadmap density, defined as the number of vertices per area of the planning space,

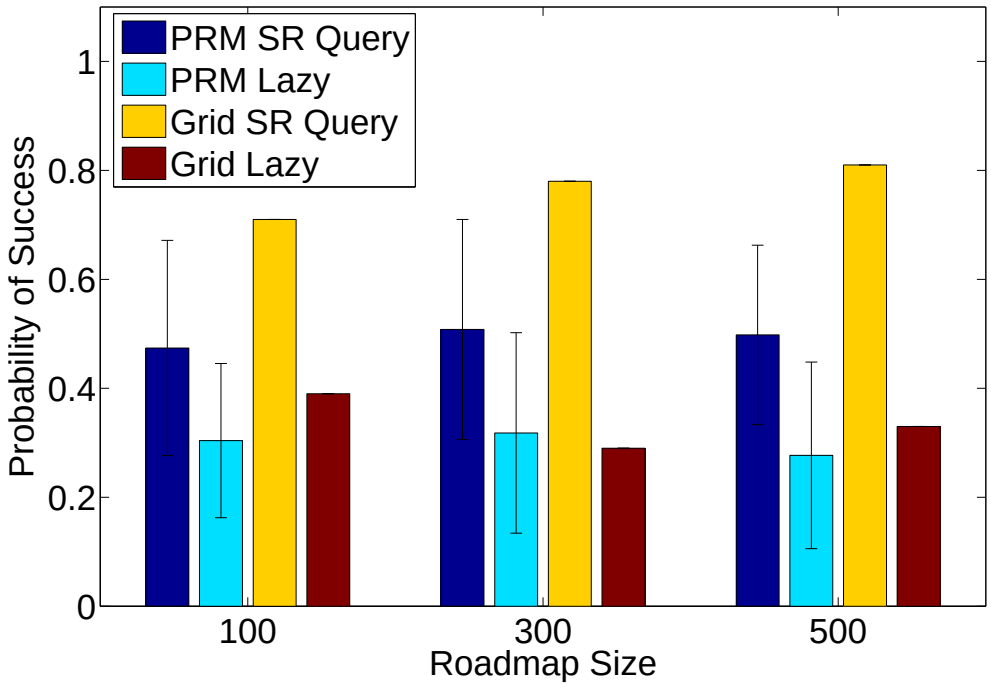


Figure 4.4: Comparison of SR Query and Lazy methods for the 50 dynamic obstacle experiment. Averaged likelihood of successfully traversing a collision-free path within the allotted time horizon for a given roadmap size, for Grid-based maps and PRM roadmaps. Note: Grid runs do not have error bars since there is only a single cell decomposition for a given roadmap size.

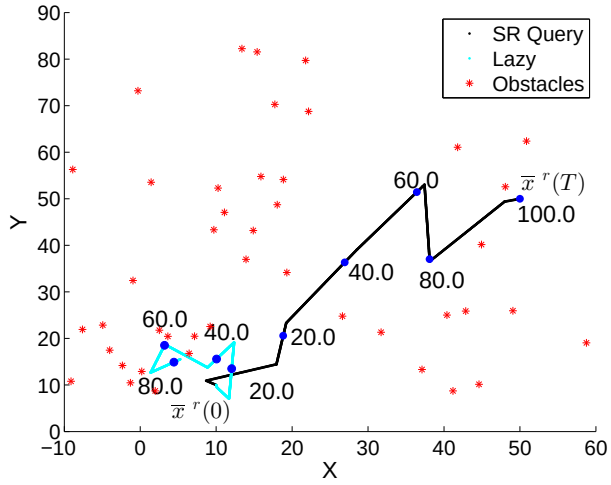
is lower with 50 obstacles than with two obstacles. Since the roadmap sizes are the same, but the area increases from 20×20 to 60×60, the 50 obstacle tests have *lower* roadmap density. Lower density roadmaps force the robot to travel greater distances before path replanning (which occurs in Algorithm 4.1 at roadmap vertices), and consequently should have more collisions. However, relatively high success rates are evident for the SR Query methods, especially via Grid methods, likely due to the even distribution of vertices that allow for consistent replanning.

Figures 4.5a and 4.5b show two sample trajectories, one in which the SR Query method significantly outperforms the Lazy method, and another in which the two

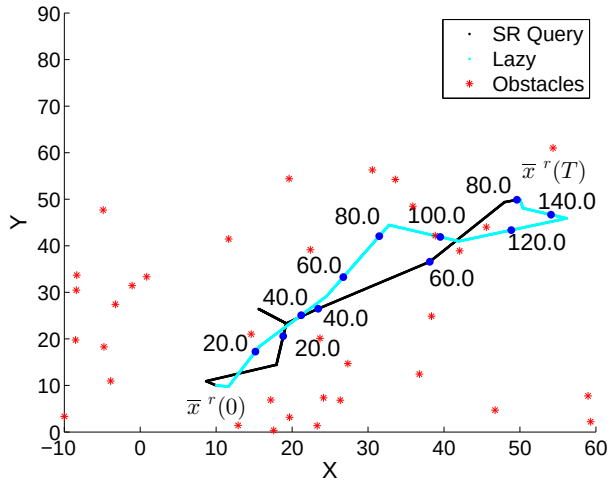
methods behave comparably.

4.1.5 Conclusions

Here stochastic reachability was successfully incorporated into motion planning roadmaps, in order to develop a novel planning algorithm that accounts for stochastically moving obstacles. SR sets for individual obstacles wre combined into a single planning solution, generating an upper bound on the total avoidance probability with several obstacles. The method was demonstrated on an example with 50 obstacles and on two methods of roadmap construction. By combining roadmaps with stochastic reachability, the algorithm significantly outperforms another existing roadmap-based method for moving obstacles.



(a) Fifty dynamic obstacles scenario in which Lazy method results in collision, SR Query method does not.



(b) Fifty dynamic obstacle scenario in which both methods successfully find collision-free paths.

Figure 4.5: Sample trajectories found by SR Query (black line) and Lazy (blue line) methods on a PRM roadmap with 50 obstacles shown at 80s into the simulation (red squares). Total simulation time $T = N\Delta = 100$s. (a) The Lazy method results in collision, but SR Query method successfully reaches the goal state without collision. (b) Sample trajectories (as in (a)), in which similar successful paths are found via both methods. *Movies of the 50 moving obstacle simulations are available at* `https://www.cs.unm.edu/amprg/Research/DO/`

4.2 APF-SR

Navigation in dynamic, uncertain environments is a difficult yet ubiquitous problem in transportation systems (e.g., autonomous driving, shipping lanes near ports, air traffic management) and distributed robotic systems (vehicle swarms in air, ground, or water environments), with application to problems in search and rescue, coordinated movement, distributed monitoring and surveillance, and others. The problem of motion planning is considered in environments with hundreds of stochastic, dynamic obstacles, in which the obstacles can arbitrarily switch between trajectories that follow a constant radius arc or a straight line, with stochastic angular or translational speeds, respectively. This kind of motion is a realistic abstraction of dynamics seen in air traffic control systems, highway and other ground transportation systems, and others.

While control theoretic methods have been developed to provide assurances of performance despite stochasticity in low dimensional systems, they are computationally infeasible when the environment has tens to hundreds of dynamic obstacles. Stochastic reachability analysis provides offline verification of dynamical systems to assess whether the state of the system will, with a certain likelihood, remain within a desired subset of the state-space for some finite time, or avoid an undesired subset of the state-space [1]. To solve problems in collision avoidance, the region in the relative state-space which constitutes collision is defined as the set of states the system should avoid [98, 49]. Unfortunately, the computation time for stochastic reachable sets (SR sets) is exponential in the dimension of the continuous state, hence assessment of collision probabilities with many simultaneously moving obstacles is not feasible.

This section proposes a solution that combines ad-hoc and formal methods, to incorporate the effect of likely obstacle motion into the desired path, and exploit

computationally efficient paradigms that can be used in real time. In brief, this method weights an artificial potential field (APF) with stochastic reachable (SR) sets, computed pairwise between the robot and each stochastic, dynamic obstacle. Computational efficiency is achieved by pre-computing the SR sets offline for a finite set of obstacle types, then querying those sets at run-time to construct a repulsive potential field around each obstacle.

Preliminary versions of this method were implemented via roadmap methods for dynamic path queries (SR-Query) [64] and via APF methods for stochastic obstacles that followed simple line or arc trajectories [17] (but did not switch between these trajectories). SR-Query was more successful in identifying collision-free paths in environments with 50 moving obstacles than a roadmap-based approach that simply pruned invalid edges during dynamic path queries [46], but was susceptible to **fast unseen** moving obstacles, due to limited reactivity and required navigation on the roadmap edges. In [17], the advantages of APF methods over roadmap-based methods in environments with up to 300 stochastic, dynamic obstacles were demonstrated. The performance of incorporating SR sets with APF methods, as opposed to ad-hoc methods [57, 108, 32] for computing repulsion fields was also evaluated.

In this section, the main contributions are to a) extend APF-SR to accommodate stochastic hybrid *obstacles, which are far more realistic and capture behavior that is much more representative of real-world dynamic obstacles, b) present a thorough parameter exploration of APF-SR and comparative analysis between APF-SR and other methods, and c) demonstrate this method in environments with up to 900 obstacles (approximately an order of magnitude more obstacles than considered in previous implementations).*

Section 4.3 describes the problem formulation, modeling, and stochastic reachability analysis. Section 4.4 presents APF-SR. In Section 4.5, an extensive parameter evaluation is conduct, and APF-SR is shown to be more robust than the related

Gaussian method. A comparison of success rate, path length and cause of failure with a holonomic and a unicycle robot in environments with up to 900 obstacles is performed. Then APF-SR is shown to outperform the Gaussian based comparison method by up to 60% in the holonomic case and up to 20% in the unicycle case with 900 obstacles. Lastly, Section 4.6 provides conclusions and directions for future work.

4.3 Modeling and Stochastic Reachability Analysis

4.3.1 Robot Dynamics

Two models for the robot are considered: 1) a holonomic point-mass model and 2) a non-holonomic unicycle model, with state $\overline{x}^r = [x^r, y^r, \theta^r] \in \mathbb{R}^3$ representing its position and heading angle. The holonomic model

$$\begin{aligned} \dot{x}^r &= u^x \\ \dot{y}^r &= u^y \\ \dot{\theta}^r &= 0 \end{aligned} \tag{4.17}$$

has velocity control input $u = [u^x, u^y] \in \mathbb{R}^2$. The non-holonomic unicycle model

$$\begin{aligned} \dot{x}^r &= u^s \cos\theta^r \\ \dot{y}^r &= u^s \sin\theta^r \\ \dot{\theta}^r &= u^w \end{aligned} \tag{4.18}$$

has control input $u = [u^s, u^w] \in \mathbb{R}^2$, such that u^s is the speed and u^w is the angular velocity of the unicycle. Discretizing the robot dynamics Equations (4.17) and (4.18) using an Euler approximation with time step Δ results in $\overline{x}^r_{n+1} =$

$\overline{x}^r_n + \Delta f^r(u_n, \theta^r_n)$, with

$$f^r(u_n, \theta^r_n) = \begin{bmatrix} u^x_n \\ u^y_n \\ 0 \end{bmatrix} \tag{4.19}$$

for the holonomic robot and

$$f^r(u_n, \theta^r_n) = \begin{bmatrix} u^s_n \cos\theta^r_n \\ u^s_n \sin\theta^r_n \\ u^w_n \end{bmatrix} \tag{4.20}$$

for the unicycle robot.

4.3.2 Obstacle Dynamics

Each obstacle is represented as a point mass with state $\overline{x}^o = [x^o, y^o, \theta^o] \in \mathbb{R}^3$, that follows either a straight-line or constant-arc trajectory with stochastic velocity w^l or stochastic angular velocity w^a, respectively. The random variables w^l and w^s take values in $\mathcal{W}^l$ and $\mathcal{W}^a$, respectively, with probability distributions $p^l(w)$ and $p^s(w)$. The obstacle dynamics discretized with time step Δ are $\overline{x}^o_{n+1} = \overline{x}^o_n + \Delta f^o(w_n, \theta^o_n)$, with

$$f^o(w_n, \theta^o_n) = \begin{bmatrix} w_n \\ \alpha w_n \\ 0 \end{bmatrix} \tag{4.21}$$

for straight-line motion, with speed $w \in \mathcal{W}$ and line slope $\alpha \in \mathbb{R}$ determined by the heading angle θ^o_n (i.e. $\alpha = \tan\theta^o_0$), and

$$f^o(w_n, \theta^o_n) = \begin{bmatrix} r w_n \cos\theta^o_n \\ r w_n \sin\theta^o_n \\ w_n \end{bmatrix} \tag{4.22}$$

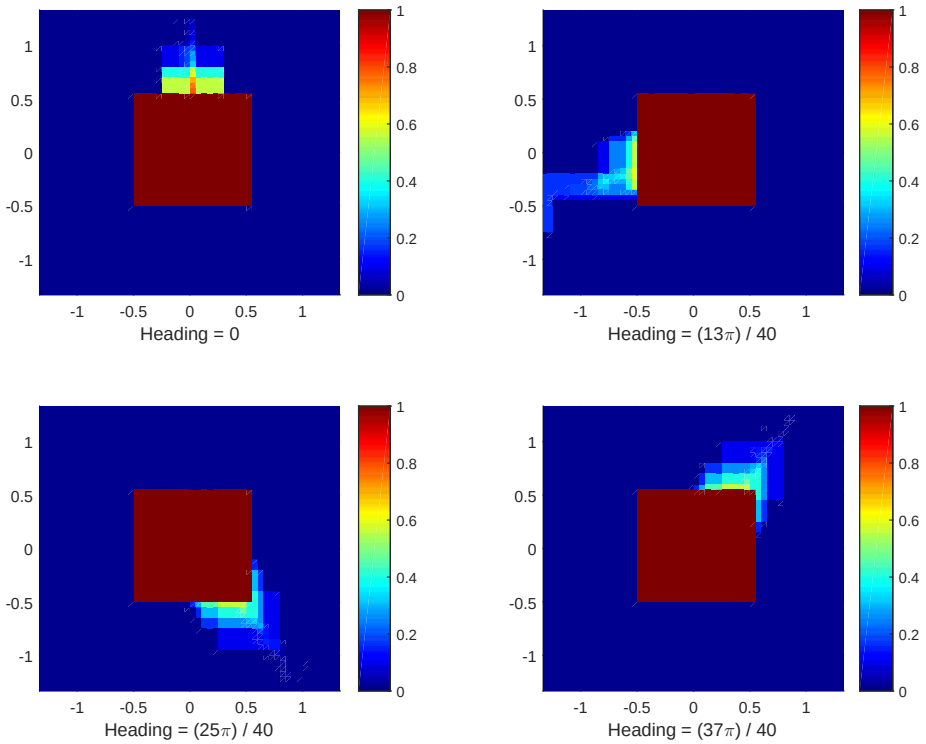


Figure 4.6: Stochastic reachable set for relative robot-obstacle dynamics (4.25) with Markov switching (4.23), (4.26) in arc mode. Since the SR set is 3D, this plot visualizes the likelihood of safety with respect to relative position $(\tilde{x}, \tilde{y})$ for four selected values of relative heading $\tilde{\theta}$.

for constant-arc movement, with angular speed $w \in \mathcal{W}$ and radius $r \in \mathbb{R}^+$.

The obstacles are allowed to switch between a straight line trajectory and one of three arc trajectories (Figure 4.6 shows the SR set for an arc). Hence at any instant, the obstacle may take on continuous dynamics associated with one of four modes, $\mathcal{Q} = \{\text{line}, \text{arc}_1, \text{arc}_2, \text{arc}_3\}$, with arc trajectories distinguished correspondingly by different radii $0 < r_1 < r_2 < r_3$. Further, continuity of the heading angle is presumed, such that the angle α of the line trajectory is completely specified by the obstacle heading at the previous instant, upon exiting an arc trajectory.

The switching dynamics are described by a stochastic process, such that the duration of time spent in a given mode is modeled similarly to an exponential distribution. This work presumes that the likelihood of switching from mode q_i to mode q_j, $q_i, q_j \in \mathcal{Q}$ is given by

$$\begin{aligned} p_Q(\text{line}, \text{arc}_i) &= \tfrac{1}{3}\left(1 - \beta_n^{\text{line}}\right) && \forall i \in \{1,2,3\} \\ p_Q(\text{line}, \text{line}) &= \beta_n^{\text{line}} && \\ p_Q(\text{arc}_i, \text{line}) &= 1 - \beta_n^{\text{arc}} && \forall i \in \{1,2,3\} \\ p_Q(\text{arc}_i, \text{arc}_i) &= \beta_n^{\text{arc}} && \forall i \in \{1,2,3\} \\ p_Q(\text{arc}_i, \text{arc}_j) &= 0 && \forall i, j \in \{1,2,3\} \end{aligned} \tag{4.23}$$

with $\beta_n^{\text{line}} = e^{-\frac{\Delta(n-n_s)}{S}(1-R_{\text{line}})}$, $\beta_n^{\text{arc}} = e^{-\frac{\Delta(n-n_s)}{S}(1-R_{\text{arc}})}$, $\Delta \cdot n_s$ the time that the obstacle last switched, S the switching–time–parameter, R_{line} the fraction of obstacles in the game space that are following line trajectories, and $R_{\text{arc}} = 1 - R_{\text{line}}$. The switching–time–parameter allows for the tuning of the switching rates of obstacles, such that lower values of S increase the switching rate and higher values of S decrease the switching rate. For example, a switching–time–parameter set to a value much greater than the simulation running time will produce a negligibly small probability of an obstacle switching. This process assures that excessive switching is unlikely, and also that the total number of obstacles in the game space following arc and line trajectories remains approximately constant.

4.3.3 Relative robot-obstacle dynamics

The relative dynamics between the robot and a single obstacle is modeled by examining the motion of the obstacle with respect to a coordinate frame affixed to

the robot, via standard kinematic analysis. The relative state is defined as:

$$\tilde{x} = \begin{bmatrix} R^T(\theta^r) \begin{bmatrix} x^o - x^r \\ y^o - y^r \end{bmatrix} \\ \theta^o - \theta^r \end{bmatrix} \in \mathbb{R}^3 \tag{4.24}$$

in terms of a standard rotation matrix $R(\cdot)$, with dynamics

$$\begin{aligned} \tilde{x}_{n+1} &= \tilde{x}_n + \Delta \cdot \left[\begin{array}{c|c} R^T(\theta_r) & 0_{1\times 2} \\ \hline 0_{2\times 1} & 1 \end{array}\right] (f^r(u_n, \theta_n^r) - f^o(w_n, \theta_n^o)) \\ &= \tilde{x}_n + \Delta \tilde{f}(u_n, w_n, \theta_n^r, \theta_n^o) \end{aligned} \tag{4.25}$$

For the purpose of computing the SR sets (but not in simulation), the Poisson-like distribution (4.23) is approximated by a Markov process by presuming constant values for

$$\begin{aligned} \beta_n^{\text{line}} &= \beta^{\text{line}} \\ \beta_n^{\text{arc}} &= \beta^{\text{arc}} \end{aligned} \tag{4.26}$$

This approximation is computed empirically for a given Poisson-like distribution by finding the average switch rate per Δ over 10,000 trials. The approximation simplifies the dynamics, and enables us to express the resulting system as a discrete-time stochastic hybrid system (DTSHS), described by the tuple $\mathcal{H} = (\tilde{\mathcal{X}}, \mathcal{Q}, \mathcal{U}, T_x, T_q)$, with

- $\tilde{\mathcal{X}} \subseteq \mathbb{R}^3$ the set of continuous states representing relative coordinates
- $\mathcal{Q} = \{\text{line}, \text{arc}_1, \text{arc}_2, \text{arc}_3\}$ a finite set of discrete modes, with $\mathcal{S} = \tilde{\mathcal{X}} \times \mathcal{Q}$ the hybrid state space
- $\mathcal{U} \subseteq \mathbb{R}^2$ a compact Borel space which contains all possible control inputs
- $T_x : \mathbb{R}^3 \times \mathcal{Q} \times \mathcal{S} \times \mathcal{U} \to [0,1]$ a stochastic transition kernel that assigns a probability distribution to $\tilde{x}_{n+1}$ conditioned on $\tilde{x}_n$, q_{n+1}, and u_n, $T_x(\tilde{x}_{n+1} | \tilde{x}_n, q_{n+1}, u_n)$ for all n

- $T_q : \mathcal{Q} \times \mathcal{Q} \to [0,1]$ a discrete transition kernel that assigns a probability distribution to q_{n+1} conditioned on q_n, $T_q(q_{n+1}|q_n)$.

The sets $\mathcal{W}^l$ and $\mathcal{W}^s$ are each assumed to be finite, and therefore can define the transition kernel T_x as follows.

$$T_x(\tilde{x}_{n+1}|\tilde{x}_n, q_{n+1}, u_n) = \begin{cases} p^l(w^*), \text{for } q_{n+1} = \text{line}, \\ w^* = \tilde{f}^{-1}(u_n, \tilde{x}_{n+1} - \tilde{x}_n, \theta_n^r, \theta_n^o) \\ p^a(w^*), \text{for } q_{n+1} = \text{arc}_i, \\ \forall\, i,\, w^* = \tilde{f}^{-1}(u_n, \tilde{x}_{n+1} - \tilde{x}_n, \theta_n^r, \theta_n^o) \end{cases} \tag{4.27}$$

If w^*, the unique solution to Equation (4.25) for a given $\tilde{x}_{n+1}$, $\tilde{x}_n$, and u_n, is not a member of $\mathcal{W}^l$ or $\mathcal{W}^a$, the probability of obtaining $\tilde{x}_{n+1}$ is zero. The transition kernel for the mode is given by p_Q, except that β_n^{line} and β_n^{arc} are replaced by β^{line} and β^{arc} from Equation (4.26), respectively. For ease of notation, the continuous and discrete state transition kernels are combined, such that

$$\tau(\tilde{x}_{n+1}, q_{n+1}|\tilde{x}_n, q_n, u_n) = T_x(\tilde{x}_{n+1}|\tilde{x}_n, u_n, q_{n+1}) \times T_q(q_{n+1}|q_n) \tag{4.28}$$

4.3.4 Stochastic Reachable Sets for Collision Avoidance

It is presumed a collision occurs between the robot and a single obstacle whenever

$$\|\overline{x}_n^r - \overline{x}_n^o\|_1 \leq \epsilon \tag{4.29}$$

for some n and some distance ϵ, and define the *avoid set*, $\overline{K}$, as the set of states in which a collision is said to occur (4.29).

Collision avoidance probabilities are generated through stochastic reachability analysis. To avoid collision with the obstacle, the robot should remain within K, the complement of $\overline{K}$. The probability that the robot remains within K over N time steps, with initial relative position $\tilde{x}_0$, can be calculated using dynamic programming [10], introduced for stochastic reachable set generation in [1]. To compute the SR set, a value function is iterated backwards in time from $n = N$ to time $n = 0$,

$$V_N^*(\tilde{x}, q) = \mathbf{1}_K(\tilde{x}) \tag{4.30}$$

$$V_n^*(\tilde{x}, q) = \max_{u \in \mathcal{U}} \mathbf{1}_K(\tilde{x}) \sum_{Q} \int_{\mathcal{X}} V_{n+1}^*(\tilde{x}', q') \tau(\tilde{x}', q' \mid \tilde{x}, u, q)\, d\tilde{x}' \tag{4.31}$$

in which an indicator function $\mathbf{1}_K(x)$ is equal to 1 if $x \in K$ and equal to 0 otherwise. The value function $V_0^*(\tilde{x}_0, q_0)$ at time $n = 0$ describes the probability of avoiding collision over N time steps when starting in some initial state $\tilde{x}_0$ and initial mode q_0.

Note that Equations (4.30) - (4.31) generally do not have a closed form expression, and must be evaluated manually for all possible $(\tilde{x}_n, q_n) \in \tilde{\mathcal{X}} \times \mathcal{Q}$. For $\tilde{\mathcal{X}} \subseteq \mathbb{R}^3$, this requires a discretization step to only evaluate Equations (4.30) - (4.31) for a finite number of $(\tilde{x}_n, q_n)$, which results in an approximate solution. This work does not consider any errors in the resulting SR set because of this approximation, or the approximation Equation (4.26), and it treats $V_0^*(\tilde{x}_0, q_0)$ as the actual probability of collision.

Figure 4.6 depicts $V_0^*(\tilde{x}_0, \text{arc})$, the SR set for an obstacle initially in constant-arc mode, with a unicycle robot. Figures 4.7a and 4.7c depict $V_0^*(\tilde{x}_0, \text{line})$, the SR set for an obstacle initially in straight-line mode with a point-mass and unicycle robot, respectively. The heat maps show a higher probability of collision when the robot is in line with the obstacle's trajectory. Intuitively, the closer the robot is to the obstacle, the higher the probability of collision. On a single core of an Intel

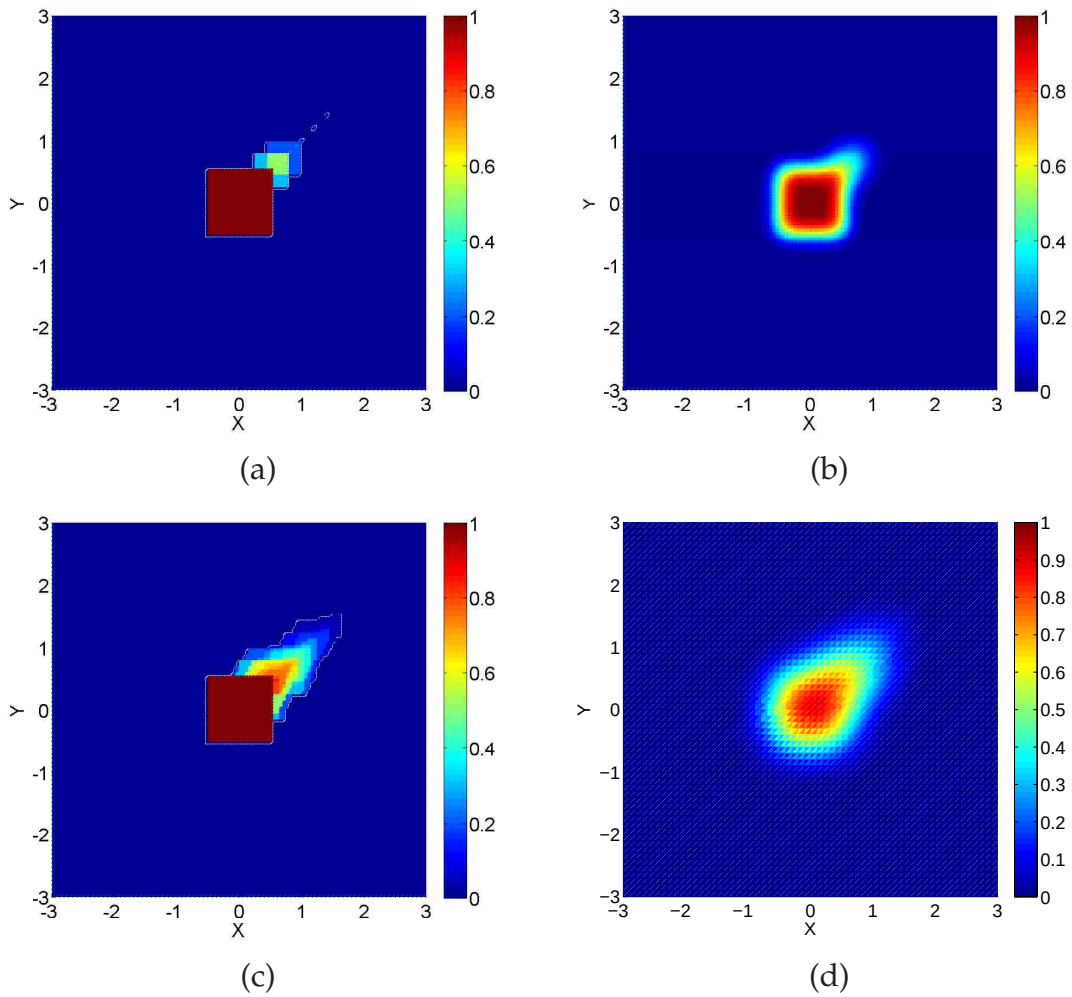


Figure 4.7: SR sets for an obstacle initially in line mode; the color represents probability of collision. (a) Raw SR set with a holonomic robot. (b) Smoothed SR set after convolution with a Gaussian ($\sigma = 0.15$). (c) Raw SR set with the unicycle robot. (d) Smoothed SR set after convolution with a Gaussian ($\sigma = 0.15$).

3.40 GHz CORE i7-2600 CPU with 8 GB of RAM, the SR set in Figure 4.7a took 1727.25 seconds to compute, over a horizon of $N = 30$ steps, with time step of length $\Delta = 1$. Convergence is observed in the stochastic reachable sets for $N > 5$ since the robot and obstacle traveled sufficiently far apart within this time frame.

When used in environments with a single obstacle, $V_0^*(\tilde{x}_0, q_0)$, Equation (4.30),

is the maximum probability of avoiding a collision, and a tight upper bound. To consider environments with multiple obstacles, let B_i correspond to the event that the robot avoids collision with obstacle $i \in \{1, \cdots, M\}$. This work presumes that collision avoidance probabilities are calculated separately for each obstacle $V_0^{*,1}(\tilde{x}_0^1, q_0^1)$, $V_0^{*,2}(\tilde{x}_0^2, q_0^2)$, ..., $V_0^{*,M}(\tilde{x}_0^M, q_0^M)$ (with relative position $\tilde{x}_0^i$ with respect to obstacle i in mode q_0^i). Then the probability of avoiding collision with *all* obstacles is

$$\begin{aligned} &\mathbb{P}[B_1 \cap B_2 \cap \cdots \cap B_M] \leq \\ &\min\{V_0^{*,1}(\tilde{x}_0^1, q_0^1), V_0^{*,2}(\tilde{x}_0^2, q_0^2), \cdots, V_0^{*,M}(\tilde{x}_0^M, q_0^M)\} \end{aligned} \tag{4.32}$$

Hence by computing the minimum value over all probabilities of collision avoidance with each obstacle individually, an upper bound to the total collision avoidance probability is obtained. While this upper bound does not provide a guarantee of safety, it can inform which paths are relatively more likely to avoid collision. Since the focus is on finding paths with higher success rates, rather than theoretically guaranteed collision-free paths, the upper bound Equation (4.32) is appropriate. Further discussion and the derivation of Equation (4.32) is given in [64].

4.4 Methods

APF-SR differs from SR-Query in several ways. Primarily, APF-SR uses SR sets to build a repulsive field while SR-Query directly maps the SR set to a roadmap. APF-SR then utilizes a traditional APF gradient planning method while SR-Query uses graph serach for path planning. The commonality between the two methods is that they both leverage SR set to construct probabilities of collisision with moving obstacles.

In this section, a novel method for integrating SR sets with APF methods is

presented. First the SR sets are smoothed, then they are incorporated into the gradient calculation and finally the robot's control law is updated based on the gradient calculation. Note that the SR sets are calculated with a time step of Δ, however the robot and obstacle states are updated at a time step of δ such that $\delta < \Delta$.

One hurdle in using SR sets to inform the potential field is the possibility of non-smoothness in the optimal value for Equation (4.31). In general, no guarantees of smoothness are possible. In fact, there is a marked discontinuity in the part of the SR set corresponding to a robot located just behind the obstacle (Figure 4.7). Since APF methods use a gradient as a warning that the robot is about to collide with an obstacle, the SR set is smoothed by convolving the set with a Gaussian, $\mathcal{N}(\mu = 0, \sigma^2)$. Figure 4.7 shows the raw SR set for a point-mass and unicycle robot (4.7a and 4.7c, respectively) as well as the the resulting set after convolution with a Gaussian (4.7b and 4.7d, respectively). As expected, the discontinuity in Figure 4.7a from 0 to 1 at the obstacle boundary is smoothed in Figure 4.7b.

Algorithm 4.4 calculates the APF gradient by summing the obstacle gradients, calculated in *getAPFGradient* (4.6), and the *goal–vector* **(Lines 5-9)**, which is then used by *calcControl* (4.7) to construct the control input u **(Line 12)**. The *goal–vector* is a small magnitude vector which always points towards the goal relative to the robot's current position. Thus, the APF gradient is the direction the robot should move in to avoid obstacles and reach the goal. Finally, the control law for the robot is updated with the control input u **(Line 10)**.

The *updateObstacle* function (Algorithm 4.5) updates the position of the obstacle at each time step. It first updates the mode of the obstacle, by comparing the probability of switching from the current mode to another mode ($probSwitch(o, t)$, which is described by the likelihood function (4.23)) against a randomly generated number between 0 and 1 ($Random.nextRandom(0, 1)$). If the probability of switch-

Algorithm 4.4 APF-SR

Input: obstacles O with precomputed smoothed SR sets, robot r

---

1: **for** $t = 0; t < maxTime; t = t + \delta$ **do**
2: $APF_{vector} = (0, 0)$
3: **for** Obstacle $o \in O$ **do**
4: updateObstacle(t,o,$o.w$,$o.p(w)$)
5: **if** dist($\overline{x}_n^o$, $\overline{x}_n^r$) $< d_{min}$ **then**
6: $APF_{vector} = APF_{vector} + o.getAPFGradient(\overline{x}_n^r)$
7: **end if**
8: **end for**
9: $APF_{vector} = APF_{vector} + goal\text{–}vector$
10: $[u, \theta] = calcControl(APF_{vector})$
11: $\overline{x}_{n+1}^r = \overline{x}_n^r + t \cdot f^r(u, \theta)$
12: **end for**

ing is greater than the randomly generated number, the mode changes from line to arc (or arc to line). Second, the function then updates the obstacle position according to the appropriate obstacle dynamics (arc or line). The speed w of the obstacle is sampled according to the distribution $p(w)$ of possible speeds (Lines 2-9), and the obstacle dynamics updated appropriately (Line 13).

The $getAPFGradient(\overline{x}_n^r)$ function, Algorithm 4.6, calculates the APF gradient for all obstacles near the robot. For every obstacle o, if o is within distance d_{min} of the robot, then the method queries the potential field influence of o on the robot. This gradient is calculated by first finding the smallest neighboring location, $p_{i,j}$, in the smoothed SR set. $p_{i,j}$ is the relative coordinate point in the stochastic reachability set, where $SR(p_{i,j})$ is the value in the stochastic reachability set at point $p_{i,j}$.

Algorithm 4.5 updateObstacle

Input: Time step n, sample interval t. obstacle o, velocities $w \in \mathcal{W} = \{w_1, w_2, ..., w_{n_W}\}$, probabilities $p(w)$

1: **if** $mod(n, t/\Delta) == 0$ **then**
2: **if** probSwitch(o,t) > Random.nextRandom(0,1) **then**
3: $swapDynamics(o)$
4: **end if**
5: $s = rand(0, 1)$
6: **for** $index = 0; index < n_W; index$++ **do**
7: **if** $s \leq p(w)[index]$ **then**
8: $o.w = w[index]$
9: break
10: **end if**
11: **end for**
12: **end if**
13: $\overline{x}^o_{n+1} = \overline{x}^o_n + \Delta \cdot f^o(o.w, o.\theta)$

The gradient is then calculated by the second order central finite difference centered at i, j. The gradient from each obstacle is then summed together to produce a net collision avoidance gradient due to all of the obstacles within some local distance d_{min}.

The *calcControl*(APF_{vector}) function, Algorithm 4.7, calculates the control input u. For the holonomic case $u = APF_{vector}$. However, for the non-holonomic case a heading and speed must be extracted from the APF_{vector} to construct $u = (u^s, u^w)$. This is done by first setting u^w to the maximum turn rate in the direction of the APF_{vector}, then setting u^s to the maximum velocity in the direction of the APF_{vector}.

Algorithm 4.6 o.getAPFGradient

Input: $\overline{x}_n^r$

Output: $o.G$

1: **if** $\|\overline{x}_n^r - \overline{x}_n^o\| \leq d_{min}$ **then**
2: $\{i, j\} = \overline{x}_n^r$
3: $o.G = \{(\frac{o.SR(p_{i-1,j}) + o.SR(p_{i-2,j})}{2}) - (\frac{o.SR(p_{i+1,j}) + o.SR(p_{i+2,j})}{2}), (\frac{o.SR(p_{i,j-1}) + o.SR(p_{i,j-2})}{2}) - (\frac{o.SR(p_{i,j+1}) + o.SR(p_{i,j+2})}{2})\}$
4: **else**
5: $o.G = 0$
6: **end if**

The maximum velocity of the unicycle is the same as the maximum velocity used in the SR calculation. Finally, u is used to update the control law for the robot.

4.5 Experiments

4.5.1 Experimental Setup

The method is evaluated in environments with hundreds of moving obstacles, and successful navigation is defined as the ability to find a path from a start state to goal state, without any collisions and within a specified time horizon. All experiments take place with the same environment, with randomized initial obstacle start locations. The environment is a toroidal circle of radius 50, and to maintain consistent obstacle density, an obstacle exiting the circle wraps around the boundary of the environment, re-entering π radians away from the point of exit, with the same velocity as upon its exit. Figure 4.8 shows an example of the environ-

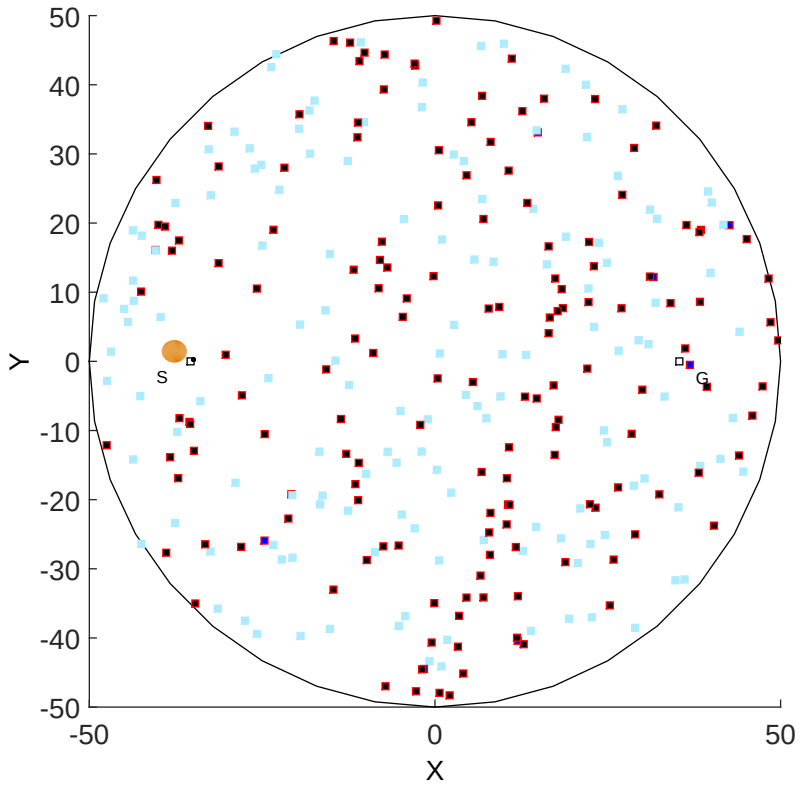


Figure 4.8: Example of the environment with 600 obstacles. The orange circle represents the robots location. S is the start, G is the goal, dark red boxes are line obstacles, and light blue boxes are arc obstacles.

ment. Red squares represent obstacles in line mode, and blue squares represent obstacles in arc mode. The robot (represented as the orange circle) must navigate from the box labeled 'S' (Start) to the box labeled 'G' (Goal).

The same values are maintained for the model parameters in all experiments. For the obstacles, the stochastic velocities in line mode are $w = \{0.1, 0.2, 0.5, 0.7\}$, with corresponding probabilities $p(w) = \{0.3, 0.2, 0.3, 0.2\}$, and in the three arc modes, the stochastic angular velocities are $w = 5\bar{w}$, $w = 10\bar{w}$, $w = 15\bar{w}$, with $\bar{w} = \{\frac{1.0811}{2\pi} \frac{1.6216}{2\pi} \frac{2.4324}{2\pi} \frac{3.2432}{2\pi}\}$, for arcs of radius 5, 10, and 15, respectively, with corresponding probabilities $p(w) = \{0.2, 0.2, 0.3, 0.3\}$. The likelihood of switching

parameters are driven by R_{line} = fraction of line obstacles, n_S = the time that that a given obstacle last switched, and S the switching–time–parameterṪhe collision distance is determined by the obstacle dimensions (it is presume that the robot has no width or height), and hence $\epsilon = 1$. The time step for the experiments is $\delta = 0.1$ and for the stochastic reachable set calculations is $\Delta = 1$. The distance around the robot in which obstacles will affect the selection of the APF gradient is $d_{min} = 3$. To implement the 3D stochastic reachable set calculations, the relative heading is discretized in increments of $\frac{\pi}{20}$ and the corresponding planar stochastic reachable set that is closest to the current value of relative heading is used.

The method is compared to other published methods that address moving obstacles: 1) a traditional Gaussian method, [70], with two parameterizations: $\mathcal{N}(0, 0.15^2)$ and $\mathcal{N}(0, 0.45^2)$, and 2) ORCA [110]. The Gaussian methods wrap a Gaussian potential field around moving obstacles. Two different standard deviation values are selected to demonstrate the impact of increasing the safety margin around obstacles, but at the expense of making some paths infeasible due to the large repulsion area. ORCA was designed for multiple robot collision avoidance and works by computing a an avoidance vector based on the current state of other agents in the environment. It also assumes that all obstacles are also attempting to avoid collision (which is not the case in this environment). However, it is one of the leading multi-robot avoidance methodologies.

4.5.2 Stochastic Reachable Set Approximation

The correct methodology for hybrid dynamic obstacles is to consider the hybrid switching in the stochastic reachability set calculation. However, this method requires complete knowledge of all possible dynamics any given obstacle can have. In practice this may not be possible, and an online path planning system will need

to learn new dynamics. An approximate but scalable and efficient solution is to construct a catalog of previously learned dynamics and match observed obstacle motion to an entry in the catalog. Thus, in Figure 4.9 the correct stochastic reachability set method is compared to the approximate stochastic reachability set method.

The effect of SR sets that use obstacle dynamics with and without switching is evaluated. Figure 4.9 shows how the success rate is affected by the switching rate of the hybrid dynamic obstacles, averaged over 100 runs for each rate. The obstacle switching rate is controlled by S, the switching–time–parameter in Equation (4.23).

Using the individual stochastic reachability set approximation means that the path planning algorithm has poor information about the probability of collision immediately before a hybrid obstacle switches dynamics. For example, if an obstacle is following line dynamics at time t and then switches to arc dynamics at time $t + \delta$, then APF-SR at time t anticipates a zero probability of collision along what will become the arc trajectory. Thus, at time $t + \delta$ the obstacle is now instantly traveling along an arc trajectory and the decisions made at time t (with a line trajectory) do not accurately reflect the motion of the obstacle at time $t + \delta$ (with an arc trajectory). The instant before switching is a potential source of failure for the algorithm.

To demonstrate that the individual SR set method is a good approximation, Figure 4.9 shows the success rate for the combined SR set (in magenta) and the individual set method (in blue). These two plots show that the individual method and the combined method achieve approximately the same success rate for every switching–time–parameter setting.

4.5.3 Method and Environmental Parameter Evaluation

In this set of experiments, the method and the environmental parameters are explored in detail. The two environmental parameters that affect APF methods are the ratio between line and arc obstacles, and the rate at which hybrid obstacles switch between different continuous states (controlled by the switching–time–parameter). While, the method parameter is the σ used to smooth the SR set. The value of σ, the obstacle ratio and the time scale are varied. All parameter evaluation experiments are run in an environment with 300 obstacles, a *goal–vector* magnitude of 0.01, and with a holonomic robot. First, Figure 4.10 shows a plot of the success rate vs. size of smoothing–Gaussian. The best performance occurs at $\sigma = 0.15$. Thus, this value for σ will be used for smoothing all SR sets for the remaining experiments.

The ratio of line obstacles to arc obstacles is also investigated. Figure 4.11 shows an experiment with 300 obstacles, a *goal–vector* magnitude of 0.01 and a variable ratio between obstacle types. Unlike the other experiments, the obstacles are not allowed to switch dynamics; instead the ratio of obstacles types are varied from 0 to 100% in a given run. For APF-SR, the success rate is approximately constant around 95% regardless of the line to arc ratio. It is important to note that the possible radii of the arc obstacles were chosen such that the difference between the line trajectory and the arc is large. Unlike APF-SR, the success rate for the Gaussian method ($\sigma = 0.15$) is reduced by 6% with 100% arcs, and the success rate for the Gaussian method ($\sigma = 0.45$) is reduced 34% with 100% arcs. Similarly, ORCA hovers around 80% success rate for all the switching–time–parameter values (15% less than APFSR), which indicates that ORCA is affected more by the number of obstacles than by their trajectories.

The rate at which obstacles switch dynamics impacts the planning space com-

plexity, with faster switching obstacles being more complex. Figure 4.9 shows that APF-SR is practically unaffected by the rate at which obstacles switch dynamics. APF-SR is unaffected because it rarely moves in front of obstacles due to the construction of the repulsive field. Thus, APF-SR will rarely be in the switching region when the obstacle switches dynamics. Figure 4.9 also shows that the Gaussian methods, which do not take into account the trajectories of the obstacles, are heavily affected by a rapid switching rate. They lose at least 10% success rate for the fastest switching rate compared to the slowest switching rate. However, APF-SR is only minimally (3%) affected by the obstacle switching rate, which indicates the state switching instant has little effect on the overall performance. Thus, the method is capable of successfully planning with hybrid dynamic obstacles regardless of their switching rate.

Finally, the *goal–vector* magnitude is the last tunable parameter. Of all the parameters, *goal–vector* showed no conclusive best value. As such all remaining experiments are shown with several *goal–vector* values. Constant magnitude vectors of $\{0.1, 0.01, 0.001\}$ are used for the *goal–vector* parameter. However, an additional test is also run with the *goal–vector* magnitude set to 0.01 when all obstacles are more than d_min (3) units away and set to 0 when at least one obstacle is closer than d_min units away. This test allows the algorithm to attempt maximal avoidance when obstacles are nearby without being drawn towards any goal.

In summary, these parameter experiments found that smoothing–Gaussian σ value of 0.15 provides the best SR set to be used as a repulsive potential field. APF-SR is unaffected by the number or type of obstacles in the environment, so this value is allowed to change based on Equation (4.23). Similarly, it is agnostic to the rate at which obstacles switch dynamics, but the comparison methods are highly sensitive to the rate. Thus, a switching–time–parameter of $S = 20$ is selected for the remaining experiments. The best value for *goal–vector* is dependent upon

the environment and all remaining experiments will show a variety of *goal–vector* magnitudes.

Algorithm 4.7 calcControl

Input: APF_{vector}, The current robot state ($u_{n-1}, heading$), maximum turn rate (α), maximum velocity (β)

Output: u, θ

---

if robot is holonomic **then**
  $u = APF_{vector}$
  $\theta = 0$
else
  $H_{desired} = APF_{vector}.normalize$
  $H_{current} = (cos(heading), sin(heading))$
  if $acos(H_{desired} \cdot H_{current}) \leq \alpha$ **then**
    $u^w = H_{desired}$
  else
    $u^w = H_{current} + (cos(\alpha), sin(\alpha))$
  end if
  $u^s = \beta$
  $u = (u^s, u^w)$
  $\theta = u^w$
end if

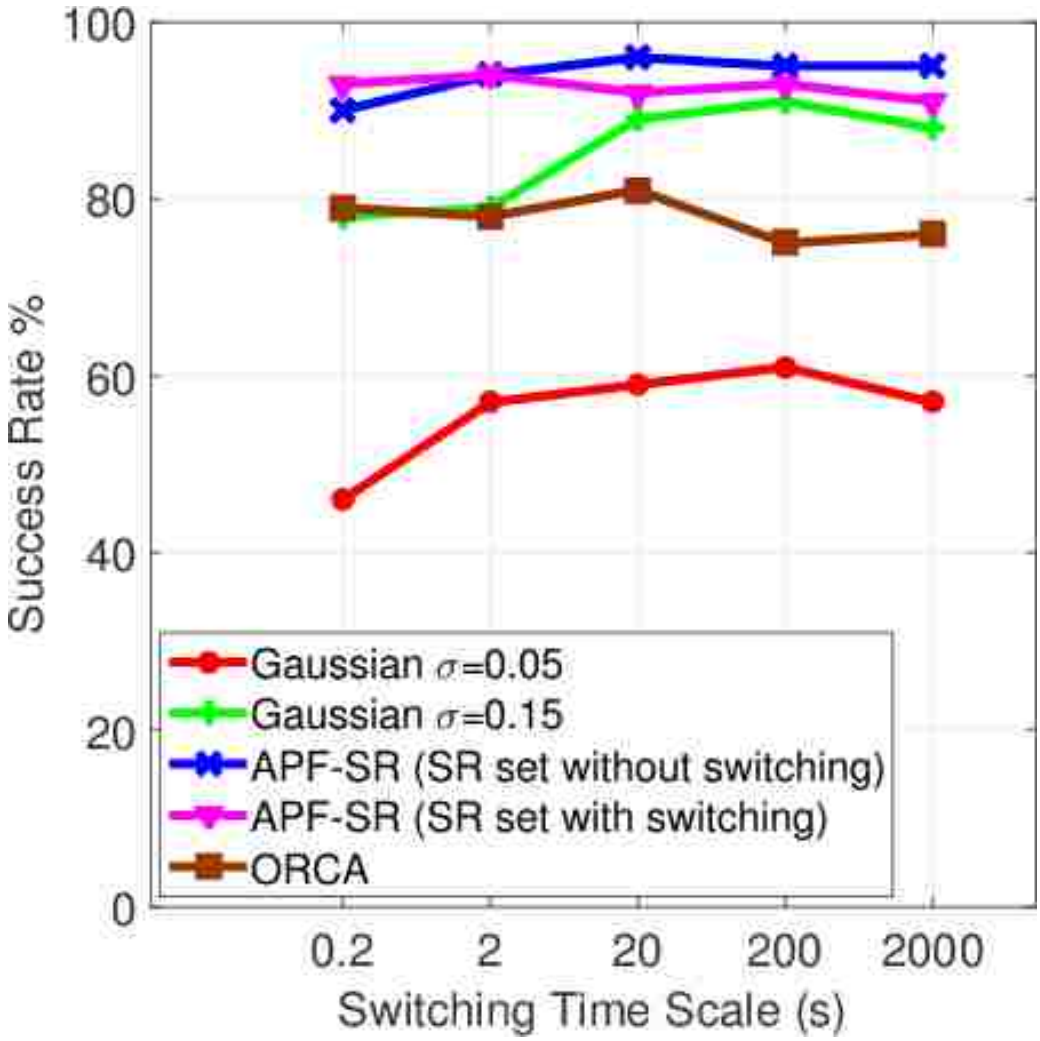


Figure 4.9: Success rate for the APF-SR, and Gaussians $\sigma = 0.15$ and $\sigma = 0.45$ vs. the switching–time–parameter (used in Equation (4.23) to determine switching probability). The smaller the switching–time–parameter value the faster the obstacles switch between a line and arc (or vice versa).

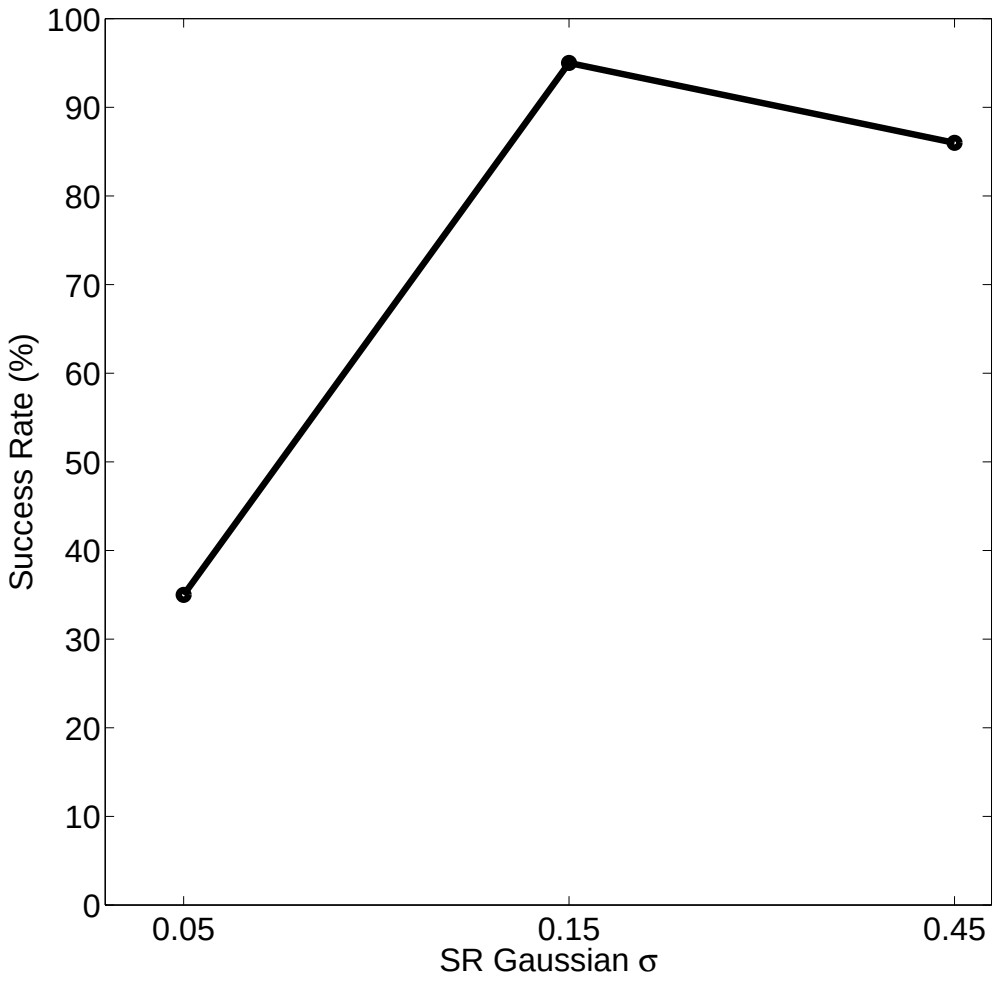


Figure 4.10: Success rate vs. σ used for smoothing the SR set. (With a *goal–vector* magnitude of $g = 0.01$)

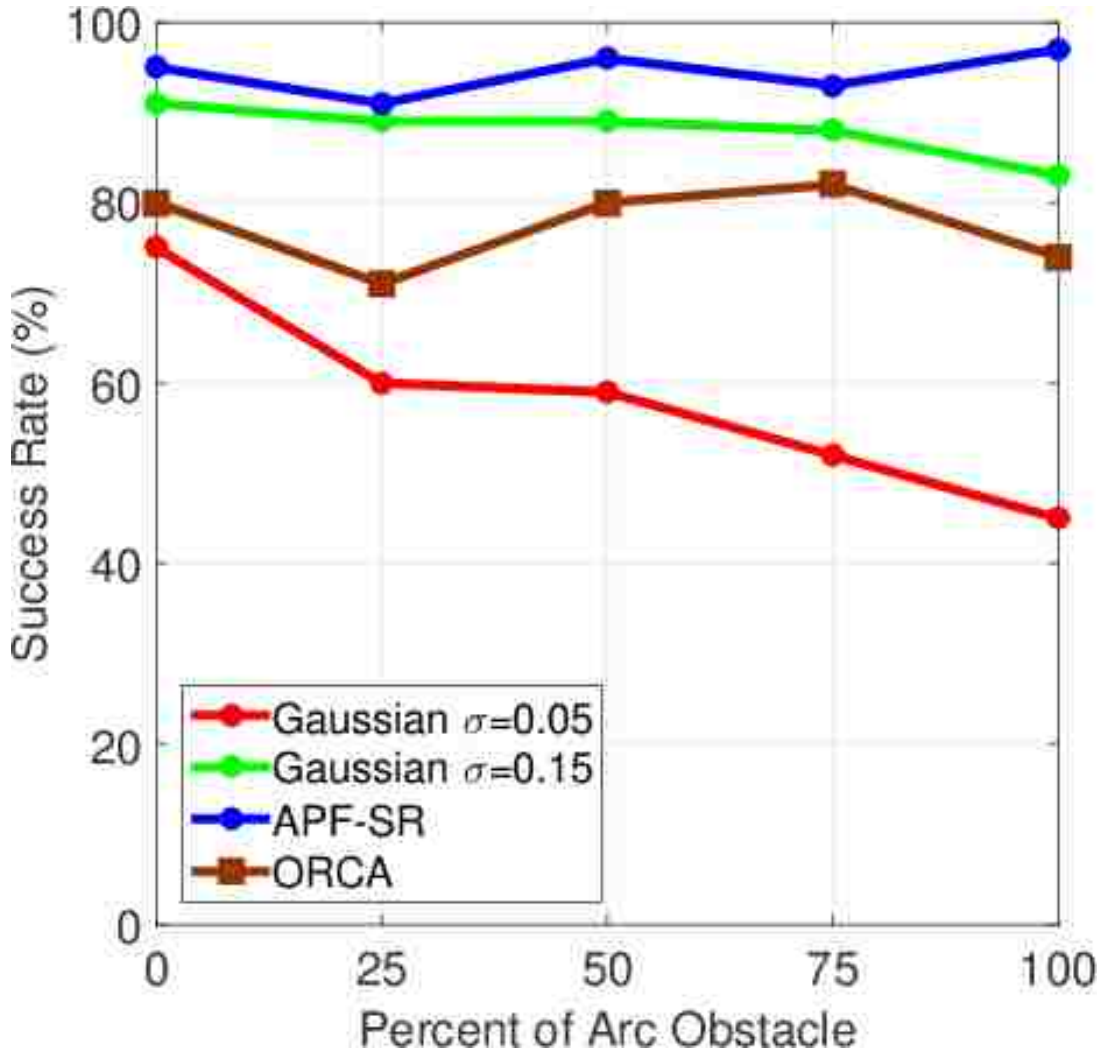


Figure 4.11: Success rate vs. the percentage of obstacles moving in an arc trajectory, with a *goal–vector* magnitude of $g = 0.01$ and 300 obstacles.

4.5.4 Holonomic Robot Experiments

In this experiment a holonomic robot must navigate across a planning space. Figure 4.12 demonstrates the success rate versus number of obstacles (300 to 900) for varying *goal–vector* magnitudes. The *goal–vector* magnitude is important because it affects how strongly the robot is attracted towards the goal. If the *goal–vector* magnitude is too strong the robot will not effectively avoid obstacles, but if it is too small it may not reach the goal. APF-SR (4.12a) and the two Gaussian comparison methods (4.12b and 4.12c) show that the *goal–vector* has a consistent effect across all three methods with the *goal–vector* magnitude of 0.01 providing on average the best overall success rate regardless of the number of obstacles. This indicates that there is an optimal *goal–vector* for APF-SR. A smaller *goal–vector* magnitude provides better success rates because the influence of the obstacle repulsion is higher and thus the robot is more reactive to the moving obstacles. Second, APF-SR does better than either of the Gaussian method parameterizations and ORCA. The slopes in figure 4.12 are approximately the same compared within each method. This indicates that the complexity of the problem increases linearly with the number of obstacles. APF-SR has a similar slope to the Gaussian $\sigma = 0.15$, but for 300 obstacles the success rate is higher (95% compared to 60% for a *goal–vector* magnitude of 0.01). Interestingly, the slope for Gaussian $\sigma = 0.45$ is steeper than the other methods, but it has a success rate of 89% for 300 obstacles, which indicates that the greater repulsion region aids in path planning for sparse environments but prevents the robot from navigating in cluttered environments. Finally, OCRA has a success rate of 80% to 45% (20% less than APF-SR for 900 obstacles). Thus, APF-SR is impacted less by the number of obstacles than the comparison methods, because its structured potential field allows for more informed path planning.

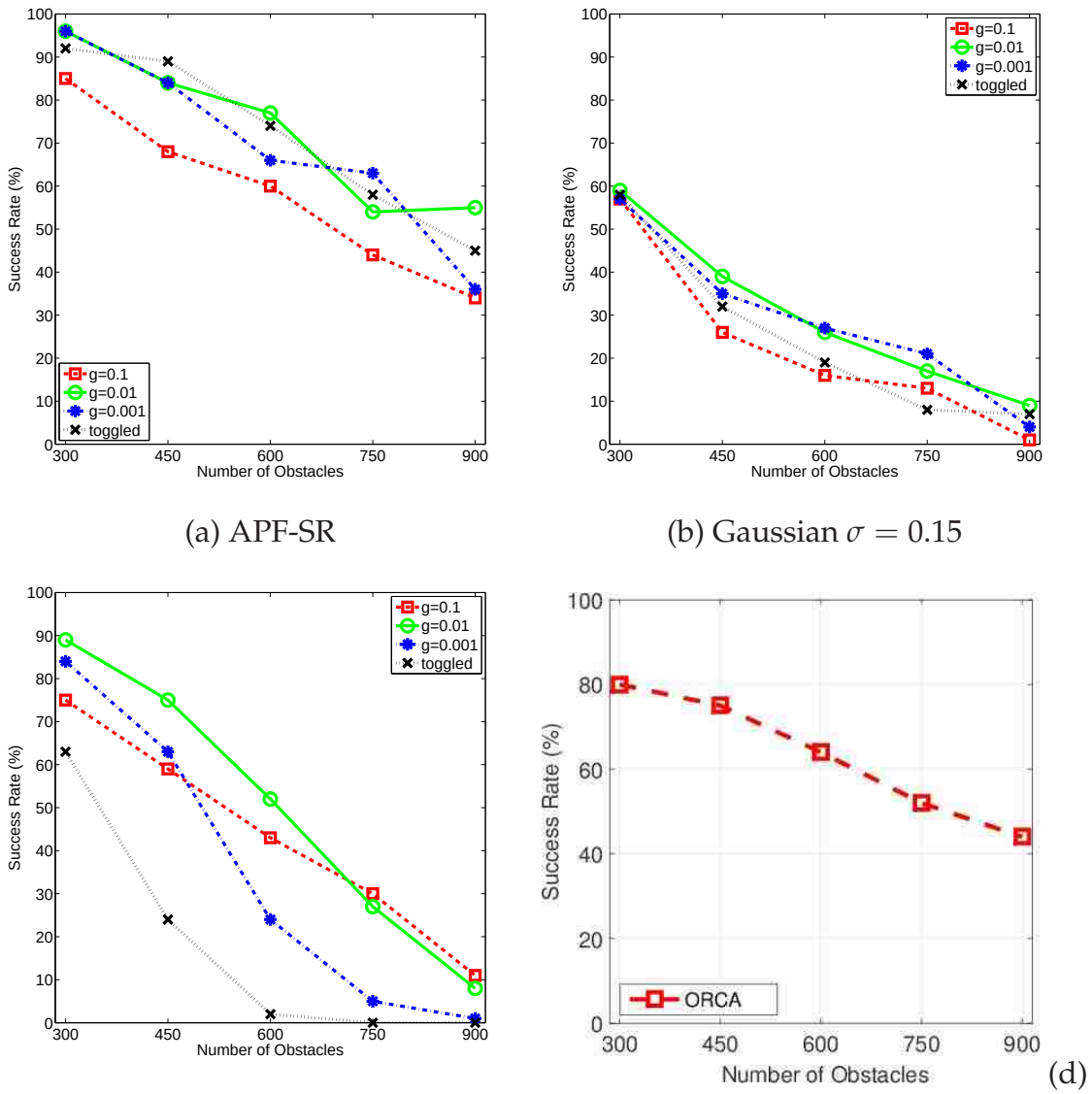
Total path length is affected by how much the robot is forced to deviate from the path because of the obstacles. Figure 4.13 shows the average path length ver-

sus the number of obstacle for APF-SR and the two comparison methods (only recorded for successful runs). Figure 4.13 shows that, as expected, the path length increases as the *goal–vector* gets smaller. This indicates that if the *goal–vector* is too strong the robot does not react enough to the obstacles' potential fields, but if the *goal–vector* is turned off in the presence of obstacles (the black dashed line) the robot does not make enough progress towards the goal and spends too much time in the obstacle field which increases the likelihood of collision. This trend holds for APF-SR and the two Gaussian comparisons, however its scaled relative to the potential fields used. The Gaussian ($\sigma = 0.45$) has the widest field and as such is affected the most by the obstacles. The Gaussian ($\sigma = 0.15$) is more similar in size to APF-SR but it does not consider the obstacle trajectory, thus its path lengths are similar but its success rate is lower. The shaped potential field of APF-SR allows the robot to navigate around obstacles in a safe manner by avoiding entering the obstacle trajectory (like the large Gaussian method) but still have a relatively small field which allows it to move between dense obstacle clusters (like the Gaussian $\sigma = 0.15$).

Finally, which situations cause APF-SR to fail are investigated. Figure 4.14 shows a histogram of the number of nearby obstacles when each method fails. Of the eight recorded collisions during the 100 trials, one of the collisions was due to a single obstacle. The rest of the collision were due to multiple obstacle interactions. Note that APF-SR is based on SR sets, which provides a probability of collision for areas inside the set. Thus, the robot does not receive any repulsion information until it is already inside the set. This means that once the robot is under an obstacle's repulsive force, there exists a probability of collision. Also, since the SR sets are only computed for single obstacle interactions, there is no guarantee of success when planning for more than one nearby obstacle. Figure 4.14 confirms this, as the APF-SR method fails most often when multiple obstacles are nearby. These obstacle create situations with conflicting APF gradients that

cause the APF-SR method to collide.

In contrast, the Gaussian $\sigma = 0.15$ in Figure 4.14 collides mostly with single obstacles. Whereas, the larger Gaussian ($\sigma = 0.45$) collides about evenly with all types of interactions. This is because the wider Gaussian provides a larger buffer between the robot and single obstacles but it still does not make the best possible planning decisions, as it does not take into account the relative robot-obstacle interaction.



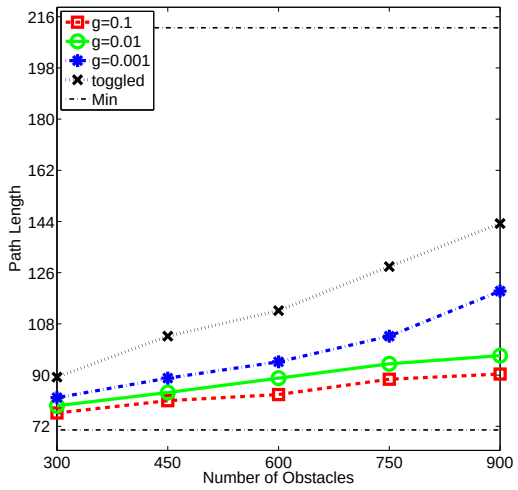
(a) APF-SR

(b) Gaussian $\sigma = 0.15$

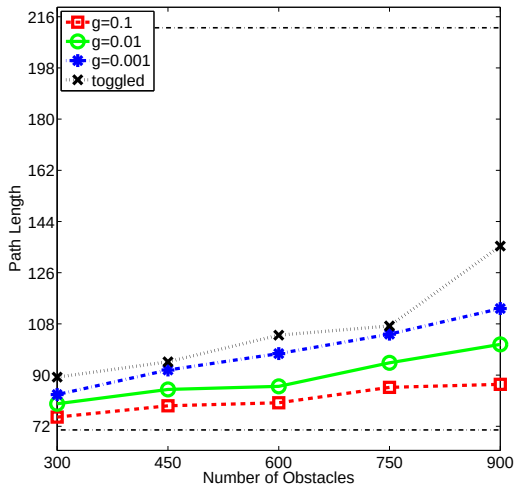
(c) Gaussian $\sigma = 0.45$

(d) ORCA

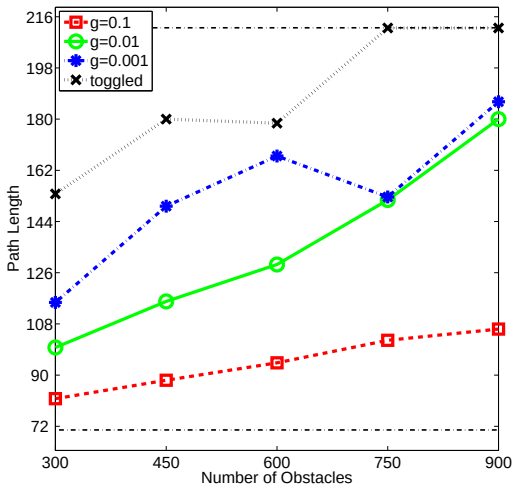
Figure 4.12: Holonomic robot success rate for (a) APF-SR, (b) Gaussian $\sigma = .15$ (c) Gaussian $\sigma = 0.45$ and (d) ORCA. g is the *goal–vector* .magnitude g = toggled indicates, that the *goal–vector* is set to 0 when the robot is under the influence of an obstacle's APF and set to 0.01 when it is not being influenced by any obstacles.



(a) APF-SR



(b) Gaussian $\sigma = 0.15$



(c) Gaussian $\sigma = 0.45$

Figure 4.13: Holonomic robot path length for (a) APF-SR, (b) Gaussian $\sigma = .15$ and (c) Gaussian $\sigma = 0.45$. The dashed line at Path Length 72 indicates the theoretical shortest path possible (ie. straight line from the start to the goal). The dashed line at path length 200 indicates a cutoff point where the run is considered a failure. g is the *goal–vector*. $g =$ toggled indicates, that the *goal–vector* is set to 0 when the robot is under the influence of an obstacle's APF and set to 0.01 when it is not being influenced by any obstacles.

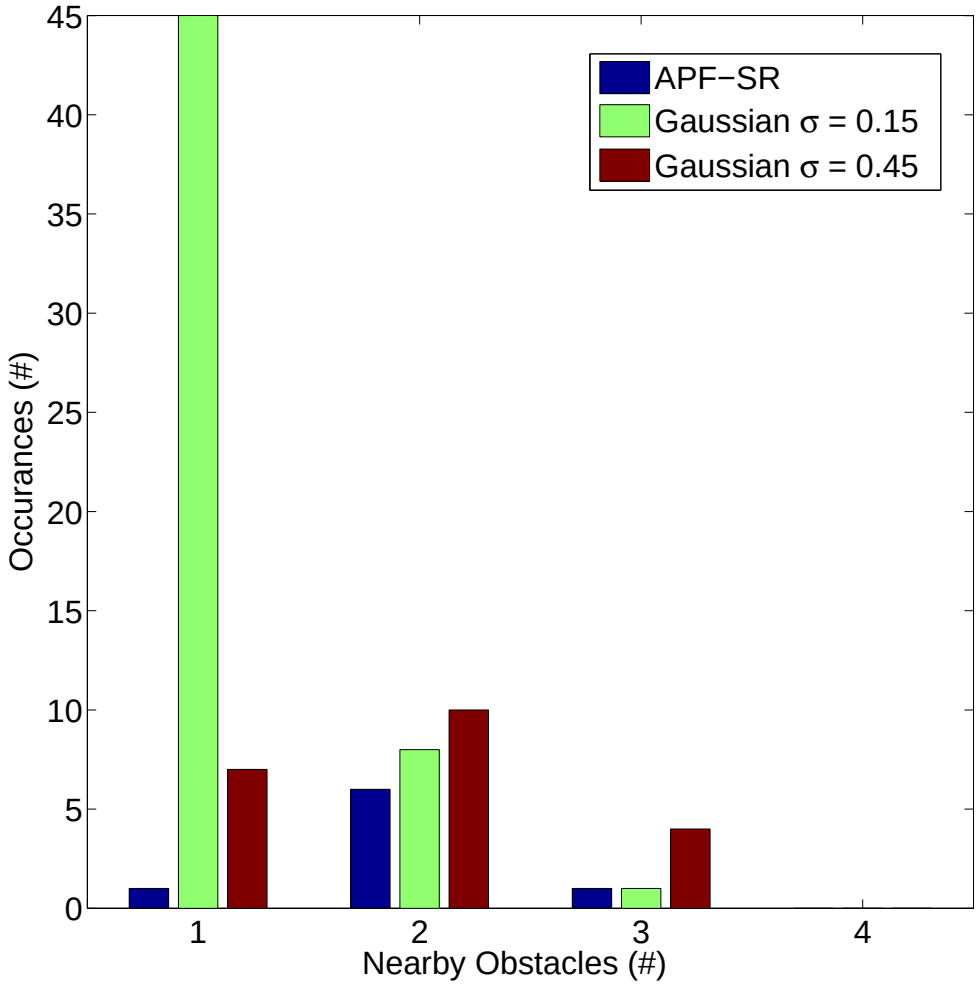


Figure 4.14: Histogram showing the number of nearby obstacle when the APF-SR method, Gaussian $\sigma = 0.15$ and Gaussian $\sigma = 0.45$ fail due to a collision. An obstacle is nearby if the distance between the robot and the obstacle is less than 3 units.

4.5.5 Unicycle Robot Experiments

In this last set of experiments, the robot's dynamics are changed from holonomic to a non-holonomic unicycle. This increases the difficulty of the problem as the robot cannot instantly change heading to avoid a collision.

The robot is limited to a turning rate of $\frac{\pi}{12}$ per Δ. Figure 4.15 shows that APF-SR and the Gaussian comparison methods all have lower success rates than the holonomic case. However, the Gaussian methods suffer much more than the APF-SR method. The best Gaussian success rate (for 300 obstacles) went from 90% in the holonomic case to 62% in the unicycle case but APF-SR went from 95% to 84% success rate. Furthermore, as the number of obstacles increase the Gaussian methods quickly approach 0% success rate. Note, that ORCA cannot be directly applied to non-holonomic robots without significant modifications, thus an ORCA comparison is not shown.

For the holonomic case a *goal–vector* magnitude of 0.01 was on average better than any other goal attraction. However, for the unicycle case, success rates for a *goal–vector* magnitude of 0.01 and 0.1 oscillate. This is likely due to the increased difficulty of the problem which greatly increases the probability of collision the longer the robot is in the environment. Furthermore, the success rate slope for APF-SR is the steepest. While this indicates that APF-SR's success degrades faster with increasing number of obstacles, the success rate is still higher than the Gaussian comparison methods.

Figure 4.16 shows the path length for the unicycle robot versus the number of obstacles. Again, as expected the path length increases as the *goal–vector* decreases, and the path length increases as the number of obstacles increases. Interestingly, in almost every case, the comparison methods became lost, whereas none of the APF-SR trials became lost. This trend combined with the success rate

indicates that APF-SR is producing plans which not only avoid the obstacles but still make progress towards the goal, where the Gaussian methods may avoid the obstacles but are not able to make progress towards the goal.

These experiments have shown that APF-SR is able to path plan in environments that have up to 900 hybrid dynamic moving obstacles with a very high success rate. Furthermore, APF-SR is significantly more robust to the hybrid dynamics than the comparison methods and the increased success is due to encoding the relative obstacle robot dynamics in the SR set used to produce the potential fields for the obstacles. Thus, APF-SR is able to make more informed path planning decisions and more easily avoids moving obstacles

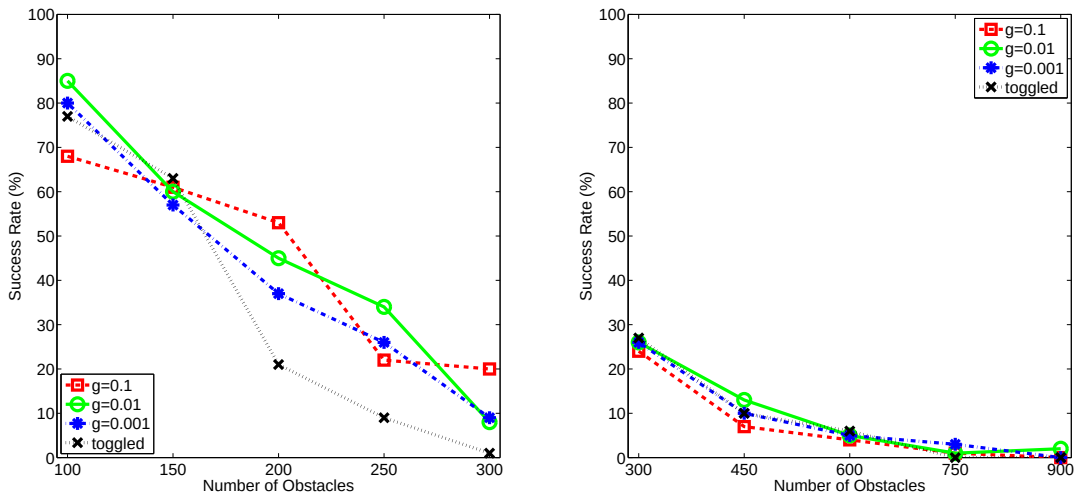
While these results demonstrate that APF-SR outperforms comparable methods, there are two key limitations. First, the point-mass robot model is a simplification of actual robot motion. However, methods such as [33] and [25] exist, which extend APF methods to non-point robots. A more realistic robot model can be easily incorporated into the SR set calculation, but with additional computational cost. Second, note that the SR set must be recalculated if the obstacle dynamics within a continuous state change. One solution is to maintain a SR set database and to then match obstacle motion to sets as [60] does with funnel libraries. Another solution is to use a method similar to [27] and learn to predict moving obstacle motions. Neither of these limitations are insurmountable, and these results maintain that the improved performance of APF-SR as compared to other approaches merits its use in many scenarios.

The *goal–vector* magnitude is the final important consideration. The results presented here indicate that the magnitude must be tuned to the problem as no particular value was always dominant. However, this is not an insurmountable task and the difference in success rates between with different *goal–vector* magnitudes was minor.

4.6 Conclusion

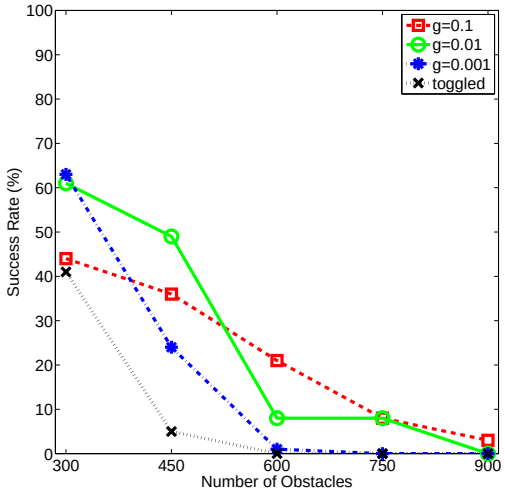
The incorporation of the formal SR sets into the ad-hoc APF method produces a more accurate representation of the relative robot-obstacle dynamics, which leads to an increased success rate during path planning. APF-SR has a success rate at least 30% higher than other methods used for comparison, even with 900 obstacles. The SR set informs the APF-SR algorithm of the direction and velocity of the obstacle, which is used to generate a repulsive potential that reflects the probability of collision. Hence the APF-SR algorithm can make informed planning decisions in the presence of multiple moving obstacles. Here, it is shown that APF-SR is robust to the primary parameters in the method, and demonstrated that the method is capable of path planning in highly complex and dynamic environments with obstacles that can switch dynamics from line to arc or arc to line.

Here, SR sets were combined with planning method and shown on a small range of environments. However, the method can be generalized to other environments and other obstacle-robot dynamics as the SR set method is not dependant on the dynamics used here.



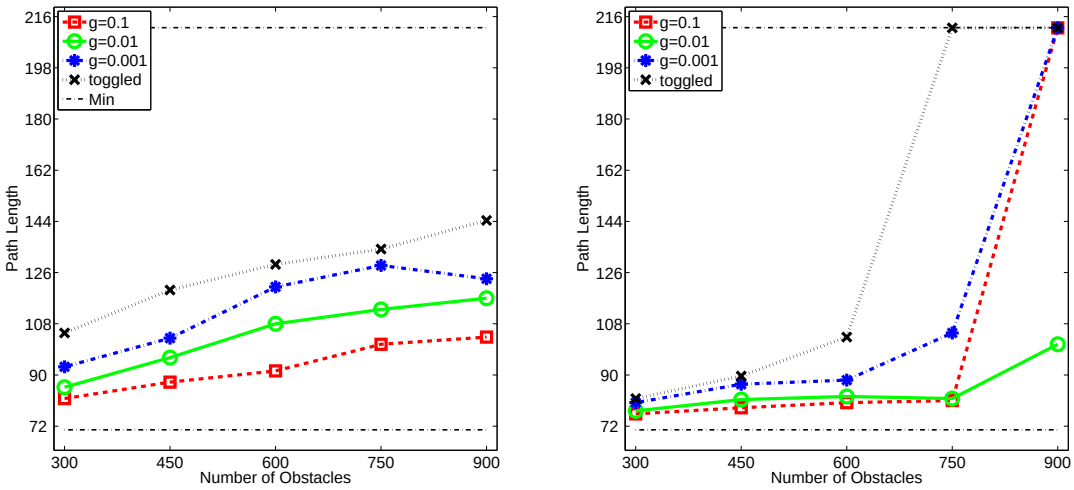
(a) APF-SR

(b) Gaussian $\sigma = 0.15$

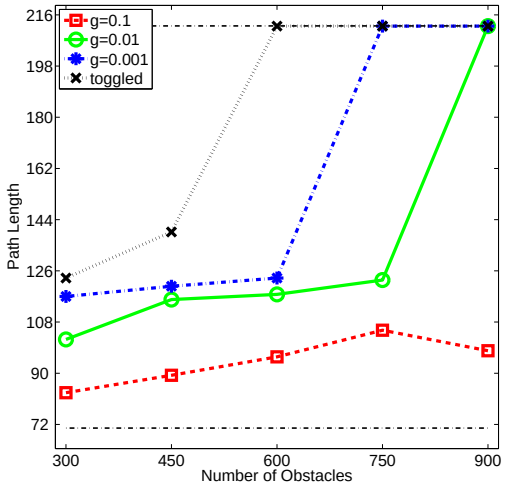


(c) Gaussian $\sigma = 0.45$

Figure 4.15: Unicycle robot success rate for (a) APF-SR, (b) Gaussian $\sigma = .15$ and (c) Gaussian $\sigma = 0.45$ g is the attractive strength towards the goal. $g =$ toggled indicates that *goal–vector* is set to 0 when the robot is under the influence of an obstacle's APF and set to 0.01 when it is not being influenced by any obstacles.



Unicycle APF-SR (a) APF-SR (b) Gaussian $\sigma = 0.15$



(c) Gaussian $\sigma = 0.45$

Figure 4.16: Unicycle robot path length for (a) APF-SR, (b) Gaussian $\sigma = .15$ and (c) Gaussian $\sigma = 0.45$. g is the attractive strength towards the goal. g = toggled indicates that *goal–vector* is set to 0 when the robot is under the influence of an obstacle's APF and set to 0.01 when it is not being influenced by any obstacles.

Chapter 5

Transition Function Uncertainty: Integrated Planning and Learning

Transition function uncertainty exists when the mapping from action to state is unknown by the path planning algorithm. This is a particularly challenging problem as the only method of determining this mapping is through exploration. Many situations would cause the transition function to be unknown, ranging from changing environments to damaged sensors and motors. Thus, a framework must be designed which can intelligently explore the transition function. This Chapter presents a framework based on the work in [68] and [62].

In order to perform tasks, robots must be able to adapt to changing environments and problems. In order to process real world information, online planning has to process higher volumes of data with tighter deadlines at every time step. The planning is subject to hardware imperfections and errors in reading

Figure 5.1: Whole Arm Manipulator (WAM).

sensory information. Online reinforcement learning (ORL), a machine learning technique, is a useful tool for robotics motion learning and planning. It provides a closed-loop feedback system continuously incorporating current environment information into the planning and producing the motions required to perform a task. However, online reinforcement learning comes with several challenges that make it potentially problematic to use on a physical system.

Implementation of an ORL algorithm must be carefully designed to be safe for the robot both in terms of collision avoidance and producing motions that don't strain hardware. Training the ORL agent from scratch on physical hardware can cause wear and tear to the hardware and thus change the dynamics of the system. Furthermore, motions take longer time to execute on hardware than in simulation, and the training phase could become impractically lengthy. Lastly, because the state-space grows exponentially with the number of degrees of freedom, the sheer size of real world state-spaces and physical laws of motion that need to be processed at every time step in real-time could make ORL prohibitively computationally expensive even for serial link manipulators with as little as 3 DoFs [63].

Reinforcement learning (RL) learns action (motion) sequences that maximize accumulated reward over the agent's lifetime. RL learns a policy, a mapping be-

tween robot's states and its actions with respect to some observed, and unknown to the agent, reward signal. The outcomes of the action-taking, transitions between the states, are also unknown *a priori* and are learned through experience. The RL problem is defined by specific state and action spaces, the ability to observe action effects on the states, and the reward associated with states. To accomplish task learning with RL, the reward structure that corresponds to the task, the state-space that corresponds to the possible robot configurations, and the actions space that corresponds to possible robot motions all need to be engineered.

Two major classes of RL methods are available, online and offline. Offline methods analyze and derive policy from an experience batch. Online RL, on the other hand, derives policy in an ongoing manner. It improves and changes the policy with every step. The advantage of the online RL is that it naturally adapts to the changing environment, something offline RL is not capable of doing. The adaptation comes at the price of longer convergence times to threshold performance. Being more computationally expensive, online RL is potentially prohibitive for systems with high degrees of freedom. In this Chapter, the slow convergence time of online RL is addressed with dimensionality reduction using PRMs for improved scalability, and with learning transfer by training first in simulation and moving goals.

Consolidating from previous work [63, 68], this Chapter presents a framework based on ORL that successfully overcomes the challenges above and learns motion-based tasks suitable for a physical robot. To jump-start the learning on hardware, and avoid a lengthy training phase, knowledge is transfered from an agent trained in simulation. To achieve performance suitable for a physical system, ensure the safety of the system, and address state-space scalability, probabilistic roadmaps (PRM) are used for dimensionality reduction. The state-space information reduced by the PRM is passed to the learning agent, which learns

to produce efficient motion plans. Here, a Brain-Emulating Cognition and Control Architecture (BECCA) [88] agent is used. It is an adaptive online reinforcement learning algorithm paired with an unsupervised hierarchical feature creator. BECCA's algorithm contains a decay feature, allowing the agent to forget features and motion plans over time. This feature is especially useful for changing environments, as the agent continuously learns and updates plans based on the current feedback from the environment.

To demonstrate the framework, a pointing task on a 7 DoF WAM (Figure 5.1) using all 7 degrees of freedom is implemented. The robot needs to autonomously learn how to point at a target location in its environment regardless of the start position. The task is first formulated in terms of RL, and four sets of experiments are performed. First, the learning scalability with and without the PRM dimensionality reduction as a function of degrees of freedom is compared. In the second series of the experiments, learning transfer impact is assessed on a stationary target. The performance of the framework is assessed by measuring how well the agent adapts to hardware imperfections and measurement noise. In the third series of experiments the target location moves and the adaptability is evaluated. Lastly, the performance of the framework is examined by looking into time savings obtained by using transfer learning.

The results show that the system task performance does not change with increase in dimensions, and shows near-identical performance between simulation and transferred hardware runs. Results show between 100 to 600 time steps of savings obtained by using transfer learning, and demonstrate an agile agent that quickly adapts to the new environment within 500 time steps.

The rest of this Chapter is organized as follows: Section 5.1 provides necessary background. Section 5.2 discusses the methodology, and section 5.3 presents the experimental results. Finally, section 5.4 concludes the Chapter with the frame-

work's benefits to online, reactive motion-based learning.

5.1 Preliminaries

The robot hardware used in this Chapter is the same as that used in Chapter 3.5. However, in this Chapter a learning agent is used to learn motion controls to complete a task.

5.1.1 BECCA

Creating a general learning machine has been one of the grand goals of artificial intelligence (AI) since the field was born. Efforts to achieve this goal may be divided into two categories. The first category uses a depth first approach, solving problems that are complex, yet limited in scope, such as playing chess. The assumption underlying these efforts is that an effective solution to one problem may eventually be generalized to solve a broad set of problems. The second category emphasizes breadth over depth, solving large classes of simple problems. The assumption underlying these efforts is that a general solution to simple problems may be scaled up to address more complex ones. An example of the first category would be a master level chess playing agent, while an example of the second category would be an agent with the capabilities of an bee worker.

The work described here falls into the second category, focusing on breadth. The motivating goal for this work is to find a solution to natural world interaction, the problem of navigating, manipulating, and interacting with arbitrary physical environments to achieve arbitrary goals. In this context, environment refers both to the physical embodiment of the agent and to its surroundings, which may include humans and other embodied agents. The agent design presented here is

loosely based on the structure and function of the human brain and is referred to optimistically as a Brain-Emulating Cognition and Control Architecture (BECCA) [88], [89].

A Brain-Emulating Cognition and Control Architecture agent interacts with the world by taking in actions, making observations, and receiving reward (see Figure 5.2). Formulated in this way, natural world interaction is a general reinforcement learning problem [99], and BECCA is a potential solution. Specifically, at each discrete time step, it performs three functions:

1. reads in an observation, a vector $o \in \Re^m \mid 0 \leq o_i \leq 1$.
2. receives a reward, a scalar $r \in \Re \mid -\infty \leq r \leq \infty$.
3. outputs an action, a vector $a \in \Re^n \mid 0 \leq a_i \leq 1$.

Because BECCA is intended for use in a wide variety of environments and tasks, it makes very few assumptions about the environment beforehand. Although it is a model-based learner, it must learn an appropriate model through experience. There are two key algorithms to do this: an unsupervised feature creation algorithm and a tabular model construction algorithm.

The feature creator component identifies repeated patterns in the input vector [88]. It then groups loosely correlated elements of the input vector. The groups are treated as subspaces and unit vectors of these subspaces are features [88]. New inputs are also projected onto existing features and the single feature in each group which has the greatest response is turned on while all others in that group are turned off [87, 88, 90].

The reinforcement learning component receives feature activity, reward, and direct input from the environment. Each feature is associated with an approximate

Figure 5.2: At each timestep, the BECCA agent completes one iteration of the sensing-learning-planning-acting loop, consisting of six major steps: 1) Reading in observations and reward, 2) Updating feature set, 3) Expressing observations in terms of features, 4) Predicting likely outcomes based on an internal model, 5) Selecting an action based on the expected reward of action options, and 6) Updating the model.

reward. It keeps track of recent actions and recent features in working memory which is then used to update the model. The actual model is a table of cause-effect pairs. The cause is the working memory and the effect is the current feature. Considering this in standard reinforcement learning language, the model can be thought of as a sequence of state-action pairs. Entries in the table which are rarely observed are deleted from the model [87, 88, 90].

To chose an action the reinforcement learner compares the current working memory to the entries in the model and selects the entry which both matches the current working memory and which has the highest recorded reward. With a set probability, an exploratory action is chosen instead [87, 88, 90].

In the context of traditional Markov Decision Process (MDP)-based reinforcement learning, the cause-effect pairs are equivalent to action-state pairs. The cause-effect table with the working memory and its expected reward roughly

corresponds to a Q-function in traditional MDP-based reinforcement learning. However, BECCA's model does not assume the Markovian property and might depend on more than one previous state. As time progresses, less frequently observed cause-effect transitions are removed from the cause-effect table. This makes BECCA inherently able to adapt to new situations and environments at the cost of a steeper learning curve. The learning curve is steeper because it depends on more than the current state and because BECCA could potentially remove critical states that are rarely observed.

5.1.2 Transfer Learning

Transfer learning typically refers to utilizing information learned in the past on a task in the present [101]. This past learning can be transferred to a new task or to the same task under different constraints. Transfer learning has also been utilized in transferring knowledge from one robot to another robot that may have a different internal architecture to represent the world [101]. Taylor and Stone [101] define jump-start and time-to-threshold performance as two metrics for transfer learning. Jump-start defines the amount of gain an agent initially receives from transferred knowledge. Time-to-threshold performance defines the amount of time it takes an agent to reach the threshold performance, which is the best the agent can do at a given task.

5.2 Methods

Here a framework is presented for online motion-based task learning. Figure 5.3 shows the framework's main components. Task definition describes process of constructing the reward structure, and state-action space encoding to describe the

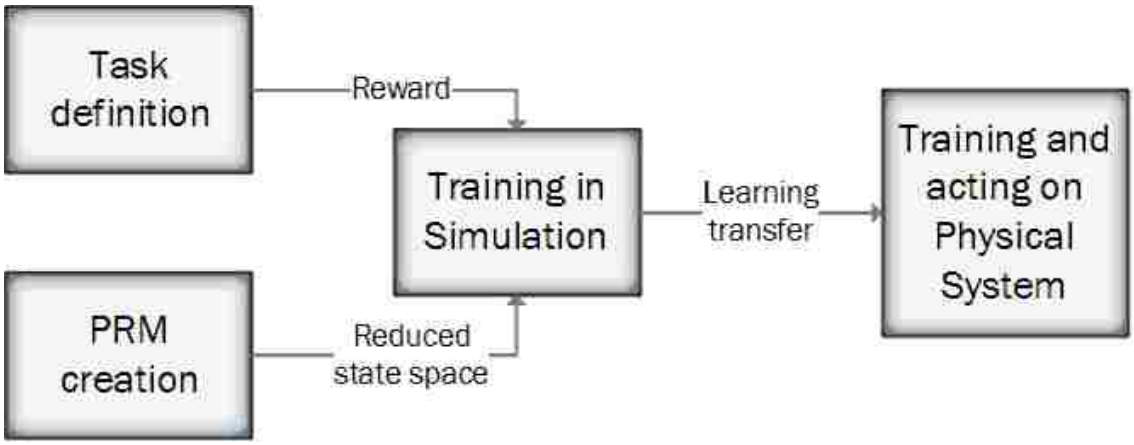


Figure 5.3: Task learning Framework

task. PRM creation segment generates a roadmap for a given environment and physical system. With the roadmap constructed and task encoded, the BECCA agent is deployed on a simulated system. Once the simulated agent's performance meets the satisfactory criteria, the entire agent is transferred to the physical system for ongoing task performing.

5.2.1 Probabilistic Roadmaps Creation

This Chapter uses PRMs combined with learning agent techniques to build a roadmap for the reinforcement learning agent to navigate by randomly sampling joint positions. The vertices in the roadmap are connected to k nearest neighbors using a straight line local planner. The learning agent's state-space is reduced to roadmap vertices, and actions are limited to edges between the vertices. The agent (robot) is constrained to making straight line movements along the edges in the adjacency matrix, thus constraining the reinforcement learner to learn how to navigate the roadmap. During a transfer, the previously learned roadmap is preserved. The PRM is the underlying state-space provided to the learning agent.

5.2.2 Task definition

While BECCA is mostly automated, an engineer must design a task to interface with BECCA via sending sensory vectors and interpreting action vectors. Such an interface is called a task. A task simply defines what information from the world will be sent to the agent, and in what format. Note that BECCA is agnostic to the task semantics. The task also defines how to read an action vector and move the robotic actuators. Again, note that BECCA is agnostic to how this is defined, and it will learn whatever format the engineer devises. To demonstrate the framework, two pointing tasks are defined and their setup explained.

Task with Stationary Target

The sensory vector is a n element binary vector, since the PRM contains n vertices. Each vertex represents a feasible, collision-free configuration of the robotic arm. When the robot is at a particular configuration the corresponding element in the sensory vector is set to 1.

Algorithm 5.8 shows how the pointing task is constructed. The action vector is a k element long binary vector and is parsed by the *interpret* function. In this task, BECCA has been constrained to only return a single 1 in the action vector.

The *interpret* function in Algorithm 5.8 does the following: The 1 in the action vector represents BECCA selecting to move to one of the k neighbors, and the $(k+1)^{th}$ element is interpreted as staying at the current configuration. For example the action vector $[0, 1, 0, 0]$ is interpreted by the task as selection to move to the second neighbor of the current configuration in the roadmap. The function then returns the configuration of the selected neighbor.

The reward structure for the PRM task assigns a reward of 100 to the target

vertex, a reward of 10 to all neighbors of the target vertex, and a reward of 1 to the neighbors of the neighbors. Every other vertex is given a reward of 0.

Pointing Task with Non-stationary Target

The formulation and the setup of the non-stationary target task is the same as in Section 5.2.2. The reinforcement learner is trained on an initial pointing task and then transferred to hardware, however upon being transferred the goal state is changed. Thus, the learning agent must compensate for the changed goal, while learning to adapt to the dynamics of the hardware system. Specifically, for this task the goal state is moved to one of the neighbors in the roadmap of the simulation goal state. The reward structure is changed so that the new goal state is reward 100 and the neighbors of the new goal 10 and the neighbors of the neighbors 0.1.

Algorithm 5.8 Task Step

Require: Task

1: $Task.agent.action = [0, 0, 0, 0]$
2: **while** *not coverging* **do**
3: $newLocation \leftarrow$ interpret($task.agent.action$)
4: $sendToWAM(newLocation)$
5: $task.currentPosition \leftarrow$ read current WAM location
6: $task.SensoryInput \leftarrow task.currentPosition$
7: $task.reward \leftarrow task.calculateReward()$
8: $task.agent \leftarrow agent_{step}(SensoryInput, Reward)$;
9: **end while**

5.2.3 Transfer Learning from Simulation to Hardware

Taylor and Stone define a taxonomy of transfer learning in the reinforcement learning domain in [101]. Using that terminology, the source task is a simulated pointing task. There are two target tasks. In one, the target task has the same goal and algorithm in both simulation and hardware runs. In the other the target is moved but it still has the same algorithm. The transferred knowledge is a set of feature groups and a cause-effect pairs.

The method transfers learned knowledge of a single task between a perfect simulation of a robot to imperfect robotic hardware. In simulation the robot always receives the exact same joint angles for a particular state, but in hardware the joint angles are subject to small error so re-entering the same state will not have the exact same state information. The source task uses the same learning agent, parameters, and reward function as the target task. The only difference is that the source task interacts with the WAM simulator while the target task interacts with the WAM hardware.

When performing the transfer, the entire agent is transferred with all its internal states and accumulated experience. Only the world model, which it interacts with from the simulator or the WAM interface, is changed.

5.2.4 WAM Simulator

The WAM simulator is a simple kinetic simulator, representing the arm with seven points each corresponding to one degree of freedom. The arm moves in the simulator by simply adding the state and action vectors. The simulator does not inject noise, and performs perfect movements. The WAM arm, on the other hand, performs the movements as described in the Section 5.2.5. The resulting motion is

subject to error in performing the movement.

5.2.5 WAM Interface

The WAM is connected to a xPC Target Kernel running Matlab Simulink 7.7.0 R2008b [71]. The controller for the WAM is written in Simulink and interfaces with remote computers via the reflective memory network. The Simulink code responsible for directly issuing commands to the WAM, henceforth the WAM controller, receives a command vector by reading a specific block of reflective memory. The command vector is a length seven vector containing the desired joint angles in radians of each for the seven WAM joints.

The WAM controller, upon receiving a command vector, places the command vector into a buffer, which only stores one move. The command vector is first sanitized so that each entry is within the WAM's joint limits. If the WAM is not executing a move, it compares its current location to the command vector buffer. If the command vector buffer is sufficiently different from the current location, the WAM controller computes a linear interpolation in joint space between the two joint angles and executes the path within the allowable WAM workspace. Where significantly different is $|norm(\hat{v_{desired}} - \hat{v_{current}})| > .01$ However, the velocity follows a fifth-order smooth polynomial as seen in Fig 5.4, and is used both for safety and for mimicking biological motion [28]. Slow beginnings and endings to moves provide safe joint torques. In the current architecture a move cannot be interrupted.

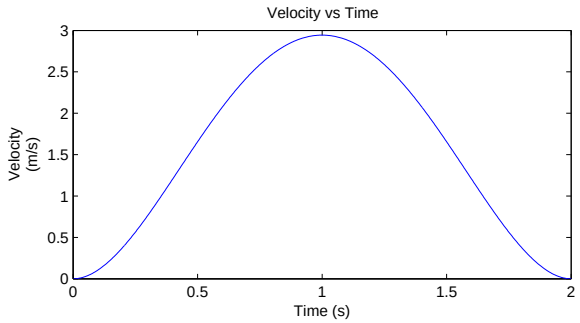


Figure 5.4: Example Velocity Profile for a Single Joint.

5.3 Experiments

There are four experiments. Two experiments involve the stationary target pointing task, one is a non-stationary pointing task, and the last evaluation assesses the training time benefits of the framework. First, the utility of the dimensionality reduction using PRMs over standard state-space binning is examined. Then, the benefits of transfer learning, including performance adaptability, is examined. All experimental results are averaged over five executions. Throughout the experiments, the performance on the learning agent is measured via cumulative reward. When the learning agent is transitioned from simulation to physical hardware, it is placed in a configuration that is as far as possible from the goal configuration.

The performance of the learning agent on hardware is compared to performance in simulation. The agent executes in time steps but the graphs are shown in blocks, where 1 block equals 100 time steps. The time savings brought on by using transfer learning, and the initial boost of performance that was obtained by knowledge transfer are evaluated. In case of the non-stationary task, the time it takes the agent to react to a change in environment and recover to the previous level of performance are presented.

Each experimental run is executed on a new roadmap of 50 configurations

generated using PRMs. Each configuration is connected to 3 neighbors and itself. A random point in the 50 configurations is chosen as the goal. The goal vertex is given a reward of 100. the neighbors of the neighbors are given a reward of 0.1. All other configurations are given a reward of 0.

5.3.1 Dimensionality reduction utility

This Section evaluates learning scalability when the state-space is reduced to PRM vertices. The goal of incorporating PRMs is to help guide the searching. The state-space dimensionality increases exponentially with DoFs. In [68], it was shown that BECCA can learn up to 3 DoF pointing tasks with binned state-space representation. Beyond that, the learning takes impractically long, and the results are affected. Incorporating PRMs allows us to reduce the state-space for 3-DoF tasks to the number of vertices in the roadmap. For the results shown, the state-space has 50 vertices, but the number of vertices can be adjusted.

The learning performance of the PRM based task is compared with two variants of the stationary pointing task (Section 5.2.2). The task is reduced to 3 DoFs by limiting the WAM to use only three joints.

Joints 1, 2, and 3, are mapped into a 3 dimensional C-*space*. Then, fifty random points are sampled in the C-*space* using a uniform distribution. The fifty points are then connected probabilistically based on the distance between the points, such that closer points have a higher probability of being connected. Figure 5.5 shows an example of a PRM generated for the 3-DoF task.

Figure 5.6 shows the cumulative reward per block for BECCA operating on the 3-DoF PRM task. The maximum reward that can be received per iteration is 100, making the maximum per block 10,000 units of reward. The PRM covers a wide area in the WAM's range of motion, but only takes 900 iterations to reach a

very high cumulative reward. 900 iterations is significantly fewer than the 5,000 iterations required for the 2-DoF task to converge [68], which indicates that PRM's are very effective at reducing the convergence time of BECCA.

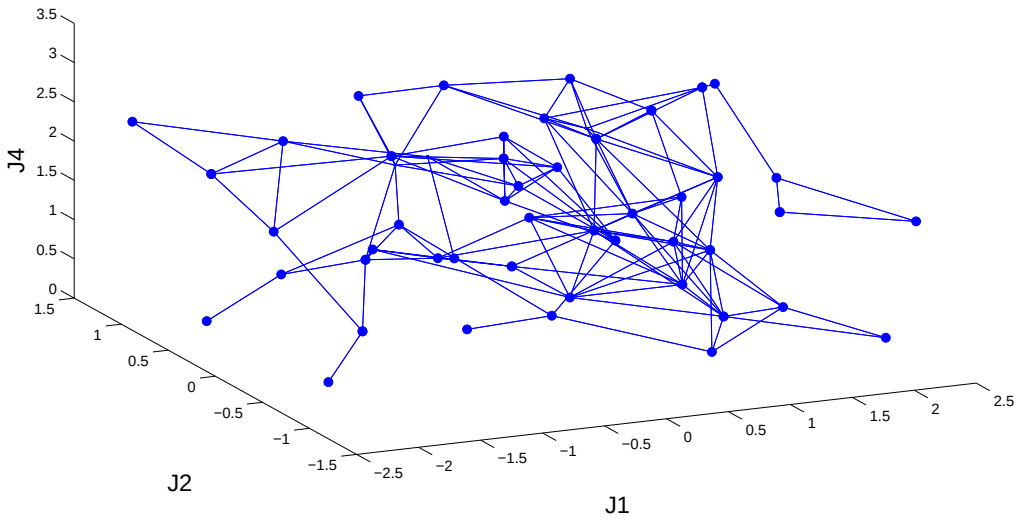


Figure 5.5: Probabilistic Roadmap for a 3-DoF WAM Task. Vertices are possible configurations. Edges are possible transitions between configurations.

To compare the PRM based task to non-PRM tasks, two 3-DoF tasks are created, a Simple and a Hard task. The Simple 3-DoF task has 3 bins per joint, and an action vector of length 12. The Hard 3-DoF task has 4 bins per joint, and an action vector of length 18.

The reward structure for both, the Simple and Hard tasks, parallel to the PRM task. It has a maximum reward of 100 per iteration and thus 10,000 per block. The Simple task has 27 possible states. The Hard task has 64 possible states and the PRM has 50 states. Thus, the Simple and Hard tasks frame the PRM in number of states. However, it is important to note that the Simple and Hard tasks have larger action vectors than the PRM task, 12 and 18 actions vs. 4.

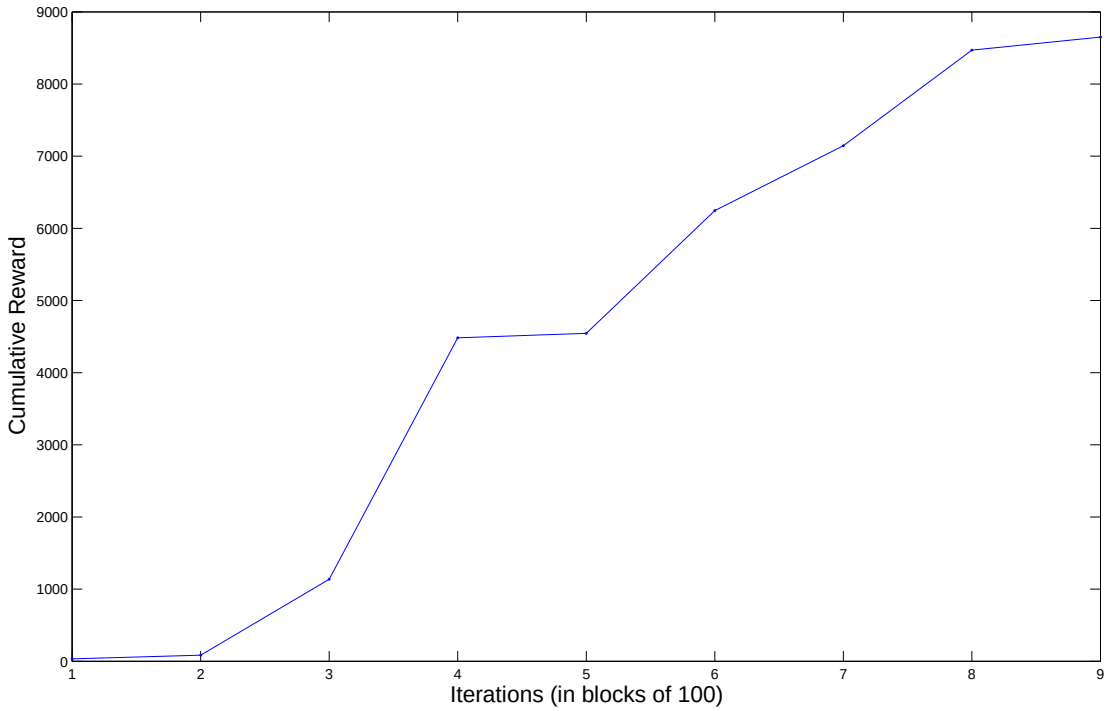


Figure 5.6: The Cumulative reward that the learning agent achieves per blocks of 100 iterations using the PRM state-space.

Figure 5.7 shows that the PRM method converges much faster than either the Simple 3-DoF or Hard 3-DoF task. The PRM method has reached the optimal of 7,000 units of reward by around 1,000 iterations while, the Simple 3-DoF task has only reached approximately 6,000 units of reward by 7,000 iterations. The 3-DoF Hard task has only reached approximately 3,500 by 7,000 iterations. Thus it can be seen that the PRM task converges much faster than either the Simple or Hard task.

Figure 5.8 which plots the average reward of 10 runs for each DoF from 1 to 7 further shows the scalability of the PRM approach. This graph confirms that PRM-BECCA is unaffected by the Degrees of Freedom with a constant number of vertices. However, there is a problem with just testing the Degrees of Freedom and holding the number of vertices constant. By holding the number of vertices

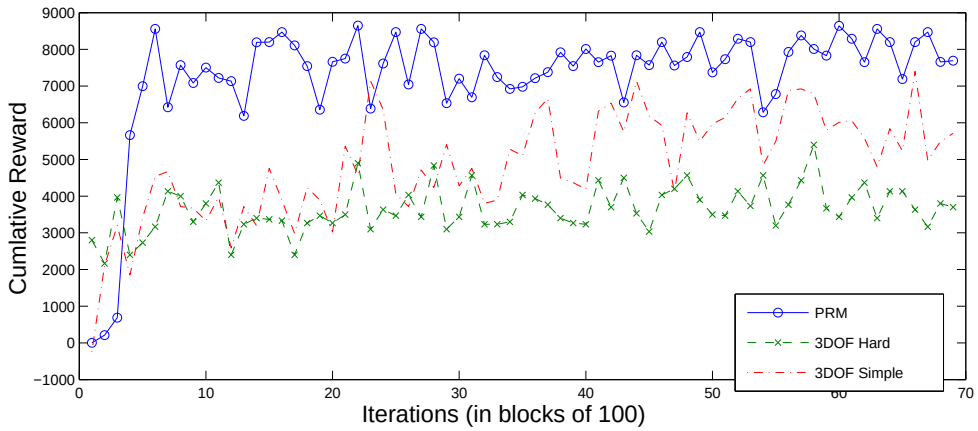


Figure 5.7: Cumulative reward for PRM, 3-DoF Simple, and 3-DoF Hard tasks. The 3-DoF Simple task has 3 bins per joint, giving a state-space of 3^3. The 3-DoF Hard task has 4 bins per and an action vector length of 18, giving a state-space of 4^3. The PRM task has 50 points which correspond to 50 states.

in the PRM constant, the density of vertices decreases as the DoF increases. Thus, BECCA must also be investigated with a varying number of vertices to see how BECCA scales with the number of vertices.

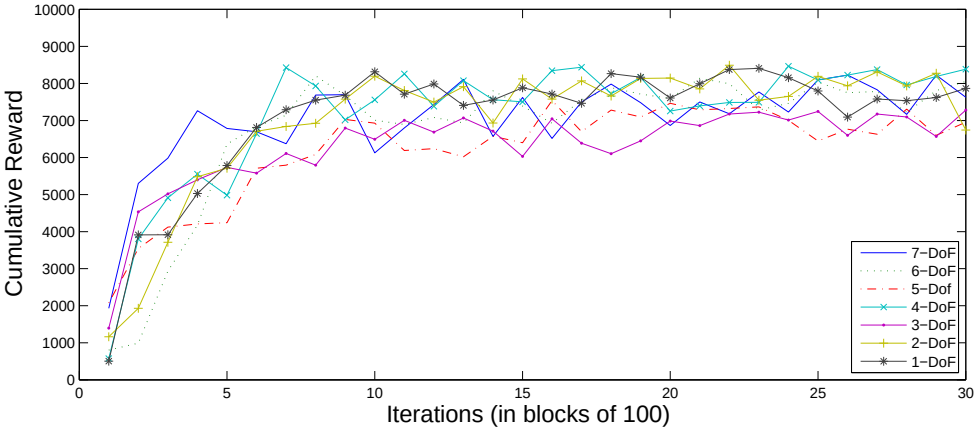


Figure 5.8: Cumulative Reward per Block of 1-DoF to 7-DoF with PRMs.

In the following experiments the number of vertices are varied from 60 to 200 in steps of 20, and the k neighbor parameter is set to 4. Since the previous experi-

ment showed that BECCA would converge at the same number of steps regardless of DoF , 3 DoF was chosen. Again 10 runs are done for each number of vertices, and the results are averaged. Figure 5.9 shows the average cumulative reward for each test. It shows that BECCA may converges at the same time regardless of number of vertices in the graph. Figure 5.9 is very similar to Figure 5.8, thus showing that BECCA converges at the same rate regardless of DoF and regardless of the number of vertices in the PRM.

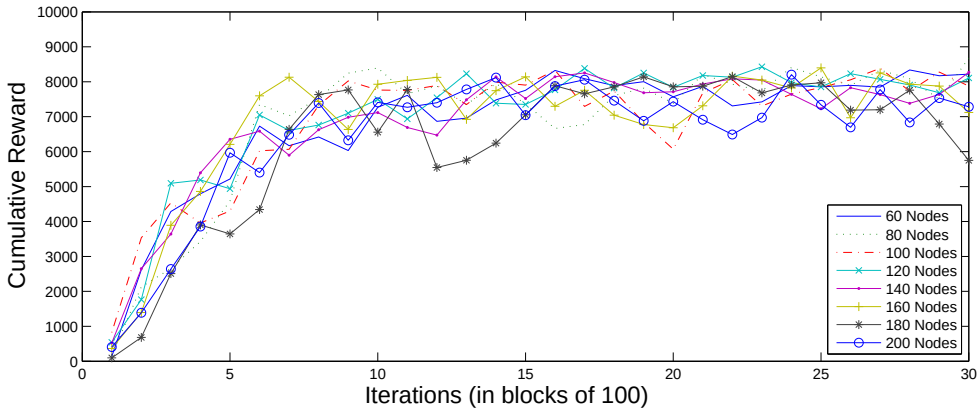


Figure 5.9: Reward per Block for Varying Number of Vertices.

It is important to note that this is a novel use of PRMs. In previous work, they have been used to plan the motions for complex robot systems [36, 37, 80]. However, by integrating PRMs with BECCA, automatic learning of controls can be achieved in complex problems.

5.3.2 Transfer Learning on Pointing Task with Stationary Target

This Section assesses the effect of the transfer learning to the system performance. Learning is first done in simulation and then the entire task is transferred to the physical system (Section 5.2.3).

Table 5.1: Average cumulative rewards in simulation and on hardware after the stabilization for 7DoF task with a stationary target and 7DoF task with a non-stationary target. Note that reward is unitless.

Task	Simulation	Hardware
Stationary Target	7460.3	7614.8
Nonstationary Target	7460.3	7491.5

Figure 5.10 shows the cumulative reward of the pointing task with the stationary target in simulation and on hardware. The vertical line indicates the transition from the simulation to the hardware. The results show near-seamless transition, and the average performance of the agent on hardware very close to the performance in the simulation.

Table 5.1 shows the average cumulative reward for each experiment after stabilization, before and after transition to the physical hardware. Stabilization in simulation occurs at 20 blocks. The performance of the agent on the hardware outperforms the agent in simulation by 154 units of reward.

To better demonstrate the advantages of using the transfer learning in the framework, the pointing task with stationary target experiments were run again in a different manner. Five completely untrained learning agents were run on hardware for 20 blocks and the results averaged together. Then five agents which were trained for 100 blocks in a simulation were run on hardware for 20 more blocks and averaged together. Figure 5.11 shows the comparison of the stationary pointing task using transfer to the same task without using transfer. The advantages of using transfer are seen primarily in the jump-start and the time to threshold metrics. Table 5.2 shows the transfer metrics for the three experiments. Jump-start shows the immediate gain from using the transfer. The pointing task starts very close to the threshold performance using the transfer and has a jump-start gain of 5716. In all random runs, the transferred learning agent outperforms the non-transferred learning agent (Table 5.2). Furthermore, the transferred task reaches

the threshold performance in 2 blocks compared to 7 blocks without transfer (Figure 5.11). It is important to note the time saved by using transfer learning. Table 5.3 shows the run times for simulation versus hardware for 20 blocks. It is clear that simulation is faster by up to 1 hour and 55 minutes. Using transfer learning it takes significantly less physical time on the robotic hardware for the agent to perform the given task as near optimal levels. This not only saves valuable time but it also saves valuable wear and tear on the hardware.

It is important to note that the learning algorithm is not executing pre-planned paths. It learns from experience which paths lead to highest reward and attempts to follow those paths. The agent is learning which actions in a given state will lead to high reward. The paths learned in simulation provide BECCA with a strong foundation to work from, however each execution of the learning problem finds different paths due to the randomness of exploration. Thus, it is possible to witness executions of BECCA on the same underlying roadmap with slightly varying performances.

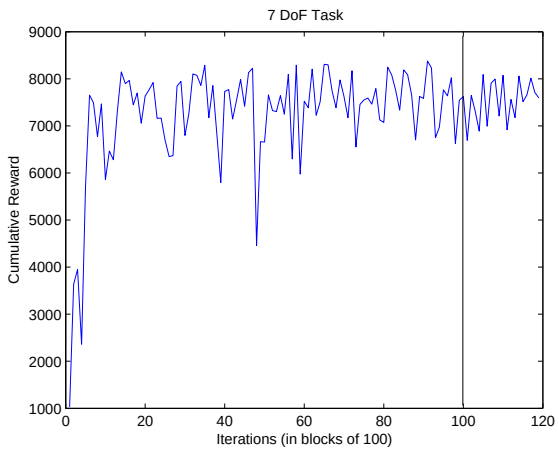


Figure 5.10: Cumulative reward for the pointing task with stationary target per time step. The vertical line indicates where the learning agent was transitioned from simulation to physical hardware.

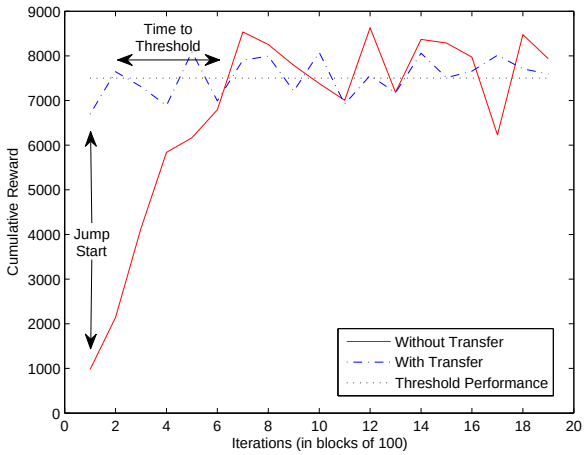


Figure 5.11: Cumulative reward for the pointing task running on hardware with stationary target task with transfer and without transfer per time step. Transfer is when an agent trained in simulation is transferred to hardware. Jump-start shows the initial gain obtained by using the transferred knowledge. Time-to-threshold indicates the time that the task without the transfer needs to achieve the same level of performance as the task with the transfer.

Table 5.2: Transfer Metrics for stationary and non stationary tasks. Jump-start shows the gain from using transfer. Threshold gain shows the reduction in time steps needed to reach the threshold performance.

Task	Metric	Average	min	max
Stationary	Jump Start (reward)	5716	2757	9280
	Threshold Gain (steps)	500	200	700
Non-stationary	Jump Start (reward)	1313	364	1702
	Threshold Gain (steps)	100	100	400

Table 5.3: Average time in minutes to run 20 blocks in simulation and on hardware for 7DoF task with a stationary target and 7DoF task with a non-stationary target.

Task	Simulation (min)	Hardware (min)
Stationary Target	23	122
Non-stationary Target	24	121

5.3.3 Pointing Task with Non-stationary Target

In this experiment the reinforcement learner is trained on an initial pointing task and then transferred to hardware. However, upon being transferred, the goal state is changed. Thus, the learning agent must compensate for the changed environment. The goal state is moved to one of the neighbors in the roadmap of the simulation goal state. The reward structure is changed so that the new goal state is reward 100 and the neighbors of the new goal 10 and the neighbors of the neighbors 0.1.

Figure 5.12 shows the results of 100 blocks of simulation and then 20 blocks of running on hardware where the goal has changed. Initially there is a steep performance drop, but the reward does not drop to zero. The agent quickly recovers and learns the new reward structure within 6 blocks. This shows the online nature of the BECCA algorithm. It is able to first learn one environment and when placed into a slightly different environment it is able to quickly compensate for the change.

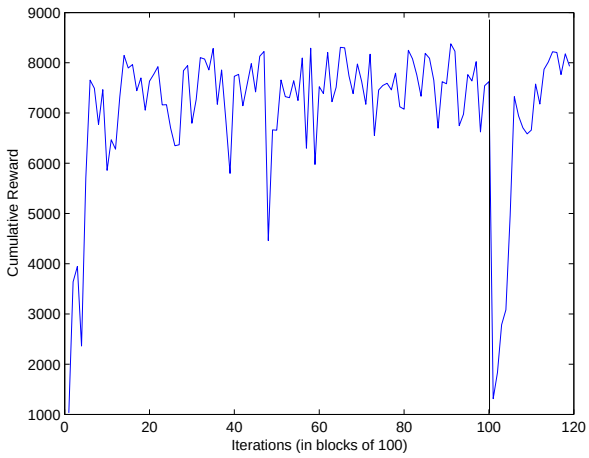


Figure 5.12: Cumulative reward for running in simulation and then transferring the task to hardware. The transfer occurs at 100 blocks.

Figure 5.13 is a comparison between the agent having previously learned a pointing task in a modified environment, to an agent without any prior knowledge. However, the agent with knowledge has learned to point to a different goal in simulation before being run on hardware. The untrained agent is also run on hardware but has a stationary target. Thus, the transferred agent has some information about the structure of the environment but it does not have the exact reward structure as the goal was moved before being placed on physical hardware. The figure shows that the agent with prior knowledge has a small jump-start of 1313 units of reward and reaches the threshold performance 1 block faster than the agent without transferred knowledge.

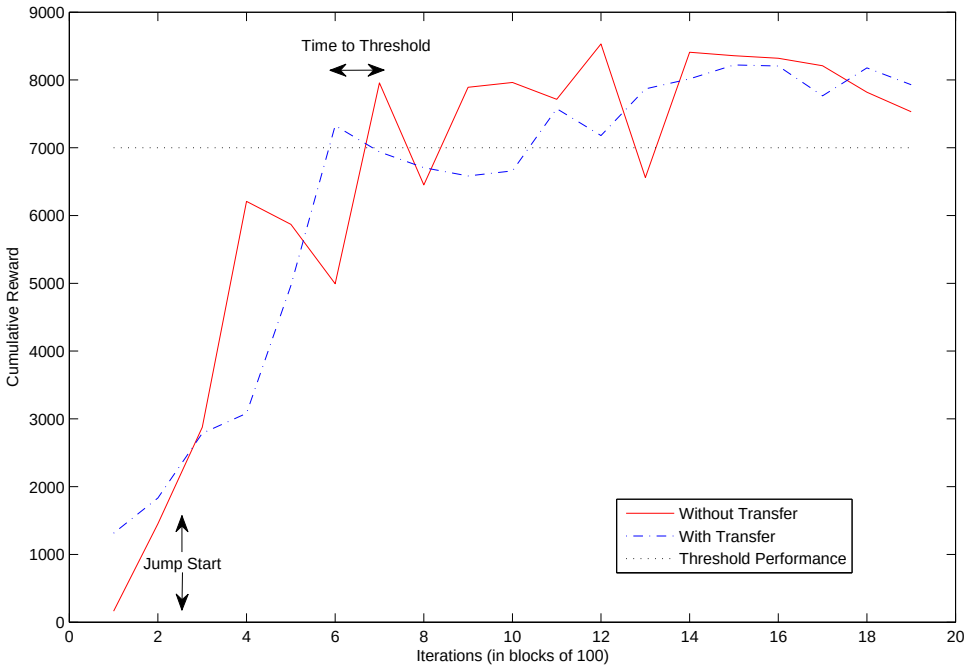


Figure 5.13: Cumulative reward for the pointing task running on hardware with a non-stationary target task with transfer and without transfer per time step. Jump-start shows the initial gain obtained by using the transferred knowledge. Time-to-threshold indicates the time that the task without the transfer needs to achieve the same level of performance as the task with the transfer.

Figure 5.14 shows a variant on the moving target. In this experiment the agent is trained in simulation until convergence to the threshold performance. After convergence in simulation, the agent is moved to physical hardware with an unchanged goal (just like the stationary target experiments). The agent is then allowed to adapt to the hardware for 10 blocks, at which point the goal is moved while still on hardware. The agent must then adapt to this change in hardware. Figure 5.14 shows that the agent does very well with the initial transfer and does better when the goal is moved than Figure 5.12, where the agent is not allowed to adapt to the hardware before the goal is moved. The threshold performance is restored after 6 time-blocks, as previously. But, the minimal reward of 2080 at that time-frame, is higher than the minimal reward of 1313 when the environment is changes right after the task transfer from the simulation to the hardware.

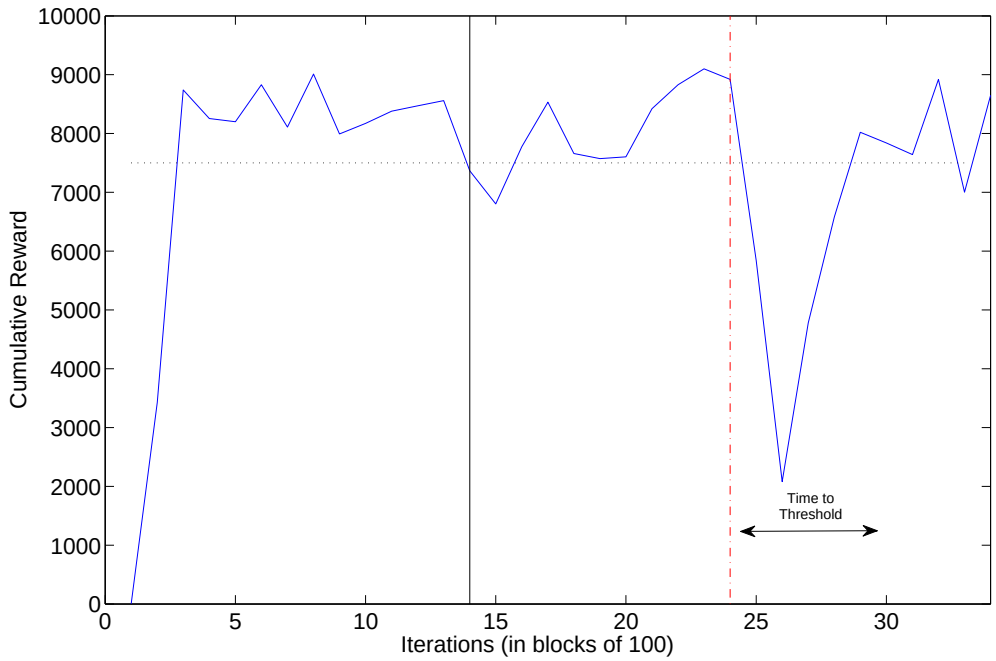


Figure 5.14: Cumulative reward for the pointing task initially trained in simulation then transferred to the robot at the solid black bar. While running on the robot the goal is then changed at the red dashed line.

5.3.4 Timing

Timing data is collected by simply measuring the difference between start time and stop time for runs. Table 5.3 shows the timing data for running the learning algorithm in simulation versus running on physical hardware. The run time on hardware is approximately 5 times longer due to the amount of time it takes to for the arm to move between configurations. Each move on the WAM takes approximately 3.5 seconds to compute and execute. This computation time includes the feature extraction and action decision time for the learning algorithm. In contrast, in simulation it only takes 0.5 seconds of time to execute a complete move.

Since BECCA is an online learning algorithm, it can adapt to changes in real

time. However, because it is an unsupervised learning agent it still requires repeated examples of the new environment.

The amount of time it takes to converge to the threshold performance is the most indicative parameter. This time is important because it represents the amount of time in which the robot is learning instead of performing the desired task. This metric is recorded by simply measuring the difference between the start time of run and the time of each step. Table 5.4 shows the average time for reaching the threshold performance with and without transfer learning. This table shows that transfer learning reduces the learning time by 29 minutes for a stationary target and 8 minutes for a non-stationary target.

For the experiment with both stationary and non-stationary targets, Table 5.4 shows the convergence time after the target is changed for simulations training and without simulation training. This experiment first transfers the simulation to the robot with a station target, and then after stabilizing the target is moved (like the non-stationary test). The version without simulation runs the whole experiment on the physical robot. The times for the recovery are very similar, which is to be expected as they are just showing the recovery from the changed target experiment. At this point both the transfer run and the no-transfer run have the same knowledge and thus exhibit the same amount of recovery time. This demonstrates the online nature of the algorithm. However, the total run-time is very different as the transfer agent does 1400 iterations in simulation (11.44 minutes vs. 58.30 minutes). This is a difference of 47.37 minutes.

Table 5.4: Average Time for Convergence to Threshold Performance.

Task	w/o Transfer (min)	w/ Transfer (min)
Stationary Target	40.8	11.7
Non-stationary Target	41.1	33.0
Task	w/o Transfer (min)	w/ Transfer (min)
Both Targets: recovery	25.9	24.7
Both Targets: total time	129.5	82.1

5.4 Discussion

Here an efficient online motion-based task learning framework based on reinforcement learning is demonstrated. The framework works in high-dimensional spaces in real-time, is reactive to changes in the environment, performs safe hardware motions, and efficiently learns on hardware. The framework is demonstrated by implementing it on a 7 DoF WAM using all joints to produce pointing motions with both stationary and non-stationary targets. The framework is robust and extensible to other robotics systems as well as with different model formulations, and for a large variety of tasks as well.

Dimensionality reduction and collision checks can be handled through PRMs for any motion-based task. When PRMs are used in this manner, they impose hard limits on the system. For example, self-collision states tend to be invariant to the type of environment or the task, and are good candidates to be precomputed ahead of time. When there is error in the model used for simulation caused by noisy sensor data, the robot can explore the validity of the simulation's roadmap and learn how to efficiently navigate in the physical environment.

Transfer learning can be used to avoid early learning phases when the agent's performance tends to be erratic, to reduce wear and tear on robot, and to speed up the learning process on the physical robot. It can be a powerful techniques to mitigate the long convergence times of reinforcement learning. Combining trans-

fer learning, reinforcement learning and probabilistic roadmap methods produces a powerful framework for solving complex robotic tasks. By harnessing each method's strengths, the weaknesses of the other methods can be mitigated.

Chapter 6

Conclusions and Future Work

Uncertainty is a challenging problem that must be carefully considered in the path planning problem. This work has shown that directly incorporating uncertainty into the planning algorithms themselves provides superior solutions in terms of clearance and success rates compared to methods which do not consider uncertainty. This body of research presents solutions for three common types of uncertainty faced in robotic motion planning tasks. First, Safety-PRM provides a method for path planning with an inaccurate workspace model. Second, SR-Query and APF-SR provide techniques for path planning with moving obstacles. Finally, BECCA combined with PRMs demonstrates a technique for path planning with an unknown transition function. These methods all provide a first step in moving robotics from controlled environments to uncontrolled and uncertain real world situations.

This thesis presented three methods and evaluated them under strenuous conditions. For Safety-PRM the amount of error in the model was increased beyond current sensor technology error amounts. SR-Query and APF-SR had the number of obstacles increased until the algorithm was no longer able to reliably produce

successful plans. Finally, BECCA combined with PRMs was stress tested by increasing the number of DoF and changing the goal in the middle of a run. These, tests both in simulation and the real world demonstrate that the methods presented are applicable to real robots and applications such as adapting to changing environments.

For modeling uncertainty, directly encoding uncertainty into the roadmap using Safety-PRM provides a tunable and high clearance method at reduced runtime cost compared to similar methods. This thesis showed that Safety-PRM was an order of magnitude less expensive than MAPRM in terms of collision detection calls. It also showed that the success rate of Safety-PRM reached 100% with a distortion amount of $\sigma = 10$ for all roadmap sizes with $\gamma > 0.9$. However, the comparison method MAPRM, which is known for high clearance paths, only reached 100% success rate for the largest roadmap size of 2000 vertices. This shows that Safety-PRM is a powerful method for handling inaccurate workspace models.

For moving obstacles, the incorporation of the formal SR sets into the ad-hoc APF method produces a more accurate representation of the relative robot-obstacle dynamics, which leads to an increased success rate during path planning. SR sets combined with APF were shown to reach a 95% success rate with 300 moving obstacles compared to an 80% success with ORCA, a method for multi-robot maneuvering in highly cluttered dynamic spaces.Thus, the incorporation of SR sets allows path planning methods to construct more informed paths.

Finally, for transition function uncertainty, an online reinforcement learning algorithm is a suitable candidate for a planner when paired with sampling based techniques. Such a reinforcement learner continuously learns and updates its policy by incorporating the most recent experience from the environment and produces motion plans that are adaptive, real-time, and reactive. Combining PRMs with reinforcement learning created an agent that was more agnostic to the state-

space size as shown by BECCA-PRM reaching the threshold performance within 600 iterations regardless of the number of joints used (1-DoF to 7-DoF). In contrast, without using PRMs, BECCA never reached the threshold performance for just 3 joints despite being allowed to run for 70,000 iterations. This indicates that sampling based techniques are a useful tool for reducing the state-space of learning agents.

Each method is complementary to the others as each presents a solution to a specific type of uncertainty. While these methods are useful by themselves, there is still a leap to be made in order to bring fully autonomous robotics into real world situations. The next logical extension of this work is to combine all three methods into a single overarching framework to handle all three uncertainty types simultaneously. This framework would bring robotics another step closer to widespread use for free linkage, free rigid body, and fixed linkage robots.

References

[1] A. Abate, M. Prandini, J. Lygeros, and S. Sastry. Probabilistic reachability and safety for controlled discrete time stochastic hybrid systems. *Automatica*, pages 2724–2734, 2008.

[2] A. Agha-mohammadi, S. Chakravorty, and N.M. Amato. On the probabilistic completeness of the sampling-based feedback motion planners in belief space. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 3983–3990, 2012.

[3] Rami Al-Hmouz, Tauseef Gulrez, and Adel Al-Jumaily. Probabilistic road maps with obstacle avoidance in cluttered dynamic environment. In *IEEE Intelligent Sensors, Sensor Networks and Information Processing Conf.*, pages 241–245, 2004.

[4] R. Alterovitz, T. Siméon, and K. Goldberg. The stochastic motion roadmap: A sampling framework for planning with markov motion uncertainty. In *Robotics: Science and Systems*, pages 246–253. Citeseer, 2007.

[5] N. M. Amato, O. B. Bayazit, L. K. Dale, C. V. Jones, and D. Vallejo. OBPRM: An obstacle-based PRM for 3D workspaces. In *Robotics: The Algorithmic Perspective*, pages 155–168, Natick, MA, 1998. A.K. Peters. Proc. Third Workshop on Algorithmic Foundations of Robotics (WAFR), Houston, TX, 1998.

[6] Tim Bailey and Hugh Durrant-Whyte. Simultaneous localization and mapping (slam): Part ii. *IEEE Robotics & Automation Magazine*, 13(3):108–117, 2006.

[7] J. Barraquand, L. E. Kavraki, J.-C. Latombe, T.-Y. Li, R. Motwani, and P. Raghavan. A random sampling scheme for path planning. *Int. J. of Rob. Res*, 16(6):759–774, 1997.

[8] E. Bertin. *Diagrammes de Voronoi 2D et 3D: applications en analyse d'images*. PhD thesis, Grenoble, France, 1994.

[9] E. Bertin, S. Marchand-Maillet, and J. M. Chassery. Optimizations in Voronoi diagrams. In J. Serra and P. Soille, editors, *Mathematical Morphology and Its Applications to Image Processing*, pages 209–216. Kluwer Academic, Dordrecht, The Netherlands, 1994.

[10] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Sci., 2005.

[11] R. Bohlin and L. E. Kavraki. Path planning using Lazy PRM. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 521–528, 2000.

[12] Robert Bohlin. Path planning in practice; lazy evaluation on a multi-resolution grid. In *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, volume 1, pages 49–54. IEEE, 2001.

[13] Oliver Brock and Oussama Khatib. Elastic strips: Real-time path modification for mobile manipulation. *International Symposium of Robotics Research*, 8:5–13, 1997.

[14] Brendan Burns and Oliver Brock. Sampling-based motion planning with sensing uncertainty. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 3313–3318. IEEE, 2007.

[15] Omer Cetin, Sefer Kurnaz, Okyay Kaynak, and Hakan Temeltas. Potential field-based navigation task for autonomous flight control of unmanned aerial vehicles. *International Journal of Automation and Control*, 5(1):1–21, 2011.

[16] S. Chakravorty and S. Kumar. Generalized sampling-based motion planners. *IEEE Trans. Sys., Man, Cybern.*, 41(3):855–866, 2011.

[17] Hao-Tien Chiang, Nick Malone, Kendra Lesser, Meeko Oishi, and Lydia Tapia. Aggressive moving obstacle avoidance using a stochastic reachable set based potential field. *Workshop on the Algorithmic Foundations of Robotics*, 2014.

[18] Hao-Tien Chiang, Nick Malone, Kendra Lesser, Meeko Oishi, and Lydia Tapia. Path-guided artificial potential fields with stochastic reachable sets for motion planning in highly dynamic environments. In *Proc. IEEE Int. Conf. Robot. Autom.(ICRA)*, 2015.

[19] H. Choset and J. Burdick. Sensor-based exploration: The hierarchial generalized voronoi graph. *Int. J. Robot. Res.*, 19(2):96–125, 2000.

[20] Howie Choset, Kevin M. Lynch, Seth Hutchinson, George A. Kantor, Wolfram Burgard, Lydia E. Kavraki, and Sebastian Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, Cambridge, MA, June 2005.

[21] Howie Choset, Sean Walker, Kunnayut Eiamsa-Ard, and Joel Burdick. Sensor-based exploration: Incremental construction of the hierarchical generalized voronoi graph. *Int. J. Robot. Res.*, 19(2):126–148, 2000.

[22] Jory Denny and N.M. Amato. Toggle prm: Simultaneous mapping of c-free and c-obstacle - a study in 2d -. In *Intelligent Robots and Systems (IROS), IEEE/RSJ International Conference on*, pages 2632–2639, 2011.

[23] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, pages 269–271, 1959.

[24] J. Ding, E. Li, Haomiao Huang, and C.J. Tomlin. Reachability-based synthesis of feedback policies for motion planning under bounded disturbances. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 2160–2165, 2011.

[25] Dmitri Dolgov, Sebastian Thrun, Michael Montemerlo, and James Diebel. Path planning for autonomous vehicles in unknown semi-structured environments. *The International Journal of Robotics Research*, 29(5):485–501, 2010.

[26] N. Engelharda, F. Endresa, J. Hessa, J. Sturmb, and W. Burgarda. Real-time 3d visual slam with a hand-held rgb-d camera. In *Proc. of the RGB-D Workshop on 3D Perception in Robotics at the European Robotics Forum, Vasteras, Sweden*, volume 2011, 2011.

[27] Sarah Ferguson, Brandon Luders, Robert C Grande, and Jonathan P How. Real-time predictive modeling and robust avoidance of pedestrians with uncertain, changing intentions. *International Workshop on the Algorithmic Foundations of Robotics*, 2014.

[28] Tamar Flash and Neville Hogans. The coordination of arm movements: An experimentally confirmed mathematical model. *Journal of neuroscience*, 5:1688–1703, 1985.

[29] Jodi Forlizzi and Carl DiSalvo. Service robots in the domestic environment: a study of the roomba vacuum in the home. In *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*, pages 258–265. ACM, 2006.

[30] Mark Foskey, Maxim Garber, Ming C. Lin, and Dinesh Manocha. Sm01-144: A voronoi-based framework for motion planning and maintainability applications. Technical report, University of North Carolina, 2001.

[31] Open Perception Foundation. Point cloud library, 2014.

[32] Shuzhi S. Ge and Yun J Cui. Dynamic motion planning for mobile robots using potential field method. *Autonomous Robots*, 13(3):207–222, 2002.

[33] Shuzhi Sam Ge and Yan Juan Cui. New potential functions for mobile robot path planning. *IEEE Transactions on robotics and automation*, 16(5):615–620, 2000.

[34] R. Geraerts and M. H. Overmars. Creating high-quality roadmaps for motion planning in virtual environments. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 4355–4361, 2006.

[35] Jeremy H. Gillula, Gabriel M. Hoffmann, Huang Haomiao, Michael P. Vitus, and Claire J. Tomlin. Applications of hybrid reachability analysis to robotic aerial vehicles. *Int. J. Robot. Res.*, pages 335–354, 2011.

[36] Sani Hashim and Tien-Fu Lu. A new strategy in dynamic time-dependent motion planing for nonholonomic mobile robots. In *IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 1692–1697, Dec 2009.

[37] Kris Hauser, Timothy Bretl, Jean-Claude Latombe, and Brian Wilcox. Motion planning for a six-legged lunar robot. In Srinivas Akella, NancyM. Amato, WesleyH. Huang, and Bud Mishra, editors, *Algorithmic Foundation of Robotics VII*, volume 47 of *Springer Tracts in Advanced Robotics*, pages 301–316. Springer Berlin Heidelberg, 2008.

[38] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox. Rgb-d mapping: Using depth cameras for dense 3d modeling of indoor environments. In *the 12th International Symposium on Experimental Robotics (ISER)*, 2010.

[39] John Hopcroft, Deborah Joseph, and Sue Whitesides. Movement problems for 2-dimensional linkages. *SIAM Journal on Computing*, 13(3):610–629, 1984.

[40] John E Hopcroft and Gordon T. Wilfong. Reducing multiple object motion planning to graph searching. *SIAM Journal on Computing*, 15(3):768–785, 1986.

[41] D. Hsu, R. Kindel, J. C. Latombe, and S. Rock. Randomized kinodynamic motion planning with moving obstacles. In *Proc. Int. Workshop on Algorithmic Foundations of Robotics (WAFR)*, pages SA1–SA18, 2000.

[42] D. Hsu, J-C. Latombe, and Hanna Kurniawati. Foundations of probabilistic roadmap planning. In *Proc. Int. Workshop on Algorithmic Foundations of Robotics (WAFR)*, 2006.

[43] A.S. Huang, A. Bachrach, P. Henry, M. Krainin, D. Maturana, D. Fox, and N. Roy. Visual odometry and mapping for autonomous flight using an rgb-d camera. In *Int. Symposium on Robotics Research (ISRR),(Flagstaff, Arizona, USA)*, 2011.

[44] Yifeng Huang and Kamal Gupta. Rrt-slam for motion planning with motion and map uncertainty for robot exploration. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 1077–1082. IEEE, 2008.

[45] Barrett Technollogy Inc. Wam arm, 2015.

[46] L. Jaillet and T. Simeon. A PRM-based motion planner for dynamically changing environments. In *Proc. IEEE Int. Conf. Intel. Rob. Syst. (IROS)*, 2004.

[47] Léonard Jaillet, Juan Cortés, and Thierry Siméon. Sampling-based path planning on configuration-space costmaps. *Robotics, IEEE Transactions on*, 26(4):635–646, 2010.

[48] Leonard Jaillet and Thierry Siméon. Path deformation roadmaps: Compact graphs with useful cycles for motion planning. *I. J. Robotic Res.*, 27(11-12):1175–1188, 2008.

[49] M. Kamgarpour, J. Ding, S. Summers, A. Abate, J. Lygeros, and C. Tomlin. Discrete time stochastic hybrid dynamical games: Verification and controller synthesis. In *IEEE Conf. on Decision and Cont.*, pages 6122–6127, 2011.

[50] R. Katz, N. Melkumyan, J. Guivant, T. Bailey, J. Nieto, and E. Nebot. Integrated sensing framework for 3d mapping in outdoor navigation. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 2264–2269. IEEE, 2006.

[51] L. E. Kavraki, P. Švestka, J. C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. Automat.*, 12(4):566–580, August 1996.

[52] Lydia E Kavraki and Jean-Claude Latombe. Probabilistic roadmaps for robot path planning. 1998.

[53] Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The international journal of robotics research*, 5(1):90–98, 1986.

[54] T. Khuswendi, H. Hindersah, and W. Adiprawita. Uav path planning using potential field and modified receding horizon a* 3d algorithm. In *Electrical Engineering and Informatics (ICEEI), 2011 International Conference on*, pages 1–6, July 2011.

[55] K. Konolige. Projected texture stereo. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 148–155. IEEE, 2010.

[56] Hanna Kurniawati and Vinay Yadav. An online pomdp solver for uncertainty planning in dynamic environment. ISRR, 2013.

[57] Chi-Pang Lam, Chen-Tun Chou, Kuo-Hung Chiang, and Li-Chen Fu. Human-centered robot navigation towards a harmoniously human–robot coexisting environment. *Robotics, IEEE Transactions on*, 27(1):99–112, 2011.

[58] Heon-Cheol Lee, Touahmi Yaniss, and Beom-Hee Lee. Grafting: a path replanning technique for rapidly-exploring random trees in dynamic environments. *Advanced Robotics*, 26(18):2145–2168, 2012.

[59] Stephen R Lindemann and Steven M LaValle. Current issues in sampling-based motion planning. In *Robotics Research*, pages 36–54. Springer, 2005.

[60] A. Majumdar and R. Tedrake. Robust online motion planning with regions of finite time invariance. In *Algorithmic Foundations of Robotics*, pages 543–558. Springer, 2013.

[61] Nick Malone, Hao-Tien Chiang, Kendra Lesser, Steven Cutlip, Nathan Rackley, Meeko Oishi, and Lydia Tapia. Hybrid dynamic moving obstacle avoidance using a stochastic reachable set based potential field. In *Under submission*.

[62] Nick Malone, Aleksandra Faust, Brandon Rohrer, Ron Lumia, John Wood, and Lydia Tapia. Efficient motion-based task learning for a serial link manipulator. *Transaction on Control and Mechanical Systems*, 3(1), 2014.

[63] Nick Malone, Aleksandra Faust, Brandon Rohrer, John Wood, and Lydia Tapia. Efficient motion-based task learning. In *Robot Motion Planning Workshop, IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct 2012.

[64] Nick Malone, Kendra Lesser, Meeko Oishi, and Lydia Tapia. Stochastic reachability based motion planning for multiple moving obstacle avoidance. In *Hybrid Systems: Computation and Control*. HSCC, 2014.

[65] Nick Malone, Kasra Manavi, Ron Lumia, John Wood, and Lydia Tapia. Motion planning using roadmaps that incorporate workspace modeling errors. In *Under submission*.

[66] Nick Malone, Kasra Manavi, John Wood, and Lydia Tapia. Construction and use of roadmaps that incorporate workspace modeling errors. In *Proc. IEEE Int. Conf. Intel. Rob. Syst. (IROS)*, pages 1264–1271, 2013.

[67] Nick Malone, Kasra Manavi, John Wood, and Lydia Tapia. Construction and use of roadmaps that incorporate workspace modeling errors. In *Intelligent Robots and Systems, 2013. (IROS 2013). Proceedings. 2013 IEEE/RSJ International Conference on*, pages 1264–1271, 2013.

[68] Nick Malone, Brandon Rohrer, Lydia Tapia, Ron Lumia, and John Wood. Implementation of an embodied general reinforcement learner on a serial link manipulator. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 862–869, May 2012.

[69] K. Margellos and J. Lygeros. Hamilton-Jacobi formulation for reach-avoid problems with an application to air traffic management. *Amer. Cont. Conf.*, pages 3045–3050, 2010.

[70] Mauro Massari, Giovanni Giardini, and Franco Bernelli-Zazzera. Autonomous navigation system for planetary exploration rover based on artificial potential fields. In *Proceedings of Dynamics and Control of Systems and Structures in Space (DCSSS) 6th Conference*, pages 153–162, 2004.

[71] MathWorks. MATLAB version 7.7.0 r2008b, 2008.

[72] MATLAB. *version 7.7.0 R2008b*. The MathWorks Inc., 2007.

[73] Microsoft. Kinect sensor, 2015.

[74] Patrycja E Missiuro and Nicholas Roy. Adapting probabilistic roadmaps to handle uncertain maps. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 1261–1267, 2006.

[75] I.M. Mitchell, A. Bayen, and C.J. Tomlin. A time-dependent Hamilton-Jacobi formulation of reachable sets for continuous dynamic games. *Trans. on Auto. Cont.*, pages 947–957, 2005.

[76] Toshiharu Mukai, Shinya Hirano, Hiromichi Nakashima, Yo Kato, Yuki Sakaida, Shijie Guo, and Shigeyuki Hosoe. Development of a nursing-care assistant robot riba that can lift a human in its arms. In *Intelligent Robots and*

Systems (IROS), 2010 IEEE/RSJ International Conference on, pages 5996–6001. IEEE, 2010.

[77] Venkatraman Narayanan, Mike Phillips, and Maxim Likhachev. Anytime safe interval path planning for dynamic environments. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 4708–4715. IEEE, 2012.

[78] Chuong V Nguyen, Shahram Izadi, and David Lovell. Modeling kinect sensor noise for improved 3d reconstruction and tracking. In *3d Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT), 2012 Second Conference on*, pages 524–530. IEEE, 2012.

[79] Dennis Nieuwenhuisen, Jur van den Berg, and Mark Overmars. Efficient path planning in changing environments. In *Proc. IEEE Int. Conf. Intel. Rob. Syst. (IROS)*, pages 3295–3301, 2007.

[80] Jung-Jun Park, Ji-Hun Kim, and Jae-Bok Song. Path planning for a robot manipulator based on probabilistic roadmap and reinforcement learning. *International Journal of Control, Automation, and Systems*, 5(6):674–680, 2007.

[81] Sachin Patil, Jur Van Den Berg, Sean Curtis, Ming C Lin, and Dinesh Manocha. Directing crowd simulations using navigation fields. *Visualization and Computer Graphics, IEEE Transactions on*, 17(2):244–254, 2011.

[82] Romain Pepy and Alain Lambert. Safe path planning in an uncertain-configuration space using rrt. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 5376–5381. IEEE, 2006.

[83] S. Prentice and N. Roy. The belief roadmap: Efficient planning in belief space by factoring the covariance. *The Int. Journal of Robotics Research*, 28(11-12):1448–1465, 2009.

[84] Sean Quinlan and Oussama Khatib. Elastic bands: Connecting path planning and control. In *In Proc. of the International Conference on Robotics and Automation*, pages 802–807, 1993.

[85] Marc Raibert, Kevin Blankespoor, Gabriel Nelson, Rob Playter, et al. Bigdog, the rough-terrain quadruped robot. In *Proceedings of the 17th World Congress*, volume 17, pages 10822–10825, 2008.

[86] Samuel Rodriguez, Jyh-Ming Lien, and Nancy M. Amato. A framework for planning motion in environments with moving obstacles. In *Proc. IEEE Int. Conf. Intel. Rob. Syst. (IROS)*, 2007.

[87] B. Rohrer. Biologically inspired feature creation for multi-sensory perception. In *Proc. Int. Conf. on Biologically Inspired Cognitive Architectures*, volume 233 of *Frontiers in Artificial Intelligence and Applications*, pages 305–313. IOS Press, Nov 2011.

[88] B. Rohrer. A developmental agent for learning features, environment models, and general robotics tasks. In *IEEE Conference on Development and Learning and Epigenetic Robotics*, Aug 2011.

[89] Brandon Rohrer. An implemented architecture for feature creation and general reinforcement learning. In *Workshop on Self-Programming in AGI Syst., Int. Conf. on Artificial General Intelligence*, Aug 2011.

[90] Brandon Rohrer. BECCA: Reintegrating AI for natural world interaction. In *AAAI Spring Symposium on Designing Intelligent Robots: Reintegrating AI 2012*, Mar 2012.

[91] M. S. Branicky S. M. Lavalle and S. R. Lindemann. On the relationship between classical grid search and probabilistic roadmaps. *Int. J. Robot. Res.*, 23(7–8):673–692, 2004.

[92] Z. Shiller, K. Yamane, and Y. Nakamura. Planning motion patterns of human figures using a multi-layered grid and the dynamics filter. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, volume 1, pages 1–8, 2001.

[93] Zvi Shiller, Frederic Large, and Sepanta Sekhavat. Motion planning in dynamic environments: obstacles moving along arbitrary trajectories. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, volume 4, pages 3716–3721, 2001.

[94] T. Simeon, J.-P. Laumond, and C. Nissoux. Visibility-based probabilistic roadmaps for motion planning. *Advanced Robotics*, 14(6):477–493, 2000.

[95] G. Song, S. L. Miller, and N. M. Amato. Customizing PRM roadmaps at query time. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 1500–1505, 2001.

[96] Qiang Song and Lingxia Liu. Mobile robot path planning based on dynamic fuzzy artificial potential field method. *International Journal of Hybrid Information Technology*, 5(4), 2012.

[97] J. Sturm, S. Magnenat, N. Engelhard, F. Pomerleau, F. Colas, W. Burgard, D. Cremers, and R. Siegwart. Towards a benchmark for rgb-d slam evaluation. In *Proc. of the RGB-D Workshop on Advanced Reasoning with Depth Cameras at Robotics: Science and Systems Conf.(RSS), Los Angeles, USA*, volume 2, page 3, 2011.

[98] Sean Summers, Maryam Kamgarpour, John Lygeros, and Claire Tomlin. A stochastic reach-avoid problem with random obstacles. In *Proc. Int. Conf. Hybrid Sys.: Comp. and Cont. (HSCC)*, pages 251–260, 2011.

[99] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning. MIT Press, MIT, 1998.

[100] R. Takei, Haomiao Huang, J. Ding, and C.J. Tomlin. Time-optimal multi-stage motion planning with guaranteed collision avoidance via an open-loop game formulation. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 323–329, 2012.

[101] Matthew E. Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *J. Mach. Learn. Res.*, 10:1633–1685, Dec 2009.

[102] S. Thrun. Robotic mapping: A survey. *Exploring artificial intelligence in the new millennium*, 1:1–35, 2003.

[103] S. Thrun, W. Burgard, and D. Fox. A real-time algorithm for mobile robot mapping with applications to multi-robot and 3d mapping. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 1, pages 321–328. IEEE, 2000.

[104] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.

[105] Goce Trajcevski, Ouri Wolfson, Klaus Hinrichs, and Sam Chamberlain. Managing uncertainty in moving objects databases. *ACM Transactions on Database Systems (TODS)*, 29(3):463–507, 2004.

[106] Dugan Um, Marco A Gutiérrez, Pablo Bustos, and Sungchul Kang. Simultaneous planning and mapping (spam) for a manipulator by best next move in unknown environments. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 5273–5278. IEEE, 2013.

[107] Prahlad Vadakkepat, Kay Chen Tan, and Wang Ming-Liang. Evolutionary artificial potential fields and their application in real time robot path planning. In *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*, volume 1, pages 256–263. IEEE, 2000.

[108] Kimon P Valavanis, Timothy Hebert, Ramesh Kolluru, and Nikos Tsourveloudis. Mobile robot navigation in 2-d dynamic environments using an electrostatic potential field. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 30(2):187–196, 2000.

[109] Jur Van Den Berg, Dave Ferguson, and James Kuffner. Anytime path planning and replanning in dynamic environments. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 2366–2371, 2006.

[110] Jur Van Den Berg, Stephen J Guy, Ming Lin, and Dinesh Manocha. Reciprocal n-body collision avoidance. In *Robotics research*, pages 3–19. Springer, 2011.

[111] Jur P van den Berg, Dennis Nieuwenhuisen, Léonard Jaillet, and Mark H Overmars. Creating robust roadmaps for motion planning in changing environments. In *Proc. IEEE Int. Conf. Intel. Rob. Syst. (IROS)*, pages 1053–1059, 2005.

[112] Jur P Van Den Berg and Mark H Overmars. Roadmap-based motion planning in dynamic environments. *IEEE Transactions on Robotics*, pages 885–897, 2005.

[113] Su Weijun, Meng Rui, and Yu Chongchong. A study on soccer robot path planning with fuzzy artificial potential field. In *Computing, Control and Industrial Engineering (CCIE), 2010 International Conference on*, volume 1, pages 386–390, June 2010.

[114] S. A. Wilmarth, N. M. Amato, and P. F. Stiller. MAPRM: A probabilistic roadmap planner with sampling on the medial axis of the free space. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, volume 2, pages 1024–1031, 1999.

[115] Y. Wu. An obstacle-based probabilistic roadmap method for path planning. Master's thesis, Department of Computer Science, Texas A&M University, 1996.

[116] Eiichi Yoshida and Fumio Kanehiro. Reactive robot motion using path replanning and deformation. In *ICRA*, pages 5456–5462, 2011.