Sampling-Based Path Planner for Guided Airdrop in Urban Environments

by

Brian Le Floch

B.S., University at Buffalo (2015)

Submitted to the Department of Aeronautics an Astronautics
in partial fulfillment of the requirements for the degree of
Master of Science in Aeronautics and Astronautics
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
June 2017

© Massachusetts Institute of Technology 2017. All rights reserved.

Author ...... Signature redacted ....
Department of Aeronautics an Astronautics
May 25, 2017

Certified by......... Signature redacted ....
Jonathan How
Richard Cockburn Maclaurin Professor of Aeronautics and Astronautics
Thesis Supervisor

Certified by Signature redacted ....
Matthew Stoeckle
Member of the Technical Staff, Draper
Thesis Supervisor

Certified by.. Signature redacted ....
Louis Breger
Member of the Technical Staff, Draper
Thesis Supervisor

Accepted by ...... Signature redacted ....
Youssef M. Marzouk
Associate Professor of Aeronautics and Astronautics
Chair, Graduate Program Committee

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
JUL 11 2017
LIBRARIES
ARCHIVES

Sampling-Based Path Planner for Guided Airdrop in Urban Environments

by

Brian Le Floch

Submitted to the Department of Aeronautics and Astronautics
on May 25, 2017, in partial fulfillment of the
requirements for the degree of
Master of Science in Aeronautics and Astronautics

Abstract

Aerial resupply can deliver cargo to locations across the globe. A challenge for modern guided parafoil systems is to land accurately in complex terrain, including canyons and cities. This thesis presents the Rewire-RRT algorithm for parafoil terminal guidance. The algorithm uses Rapidly-Exploring Random Trees (RRT) to efficiently search for feasible paths through complex environments. Most importantly, Rewire-RRT provides a mechanism to build and rewire the tree to explicitly minimize the risk of collision with obstacles along each path and to minimize the expected final miss distance from the target. This key adaptation allows for parafoil guidance in urban drop zones not previously considered for airdrop operations. The Rewire-RRT algorithm is first developed and tested in two dimensions and demonstrated to have greater performance than RRT for simple dynamical systems, finding paths that are shorter and safer than those found by RRT. Then, Rewire-RRT is shown to be an effective path planner for a guided parafoil with complex dynamics. Paths planned by Rewire-RRT better meet the performance objectives of guided parafoils than those planned by RRT. Finally, simulation results show that Rewire-RRT performs better than state-of-the-art terminal guidance strategies for guided parafoils when the target location is cluttered with multiple three-dimensional obstacles.

Thesis Supervisor: Jonathan How
Title: Richard Cockburn Maclaurin Professor of Aeronautics and Astronautics

Thesis Supervisor: Matthew Stoeckle
Title: Member of the Technical Staff, Draper

Thesis Supervisor: Louis Breger
Title: Member of the Technical Staff, Draper

Acknowledgments

Thank you to everyone who provided guidance and support towards the completion of this thesis. In particular, I appreciate the effort of my triad of advisors: Professor Jonathan How at MIT, and Matthew Stoeckle and Louis Breger at Draper. Also, thank you to the Natick Soldier Research, Development and Engineering Center (NSRDEC) for funding and inspiring this work.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Airdrop operations are used to deliver cargo to locations across the globe. The capability to deliver cargo from the air is useful when ground transportation is dangerous or otherwise infeasible. Applications range from military resupply to disaster relief [1]. It is imperative that the cargo land as accurately as possible in all conditions. Landings that are far from the intended impact point (IP) can result in damage or theft of cargo and difficult recovery efforts [2]. Historically, round unguided parachutes, such as those shown in Figure 1-1, have been used during airdrop operations [3].

More recently, greater accuracy has been achieved using guided parafoils that can autonomously steer towards the IP [5]. Guided parafoils have the additional advantage of being deployable from higher altitudes than unguided systems without a decrease in accuracy [5]. This can allow for deployment farther away from the IP, decreasing the risk to cargo aircraft that might otherwise have to make a low approach over hostile areas or rugged terrain. An example of a guided airdrop system is shown in Figure 1-2.

The three main components of a guided parafoil are a ram air canopy, an airborne guidance unit (AGU), and a payload [7], as shown in Figure 1-3. The ram air canopy slows the descent of the payload and allows for steering via asymmetric deflection of the trailing edge. The AGU contains motors used to deflect the trailing edge, along

Figure 1-1: Round parachutes used for unguided airdrop [4]

Figure 1-2: Guided airdrop parafoil system on final approach [6]

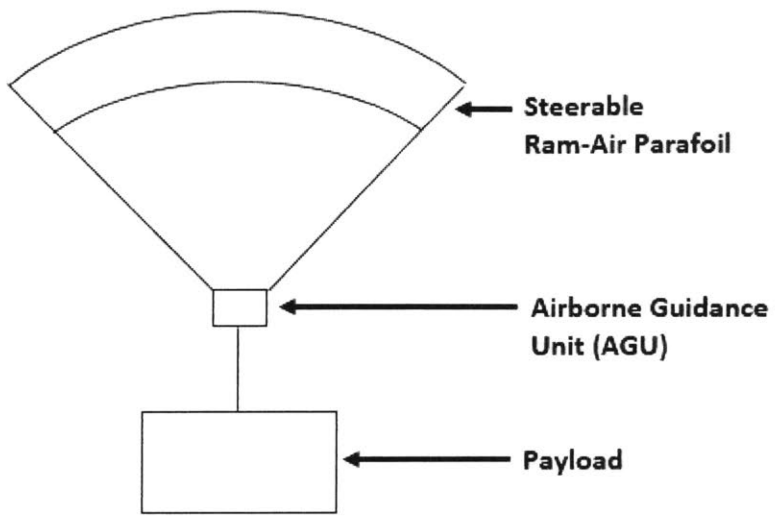


Figure 1-3: Illustration of the three main components of a guided parafoil in flight - front view (Note: illustration not to scale)

with the required hardware for autonomous guidance, navigation, and control of the parafoil. The payload, which can range from 10 to 42,000lb [8] as appropriate for the chosen canopy, is suspended below the AGU.

Guidance of parafoils is difficult due to their underactuated dynamics. They have large turning circles and little or no control over vertical descent rate [9]. Further, guided parafoils may operate in environments with highly uncertain winds. The addition of complex terrain near the target landing area, including canyons, ridges, or multiple large buildings, adds to the complexity of the guidance problem even when terrain is mapped in advance [10].

The flight profile of a guided parafoil is typically separated into three main phases [11], as shown in Figure 1-4. After a brief stabilization period, the homing phase steers the parafoil directly towards the IP. Next, the energy management phase guides the parafoil through a series of figure-eight maneuvers that decrease the system altitude while staying in proximity of the IP. Finally, the terminal guidance phase steers the parafoil to its final landing point on the ground. Terminal guidance is the most critical phase in determining parafoil accuracy, and is therefore the subject of most parafoil

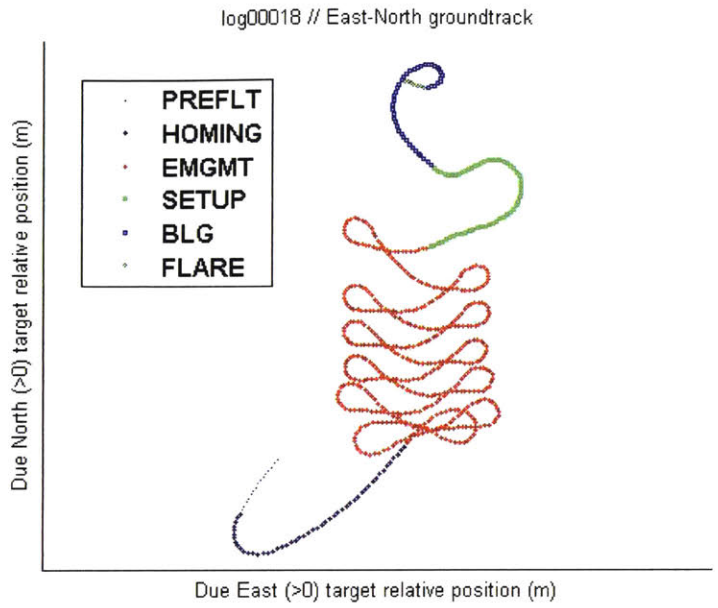


Figure 1-4: Overhead view of a guided parafoil flight profile. The two-part terminal guidance phase consists of 'SETUP' followed by 'BLG'.

guidance research.

This thesis presents a novel parafoil terminal guidance strategy. It is particularly well adapted to the challenge of landing in urban areas, where multiple large buildings may be within close proximity of the IP. This adds significant challenges to the terminal guidance problem, because the planner must be flexible enough to provide long-term guidance around multiple obstacles. Further, collisions with buildings would result in severe damage and are highly undesirable. An enhanced version of the sampling based parafoil Rapidly-Exploring Random Trees algorithm [12] is used to address these issues.

1.2 Literature Review

Several terminal guidance methods for guided airdrop have been proposed. Glide-slope-based methods use the concept of the glide-slope cone, the set of all position and heading states which, assuming constant velocity and disturbances, would guide the parafoil to the target location. Calise and Preston [13] use a series of scripted maneuvers online to estimate glide-slope parameters, then execute turning maneuvers to drive the parafoil to the glide-slope. This method requires long-term glide-slope tracking and does not consider the presense of obstacles along the planned trajectory, limiting its use in complex terrain.

Trajectory-based approaches generate reference trajectories online to optimize a pre-defined cost function, then use various control strategies to track these trajectories. The Band-Limited Guidance (BLG) algorithm [14] uses Nelder-Mead simplex search to minimize a cost function based on the parafoil's predicted terminal state. BLG guarantees that control bandwidth constraints are satisfied to ensure accurate trajectory following, and its computational efficiency enables the use of online replanning. However, BLG does not consider the possibility of collison with obstacles due to a deviation from the planned trajectory caused by changing wind conditions. Additionally, the speed of optimization is sensitive to the intial guess, starting altitude, and number of control points, leading to slow convergence to the globally optimal solution in some cases.

Work by Rogers and Slegers considers robustness to wind variations by using graphics processing units (GPUs) to parallelize a Monte Carlo simulation of possible future winds, and the resulting parafoil trajectories, based on available measurements [15, 16]. However, significant computational effort is required to run these Monte Carlo simulations online. Although the presence of environmental obstacles is considered in the vicinity of the target, terminal guidance is also assumed to begin in relative proximity to the target location due to the selected parameterization of candidate trajectories (i.e., using a single constant-rate turn and straight line segment). This approach may therefore prove difficult to implement in constrained terrain geometries (such as valleys and canyons), where robust planning and obstacle avoidance must begin from high initial altitudes, and greater path flexibility is required.

Rapidly-Exploring Random Trees (RRT) [17] are used to plan paths by the parafoil Chance-Constrained RRT (CC-RRT) algorithm [12]. Real-time wind modeling and classification is used to anticipate future disturbances, while an uncertainty-sampling technique ensures that robustness to possible future variation is efficiently maintained.

Yet, despite this success in planning robustness, one of the limitations of CC-RRT relative to state-of-the-art parafoil terminal guidance algorithms is the suboptimal nature of the RRT-based trajectory design. Analytic CC-RRT algorithm lacks a method for explicitly minimizing the risk of constraint violation and the final miss distance. A parafoil Analytic CC-RRT* [18] algorithm was considered based on the optimal RRT* algorithm [19] which is guaranteed asymptotically optimal. However, significant challenges were found in implementation of this algorithm, including developing a steering law to connect any two parafoil states in 3D space. Position and heading at the beginning and end states must be matched exactly, creating a two-point boundary value problem (BVP) that is as difficult to solve as the initial two point boundary value problem between the parafoil initial and goal states.

The CC-BLG algorithm [10] includes a cost function that can incorporate risk into the trajectory optimization framework of BLG. While this allows CC-BLG to minimize the risk of collision, it does not address the limitation of the algorithm in cluttered environments. The optimization algorithm used by BLG scales poorly with starting altitude and number of obstacles, and is very sensitive to the initial guess [14]. In a complex terrain environment that requires turns around several obstacles beginning from a high initial altitude, an accurate intial guess is unlikely and convergence to the globally optimal solution will be slow.

This thesis presents the Rewire-RRT algorithm. It uses the parafoil RRT framework to efficiently find dynamically feasible trajectories from any starting altitude through complex terrain. A cost function is developed to compare the risk of collision and predicted final miss distance at each node in the tree. Two new functions are introduced that minimize this cost function at each node during tree growth. The result is a terminal guidance algorithm that can find dynamically feasible paths through complex terrain environments that are explicitly optimized to reduce risk of collision and to minimize miss distance. Rewire-RRT is demonstrated to perform better than existing terminal guidance algorithms in complex environments.

1.3 Contributions

This thesis is divided into three main sections:

- **Chapter 2** The Rewire-RRT algorithm is presented and used to plan paths for a vehicle with simple dynamics in two dimensions. Path-planning experiments demonstrate that Rewire-RRT performs much better than RRT and nearly as well as RRT*.
- **Chapter 3** It is demonstrated that unlike RRT*, the Rewire-RRT algorithm can be easily applied to the underactuated parafoil dynamics. Experiments show that Rewire-RRT performs better than RRT at finding dynamically feasible paths for guided parafoils through complex environments.
- **Chapter 4** Rewire-RRT is compared to state-of-the-art parafoil guidance algorithms CC-BLG and CC-RRT. Simulation experiments in a high-fidelity simulator demonstrate that Rewire-RRT performs strongly in urban-style environments.

Finally, recommendations are made for future work related to these topics.

Chapter 2

Sampling-based Path Planning in Two Dimensions

2.1 Introduction

Path planning algorithms find dynamically feasible trajectories between starting and goal states. In this section, it will be assumed that the environment is pre-mapped and known to the robot during the planning process. Sampling based path planners, such as Rapidly-exploring Random Trees (RRT) [17], have been used successfully by an array of ground vehicles to find 2D trajectories through a variety of environments [20].

When planning paths for ground vehicles in two dimensions, it is often desirable to minimize the total path length between the start and goal positions. RRT is often successful at finding feasible paths through high-dimensional spaces much faster than optimization-based methods [17]. However, though RRT is probabilistically complete, it is not asymptotically optimal [19]. This means that trajectories will almost certainly be suboptimal with respect to the goal of minimizing path length. This is true even if infinite computation time is available.

The asymptotically optimal RRT-based algorithm RRT* was proposed by Karaman and Frazzoli [19]. A rewiring technique that reconnects nodes in the tree with lower-cost paths is shown to provide an asymptotic optimality guarantee under certain conditions. The rewire procedure requires the ability to connect any two existing nodes in the tree with a dynamically feasible optimal path. Analytical solutions to this two-point boundary value problem exist for systems with simple dynamics (e.g. single integrator, double integrator, Dubins vehicle [21]). In general, especially for more complex or underactuated systems, analytic solutions do not exist [22]. The underactuated dynamics of a guided parafoil are an example of a case for which this analytic solution is not known. These parafoil dynamics are thoroughly described in Chapter 3.

There have been attempts to overcome this limitation of RRT* for ground robots. Jeon et al. [23] suggest methods of relaxing the end constraint during the rewiring process and then deleting or repropagating child nodes. An alternative is the Dual-Tree RRT algorithm [24] which also relaxes the end constraint during rewiring, but keeps the original tree structure intact. The use of two trees allows for faster identification of nearest nodes using search schemes such as kd-trees [25]. Successful real-time implementation of the Dual-Tree RRT algorithm is shown on a wheeled robot.

In this chapter, path planning is considered for a vehicle with the dynamics of a simplified guided parafoil. This is done in two dimensions, ignoring the altitude dynamics of the parafoil (i.e., planning will be done in the plane parallel to the ground plane). The Rewire-RRT algorithm is introduced and evaluated. It is able to find paths that are better optimized for the system requirements (e.g., path length) than paths found by RRT and Rewire-RRT does not have the same limitations of RRT* for vehicles with complex dynamics.

2.2 The Rewire-RRT Algorithm

2.2.1 Rapidly Exploring Random Trees

The RRT algorithm grows a tree of dynamically feasible trajectories through a pre-mapped environment. It does so by sampling a random position in the 2D environment, x_{samp}. The nearest existing node to x_{samp} in Euclidean distance, x_{near}, is

Algorithm 1 RRT

```
1: Tree.init($x_0$)
2: while $t < t_{growth\_limit}$ do
3:     $x_{samp}$ = sample()
4:     $x_{near}$ = near($x_{samp}$)
5:     $x_{new}$ = steer($x_{near}$, $x_{samp}$)
6:     if exists($x_{new}$) then
7:         Tree.addNode($x_{new}$)
8:         Tree.addEdge($x_{near}$,$x_{new}$)
9:     end if
10: end while
```

identified. Then, an attempt is made to connect x_{near} to x_{samp} with a dynamically feasible path using a "steering" procedure. (Alternatively, a dynamically feasible path extending a fixed distance from x_{near} towards x_{samp} may be identified.) The ending state of this path is called x_{new}. If the path from x_{near} to x_{new} is free from obstacles, then x_{new} is added to the tree along with an edge from x_{near} to x_{new}. This tree growth method is repeated until a time limit t_{growth_limit} is exceeded. Once tree growth is complete, the best path in the tree is chosen and followed by the robot. Often, the best path is the one that gets to the goal with the shortest total path length. Algorithm 1 shows the pseudocode for the RRT tree growth procedure.

2.2.2 Rewire-RRT

Like Dual-tree RRT [24], The Rewire-RRT algorithm adds two new functions to the RRT algorithm, *chooseParent* and *reconnectTree*. The *chooseParent* function considers multiple possible nodes as parent nodes for x_{new} during tree growth. This allows the tree to be incrementally built with lower-cost paths. Once a new node is added to the tree, *reconnectTree* considers the new node as a possible parent for existing nodes in the tree. This "rewires" the tree to make new lower cost paths. The pseudocode for Rewire-RRT is shown in Algorithm 2.

Algorithm 2 Rewire-RRT

```
1: **Tree**.init($x_0$)
2: **while** $t < t_{growth\_limit}$ **do**
3:     $x_{samp}$ = sample()
4:     $X_{near}$ = nearest($x_{samp}$)
5:     [$x_{parent}$,$x_{new}$] = chooseParent($x_{samp}$,$X_{near}$)
6:     **if** exists($x_{new}$) **then**
7:         **Tree**.addNode($x_{new}$)
8:         **Tree**.addEdge($x_{parent}$, $x_{new}$)
9:         rewire($x_{new}$,$X_{near}$)
10:    **end if**
11: **end while**
```

2.2.3 Choosing the parent node

Instead of identifying the nearest node, x_{near}, Rewire-RRT identifies a set of nearest nodes, X_{near}. For each $x_{near} \in X_{near}$, an attempt is made to connect x_{near} to x_{samp} with a dynamically feasible path terminating at a new node, $x_{new\_candidate}$. If $x_{new\_candidate}$ is within a 2D radius r_e of x_{new} and the path is collision free, the cost of the node $x_{new\_candidate}$ is calculated. The cost of any node x is determined by calculating the total path length from the root node to node x. This definition of cost is chosen because the objective of the path planner is to find the feasible path of shortest length from the starting position to the goal. The candidate node with the lowest cost is chosen as x_{new}. The node x_{new} is added to the tree along with an edge from its associated parent, x_{parent}. The *chooseParent* function increases the success rate of node extension, and incrementally builds a tree that minimizes the cost associated with each node in the tree.

Consider a tree consisting of two nodes, x_0 and x_1, with an edge between them. This scenario is shown in Figure 2-1. When a random point x_{samp} is sampled from the environment, the RRT algorithm only identifies the nearest node, x_1. A feasible path from x_1 towards x_{samp} and ultimately ending at x_{new} is identified. Since the path from x_1 to x_{new} is free from obstacles, the new node and edge will be added to the tree.

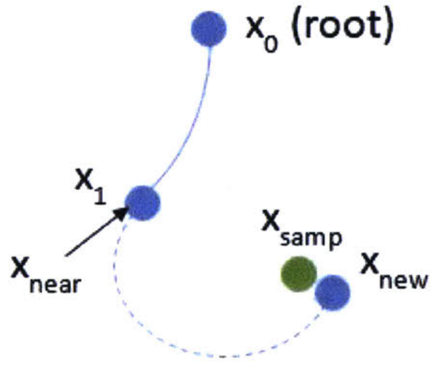
Alternatively, the Rewire-RRT algorithm identifies all the nearest nodes within

Algorithm 3 chooseParent(x_{samp},X_{near})

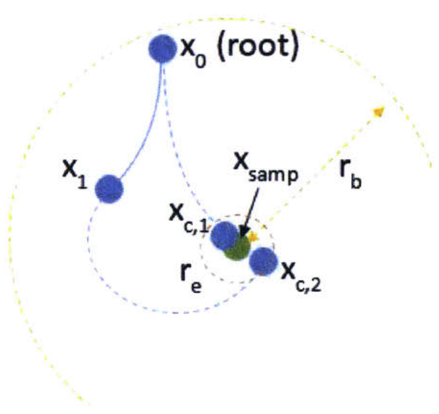
```
1: $x_{new} = null$; $x_{parent} = null$
2: $min\_cost = \infty$
3: for each $x_{near} \in X_{near}$ do
4:     $x_{new\_candidate}$ = steer($x_{near}, x_{samp}$)
5:     if exists($x_{new\_candidate}$) && $d_{horizontal} < r_e$ then
6:         if cost($x_{new\_candidate}$)< $min\_cost$ then
7:             $x_{new} = x_{new\_candidate}$
8:             $x_{parent} = x_{near}$
9:             $min\_cost$ = cost($x_{new\_candidate}$)
10:        end if
11:    end if
12: end for
13: return([$x_{new}$,$x_{parent}$])
```

a radius r_b of the x_{samp}. In this case, that is both x_0 and x_1. Starting at each x_0 and x_1, an attempt is made to steer towards x_{samp}. This results in two new edges ending at candidate nodes $x_{c,1}$ and $x_{c,2}$. Both candidate nodes are within a radius r_e of x_{samp}, so their costs will be evaluated. Notice that candidate node $x_{c,1}$ has a lower cost than $x_{c,2}$, because the path length from x_0 to $x_{c,1}$ is shorter than from x_0 to $x_{c,2}$. Therefore, $x_{c,1}$ will be added to the tree, along with an edge from x_0 to $x_{c,1}$.

After this iteration of the tree growth loop, the trees in Figure 2-1a and 2-1b are different. The *chooseParent* function of Rewire-RRT has resulted in a shorter path being added to the tree that ends near x_{samp} than the RRT algorithm achieved.



(a) RRT



(b) Rewire-RRT

Figure 2-1: The *chooseParent* function comparison between RRT and Rewire-RRT

Algorithm 4 reconnectTree(x_{new},X_{near})

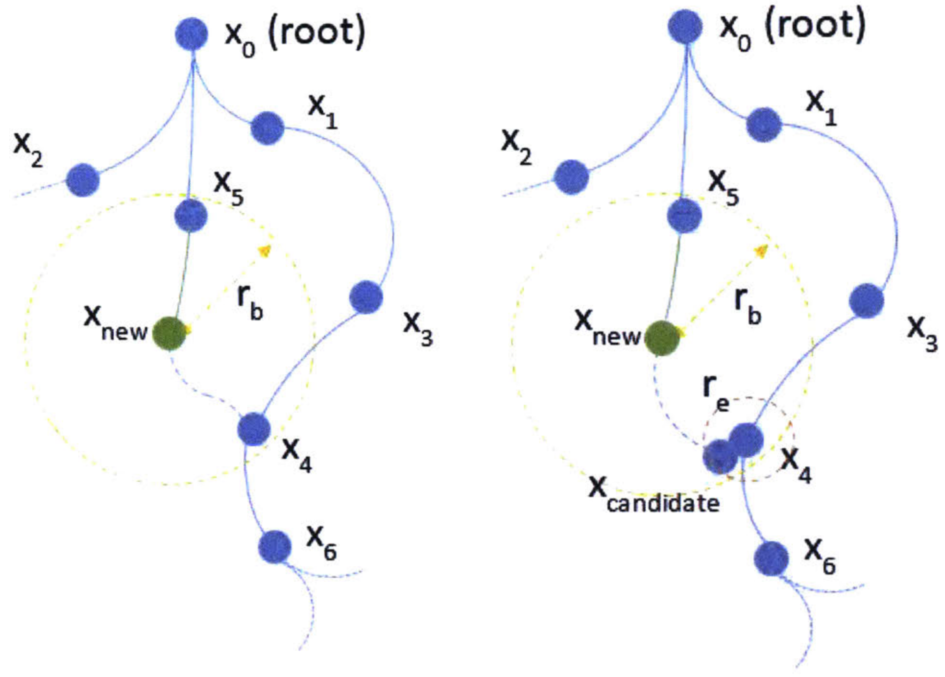
```
for each $x_{near} \in X_{near}$ do
    $x_{new\_candidate} = steer(x_{new}, x_{near})$
    if $d_{horizontal} < r_e$ then
        if cost($x_{new\_candidate}$) < cost($x_{near}$)) then
            Tree.removeNode($x_{near}$)
            Tree.addNode($x_{new\_candidate}$)
            Tree.addEdge($x_{new}$,$x_{new\_candidate}$)
        end if
    end if
end for
```

2.2.4 Rewiring the tree

After a new node is added to the tree, the *reconnectTree* function considers x_{new} as a possible parent for the nodes in the set X_{near}. An attempt is made to connect x_{new} with each $x_{near} \in X_{near}$. Using the same steering procedure as RRT, a new node $x_{new\_candidate}$ is found. If $x_{new\_candidate}$ is within a 2D radius r_e of x_{near}, and the path to it is collision free, the cost of the node $x_{new\_candidate}$ is calculated. If the cost of $x_{new\_candidate}$ is less than that of x_{near}, then $x_{new\_candidate}$ is added to the tree along with an edge from x_{new} to $x_{new\_candidate}$. Also, if x_{near} has children nodes, x_{near} is deleted and the edges between its parent and children are concatenated. This step encourages future tree growth from the lower-cost node.

Consider a tree consisting of seven nodes x_0,x_1,...,x_6, as in Figure 2-2. After the addition of a new node, x_{new}, the rewire procedure of the RRT* algorithm will try to find a path from x_{new} to each node within a radius r_b of x_{new}. In this case, that means finding a path from x_{new} to x_4. For vehicles with simple dynamics, finding an optimal feasible path that matches the position and heading at x_{new} and x_4 may be trivial. Recall, however, that an analytic solution to this two-point boundary value problem does not necessarily exist for complex dynamical systems.

The *reconnectTree* function of the Rewire-RRT algorithm relaxes one end constraint of the BVP and finds a path starting at x_{new} that ends near x_4 at the node $x_{candidate}$. If $x_{candidate}$ is within a radius r_e of x_4, then its cost is calculated. If the cost of $x_{candidate}$ is less than that of x_4, then $x_{candidate}$ is added to the tree along with



(a) RRT* (b) Rewire-RRT

Figure 2-2: The *reconnectTree* function comparison between RRT* and Rewire-RRT

the edge from x_{new}. Further, the node x_4 is deleted and the edge from x_3 to x_6 is concatenated.

2.3 Experiments

To evaluate the performance or Rewire-RRT versus RRT and RRT*, all three algorithms were used to plan paths through simulated environments.

2.3.1 Experiment Setup

Environment

Paths were planned through the environment shown in Figure 2-3. Obstacles are shown in black, and free areas are shades of green and yellow. The color bar on the side of the figure indicates the distance from the nearest obstacle. The vehicle starts at the light green circle at the bottom of the figure and the goal region is the red

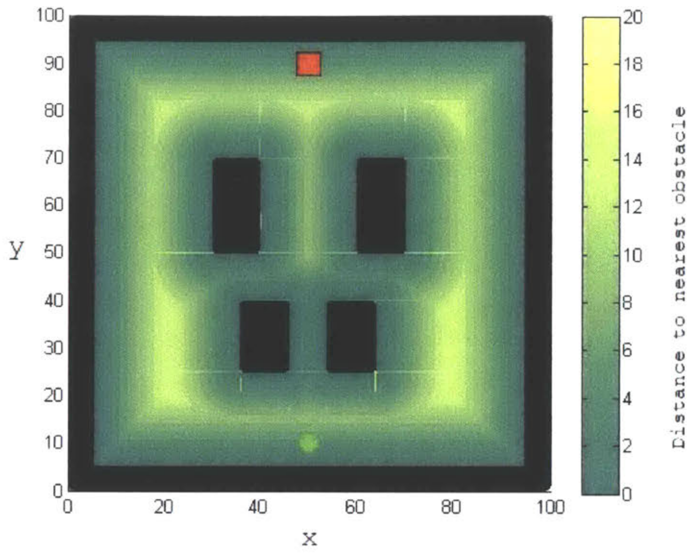


Figure 2-3: 2D path planning environment. The green dot at the bottom of the figure is the start location and the red square is the goal. Shades of green and yellow represent distance to the nearest obstacle.

square at the top of the figure. The initial velocity is in the "up" direction directly towards the red square.

Simplified Parafoil Dynamics

The simplified parafoil dynamics in two dimensions are represented by a Dubins vehicle [21] with fixed velocity, v. Therefore, the vehicle configuration space is three-dimensional, represented by a position $p = (p_x, p_y)$ and a heading ψ. The single control input u is the commanded heading rate, $\dot{\psi}$. The simplified dynamics have the form

$$\dot{p}_x = v \cos\psi, \tag{2.1}$$

$$\dot{p}_y = v \sin\psi, \tag{2.2}$$

$$\dot{\psi} = u. \tag{2.3}$$

The steering procedure in the *chooseParent* and *reconnectTree* functions must find a dynamically feasible path from one node to another. At the starting node, two-dimensional position and heading must be matched exactly. At the terminal node, position must be reached within a specified margin of error, r_e. In fact, with the given dynamics, an analytic solution using one constant rate turn is available that can match the terminal position exactly [12]. Consider nodes x_n and x_s in Figure 2-4. In order to steer from x_n, with specified position and heading, to x_s, with specified position only, first define

$$\delta_{p_x} = p_{x_s} - p_{x_n}, \tag{2.4}$$

$$\delta_{p_y} = p_{y_s} - p_{y_n}, \tag{2.5}$$

$$\delta = \sqrt{\delta_{p_x}^2 + \delta_{p_y}^2}. \tag{2.6}$$

Then, the radius of the arc can be determined via

$$R = \frac{\delta^2}{2(\delta_{p_y}\cos\psi_n - \delta_{p_x}\sin\psi_n)}. \tag{2.7}$$

The control input is given by $u = v/R$. Also of interest are the arc subtend angle γ and the time required to traverse the arc, t.

$$\gamma = 2\sin^{-1}\frac{\delta}{|2R|}, \tag{2.8}$$

$$t = \gamma/u, \tag{2.9}$$

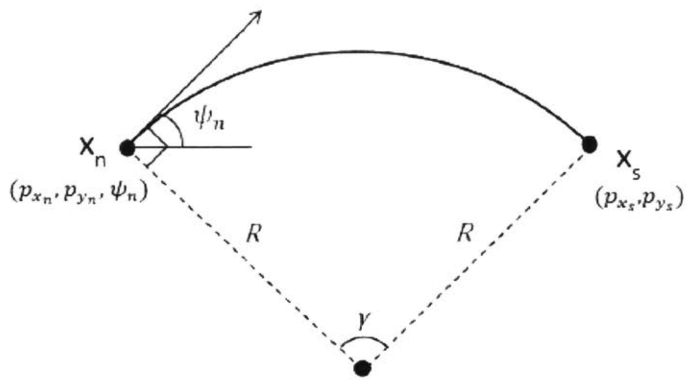


Figure 2-4: 2D steering method, adapted from [26]

Implementation

RRT, RRT*, and Rewire-RRT were implemented in MATLAB. Path planning experiments were run on a desktop computer with a 3.2-GHz Intel ® i5 processor.

2.3.2 Minimum Length Paths

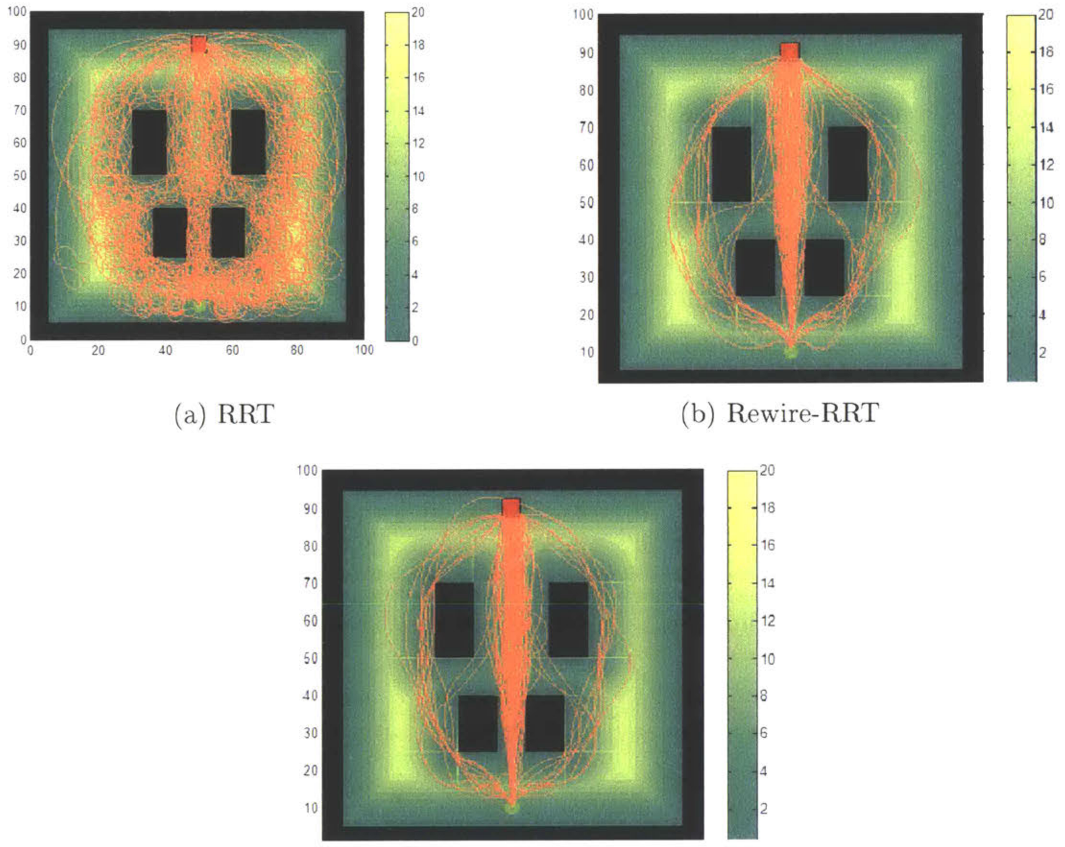
The first set of experiments will compare the ability of RRT, Rewire-RRT and RRT* to find minimum length paths through the environment in Figure 2-3. This initial scenario will reveal the ability of each algorithm not only to find dynamically feasible paths through the environment, but to also minimize an objective cost. To compare the ability of each algorithm to find paths quickly given limited computational resources, some experiments give each algorithm a fixed time limit for planning. Another set of experiments compare the effectiveness of each loop in the tree growth procedure by giving each algorithm a fixed number of loop iterations to plan a path.

Time Limit

In the first experiment, each algorithm is allowed 10 seconds of tree growth ($t_{growth\_limit} = 10s$). After this time has elapsed, the shortest path to the goal in the tree is recorded. This process is repeated 100 times for each algorithm. The

Table 2.1: Average path length for 100 trials - 10s tree growth

Algorithm:	**RRT**	**Rewire-RRT**	**RRT***
Average Path Length:	161.4	83.60	83.30



(a) RRT

(b) Rewire-RRT

(c) RRT*

Figure 2-5: Best paths for 100 trials in 2D environment - 10s tree growth - Minimum path length objective

average path length for the 100 trials is reported in Table 2.1 and an image of all 100 paths is shown in Figure 2-5.

Without the ability to specifically minimize path length during tree growth, RRT generates paths that are nearly twice as long on average than those generated by Rewire-RRT and RRT*. Even though Rewire-RRT is not guaranteed asymptotically optimal, as is RRT*, paths found in the same amount of time are under one percent longer on average.

Table 2.2: Average path length and average time per trial for 100 trials - 1000 iteration limit

Algorithm:	**RRT**	**Rewire-RRT**	**RRT***
Average Path Length:	190.3	84.36	83.30
Average Time per Trial (s):	0.9039	4.315	4.000

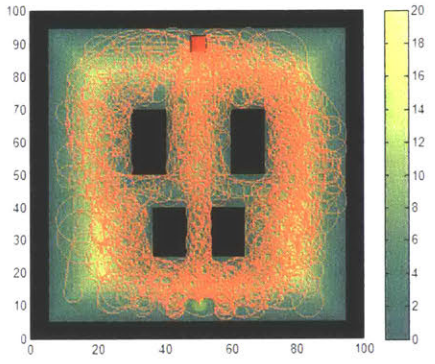
Iteration Limit

The previous experiment was repeated, but this time tree growth was not limited by a specific time limit. Instead, tree growth continued until the tree growth loop had completed 1000 iterations (i.e., each algorithm was run until 1000 points were sampled in the environment). Once the 1000 iterations were complete, the shortest path to the goal was recorded. Again, 100 trials of each algorithm were completed. The average path length for the 100 trials is reported in Table 2.2 and an image of all 100 paths for each algorithm is shown in Figure 2-6.

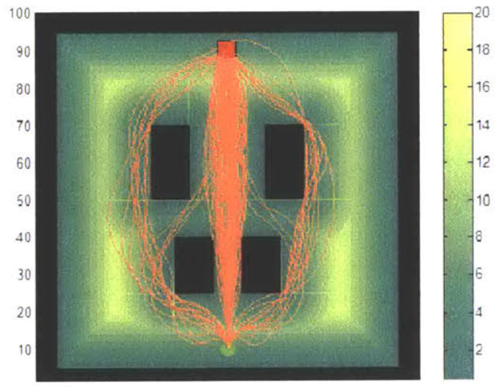
The average path length increased for all three algorithms, because the computation for 1000 iterations took less than 10s - the limit set in the previous section. Understandably, RRT was much faster that both Rewire-RRT and RRT*, because it only considers one possible parent node and does not include a rewiring function. However, we know from the first experiment (Table 2.1) that when RRT is given equal computation time as the two competing algorithms it does not perform as well. Interestingly, 1000 iterations of Rewire-RRT takes slightly more time that 1000 iterations of RRT*. An explanation for this is that the steering procedure used by Rewire-RRT more often resulted in paths that were free from obstacles. This means that each clear path had to be lazily checked against 8 obstacles (four in the middle and four walls).

Iteration Limit - single obstacle case

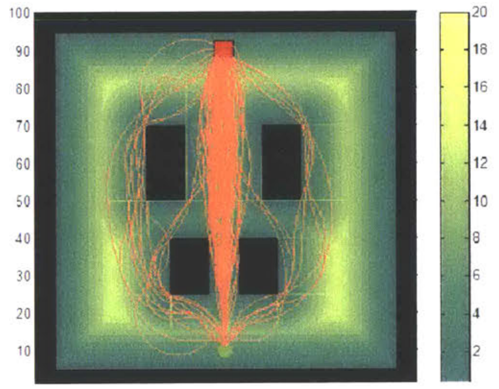
The 1000-iteration test is now repeated for the RRT* and Rewire-RRT algorithms in an environment with one single obstacle to see the affect on computation time.



(a) RRT



(b) Rewire-RRT

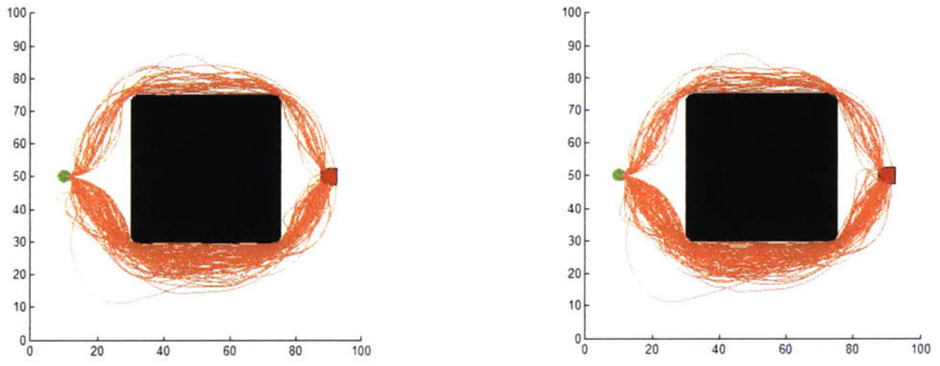


(c) RRT*

Figure 2-6: Best paths for 100 trials in 2D environment - 1000 iterations - Minimum path length objective

Table 2.3: Average path length and average time per trial for 100 trials - 1000 iteration limit - Single obstacle case

Algorithm:	**Rewire-RRT**	**RRT***
Average Path Length:	107.6	107.5
Average Time per Trial (s):	3.437	3.6047



(a) Rewire-RRT (b) RRT*

Figure 2-7: Best paths for 100 trials in single obstacle 2D environment - 1000 iterations - Minimum path length objective

In this case, Rewire-RRT is faster due to only one obstacle check being needed. RRT* still holds a slight advantage in path length, though this is minimal.

2.3.3 Safest Paths

Safety cost

In the previous experiments, the objective of the path planner was to find paths of minimum length from the initial state to the goal. In Chapter 3, when the parafoil guidance problem is formally introduced, path length is no longer a goal of the planner. One objective is to remain as far as possible from terrain features and other objects in order to avoid collisions due to unexpected winds. In two dimensions, the goal of staying far away from obstacles can be achieved by replacing the path-length cost with a safety-cost. Given a set of n states x_i, $i \in \{0, \ldots, n\}$ spaced at uniform intervals along a trajectory from the root node to a node with state x_n, d_i is defined as the distance from x_i to the nearest obstacle. The safety cost J_s penalizes every state that is within a specified distance d_{max} of any obstacle,

$$J_s = \sum_{i=1}^{n} (d_{max} - \min(d_{max}, d_i))^2 / d_{max}^2. \tag{2.10}$$

Because the cost increases exponentially with proximity to obstacles, paths that pass very close to an obstacle, even briefly, are strongly discouraged. Further, paths that remain moderately close to obstacles for extended periods are discouraged due to the additive nature of the cost formulation.

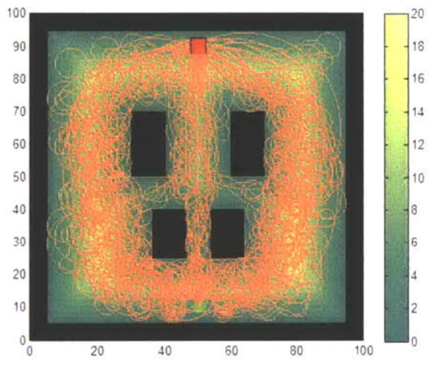
Time Limit

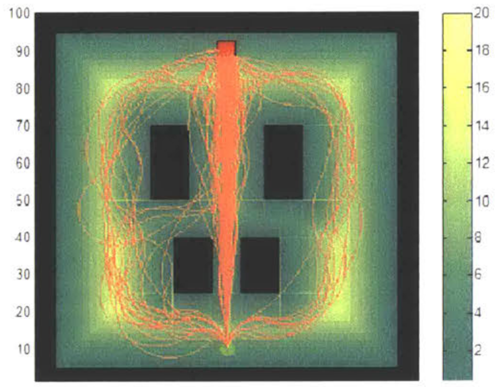
In this experiment, each algorithm is allowed 10 seconds of tree growth ($t_{growth\_limit} = 10s$). After this time has elapsed, the lowest-cost path to the goal in the tree is recorded. This process is repeated 100 times for each algorithm. The average path length for the 100 trials is reported in Table 2.4 and an image of all 100 paths is shown in Figure 2-8.

Table 2.4: Average safety-cost for 100 trials - 10s tree growth

Algorithm:	**RRT**	**Rewire-RRT**	**RRT***
Average Safety Cost:	763.0	349.4	351.0

Once again, RRT is unable to optimize the cost of nodes in the tree, and therefore has a safety cost that is on average more than twice as large as either Rewire-RRT or RRT*. RRT* yields an average safety cost that is only slightly lower than that of Rewire-RRT. In general, paths found by Rewire-RRT and RRT* stay towards the middle of obstacles, much more so than paths planned using RRT.


(a) RRT


(b) Rewire-RRT

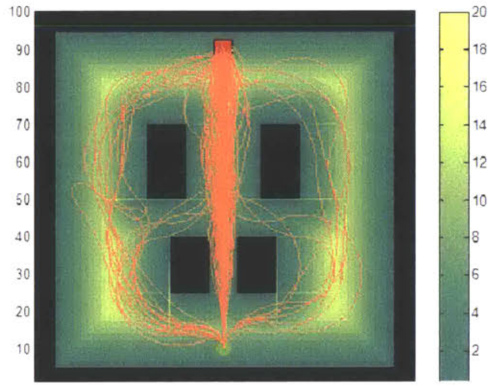

(c) RRT*

Figure 2-8: Best paths for 100 trials in 2D environment - 10s tree growth - Safety-cost objective

Iteration Limit

The previous experiment was repeated, but this time tree growth was limited by a specific time limit. Tree growth continued until the tree growth loop completed 1000 iterations (i.e., each algorithm was run until 1000 points were sampled). Once the 1000 iterations were complete, the safest path to the goal was recorded. Again, 100 trials of each algorithm were completed. The average safety-cost for the 100 trials is reported in Table 2.5 and an image of all 100 paths is shown in Figure 2-9.

Table 2.5: Average safety-cost for 100 trials - 10s tree growth - 1000 iteration limit

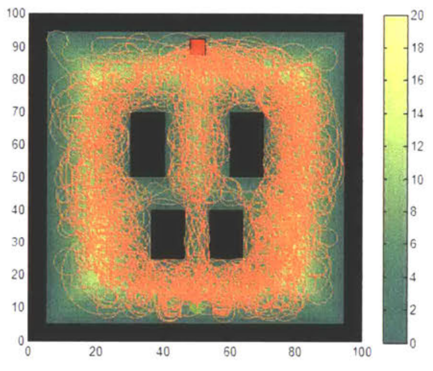
Algorithm:	**RRT**	**Rewire-RRT**	**RRT***
Average Safety Cost:	842.8	347.3	346.6
Average Time per Trial (s):	1.54	20.0	19.7

Hard Constraint on Distance from Obstacles

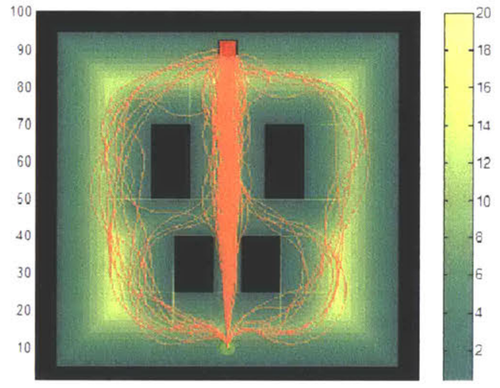
One way to improve the safety of paths found by RRT is to impose a hard constraint on the minimum distance between the path and any obstacle. This minimum distance constraint, λ, is imposed on the RRT path planner at different values in the following experiment. 100 trials are run for $\lambda = 2$ and $\lambda = 5$, with 10s of tree growth allowed per trial. Results are shown in Table 2.6 and Figure 2-10.

Table 2.6: 100 Trials - 10s - Min Distance

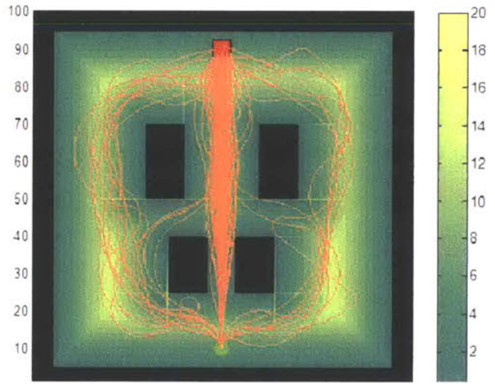
Hard Radius λ:	**2**	**5**
Average Safety Cost:	915.3	752.7



(a) RRT

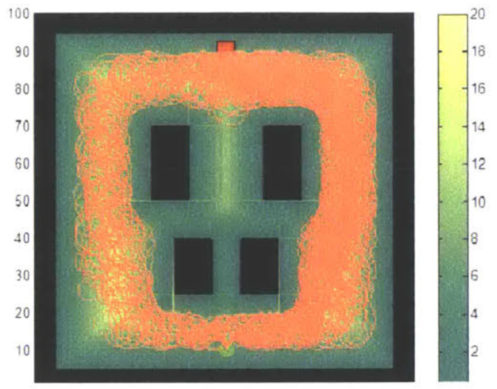


(b) Rewire-RRT



(c) RRT*

Figure 2-9: Best paths for 100 trials in 2D environment - 1000 iteration limit - Safety-cost objective


(a) RRT

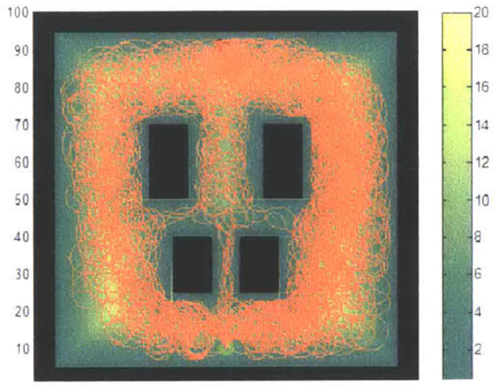

(b) Rewire-RRT

Figure 2-10: 100 trials 2D environment

When the hard constraint is imposed, the average safety-cost over 100 trials decreases only slightly for the $\lambda = 5$ case versus when no hard constrain was imposed. The constraint makes tree growth much slower because many potential paths become infeasible. Therfore, after 10 seconds of tree growth, few complete paths to the goal exist to choose from, and they may not necessarily be low-cost despite the hard constraint. It is much more efficient to use the Rewire-RRT algorithm to find low-cost paths.

2.4 Conclusion

The Rewire-RRT algorithm has been shown to find paths that are shorter and safer than those found by RRT. Further, Rewire-RRT is nearly as fast and successful in 2D at finding low-cost paths as the RRT* algorithm. In the next chapter, the parafoil guidance problem will be presented and is demonstrated that Rewire-RRT is compatible with such underactuated systems, unlike RRT*. This combination of being near-optimal and easy to apply to complex or underactuated dynamic systems will allow for strong performance by the Rewire-RRT algorithm on the parafoil guidance problem.

Chapter 3

The Rewire-RRT Algorithm for Parafoil Guidance

3.1 Introduction

The Rewire-RRT algorithm has been demonstrated to find near-optimal paths through 2D environments for a vehicle with simple dynamics. In this chapter, Rewire-RRT is used to plan paths for a guided parafoil. Experiments show that Rewire-RRT can identify dynamically feasible paths for guided parafoils through complex environments. Further, these paths are better at achieving the objectives of landing near the IP and staying clear of obstacles than paths planned by RRT.

3.2 Parafoil Dynamics and Problem Statement

The goal of this path planning problem is to guide a parafoil from some initial position p_I and heading Ψ_I (full state x_I) to some target location p_G (full state x_G), where $p = (p_x, p_y, p_z)$ is the position in the inertial reference frame. A north east down (NED) coordinate system centered at the target location is used. The parafoil dynamics are represented as the nonlinear state-space system

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}, \mathbf{w}), \quad \mathbf{x}(t_I) = \mathbf{x}_I, \tag{3.1}$$

where t_I is the initial time, $\mathbf{u}$ are the control inputs, and $\mathbf{w} = (w_x, w_y, w_z)$ are the wind disturbances. This work assumes a prediction of the winds at future flight times is available [14], based on either a priori wind forecasts or estimates of the wind experienced earlier in flight. The parafoil is modeled as a Dubins vehicle [21] descending at a rate governed by atmospheric conditions subject to updrafts/downdrafts, with the input-to-heading-rate mapping governed by complex lag dynamics. The vertical velocity is a function of the vehicle altitude via [1]

$$v(p_z) = v_0 e^{p^z/2\tau_z} \tag{3.2}$$

where $\tau_z = 10^4 m$, and v_0 is the nominal vehicle airspeed at sea level. The heading rate of the parafoil is modeled as a second-order approximation of the canopy Dutch roll lateral mode suggested by Carter et al. [11]. A first-order lag is also used to model the differential toggle control input mechanism for trailing edge deflection, while the controller is a PID with feedforward gains tuned to achieve the desired performance [11]. In total, this yields a 5th order state s and dynamics (A, B, C, D), augmented to the state vector x and dynamics (3.1), respectively. The control input is a scalar, $\mathbf{u} \equiv u \equiv \Psi_d$ representing the desired heading rate, subject to the symmetric input bounds $\mathbf{u} = \{||u|| \leq \omega_{max}\}$. The overall parafoil dynamics (3.1) thus take the form

$$\dot{p}_x = v(p_z)\cos\Psi + w_x, \tag{3.3}$$

$$\dot{p}_y = v(p_z)\sin\Psi + w_y, \tag{3.4}$$

$$\dot{p}_z = \frac{-v(p_z)}{L_D} + w_z, \tag{3.5}$$

$$\dot{s} = A\mathrm{s} + Bu, \tag{3.6}$$

$$\dot{\Psi} = sat(C\mathrm{s} + Du, -\omega_{max}, \omega_{max}), \tag{3.7}$$

where the saturation function $sat(a, b, c)$ bounds a between b and c.

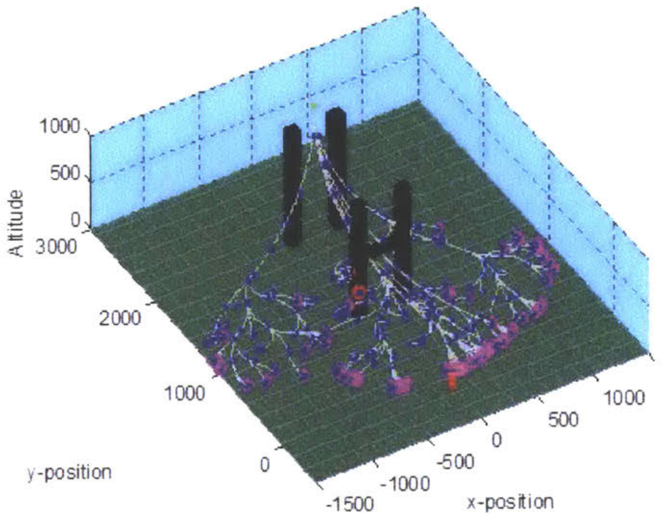


Figure 3-1: Example tree grown by the parafoil RRT algorithm. The landing points of paths are shown by pink dots. Infeasible paths that intersect obstacles are terminated by a red dot.

3.3 The Parafoil Rewire-RRT Algorithm

3.3.1 Parafoil RRT

The parafoil RRT algorithm [12] incrementally constructs a tree of dynamically feasible trajectories from the current state. An example tree is shown in Figure 3-5. An RRT-based approach is particularly well-suited to this application. RRT can quickly identify feasible solutions online within the 9-dimensional configuration space (3 for position, 1 for heading, 5 for lag dynamics), without discretizing the solution space. Further, its incremental construction and constraint checking allows it to scale with both problem complexity and available computational resources.

The RRT algorithm grows a tree by sampling a random point in the environment, x_{samp}. The nearest node to x_{samp}, x_{near}, is identified using heuristics [12]. Then, an attempt is made to connect x_{near} to x_{samp} with a dynamically feasible path. This "steering" procedure is done by generating a reference trajectory [12] between x_{near} and x_{samp} in 2D space assuming lag free dynamics. The reference trajectory consists of a single constant-rate turn that can be found using Equations (2.4)-(2.9). The

Algorithm 5 Parafoil RRT [12]

```
1: Tree.init($x_0$)
2: while $t < t_{growth\_limit}$ do
3:     $x_{samp}$ = sample()
4:     $x_{near}$ = near($x_{samp}$)
5:     $x_{new}$ = steer($x_{near}$, $x_{samp}$)
6:     if exists($x_{new}$) then
7:         Tree.addNode($x_{new}$)
8:         Tree.addEdge($x_{near}$,$x_{new}$)
9:     end if
10: end while
```

control u identified from the reference trajectory is used to propagate the full 3D lag dynamics from x_{near} to a new state x_{new}, close to x_{samp}. The likely difference in the altitude p_z between x_{samp} and x_{new} is ignored. If the path from x_{near} to x_{new} is free from obstacles, then x_{new} is added to the tree along with an edge from x_{near} to x_{new}. This tree growth method is repeated until a time limit t_{growth_limit} is met. This time limit is dictated by the replanning frequency required by the system, typically 1 Hz [14].

3.3.2 Parafoil Rewire-RRT

The Rewire-RRT algorithm assigns a cost to each node in the tree. Two functions are added to the basic RRT framework in order to build and rewire the tree to explicitly minimize the cost at each node. *ChooseParent* considers multiple candidate parent nodes and chooses the lowest-cost option. *Rewire* rewires the tree by reconnecting existing nodes with lower cost paths thorugh x_{new}. The pseudocode for parafoil Rewire-RRT tree growth is shown in Algorithm 6.

3.3.3 Cost function

The parafoil Rewire-RRT algorithm assigns a cost to each node. The cost function has two components. The first component, cost-to-go or J_{c2g}, is the estimated final miss distance of a system that has initial conditions associated with the node. This component encourages the construction of paths that will terminate close to the

Algorithm 6 Parafoil Rewire-RRT

```
1: while $t < t_{growth\_limit}$ do
2:     $x_{samp}$ = sample()
3:     $X_{near}$ = nearest($x_{samp}$)
4:     [$x_{parent}$,$x_{new}$] = chooseParent($x_{samp}$,$X_{near}$)
5:     if exists($x_{new}$) then
6:         Tree.addNode($x_{new}$)
7:         Tree.addEdge($x_{parent}$, $x_{new}$)
8:         rewire($x_{new}$,$X_{near}$)
9:     end if
10: end while
```

target, resulting in small miss distances. The second cost component, J_s, represents the safety of getting to the node from the current parafoil state (the root node). This component encourages construction of paths that do not pass close to terrain or obstacles, thus preventing early impacts with obstacles should the parafoil deviate from the planned trajectory due to unexpected winds. It is important for the parafoil to maintain as large horizontal and vertical clearances from obstacles as possible, because inaccurate wind predictions are common and can cause the parafoil to deviate from the planned path in any direction. The total cost, J, is a weighted sum of the two components,

$$J = w_1 J_{c2g} + w_2 J_s. \tag{3.8}$$

The parafoil Rewire-RRT algorithm builds and rewires the tree to minimize J at each node. Therefore the tree develops partial and complete paths that land near the goal with little risk of collision with obstacles along the way.

Cost-to-go

The cost-to-go component is based on an approximate reachability set for a parafoil with a given state [12]. Given a parafoil with position $x_0 = (p_{x0}, p_{y0}, p_{z0})$, several possible future states $x_1, x_2, \ldots, x_n$ are determined by propagating the lag-free parafoil dynamics forward a fixed time τ, each with a different control input u, spanning the range between -ω_{max} and ω_{max}. A cost J_i is assigned to each state x_i, $i \in \{0, \ldots, n\}$, based on the distance from x_i to the goal state, accounting for drift due to predicted mean wind over the estimated remaining flight time, t_{zG},

$$J_i = \sqrt{(p_{xi} - p_{zG} - t_{zG}\bar{w}_x)^2 + (p_{yi} - p_{zG} - t_{zG}\bar{w}_x)^2} + \sqrt{(p_{zi} - p_{zG})^2}. \tag{3.9}$$

The final cost-to-go function takes the maximum between the cost of the initial point, J_0, and the minimum cost in the approximate reachability set,

$$J_{c2g} = max(J_0, min(J_1, J_2, \ldots, J_{NP})). \tag{3.10}$$

Safety cost

The safety cost component is designed to discourage paths that come near mapped terrain or other obstacles, such as buildings. It is expected that the parafoil will drift from the intended trajectory due to changing winds, so paths that stay farther from obstacles at all times reduce the chance of collisions. Given a set of n parafoil states x_i, $i \in \{0, \ldots, n\}$ spaced at uniform intervals along a trajectory from the root node to a node with state x_n, d_i is defined as the distance from x_i to the nearest terrain feature or obstacle. The safety cost penalizes every state that is within a specified distance d_{max} of terrain or obstacles.

$$J_s = \sum_{i=1}^{n} (d_{max} - \min(d_{max}, d_i))^2 / d_{max}^2 \tag{3.11}$$

3.3.4 Choosing the parent node

Instead of identifying the nearest node, x_{near}, Rewire-RRT identifies a set of nearest nodes, X_{near}. For each $x_{near} \in X_{near}$, an attempt is made to connect x_{near} to x_{samp}. Using the same steering procedure as parafoil RRT, a new node $x_{new\_candidate}$ is found. If $x_{new\_candidate}$ is within a 2D radius r_e of x_{new} and the path is collision free, the cost of the node $x_{new\_candidate}$ is calculated. The candidate node with the lowest cost is chosen as x_{new}. The node x_{new} is added to the tree along with an edge from its

Algorithm 7 chooseParent(x_{samp},X_{near})

```
1: $x_{new} = null$; $x_{parent} = null$
2: $min\_cost = \infty$
3: for each $x_{near} \in X_{near}$ do
4:     $x_{new\_candidate} = \text{steer}(x_{near}, x_{samp})$
5:     if exists($x_{new\_candidate}$) && $d_{horizontal} < r_e$ then
6:         if cost($x_{new\_candidate}$)< $min\_cost$ then
7:             $x_{new} = x_{new\_candidate}$
8:             $x_{parent} = x_{near}$
9:             $min\_cost = \text{cost}(x_{new\_candidate})$
10:        end if
11:    end if
12: end for
13: return([$x_{new}$,$x_{parent}$])
```

associated parent, x_{near}. The *chooseParent* function increases the success rate of node extension, and incrementally builds a tree that minimizes the cost associated with each node in the tree.

Consider the scenario presented in Figure 3-2. Two possible parent nodes along with two candidate nodes are identified. Notice that the candidate nodes do not have be close in altitude to the sampled node, x_{rand}. The cost-to-go portion of the cost function will account for the differences in altitude between various candidate nodes, and it is useful to be able to choose between several different candidate nodes in order to identify a low-cost option.

3.3.5 Rewiring the tree

After a new node is added to the tree, the *reconnectTree* function considers x_{new} as a possible parent for the nodes in the set X_{near}. An attempt is made to connect x_{new} with each $x_{near} \in X_{near}$. Using the same steering procedure as RRT, a new node $x_{new\_candidate}$ is found. If $x_{new\_candidate}$ is within a 2D radius r_e and vertical distance h_e of x_{near}, and the path to it is collision free, the cost of the node $x_{new\_candidate}$ is calculated. Notice that $x_{new\_candidate}$ must now fall within the volume of a cylinder centered on x_{near}, as shown in Figure 3-3. If the cost of $x_{new\_candidate}$ is less than that of x_{near}, then $x_{new\_candidate}$ is added to the tree along with an edge from x_{new} to

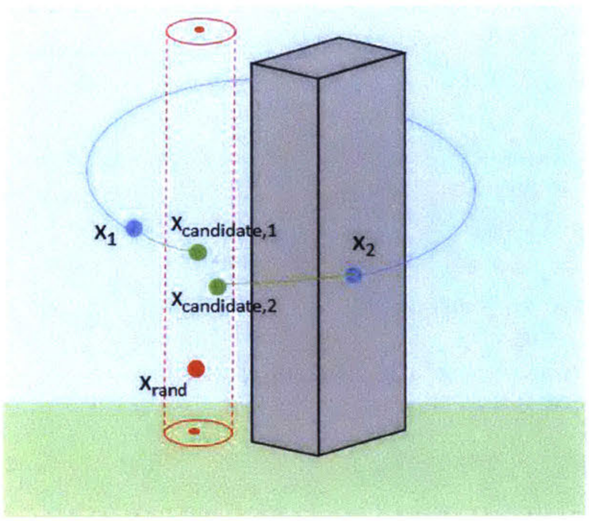


Figure 3-2: Illustration of choosing between two possible parent nodes during tree growth

$x_{new\_candidate}$. Also, if x_{near} has children nodes, x_{near} is deleted and the edges between its parent and children are concatenated. This step encourages future tree growth from the lower-cost node.

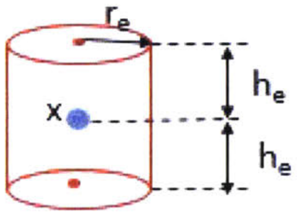


Figure 3-3: Cylinder

Algorithm 8 reconnectTree(x_{new},X_{near})

```
1: for each $x_{near} \in X_{near}$ do
2:     $x_{new\_candidate} = steer(x_{new}, x_{near})$
3:     if $d_{horizontal} < r_e$ && $d_{vertical} < h_e$ then
4:         if cost($x_{new\_candidate}$) < cost($x_{near}$)) then
5:             Tree.removeNode($x_{near}$)
6:             Tree.addNode($x_{new\_candidate}$)
7:             Tree.addEdge($x_{new}$,$x_{new\_candidate}$)
8:         end if
9:     end if
10: end for
```

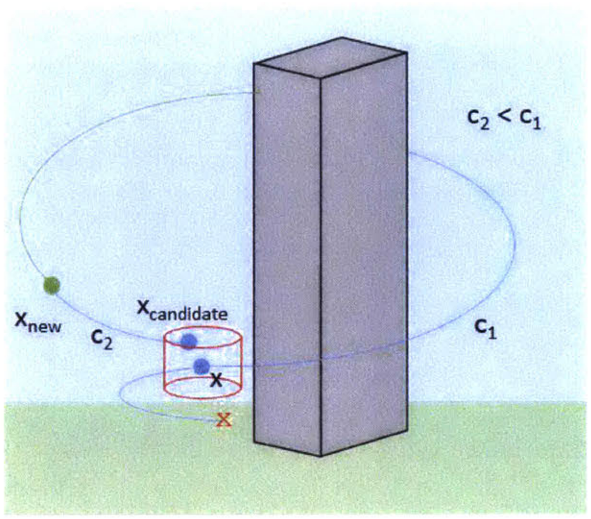


Figure 3-4: Illustration of the rewiring process during tree growth

Consider the scenario in Figure 3-4. After x_{new} is added to the tree, it is possible to create a path from x_{new} to x that stays farther away from the building. Assuming the cost-to-go of $x_{candidate}$ is equal to that of x, the total cost of node $x_{candidate}$ is therefore less than that of node x. The node $x_{candidate}$ will be added to the tree. Node x will be deleted, and the edge between its parents and children will be concatenated.

3.4 Experiments

The performance of the parafoil Rewire-RRT algorithm will be evaluated by planning paths through several environments.

3.4.1 Flat terrain performance

First, the performance of the Rewire-RRT algorithm will be evaluated in an environment with flat terrain and no obstacles. Figure 3-5 shows the trees generated by RRT and Rewire-RRT after two seconds of tree growth. The parafoil starts at the green dot with an initial velocity in the negative y-direction (towards the bottom of the page). The center of the target represents the desired landing location. Paths in the tree that impact the ground have their landing locations denoted by a magenta

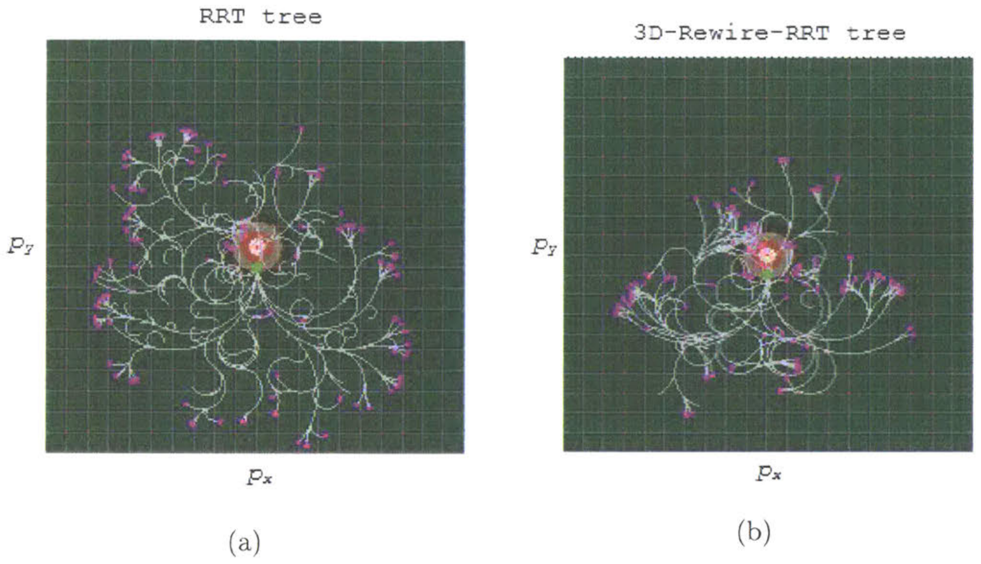


Figure 3-5: Trees of feasible paths in flat terrain

Table 3.1: Miss distance statistics normalized by mean Rewire-RRT performance on flat terrain for 50 trials of RRT and Rewire-RRT

	Mean	**Std**	**Min**	**Max**
RRT	1.78	1.45	0.109	7.02
Rewire-RRT	1	0.815	0.0518	4.03

dot. The tree grown by RRT is very spread out, demonstrating the ability of RRT to rapidly explore a large state space. However, there are very few paths that terminate near the target. The Rewire-RRT tree is grown and rewired specifically to generate paths that will terminate near the target. Paths tend to converge towards the goal, resulting in many more paths that terminate on or near the target, compared to RRT.

To compare the performance of RRT and Rewire-RRT on flat terrain, both algorithms are subjected to 50 trials. During each trial, each algorithm is allowed three seconds of tree growth. After three seconds, the complete path to the ground with the shortest miss distance for each algorithm is recorded. The results are shown in Table 3.1. The average miss distance for paths found by Rewire-RRT was 44% less than paths found by RRT. Both minimum and maximum miss distances were smaller for Rewire-RRT as well. In flat terrain, Rewire-RRT is able to outperform RRT given equal computation times.

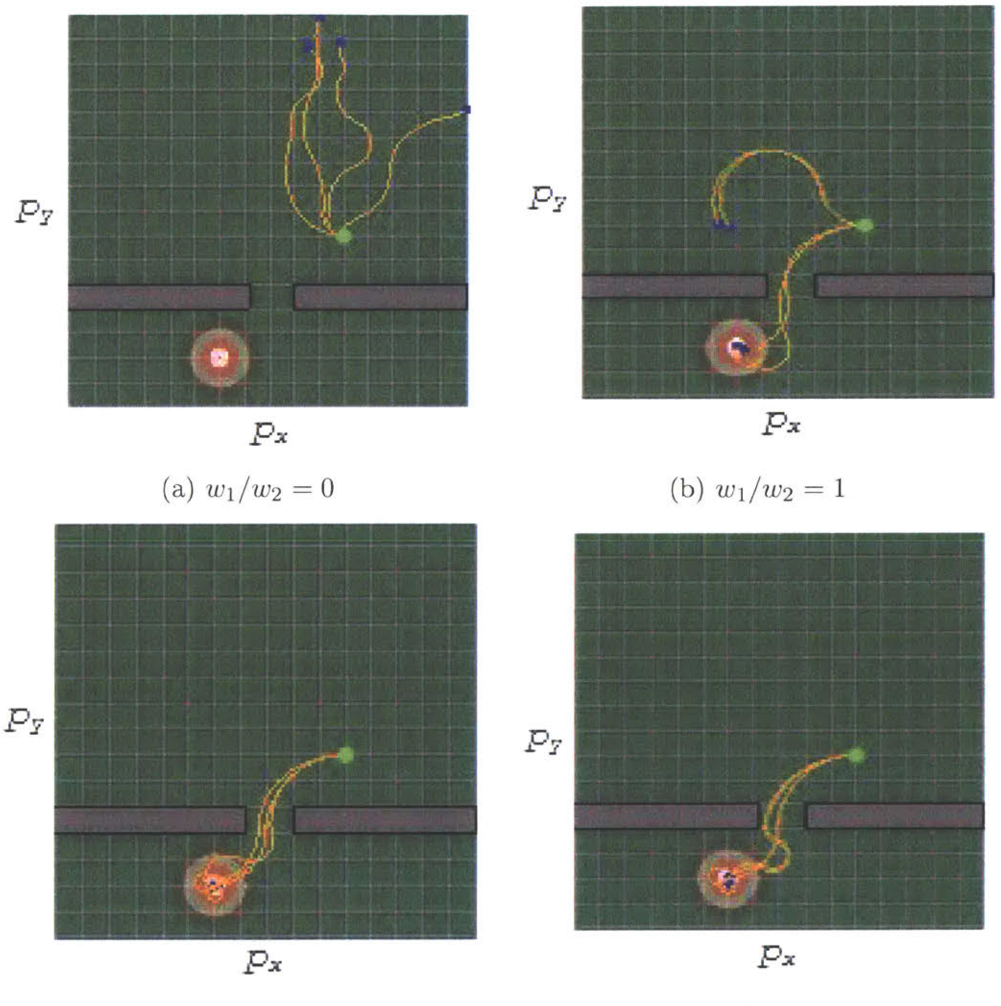
3.4.2 Weighting objective cost terms

Recall that when obstacles or terrain features are present along the flight path, Rewire-RRT builds a tree with that aims to minimize a cost function that is the weighted sum of two components. The cost-to-go component J_{c2g} helps develop paths that end near the target while the safety cost component J_s keeps the planned paths away from obstacles. Balancing these two components is necessary in order to find paths that satisfy both goals.

In Figure 3-6, several paths found by Rewire-RRT through a more complicated environment are shown. The initial state is at the green dot with initial velocity in the negative x-direction (towards the left of the page). All paths to the target must pass through a narrow opening between two obstacles. In Figure 3-6a, the ratio between w_1 and w_2 from (3.8) is zero, meaning only the safety cost is active. For four trials, we grow the tree for 10s and then choose the landed path with the lowest total cost J of the last node in the path. Note that because this node is on the ground, the cost-to-go equals the miss distance. When only the safety cost is active, the lowest cost paths stay far from any obstacles, regardless of whether they terminate near the goal. In Figure 3-6b, the ratio w_1/w_2 is one. Sometimes, paths still terminate unacceptably far from the goal in order to avoid obstacles. Figure 3-6c shows a nice balance where paths stay in the middle of the gap and terminate near the target. In Figure 3-6d, all paths terminate very close to the target, but because the cost-to-go term dominates the cost function paths sometimes pass close to obstacles, risking collision.

In order to determine the ideal weighting, 50 trials each of Rewire-RRT are run for several possible weightings. The average cost components for the best paths found during each trial are recorded. The results are shown in Figure 3-7. Confirming the findings from Figure 3-6, the best ratio is 10. Below 10, the cost-to-go, and therefore miss distances, are too high. Above 10, the risk of collision with obstacles increases too much as the safety cost increases sharply.

Given this choice of weighting, Rewire-RRT and RRT can be compared in the scenario presented in Figure 3-6. The results for 50 trials of each algorithm with



(a) $w_1/w_2 = 0$

(b) $w_1/w_2 = 1$

(c) $w_1/w_2 = 10$

(d) $w_1/w_2 = 100$

Figure 3-6: Sample trajectories for various weightings of w_1 and w_2

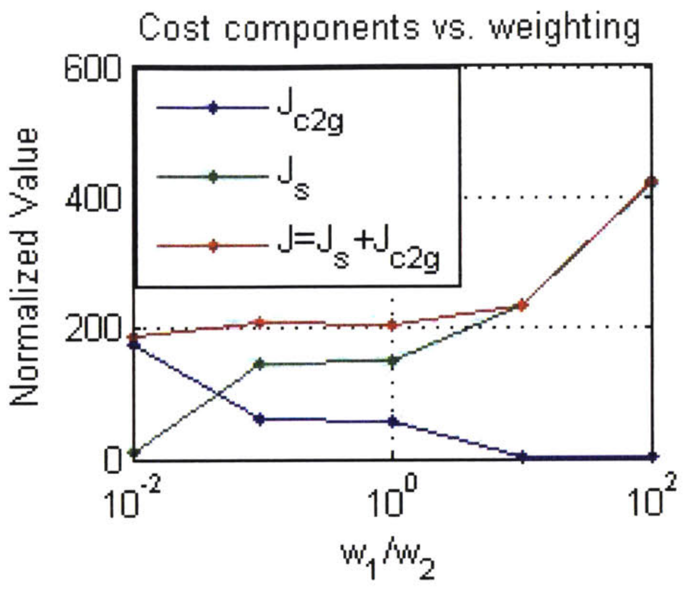


Figure 3-7: The average over 50 trials of the cost-to-go (J_{c2g}), safety(J_s), and total (J) costs at the final node of the chosen path for various weightings

Table 3.2: Miss distance statistics normalized by mean Rewire-RRT performance for 50 trials of RRT and Rewire-RRT

	Mean Miss	**Mean Safety**
RRT	1.24	45.8
Rewire-RRT	1	43.47

10s allowed for tree growth during each trial are shown in Table 3.2. Rewire-RRT provides better results in terms of mean miss distance and safety.

It is expected that this choice of weighting will work well for other scenarios with no need for tuning. This will be demonstrated by further testing in different environments without modifying the weighting.

3.4.3 Complex scenarios

A simplified urban environment is shown if Figure 3-8. It consists of 12 "buildings" and a low bridge between two of the buildings. The parafoil's initial position is at the green dot with initial velocity in the negative y-direction.

The results of 50 trials for both RRT and Rewire-RRT are shown in the

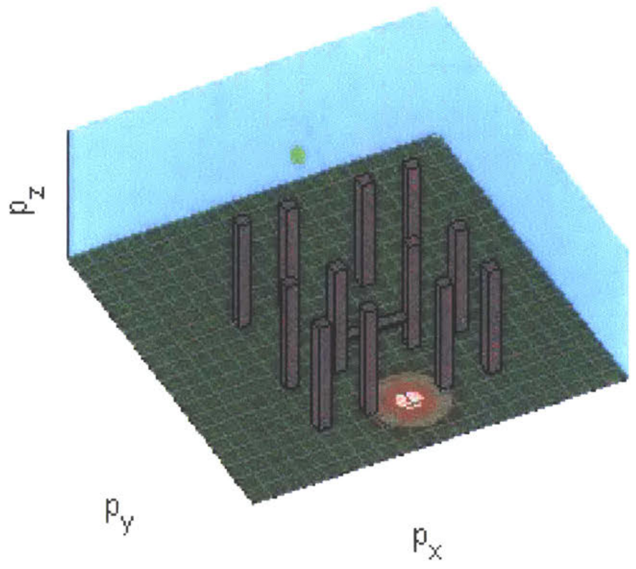
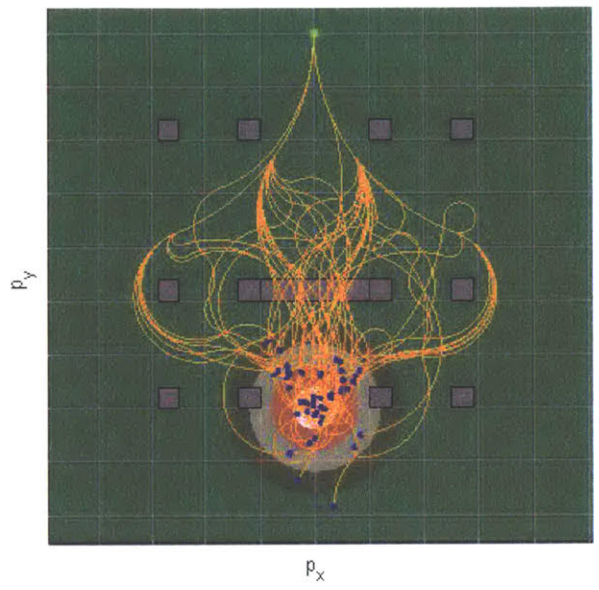


Figure 3-8: Simplified urban environment consisting of twelve "buildings" and a single bridge. The initial state is at the green dot and the goal is the center of the target.

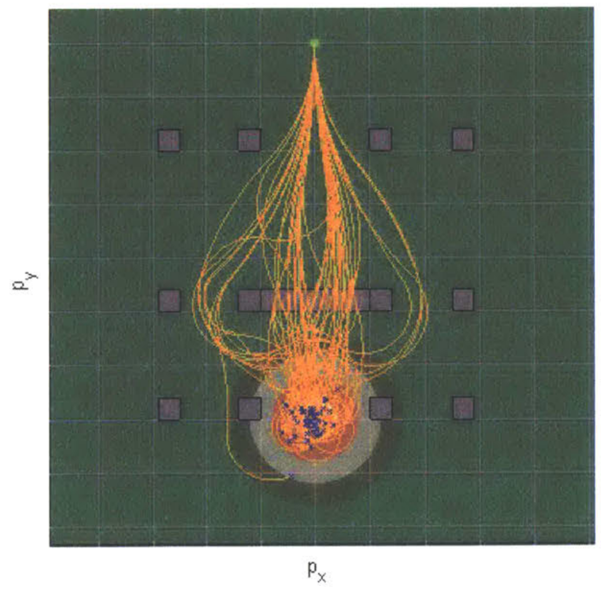
Table 3.3: Normalized Results for 50 trials of RRT and Rewire-RRT in a Simplified Urban Environment

	Mean Miss	**Mean Safety**
RRT	1.41	81.2
Rewire-RRT	1	58.6

Figure 3-9 and Table 3.3. During each trial, 10s of tree growth was allowed. Rewire-RRT provides better safety and miss distance on average by a significant margin. Figure 3-9b shows that paths generated by Rewire-RRT mostly stay in the middle of openings between obstacles and terminate near the center of the target.



(a) RRT



(b) Rewire-RRT

Figure 3-9: 50 trials in simplified urban environment

3.5 Conclusion

The Rewire-RRT algorithm is able to find dynamically feasible paths for guided parafoils even through cluttered 3D environments. These paths better achieve the objectives of landing near the IP and minimizing risk of collision than paths planned by RRT. Given this result, it is expected that Rewire-RRT will be an effective online path planner for guided airdrop.

Chapter 4

Simulations

4.1 Introduction

In previous chapters, the Rewire-RRT algorithm has been used to plan paths from a fixed point in the environment to a goal region. Parafoil guidance, however, is performed onboard a moving parafoil system. Typically, guidance, navigation, and control are performed in series at a fixed rate. Thus, replanning of the desired parafoil trajectory occurs at regular intervals during terminal guidance. This section first introduces the necessary steps to use Rewire-RRT for parafoil guidance online. Then, a series of simulations are conducted in the high-fidelity Draper simulator, as used in [26] and [27], to quantify the performance of Rewire-RRT in various environments.

4.2 Online Rewire-RRT Planning

The Parafoil Rewire-RRT algorithm (Algorithm 6) is the mechanism by which a tree of feasible paths is grown through the environment. However, tree growth is just one of many procedures that must be performed in series during each replanning interval for terminal guidance [12]. Algorithm 9 shows the high-level procedure for the entire replanning cycle. Typically, replanning occurs at a rate of 1Hz ($\Delta t = 1s$) [14].

The first step is to update the current vehicle state and current wind estimate from GPS sensor data. The current wind estimate is used to update predicted winds as a

Algorithm 9 Rewire-RRT Execution

```
1: Initialize parafoil state and time: $x = x_0, t = t_0$
2: while not landed do
3:     update vehicle state and current wind estimate
4:     update expected winds
5:     propagate current state $x_t$ by computation time $\rightarrow x_{t+\Delta t}$
6:     update tree feasibility and costs
7:     while time remaining for this step do
8:         grow tree by adding nodes (Algorithm 6)
9:     end while
10:    select lowest cost path
11:    if at least one path exists then
12:        apply best path
13:    else
14:        apply "safe" action
15:    end if
16:    $t = t + \Delta t$
17: end while
```

function of altitude. Next, the vehicle state is propagated forward by the computation time Δt using (3.3)-(3.7). This yields the expected vehicle state after computation for the current guidance iteration has finished. The current tree is repropagated from this new state $x_{t+\Delta t}$, and the feasibility and cost of each node is updated. For as much time as is remaining to complete this guidance iteration, the tree is grown using Algorithm 6. The lowest-cost path in the tree is identified, and the associated control input is used to direct the parafoil. If no feasible paths exist, a "safe" action is performed. This safe action is typically either a full left turn, full right turn, or straight ahead flight. The choice between these options is made by propagating the lag free dynamics of the parafoil ahead for a fixed amount of time using the reachability set approximation in (3.9)-(3.10). The option that remains collision free for the longest time is chosen.

4.3 Simulator

All simulation experiments are performed in the Draper high-fidelity simulation environment. This simulator has been used in several recent works to evaluate parafoil guidance strategies. Ellertson [26] used the simulator to validate the CC-BLG algorithm. Additionally, Stoeckle [27] used the simulator as a realistic training platform to design and test a Fault Detection, Isolation, and Recovery (FDIR) algorithm for autonomous parafoil guidance.

In the simulator, the parafoil is represented using a full nonlinear dynamics model [27], which incorporates the effects of the parafoil aerodynamics described in [9] and [28]. Feedback is provided for guidance in the form of simulated GPS position and ground velocity measurements. An Extended Kalman Filter (EKF) is applied in order to estimate parafoil airspeed, heading, and the true wind velocity during descent [11].

Monte Carlo experiments are conducted which vary several parameters between trials. The initial conditions of parafoil position, velocity, altitude, and heading are randomly varied during each simulation trial. The parafoil is simulated from the point of release at altitudes uniformly sampled over the range from 3,048 to 4,572 meters (10,000 to 15,000 ft), and lateral distances from 0 to 8,524 meters (0 to 28,000 ft). The terminal guidance phase begins at a preselected altitude of 500m for RRT-based guidance methods. The parafoil system parameters including payload weight, turn rate bias, and lift-to-drag ratio are also randomized over a range of values suitable for each canopy type [11]. This chapter considers simulations using the UltraFly parafoil system (JPADS-ULW) developed by Wamore Inc. [29, 30]. The system weight is uniformly sampled within the range from 250 to 750 lbs, while the turn rate bias and lift-to-drag ratio are sampled from a Gaussian distribution centered about each nominal value with standard deviations of 0.1.

4.3.1 Wind Profiles

In addition to initial conditions and parafoil system parameters, wind profiles are varied between simulation trials. A total of 25 wind profiles are used, of which 18 profiles are from collected drop data and 7 profiles are artificially generated. Of the 7 artificially generated profiles, 6 are constant-wind profiles varying in intensity from 0 to 25 knots (over 70% of the parafoil airspeed). The final artificially generated profile represents an exponentially decaying wind, with average and maximum wind speed changes with respect to altitude of 0.0025 $\frac{\text{m/s}}{\text{m}}$ and 0.05 $\frac{\text{m/s}}{\text{m}}$, respectively. The profiles from real drop data are more aggressive. Over all 18 profiles, the average wind velocity is 6.7 m/s and the maximum gust is at 17.1 m/s (nearly matching the parafoil airspeed). These profiles are subject to average and maximum wind speed changes with respect to altitude of 0.025 $\frac{\text{m/s}}{\text{m}}$ and 2.4 $\frac{\text{m/s}}{\text{m}}$, respectively. They are also subject to rapid directional changes as large as 115 $\frac{\text{deg}}{\text{m}}$.

4.4 Algorithms for comparison

The Rewire-RRT algorithm will be compared to two state-of-the art parafoil terminal guidance strategies. Chance-Constrained Band Limited Guidance (CC-BLG) [10] and Chance-Constrained RRT (CC-RRT) [12, 26] will be standards for comparison.

4.4.1 Chance-Constrained Band Limited Guidance (CC-BLG)

Band Limited Guidance (BLG)

BLG determines an optimized control input by choosing coefficients ψ_k for the heading rate profile

$$\psi'(z) = \sum_{k=0}^{N} \psi_k \frac{\sin\left(\pi(p_z - k\Delta h)/\Delta h\right)}{\pi(p_z - k\Delta h)/\Delta h} \tag{4.1}$$

based on simulating forward the simplified parafoil kinematics

$$p_x' = -L_D \cos(\psi) + w_x/\dot{p}_z, \tag{4.2}$$

$$p_y' = -L_D \sin(\psi) + w_y/\dot{p}_z, \tag{4.3}$$

$$(\cos(\psi))' = -\psi(p_z)' \sin(\psi), \tag{4.4}$$

$$(\sin(\psi))' = -\psi(p_z)' \cos(\psi), \tag{4.5}$$

where $(\cdot)'$ denotes a derivative respect to altitude p_z [14]. BLG formulates the terminal guidance as an unconstrained optimization problem designed to minimize the cost function

$$J_{BLG} = w_1(\Delta p_x^2 + \Delta p_y^2) + w_2(\sin(\Delta\psi/2))^2 \tag{4.6}$$

via the propagation of (4.1)-(4.5), where Δp_x^2 and Δp_y^2 are squared miss distances and $\Delta\psi$ is the difference between the final heading and the desired heading at the terminal trajectory state. The terms w_1 and w_2 in (4.6) denote user-specified weights selected to penalize the landing error for position and heading, respectively.

The BLG optimization is solved repeatedly online using the Nelder-Mead simplex algorithm, while the integration of the kinematics is performed using fixed-point arithmetic for computational efficiency [14]. In addition, BLG periodically compares the current optimization cost against a set of randomly generated trajectory solutions to prevent possible convergence to local minimum [8]. Lastly, through the selection of appropriate values for N and Δh in (4.1), the BLG algorithm ensures accurate trajectory tracking by considering only those heading rate profiles with frequencies sufficiently less than the control bandwidth constraints. These parameters serve to enforce the "Band-Limited" quality of the trajectory design so as to avoid excitation of payload and canopy modes [14]. Due to the difficulty of optimizing a trajectory from a large starting altitude, BLG is often run in two phases, with the first phase guiding the parafoil to a specified position in the air and the second phase guiding the parafoil to the ground.

Chance-Constrained Band Limited Guidance

A major contribution of the CC-BLG algorithm is the addition of terms to the BLG objective function that capture the probability of the parafoil prematurely colliding with terrain [10]. This is done by developing an uncertainty model for the wind. This model is used along with a method of weighted analytic uncertainty sampling to estimate the probability of collision with terrain. Next, this probability is converted into a cost term that is added to the BLG objective function for trajectory optimization.

Additionally, a discrete reachability set approximation is used for robust obstacle detection and avoidance during the first phase of BLG. This "looks ahead" to prevent collisions with terrain features immediately after the transition to phase two.

4.4.2 Chance-Constrained RRT (CC-RRT)

Chance-Constrained RRT [12, 26] builds upon the parafoil RRT algorithm. CC-RRT builds a tree of paths that have a risk of collision with terrain or obstacles below a specified threshold. This is done by using a wind uncertainty model to build an uncertainty distribution over future parafoil states. While this provides a theoretical guarantee of safety to within some threshold, paths developed by CC-RRT are not explicitly optimized for miss distance or safety.

4.4.3 Presentation of Results

Results are presented in the form of cumulative distribution functions (CDF) and tabular data of normalized parafoil miss distance performance. In each experiment, the data is normalized by the median landing accuracy from the set of Rewire-RRT trials.

4.5 Rewire-RRT Parameters

Several parameters are held constant during all simulation experiments.

- **Radius** r_b - The *chooseParent* and *rewireTree* algorithms will consider nodes within a 2D radius r_b of the sampled node x_{samp} as possible parent nodes and for rewiring, respectively. This radius is set at 300m. Note that if more than 10 nodes are within this radius, only the closest 10 are considered. If there are no nodes in this radius, the single closest node is considered.

- **Radius** r_e - When an attempt is made to steer from a possible parent node to x_{samp} or from x_{new} to x_{near} during rewiring, the terminus of the exended path must be within a 20m radius of the target node.

- **Height** h_e - During rewiring, the target cylinder has dimension $h_e = 50m$ (total height of cylinder is 100m). Due to the lack of parafoil control in the vertical direction, this height is much larger than the radius r_e.

- **Cost term weighting** - The result from Section 3.4.2, $w_1/w_2 = 10$ remains unchanged in this section.

- **Maximum iterations of tree growth** - Each time tree growth occurs (line 8 in Algorithm 9), 165 iterations of the tree growth procedure are allowed. This result was derived from an analysis by Ellertson [26] that concluded 165 was the average number of samples generated in 1 Hz planning cycle with 60% duty cycle by the nominal RRT algorithm.

4.6 Simulation Results

To understand the performance of the Rewire-RRT algorithm for parafoil guidance, the algorithm will be tested in three main types of environments ranging from simple to very complex:

1. **Flat Terrain** The simplest testing environment consists of completely flat ground. No terrain features or man-made obstacles are present.

2. **Canyon Terrain** The canyon terrain represents a complex natural environment. It is free from any man-made obstacles.

3. **Urban Environment** The urban environments contain multiple large buildings near the IP that pose a hazard to landing parafoils. The terrain is completely flat.

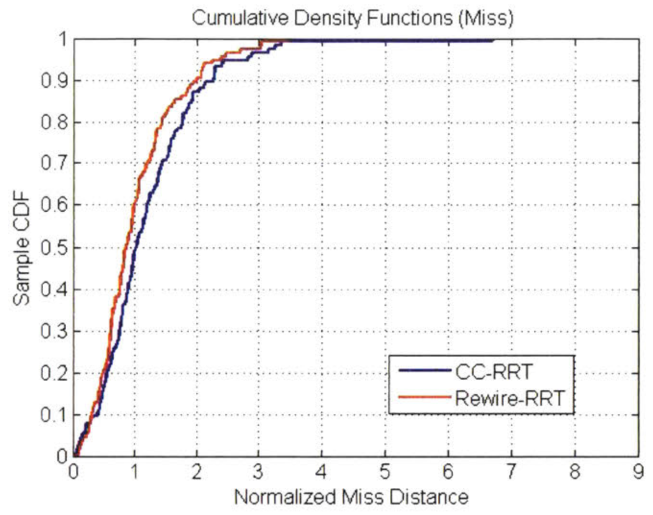


Figure 4-1: Normalized miss distance CDF for flat terrain - 116 trials

Table 4.1: Normalized miss distance for flat terrain - 116 trials

Algorithm	**Mean**	**StDev**	**50%**	**80%**	**90%**	**95%**	**98%**
CC-RRT	1.41	1.02	1.19	2.05	2.62	3.15	3.82
Rewire-RRT	1.19	0.771	1	1.67	2.32	2.73	3.54

4.6.1 Flat terrain

The flat terrain case should be the easiest case for all algorithms. The lack of any terrain features or obstacles means that there is no potential for unwanted impacts if the parafoil strays from the planned path due to unexpected winds. Therefore, there is no need for the planner to consider leaving extra space around obstacles to prevent such collisions. Only miss distance from the IP needs to be optimized.

First, the performance or Rewire-RRT is compared to that of CC-RRT. Each algorithm is run for 116 trials. The results of this experiment are shown in Figure 4-1 and Table 4.1.

In flat terrain, Rewire-RRT outperforms CC-RRT on average and at all percentiles. This indicates that the cost-to-go term in Rewire-RRT's cost function accurately determines which nodes are more likely to terminate near the IP. Further, the consideration of multiple possible parent nodes and the rewiring or the tree, both of which consider the cost-to-go, are working as intended to identify paths that land

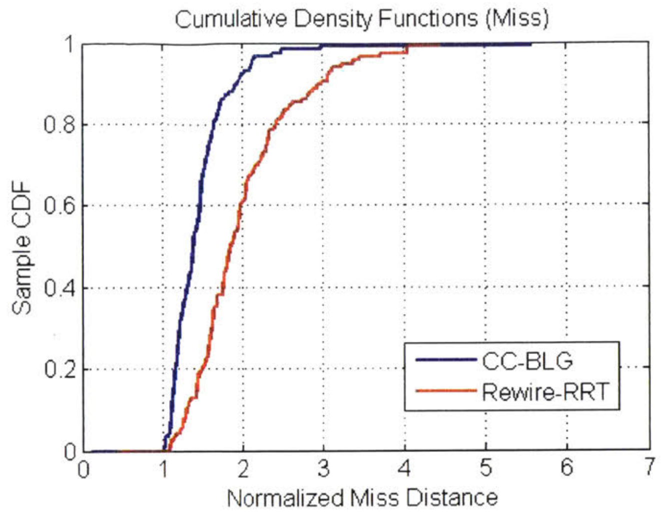


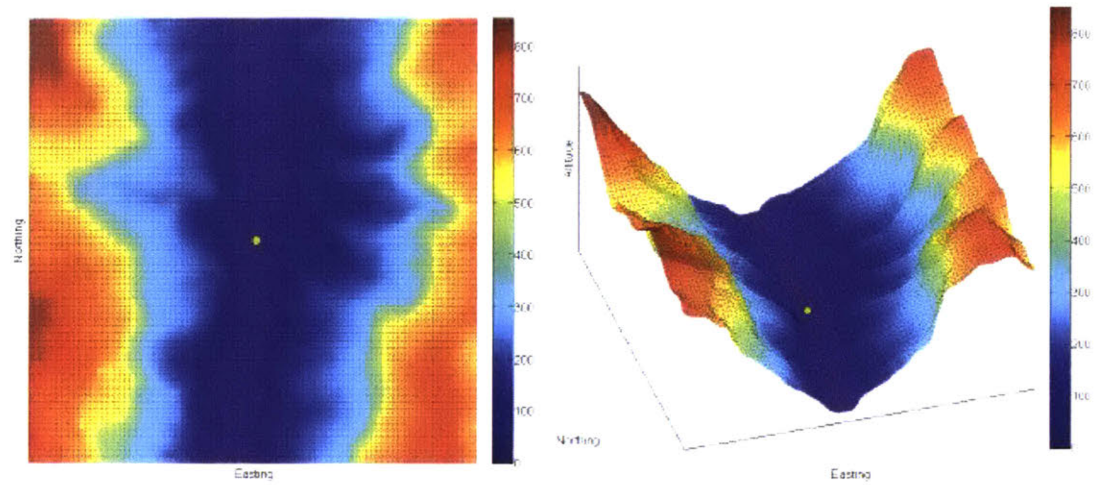
Figure 4-2: Normalized miss distance CDF for flat terrain - 116 trials

Table 4.2: Normalized miss distance for flat terrain - 116 trials

Algorithm	**Mean**	**StDev**	**50%**	**80%**	**90%**	**95%**	**98%**
CC-BLG	0.554	0.641	0.416	0.726	1.12	1.32	1.80
Rewire-RRT	1.19	0.77	1	1.67	2.32	2.73	3.54

near the target. While the improvement in mean miss distance of Rewire-RRT over CC-RRT is 15.6%, it is important to note that CC-RRT's performance on flat terrain is poor relative to the state-of-the art. Consider the comparison of performance between Rewire-RRT and CC-BLG shown in Figure 4-2 and Table 4.2.

The miss distances achieved by CC-BLG are about half of those acheived by Rewire-RRT on average and at all percentiles. When no terrain features or obstacles are present, BLG's optimization framework is better able to find paths that land near the IP than Rewire-RRT. In flat terrain where there are no obstacles, the ability of BLG to minimize miss distance is very strong. Because there are few constraints on the parafoil trajectory shape and terminal guidance can begin at a relatively low altitude, the Nelder Mead simplex search used by BLG is very fast at optimizing the trajectory for miss distance. Rewire-RRT's advantages in path flexibility and invariance to starting altitude do not help in the flat terrain setting.



(a) Top view (b) 3D view

Figure 4-3: Canyon terrain environment (meters)

4.6.2 Canyon terrain

Canyon terrain is a very challenging natural environment for a parafoil landing. Simulations were performed in a section of the Grand Canyon shown in Figure 4-3. The IP is represented by a yellow dot at the very bottom of the canyon. The steep slope of the canyon walls, along with the great depth, create a risk for the parafoil to collide with the walls before reaching the IP. A naive planner such as BLG, which considers only miss distance and terminal heading during optimization, may plan a path that comes very close to the canyon walls. Should the parafoil by pushed off the desired trajectory by unexpected winds, a collision with the wall is possible. This collision could damage cargo and greatly increase the miss distance from the IP. By considering the distance between the planned trajectory and terrain features in some way, Rewire-RRT, CC-RRT, and CC-BLG all provide some measure of protection against this type of collision. Note that in canyon terrain simulation experiments, a parafoil is always considered landed at first contact with terrain, even if this was an unplanned collision.

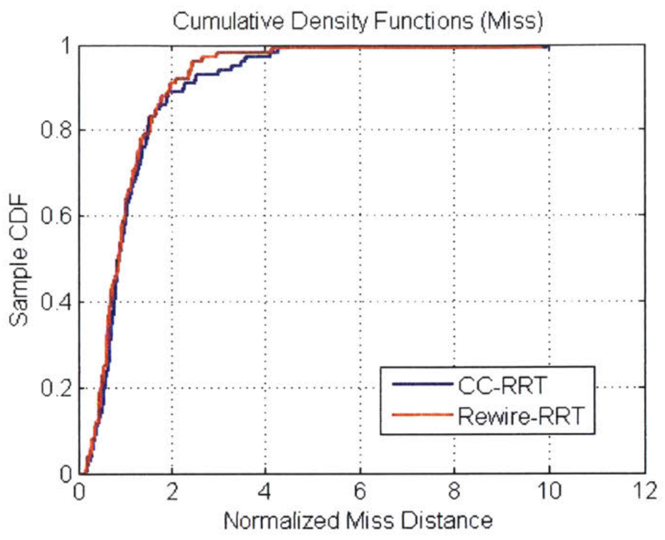


Figure 4-4: Normalized miss distance CDF for canyon terrain - 100 trials

Table 4.3: Normalized miss distance for canyon terrain - 100 trials

Algorithm	**Mean**	**StDev**	**50%**	**80%**	**90%**	**95%**	**98%**
CC-RRT	1.40	1.40	0.979	1.72	2.63	3.94	4.88
Rewire-RRT	1.28	1.29	1	1.79	2.30	2.83	4.15

First, the sampling based planners were be compared in the canyon terrain. Both Rewire-RRT and CC-RRT were run for 100 trials. The results of this experiment are shown in Figure 4-4 and Table 4.3.

Rewire-RRT has a mean miss distance that is 8.6% better than CC-RRT. At the 90th percentile and greater, Rewire-RRT also has an advantage. Overall, however, CC-RRT and Rewire-RRT are fairly evenly matched in this terrain. Though the canyon walls are steep, the width of the canyon is still several kilometers. Both Rewire-RRT and CC-RRT are successful at avoiding unexpected collisions with the terrain in all but the most vigorous wind profiles. Rewire-RRT maintains its advantage by considering cost-to-go during tree growth, as in the flat terrain.

Next, Rewire-RRT is compared with 100 trials of CC-BLG in the same canyon terrain. Recall that CC-BLG incorporates risk of collision with terrain into its objective function. The results of this experiment are displayed in Figure 4-5 and Table 4.4.

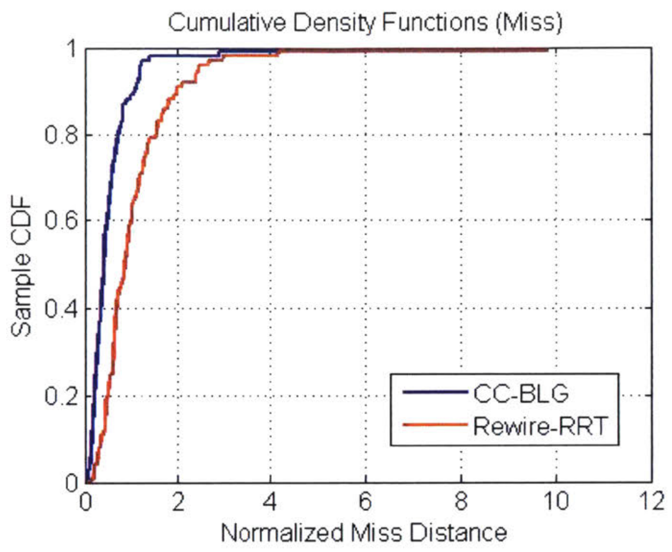
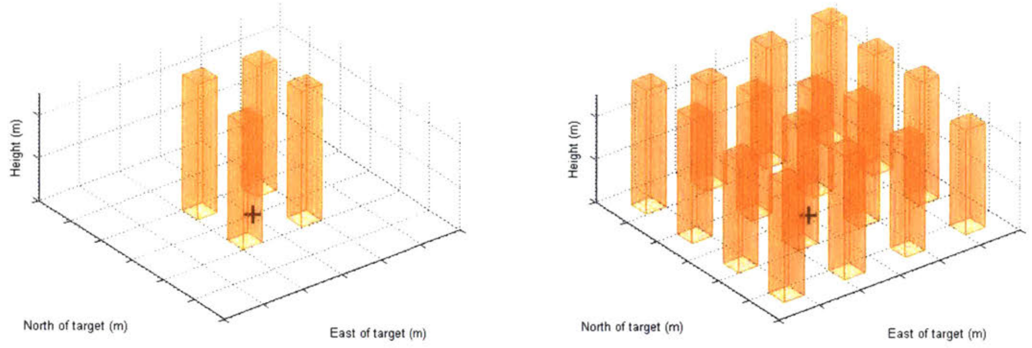


Figure 4-5: Normalized miss distance CDF for canyon terrain - 100 trials

Table 4.4: Normalized miss distance for canyon terrain - 100 trials

Algorithm	**Mean**	**StDev**	**50%**	**80%**	**90%**	**95%**	**98%**
CC-BLG	0.640	0.758	0.470	0.890	1.20	1.55	2.22
Rewire-RRT	1.28	1.29	1	1.79	2.30	2.83	4.15

In the canyon terrain, CC-BLG maintains an advantage over Rewire-RRT. Because of the width of the canyon, a simple J-hook style approach consisting of a downwind leg parallel to the canyon followed by a 180deg turn upwind is possible, as long as the parafoil stays towards the center of the canyon. No complex series of maneuvers around obstacles is necessary. Because phase two of CC-BLG can occur at relatively low altitude, even in the canyon terrain, optimization of the objective function is fast and converges towards the global minimum in most trials. CC-BLG achieves miss distances of about half of those achieved by Rewire-RRT on average and at all percentiles.



(a) City Block Environment (b) Dense Urban Environment

Figure 4-6: Two urban style environments

4.6.3 Urban environment

Finally, two urban-syle envrionments are considered. They consist of rectangular prism shaped obstacles, representing large skyscrapers, arranged on a grid around the IP. The "City Block" environment (Figure 4-6a) has four obstacles and the "Dense Urban" environment (Figure 4-6b) has 16 obstacles. These urban cases represent a unique challenge compared to the canyon terrain. Though a "straight-in" approach is possible, the large height of the buildings combined with the randomized initial conditions will require that the parafoil maneuver around multiple obstacles in at least some cases.

This represents a challenge for the optimization-based BLG and CC-BLG algorithms for several reasons. First, turns around multiple obstacles requires BLG to use more control points (larger N in (4.1)). This decreases the speed of optimization. Further, the presence of multiple large obstacles, in contrast to the canyon terrain, makes the objective function non-convex. This makes it easy for the optimization to fall into a local minimum, even with the random comparisons to random trajectories. For these reasons, only the sampling-based planners are tested in these environments. The results for 100 trials each of CC-RRT and Rewire-RRT are shown in Figure 4-7 and Table 4.5. In these urban environment simulations, parafoils that collide with obstacles are allowed to continue flight to the ground. Statistics of these constraint

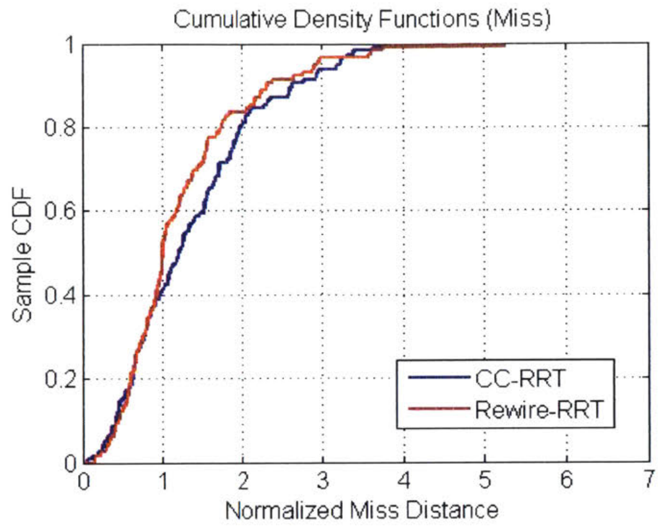


Figure 4-7: Normalized miss distance CDF for city block environment - 100 trials

Table 4.5: Normalized miss distance for city block environment - 100 trials

Algorithm	**Mean**	**StDev**	**50%**	**80%**	**90%**	**95%**	**98%**
CC-RRT	1.37	0.867	1.21	1.98	2.64	3.21	3.43
Rewire-RRT	1.25	0.85 0	1	1.73	2.33	2.93	3.64

violations are reported in the analysis of each set of simulations.

Rewire-RRT performs better than CC-RRT on average and at all percentiles other than the 98th in the City Block environment. Because CC-RRT requires large margins between the planned trajectory and obstacles, it is difficult for the planner to identify feasible paths into the middle of the buildings. In many cases, it must rely on replanning at lower altitudes, when the required margins are not as large, to identify paths that terminate near the target. This problem is exacerbated in the Dense Urban environment. Rewire-RRT, on the other hand, does not require these safety margins. Paths that come near the IP can be identified early on and then refined for safety and miss distance in subsequent replanning intervals. This gives Rewire-RRT an advantage in this environment. In terms of safety, CC-RRT and Rewire-RRT were evenly matched. Five trials for each algorithm violated the obstacle boundary constraints. Results from 100 trials in the Dense Urban environment are displayed in Figure 4-8 and Table 4.6.

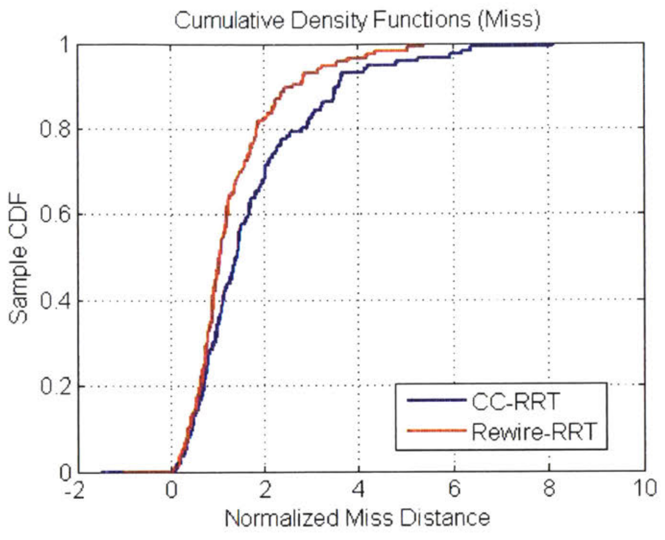


Figure 4-8: Normalized miss distance CDF for urban environment - 100 trials

Table 4.6: Normalized miss distance for urban environment - 100 trials

Algorithm	**Mean**	**StDev**	**50%**	**80%**	**90%**	**95%**	**98%**
CC-RRT	1.78	1.44	1.40	2.85	3.60	4.63	6.23
Rewire-RRT	1.34	0.971	1	1.86	2.61	3.46	4.49

In the Dense Urban environment, the advantage of Rewire-RRT is more pronounced. At the 80th percentile Rewire-RRT performs 34.7% better than CC-RRT. The over-conservative hard constraints imposed by CC-RRT, along with its lack of *chooseParent* and *reconnectTree* functions make it more difficult to find quality paths than terminate near the IP in such a cluttered environment. The increased success rate of node extension provided by the *chooseParent* function, along with the increased path quality provided by *reconnectTree* allow for Rewire-RRT to perform better in this scenario. Cluttered environments such as this one are where the Rewire-RRT algorithm has the most potential for success. Eight trials of Rewire-RRT violated obstacle constraints, compared to seven trials of CC-RRT.

4.7 Conclusion

Simulation results show that Rewire-RRT is a potential path planner for guided parafoils. In all three simulated environment types, flat, canyon, and urban, Rewire-RRT was as good or better than the CC-RRT algorithm. In the simpler flat and canyon terrains, CC-BLG kept an advantage. In relatively open spaces, an optimization-based planner such as CC-BLG has an advantage over sampling-based algorithms. However, as the environment gets more cluttered with multiple obstacles, a sampling-based approach becomes more competitive. With further testing, Rewire-RRT may prove to be the best guided airdrop guidance algorithm in certain complex terrains. This suggests that the best guidance algorithm for guided airdrop may be dependent on the environment, and worth tailoring to each mission for optimal performance.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

Guided airdrop is a useful method for delivery of cargo when ground transportation is dangerous or otherwise infeasible. It is imperative that the cargo land as accurately as possible in all conditions. However, landing accurately remains difficult, especially in complex environments with multiple obstacles near the IP. Path planning for guided airdrop, especially in the terminal phase, is an active area of research.

One method to increase the accuracy of guided airdrop systems in complex terrain is to use an advanced sampling-based planner such as RRT. The ability of RRT to rapidly explore large state spaces makes the algorithm attractive for the guidance of many robots. However, the sub-optimality of the RRT algorithm makes it less useful in cases where an objective needs to be met very closely. The RRT* algorithm provides a guarantee of optimality in certain conditions, but has been shown to be difficult to implement for underactuated systems.

In Chapter 2, a method of approximating the *chooseParent* and rewiring operations of RRT* for underactuated systems is identified. Rewire-RRT, as the algorithm is called, is used to find both shortest and safest paths through an environment for a vehicle with simple dynamics. It is shown that Rewire-RRT finds paths that are lower-cost than RRT and only slightly higher-cost relative to RRT*.

Following this work, Rewire-RRT is applied to a system with underactuated dynamics in Chapter 3. Rewire-RRT is shown to find paths for guided parafoils that better meet the system objectives of safety and miss distance than paths planned by RRT. This is true in flat terrain and in an urban-style environment.

Finally, in Chapter 4, the Rewire-RRT algorithm is used to guide parafoils online is several simulation experiments. In flat terrain, canyons, and urban environments, Rewire-RRT meets or exceed the performance of the state-of-the-art CC-RRT algorithm. In the simpler flat and canyon terrains, Rewire-RRT is outperformed by the CC-BLG algorithm. This suggests that Rewire-RRT has strong potential in very cluttered environments such as cities, but optimization based algorithms are better suited for more open environments.

5.2 Future Work

This work can be extended in several ways. Four possibilities are suggested below.

5.2.1 Further Simulations

There exists great opportunity to run more simulations with the Rewire-RRT algorithm. Further simulation in many different urban environments, including models of real cities where such map data is available, would provide insight into whether real urban airdrop flights are feasible. Running large numbers of trials will increase the accuracy of the miss distances reported at higher percentiles.

5.2.2 Obstacle Checking

Significant computation time is used to identify the nearest terrain feature or obstacle to a given parafoil state. Currently, terrain data is stored in the DTED2 file format [31]. The elevation of terrain is available at gridpoints on the map spaced at intervals determined by the map resolution. In order to estimate the distance to terrain, terrain elevation must be sampled at several points at various radii from the parafoil state.

Obstacles such as buildings are mapped using a separate system. Each obstacle is represented as a polygon of specified height. The distance from a parafoil state to the nearest obstacle is calculated by computing the distance to all obstacles and choosing the lowest value. More efficient representations of the environments and efficient constraint checking against the environment can ease this computational burden. For example, Octomaps [32] have been used successfully for 3D mapping for many robotic applications. However, given the large area covered by a guided parafoil during flight, it is not clear that this solution would scale well to fit the parafoil domain.

5.2.3 Sensed Obstacles

If parafoils are deployed into urban environments, a map of the environment may not always be available *a priori*. It may be neccessary to equip the parafoil with onboard sensors to detect the environment. As sensors such as Lidar [33] become lighter and cheaper, they may become useful for parafoil guidance. Because Rewire-RRT grows a tree of feasible trajectories, back up trajectories are available should the currently chosen path become obstructed by a newly sensed obstacle. For this reason, Rewire-RRT may be more useful should obstacles need to be sensed online than optimization-based methods which would likely need time to converge to a new feasible solution should the current solution become infeasible. Simulation experiments to study the viability of current parafoil guidance methods when sensors are used to detect obstacles online would provide valuable information.

5.2.4 Dynamic Obstacles

Additionally, obstacles in urban areas may not be static. An extension to the case of sensed obstacles, decribed above, would be to allow the obstacles to move in addition to being sensed.

Bibliography

[1] Branden J Rademacher, Ping Lu, Alan L Strahan, and Christopher J Cerimele. In-flight trajectory planning and guidance for autonomous parafoils. *Journal of Guidance, Control, and Dynamics*, 32(6):1697–1712, 2009.

[2] Yves de Lassat de Pressigny, Raphael Bechet, Richard Benney, Michael Henry, and Jan-Henrik Wintgens. Pacd 2008: Operational requirements fullfilled. In *20th AIAA Aerodynamic Decelerator Systems Technology Conference and Seminar*, page 2993, 2009.

[3] WR Barton and CF Knapp. Controlled recovery of payloads at large glide distances, using the para-foil. *Journal of Aircraft*, 5(2):112–118, 1968.

[4] The warrior. *http://www.natick.army.mil/about/pao/pubs/warrior/03/mayjune/*, May-June 2003. Accessed: 05-02-2017.

[5] Philip Hattis, Brent Appleby, Thomas Fill, and Richard Benney. Precision guided airdrop system flight test results. In *14th Aerodynamic Decelerator Systems Technology Conference*, page 1468, 1997.

[6] Natick uses efficient airdrop testing. *https://www.army.mil/article/72525/Natick_uses_efficient_airdrop_testing*, January 2012. Accessed: 05-02-2017.

[7] Sean George, David Carter, Jean-Christophe Berland, Storm Dunker, Steven Tavan, and Justin Barber. The dragonfly 4,500 kg class guided airdrop system. In *Infotech@ Aerospace*, page 7095. 2005.

[8] Keith Bergeron, Gregory Noetscher, Michael Shurtliff, and Frank Deazley. Longitudinal control for ultra light weight guided parachute systems. In *23rd AIAA Aerodynamic Decelerator Systems Technology Conference*, page 2108, 2015.

[9] Nathan Slegers and Mark Costello. Model predictive control of a parafoil and payload system. *Journal of Guidance, Control, and Dynamics*, 28(4):816–821, 2005.

[10] Aaron Ellertson, Jonathan P How, and Louis S Breger. Analytic chance constraints for the robust guidance of autonomous parafoils. In *AIAA Infotech@ Aerospace*, page 1408. 2016.

[11] David Carter, Sean George, Philip Hattis, Marc W McConley, Scott Rasmussen, Leena Singh, and Steve Tavan. Autonomous large parafoil guidance, navigation, and control system design status. In *19th AIAA Aerodynamic Decelerator Systems Technology Conference and Seminar*, page 2514, 2007.

[12] Brandon D Luders, Ian Sugel, and Jonathan P How. Robust trajectory planning for autonomous parafoils under wind uncertainty. In *AIAA Infotech@ Aerospace (I@ A) Conference*, page 4584, 2013.

[13] Anthony Calise and Daniel Preston. Swarming/flocking and collision avoidance for mass airdrop of autonomous guided parafoils. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*, page 6477, 2005.

[14] David Carter, Leena Singh, Leonard Wholey, Marc McConley, Steve Tavan, Brian Bagdonovich, Tim Barrows, Chris Gibson, Sean George, and Scott Rasmussen. Band-limited guidance and control of large parafoils. In *20th AIAA Aerodynamic Decelerator Systems Technology Conference and Seminar*, page 2981, 2009.

[15] Nathan Slegers and Jonathan D Rogers. Terminal guidance for complex drop zones using massively parallel processing. In *AIAA Aerodynamic Decelerator Systems (ADS) Conference*, page 1343, 2013.

[16] Nathan Slegers, Andrew Brown, and Jonathan Rogers. Experimental investigation of stochastic parafoil guidance using a graphics processing unit. *Control Engineering Practice*, 36:27–38, 2015.

[17] Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.

[18] Brandon Luders. *Robust sampling-based motion planning for autonomous vehicles in uncertain environments*. PhD thesis, Massachusetts Institute of Technology, 2014.

[19] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894, 2011.

[20] Yoshiaki Kuwata, Justin Teo, Gaston Fiore, Sertac Karaman, Emilio Frazzoli, and Jonathan P How. Real-time motion planning with applications to autonomous urban driving. *IEEE Transactions on Control Systems Technology*, 17(5):1105–1118, 2009.

[21] Lester E Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of mathematics*, 79(3):497–516, 1957.

[22] Arthur E Bryson. Optimal control-1950 to 1985. *IEEE Control Systems*, 16(3):26–33, 1996.

[23] Jeong hwan Jeon, Sertac Karaman, and Emilio Frazzoli. Anytime computation of time-optimal off-road vehicle maneuvers using the rrt. In *Decision and Control and European Control Conference (CDC-ECC), 2011 50th IEEE Conference on*, pages 3276–3282. IEEE, 2011.

[24] Chang-bae Moon and Woojin Chung. Kinodynamic planner dual-tree rrt (dt-rrt) for two-wheeled mobile robots using the rapidly exploring random tree. *IEEE Transactions on Industrial Electronics*, 62(2):1080–1090, 2015.

[25] Andrew W Moore. An intoductory tutorial on kd-trees. 1991.

[26] Aaron Cole Ellertson. Analytic chance constraints for the robust guidance of autonomous parafoils. Master's thesis, Massachusetts Institute of Technology, 2015.

[27] Matthew Robert Stoeckle. Fault detection, isolation, and recovery for autonomous parafoils. Master's thesis, Massachusetts Institute of Technology, 2014.

[28] Timothy M Barrows. Apparent mass of parafoils with spanwise camber. *Journal of Aircraft*, 39(3):445–451, 2002.

[29] Richard Benney, Mike Henry, Kristen Lafond, Andrew Meloni, and Sanjay Patel. Dod new jpads programs and nato activities. In *20th AIAA Aerodynamic Decelerator Systems Technology Conference and Seminar*, page 2952, 2009.

[30] Ultrafly precision guided airdrop system. *http://www.waymore.com/Products/Military-Products*. Accessed: 04-15-2017.

[31] Digital terrain elevation data level 2 (dted2). *http://www.gcs.gov.sa/En/ProductsAndServices/Products/DigitalElevationModels/Pages/default.aspx*. Accessed: 12-05-2016.

[32] Armin Hornung, Kai M Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. Octomap: An efficient probabilistic 3d mapping framework based on octrees. *Autonomous Robots*, 34(3):189–206, 2013.

[33] Yangming Li and Edwin B Olson. Extracting general-purpose features from lidar data. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 1388–1393. IEEE, 2010.