3D Motion Planning using Kinodynamically Feasible Motion Primitives in Unknown Environments

by

Peiyi Chen

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Mechanical Engineering

Waterloo, Ontario, Canada, 2011

Abstract

Autonomous vehicles are a great asset to society by helping perform many dangerous or tedious tasks. They have already been successfully employed for many practical applications, such as search and rescue, automated surveillance, exploration and mapping, sample collection, and remote inspection. In order to perform most tasks autonomously, the vehicle must be able to safely and efficiently navigate through its environment. The algorithms and techniques that allow an autonomous vehicle to find traversable paths to its destination defines the set of problems in robotics known as *motion planning*.

This thesis presents a new motion planner that is capable of finding collision-free paths through an unknown environment while satisfying the kinodynamic constraints of the vehicle. This is done using a two step process. In the first step, a collision-free path is generated using a modified Probabilistic Roadmap (PRM) based planner by assuming unexplored areas are obstacle-free. As obstacles are detected, the planner will replan the path as necessary to ensure that it remains collision-free. In complex environments, it is often necessary to increase the size of the PRM graph during the replanning step so that the graph remains connected. However, this causes the algorithm to slow down significantly over time. To mitigate these issues, the novel local sampling and PRM regeneration techniques are used to increase the computational efficiency of the replanning step. The local sampling technique biases the search towards the neighborhood of the obstacle blocking the path. This encourages the planner to generate small detours around the obstacle instead of rerouting the whole path. The PRM regeneration technique is used to remove all non-critical nodes from the PRM graph. This is used to bound the size of the PRM graph so that it does not grow increasingly large over time.

In the second step, the collision-free path is transformed into a series of kinodynamically feasible motion primitives using two novel algorithms: the heuristic re-sampling algorithm and the transformation algorithm. The heuristic re-sampling algorithm is a greedy heuristic algorithm that increases the clearance around the path while removing redundant segments. This algorithm can be applied to any piece-wise linear path, and is guaranteed to produce a solution that is at least as good as the initial path. The transformation algorithm is a method to convert a path into a series of kinodynamically feasible motion primitives. It is extremely efficient computationally, and can be applied to any piece-wise linear path.

To achieve good computational performance with PRM based planners, it is necessary to use sampling strategies that can efficiently form connected graphs through narrow and complex regions of the configuration space. Many proposed sampling methods attempt to bias the sample density in favor of these difficult to connect areas. However, these methods do not distinguish between samples that lie inside narrow passages and those that lie along convex borders. The orthogonal bridge test is a novel sampling technique that can identify and reject samples that lie along convex borders. This allows connected PRM graphs to be constructed with fewer nodes, which leads to less collision checking and reduced runtimes.

The presented algorithms are experimentally verified using an AR.Drone quadrotor unmanned aerial vehicle (UAV) and a custom built skid-steer unmanned ground vehicle (UGV). Using a simple kinematic model and a basic position controller, the AR.Drone is able to traverse a series of motion primitives with less than 0.3 m of crosstrack error. The skid-steer UGV is able to navigate through unknown environments filled with obstacles to reach a desired destination. Furthermore, the observed runtimes of the proposed motion planner suggest that it is fully capable of computing solution paths online. This is an important result, because online computation is necessary for efficient autonomous operations and it can not be achieved with many existing kinodynamic motion planners.

Acknowledgements

This thesis would not have been possible without the support and guidance of my supervisor, Professor Steven Waslander. His encouragement and advice helped me overcome many difficult challenges, and allowed me to grow as a student and researcher over the past two years.

I would also like to thank my fellow lab mates in the Waterloo Autonomous Vehicles Lab: Yassir Rizwan, Carlos Wang, PJ Mukherjee, Arun Das, Yan Ma, Ryan Gariepy, John Daly, Mike Tribou, and Sid Ahuja. Their assistance on countless occasions has proven to be invaluable, and the camaraderie made the experience genuinely fun and interesting.

I am also grateful to the University of Waterloo Robotics Team, and especially Ryan Turner, for their assistance in the design and construction of the custom skid-steer vehicle that was entered into the 2011 International Ground Vehicle Competition. This vehicle is an excellent test platform, and was used to generate a large portion of the experimental results in this thesis.

Finally, I would like to thank my family and all my friends who have stood beside me every step of the way.

Dedication

This is dedicated to my family and friends whose support made this work possible.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Robots allow many tasks to be accomplished more safely and efficiently. They have proven to be highly valuable in many areas including military (e.g. combat and spy drones, mine detection robots), industrial (e.g. automated assembly lines), and aerospace (e.g. satellites, Canadarm). As robots become more prevalent, significant effort is being made to improve their level of autonomy so that they can operate with minimal human input. This increases efficiency by allowing fewer human operators to control a larger number of robots. As well, increased autonomy allows the robot to be less dependent on human intervention. This is beneficial in cases where there exists a significant delay in transmitting commands to and from the robot (i.e. teleoperating a robot deployed on another planet).

Unmanned vehicles are autonomous robots that grant people the ability to access inconvenient or hostile locations remotely. There are different types of unmanned vehicles including unmanned aerial vehicles (UAVs), unmanned ground vehicles (UGVs), and unmanned underwater vehicles (UUVs). These vehicles serve many practical applications such as search and rescue, surveillance, exploration and mapping, sample collection, and fault inspection of various structures like bridges or power lines. The presented research will primarily be focused on the quadrotor UAV and skid-steer UGV platforms.

Motion planning is one of the fundamental problems that must be solved before vehicles can operate autonomously. The ability to safely maneuver within a given environment is required for almost any robotic application. Using information gathered through on-board sensors, the motion planner is responsible for finding collision-free paths that will allow the vehicle to transition into a desired state. Unfortunately, all vehicles are subject to the laws of physics, and hence its motion will be limited by kinematic and dynamic (kinodynamic) constraints that restrict the possible degrees of freedom and the achievable velocities and accelerations. Thus, as an additional caveat, the motion planner should enforce these constraints while generating solution paths in order to ensure that they are traversable by the target vehicle. This describes the class of problems known as ”kinodynamic motion planning” which is the focus of this work.

The primary objective of this research is to develop a kinodynamic motion planner that is capable of guiding an autonomous vehicle, from an initial location to a specified target location, through a 3D environment filled with static unknown obstacles. To simplify the kinodynamic complexity, the vehicle platforms will be restricted to those capable of turning without translation (i.e. minimum turning radius of zero). The target experimental platforms used in this work is the quadrotor UAV and a skid-steer UGV. For these platforms, the common sensors for environmental perception include monocular and stereo visible light and infrared (IR) cameras, LIght Detection And Ranging (LIDAR), and SOund Navigation And Ranging (SONAR). Furthermore, to decouple the motion planning problem from the localization problem, a Global Positioning System (GPS) sensor or an indoor equivalent (i.e. OptiTrack$^{\text{TM}}$) is assumed to be available on the vehicle. To help position the presented research within the existing motion planning literature, an overview of several existing motion planning techniques will be given in Section 1.1. Then, the approach and contributions of this work will be described in Section 1.2.

1.1 Related Work

There are many existing motion planning techniques for autonomous robots. The key working principles of many of these techniques are platform independent and can therefore be adapted to work with a variety of different robots. This section will provide a brief introduction to several motion planning techniques that are commonly used in autonomous robotic applications. Note that this section is only an overview, and a more focused examination of related works will be provided at the start of each chapter.

1.1.1 Potential Field and Level Set Methods

Artificial potential fields (APF) [4] is a greedy path planning algorithm that works on the principle of gradient descent. In order to find a path, it is necessary to convert the map of the environment into a corresponding potential map that contains the potential values at each location. The vehicle is attracted to the destination (which has the lowest potential) and repelled from obstacles (which have the highest potentials). Given a starting position, the solution path can be obtained through greedy gradient descent of the potential map. APF techniques are simple to implement and have been shown to work well in environments with relatively sparse obstacles. Furthermore, after the potential map has been calculated, paths can be very efficiently generated from any starting location. Unfortunately, APF techniques also suffer from several shortcomings. First and most significantly, the vehicle can often get stuck in local minima (i.e. non-destination locations that have lower potential than all its adjacent neighbors) and become unable to proceed to the destination. This issue is especially prominent in cluttered environments with concave obstacles, and has been the subject of a lot of research. Many methods have been proposed to alleviate this problem by either detecting and escaping local minima [65], or by generating potential maps with only one minimum at the desired destination [52]. The second limitation of APF techniques is that they do not consider the kinodynamic vehicle constraints, and thus there is no guarantee that the solution path can be traversed by the given vehicle platform. Finally, because the potential map assigns a value to every location, APF techniques generally rely on an occupancy grid based map of the environment which can lead to discretization and resolution issues.

Level set methods such as wavefront [64] or fast marching [55] share many similarities with APF techniques. The main idea behind these methods is that if a wave that flows around obstacles is propagated outwards from the desired destination, it will eventually reach every possible starting location in the environment for which there exists a path to the destination. An analogy that is commonly used is to imagine the destination as a heat source and the free space as some conductive material. Eventually, the heat will transfer to all free space that is connected to the destination through some obstacle free path. The resulting map containing the propagated wavefronts will closely resemble the potential maps of the APF techniques. Finally, the vehicle reconstructs the path by following the wavefronts back to their source. Unlike APF techniques, level set methods do not have local minima where the vehicle can get stuck during path reconstruction. Since the wavefront is propagated from the destination, every point reached by the wave can be traced backwards to the destination. A variation of this technique, known as dual wavefront [48], attempts to decrease runtimes by propagating two waves (one from the starting location and one from the destination) simantaneously. Once the two waves meet, the solution path is reconstructed in the same manner using both waves.

The wavefront technique accomplishes the propagation of the simulated wave through dynamic programming on grid based maps. The method is straight forward to implement, but its performance is limited by having to enumerate through each cell of the grid based maps. As a result, the method scales poorly with the size and dimension of the environment space and there is a trade-off between map resolution and computational efficiency. If the environment is large and the map has a very fine resolution, the algorithm will slow down due to the sheer number of cells it has to propagate the wave through. However, if the map resolution is increased for computational efficiency, many discretization issues, such as obstacle dilation and loss of detail, will arise. Finally, like APF techniques, this method does not consider kinodynamic vehicle constraints, and in fact the solution paths are frequently not smooth due to the discretization of the grid based maps.

Fast marching is a numerical method used for approximating solutions to the non-linear Eikonal equation which can be used to describe the evolution of a closed curve (i.e. the boundary of the simulated wave as it is propagated through the environment). Fast marching is an extension of the wavefront technique with several advantages. By using the Eikonal equation, the wave propagation becomes continuous and is regulated by a user selected speed function. This eliminates the dependency on grid based maps and any discretization issues that may result. Additionally, the ability to vary the speed function throughout the environment allows for factors such as terrain type to be considered in the solution path. For instance, by propagating the wave at speeds corresponding to how fast the vehicle can travel over each type of terrain, the resulting solution path will be time optimized based on both distance and achievable velocities. However, other kinodynamic constraints, such as maximum accelerations and angular rates, are still ignored.

1.1.2 Graph Methods using Map Decomposition

Graphs are an efficient way of representing the traversable free space in an environment. Given such a graph, many existing graph theory methods, such as Dijkstra's Algorithm [13], A* search [22], or even fast marching [19], can be used to find a solution path linking the two desired locations. Therefore, the challenge lies in efficiently converting the map of the environment into an equivalent graph representation. This section will examine some of the deterministic methods that can be used to create graph representations of the environment.

One method of efficiently creating a graph representation of the environment is to construct a visibility graph [46]. This is done by setting the starting location, destination, and the vertices of all the obstacles as the nodes of the graph, and then forming collision-free edges between each pair of nodes where possible (edges along the boundaries of obstacles are also included).The main advantage of this technique is that in 2D environments the graph will contain the shortest solution path (globally optimal with respect to distance) which is then easily found using Dijkstra's algorithm or A* search. Unfortunately, this will generally not hold true for 3d environments. In addition, since the solution path will lie very close to obstacles as a result of how the visibility graph is constructed, there will be an increased risk of collision during traversal. Furthermore, in the presence of obstacles without vertices (e.g. circular obstacles), constructing the visibility graph becomes significantly more difficult and complex. Finally, the kinodynamic vehicle constraints are not considered so it may be difficult to traverse the solution path efficiently.

Voronoi decomposition [3] is a method of dividing the environment into a graph (Voronoi diagram) in which each edge is equidistant to the two nearest obstacles, and each node is equidistant to the nearest three or more obstacles. There exists a variety of different algorithms to generate the Voronoi diagram, and the choice of algorithm will depend largely on the properties of the given environment. For instance, Fortune's algorithm [15] can efficiently construct Voronoi diagrams of 2D environments containing point obstacles, the Bowyer-Watson algorithm [10, 61] can construct Voronoi diagrams of 3D environments containing point obstacles, and other more advanced algorithms, such as [6] and [23], can construct generalized Voronoi diagrams containing non-point obstacles. Performing Dijkstra's algorithm or A* search on the Voronoi diagram will yield a solution path that attempts to maximize the clearance (distance to nearest obstacle) at every point. The main drawback of this method is that it does not consider the kinodynamic vehicle constraints. However, this is somewhat mitigated since the path attempts to maximize clearance thus reducing the risk of collision for small deviations off the path.

Cell decomposition methods, such as trapezoidal decomposition [58] and Morse decomposition [1], divides the free space into geometrically simpler subspaces (cells) such that their sum will be equal (or approximately equal) to the entire free space. The graph representation of the environment depicts each cell as a node with edges connecting pairs of nodes whose cells are adjacent to each other. Unfortunately, the resulting graph only captures the adjacency information of the cells, so performing Djikstra's algorithm or A* search will generally not yield a solution path, but rather a series of cells that contains a solution path. However, solving for the solution path within the cells is generally non-trivial, so additional effort (i.e. another method) is often required.

1.1.3 Graph Methods using Random Sampling

Graph representations of the environment can also be generated using random sampling based approaches such as probabilistic roadmaps (PRM) [34] and rapidly-expanding random trees (RRT) [41]. These techniques (and their many variations) are very popular due to scalability and robustness. The main idea behind these techniques is to capture the connectivity of complex spaces using random samples.

The traditional PRM algorithm generates graph nodes randomly throughout a given environment, and then connects each pair of nodes with a straight collision-free edge where possible. Given a starting location and destination, these two points can be appended to the PRM graph as nodes and connected to the rest of the graph through collision-free edges. Then, the solution path can be obtained by performing Dijkstra's algorithm or A* search on the resulting graph. PRM techniques are efficient and work well for large, complex environments even in high dimensions. However, the random nature of the PRM nodes can often lead to a path that is circuitous. As well, this method ignores the kinodynamic vehicle constraints and tends to generate piece-wise linear paths that are difficult to traverse efficiently.

RRT planners also generate graph representations of the environment through random sampling. However, unlike traditional PRM planners, RRT takes the vehicle model into consideration and will thus satisfy all kinodynamic constraints. Instead of naively connecting the sampled nodes to form a graph, the RRT method builds a tree (rooted at the starting location) by propagating control inputs through the vehicle model. Each node in the constructed tree represents not only a location, but also the full set of vehicle states as well. During each iteration, the algorithm will sample a random location in the environment and then find the node in the current tree that is closest to that sampled location. Then, starting at the closest node in the tree, the set of feasible control inputs are propagated through the vehicle model for one time step. The control input that allows the vehicle to move the closest to the sampled location will be selected, and the resulting set of vehicle states will be added to the tree as a new node. When the destination is reached by a branch of the tree, the path can be reconstructed using backward induction.

Despite their many strengths, RRT planner also have several limitations. First, the performance of the overall algorithm strongly depends on the local planner responsible for selecting the control inputs. For more complex vehicle models, finding an appropriate local planner can be a challenge. Additionally, due to the random nature of the algorithm, the solution path will often contain many unnecessary deviations. While these deviations also appear in paths generated by PRM algorithms, the effects are more prominent because any redundant segments in the path can't be removed using Dijkstra's algorithm or A* search due to the tree structure. As well, since the control inputs are selected to move greedily towards the sampled location at each time step with no regard for subsequent moves, the vehicle state at the start of each segment may be undesirable and require additional control effort to correct. Many modifications have been proposed to mitigate these issues in RRT planners, and these will be examined in Chapter 2.

1.2 Research Approach and Contribution

Many of the existing motion planners are unable to generate kinodynamically feasible paths (e.g. PRM planners) or are too slow computationally to be performed online (e.g. RRT planners). Therefore, the goal of this presented research is to develop a computationally efficient motion planner that can generate collision-free paths, from a starting location to a specified destination, through an unknown 3D environment while satisfying the kinodynamic constraints of the vehicle. As previously mentioned, the problem will be simplified by restricting the vehicle platforms to those with a minimum turning radius of zero, and by assuming the availability of reliable localization information (decoupling of the localization and planning problems). In unknown environments, it will often be necessary for the proposed algorithm to recompute paths online (e.g. when newly detected obstacles is blocking the current path). In order to prevent these online path replans from becoming bottlenecks in the traversal process, the algorithm must be able to calculate each path segment in less time than it takes the vehicle to traverse it. Assuming that all the path segments require approximately equal time to traverse, this enables the algorithm to always have at least one path segment calculated ahead of the vehicle.

The research built an incremental solution to the overall problem by solving a series of subproblems. The first subproblem, presented in Chapter 2, will develop a motion planner for a known 3D environment that satisfies the kinodynamic vehicle constraints. There already exists many methods for finding a collision-free path through a 3D environment, however most of these either ignore the kinodynamic vehicle constraints or require long computational times. Thus, the approach will be to use an existing motion planner, which ignores kinodynamic constraints, to compute an initial collision-free path. Then, using the presented heuristic re-sampling and transformation algorithms, this collision-free path is transformed into a series of kinodynamically feasible motion primitives.

The second subproblem, presented in Chapter 3, will incorporate the transformation algorithm (from the first subproblem) with a motion planner capable of generating collision-free paths in unknown environments. The motion planner for unknown environments must be capable of calculating an initial candidate path and then re-plan as necessary as the vehicle explores the environment. The proposed algorithm for planning in unknown environments is a modified version of the PRM planner with a D* (dynamic A*) search [56]. The transformation algorithm is then applied to the resulting piece-wise linear path to satisfy any kinodynamic vehicle constraints.

The third subproblem attempts to increase the computational efficiency of the PRM based algorithm (developed in the second stage) by using sampling strategies that bias the samples towards regions critical to the connectivity of free space, thus allowing paths to be found with fewer samples. Chapter 4 will introduce a novel sampling technique (the orthogonal bridge test) that can be used to improve the performance of PRM based motion planners.

The proposed algorithms were experimentally verified on two vehicle platforms. The quadrotor UAV platform was used to confirm the traversability of the motion primitives used in the transformation algorithm, and to verify the feasibility of the motion planner for unknown environments. The complete solution was then implemented on a skid-steer UGV. The implementation details and experimental results are presented in Chapter 5.

The main novel contributions that will be presented in this thesis are:

- A heuristic re-sampling algorithm capable of removing redundant path segments and increasing the clearance around any piece-wise linear path.
- A transformation algorithm that can convert any piece-wise linear path into a series of kinodynamically feasible motion primitives.
- A modified PRM based planner, with local sampling and PRM regeneration, that is capable of finding collision-free paths through unknown environments.
- A new sampling technique, the orthogonal bridge test, that can increase sample density inside narrow passages while minimizing the number of less effective samples along convex boundaries.

Chapter 2

Probabilistic Roadmap Based Planner for Known 3D Environments

The focus of this chapter will be motion planning in known static 3D environments while satisfying kinodynamic vehicle constraints. The proposed approach is a two-step process where the first step finds a piece-wise linear collision-free path and the second step transforms that path using motion primitives to satisfy the kinodynamic vehicle constraints. The first step can be accomplished using many of the existing motion planning techniques. For this work, the probabilistic roadmap (PRM) planner is selected due to its robustness and efficiency in complex 3D environments. The second step uses the novel transformation algorithm that will be developed in this chapter.

This chapter will begin by reviewing a variety of existing algorithms for computing kinodynamically feasible paths in known environments in Section 2.1. Then, the standard PRM algorithm will be stated for completeness in Section 2.2. Sections 2.3 and 2.4 will describe the proposed heuristic re-sampling and transformation algorithms respectively. Finally, simulations of these algorithms will be presented in Section 2.5.

2.1 Background and Theory

The rapidly-exploring random tree (RRT) algorithm is one of the most well known kinodynamic motion planners. While the standard form of the RRT planner, briefly introduced in Chapter 1, suffers from issues such as slow runtimes and circuitous paths, many variations and extensions have been proposed to mitigate these problems. RRT-Connect [40] attempts to reduce computation time by expanding the tree more aggressively. Once the sampled location is chosen during the expansion step, the control inputs are repeatedly propagated for multiple time steps until either the sampled location is reached or a collision becomes inevitable. In environments with sparse obstacles, this generally reduces the runtime significantly. Furthermore, RRT-Connect uses the dual tree approach, where one tree rooted at the start and another rooted at the destination are grown simultaneously until they meet in the middle. This allows a connected path to be found faster because it is much easier for the branches of two trees to find each other, than for the branch of a single tree to find a specific location. However, in cluttered environments, the runtime is still insufficient for online applications.

In [12], another variation of RRT is proposed to reduce metric sensitivity. This issue arises in traditional RRT planners because the Euclidean norm on the vehicle position is used as the metric to determine which node to expand towards the sampled location. However, it may not always be desirable to expand from the closest node due to velocity, acceleration, or non-holonomic constraints. Thus, [12] suggests that a *constraint violation frequency* metric, which represents the percentage of potential expansions that will lead to a collision, be calculated for each node. Then, the probability of expanding each node is inversely proportional to its constraint violation frequency. Furthermore, all inputs that have been evaluated (i.e. have led to a collision or successfully generated a new node) are removed from the set of available inputs on their respective nodes. This ensures that time will not be spent re-evaluating these same inputs during future iterations. An extension of [12] is the RRT-Blossum algorithm presented in [30]. The main contribution of RRT-Blossum is the addition of a mechanism that prevents new nodes from being generated in locations where the parent node is not the closest node. This retains the benefits of [12], while encouraging the tree to expand into new areas. Unfortunately, this modification can lead to deadlocks in the presence of non-holonomic constraints. To prevent this, RRT-Blossum introduces a priority system that allows previously forbidden expansions after all other alternatives have been exhausted. While these modifications do offer significant speed ups, the runtime in complex environments is still insufficient for online applications.

RRT* [32] is a variation of the RRT planner that attempts to eliminate circuitous paths. Unlike traditional RRT planners, which generate new nodes by expanding the tree from the node closest to the sampled location, RRT* expands the node that will yield the shortest overall path to the new node. It is shown in [32] that the solution path found by RRT* will approach the optimal path as the number of iterations and nodes increase. Unfortunately, this method is not suited to online applications, because the large number of nodes and iterations required to benefit from this method will increase computation time.

Another variation of the RRT planner is presented in [16]. One major contribution of this work is the concept of *optimal control policy*, which is the set of control inputs that allows the vehicle to traverse a distance optimal path to the destination in the absence of obstacles. This optimal control policy can be calculated for any node, and is used to guide tree expansion. To generate a new node, the planner iterates through all the nodes in the tree until it finds a node whose optimal control policy brings the vehicle within some defined neighborhood of the sampled location without collision. The new node is then generated by propagating the optimal control policy from that node until the vehicle reaches the neighborhood of the sampled location. The order in which the nodes are evaluated during the expansion step differs depending on the phase of the planner. During the exploration phase, the goal is to increase the reachable set of tree, so nodes that are closest to the sampled location are evaluated first. After a feasible path as been found, the planner enters the optimization phase where it attempts to improve the path. During this phase, nodes that yield the shortest overall path to the sampled location are evaluated first (note that this is the same strategy used in RRT*). Furthermore, to increase computational efficiency, a tree pruning technique is introduced, where any child node (and its subtree) is removed if its optimal control policy yields a path (i.e. best potential path from child) that is longer than the current solution path from the parent node (i.e worst potential path from parent). Finally, even though the results presented in [16] suggest that the runtime of this planner is sufficient for online applications in 2D environments, it is still likely to be too slow for online applications in more complex 3D environments.

Motion primitives are also commonly used to generate kinodynamically feasible paths. A motion primitive can be informally defined as a time and position invariant path segment that results from the application of a simple control input. The traditional approach to path planning with motion primitives is to generate a reachability graph by concatenating different motion primitives together. This method is often used to generate paths for kinematic models of non-holonomic vehicles, such as Dubin's car [42]. One drawback of reachability graphs, however, is that if the vehicle has a large number of motion primitives, the graph will quickly become cluttered and contain many redundancies. To mitigate this issue, [49] proposes a method for removing redundancies in reachability graphs by identifying equivalent paths. However, this method can only be applied to kinematic motion primitives, because it does not take the velocity and acceleration of the vehicle into account when concatenating or removing motion primitives. To accommodate vehicles with more complex dynamic motion primitives, [17] introduced the notion of *trim motion primitives* and *maneuvers*. A trim motion primitive is a motion primitive in which all the control inputs are held constant, and a maneuver is a non-trivial series of motion primitives that begins and ends in steady state conditions (i.e. can be concatenated with trim motion primitives on both ends). This enables dynamic motion primitives to be concatenated into a variety of complex paths. Unfortunately, attempting to create a reachability graph with such complex motion primitives is quite inefficient, especially in large 3D environments, and will most likely be too slow for online applications.

Optimization techniques have also been used to solve for kinodynamically feasible paths. In [54], the motion planning problem is converted into an equivalent mixed integer linear program (MILP), which attempts to optimize some metric subject to obstacle and vehicle constraints. The main advantages of this approach is the ability to leverage off the many existing techniques for solving optimization problems, and the possibility of optimizing the path using a non-distance metric (i.e. finding paths that minimize fuel consumption). A number of proposed methods have attempted to incorporate optimization techniques with other existing motion planners. For instance, both [33] and [18] used a non-linear program (NLP) in the expansion step of the RRT algorithm to solve for feasible control inputs that will allow the vehicle to reach the sampled location. Another approach, which is presented in [9] and [59], is to first generate an initial collision free path by ignoring kinodynamic constraints. Then, NLP (in [9]) or MILP (in [59]) is used to generate a kinodynamically feasible path in the vicinity of the initial collision-free path. This greatly reduces the size of the search space, because the optimization algorithm only has to look in the neighborhood of the initial collision-free path. Unfortunately, optimization algorithms scale poorly with the size and complexity of the environment. Thus, in large or cluttered environments, the runtimes of all these algorithms will be too slow for online applications.

Two step planners, where an initial collision-free path is later transformed to satisfy kinodynamic constraints, are also commonly used. In fact, the optimization based techniques presented in [9] and [59] also fall into this category. Another example is [37], where B-splines are used to smooth out the the initial path generated by a PRM planner. However, even though the technique generates a smooth path, it does not explicitly account for the kinodynamic constraints of the vehicle. Thus, the feasibility of the path can not be guaranteed. The two step planner presented in [26] accounts for the kinodynamic constraints by computing velocity and acceleration references (based on the vehicle limitations) at key points along the initial collision-free path. These velocity and acceleration references are then used as feed-forward inputs to a tracking controller, which generates control inputs that allow the vehicle to traverse the path. The main drawback of this method is that it does not account for the deviations that will result when the vehicle transitions from one path segment to the next. Thus, it is necessary for the vehicle to slow down significantly in order to traverse corners safely. The proposed method of transforming the path using motion primitives draws inspiration from [26], but aims to mitigate the limitations mentioned above.

2.2 The Probabilistic Roadmap Planner

Given an environment $W \in \mathbb{R}^3$ that contains a set of obstacles $O \subseteq W$ and a robot with n degrees of freedom represented by $X = \{x_1, x_2, ..., x_n\} \in \mathbb{R}^n$, the configuration space $\mathcal{C} \in \mathbb{R}^n$ is the space spanned by X. The untraversable regions within $\mathcal{C}$ can then be defined as $\mathcal{C}_{obstacle} = \{X \in \mathcal{C} : (X \bigcap O) \bigcup (x_i \notin [x_{i_{min}}, x_{i_{max}}])\}$, where $[x_{i_{min}}, x_{i_{max}}]$ is the achievable range for state x_i as defined by the boundaries of W and the vehicle constraints. The traversable free space within $\mathcal{C}$ will therefore be $\mathcal{C}_{free} = \mathcal{C} \setminus \mathcal{C}_{obstacle}$. The objective of the motion planning problem is to find a path that lies entirely within $\mathcal{C}_{free}$. The reason that the planning is done in $\mathcal{C}$, instead of W, is to account for the volume and pose of the robot when checking for potential collisions.

The PRM algorithm that is used to generate the initial collision-free path consists of two parts. The first part constructs the roadmap (or graph), which attempts to approximate $\mathcal{C}_{free}$, through random sampling. The second part appends the starting and ending locations to the roadmap and queries for a solution path connecting them. This allows for multiple queries with different starting and ending configurations to be performed on a single roadmap. For single query applications, the starting and ending configurations can be directly added to the roadmap during initialization. The details of the PRM algorithm are presented in Algorithm 1, where the nodes and edges in the roadmap are represented by the sets S and E respectively, $\overline{N}$ is the maximum number of samples, $\sim U[\mathcal{C}]$ denotes a sample being drawn from an uniform distribution spanning the space $\mathcal{C}$, e_{ij} is the edge connecting nodes s_i and s_j, and s_s and s_e are the start and end configurations respectively. Note that A* search can be used in place of Dijkstra's algorithm in Algorithm 1.

There are many modifications that can be made to the PRM algorithm in order to increase efficiency. One of the most computationally expensive operations of the algorithm is performing collision checks on each potential edge. Since a fully connected graph is not necessary to obtain a path, it is common practice to only connect each node to its n closest neighbors, where n becomes a parameter used to tune the algorithm. The idea is that the closer two nodes are to each other, the higher the chances that they can be connected with a collision-free edge. However, there are several trade-offs to this approach. For instance, the resulting path is more likely to be comprised of many shorter segments when fewer longer segments may have sufficed. As well, if the nodes happen to be naturally clustered, connectivity of the graph may suffer since the nodes will only connect to the closest neighbors which lie in the same cluster. It is necessary for the PRM graph to be connected in order to guarantee that a path will be found during the query phase. However, creating a connected graph can be quite challenging in the presence of cluttered obstacles and narrow passages. It has been shown that as the number of uniform samples

Algorithm 1 PRM Algorithm

```
1: $N = 0$, $S = \emptyset$, $E = \emptyset$
2: while $N < \overline{N}$ do
3:     Select $s \sim U[\mathcal{C}]$
4:     if $s \in \mathcal{C}_{free}$ then
5:         $S = S \bigcup s$
6:         $N = N + 1$
7:     end if
8: end while
9: for all pairwise distinct $s_i \in S$ and $s_j \in S$ do
10:     if $e_{ij} \subseteq \mathcal{C}_{free}$ then
11:         $E = E \bigcup e_{ij}$
12:     end if
13: end for
14: $S = S \bigcup s_s \bigcup s_e$
15: for all $s_i \in S$ do
16:     if $e_{si} \subseteq \mathcal{C}_{free}$ then
17:         $E = E \bigcup e_{si}$
18:     end if
19:     if $e_{ei} \subseteq \mathcal{C}_{free}$ then
20:         $E = E \bigcup e_{ei}$
21:     end if
22: end for
23: path = Dijkstra's Algorithm($s_s$, $s_e$, $S$, $E$)
```

tends to infinity, the probability that the graph will be connected and a path (assuming one exists) will be found tends to 1. This is concept is known as *probabilistic completeness*. Unfortunately, increasing the number of uniform samples until the graph is connected will introduce many redundant samples in expansive regions and lead to an overall decrease in efficiency. Therefore, various sampling strategies have been proposed to bias the samples towards critical areas that are likely to cause connectivity issues. These sampling techniques will be examined in detail in Chapter 4.

2.3 Heuristic Re-sampling Algorithm

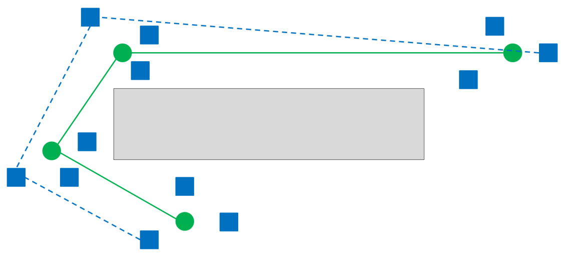
As previously mentioned, the paths generated by PRM techniques often circuitous (especially in 3D environments using the maximum neighbors constraint) and can venture unnecessarily close to obstacles. Thus, before attempting to transform the path to satisfy kinodynamic vehicle constraints, it is advantageous to mitigate these effects so the control effort require for traversal will be minimized later on. To accomplish this, a novel greedy heuristic algorithm will be used to iteratively maximize the clearance around the path while removing redundant segments.

The heuristic function used for this algorithm is the average minimum distance of each node (in the path) to its nearest obstacle. By maximizing this heuristic, the algorithm attempts to maximize the clearance around the path. Given a path defined by the set of nodes S_p, this can be represented mathematically as

$$H = \frac{1}{N_p} \sum_{i=1}^{N_p} \min(D(s_i, O), D_{max}) \tag{2.1}$$

where N_p is the number of nodes in S_p, $D : \mathbb{R}^3 \times O \rightarrow \mathbb{R}_+$ is a function that returns the minimum distance between a node $s_i \in S_p$ and the obstacles in O, and D_{max} is the maximum contribution that a node can make to the heuristic function. The purpose of this D_{max} term is to prevent unnecessarily wide paths in expansive environments, where the nodes can be continuously shifted further and further from the obstacles. Note that this heuristic only considers the distance to nearest obstacle at the nodes of the path, so it is possible that the middle sections of certain path segments remain close to obstacles. The heuristic function can be continuously refined by including increasing bisections of the each path segment in the calculations, however there will be a trade-off in computational efficiency. As well, this heuristic function favors cluttered or indoor environments where there are obstacles enclosing the obstacle.

To shift the path away from obstacles, a local re-sampling is performed at each node in the path. For each node $s_i \in S_p$, a set of new nodes S_n is generated using uniform sampling within a ball of radius r around s_i. Then, the set of candidate nodes for the modified path at s_i will be the set $\overline{S}_i = s_i \bigcup S_n$. Starting at the first node in the path, the value of $D(\overline{s}, O)$ is calculated for each $\overline{s} \in \overline{S}_i$. Using a greedy approach, the algorithm attempts to replace s_i with $\overline{s}_{max} \in \overline{S}$, where $\overline{s}_{max}$ is the node with the maximum $D(\overline{s}, O)$ value in S_i. To ensure the path remains collision-free, $\overline{s}_{max}$ is connected to nodes s_{i-1} and s_{i+1} in the current path. If a collision results, the current $\overline{s}_{max}$ is removed from $\overline{S}_i$, and a new $\overline{s}_{max}$ is selected from the remaining candidates. This process is repeated for each node along the path until the destination is reached. In the worst case scenario, the algorithm will return the initial path since it is collision-free and its nodes are also included in the selection process. The effect of this re-sampling technique is illustrated in Figure 2.1(a), where the initial path is shown in green and the shifted path is shown in blue. Next, the algorithm removes redundant segments from the path. Starting at the first node, the algorithm attempts to connect s_i to s_{i+2}. If the resulting edge is collision-free, then s_{i+1} is redundant and removed from the path. This step is repeated until it is no longer possible to remove s_{i+1} without introducing a collision in the path. The algorithm repeats this process for the subsequent nodes in the path until the end of the path is reached. The effects of this process is shown in Figure 2.1(b).The algorithm will repeat the two steps described above until the improvements in the heuristic function stay less than some specified threshold ε for several consecutive iterations.


(a) Shifting path away from obstacles.

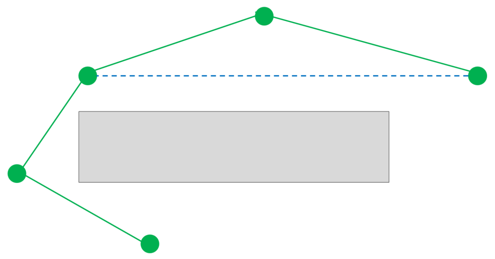

(b) Removing unnecessary segments.

Figure 2.1: Heuristic re-sampling algorithm.

Like many iterative algorithms, premature termination due to local optima can be a large issue for this algorithm. Even though it is not necessary to find the global optimum, it is desirable to get as close to it as possible. Since the algorithm is heuristic based and iterative, simulated annealing [35] is a readily applicable technique that can be incorporated to reduce these issues.

In traditional simulated annealing algorithms, the energy function determines the probability of accepting a less desirable solution at each iteration. The motivation is to allow the algorithm access to a large region of the search space initially, and slowly decrease the size of the search as the solution is refined. This same idea can be applied to the presented algorithm with one important difference: the algorithm will remain greedy, and the energy function will instead determine the re-sampling radius r around each node. Thus, the nodes in S_n are allowed to span a large space around s_i during the initial re-sampling steps. Then, as the energy decreases with each iteration, the nodes in S_n will be restricted to a smaller and smaller neighborhood of s_i until they converge upon a solution. Furthermore, by tuning the rate at which r decreases, it is possible to control the rate at which the algorithm converges. Even though forcing a faster convergence will tend to produce less optimal solutions, it may be necessary to meet efficiency requirements. This implementation is quite similar to the physical annealing process, where each path node is like a metal particle that has high mobility at high temperatures, and as the temperature cools these particles have lower and lower mobility until they stop moving and form the final solution.

2.4 Kinodynamically Feasible Motion Primitives

The transformation algorithm presented in this section will be used to convert the PRM path into a dynamically feasible trajectory using kinematic motion primitives. It is important to note that any piece-wise linear path can be traversed by vehicles with zero turning radius, such as a rotorcraft UAVs and skid-steer UGVs, without violating kinodynamic constraints. To do so, the vehicle simply has to stop at the end of each segment and rotate to align itself with the upcoming path segment before proceeding onwards. However, traversing paths in such a manner is extremely inefficient and should only be used as a last resort.

2.4.1 Algorithm Overview

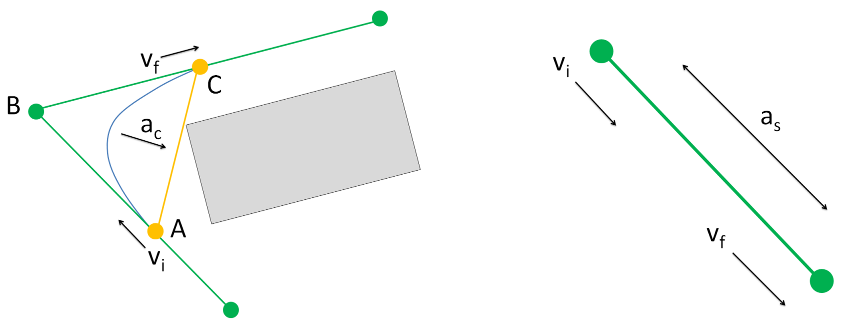
The transformation algorithm divides the path into a series of corner and straight segments. This corner segments will be defined as the largest obstacle-free triangle ABC that subtends the corner where the two adjacent PRM path segments intersect as shown in Figure 2.2(a). Since any arbitrary pair of intersecting lines in 3D space lie within a 2D plane, each corner segment is contained within the plane defined by the two intersecting path segments it spans. Given the triangle ABC, the motion primitive for the corner segment can be defined as a constant inward acceleration a_c that allows the vehicle to travel from point A to point C, such that its velocity transitions from v_i at A to v_f at C, subject to the following constraints: [1]

$$\hat{v}_i \cdot \hat{AB} = \hat{v}_f \cdot \hat{BC} = 1 \tag{2.2}$$

$$\frac{||v_f||_2}{||v_i||_2} = \frac{||BC||_2}{||AB||_2} \tag{2.3}$$

The resulting trajectory from point A to point C (curved blue line on Figure 2.2(a)), produced by a_c, v_i, v_f, and constraints (2.2) and (2.3), will be referred to as the *corner motion*. Note that due to constraint (2.3), triangle ABC must be chosen such that the ratio of AB and BC is equal to the ratio of the desired velocities v_i and v_f.

The straight segments will be defined as the sections of the PRM path that connect the corner segments together. Since these segments do not deviate from the PRM path, they will remain collision-free. The motion primitive for straight segment will be a constant acceleration a_s applied tangent to the segment as shown in Figure 2.2(b). The purpose of this acceleration is to ensure that the vehicle is at an appropriate velocity as it enters the subsequent corner segment.



(a) Corner segment.

(b) Straight segment.

Figure 2.2: Motion primitives on straight and corner path segments.

After dividing the path into a series of straight and corner segments, the maximum velocity with which the vehicle can enter and exit each segment without exceeding its dynamic limits can be found. This is done using backward propagation of the velocity limits with a final velocity of zero (i.e. the vehicle comes to a stop as it reaches its

[1]The hat notation denotes the normalized version of a vector. i.e. $\hat{v} = \frac{v}{||v||_2}$

destination). The general procedure for finding the velocity bounds along the path are as follows:

1. Set the v_f of the last straight segment to zero. This is denoted as V_3 in Figure 2.3.
2. The v_i of the last straight segment is bounded by $V_3 + \Delta v$, where Δv is the change in velocity assuming maximum decceleration along the straight segment.
3. Find the maximum v_i of the corner segment (at point A) such that the required a_c to perform the corner motion does not exceed the vehicle limits. This value is denoted as V_c in Figure 2.3.
4. Set the v_f of the corner segment, denoted V_2 in Figure 2.3, as $\mathrm{V}_2 = \min(\mathrm{V}_3{+}\Delta\mathrm{v}, \gamma\mathrm{V}_c)$ where $\gamma = \frac{||BC||_2}{||AB||_2}$. Then, the initial velocity of the corner segment, denoted V_1 in Figure 2.3, will be $\mathrm{V}_1 = \frac{1}{\gamma}\mathrm{V}_2$. Note that to simplify the implementation of the algorithm, the additional constraint $\gamma = 1$ will be imposed for all corners. By constraint (2.3), this restricts the initial and final velocities of each corner segment to be equal in magnitude, and all obstacle-free triangles ABC to be isosceles with $AB = BC$.
5. Repeat backwards along the path by setting the v_f of the next straight segment as the new v_i of the corner segment. Hence, V_1 is both the v_f of the first straight segment and v_i of the corner segment in Figure 2.3.

As long as the velocity of the vehicle does not exceed the calculated velocity bounds at each point, it will be able to track the path while satisfying its kinodynamic constraints. Furthermore, if the vehicle velocity is within the velocity bound at any point in the path, the vehicle will be capable of satisfying all subsequent velocity bounds without violating kinodynamic constraints. Thus, any set of a_c and a_s accelerations that does not cause the vehicle velocity to surpass the calculated bounds can be used to traverse the path. In the worst case scenario, the corner segments defined by triangle ABC will be a point at B, and the velocity bounds at these corners will be zero. In such cases, the vehicle can still traverse the path by coming to a stop before continuing onto the next segment (i.e. the naive method of traversing piece-wise linear paths mentioned earlier).

This algorithm also allows for the path to be extended without having to recalculate all the velocity bounds. This is useful for applications where multi-stage path planning is required. To extend the path, begin the new path at the penultimate node of the current path such that the two paths will have one straight segment overlapping (Figure 2.4).

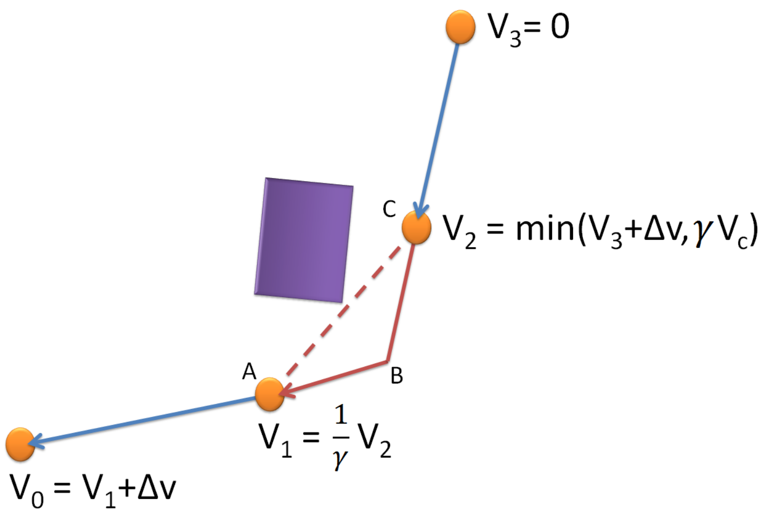

$V_3 = 0$
C
$V_2 = \min(V_3+\Delta v, \gamma V_c)$
A
B
$V_1 = \frac{1}{\gamma} V_2$
$V_0 = V_1+\Delta v$

Figure 2.3: Backward propagation of velocity bounds.

Then, the velocity bounds for the new path can be calculated independently of the current path. As the vehicle reaches the penultimate node of the current path, it will simply choose acceleration inputs based on the velocity bounds of the new path instead of decelerating to a stop. Barring the case where the newly formed corner segment at the end of the current path does not exist (i.e. is a point), the vehicle will be able to transition onto the new path without having to stop.

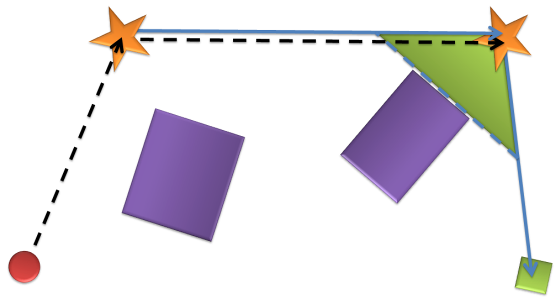
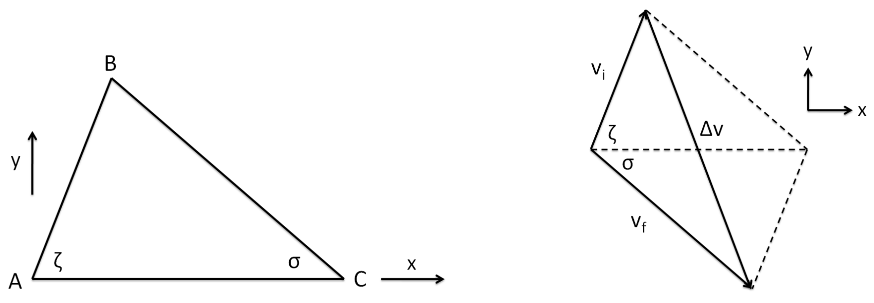


Figure 2.4: Extending the current path. The dashed black line is the current path, the solid blue line is the new path, the overlapping segment lies between the orange stars, and the shaded green triangle marks the newly formed corner segment.

2.4.2 Corner Motion: Feasible and Bounded

For the transformation algorithm to be effective, collision-free corner motions must exist for any arbitrary piece-wise linear path. Thus, it is necessary to prove that as long as constraints (2.2) and (2.3) are satisfied, there will always exist a corner motion that is bounded by the obstacle-free triangle ABC.

Without loss of generality, a local frame will be assigned to triangle ABC such that the base, AC, lies along the positive x axis and the height lies along the positive y axis as shown in Figure 2.5(a). The corresponding velocity diagram for the corner motion, which shows the transition of the velocity vector from v_i to v_f during time t_t, is presented in Figure 2.5(b) in the same local frame. The interior angle between AB and AC is represented by ζ, and the interior angle between CA and CB is represented by σ.



(a) Triangle ABC in local frame. (b) Corner motion velocity diagram in local frame.

Figure 2.5: Position and velocity diagrams of corner motion in assigned local frame.

Theorem 1. *For any arbitrary triangle ABC and velocities v_i and v_f that satisfy constraints* (2.2) *and* (2.3)*, there exists a unique constant acceleration, a, that produces a corner motion from point A to point C.*

Proof of Theorem 1. The displacement of the vehicle, Δp, as a function of time, t, during the corner motion is

$$\Delta p(t) = v_i t + \frac{1}{2} a t^2 = v_i t + \frac{1}{2}\left(\frac{v_f - v_i}{t_t}\right) t^2 \tag{2.4}$$

For the corner motion to exist between point A and C, the final displacement after time t_t must equal AC. Based on the local frame, this means the final displacement must contain only a positive component along the x axis. The final displacement can be expressed as

$$\Delta p(t_t) = v_i t_t + \frac{1}{2}\left(\frac{v_f - v_i}{t_t}\right) t_t^2 = \frac{1}{2}(v_f + v_i)t_t \tag{2.5}$$

Equation (2.5) can be split into $\hat{x}$ and $\hat{y}$ components. The displacement along $\hat{x}$ can be expressed as:

$$\Delta p_x(t_t) = \frac{1}{2}(v_f \cos\sigma + v_i \cos\zeta)t_t = \eta t_t \tag{2.6}$$

where η is a positive constant equal to $\frac{1}{2}(v_f \cos\sigma + v_i \cos\zeta)$. Note that η is positive for all values of ζ and σ because the constraints dictate that the vectors AC and $v_f + v_i$ have the same direction, which by choice of the local frame has a positive x component. Similarly, the displacement along $\hat{y}$ can be expressed as:

$$\Delta p_y(t_t) = \frac{1}{2}(v_f \sin\sigma + v_i \sin\zeta)t_t = \frac{1}{2}(v_f \sin\sigma - v_f \sin\sigma)t_t = 0 \tag{2.7}$$

since $v_i \sin\zeta = -v_f \sin\sigma$ due to constraint (2.3). Therefore, the final displacement after any arbitrary t_t will only have components along the x axis of the local frame and can be entirely represented by Equation (2.6). Since the required final displacement $\Delta p(t_t) = \Delta p_x(t_t) = AC$ and the known constant η are both positive, there exists an unique positive time t_t that will satisfy Equation (2.6). Thus, for any triangle ABC and velocities v_i and v_f satisfying constraints (2.2) and (2.3), there will exist an a that allows the vehicle to travel from point A to C, while its velocity transitions from v_i to v_f, in time t_t. □

Theorem 2. *For any arbitrary triangle ABC and velocities v_i and v_f that satisfy constraints* (2.2) *and* (2.3)*, the corner motion from point A to point C will be completely contained within triangle ABC.*

Proof of Theorem 2. First, it is important to note the following two points:

1. The corner motion connecting point A and C is part of a smooth function that is defined by twice integrating the constant acceleration a. Thus, the path segment is continuous.

2. At any point of an arbitrary path that is at least twice differentiable, the gradient of the path and instantaneous velocity will have the same direction.

Since the velocity of the vehicle must transition from v_i to v_f at a constant rate equal to the acceleration a during the corner motion, the gradient direction of the path segment must transition from $\hat{v}_i$ to $\hat{v}_f$ at the same rate. This means that there exists a time $t_{\frac{1}{2}}$ such that the direction of the gradient at a point $p_1 = \{(A + \Delta p(t)) \in \mathbb{R}^3 : 0 \leq t < t_{\frac{1}{2}}\}$ lie in Region 2 on Figure 2.6, and the direction of the gradient at a point $p_2 = \{(A + \Delta p(t)) \in \mathbb{R}^3 : t_{\frac{1}{2}} < t \leq t_t\}$ lie in Region 3 on Figure 2.6. From the proof of Theorem 1, it is possible to show that $t_{\frac{1}{2}} = \frac{t_t}{2}$.

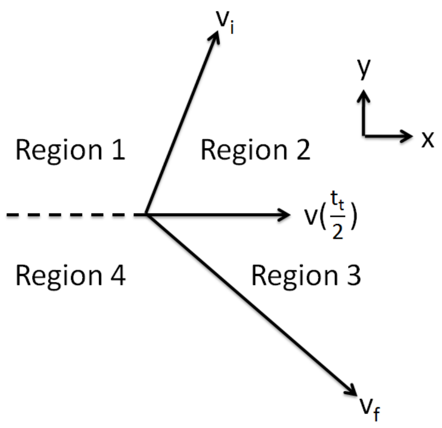


Figure 2.6: Velocity/gradient directions during a corner motion.

The corner motion will not exit triangle ABC via side AB. Since the corner motion starts at point A, it is necessary for at least one point on the path segment to have a gradient direction within Region 1 of Figure 2.6 in order to cross outside triangle ABC through side AB. However, no such velocity, and hence gradient, direction exists during the corner motion. Thus, side AB successfully bounds the corner motion.

Using the same logic as above, the corner motion can not exit triangle ABC through side AC when $t \in [0, t_{\frac{1}{2}}]$ since the gradient directions lie in Region 2 of Figure 2.6. Now assume the vehicle reaches a point outside of triangle ABC via side AC when $t \in (t_{\frac{1}{2}}, t_t]$. From such a point, a gradient direction with a positive y component (Region 1 and 2 of Figure 2.6) is needed on at least one point of the remaining path segment in order for the vehicle to reach point C. However, this is not possible because the gradient direction must stay in Region 3 for all $t \in (t_{\frac{1}{2}}, t_t]$. By design, the vehicle must reach point C at the end of the corner motion, thus it is not possible for the path segment to exit triangle ABC through side AC.

Similarly, assume the corner motion reaches a point outside triangle ABC via side BC at some t. From such a point, a gradient direction within Region 4 of Figure 2.6 will be needed on at least one point of the remaining path segment for the vehicle to reach point C as required by design. However, the gradient direction stays within Regions 2 and 3 for all t. Thus by contradiction, the path segment must not exit triangle ABC through side BC.

Since the corner motion is bounded at all points by sides AB, AC, and BC, it must be entirely contained within triangle ABC. □

2.4.3 Velocity Bound Calculations

The general approach for calculating the velocity bounds will be presented in this section. The calculations will assume that the maximum achievable acceleration along any vector in 3D space can be derived from the vehicle model, and that the necessary control inputs to achieve these accelerations can be found.

Acceleration and Velocities along a Straight Segment

Given two adjacent nodes s_i and s_j, the goal is to find the maximum feasible acceleration along e_{ij}. First calculate the unit vector $\hat{e}_{ij} \in \mathbb{R}^3$ corresponding to e_{ij}, using

$$\hat{e}_{ij} = \frac{s_j - s_i}{||s_j - s_i||_2} \tag{2.8}$$

where $s_i, s_j \in S$. Then, the set of dynamically feasible accelerations that lie entirely along $\hat{e}_{ij}$ will be defined as

$$A_{\hat{e}_{ij}} = \{a \in A : a \cdot \hat{e}_{ij} = a\} \tag{2.9}$$

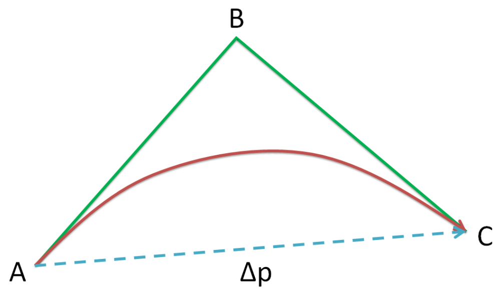
Thus, the maximum acceleration along $\hat{e}_{ij}$ will be

$$A_{\hat{e}_{ij}max} = \max\{||a||_2 : a \in A_{\hat{e}_{ij}}\} \tag{2.10}$$

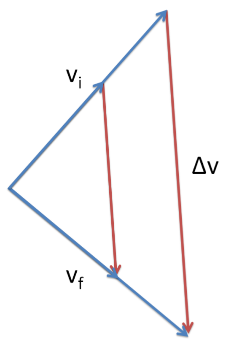
Since acceleration along $\hat{e}_{ij}$ is equivalent to deceleration along $\hat{e}_{ji}$, once the maximum velocity at node s_i or s_j is known, the maximum velocity at the other node can be found.

Acceleration and Velocities at a Corner Segment

At each corner segment of the path, the goal is to solve for the maximum velocity the vehicle can enter and exit the corner, such that the acceleration required to complete the turn is feasible. Since v_i of the corner segment is equal to the v_f of the previous straight segment, and the v_f of the corner segment is equal to v_i of the upcoming straight segment, the directions of v_i and v_f of the corner segment are known. For instance, the directions of v_i and v_f for the corner depicted in Figure 2.7(a) will correspond to the directions of vectors AB and BC respectively. Furthermore, since the magnitude of v_i and v_f are related by the constant γ, the acceleration that produces the necessary change in velocity, $\Delta v \in \mathbb{R}^3$, will have a constant direction independent of the actual magnitude of the velocities (Figure 2.7(b)). Finally, as the vehicle's velocity reaches the desired v_f, its position must coincide with the end of the corner segment (point C in Figure 2.7(a)). Fortunately, since both the entry and exit positions are known, this can be accomplished by setting the total displacement of the vehicle during the turn, $\Delta p \in \mathbb{R}^3$ in Figure 2.7(a), to be equal to the difference between the two positions.



(a) Vehicle displacement, Δp, during a corner motion.



(b) Velocity vector diagram. The direction of Δv depends soley on γ.

Figure 2.7: Change in position and velocity at a corner segment.

The first step is to solve for the maximum achievable acceleration, a, in the direction of the vector Δv. This is accomplished in the same manner as finding the maximum acceleration along a straight path segment. The unknowns at this point are the magnitudes of v_i and v_f which are related by the known constant γ. To simplify the calculations, $\gamma = 1$ will be imposed so that the magnitude of v_i and v_f are equal. The relationship between these variables is described by

$$\Delta p = v_i t + \frac{1}{2}at^2 \tag{2.11}$$

where the time variable, t, can be expressed as

$$t = \frac{\Delta v}{a} = \frac{v_f - v_i}{a} = \frac{\alpha \hat{v}_f - \alpha \hat{v}_i}{a} = \frac{\alpha v_d}{a} \tag{2.12}$$

Here, the initial and final velocities are divided into magnitude α (scalar) and its unit vector direction $\hat{v}_f, \hat{v}_i \in \mathbb{R}^3$, and $v_d \in \mathbb{R}^3$ represents the vector difference between $\hat{v}_i$ and $\hat{v}_f$ (note that v_d is not a unit vector and that $\Delta v = \alpha v_d$). Finally, substituting Equation (2.12) into Equation (2.11) produces

$$\Delta p = \alpha \hat{v}_i \left(\frac{\alpha v_d}{a}\right) + \frac{1}{2} a \left(\frac{\alpha v_d}{a}\right)^2 = \frac{1}{a}\left(\hat{v}_i \cdot v_d + \frac{1}{2} v_d \cdot v_d\right) \alpha^2 \tag{2.13}$$

Since all other variables are known, it is possible to solve for α (and thus v_i and v_f) using

$$\alpha = \sqrt{\frac{\Delta p \cdot a}{\hat{v}_i \cdot v_d + \frac{1}{2} v_d \cdot v_d}} \tag{2.14}$$

2.4.4 Modified Velocity Bound Calculations with Acceleration Transients

The velocity bounds calculated previously assume that the vehicle is capable of changing its acceleration instantaneously as it transitions from one segment to the next. Unfortunately, this is generally not the case and for vehicles with slow dynamics the effects of these transients can be significant. Thus, the motion primitives will be modified using the assumption that the vehicle accelerations undergo linear transitions. In order to maintain consistency between all path segments, the vehicle should not directly transition from one segment's required acceleration to the next. Instead, at the start and end of each segment the vehicle will transition from and to a zero acceleration state respectively. This results in an acceleration profile with three regions per motion primitive as shown in Figure 2.8. It is assumed that the vehicle requires the same amount of time, t_o, to transition to the desired acceleration (Region A) and to transition back to a zero acceleration state (Region C). This implies that the linear rate at which the vehicle acceleration changes are equal (i.e. $|K| = |-K|$). In region B, the quadrotor has reached its desired acceleration and maintains it for the duration of t_s.

Velocities on Straight Segments

On straight segments, the key criterion is that the vehicle must return to a zero acceleration state at the end of the segment. This means that the vehicle must undergo a displacement

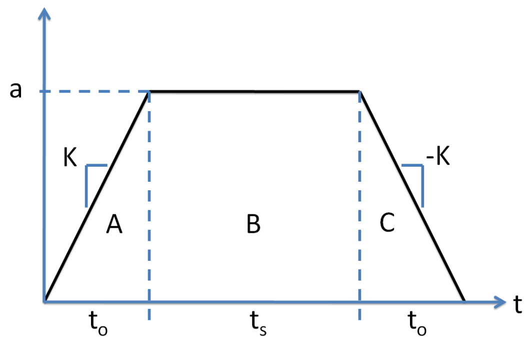


Figure 2.8: Vehicle acceleration during a motion primitive assuming linear acceleration transients.

of Δp in $2t_o + t_s$ seconds while undergoing the accelerations depicted in Figure 2.8. The goal is to find the change in velocity, Δv, that occurs during the straight segment.

First, the maximum feasible acceleration, a, tangent to the straight segment is derived using the process described in Section 2.4.3. This will be the target acceleration that is sustained during region B. Depending on the vehicle model, it should be possible to calculate or approximate either K or t_o. For instance, if the acceleration transient is modelled using a first order response governed by a time constant, t_o can be approximated given a. Conversely, if the acceleration transient is independent of a, it should be possible to calculate or approximate K. Then, the remaining variable can be solved using the relationship $a = Kt_o$.

Next, the displacement of the vehicle during each acceleration region can be calculated using the following equations

$$\Delta p_A = \frac{1}{6}Kt_o^3 + v_i t_o \tag{2.15}$$

$$\Delta p_B = \frac{1}{2}Kt_o t_s^2 + \left(v_i + \frac{1}{2}Kt_o^2\right) t_s \tag{2.16}$$

$$\Delta p_C = \frac{1}{3}Kt_o^3 + \left(v_i + \frac{1}{2}Kt_o^2 + Kt_o t_s\right) t_o \tag{2.17}$$

for regions A, B, and C respectively. Thus, the total displacement of the vehicle over the entire acceleration curve (after simplification) will be

$$\Delta p = \frac{1}{2}Kt_o^3 + \frac{1}{2}Kt_o t_s^2 + v_i(2t_o + t_s) + \frac{1}{2}Kt_o^2(t_o + t_s) + Kt_o^2 t_s \tag{2.18}$$

The total change in velocity, Δv, is simply the area under the acceleration curve given by

$$\Delta v = Kt_o^2 + Kt_o t_s \tag{2.19}$$

However, both Δv and t_s are unknown in Equation (2.19). So solving for t_s and substituting the resulting expression into Equation (2.18) yields (after simplification)

$$\Delta v^2 + (Kt_o^2 + 2v_i) \cdot \Delta v + 2K(v_i t_o - \Delta p)t_o = 0 \tag{2.20}$$

which can be solved to obtain Δv. Note that this Equation (2.20) is a quadratic and will yield two solutions for Δv. It is important to use the solution that corresponds real and non-negative values of t_o and t_s. If such a solution does not exist, it means that the path segment is too short and the desired a is not attainable within the given Δp. In such cases, the above calculations should be repeated using a new target acceleration, $a_{new} = \beta a$ where $\beta \in (0, 1)$. The final velocity of the vehicle at the end of the straight segment will therefore be $v_f = v_i + \Delta v$.

Velocities on Corner Segments

To find the v_i and v_f at the corner segments, Δp, $\hat{v}_i$, $\hat{v}_f$, and v_d must first be found using the method described in Section 2.4.3. Then, the maximum feasible acceleration, a, tangent to v_d should be derived from the vehicle model. This allows the corresponding values for K and t_o to be calculated in the same manner as straight segments. Next, rewriting the velocity vectors in Equation (2.20) as a magnitude, α, and a corresponding unit vector, and rearranging to solve for α yields

$$(2\hat{v}_i + v_d) \cdot v_d \alpha^2 + Kt_o(2\hat{v}_i + t_o v_d)\alpha - 2K\Delta p t_o = 0 \tag{2.21}$$

This equation is also a quadratic, so the solution of interest will again correspond to real and non-negative values for t_o and t_s. If such a solution does not exist, the target acceleration should be decreased by setting $a_{new} = \beta a$, where $\beta \in (0, 1)$, as was done for straight segments.

2.5 Simulation Results

The following simulations were all performed on a Lenovo W500 Thinkpad with a 2.80 GHz Intel Core 2 Duo Processor (T9600) and 4.00 GB of RAM. The Panda3D engine is used for visualization, and the Open Dynamics Engine is used for collision detection within the simulated environments.

2.5.1 Heuristic Re-sampling Algorithm

In simulations, the heuristic re-sampling algorithm succeeded in shifting the path away from obstacles and managed to remove most of the redundant segments in the path. The effects of this algorithm can be clearly seen in Figure 2.9, where the initial PRM path is shown in yellow and the modified path is shown in light blue. As shown in the top view, Figure 2.9 (right), many of the smaller segments in the path have been replaced with longer, more direct segments. As a result, fewer motion primitives are needed to traverse the path, which reduces both the complexity of the transformation algorithm and the control effort required (i.e. there are fewer acceleration transitions). As well, the increased clearance means that the obstacle-free *ABC* triangles at each corner segment can be made larger, thus allowing the vehicle to safely deviate further from the path during corner motions. This results in higher velocity bounds and thus faster traversals.

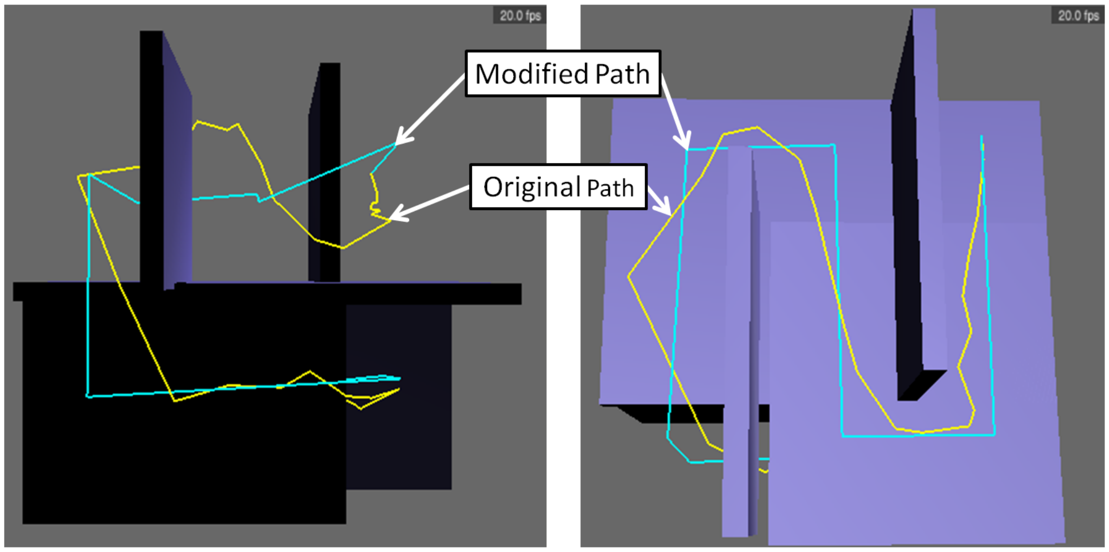


Figure 2.9: Effects of the heuristic re-sampling algorithm. Side view (left) and top view (right).

For the particular example shown in Figure 2.9, the average minimum distance between each node and the nearest obstacle increased from 0.76 m to 0.88 m. As a consequence of shifting the path away from obstacles, the total length of the modified path did increase by about 9% (from 48.37 m to 52.85 m). However, this is somewhat mitigated by the fact that the vehicle is able to traverse the path more quickly. In this case, the algorithm terminated after four iterations resulting in a total runtime of 0.68 s.

Table 2.1: Simulation Results of the Heuristic Re-sampling Algorithm

	Without SA		With SA		
Initial Sampling Radius (m)	1.0	5.0	5.0	5.0	5.0
Decay Rate (%)	N/A	N/A	25.0	50.0	75.0
Runtime (s)*	0.72	0.11	1.54	1.28	0.78
Iterations*	7.05	5.17	15.31	10.89	5.80
Runs with Optimal Number of Waypoints (%)	69.9	11.2	91.3	91.5	64.3
Heuristic (m)*	0.87	0.73	0.96	0.96	0.90

* These values are averages over 1000 simulations.

In Section 2.3, it was proposed that principles of simulated annealing be incorporated into the heuristic re-sampling algorithm to help escape local optima and reduce premature termination. In simulation, the performance of the heuristic algorithm with and without simulated annealing was compared by executing each version of the algorithm 1000 times for the problem shown in Figure 2.9. The results are summarized in Table 2.1. The *decay rate* is the percentage by which the sampling radius decreases each iteration during simulated annealing. The *runs with optimal number of waypoints* is the percentage of total runs that yielded a path with the same number of nodes as the globally optimal path. For this particular environment, the globally optimal path has 12 nodes (one in each corner of the hallway). Therefore, solutions with more nodes will contain unnecessary segments, while solutions with less will have sections with only one node per two corners resulting in the path cutting very close to the wall at certain points. In these simulations, both versions of the algorithm were set to terminate if the heuristic value improves by less than an ε of 0.002 m for three consecutive iterations.

From these results, it is evident that a sampling radius of 1.0 m generates better solutions than a sampling radius of 5.0 m when simulated annealing is not used. Compared to a sampling radius of 1.0 m, a 5.0 m sampling radius produced solutions with 16% lower heuristic values (on average) and was 59% less likely to find solution with the optimal number of waypoints. Since the algorithm is restricted to the set of solutions contained within the search space, a larger sampling radius should allow for higher quality solutions. However, it is also harder to find these quality solutions because there is a larger set of solutions to search through. In this case, the algorithm with the 5.0 m sampling radius is terminating pre-maturely because it is unable to find sufficient improvements to the path within three iterations. Adjusting the termination criterion to allow for more iterations is not sufficient to solve this problem, because doing so will lead to an increase in runtime and the same issue will still be present for larger sampling radii.

By including simulated annealing, the heuristic re-sampling algorithm is able to generate better solutions more reliably. Comparing the solutions obtained using simulated annealing (using a 5.0 m sampling radius and a 50% decay rate) to those obtained without simulated annealing (using a 1.0 m sampling radius), the solutions found using simulated annealing have higher heuristic values that are 10% higher (on average) and is 22% more likely to contain the optimal number of waypoints. However, the improved solution quality comes at the cost of increased iterations and runtime, with the simulated annealing solutions taking 78% longer to compute on average. The decay rate is an important parameter and greatly affects the performance of the heuristic re-sampling algorithm when simulated annealing is used. From Table 2.1, it is clear that the runtime of the algorithm is inversely proportional to the decay rate. However, selecting a high decay rate will force the algorithm to converge pre-maturely, thus losing much of the benefits provided by simulated annealing. This can be seen when comparing the solutions generated using decay rates of 50% and 75%. With a 75% decay rate, the average heuristic value of the solutions is 6% lower and the solutions are 27% less likely to contain the optimal number of waypoints. The solution quality and runtimes obtained with a decay rate of 75% is in fact comparable to those obtained without simulated annealing using a sampling radius of 1.0 m. This supports the fact that forcing a rapid convergence will nullify many of the benefits of simulated annealing. However, slowing down the decay rate and allowing the algorithm more time to explore the search space does not guarantee a better solution. This is evident when examining the solutions generated using decay rates of 25% and 50%. The quality of the solutions generated using both decay rates are almost identical, even though the algorithm has a significantly higher runtimes with a decay rate of 25%. It is also interesting to note that selecting a decay rate of 0% will produce results identical to those obtained without simulated annealing (using a 5.0 m sampling radius). This suggests that at a sufficiently small decay rate, the algorithm with simulated annealing will also start to generate poor solutions due to pre-mature termination.

2.5.2 Kinodynamically Feasible Motion Primitives

In this section, the transformation algorithm is applied to a PRM path that has been modified by the heuristic re-sampling algorithm. Since, the transformation algorithm is dependent upon a vehicle model, a simple quadrotor model will be used to calculate the motion primitives. The resulting path is then traversed by a simulated quadrotor with additional aerodynamic drag. A standard PID position controller will be used for disturbance rejection.

Simple Quadrotor Model

The quadrotor vehicle motion is modeled with respect to North, East, Down (NED) inertial coordinates denoted by e_N, e_E, and e_D. The attitude of the vehicle is represented as follows:

- A rotation of angle ψ representing the yaw of the vehicle with respect to the inertial frame. This defines the intermediate frame x_i, y_i, z_i.
- Roll angle ϕ and pitch angle θ both measured with respect to the intermediate frame x_i, y_i, z_i.

Note that ϕ and θ are not successive rotations, but rather variables in the control space that determine the direction of the thrust force [11, 53]. The inertial frame, intermediate frame, and vehicle states are illustrated Figure 2.10. As well, the following assumptions

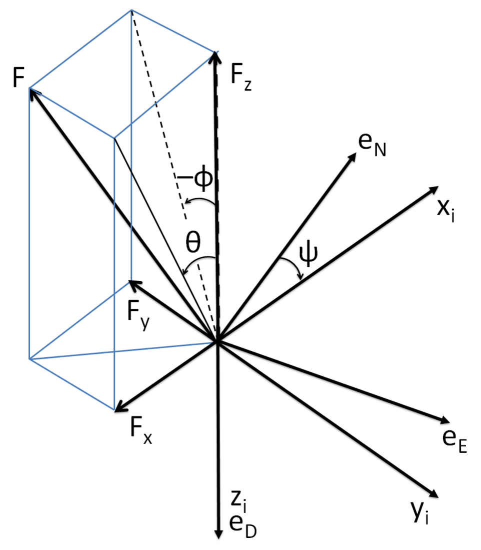


Figure 2.10: Illustration of vehicle attitude and thrust within the intermediate and inertial frames. [53]

will be made with regards to the vehicle model:

1. The quadrotor has a mass of 2.5 kg, a thrust output in the interval of [0 N, 32 N], and sustainable pitch and roll angles in the interval [-30°, +30°].

2. The quadrotor will exhibit holonomic motion (accelerations along x$_i$, y$_i$, and z$_i$ are decoupled and can be controlled independently) within the operational limits stated above.

3. The aerodynamic drag will be modeled as $F_{drag} = C_{drag} v$, where $C_{drag} = 0.1 \frac{Ns}{m}$ and v is the velocity of the vehicle.

4. More advanced aerodynamic effects such as blade flapping [24], wind, and rotor backwash will be ignored.

With these assumptions, the thrust force produced by the four rotors can be combined into a single thrust force F. Given the magnitude of F, $|F|$, and the angles ϕ, θ, and ψ, the resulting forces in the inertial NED frame will be [11]:

$$\begin{bmatrix} F_N \\ F_E \\ F_D \end{bmatrix} = R_{z,\psi}^T \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} \tag{2.22}$$

where

$$\begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} = |F| \cdot \frac{1}{\sqrt{1 - \sin^2\theta \sin^2\phi}} \cdot \begin{bmatrix} -\sin\theta\cos\phi \\ \cos\theta\sin\phi \\ -\cos\theta\cos\phi \end{bmatrix} \tag{2.23}$$

and

$$R_{z,\psi}^T = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.24}$$

Therefore, the resulting accelerations in the inertial frame will be

$$\begin{bmatrix} a_N \\ a_E \\ a_D \end{bmatrix} = \frac{1}{m} \left(\begin{bmatrix} F_N \\ F_E \\ F_D \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} \right) \tag{2.25}$$

Given the desired forces (or the accelerations) in the inertial frame and the angle ψ, the corresponding control inputs ($|F|$, θ, ϕ) can be calculated using the following equations:

$$|F| = \sqrt{F_N^2 + F_E^2 + F_D^2} \tag{2.26}$$

$$\theta = \tan^{-1}\left(\frac{-F_x}{-F_z}\right) \tag{2.27}$$

$$\phi = \tan^{-1}\left(\frac{F_y}{-F_z}\right) \tag{2.28}$$

where

$$\begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} = R_{z,\psi} \begin{bmatrix} F_N \\ F_E \\ F_D \end{bmatrix} \tag{2.29}$$

and

$$R_{z,\psi} = \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.30}$$

Path Generation and Runtime

In developing this path planning solution, one of the key requirements is that the proposed method be computationally efficient enough to plan paths online. To satisfy this criterion, it was decided that the computational time of a path should be less than the time it takes to traverse it. To show that the proposed algorithm meets this criterion, the problem instance illustrated in Figure 2.11 was simulated 200 times. In this particular environment, the shortest paths connecting the starting location to the target destination will take approximately 15 seconds to traverse. The average runtime for 200 simulations of this problem was 1.98s, with the maximum runtime being 4.56s and the minimum runtime being 1.01s. Note that these runtimes include all three phases of the algorithm (i.e. finding the initial PRM path, the heuristic re-sampling algorithm, and the transformation algorithm). The distribution of runtimes is shown in Figure 2.12, and demonstrates the feasibility of real-time planning with this algorithm.

Figure 2.13 shows the proposed algorithm being applied to another environment. In this example, the solution path (yellow), which takes 24.97s to traverse, was computed in 2.83s.

Path Tracking

One of the biggest challenges in using the transformation algorithm is to derive the feasible accelerations and corresponding control inputs from the vehicle model. While more accurate motion primitives can be calculate using detailed vehicle models, the process can quickly become very complex. In many cases, a simplified vehicle model is sufficient for calculating the motion primitives, however over time the vehicle will tend to drift off the

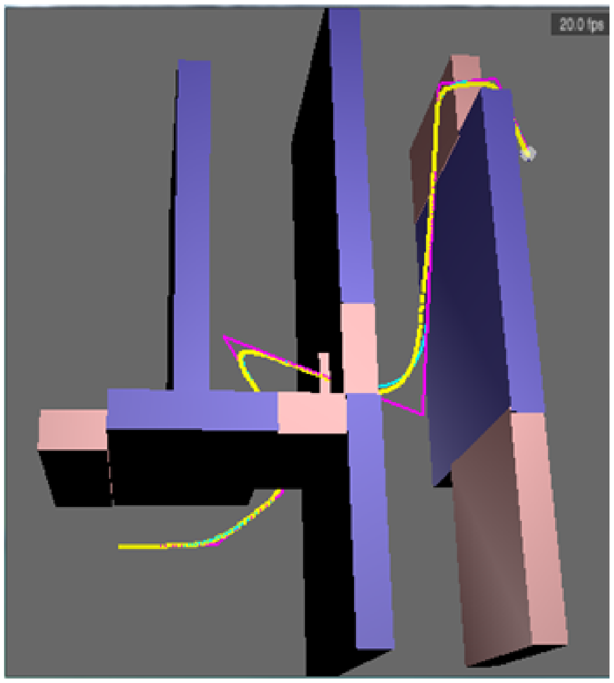


Figure 2.11: Solution path (yellow) through a 3D environment found using the proposed algorithm in 3.29s.

path. To compensate, a standard PID position controller is used to reject the effects of the unaccounted dynamics while the vehicle traverses the path. For large deviations, there is a risk that the control effort exterted by the controller can exceed the vehicle limits, however this was rarely seen in simulation. In practical applications, this may be a greater concern since the controller will have to reject external disturbances as well.

From Figure 2.14, it can be seen that even with the PID controller, the traversed path (dotted yellow) still deviates slightly from the calculated path (light blue). The largest deviations tend to be at the corners; however most of these are still contained within the obstacle-free triangle (as is the case in Figure 2.14), and hence safe. Along the straight segments, a small margin of safety is usually sufficient because the PID controller quickly corrects any deviations. Furthermore, due to the heuristic re-sampling algorithm, the path is generally far enough away from any obstacles that the risks of collision are minimal.

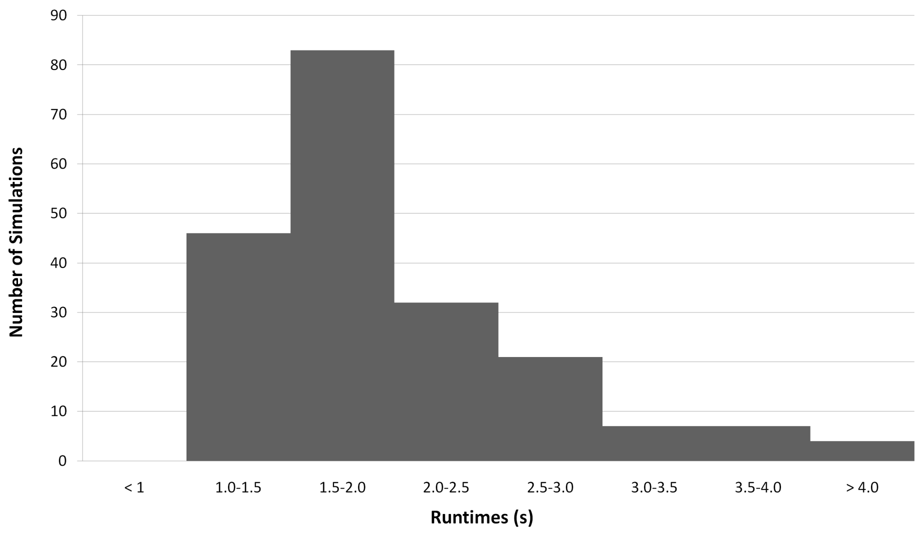


Figure 2.12: Distribution of runtimes for 200 simulations.

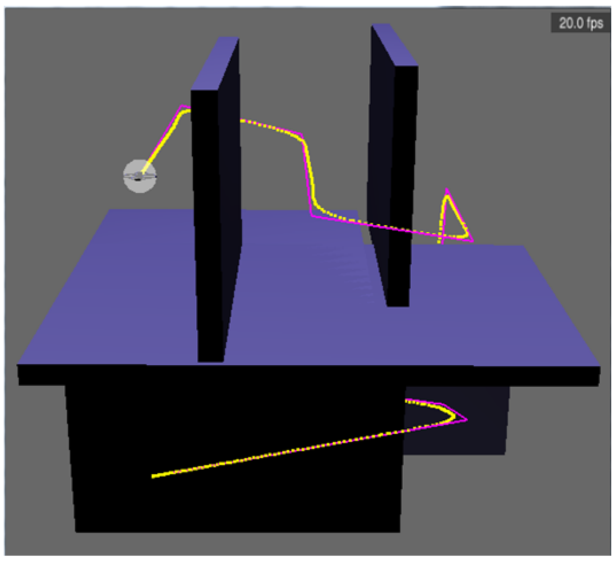


Figure 2.13: Solution path (yellow) through a 3D environment found using the proposed algorithm in 2.83s.

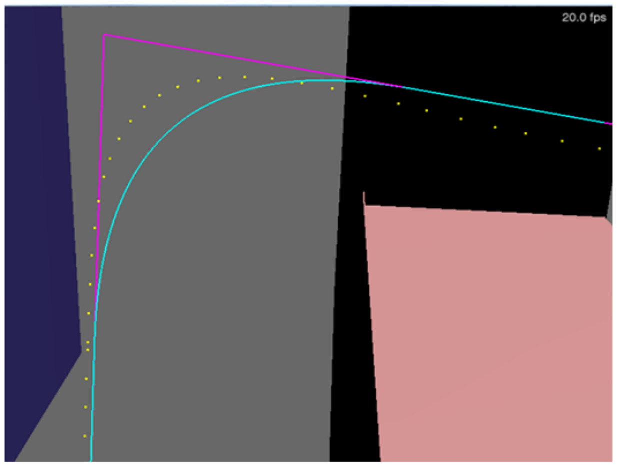


Figure 2.14: The traversed path (dotted yellow) deviates slightly from the calculated path.

Chapter 3

Probabilistic Roadmap Based Planner for Unknown 3D Environments

This chapter will focus on motion planning in unknown static 3D environments while satisfying kinodynamic vehicle constraints. It will be assumed that localization information is available, and that the vehicle has the capability to detect and map any obstacles it encounters during traversal. The proposed solution will use a modified PRM planner to find a collision-free path through the environment based on the partial map constructed by the vehicle. Then, using the transformation algorithm introduced in Chapter 2, this path will be modified to satisfy to any kinodynamic vehicle constraints. As the vehicle uncovers new obstacles while traversing the environment, the algorithm will update the solution path as necessary to ensure that it remains collision-free.

3.1 Background and Theory

The problem of motion planning in an unknown or changing environment has been the subject of much research. This section will introduce some of the existing algorithms that have been proposed to solve this problem.

One of the most basic algorithms for finding a path through an unknown environment is the Bug algorithm [47]. The main idea is to have the vehicle head towards the destination whenever possible, and to use boundary following to circumvent encountered obstacles. In its most basic form, Bug 0, the vehicle simply turns either left or right when an obstacle is detected, and follows the obstacle boundary until it can move towards the goal again. Unfortunately, Bug 0 will fail to find the solution path in certain concave obstacle configurations. In more sophisticated forms, such as Bug 1 or Bug 2, additional mechanisms are included to ensure the vehicle will eventually reach its destination. In Bug 1, the vehicle circumvents the entire obstacle before returning to the closest point to the goal. In Bug 2, the vehicle circumvents the obstacle until it reaches the M-line (straight line from starting location to destination), and if it is unable to further progress towards the destination, it will attempt to circumvent the obstacle in the opposite direction. With these modifications, the Bug algorithms will eventually reach the goal, however with more complex obstacles, the resulting paths are unnecessarily long and redundant since the vehicle will often be forced to circle an obstacle many times before it can progress towards the goal. Several extensions of the Bug algorithm, such as [21] and [31], have been proposed to mitigate some of these issues, but in cluttered environments with complex obstacles, the solution paths still traverse the environment inefficiently.

Fuzzy logic based approaches, such as [63], [39], [43], and [60], have also been widely explored for solving the problem of motion planning in unknown environments. These methods use fuzzy logic to combine various vehicle behaviors such as goal seeking, exploration, and obstacle avoidance to produce aggregated control inputs. The working principle of these fuzzy logic methods is very similar to that of artificial potential fields techniques in that the vehicle's behavior attempts to balance goal seeking and obstacle avoidance depending on the proximity of the vehicle to an obstacle. Unfortunately, like APF techniques, these types of fuzzy logic motion planners suffer from local minima issues that cause the vehicle to become stuck in a region of the environment. Various strategies have been proposed to mitigate these local minima issues in fuzzy logic motion planners. These include the use of virtual targets [63], memory grids [60], and landmark recognition through artificial neural network techniques [39]. However, these strategies can be fairly complex to implement, and even with their inclusion, the fuzzy logic algorithms will often require careful tuning to perform well.

The Intelligent Global Path Planner with Replanning (IGPPR) [51, 50] is an algorithm that is capable of finding collision-free paths through unknown environments while satisfying non-holonomic vehicle constraints. The algorithm discretizes the configuration space of the vehicle, and calculates the heuristic value of each discretized cell by propagating a cost value (corresponding to distance) outwards from the goal location. This is similar to the technique used in level set path planners, however the key difference is that the IGPPR algorithm only propagates the cost to adjacent cells whose configurations are achievable under the kinematic constraints of the vehicle. Then, the solution path can be achieved by performing A* search on the heuristic map from any starting location. When new obstacles are uncovered, any cells newly in collision will be removed from the free space, and the heuristic values of their successor cells will be re-evaluated. A new solution path can then be found by performing A* search on the updated heuristic map. This algorithm has many strengths including the ability to plan under non-holonomic constraints, and efficient replanning upon finding new obstacles. However, one main disadvantage of this method is the need to discretize the configuration space. This introduces discretization error, which dilates obstacles, thus making it more difficult to find connected paths. In extreme cases, the entire problem can become infeasible if the only solution path is invalidated by the dilated obstacles. As well, for larger environments or high dimensional configuration spaces, there will be a severe trade-off between precision and computationally efficiency when choosing the discretization resolution.

Another approach to motion planning in unknown environments is to formulate the problem as a minimum cost flow optimization problem [14]. The idea is to transform the original path planning problem into an equivalent optimization problem in order to leverage existing algorithms. In [14], the configuration space is first discretized to form a graph of nodes and edges. Then, each edge is assigned a cost based on its length and its collision state (i.e. edges that are in collision with obstacles are given high costs and are thus avoided). Finally, the starting node is given a unit supply, the goal node is given a unit demand, and all other nodes have zero supply and demand. To find the solution path, the minimum cost flow problem is solved using the network simplex algorithm which balances the supply and demand of all nodes using the minimum cost possible. This essentially finds the minimum cost path to move the supply from the starting node to satisfy the demand at the goal node. When new obstacles are found, the costs of all affected edges are updated and the solution path is recomputed if necessary. Like the IGPPR algorithm, the algorithm presented in [14] constructs a graph by discretizing the configuration space and will thus have all the associated drawbacks mentioned previously. Fortunately, there is nothing preventing the minimum cost flow approach from being applied to more efficient and robust graph representations of the environment, such as those generated by PRM planners. However, alternative algorithms for finding cost optimal paths through time varying graphs, such as incremental search techniques, will generally be more computationally efficient and will not require the problem to be formulated as an equivalent minimum cost flow optimization problem.

Incremental search methods, including dynamic A* (D*) search [56], D* lite [36], focussed D* search [57], and anytime D* (AD*) search [44], are widely used for motion planning in unknown or changing environments. These algorithms are based on the A* search, but modified to efficiently handle changing costs of nodes and edges. The idea is that when the cost of an edge changes, usually only a subset of nodes are affected, so it is not necessary to redo the entire A* search. Various modifications have been proposed to the basic D* search to help improve its efficiency (focussed D* search) and reduce its complexity (D* lite). One interesting modification is the AD* search which allows the D* search to become an anytime algorithm (i.e. an algorithm that can find a suboptimal solution very fast and then incrementally refine it) by incorporating aspects of anytime A* (ARA*) search [45]. It is important to note that these are graph search algorithms and hence require a graph representation of the configuration space. While it is sufficient to simply discretize the configuration space to obtain the graph (as is done in many sources on incremental search methods), this approach suffers from all the aforementioned discretization issues. More efficient methods of generating the graph, such as PRM planners, have also been suggested. For instance, [5] combines AD* search with the PRM planner to produce an anytime motion planner with replanning capabilities. The main drawback of these incremental search methods is that the performance gain does not always justify the increase in complexity. This is especially true with PRM graphs, which are generally more sparse and contain fewer nodes than graphs formed by discretization, and thus the time required to perform an A* search is negligible even in large, high dimensional spaces.

3.2 Algorithm Definition

This section will present the proposed algorithm for finding kinodynamically feasible paths in unknown environments. First, a simple PRM based algorithm capable of finding piecewise linear paths through unknown environments will be defined. Then, the transformation algorithm described in Chapter 1 will adapted so that it can be applied to solution paths in unknown environments.

3.2.1 Finding Collision-free Paths in Unknown Environments

The algorithm that will be used for finding collision-free paths through the unknown environment will be an extremely simplified version of D* search applied to a PRM based graph. The main idea is to generate a graph representation of the configuration space using PRM techniques, and then use Djikstra's algorithm or A* search to find a feasible path. Then, when newly found obstacles invalidate the solution path, the PRM graph is updated by removing any edges that are in collision, and a new solution path is found by performing a new graph search (from the vehicle's current position) on the updated PRM graph. If the PRM graph is no longer connected, then additional samples will be added.

This approach is inspired by and shares many similarities with lazy collision checking techniques [7] that are often used to increase computational efficiency in PRM planners for known environments.

The detailed algorithm is presented in Algorithm 2, where p is the current solution path, s_c is the current vehicle configuration, and s_e is once again the desired vehicle configuration. When a collision is detected along the current path (Line 4), the PRM graph is updated by removing all edges in collision (Line 5). Then, the vehicle's current configuration is added to the PRM graph as a new node (Lines 6 - 11). This is to allow the graph search, and thus the new solution path, to start from the current configuration. However, if the current configuration is not connected to the destination node through the PRM graph, additional samples are generated and appended to the PRM graph until connectivity is achieved (Lines 12 - 22). Finally, a graph search is performed to obtain the new solution path (Line 23). Note that when adding nodes s_c and s_n to the graph, the number of edges can (and usually should) be restricted to the n closest neighbors for computational efficiency. As well, like in Algorithm 1, the graph search can be performed using A* search instead of Dijkstra's algorithm.

The main advantage of this algorithm is its simplicity in concept and implementation. While incremental search algorithms, such as D* search and its many variations, allow the graph search to be performed more efficiently, the performance gain is very small in practice and can be sacrificed for reduced complexity with little consequences. This is because the bottleneck of the algorithm is caused by collision checking, and in comparison the runtimes of the graph search algorithms are negligible.

3.2.2 Algorithm Extensions for Better Efficiency

The PRM based planner for unknown environments can be made more efficient through the use of local sampling and PRM regeneration.

Local Sampling

When a collision along the solution path is detected in Algorithm 2, the planner attempts to find a new solution path within the PRM graph. If such a path does not exist because the start and end configurations are not connected, then random uniform samples are added to the graph until connectivity occurs. In many cases, when a collision along the path occurs, only a slight detour is needed to reroute the path around the obstacle. However, in the current approach, if these small detours are not part of the PRM graph

Algorithm 2 PRM Based Planner for Unknown Environments

```
1: Perform Algorithm 1 assuming all unexplored areas $\subseteq \mathcal{C}_{free}$
2: Let $p$ = the solution path, $S$ = node set, and $E$ = edge set from Algorithm 1
3: while $s_c \neq s_e$ do
4:    if $p \nsubseteq \mathcal{C}_{free}$ then
5:       $E = E \setminus \{e \in E : e \nsubseteq \mathcal{C}_{free}\}$
6:       for all $s_i \in S$ do
7:          if $e_{ci} \subseteq \mathcal{C}_{free}$ then
8:             $E = E \bigcup e_{ci}$
9:          end if
10:      end for
11:      $S = S \bigcup s_c$
12:      while $s_c$ and $s_e$ are not connected do
13:         select $s_n \sim U[\mathcal{C}]$
14:         if $s_n \in \mathcal{C}_{free}$ then
15:            for all $s_i \in S$ do
16:               if $e_{ni} \subseteq \mathcal{C}_{free}$ then
17:                  $E = E \bigcup e_{ni}$
18:               end if
19:            end for
20:            $S = S \bigcup s_n$
21:         end if
22:      end while
23:      $p$ = Dijkstra's Algorithm $(s_c, s_e, S, E)$
24:   end if
25: end while
```

and another potential path exists, the new solution path may be completely re-routed. This is disadvantageous for two reasons. First, the vehicle is forced to take a larger detour than necessary thus increasing the length of the path. Second, the planner is not fully exploring each area before fully moving on. This is generally not a good strategy because the probability that a valid path exists in each region is equal (since no prior environment information is available), so it is more efficient to fully explore the closest regions one by one instead of jumping around from one region to another. Furthermore, in the case where the PRM graph is not connected, uniform sampling will search the entire configuration space and will likely generate many unnecessary samples before connection occurs.

Unfortunately, the random nature of PRM based planners makes it extremely difficult to completely resolve these issues. However, many of these issues can be mitigated by incorporating local sampling. In local sampling, the samples s_l are generated uniformly within some neighborhood δ of a configuration X' (i.e. $s_l \sim U[\{x \in \mathcal{C} : ||x - X'|| \leq \delta\}]$). The goal of local sampling is to allow the algorithm to perform a more detailed search of a targeted area by increasing the sample density there. Thus, whenever a collision is detected along the solution path, a set of local samples generated about the point of collision should be added to the graph before searching for a new solution path. If the graph is not connected, then a mix of uniform and local samples should be added to the graph until connectivity is achieved. Note that local sampling should never replace uniform sampling because there may not exist any valid paths near the collision site.

PRM Regeneration

The most expensive operation in PRM based algorithms is performing collision checks along edges. Thus, the fewer nodes and edges there are, the more efficiently the algorithm will run. In unknown environments where all unexplored areas are assumed to be obstacle-free, the initial solution path will be a straight line from start to end. There is very little benefit to constructing the initial PRM graph beyond this, because any additional nodes and edges contribute nothing to the connectivity of the graph at that time and will simply slow down computations. It is more advantageous to only increase the size of the PRM when necessary, and to do so with the help of targeted sampling methods, such as local sampling and other techniques which will be discussed in Chapter 4. The goal is to maintain a connected PRM graph, using as few nodes and edges as possible, at all times.

Unfortunately, even with such an approach it is inevitable that the PRM graph will continue to increase in size as new obstacles are detected and the solution path is updated. As a result, the longer the algorithm runs, the worse its computational performance will be. Fortunately, many of the nodes and edges in the PRM graph become irrelevant after the vehicle has passed through their region, and thus can be safely removed from the graph. For instance, when the vehicle finds a narrow tunnel, numerous uniform and local samples may have to be generated before a connected graph can be formed. Of these, only a small subset will actually be part of the solution path through the tunnel. After a path through the tunnel has been found, all of the samples that were generated during the search but are not part of the path are no longer useful because they do not contribute to the connectivity of the graph and they are no longer needed to facilitate the search for connectivity through that area. It is important, however, to retain all of the nodes and edges that have been traversed by the vehicle, because this allows the vehicle to backtrack through previously visited areas, if necessary, without having to redo all the searches. Thus, after the PRM graph has reached a certain size threshold, it can be regenerated by removing all nodes and edges except for those that the vehicle has traversed and the desired end configuration. It is important to ensure that this threshold is sufficiently large that enough samples can be retained to successfully find paths through difficult areas. The main purpose of this PRM regeneration technique is not necessarily to keep the PRM graph as small as possible, but rather to prevent it from growing increasingly larger throughout the duration of the algorithm.

3.2.3 Applying the Transform Algorithm

The PRM based planner given in Algorithm 2 will produce a piece-wise linear solution path through the unknown environment, and update this path as necessary when new obstacles are found. This solution path will have all the same properties as the PRM paths generated for known environments. Therefore, the heuristic re-sampling and transformation algorithms presented in Chapter 2 can also be applied to the solution paths found in the unknown environments to enforce any kinodynamic vehicle constraints. The one key difference, however, is that in unknown environments, the solution path might become invalid at any given time due to newly found obstacles. Thus, to ensure that the vehicle is capable of switching trajectories or stopping before colliding with detected obstacles, an additional velocity bound, V_{abs}, will be imposed on all motion primitives. Unlike the other velocity bounds found in the transformation algorithm which are only enforced at the nodes (i.e. start and end of each motion primitive), V_{abs} will be enforced along the entire path and thus becomes an upper bound on the vehicle velocity.

The choice of V_{abs} is a trade-off between safety and efficiency. If it is too high, then there is a risk of collision, but if it is too low, then the path traversal becomes slow and inefficient. A good choice of V_{abs} will be the maximum velocity at which the vehicle can still come to a stop (zero velocity) before reaching any potential obstacles. The braking distance used to calculate V_{abs} will thus be the distance along the path that the obstacle sensor can detect unobstructed. In most cases, this will simply be the maximum range of the sensor, however for corners this distance may be reduced due to obstruction by other non-colliding (with the path) obstacles. Assuming that the obstacle sensors are consistent (i.e. unobstructed obstacles are detected as soon as they are in range of the sensor), this is necessary to ensure the safety of the vehicle in the worst case scenario where an alternate path can't be found. Note that a safety factor should be built into V_{abs}, because the vehicle may require some time to adhere to V_{abs} in cases where it changes drastically due to newly found non-colliding obstacles obstructing the sensor's view of the path. For vehicles that operate in 3D space, such as UAVs and UUVs, the achievable accelerations (and hence stopping distances for a given velocity) will depend on the velocity direction due to the effects of gravity. In such cases, the value of V_{abs} will also depend on the direction of the motion primitives.

When a collision is detected, the vehicle should start to decelerate and prepare to stop before hitting the obstacle in preparation for the worst case where a new solution path can't be found or takes too long to compute. If the collision does not occur on the current motion primitive being traversed, then the new solution paths will retain the current motion primitive and transition the vehicle onto the new path using the method described in Section 2.4.1 for extending solution paths in multi-stage planning. On the other hand, if the collision is on the current motion primitive, then the vehicle will simply come to a stop before transitioning onto the new solution path.

3.3 Simulation Results

In this section, the proposed motion planner for unknown environments will be simulated in a variety of 3D environments. The first simulation will focus on the behavior of the PRM based planner for unknown environments with local sampling and PRM regeneration. The second simulation will incorporate the transformation algorithm to make the paths kinodynamically feasible for a skid-steer vehicle. All of the simulations will be performed on a Lenovo W500 Thinkpad with a 2.80 GHz Intel Core 2 Duo Processor (T9600) and 4.00 GB of RAM.

3.3.1 Occupancy Grid Mapping

To simulate the motion planning algorithms for unknown environments, a standard occupancy grid mapping algorithm [58, 62] will be used to generate the map of the environment.

The algorithm discretizes the map of the environment into distinct cells and calculates the probability that each cell is occupied by an obstacle by performing Bayesian updates using the sensor measurements. In order to avoid truncation issues when the probability is close to 0 or 1, the algorithm stores the map and performs the Bayesian updates in *log odds form.* For a given probability p, the log odds form $L(p)$ is given by

$$L(p) = \log\left(\frac{p}{1-p}\right) \tag{3.1}$$

Inversely, the log odds form can be converted back into the corresponding probability using

$$p = \frac{1}{1+e^{L(p)}} \tag{3.2}$$

The Bayesian update using the log odds form will be

$$L(p(m_i|z_{1:t})) = L(p(m_i|z_{1:t-1})) + L(p(m_i|z_t)) - L(p(m_i)) \tag{3.3}$$

where m_i is the ith grid in the map, $z_{1:t}$ is the set of all sensor measurements up until time t, and $p(m_i|z_{1:t})$ is the probability that grid m_i is occupied given $z_{1:t}$. The term $p(m_i)$, which corresponds to the initial probability that grid i is occupied, will be set to 0.5 since no obstacle information is available at the start. The $p(m_i|z_t)$ term (i.e. the probability that grid i is occupied given the current measurement at time t) is derived from the the inverse sensor model which will depends on the sensor being used.

The simulations and experiments (discussed in Chapter 5) will use LIDAR sensors to detect obstacles. The inverse sensor model for a LIDAR sensor can be represented as

$$p(m_i|z_t) = \begin{cases} p_{occ} & \text{if lidar ray hit an obstacle in grid} \\ p_{free} & \text{if lidar ray passed through grid} \\ 0.5 & \text{otherwise} \end{cases} \tag{3.4}$$

In the experiments, the Octomap [62] implementation of the occupancy grid mapping algorithm will be used for convenience. Thus, in the simulations, the values of p_{occ} and p_{free} are selected to be 0.7 and 0.4 respectively to match with those used in the Octomap implementation. Furthermore, the Octomap implementation sets clamping thresholds of 0.12 and 0.97 which are used as the lower and upper bounds of $p(m_i : z_{1:t})$ during the Bayesian updates. This prevents overconfidence in the map, and allows the algorithm to respond more rapidly to changes in the environment. However, since the motion planner for unknown environments assumes a static environment, this clamping threshold is not implemented in the simulations.

3.3.2 PRM Based Planner for Unknown Environments

The PRM based planner for unknown environments is simulated using the Panda3D visualization engine, and a custom collision checker using the standard bisection method [42]. The purpose of this simulation is to demonstrate that the planner is capable of finding collision-free paths through unknown environments, and that it can efficiently replan the path when necessary. The local sampling and PRM regeneration techniques have also been included for increased efficiency. Since the focus is on testing the PRM based planner for unknown environments, the transformation algorithm will not be included in the simulations. As a result, the vehicle will be made to traverse along the prescribed path without any kinodynamic considerations (i.e. the vehicle model will be ignored). Finally, a simulated rotating LIDAR and the occupancy grid mapping algorithm is used to generate the map of the environment.

The time lapsed view of a typical simulation run is shown in Figure 3.1. The red lines show the PRM graph, the yellow line is the current solution path, and the white line shows the traversed path. The vehicle's knowledge of the environment is limited to the occupancy grid map, which is shown in purple. The green lines emanating from the vehicle shows the range and pose of the simulated rotating LIDAR. At the start of the simulation, the initial solution path passes through many obstacles due to limited knowledge of the environment (Figure 3.1(a)). Then, as the vehicle starts mapping the environment, the solution path is replanned to avoid the uncovered obstacles. In order to maintain a connected PRM graph, it is often necessary to add more samples to the PRM graph (Figure 3.1(b)). In this simulation, 10 uniform samples and 10 local samples are added each time. However, after the PRM graph has grown past some defined threshold, it can be regenerated to maintain efficiency (Figure 3.1(c)). Note that in the simulations, the PRM graph is regenerated once the time required to collision check the entire PRM graph surpasses 1 second, instead of after the graph has surpassed a certain number of nodes or edges. The effects of PRM generation and node addition using local sampling are very apparent in Figures 3.1(e) - 3.1(g). Here, local sampling creates dense PRM graphs that very thoroughly search the area that the vehicle has to immediately navigate through. At the same time, PRM regeneration removes the portions of the PRM graph that are no longer needed. This allows the PRM graph to perform detailed searches of critical areas without getting bogged down. Finally, the complete solution/traversed path is shown in Figure 3.1(g).

As with any PRM based algorithm, the largest runtime bottleneck is performing collision checks on the edges in the graph. Unfortunately, this has to be done before each replan so that all of the new obstacle information is accounted for. Using the environment shown in Figure 3.1, the relationship between the time it takes to update the PRM during replanning and the size of the PRM graph is studied, and the average results of 10 simulations is summarized in Figure 3.3. Note that PRM regeneration is applied whenever the update time exceeds 1 second. From Figure 3.2(a), it is apparent that the time required to collision check the PRM graph shows a polynomial increase with respect to the number of nodes in the graph. In Figure 3.2(b), a similar relationship is seen between the time required to regenerate the PRM and the number of retained nodes (i.e. the previously traversed nodes). However, regenerating the PRM is still very efficient because the number of retained nodes will always be relatively small compared to the size of the PRM, and regenerating the PRM is approximately 100 times more efficient than updating collisions in the PRM (assuming same number of nodes). The effects of PRM regeneration for a single run is seen in Figure 3.2(c). As samples are added to the PRM graph during the graph updates to preserve connectivity, the time required to perform each update increases. However, by including PRM regeneration, it is possible to bound the runtime of these updates by essentially resetting the PRM size after a threshold has been exceeded. This results in the sawtooth planning times visible in Figure 3.2(c). While replanning the solution path, the algorithm also has to generate additional samples and perform a graph search. However, the runtimes of these steps are negligible compared to the runtime of the collision update step (less than 10%), and will generally not increase significantly with the size of the PRM graph.

The PRM based planner for unknown environments has several limitations as well. Given a sufficiently long path, there are two potential ways in which PRM regeneration can fail as a result of having too many retained nodes. First, the sheer number of retained nodes may cause the PRM regeneration to become prohibitively expensive computationally. Second, the number of retained nodes may start exceeding the PRM regeneration threshold and deadlock the algorithm (since no additional nodes can be generated). While neither issue has been seen in simulation or experiments (discussed in Chapter 5), they will be further studied in future work. Another limitation to the algorithm is that in complex environments, it is possible that PRM regeneration occurs before a valid solution path is found. This usually occurs due to a poor set of samples which fail to find a path before the regeneration threshold is reached. This significantly increases the runtime since the PRM regeneration also resets the search process. During simulations, this issue has been observed in more complex regions of the environment, however a solution path is usually found within 3 PRM regenerations. The observed longest time required to replan the solution path during simulation was 4.23 seconds. While the numerical runtime can vary greatly depending on factors such as the complexity (i.e. dimension, size, number of obstacles, etc.) of the configuration space, the efficiency of the collision checking algorithms, and the capabilities of the hardware it is being run on, the performance of this algorithm shows great potential for being used in online planning and replanning.

3.3.3 Replanning with Motion Primitives for a Skid-steer Vehicle

The purpose of this simulation is to demonstrate that the transformation algorithm is capable of finding motion primitives for the paths generated from the PRM based planner for unknown environments. This simulation will be performed using Robot Operating System (ROS). Visualization is done through RVIZ (a ROS visualization stack), and collision checking will be performed using the same method as Section 3.3.2. The motion primitives generated by the transformation algorithm will be based on a simple kinematic model of a skid-steer UGV. As well, instead of simulating the map generation, an actual 3D occupancy grid map generated using the Octomap library and a Hokuyo URG-04LX-UG01 LIDAR is used. Note that the map is not built from the perspective of a vehicle traversing the solution path.

Kinematic Skid-steer UGV Model and Implementation Details

A standard kinematic model for skid-steer UGVs [38] will be used to generate the motion primitives. The model will be derived under the assumption that the inertial XYZ frame is a right hand frame with positive Z pointing upwards, and the body fixed xyz frame is a right hand frame with positive x pointing towards the front of the vehicle and positive z pointing upwards. As well the model will assume that the roll and pitch of the vehicle is 0 (i.e. level ground). The two control inputs are the angular velocities of the wheels on the left side (ω_l) and the wheels on the right side (ω_r) of the vehicle. Given wheel radii of r_w and a wheel base of l_b, the forward velocity v_x (along the body fixed x axis) and the angular velocity $\dot{\psi}_z$ (about the body fixed z axis) can be found by

$$\begin{bmatrix} v_x \\ \dot{\psi}_z \end{bmatrix} = \begin{bmatrix} \frac{r_w(\omega_r+\omega_l)}{2} \\ \frac{r_w(\omega_r-\omega_l)}{2l_b} \end{bmatrix} \tag{3.5}$$

Then, the corresponding velocities ($\dot{X}$ and $\dot{Y}$) and yaw rate ($\dot{\Psi}$) in the inertial frame will be

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\Psi} \end{bmatrix} = \begin{bmatrix} \cos\Psi & x_{ICR}\sin\Psi \\ \sin\Psi & -x_{ICR}\cos\Psi \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_x \\ \dot{\psi}_z \end{bmatrix} \tag{3.6}$$

where Ψ is the heading of the vehicle, and x_{ICR} is the location of the instantaneous center of rotation projected onto the body fixed x axis. Note that when $x_{ICR} > 0$, the model assumes that the vehicle undergoes constant slippage at the wheel contacts (which is an operating condition for skid-steer UGVs).

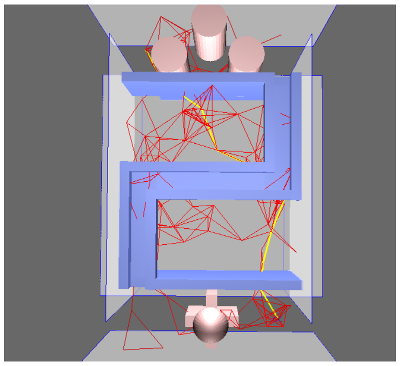
Since the skid-steer UGV is non-holonomic, several modifications must be made to the planner. Normally, each node in the PRM graph represents a distinct configuration, which in this case would correspond to $\{X, Y, \Psi\}$. However, because the skid-steer UGV is unable to move laterally, such a node would only be able to connect to other nodes that lie in the direction of Ψ. This severely increases the difficulty in achieving connected PRM graphs. Therefore, to accommodate the non-holonomic constraint, each node will have a distinct $\{X, Y\}$ state with a free floating Ψ. Then, whenever an edge is collision checked, the Ψ of the two end nodes will be set to align with the edge. Using this approach, a solution path consisting of any collection of edges from the PRM graph will be satisfy the non-holonomic constraint, because the edges were constructed under the assumption that the vehicle is traversing it with the required heading.

Since the vehicle is a UGV, the friction forces between the ground and wheels are quite significant and allow the vehicle to decelerate very rapidly. In many cases, the maximum vehicle velocity will be relatively low due to limiting factors such as motor capabilities, sensor update rates, and the efficiency of on-board estimation, control, mapping, or path planning algorithms. As a result, many of the velocity bounds along the straight segments become trivial (i.e. higher than the maximum velocity allowed by the other limiting factors), and the vehicle will be able to quickly stop whenever necessary.

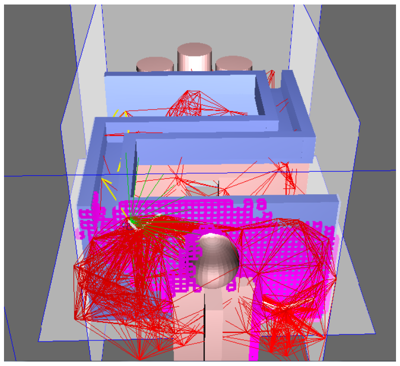
During corner motions, the necessary inward acceleration is provided by the friction forces (instead of the vehicle as in the case of holonomic UAVs). However, due to the non-holonomic constraint, the corner motion is also limited by the maximum achievable $\dot{\psi}$ of the vehicle. Given the maximum $\dot{\psi}$ and assuming the friction forces are sufficient to hold the vehicle in the corner motion, it is possible to calculate the maximum v_x that the vehicle can traverse each corner at. This method will be used to define the velocity bounds of the corner motion primitives when the transformation algorithm is applied. In this simulation, the maximum v_x is set to be $1.0\frac{\text{m}}{\text{s}}$ and the maximum $\dot{\psi}$ is set to be $0.3\frac{\text{rad}}{\text{s}}$.

Simulation Results

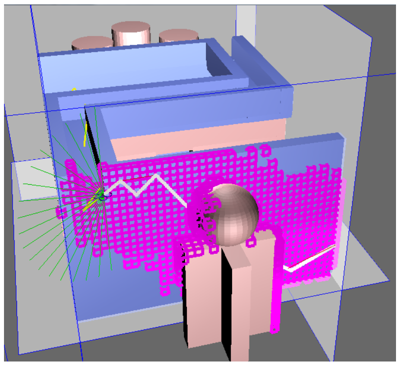
The results of the simulation can be seen in Figure 3.3, where the solution path is shown in yellow, the PRM graph is shown in purple, the occupancy grid map is shown in green, and line AC of each corner motion primitive is shown in red. Figures 3.3(a) - 3.3(c) show the solution path being replanned with motion primitives as new obstacles are uncovered. Figure 3.3(d) shows the final solution path through the environment on the final map. The runtimes for the replanning portion of the algorithm are similar to Section 3.3.2, and the average runtime of the transformation algorithm over 50 simulations is about 0.3 seconds. Of course, these runtimes will be affected by all the factors previously discussed in Section 3.3.2. However, since the transformation algorithm is relatively efficient compared to the PRM based planner for unknown environments, there will not be a significant decrease in performance by incorporating the transformation algorithm. Therefore, the combined planner will still be viable for online planning and replanning.



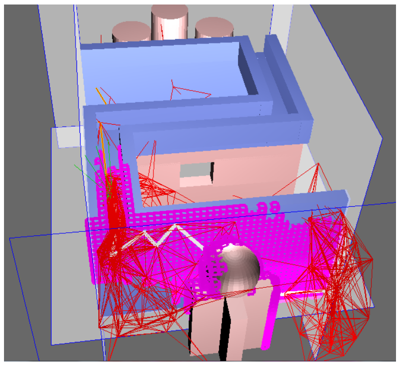
(a) t = 0 s



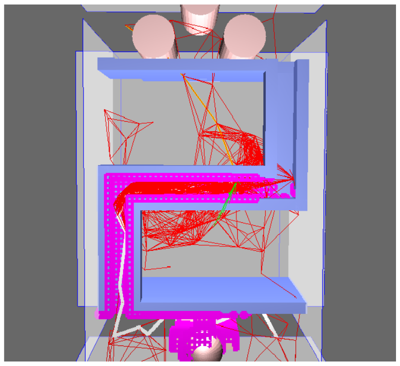
(b) t= 15 s



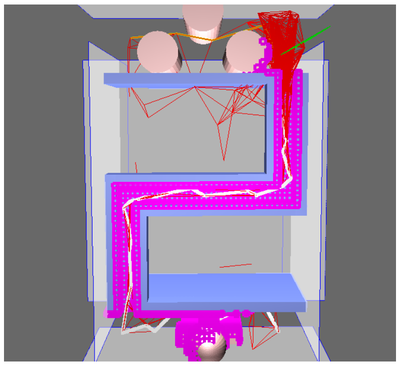
(c) t= 17 s



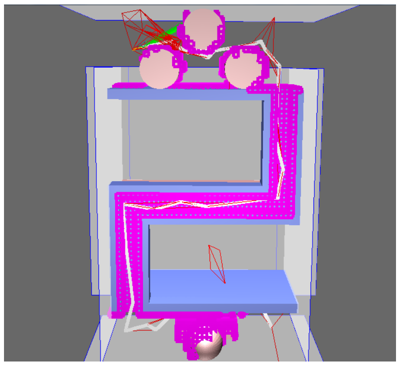
(d) t = 20 s



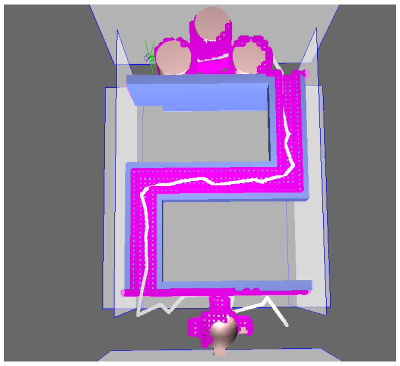
(e) t = 25 s



(f) t = 35 s

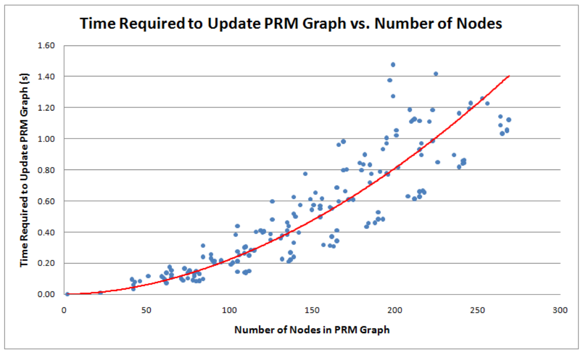


(g) t = 40 s

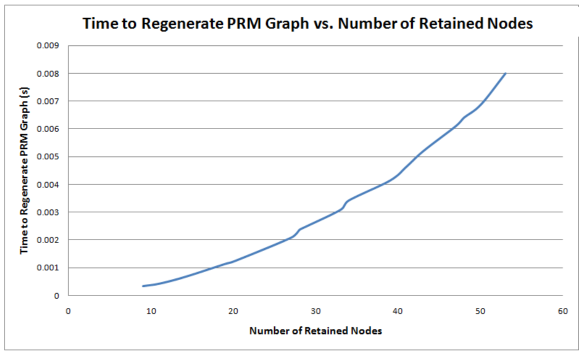


(h) t = 42 s

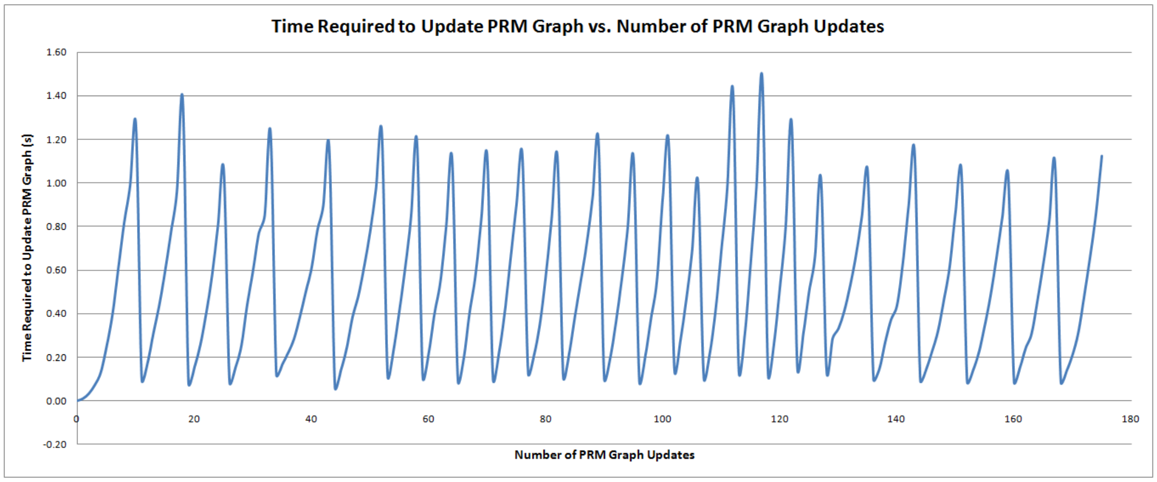
Figure 3.1: Simulation of the PRM based planner for unknown environments (time lapsed screen captures).



(a) Relationship between time needed to update PRM graph and number of nodes.

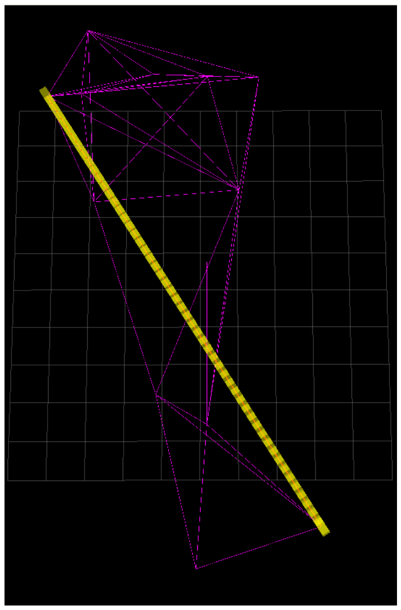


(b) Relationship between time needed to regenerate PRM graph and number of retained nodes.

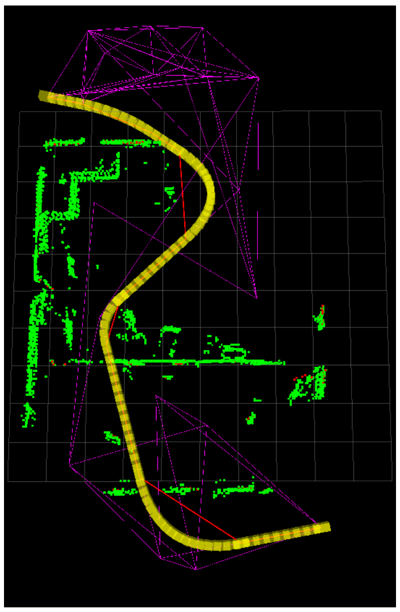


(c) Relationship between time needed to update PRM graph and number of performed updates.

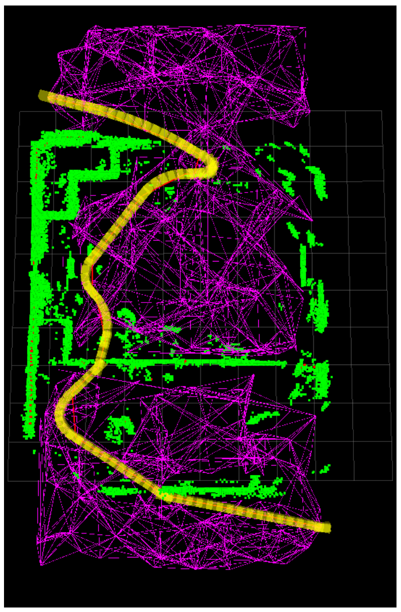
Figure 3.2: Effect of PRM graph size and PRM regeneration on algorithm runtimes.



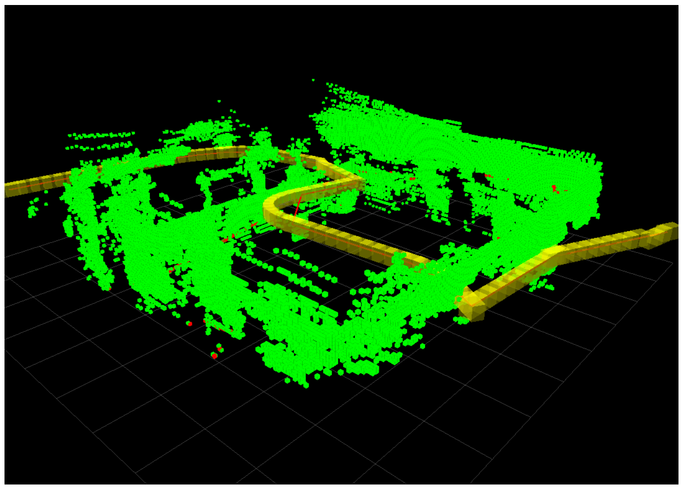
(a) t = 0 s



(b) t = 10 s



(c) t = 20 s



(d) 3D view of the final path.

Figure 3.3: Applying the transformation algorithm to the PRM based planner for unknown environments (time lapsed screen captures).

Chapter 4

The Orthogonal Bridge Test

The efficiency of PRM based planners comes from their ability to approximate the configuration space using a connected graph of random samples. In order for the PRM graph to be effective, it is important that the samples are representative of the configuration space and form a connected graph with collision-free edges. However, using traditional uniform sampling, it is difficult to generate such connected graphs through narrow regions due to poor expansiveness [29]. To alleviate these issues, targeted sampling strategies that bias the sample density in favor of narrow or complex regions are used. Since collision checking is computationally expensive, performance gains can be achieved by constructing connected PRM graphs with as few nodes and edges as possible. This chapter will introduce a novel sampling strategy (the orthogonal bridge test) that attempts to minimize the size of the PRM graph by focusing the samples in critical regions.

4.1 Background and Theory

There exists many sampling strategies for PRM planners that attempt to densely sample narrow or potentially complex areas in the configuration space. Many of these techniques use collision information to bias the sample distribution towards areas containing obstacles. While obstacles do reduce the expansiveness of a region, thus making it harder to form connected graphs, simply being in close proximity to an obstacle is not a sufficient condition for the existence of a narrow or complex area. Thus, it is common for many samples to be generated in relatively unconstrained areas of the configuration space as byproducts. To better position the proposed sampling strategies within the existing literature, several of the most widely used sampling strategies for PRM based planners are outlined below.

Gaussian sampling [8] increases the sampling density near the boundaries of free space (i.e. edges or surfaces of obstacles). This is achieved by picking pairs of random samples, separated by some distance chosen from a Gaussian distribution, until one sample is within an obstacle while the other is in free space. The sample in free space is then kept since it is in close proximity to an obstacle. However, this yields many samples near the boundaries of expansive free space (i.e. along the surface of the walls of a big room).

Obstacle-based sampling, described as part of OBPRM [2], is similar to Gaussian sampling in that it aims to sample densely along the surfaces of obstacles. The idea is to find a sample within an obstacle, and then move it in a random direction until it enters free space. However, like Gaussian sampling this also produces many samples near the boundaries of expansive free space.

The free space dilation approach [28] first dilates the free space by allowing samples to penetrate a certain distance into the obstacles. This widens the narrow areas of the configuration space and makes it easier to connect the roadmap. Then, any sample inside an obstacle is pushed back into free space through local re-sampling. Like the methods above, this technique does not distinguish between narrow areas and boundaries of expansive free space. More complex geometric operations are also required for higher dimensions.

Bridge sampling [27] attempts to sample within narrow passages instead of near obstacles. It finds two random samples, separated by a distance chosen from a probability density, that are inside obstacles but whose midpoint is in free space. The midpoint is then kept since it lies in between two obstacles, which suggests the existence of a narrow passage. However, as explained in Section 4.2, this is often not the case. Nevertheless, this method shows great potential and is the basis and inspiration for the proposed orthogonal bridge test.

4.2 Limitations of Bridge Sampling

As motivation, the limitations of bridge sampling and the corresponding effects in 2D and 3D configuration spaces will be examined. The purpose of bridge sampling is to increase the sample density within narrow passages in the configuration space, $\mathcal{C}$. This is accomplished by generating samples using the *bridge test*. First, two uniform samples, x_1 and x_2, are generated within close proximity of each other (the distance between x_1 and x_2 is usually selected according to some probability distribution). Then, the midpoint of the line segment $\overline{x_1 x_2}$ will be the potential sample s. If x_1 and x_2 are both in collision (i.e. $x_1 \notin \mathcal{C}_{free}$ and $x_2 \notin \mathcal{C}_{free}$) but s is in $\mathcal{C}_{free}$, then s is said to have passed the bridge test and is added to the PRM graph. The details of the bridge sampling algorithm is presented below in Algorithm 3 [27], where the following notations are used: $x \sim P$ denotes a point x being drawn from a probability distribution P, $U[S]$ and $\Lambda[S]$ represent uniform and user selected probability distributions that span a space S respectively, $\mathcal{C} \in \mathbb{R}^n$ is the configuration space, $\mathcal{C}_{free} \subseteq \mathcal{C}$ is the free space (i.e. obstacle-free subspace of $\mathcal{C}$), and $\overline{N}$ is the desired number of samples.

Algorithm 3 Bridge Sampling

```
1: $N = 0$
2: while $N < \overline{N}$ do
3:     Select $x_1 \sim U[\mathcal{C}]$
4:     if $x_1 \notin \mathcal{C}_{free}$ then
5:         Select $x_2 \sim U[\{x \in \mathcal{C} : ||x - x_1|| = d, d \sim \Lambda[\mathbb{R}_+]\}]$
6:         if $x_2 \notin \mathcal{C}_{free}$ then
7:             $s = \frac{x_1+x_2}{2}$
8:             if $s \in \mathcal{C}_{free}$ then
9:                 Accept $s$ as a valid sample
10:                N = N+1
11:            end if
12:        end if
13:    end if
14: end while
```

Let $E = \{x \in \mathcal{C} : \rho(x, \Lambda[\mathbb{R}_+]) > k\}$ where $\rho(x, \Lambda[\mathbb{R}_+])$ is the probability of x passing the bridge test and $k > 0$ is some defined threshold. Thus, E is the set of all $x \in \mathcal{C}$ where bridge samples can be generated with a probability greater than k. Then, a site, B_i, is defined as any disjoint, compact subset of E such that $\bigcup_{i \in I} B_i = E$. With a sufficiently small k, all the bridge samples will lie in one of the sites within the configuration space. Unfortunately, while sites exist in narrow passages, they can also exist along convex boundaries of free space. This is illustrated in Figure 4.1, where the shaded areas represent obstacles, the black dot is the bridge sample p, and the black crosses are x_1 and x_2 (the two samples used to generate p). Since most robots have finite regions of operation, the free space will generally be enclosed by obstacles causing convex boundaries to appear in at least one section.

The probability, φ_i, that a bridge sample is generated at site B_i can be approximated using

$$\varphi_i \approx \frac{\int_{B_i} \bar{\rho}(B_i)}{\sum_j \int_{B_j} \bar{\rho}(B_j)} \tag{4.1}$$

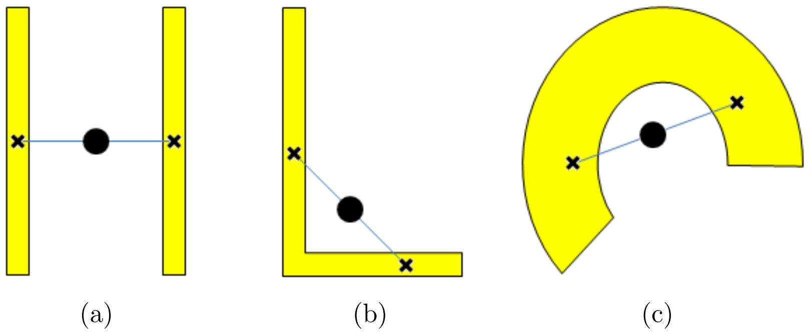


Figure 4.1: Bridge sample is generated within a narrow passage in (a), and along the convex boundary of free space in (b) and (c).

where $\bar{\rho}(B_i)$ is the mean probability that a sample in B_i will pass the bridge test. Let the set of target sites, T, be a user selected subset of sites. Generally, the sites formed by narrow passages in the configuration space are chosen for T. Since there is an abrupt change in $\rho(x, \Lambda[\mathbb{R}_+])$ at the end of the narrow passages in the examples below, it is possible to select k such that the boundaries of the sites in T (indicated by red boxes in Figures 4.2 and 4.3) match the boundaries of the narrow passages. Then, all samples $x \in T$ are defined as desired and samples $x \notin T$ as undesired. Since φ_i is proportional to the relative volume of B_i (Equation (4.1)), if the volume of T is relatively small compared to the set of non-target sites $T' = \{B : B \notin T\}$, then it will generally be more difficult to generate desired samples (Figure 4.2).

As the dimension of the configuration space increases, the volume of non-target sites will increase as they are extruded into the additional dimensions. For example, the convex boundary of free space is a line in 2D, but becomes a plane in 3D. While the target sites can similarly increase in volume, they are often more constrained along the additional dimensions. In addition, the number of sites in T' will increase as new convex regions are created in the boundary each time a dimension is added. For instance, the boundary of a 2D square is only convex at its four corners, but the boundary of a 3D cube is convex at all eight corners and twelve edges. This causes more undesired samples to be generated in higher dimensional configuration spaces as shown in Figure 4.3.

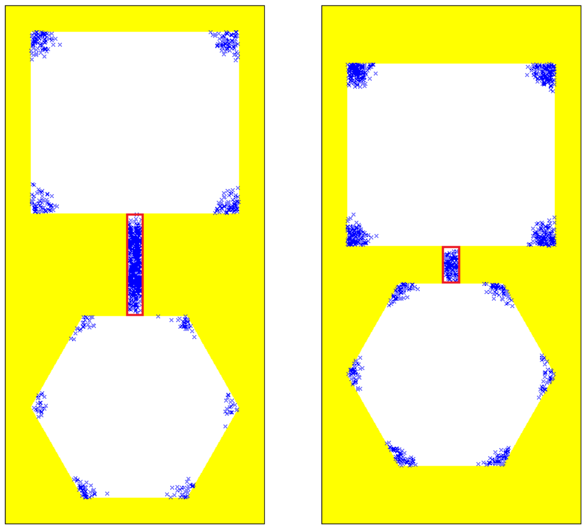


Figure 4.2: 1000 bridge samples generated in a 2D configuration space. Percentage of desired samples decreases from 53.5% in (a) to 12.3% in (b) as the area of the target site (red box) decreases by 62.5%.

4.3 Algorithm Overview

The o-bridge test attempts to deduce whether a bridge sample is in a narrow passage by taking additional samples in the neighborhood and analyzing their collision information. Whereas the bridge sample uses the line segment $\overline{x_1x_2}$ to detect narrowness along a single dimension, the o-bridge test will construct additional line segments orthogonal to $\overline{x_1x_2}$ in order to obtain similar information along the other dimensions. The detailed algorithms for 2D and 3D configuration spaces are described below.

4.3.1 Orthogonal Bridge Test in 2D Configuration Spaces

In a 2D configuration space, the o-bridge test will generate two additional points, y_1 and y_2, such that $\overline{x_1x_2} \perp \overline{y_1y_2}$ and the midpoint of $\overline{y_1y_2}$ coincides with the bridge sample p. Depending on the orientation of $\overline{x_1x_2}$, the points y_1 and y_2 will tend to both be in obstacles or in free space when the sample is in a narrow passage (Figure 4.4(a) and 4.4(b)). On the other hand, if the bridge sample is along a convex boundary, then only one endpoint of

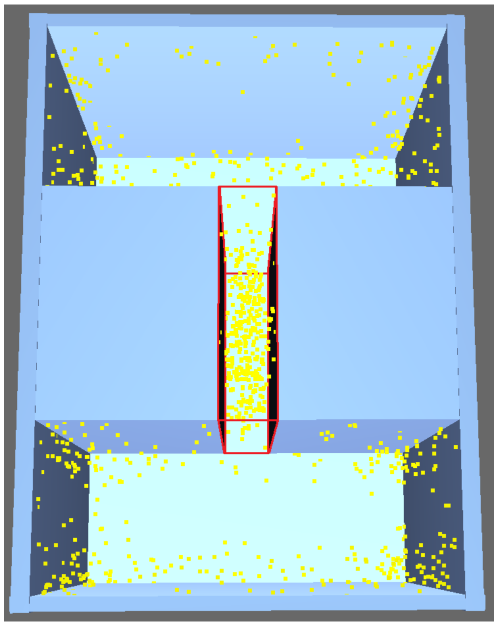


Figure 4.3: 1000 bridge samples generated in a 3D configuration space, with 31.7% of samples at the target site (red box).

$\overline{y_1 y_2}$ will be in an obstacle while the other resides in free space (Figure 4.4(c) and 4.4(d)). Thus, the o-bridge test rejects the sample if y_1 and y_2 are not both in obstacles or in free space. The details of the 2D o-bridge test are presented in Algorithm 4, where $\lambda \in \mathbb{R}$ is a scaling factor discussed in Section 4.3.4.

4.3.2 Orthogonal Bridge Test in 3D Configuration Spaces

In a 3D configuration space, the o-bridge test generates two sets of points, $\{y_1, y_2\}$ and $\{z_1, z_2\}$, such that $\overline{x_1 x_2} \perp \overline{y_1 y_2} \perp \overline{z_1 z_2}$. In the 2D case, the orientation of $\overline{y_1 y_2}$ is uniquely determined by the given orientation of $\overline{x_1 x_2}$. However, in 3D there are an infinite number of orientations that satisfy the orthogonality criterion (i.e. $\overline{y_1 y_2} \perp \overline{z_1 z_2}$ are free to rotate in the plane orthogonal to $\overline{x_1 x_2}$). To ensure that the o-bridge test is robust and unbiased for all configuration spaces, it is important to generate the orientation of $\overline{y_1 y_2} \perp \overline{z_1 z_2}$ randomly within the aforementioned plane. As in 2D, the sample is rejected if either y_1 and y_2 or z_1 and z_2 are not both in obstacles or in free space. The details of the 3D o-bridge test are presented in Algorithm 5.

Algorithm 4 Orthogonal Bridge Test in 2D

```
1: for all Bridge samples $s \in \mathbb{R}^2$ and corresponding $x_1 \in \mathbb{R}^2$ do
2:     $\vec{v} = x_1 - s = \begin{Bmatrix} v_1 \\ v_2 \end{Bmatrix}$
3:     $\vec{r} = \lambda \begin{Bmatrix} -v_2 \\ v_1 \end{Bmatrix}$
4:     $y_1 = s + \vec{r}$
5:     $y_2 = s - \vec{r}$
6:     if $(y_1 \in \mathcal{C}_{free}$ and $y_2 \in \mathcal{C}_{free})$ or $(y_1 \notin \mathcal{C}_{free}$ and $y_2 \notin \mathcal{C}_{free})$ then
7:         Accept $s$ as a valid sample
8:     end if
9: end for
```

Algorithm 5 Orthogonal Bridge Test in 3D

```
1: for all Bridge samples $s \in \mathbb{R}^3$ and corresponding $x_1 \in \mathbb{R}^3$ do
2:     $\vec{v} = x_1 - s$
3:     Select $\hat{u} \sim U[\{u \in \mathbb{R}^3 : ||u||_2 = 1\}]$
4:     $\vec{r}_1 = \vec{v} \times \lambda\hat{u}$
5:     $y_1 = s + \vec{r}_1$
6:     $y_2 = s - \vec{r}_1$
7:     if $(y_1 \in \mathcal{C}_{free}$ and $y_2 \in \mathcal{C}_{free})$ or $(y_1 \notin \mathcal{C}_{free}$ and $y_2 \notin \mathcal{C}_{free})$ then
8:         $\hat{r}_1 = \frac{\vec{r}_1}{||\vec{r}_1||_2}$
9:         $\vec{r}_2 = \vec{v} \times \lambda\hat{r}_1$
10:        $z_1 = s + \vec{r}_2$
11:        $z_2 = s - \vec{r}_2$
12:        if $(z_1 \in \mathcal{C}_{free}$ and $z_2 \in \mathcal{C}_{free})$ or $(z_1 \notin \mathcal{C}_{free}$ and $z_2 \notin \mathcal{C}_{free})$ then
13:            Accept $s$ as a valid sample
14:        end if
15:    end if
16: end for
```

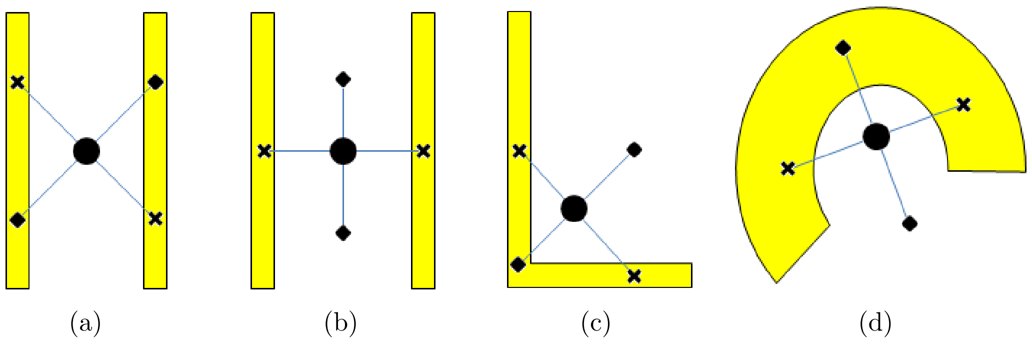


Figure 4.4: O-bridge test (denoted by diamonds) performed on bridge samples in a 2D configuration space. (a) and (b): A sample inside a narrow passage. (c) and (d): Samples along a convex boundary of free space.

4.3.3 Orthogonal Bridge Test in nD Configuration Spaces

The principles of the o-bridge test can be extended to higher dimensions. In n dimensions, the o-bridge test requires $n-1$ mutually orthogonal line segments that are orthogonal to $\overline{x_1x_2}$ as well. Such a set of line segments can be found using the Gram-Schmidt process [20] by choosing the initial vector to be parallel to $\overline{x_1x_2}$. The remaining $n-1$ linearly independent input vectors required by the Gram-Schmidt process will be randomly generated to avoid biases. The details of the nD o-bridge test are presented in Algorithm 6.

It should be noted that the conditions of the o-bridge test can be adjusted to meet specific requirements. For instance, to favor areas that are constrained in multiple dimensions, valid samples may be required to have several line segments with endpoints inside obstacles. Alternatively, the criterion can be relaxed by keeping samples that have failed the o-bridge test if they are sufficiently constrained along the other dimensions.

4.3.4 Choosing a Scaling Factor λ

The scaling factor, λ, is defined by the ratio $\frac{||\overline{y_1y_2}||_2}{||\overline{x_1x_2}||_2}$, and determines the lengths of the generated line segments. During bridge sampling a dimension of free space is defined to be narrow when it can be spanned by $\overline{x_1x_2}$. In the subsequent o-bridge test, the same measure of narrowness (i.e. $||\overline{x_1x_2}||_2$) should be kept for consistency. Thus, to determine if the remaining dimensions are similarly narrow, the o-bridge test will generally require a $\lambda \geq 1$.

Algorithm 6 Orthogonal Bridge Test in nD

```
1: for all Bridge samples $s \in \mathbb{R}^n$ and corresponding $x_1 \in \mathbb{R}^n$ do
2:     keep = TRUE
3:     $\vec{v_1} = x_1 - s$
4:     repeat
5:         $u = \begin{bmatrix} \vec{v}_1 & \vec{u}_2 & \cdots & \vec{u}_n \end{bmatrix}$, where $\vec{u}_{2\dots n} \sim U[\mathbb{R}^n]$
6:     until det($u$)$\neq 0$
7:     for $i = 2$ to $n$ do
8:         $\vec{v}_i = \vec{u}_i - \sum_{j=1}^{i-1} \frac{\vec{v}_j \cdot \vec{u}_i}{\vec{v}_j \cdot \vec{v}_j} \vec{v}_i$
9:         $\vec{v}_i = \frac{\lambda ||\vec{v}_1||_2}{||\vec{v}_i||_2} \vec{v}_i$
10:        $y_1 = s + \vec{v}_i$
11:        $y_2 = s - \vec{v}_i$
12:        if $(y_1 \in \mathcal{C}_{free}$ and $y_2 \notin \mathcal{C}_{free})$ or $(y_1 \notin \mathcal{C}_{free}$ and $y_2 \in \mathcal{C}_{free})$ then
13:            keep = FALSE
14:            break
15:        end if
16:    end for
17:    if keep = TRUE then
18:        Accept $s$ as a valid sample
19:    end if
20: end for
```

The precise choice of λ will depend heavily on the configuration space. For the o-bridge test to be effective, the generated line segments need to be long enough to reach the apex of the convex boundary. Thus, as shown in Figure 4.5(a) and 4.5(b), the smaller the angle formed by the convex boundary, the larger λ has to be. However, as λ increases, so does the probability that the line segments will extend through thin obstacles and back into free space (Figure 4.5(c)). This may cause the o-bridge test to fail because the collision state at the endpoints no longer represent that of the line segment. This issue can be mitigated by performing additional collision checks at the endpoints of successive bisections of long line segments. Unfortunately, the additional collision checks will rapidly degrade performance and produce unreasonable runtimes. An additional consideration is that as the angle of the convex border decreases, the area may start to resemble a narrow passage with low visibility (Figure 4.5(d)). It is not necessary to select a λ large enough to reject samples from these areas, since such samples can actually improve the connectivity of the PRM.

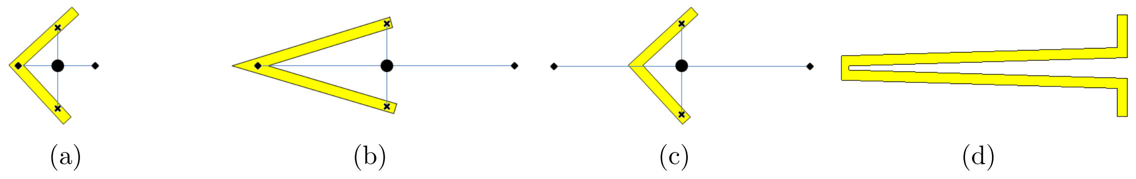


Figure 4.5: Relationship between λ and the angle of the convex border. (a) and (b): As the angle of the convex boundary decreases, λ has to increase to reach the apex. (c): If λ is too large, the generated line segments extend through the obstacle. (d): For small angles, the convex boundary closely resembles a narrow passage.

Even with an ideal choice of λ, it is not possible to perfectly distinguish between desired and undesired samples. Samples in target sites can be rejected by the o-bridge test due to disagreeable orientations of $\overline{x_1 x_2}$ (Figure 4.6(a)). Conversely, samples in non-target sites can pass the o-bridge test (Figure 4.6(b)), although this should be relatively rare for most configuration spaces. Since the o-bridge test tends to accept samples in targets sites more often than in non-target sites, it will still provide a favorable bias in the sample.

4.4 Simulation Results

The o-bridge test was simulated on a Lenovo W500 Thinkpad with a 2.80 GHz Intel Core 2 Duo processor and 4.00 GB of RAM. The 2D simulations were created in MATLAB$^{\text{TM}}$,

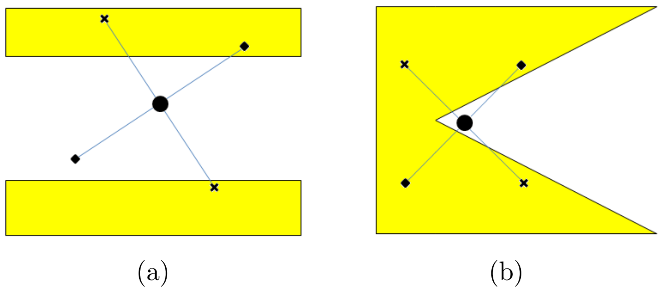


Figure 4.6: (a): Samples in a narrow passage can fail the o-bridge test. (b): Samples along a convex border can pass the o-bridge test.

while the 3D simulations used the Panda3D engine for visualization and the Open Dynamics Engine for collision checking. The results of the simulations are presented below.

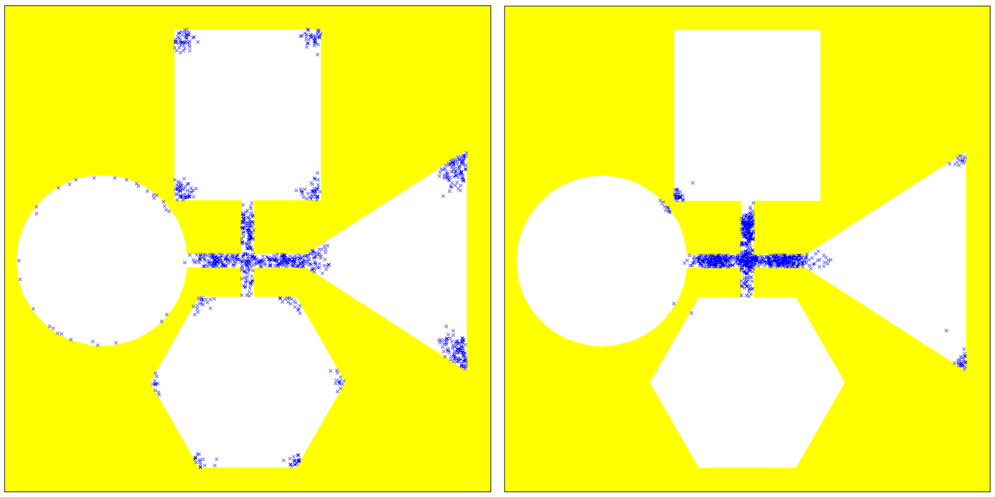
4.4.1 2D Configuration Space

The configuration space of the 2D simulation contains four large areas of free space connected by narrow passages. To study the robustness of the o-bridge test, the angles formed by the free space boundaries are varied from small (triangle) to large (circle), and thin obstacles are also included in between the large areas of free space (Figure 4.7). One thousand sample points were generated using bridge sampling with and without the o-bridge test. The same random seed was used for both simulations, and the o-bridge test used $\lambda = 1.5$ with collision checking performed on one additional bisection. The results are shown in Figure 4.7 and summarized in Table 4.1.

Table 4.1: Bridge sampling with and without the o-bridge test in 2D.

	Desired Samples (%)	Runtime (s)	Runtime/Desired Sample (s)
Without o-bridge	42.7	100.6	0.24
With o-bridge	91.6	284.9	0.31

In the the 2D simulation, undesired samples remain in the areas next to thin obstacles and in the corners of the triangle area. It is possible to eliminate the samples near the thin obstacles, by collision checking the bisections of the generated line segments, at the cost of computational efficiency. The majority of samples at the corners of the triangle



(a) (b)

Figure 4.7: 1000 bridge samples generated (a) without the o-bridge test and (b) with the o-bridge test in $\mathcal{C} \in \mathbb{R}^2$.

area passed the o-bridge test because both y_1 and y_2 are inside obstacles (Figure 4.6(b)). Unfortunately, it is not possible to distinguish between these samples and the ones in very confined areas (e.g. an sharp bend in a narrow tunnel). To avoid potential connectivity issues, these samples should not be removed.

4.4.2 3D Configuration Space

The bridge sample generated in the 3D configuration space shown in Figure 4.3 was re-simulated with the o-bridge test using the same random seed. One thousand samples were generated with $\lambda = 1.5$. The results are shown in Figure 4.8 and summarized in Table 4.2.

Table 4.2: Bridge sampling with and without the o-bridge test in 3D.

	Desired Samples (%)	Runtime (s)	Runtime/Desired Sample (s)
Without o-bridge	31.7	0.364	0.0011
With o-bridge	100	4.24	0.0042

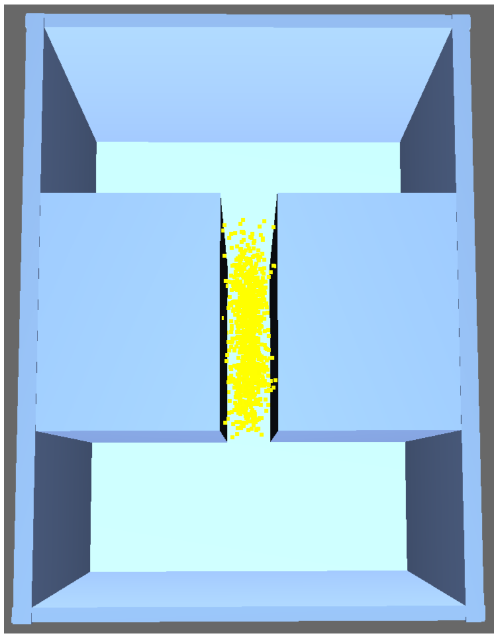


Figure 4.8: 1000 bridge samples generated with the o-bridge test in $\mathcal{C} \in \mathbb{R}^3$.

In the 2D and 3D simulations, the o-bridge test significantly reduced the number of undesired samples. However, the o-bridge test increased the adjusted runtime (runtime per desired sample) by about 30% in 2D and 380% in 3D.

4.4.3 6D Configuration Space

To examine the o-bridge test in higher dimensional configuration spaces, 20 samples were generated in a 6D configuration space without and with the o-bridge test (Figure 4.9(a) and 4.9(b) respectively) using the same random seed. For the o-bridge test, a scaling factor of $\lambda = 1$ was used. The configuration space spans the possible states of a 3D rectangular robot with 6 degrees of freedom (i.e. 3 translational and 3 rotational) within a 3D environment. The colored lines attached to each sample show the orientation of the body frame attached to the robot. In this simulation, the o-bridge test allowed more samples to be generated within narrow corridors compared to traditional bridge sampling.

Since the overall objective is to reduce the number of samples and collision checks required to construct a connected PRM, it is necessary to simulate PRM construction

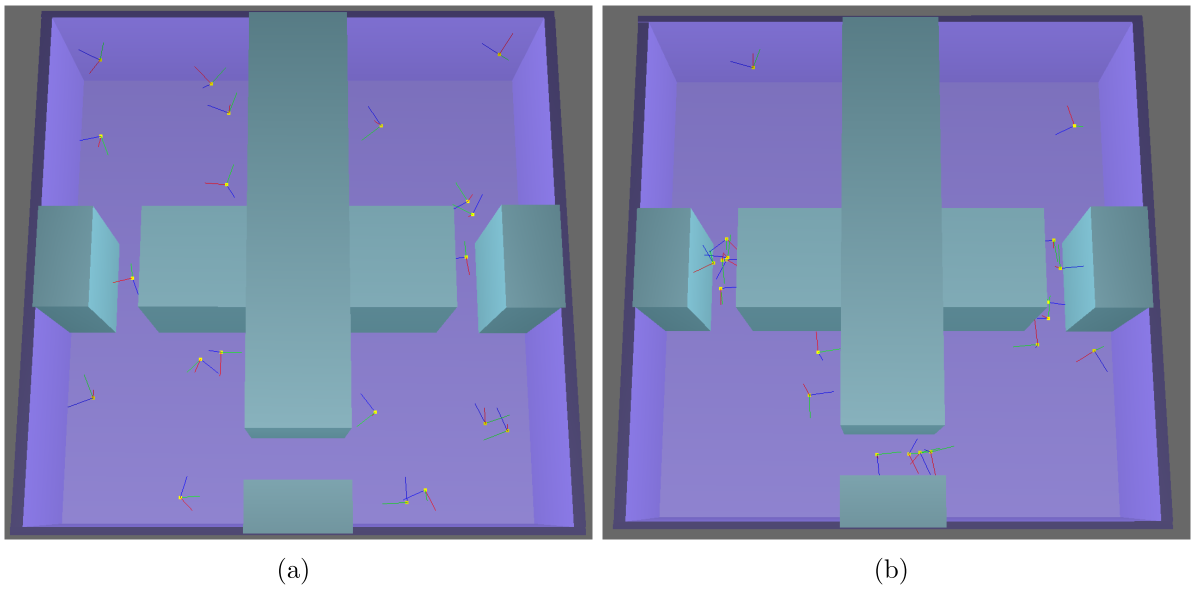


Figure 4.9: 20 samples generated (a) without the o-bridge test and (b) with the o-bridge test in $\mathcal{C} \in \mathbb{R}^6$.

with and without the o-bridge test. This was done using the 3D rectangular robot and environment from the previous simulation (Figure 4.10). As suggested in [27], a hybrid sampling strategy mixing uniform and bridge samples is used. In each simulation, 50 uniform samples are generated and then bridge samples are added 5 at a time until the roadmap is connected. Each sample in the PRM is restricted to 5 neighbors maximum in the interest of efficiency. The simulation was performed with and without the o-bridge test (20 times each) using the same random seeds. The results are presented in Table 4.3.

Table 4.3: PRM construction with and without the o-bridge test in 6D (average values of 20 simulations).

	Bridge Samples	Collision Checks	Sample Runtime (s)	Total Runtime (s)
Without o-bridge	92	94569	14.44	546.59
With o-bridge	39	18131	263.67	369.21

Based on these results, the o-bridge test is effective in reducing the number of samples and collision checks required to form a connected PRM. Although it caused a large increase

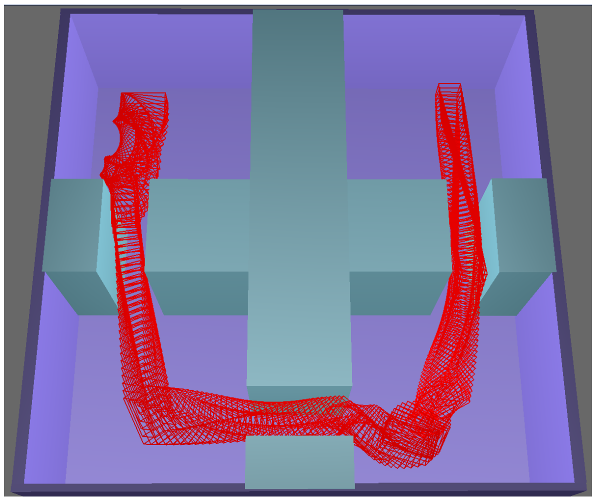


Figure 4.10: PRM path of a 3D rectangular robot with 6 degrees of freedom moving through a 3D environment.

in sampling time, the o-bridge test lowered the overall runtime of the PRM algorithm by about 30%. This performance gain is achieved primarily from having to perform fewer computationally expensive collision checks while connecting the PRM.

Chapter 5

Experimental Results

Two test sets were performed to experimentally verify the presented algorithms. The purpose of the first test set is to determine whether the motion primitives generated by the transformation algorithm are in fact kinodynamically feasible and can be tracked accurately. The second test set is a full implementation of the PRM based planner for unknown environments with the transformation algorithm. This experiment determines whether the proposed motion planner satisfies the goal of online navigation through an unknown environment.

5.1 Tracking Motion Primitives with a Quadrotor

5.1.1 Experimental Platform and Setup

The AR.Drone Parrot quadrotor (shown in Figure 5.1) was used to test how well an autonomous vehicle can track the motion primitives generated by the transformation algorithm. A UAV was used for this test instead of a UGV, because UAVs tend to be less stable (more sensitive to disturbances) and harder to control (more inputs and degrees of freedom). Thus, if an UAV can successfully track the motion primitives, the results can be extended to UGVs as well. Of the possible UAV platforms, the AR.Drone was chosen due to availability, convenience (pre-programmed inner loop control and self stabilizing capabilities), and safety features (light weight and rotor shroud).

The AR.Drone has an onboard 468 MHz ARM9 processor with 128 MB of DDR RAM, and runs on a custom Linux operating system. For state estimation, the AR.Drone uses

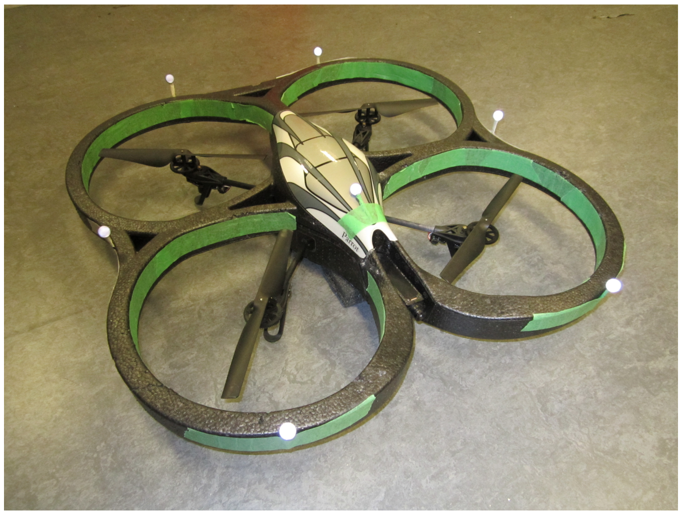


Figure 5.1: AR.Drone Parrot quadrotor.

an onboard inertial measurement unit (IMU) consisting of a MEMS 3-axis accelerometer, a 2-axis gyrometer for roll and pitch, and a 1-axis precision gyrometer for yaw. It also has two onboard monocular cameras facing front and downwards. The front fracing camera has a 93 degree wide angle diagonal lens, a video frequency of 15 frames per second, and a resolution of 640x480 pixels. The downward facing camera has a 64 degree diagonal lens, a video frequency of 60 frames per second, and a resolution of 176x144 pixels. There is also an ultrasound altimeter mounted at the bottom of the vehicle for height estimation of up to about 6 meters. Finally, the entire vehicle, which weighs approximately 420 g, is powered by 4 brushless motors (35,000 rpm at 15W) and a 11.1V 3-cell LiPo battery (1000 mAh).

For the localization, the OptiTrack Tracking Tools$^{\text{TM}}$ motion capture system was used. The system consists of 6 OptiTrack V100:R2 cameras, each of which have a resolution of 640x480 pixels and a maximum frame rate of 100 FPS. Each camera has a ring of 26 LEDs that emit 850 nm infrared (IR) light which is reflected off of special markers mounted on the vehicle body. The cameras are set to track the IR reflections and can calculate the position of the vehicle, through triangulation, to sub-centimeter accuracies. The system is capable of tracking a single vehicle at 50 Hz.

It was not possible to modify the proprietary software running onboard the AR.Drone, thus an offboard computer running Robot Operating System (ROS) was used to run the algorithms being tested and to receive the localization information from the OptiTrack system. Then, the computed control inputs are sent to the AR.Drone using the AR.Drone

ROS stack written by the Mobile Robotics Lab at Southern Illinois University Edwardsville (SIUE).

5.1.2 Vehicle Model and Control

As mentioned before, the AR.Drone has onboard inner loop controllers that are capable of stabilizing the vehicle at hover and commanding the vehicle to a specified pose. Thus, it was only necessary to develop the outer loop position controllers. From previous experiments, it has been verified that the inner loop controls for thrust and attitude are many times faster than outer loop controllers need to be, so it is possible to approximately decouple the inner and outer control loops.

The quadrotor model presented in Section 2.5.2 was used to derive the outer loop position controller. The mass and maximum thrust parameters have been changed to 0.42 kg and 5 N respectively in order to more accurately reflect the capabilities of the AR.Drone. The control strategy assumes that the outer loop position control is approximately decoupled in the body fixed x, y, and z directions. This assumption is once again made on the basis that the inner loop controls are many times faster than the outer loop position controller, and is standard practice for quadrotor control [24]. This allows a single-input single-output PID controller to regulate the vehicle position along each body fixed axis. However, instead of linearizing the model, the non-linearities were canceled out by inverting the dynamics, thus allowing for a larger region of operation. This was done by tuning the PID controllers for a plant of $\frac{1}{s^2}$ so that it can find the target accelerations (along each axis of the inertial frame) necessary for the vehicle to reach the desired position. These inertial accelerations can then transformed into an equivalent set of control inputs using Equations (2.26), (2.27), and (2.28).

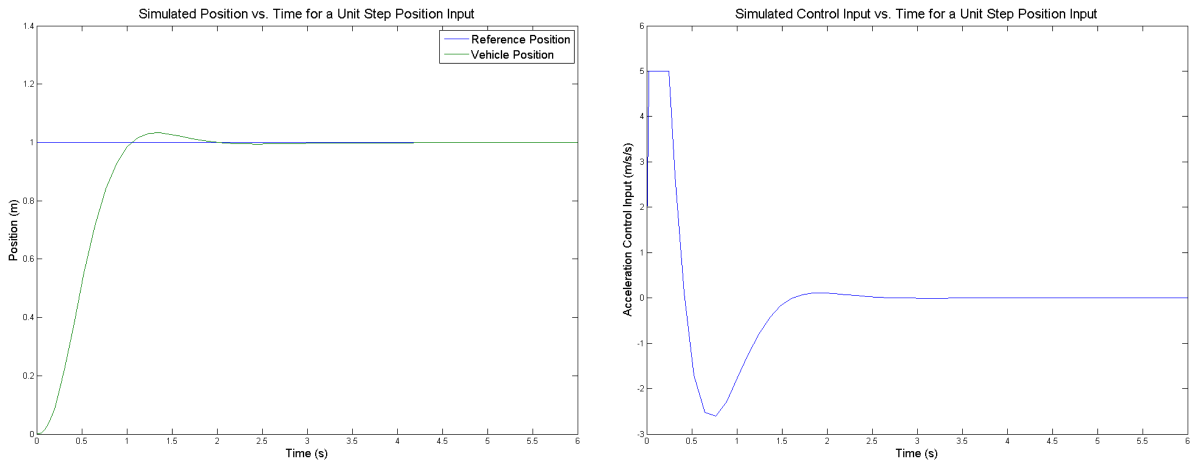
Using root locus techniques, the outer loop PID controller, which regulates the vehicle position using acceleration inputs, was designed to be

$$\frac{u(s)}{e(s)} = \frac{125(2s+1)}{s^2+20s+100} \tag{5.1}$$

where $u(s)$ is the target accelerations and $e(s)$ is the position errors (both in the frequency domain). Using the bilinear approximation, this PID controller was discretized at 50 Hz to match the rate the vehicle position is updated by the OptiTrack system. The discretized controller has the form

$$\frac{u(z)}{e(z)} = \frac{2.076z^2+0.02066z-2.056}{z^2-1.636z+0.6694} \tag{5.2}$$

The response of this controller for an input step of 1 m is shown in Figure 5.4. Note that a saturator of $\pm 5\frac{\mathrm{m}}{\mathrm{s}^2}$ was used to ensure that the commanded accelerations stay within reasonable limits. Despite the acceleration saturating near the beginning of the response in Figure 5.2(b), the vehicle position still converges quickly to the reference input, with only a small overshoot and minimal oscillations, in Figure 5.2(a).



(a) Vehicle position over time.

(b) Control input over time.

Figure 5.2: Response of PID position controller to a unit step input.

5.1.3 Results

A reference path consisting of a straight segment, three corner segments, and a final straight segment was used to test how well the motion primitives can be tracked with the AR.Drone. The experimental setup is shown in Figure 5.3, where the reference path passes through all three hoops. Due to limited flight space, the velocities bounds were set to be 0.75 $\frac{\mathrm{m}}{\mathrm{s}}$. The results of two typical runs are shown in Figure 5.4 (note that the vehicle is moving clockwise). Out of the total 20 runs performed, the maximum bound on the crosstrack error was approximately 0.30 m. One interesting observation is that the crosstrack error during the last straight segment, where the vehicle is decelerating to a stop, tends to be higher than the rest of the path. This is because the quadrotor is undergoing rapid transitions from acceleration to deceleration and finally to hover in a very short amount of time. This causes the vehicle to start surpassing the region of operation in which the assumption of decoupled dynamics is valid, and thus the performance on the controller is degraded. More advanced quadrotor control strategies that can more robustly handle dynamic coupling and non-linearities will be studied in future work.

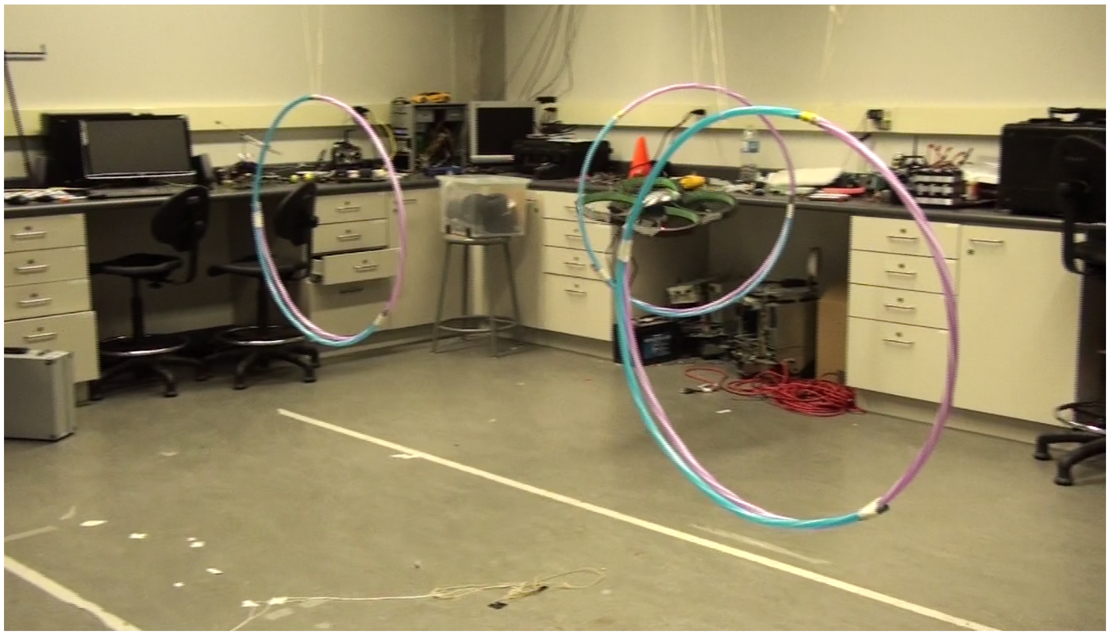
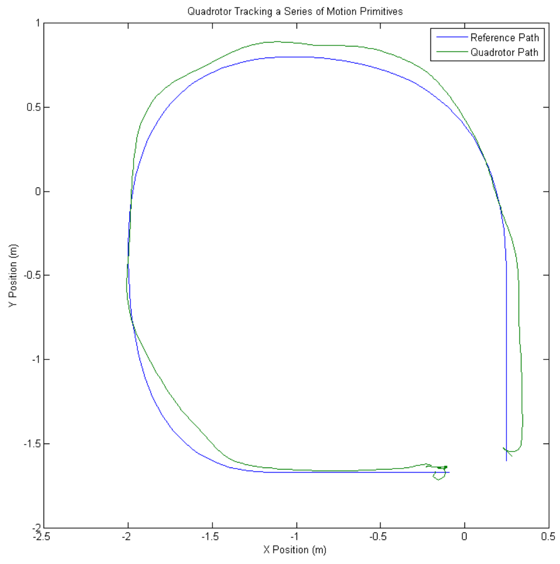


Figure 5.3: Motion primitive tracking experiment with the AR.Drone.

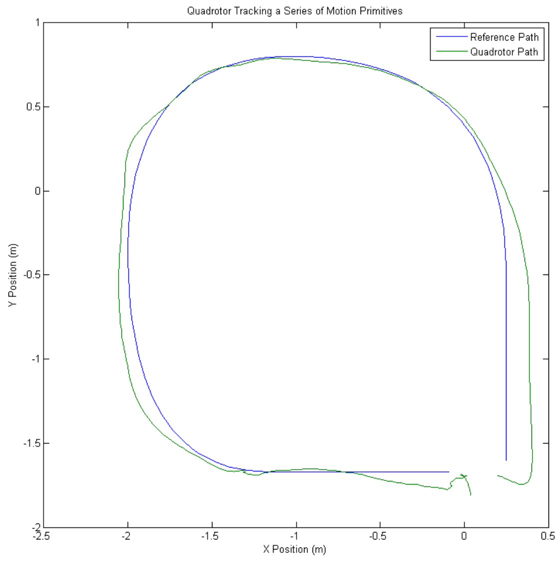
5.2 Navigating Unknown Environments with a Skid-steer Ground Vehicle

5.2.1 Experimental Platform

The full PRM based motion planner for unknown environments including the transformation algorithm was implemented on the custom built skid-steer UGV shown in Figure 5.5. This platform was designed and built by members of the Waterloo Autonomous Vehicles Lab and the University of Waterloo Robotics Team for the 2011 Intelligent Ground Vehicle Competition (IGVC), where the vehicle received second place in the design competition.



(a) Run 1



(b) Run 2

Figure 5.4: AR.Drone tracking a series of motion primitives.

The chassis is 0.85 m wide, 1.05 m long, and 1.7 m tall, and weighs approximately 300 lbs without any external payload. It is driven by 4 brushed DC motors, and can maintain a speed of 4.5 $\frac{m}{s}$. Each motor has a shaft mounted quadrature encoder which allow for closed loop velocity control. The vehicle is equipped with Mecanum wheels that allow for omni-directional control, which allows the vehicle to move laterally and independent of yaw (i.e. removes the non-holonomic constraints on the vehicle). However, this capability is not used during the presented experiments in order to demonstrate that the proposed algorithms can also be applied to non-holonomic vehicles with zero turning radius.

For localization, the vehicle has a NovAtel OEMV-3 GPS receiver that was used with an OmniSTAR differential GPS service to obtain GPS position information with sub 0.1 m accuracy in the horizontal plane. The vehicle heading is measured using a MicroStrain 3DM-GX3-25 IMU sensor. Obstacle detection is done using a front mounted SICK LMS-111 LIDAR, which has a scanning angle of 270° and a maximum range of about 20 m. For the presented experiments, the scanning angle is restricted to 180° to avoid detecting parts of the vehicle, and the maximum range is reduced to 5 m to prevent the LIDAR from picking up the ground due to uneven terrain. All of the algorithms and controls ran on a Toshiba Tecra laptop (with a 2.66 GHz dual core Intel i7 processor, 4 GB of RAM, and a

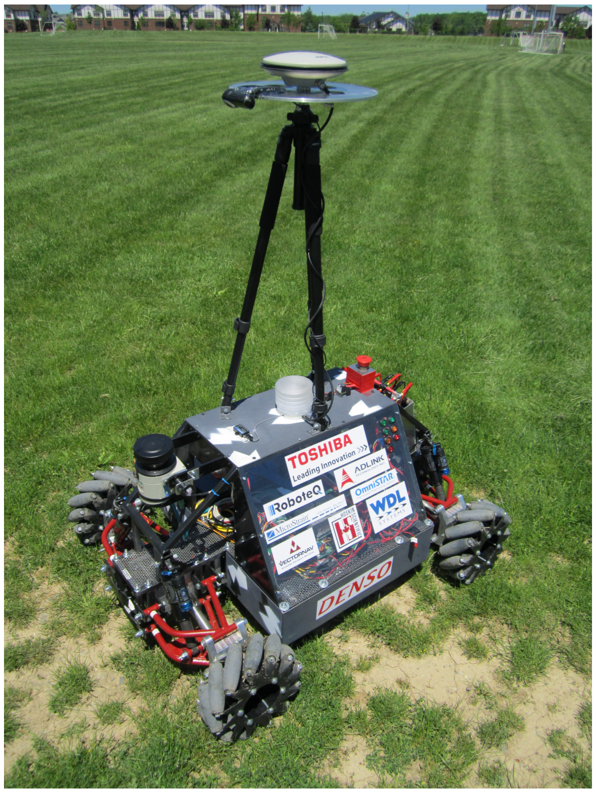


Figure 5.5: Custom built skid-steer UGV (University of Waterloo's 2011 IGVC entry).

128 GB solid state hard drive) that is mounted onboard the vehicle.

5.2.2 Vehicle Model and Control

The vehicle was modeled using the simple kinematic skid-steer model presented in Section 3.3.3, and controlled using the control strategy presented in [25]. This controller has been proven to be asymptotically stable, and is commonly used to control kinematic models of non-holonomic UGVs, such as those driven by Ackermann steering. The controller is given by

$$\dot{\psi}_z = e_{yaw} + \tan^{-1}\left(\frac{ke_{ct}}{v_x}\right) \tag{5.3}$$

where $\dot{\psi}_z$ is the desired yaw rate in the body frame (i.e. the steering control input), e_{yaw} is the vehicle heading error, k is a tunable controller gain, e_{ct} is the crosstrack error, and v_x

is the forward velocity of the vehicle. In simulation, the kinematic skid-steer vehicle model successfully converged onto the reference path, from an initial heading and crosstrack error, using this controller (Figure 5.6).

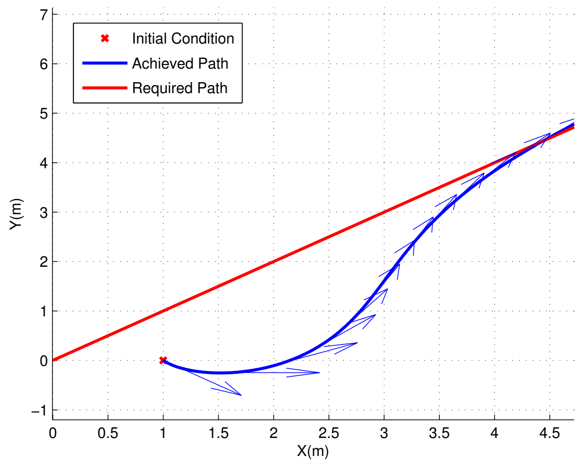


Figure 5.6: Path tracking simulation of a skid-steer vehicle under the proposed control law.

The forward velocity of the vehicle, v_x, is regulated by closed loop controllers on the motor drivers, which estimate the velocity using the encoders mounted on the motor shafts. Instead of commanding a constant v_x, this term was calculated for each motion primitive. First, the total heading change over each segment was found, and then assuming some maximum yaw rate, the minimum time required to perform this heading transition was calculated. Then, the maximum forward velocity was approximated by dividing the total length of the segment by this time. In the presented experiments, the maximum forward velocity was set to 1.0 $\frac{\text{m}}{\text{s}}$, the maximum yaw rate was set to 0.3 $\frac{\text{rad}}{\text{s}}$, and k was set to 1.0.

5.2.3 Results

Occupancy Grid Mapping

In order to successfully navigate unknown environments, it is critical that the mapping algorithm is reliable and accurate. For the presented experiments, the Octomap library in ROS was used to perform occupancy grid mapping using the front mounted LIDAR on the vehicle. To verify the quality of the generated maps, the vehicle was used to map the environment shown in Figure 5.7. The obstacles were laid out according to Figure 5.8(a) (note that the orange circles are not drawn to scale). The vehicle was driven through the obstacles and the occupancy grid map shown in Figure 5.8(b) was generated (note that each grid cell represents a 1.0 m $\times$ 1.0 m area and the obstacles are marked in blue). After three separate runs, the average values for the inter-obstacle distances were found to be: $L_1 = 7.1$ m, $L_2 = 6.8$ m, $L_3 = 5.2$ m, and $L_4 = 4.8$ m. These values suggest that the generated occupancy grid maps are accurate to within 0.2 m. Given the scale of the environment and the size of the vehicle, this result is quite sufficient for path planning purposes.

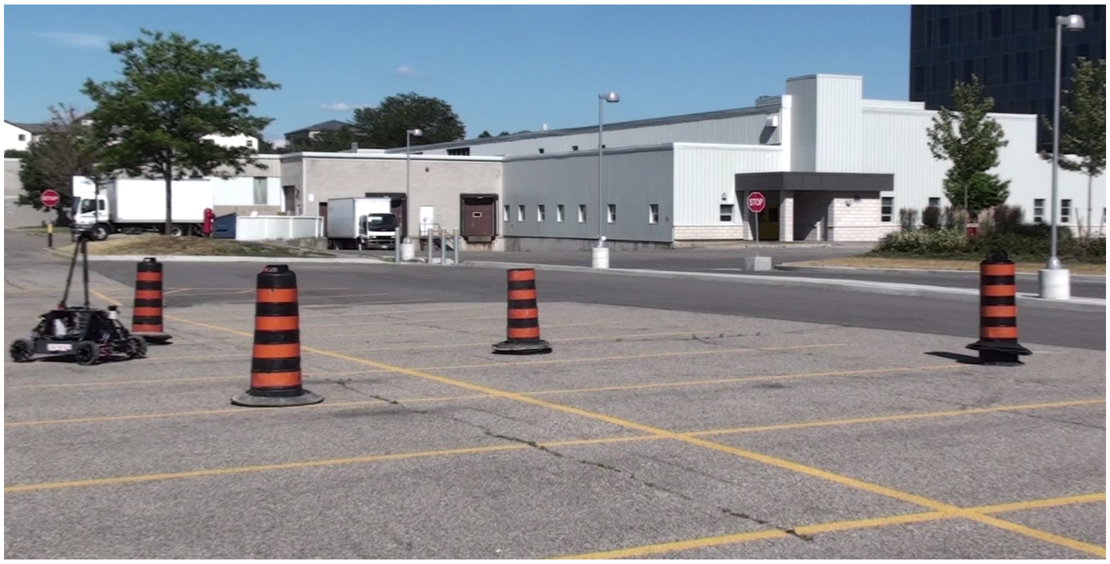
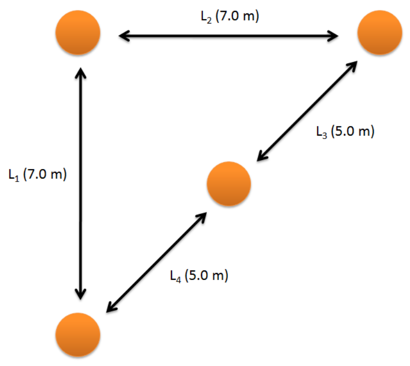


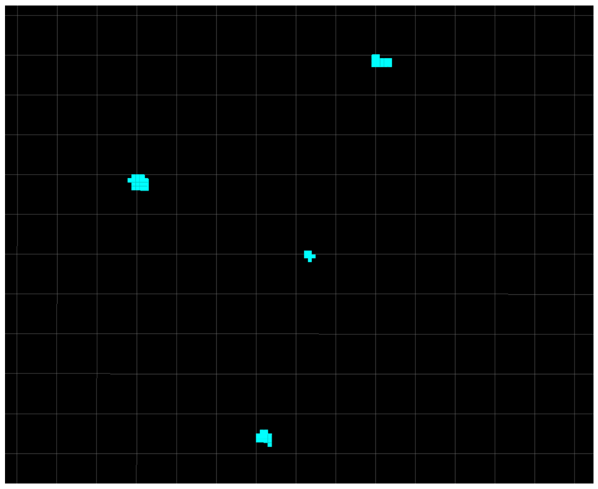
Figure 5.7: Experiment to to verify the accuracy of the mapping algorithm.

Navigating through Unknown Environments

The vehicle used the proposed motion planner to navigate through two different unknown environments. The first environment (Figure 5.9) was small in area, but had a high obstacle density. This made it challenging to navigate because there was very little clearance between the obstacles. The results of a typical run through the first environment is shown in Figure 5.10, where the reference path is shown in yellow, the obstacles are marked in blue, and each grid cell represents a 1.0 m $\times$ 1.0 m area (note that the vehicle is traveling towards the right).



(a) Obstacle layout.



(b) Constructed map.

Figure 5.8: Obstacle layout and the resulting occupancy grid map.

The second environment (Figure 5.11) was much larger in area, but contains relatively sparse obstacles, so there was lots of free space through which the vehicle could traverse. The main purpose of this test was to verify the scalability of the proposed motion planner. The results of a typical run through the second environment are shown in Figure 5.12. One interesting observation is that in sparse environments, the vehicle sometimes takes large and unnecessary detours around obstacles, as can be seen in Figure 5.12(f) - 5.12(g). This issue occurs when local sampling fails to generate a solution path in the vicinity of the old path, so the graph search finds a solution path using other nodes that are farther away. This issue can be mitigated by generating denser PRM graphs, or by checking for more optimal paths even when there is imminent collision along the current solution path. However, the potential benefits these two methods come at the cost of computational efficiency, and thus should be studied further in future work.

Finally, the computational efficiency of the proposed motion planner successfully met the sufficiency requirements for online path planning (i.e. capable of replanning a path segment in less time than it takes the vehicle to traverse it). After 5 runs, the maximum runtime for replanning an entire solution path in the first environment was 0.63 s. In the second environment, the maximum runtime for replanning an entire solution path was 0.82 s after 5 runs. In comparison, even the shortest path segments in the first environment required at least 5 seconds to traverse. As well, the runtime increase for the

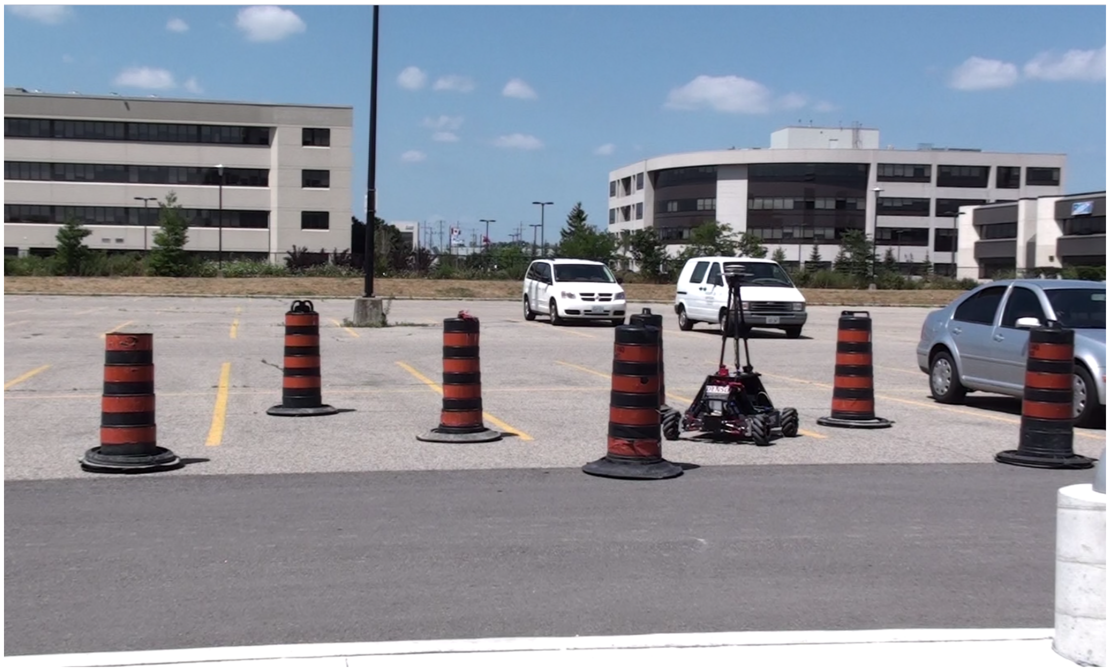
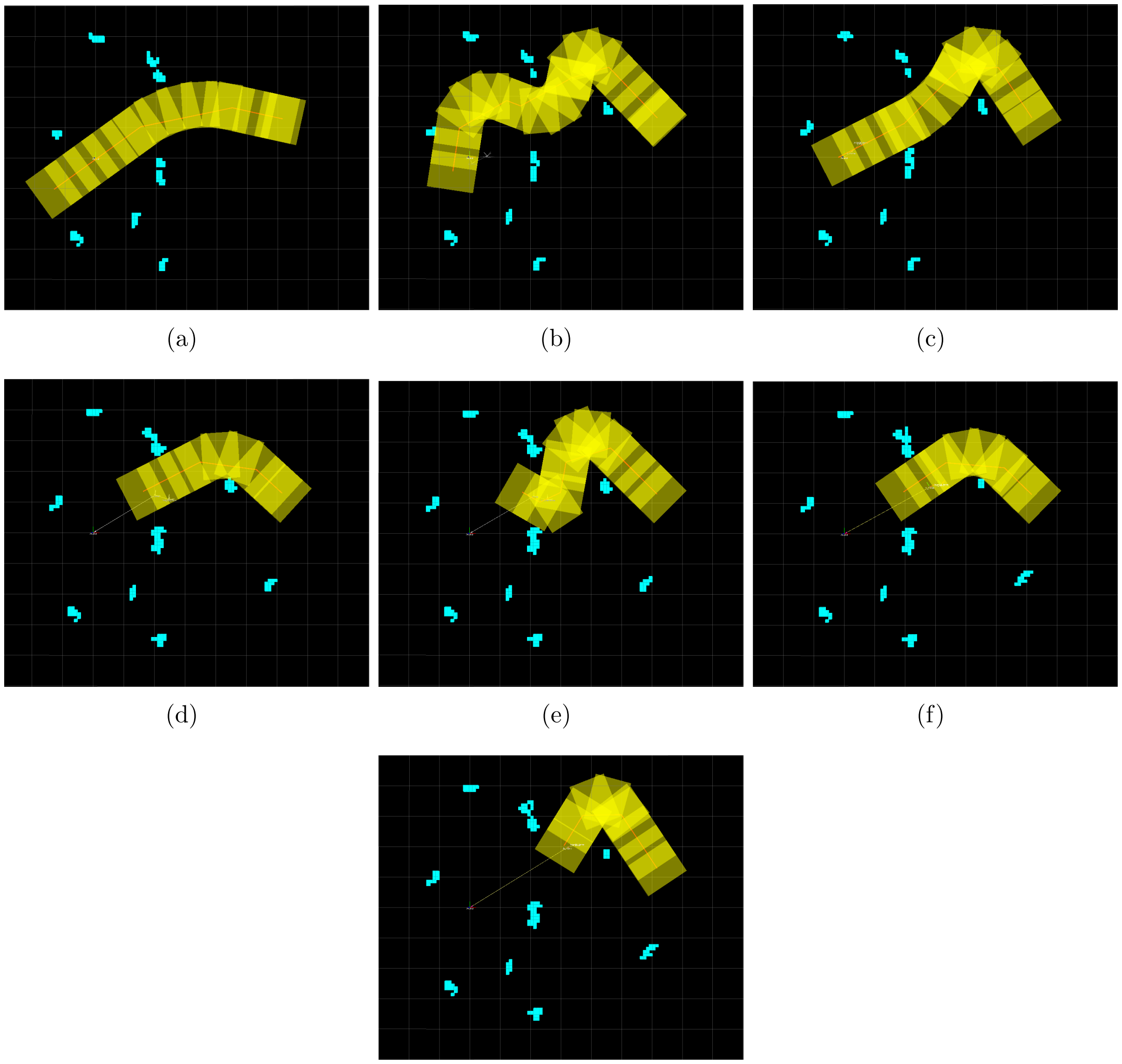


Figure 5.9: Navigation experiment in an environment with dense obstacles.

larger environment was expected, because collision checking must be performed over longer distances and will thus be slower. However, as the environments increase in size, the path segments will also become longer thus allowing more time for replanning.



(a) (b) (c) (d) (e) (f) (g)

Figure 5.10: Skid-steer vehicle navigating through a small unknown environment with dense obstacles. (a)-(g) shows the solution path after each successive replan. Note that the vehicle is traveling from left to right.

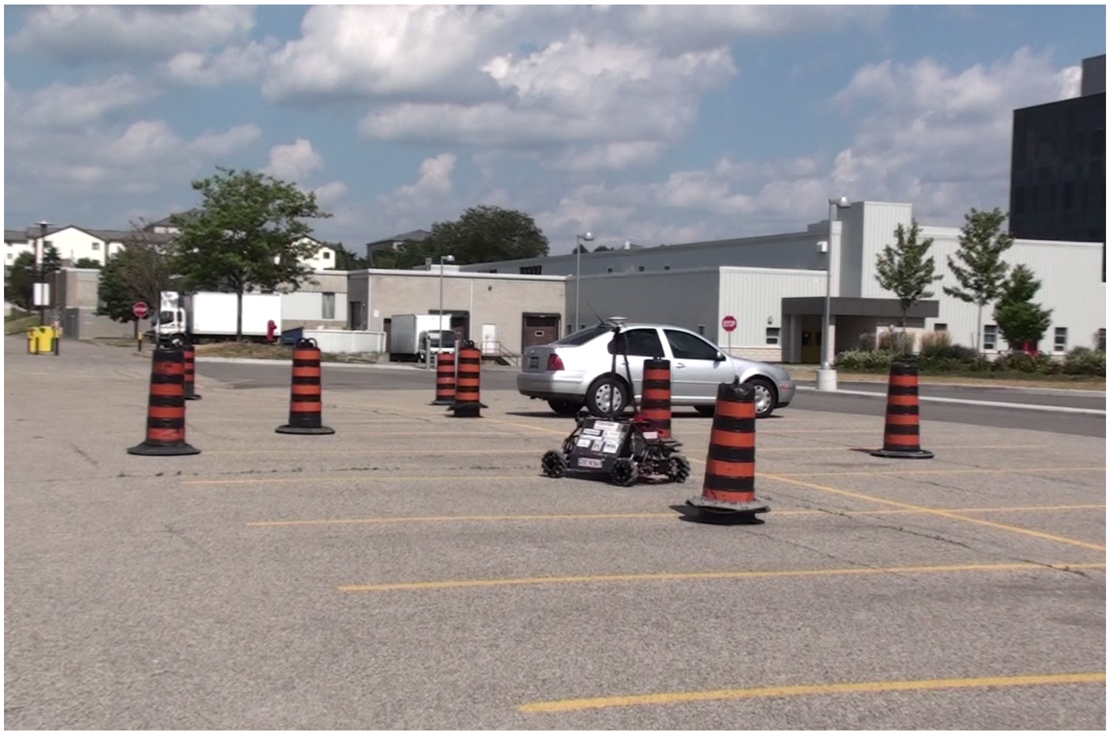


Figure 5.11: Navigation experiment in an environment with sparse obstacles.

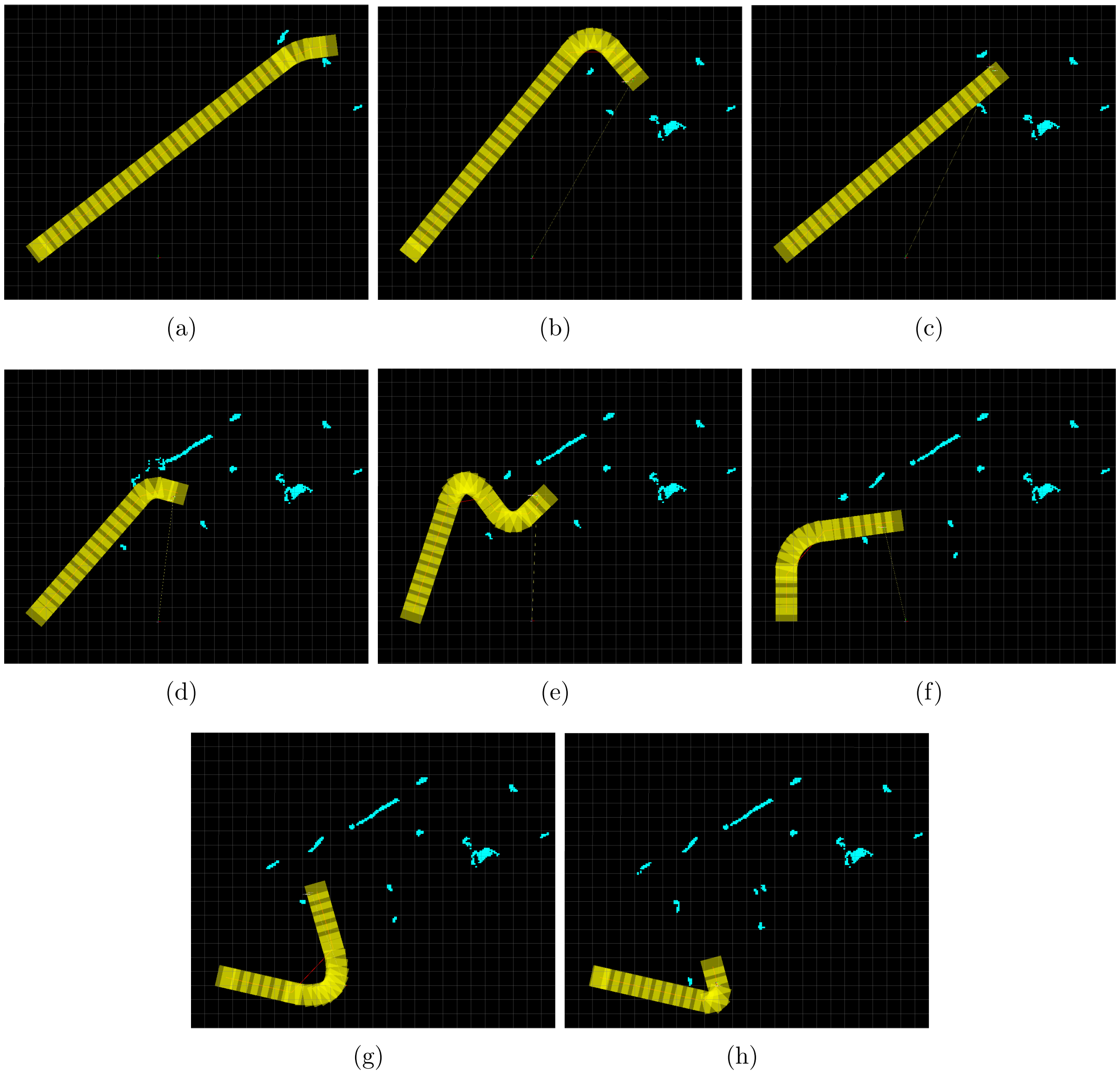


Figure 5.12: Skid-steer vehicle navigating through a large unknown environment with sparse obstacles. (a)-(h) shows the solution path after each successive replan. Note that the vehicle is traveling from right to left.

Chapter 6

Conclusions

Unmanned vehicles provide numerous benefits by allowing dangerous and tedious tasks to be performed more safely and efficiently. They are commonly employed in many practical applications such as search and rescue, surveillance, exploration and mapping, sample collection, and fault inspection. In many cases, such as when multiple vehicles are deployed or when a vehicle is deployed to a very remote location, human inputs are frequently unavailable. Thus, it is important that these vehicles are capable of operating autonomously.

In order to operate autonomously, a vehicle must be able to navigate safely and efficiently through its environment. Therefore, motion planning is a key problem in the study of unmanned vehicles. Unfortunately, many of the existing motion planners suffer from limitations such as inability to calculate paths online, susceptibility to getting stuck in local minima, failure to consider kinodynamic vehicle constraints, or dependence on prior knowledge of the environment. The work presented in this thesis aims to mitigate these issues by developing a motion planner that is capable of online navigation through unknown environments while satisfying kinodynamic vehicle constraints.

The proposed motion planner calculates the solution paths using a two step process. First, a PRM based algorithm is used to find a candidate collision-free path through the environment. As newly discovered obstacles invalidate this path, the algorithm will perform a replan using the most recent environment map to find a new candidate path. The second step applies the novel heuristic re-sampling and transformation algorithms to the candidate path found by the PRM based planner in the first step. This transforms the piece-wise linear path into a series of kinodynamically feasible motion primitives that the vehicle can track without exceeding its dynamic limits. In addition, several techniques for increasing the computational efficiency of PRM based motion planners are presented. These include targeted sampling methods, such as local sampling and the orthogonal bridge test, and a PRM regeneration technique to prevent the PRM graph from growing increasingly large over time.

The proposed motion planner was experimentally verified using an AR.Drone quadrotor and a custom built skid-steer UGV. The AR.Drone was used to to traverse a series of motion primitives generated by the transformation algorithm. This ensures that the motion primitives are indeed kinodynamically feasible and can be tracked with small errors. The AR.Drone managed to traverse a series of 5 motion primitives with a maximum crosstrack error of 0.30 m. This result is very promising considering the simplicity of the vehicle model and position controller used. The full motion planner was then implemented on a custom built skid-steer UGV. Using the front mounted LIDAR and the Octomap library, the vehicle was able to successfully map the obstacles in the environment with an accuracy of 0.2 m. Then, the vehicle was made to navigate through two unknown environments: a small environment with dense obstacles and a large environment with sparse obstacles. In both cases, the vehicle as able to successfully reach the desired destination. The maximum computational runtimes observed for replanning a path during the experiments were 0.63 sand 0.82 s for the dense and sparse environments respectively, while the shortest path segments required at least 5 s to traverse. Based on these results, the proposed motion planner shows great potential for online motion planning through unknown environments.

In conclusion, the proposed motion planner showed great potential for online path planning in unknown environments during simulations and experiments. Several aspects of the presented algorithms will be studied in more detail during future work. These include finding better methods to prevent large inefficient detours during replanning, and to analyze the diminishing effectiveness of the PRM regeneration algorithm over long periods of time. As well, to allow for better tracking in the presence of coupled dynamics and non-linearities, more effective control strategies for the quadrotor UAV will be examined. Finally, the proposed motion planner assumes static obstacles and the availability of localization information. In future research, it will be interesting to extend the algorithms and results presented in this thesis to environments that contain dynamic obstacles or lack localization information.

References

[1] E.U. Acar, H. Choset, A.A. Rizzi, P.N. Atkar, and D. Hull. Morse decompositions for coverage tasks. *The International Journal of Robotics Research*, 21(4):331–344, 2002.

[2] N.M. Amato, O.B. Bayazit, L.K. Dale, C. Jones, and D. Vallejo. OBPRM: An obstacle-based PRM for 3D workspaces. In *Robotics: The Algorithmic Perspective: 1998 Workshop on the Algorithmic Foundations of Robotics*, WAFR '98, pages 155–168, Houston, TX, United States, 1998. A. K. Peters, Ltd.

[3] F. Aurenhammer. Voronoi diagrams – a survey of a fundamental geometric data structure. *ACM Computing Surveys (CSUR)*, 23(3):345–405, 1991.

[4] J. Barraquand, B. Langlois, and J.C. Latombe. Numerical potential field techniques for robot path planning. *Systems, Man and Cybernetics, IEEE Transactions on*, 22(2):224–241, 1992.

[5] K. Belghith, F. Kabanza, L. Hartman, and R. Nkambou. Anytime dynamic path-planning with flexible probabilistic roadmaps. In *Robotics and Automation, 2006. Proceedings. 2006 IEEE/RSJ International Conference on*, pages 2372–2377, Orlando, FL, USA, 2006.

[6] I. Boada, N. Coll, N. Madern, and J.A. Sellares. Approximations of 2D and 3D generalized voronoi diagrams. *International Journal of Computer Mathematics*, 85(7):1003–1022, 2008.

[7] R. Bohlin and L.E. Kavraki. Path planning using lazy prm. In *Robotics and Automation, 2000. Proceedings. 2000 IEEE/RSJ International Conference on*, volume 1, pages 521–528, San Francisco, CA, USA, 2000.

[8] V. Boor, M.H. Overmars, and A.F. van der Stappen. The gaussian sampling strategy for probabilistic roadmap planners. In *Robotics and Automation, 1999. Proceedings.*

1999 IEEE/RSJ International Conference on, volume 2, pages 1018–1023, Detroit, MI, USA, 2002.

[9] P.M. Bouffard and S.L. Waslander. A hybrid randomized/nonlinear programming technique for small aerial vehicle trajectory planning in 3D. In *Intelligent Robots and Systems, 2009. Proceedings. 2001 IEEE/RSJ International Conference on*, pages 63–68, St. Louis, MO, USA, 2009.

[10] A. Bowyer. Computing dirichlet tessellations. *The Computer Journal*, 24(2):162, 1981.

[11] L. Chen and P. McKerrow. Modelling the lama coaxial helicopter. In *In Australasian Conference on Robotics & Automation*, Brisbane, QLD, Australia, 2007.

[12] P. Cheng and S.M. LaValle. Reducing metric sensitivity in randomized trajectory design. In *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, volume 1, pages 43–48, Maui, HI, USA, 2001.

[13] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to algorithms*. The MIT Press, Cambridge, MA, USA, 2nd edition, 2001.

[14] T. Ersson and X. Hu. Path planning and navigation of mobile robots in unknown environments. In *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, volume 2, pages 858–864, Maui, HI, USA, 2001.

[15] S. Fortune. A sweepline algorithm for voronoi diagrams. *Algorithmica*, 2(1):153–174, 1987.

[16] E. Frazzoli, M.A. Dahleh, and E. Feron. Real-time motion planning for agile autonomous vehicles. *Journal of Guidance Control and Dynamics*, 25(1):116–129, 2002.

[17] E. Frazzoli, M.A. Dahleh, and E. Feron. Maneuver-based motion planning for nonlinear systems with symmetries. *Robotics, IEEE Transactions on*, 21(6):1077–1091, 2005.

[18] I. Garcia and J.P. How. Improving the efficiency of rapidly-exploring random trees using a potential function planner. In *Decision and Control, 2005 and European Control Conference, 2005. Proceedings. 44th IEEE Conference on*, pages 7965–7970, Seville, Spain, 2005.

[19] S. Garrido, L. Moreno, M. Abderrahim, and F. Martin. Path planning for mobile robot navigation using voronoi diagram and fast marching. In *Intelligent Robots*

and Systems, 2006. Proceedings. 2006 IEEE/RSJ International Conference on, pages 2376–2381, Beijing, China, 2006.

[20] S.I. Grossman. *Elementary Linear Algebra*, pages 395–396. Brooks/Cole Thomson Learning, Toronto, ON, Canada, fifth edition, 1994.

[21] H. Han-Pang and L. Pei-Chien. A real-time algorithm for obstacle avoidance of autonomous mobile robots. *Robotica*, 10(03):217–227, 1992.

[22] P.E. Hart, N.J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, 1968.

[23] K.E. Hoff III, J. Keyser, M. Lin, D. Manocha, and T. Culver. Fast computation of generalized voronoi diagrams using graphics hardware. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '99, pages 277–286, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[24] G.M. Hoffman, H. Huang, S. L. Waslander, and C.J. Tomlin. Quadrotor helicopter flight dynamics and control: Theory and experiment. In *Proceedings of the AIAA Guidance, Navigation and Control Conference and Exhibit*, Hilton Head, SC, USA, 2007.

[25] G.M. Hoffmann, C.J. Tomlin, M. Montemerlo, and S. Thrun. Autonomous automobile trajectory tracking for off-road driving: Controller design, experimental validation and racing. In *American Control Conference, 2007. ACC'07*, pages 2296–2301, New York, NY, USA, 2007.

[26] G.M. Hoffmann, S.L. Waslander, and C.J. Tomlin. Quadrotor helicopter trajectory tracking control. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference and Exhibit*, Honolulu, HI, USA, 2008.

[27] D. Hsu, T. Jiang, J. Reif, and Z. Sun. The bridge test for sampling narrow passages with probabilistic roadmap planners. In *Robotics and Automation, 2003. Proceedings. 2003 IEEE/RSJ International Conference on*, volume 3, pages 4420–4426, Taipei, Taiwan, 2003.

[28] D. Hsu, L.E. Kavraki, J.C. Latombe, R. Motwani, and S. Sorkin. On finding narrow passages with probabilistic roadmap planners. In *Robotics: The Algorithmic Perspective: 1998 Workshop on the Algorithmic Foundations of Robotics*, WAFR '98, pages 141–153, Houston, TX, United States, 1998. A. K. Peters, Ltd.

[29] D. Hsu, J.C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. In *Robotics and Automation, 1997. Proceedings. 1997 IEEE/RSJ International Conference on*, volume 3, pages 2719–2726, Albuquerque, NM , USA, 1997.

[30] M. Kalisiak and M. van de Panne. Rrt-blossom: Rrt with a local flood-fill behavior. In *Robotics and Automation, 2006. Proceedings. 2006 IEEE International Conference on*, pages 1237–1242, Orlando, FL, USA, 2006.

[31] I. Kamon and E. Rivlin. Sensory-based motion planning with global proofs. *Robotics and Automation, IEEE Transactions on*, 13(6):814–822, 1997.

[32] S. Karaman and E. Frazzoli. Incremental sampling-based algorithms for optimal motion planning. In *Robotics: Science and Systems (RSS)*, Zaragoza, Spain, 2010.

[33] T. Karatas and F. Bullo. Randomized searches and nonlinear programming in trajectory planning. In *Decision and Control, 2001. Proceedings. 40th IEEE Conference on*, volume 5, pages 5032–5037, Orlando, FL, USA, 2001.

[34] L.E. Kavraki, P. Svestka, J.C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on*, 12(4):566–580, 1996.

[35] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *science*, 220(4598):671, 1983.

[36] S. Koenig and M. Likhachev. Improved fast replanning for robot navigation in unknown terrain. In *Robotics and Automation, 2002. Proceedings. 2002 IEEE/RSJ International Conference on*, volume 1, pages 968–975, Washington, DC, USA, 2002.

[37] E. Koyuncu and G. Inalhan. A probabilistic b-spline motion planning algorithm for unmanned helicopters flying in dense 3d environments. In *Intelligent Robots and Systems, 2008. Proceedings. 2008 IEEE/RSJ International Conference on*, pages 815–821, Nice, France, 2008.

[38] K. Kozłowski and D. Pazderski. Modeling and control of a 4-wheel skid-steering mobile robot. *International Journal of Applied Mathematics and Computer Science*, 14(4):477–496, 2004.

[39] K.M. Krishna and P.K. Kalra. Perception and remembrance of the environment during real-time navigation of a mobile robot. *Robotics and Autonomous Systems*, 37(1):25–51, 2001.

[40] J.J. Kuffner Jr and S.M. LaValle. RRT-connect: An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings. 2000 IEEE/RSJ International Conference on*, volume 2, pages 995–1001, San Francisco, CA, USA, 2000.

[41] S.M. LaValle. Rapidly-exploring random trees: A new tool for path planning. Technical report, Computer Science Dept., Iowa State University, Ames, IA, USA, 1998.

[42] S.M. LaValle. *Planning algorithms*. Cambridge University Press, New York, NY, USA, 2006.

[43] W. Li. Fuzzy-logic-based reactive behavior control of an autonomous mobile system in unknown environments. *Engineering Applications of Artificial Intelligence*, 7(5):521–531, 1994.

[44] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun. Anytime dynamic A*: An anytime, replanning algorithm. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pages 262–271, Monterey, CA, USA, 2005.

[45] M. Likhachev, G. Gordon, and S. Thrun. ARA*: Anytime A* search with provable bounds on sub-optimality. 2003.

[46] T. Lozano-Pérez and M.A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, 1979.

[47] V. Lumelsky and A. Stepanov. Dynamic path planning for a mobile automaton with limited information on the environment. *Automatic Control, IEEE Transactions on*, 31(11):1058–1063, 1986.

[48] L.E. Parker, B. Birch, and C. Reardon. Indoor target intercept using an acoustic sensor network and dual wavefront path planning. In *Intelligent Robots and Systems, 2003. Proceedings. 2003 IEEE/RSJ International Conference on*, volume 1, pages 278–283, Las Vegas, NV, USA, 2003.

[49] M. Pivtoraiko and A. Kelly. Generating near minimal spanning control sets for constrained motion planning in discrete state spaces. In *Intelligent Robots and Systems, 2005. Proceedings. 2005 IEEE/RSJ International Conference on*, pages 3231–3237, Edmonton, AB, Canada, 2005.

[50] L. Podsdkowski, J. Nowakowski, M. Idzikowski, and I. Vizvary. A new solution for path planning in partially known or unknown environment for nonholonomic mobile robots. *Robotics and Autonomous Systems*, 34(2-3):145–152, 2001.

[51] L. Podsedkowski. Path planner for nonholonomic mobile robot with fast replanning procedure. In *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, volume 4, pages 3588–3593, Leuven, Belgium, 1998.

[52] E. Rimon and D.E. Koditschek. Exact robot navigation using artificial potential functions. *Robotics and Automation, IEEE Transactions on*, 8(5):501–518, 1992.

[53] D. Schafroth, C. Bermes, S. Bouabdallah, and R. Siegwart. Modeling and system identification of the mufly micro helicopter. *Journal of Intelligent & Robotic Systems*, 57(1–4):27–47, 2009.

[54] T. Schouwenaars, B. De Moor, E. Feron, and J. How. Mixed integer programming for multi-vehicle path planning. In *Proceedings of European Control Conference*, pages 2603–2608, Porto, Portugal, 2001.

[55] J.A. Sethian et al. Level set methods and fast marching methods. *Journal of Computing and Information Technology*, 11(1):1–2, 2003.

[56] A. Stentz. Optimal and efficient path planning for partially known environments. In *Robotics and Automation, 1994. Proceedings. 1994 IEEE/RSJ International Conference on*, volume 4, pages 3310–3317, San Diego, CA, USA, 1994.

[57] A. Stentz. The focussed D* algorithm for real-time replanning. In *International Joint Conference on Artificial Intelligence*, volume 14, pages 1652–1659, Montreal, QC, Canada, 1995.

[58] S. Thrun, W. Burgard, and D. Fox. *Probabilistic robotics.* The MIT Press, Cambridge, MA. USA, 2006.

[59] M.P. Vitus, V. Pradeep, G. Hoffmann, S.L. Waslander, and C.J. Tomlin. Tunnel-milp: Path planning with sequential convex polytopes. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference and Exhibit*, Honolulu, HI, USA, 2008.

[60] M. Wang and J.N.K. Liu. Fuzzy logic-based real-time robot navigation in unknown environment with dead ends. *Robotics and Autonomous Systems*, 56(7):625–643, 2008.

[61] D.F. Watson. Computing the n-dimensional Delaunay tessellation with application to voronoi polytopes. *The computer journal*, 24(2):167, 1981.

[62] K.M. Wurm, A. Hornung, M. Bennewitz, C. Stachniss, and W. Burgard. OctoMap: A probabilistic, flexible, and compact 3D map representation for robotic systems. In *Robotics and Automation, 2010. Workshop on Best Practice in 3D Perception and Modeling for Mobile Manipulation. Proceedings. 2010 IEEE/RSJ International Conference on*, Anchorage, AK, USA, 2010.

[63] W.L. Xu, S.K. Tso, and Z.K. Lu. A virtual target approach for resolving the limit cycle problem in navigation of a fuzzy behaviour-based mobile robot. In *Intelligent Robots and Systems, 1998. Proceedings. 1998 IEEE/RSJ International Conference on*, volume 1, pages 44–49, Victoria, BC, Canada, 1998.

[64] A. Zelinsky, R.A. Jarvis, J. C. Byrne, and S. Yuta. Planning paths of complete coverage of an unstructured environment by a mobile robot. In *Proceedings of the International Conference on Advanced Robotics*, pages 533–538, Tokyo, Japan, 1993.

[65] X. Zou and J. Zhu. Virtual local target method for avoiding local minimum in potential field based robot navigation. *Journal of Zhejiang University-Science A*, 4(3):264–269, 2003.