UNIVERSITY OF CALIFORNIA
SANTA CRUZ

COOPERATIVE NAVIGATION IN UNCERTAIN ENVIRONMENTS

A dissertation submitted in partial satisfaction of the requirements for the degree of

BACHELOR OF SCIENCE

in

COMPUTER ENGINEERING

by

Jeremy Christopher Crowley

June 2018

The Dissertation of Jeremy Christopher Crowley
is approved:

Professor Ricardo Sanfelice, Chair

Professor Gabriel Elkaim

Dean Alexander L. Wolf

Contents

List of Figures

Abstract

Cooperative Navigation in Uncertain Environments

by

Jeremy Christopher Crowley

Navigating in a partially known environment lends to a range difficulties in decision making. Cooperative navigation can alleviate certain issues in navigating in uncertain environments, however, the decision making is still not trivial. In this thesis, a pattern based waypoint recognition and cost based trajectory optimization scheme is presented for a quadrotor with a fixed radius of view in the environment. The path planning algorithm is tested in a range of environments and a specific type of environment, a complex environment with one or more dead ends, is investigated in depth.

Acknowledgments

Ricardo Sanfelice,

for his guidance and support throughout my undergraduate career.

William Van Hyning,

for his complementary work in obstacle detection

Chapter 1

Navigation in Partially Known Environments

Autonomous navigation of Unmanned Aerial Vehicles (UAVs) has recently become a point of interest in academic research due to the versatility of UAVs and their potential to be applied to a wide range of applications. The research community for guidance, navigation, and control of UAVs is thriving, but commercial deployment is significantly delayed due to the safety concerns of having UAVs flying over populated areas. While certain commercial applications like delivery and surveillance place a lot of importance on these safety concerns, emergency and natural disaster UAV aided response is a field where the potential damage caused by a small UAV crashing is negligible compared to danger the emergency/disaster imposes on the area. In a disaster stricken urban environment, a UAV would be able to provide an invaluable amount of help in the form of medical supplies, nutritional resources, and survivor identification.

Several factors are introduced when operating UAVs in an environment where an accident or disaster just took place. The UAV must be capable of mapping its surrounding environment due to the rapid and significant changes that takes place in urban environments during natural disasters. Thus, it cannot be assumed that the previous knowledge of the urban environment is still correct and appropriate imaging systems should be used like in [5]. The changes that occur are often violent and chaotic, introducing a large degree of danger to anything in close proximity. For this reason, navigating as a single UAV carrying the expensive imaging technology as a payload is not ideal. Cooperative navigation is a well known solution to increase efficiency and safety for the UAVs [4].

This project proposes a cooperative path planning scheme for two quadrotors in a partially known environment that aims to minimize the required sensors and potential danger to expensive hardware while achieve similar results to path planing with full knowledge of the environment. The proposed solution employs a high altitude UAV to gather information about the environment below and a low altitude UAV navigating in uncertain environment. We assume the high altitude UAV is not in any danger of colliding with an obstacle while the low altitude UAV has a high chance of colliding with an obstacle. By using a high altitude UAV to scan the environment below, we can obtain a large amount of information from a different perspective, which is particularly beneficial in urban areas where thoroughfares can be seen from high altitudes. The high altitude UAV is assumed to retain the same position as the low altitude UAV, obtaining environmental data about a fixed radius around the low altitude UAV.

Chapter 2

Agent and Environment Representation in Simulation

For simulations, we will assume the that the high altitude quadrotor is able to accurately map the environment around the low altitude quadrotor and maintain the same (x, y) position as the low altitude quadrotor. We model the high altitude quadrotor, x_{h}, and the low altitude quadrotor, x_{l}, as a first-order integrator, a reasonable model when feedback controllers like in [3] are in the loop.

$$\dot{x}_{\mathrm{l}} = u \qquad\qquad \dot{x}_{\mathrm{h}} = u \tag{2.1}$$

where $x_{\mathrm{l}} = (x_{\mathrm{l1}}, x_{\mathrm{l2}}, x_{\mathrm{l3}}) \in \mathbb{R}^3$ and $x_{\mathrm{h}} = (x_{\mathrm{h1}}, x_{\mathrm{h2}}, x_{\mathrm{h3}}) \in \mathbb{R}^3$ is the position of the low altitude quadrotor and the high altitude quadrotor, respectively, and $u \in \mathbb{R}^3$ is the input. We apply the constraint

$$X_l = ([1,n] \times [1,m] \times \{1\}) \tag{2.2}$$

to the low altitude quadrotor, where m and n are the height and length of the environment. The z axis constraint of 1 is selected as a low value to indicate that the low altititude quadrotor is not able to fly over the obstacles int he environment. We apply the constraint

$$X_h = ([1,n] \times [1,m] \times \{\hat{x}_{\mathrm{h3}}\}) \tag{2.3}$$

to the high altitude quadrotor. The constraint for the z axis is given by, $\hat{x}_{\mathrm{h3}} = r(tan^{-1}(\frac{f_{\mathrm{v}}}{2}))$, where $0 < f_{\mathrm{v}} < \pi$ is the angle of the field of view of the high altitude quadrotor's imaging sensor and $r \in \mathbb{Z}_{\geq 0}$ is the desired radius of view. We take the input constraint of the low altitude quadrotor and the high altitude quadrotor to be

$$U_{\mathrm{l}} = U_{\mathrm{h}} = \cup_{i=1}^{K} v(\cos(2\pi i/K), \sin(2\pi i/K), 0) \tag{2.4}$$

where v denotes the speed and K is a discretization parameter for the number of allowable trajectories. We set the variables in all simulations to $v = 1$ and $K = 8$. This restricts the the quadrotor to make a movement to a coordinate that is adjacent to the current coordinate the quadrotor occupies.

For simplicity, we will only consider the low altitude quadrotor for the simulations and we will ignore the z position when defining simulation parameters. To reflect this, the environment will be represented with 2 dimensions as an $m \times n$ cell decomposed map similar to the approximate cell decomposition used in [1], except the quadtree method is not employed and instead all cells have the same fixed dimensions. Just like the connectivity graph described in [2], the cell decomposed map shares a one to one relationship with a directed graph, G, where each node, $N_{ij} \in G$, corresponds to a cell, C_{ij}, in the cell decomposed map. Each node has two parameters, type, N_{ij}^T, and location N_{ij}^L, that are used in the path planning algorithm. The type parameter can be one of the following three: free space - $\mathcal{F}$, obstacle boundary - $\mathcal{B}$, or obstacle $\mathcal{O}$. The location parameter can be one of the following two: edge - $\mathcal{E}$ or middle - $\mathcal{M}$.

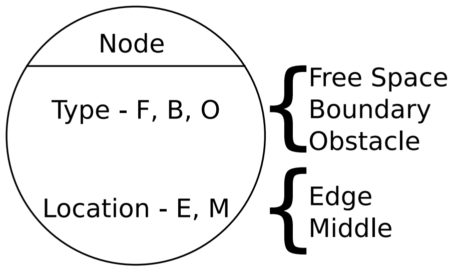

Figure 2.1: Node representation of a cell in the cell decomposed environment

The directed edge, $E_{i_1,j_1 \to i_2,j_2}$, denotes the edge from node N_{i_1,j_1} to node N_{i_2,j_2} and the weight, $W_{i_1,j_1 \to i_2,j_2}$, corresponds to the cost of the traveling from cell C_{i_1,j_1} to cell C_{i_2,j_2} in the environment. Two edges exist between all adjacent cells in the cell decomposed map, resulting in a strongly connected graph. The logic behind creating these edges is described in the algorithm below

Algorithm 1: Add Edges to Directed Graph

```
1 for Each cell, $C_{i_1,j_1}$, in the environment do
2     for Each cell, $C_{i_2,j_2}$, adjacent to cell $C_{i_1,j_1}$ in the environment do
3         Create edge $E_{i_1,j_1 \to i_2,j_2}$ with weight $W_{i_1,j_1 \to i_2,j_2}$
4         Create edge $E_{i_2,j_2 \to i_1,j_1}$ with weight $W_{i_2,j_2 \to i_1,j_1}$
5     end
6 end
```

The weights $W_{i_1,j_1 \to i_2,j_2}$ and $W_{i_2,j_2 \to i_1,j_1}$ in Algorithm 1 are determined by a scalar multiplier that is dependent on the node types $N^T_{i_2,j_2}$ and $N^T_{i_1,j_1}$, respectively, and the distance from cell C_{i_1,j_1} to C_{i_2,j_2} and C_{i_2,j_2} to C_{i_1,j_1}, respectively. The multipliers for traveling to a free space cell, an obstacle boundary cell, and an obstacle cell are defined by $\lambda_{\mathcal{F}}$, $\lambda_{\mathcal{B}}$, and $\lambda_{\mathcal{O}}$, respectively. Given there are three possible multipliers, we can represent the value of the weight of each edge as

$$W_{i_1,j_1 \to i_2,j_2} = \begin{cases} d(\lambda_{\mathcal{F}}) & N^T_{i_2,j_2} = \mathcal{F} \\ d(\lambda_{\mathcal{B}}) & N^T_{i_2,j_2} = \mathcal{B} \\ d\lambda_{\mathcal{O}}) & N^T_{i_2,j_2} = \mathcal{O} \end{cases} \tag{2.5}$$

where d is the distance from cell C_{i_1,j_1} to C_{i_2,j_2}. Referring back to Algorithm 1, only adjacent cells are connected by an edge, thus $d = 1$ under the condition $i_1 = i_2 \vee j_1 = j_2$ (traveling one cell in the x direction or y direction) and $d = \sqrt{2}$ under the condition $i_1 \neq i_2 \wedge j_1 \neq j_2$ (traveling once cell diagonally). This means that there are six possible values for the weight of an edge in graph G.

The logic in Algorithm 1 results in the property that all edges in the graph have an opposite counterpart to them, and the two do not necessarily have equal weights. That is to say, if $E_{i_1,j_1 \to i_2,j_2}$ exists, then $E_{i_2,j_2 \to i_1,j_1}$ exists, and $W_{i_1,j_1 \to i_2,j_2}$ may or may not equal $W_{i_2,j_2 \to i_1,j_1}$ depending on the type parameter of each node. Applying Algorithm 1 to a 3×2 cell decomposed map results in the graph shown in Figure 2.2.

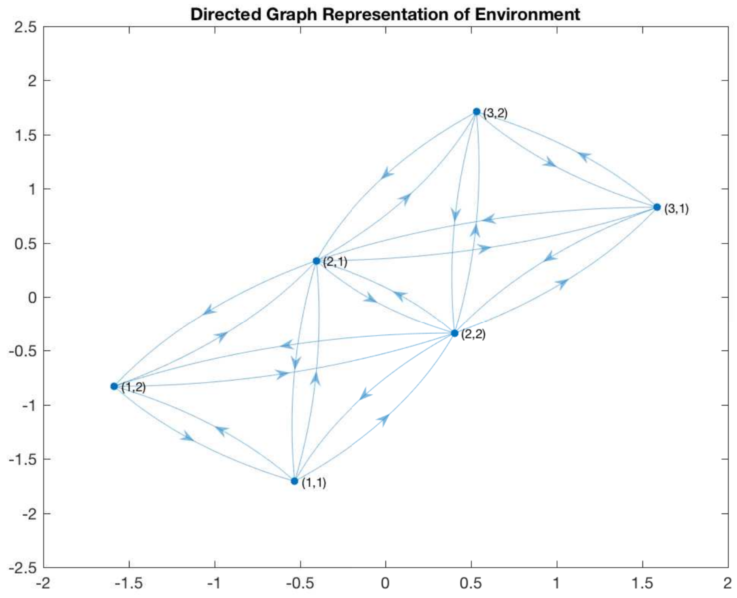

Figure 2.2: Directed Graph Representation of a 3×2 Cell Decomposed Environment

It can be seen that each edge in the graph has an opposite counterpart to it and that all edge pairs only exist between adjacent coordinates. For the duration of a simulation, a known map of the environment will be constructed based on previous and current areas the high altitude quadrotor has observed. The obstacles in the environment will be static and a continuous path of free space and obstacle boundary nodes from the initial coordinate, x_1^0, to the target coordinate, x_1^* is assumed to be present in each environment (i.e. a feasible path between x_1^0 and x_1^* exists).

Chapter 3

Pattern Based Detection for Points of Interest in Known Environment

We define a point of interest, $p_i \in P$, as a cell in the cell decomposed environment that is determined to be a feasible option to minimize the time taken to navigate from x_1 and x_1^*. Notation for expressing all points of interest is given by $P = \{p_1, p_2, \ldots, p_n\} = \{C_{i_1,j_1}, C_{i_2,j_2}, \ldots, C_{i_n,j_n}\}$. We define the discovered edge, D, as the set of all nodes N_{ij} such that $N_{ij}^L = \mathcal{E}$. An example of the discovered edge is shown in Figure 3.1. A characteristic of the discovered edge is that all nodes in the discovered edge have exactly two non-diagonal, adjacent nodes that are also part of the discovered edge. Thus, a circular doubly linked list is used to represent the discovered edge where each node in the linked list represents one cell in the discovered edge. Traveling from one node to the next or previous node in the linked list represents moving along the discovered edge. This allows for information to be extracted from the circular doubly linked list about the sequence of node types in the discovered edge of the environment.

Shown in Figure 3.1 is the initial state of a simulation. The quadrotor's initial coordinate is given by $x_1^0 = (6, 8)$ and the target destination is $x_1^* = (14, 14)$. A green cell denotes free space, a blue cell denotes an obstacle boundary, and a red cell denotes an obstacle. The darkened cells denote undiscovered nodes, The path planning algorithm only has access to the discovered nodes. The simulation shown in Figure 3.1 has a radius of view $r = 3$ and depicts the discovered edge as a closed shape with a black outline.

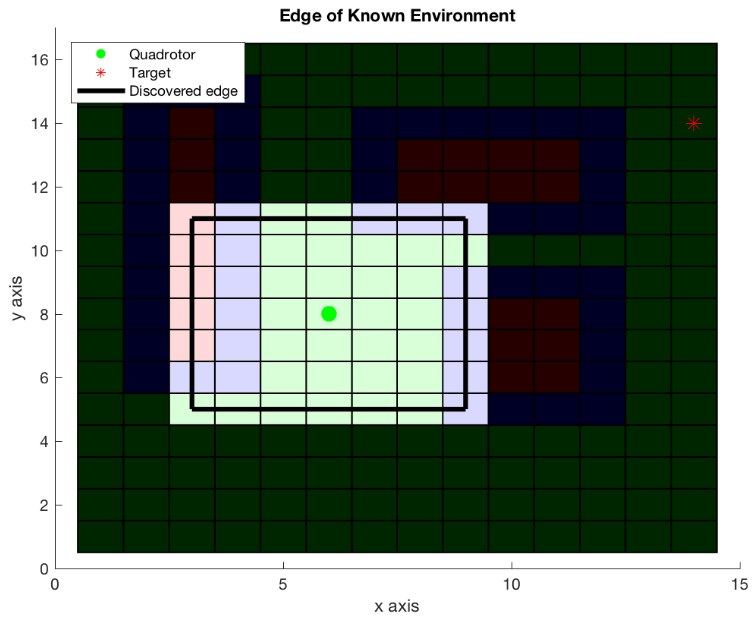

Figure 3.1: Initial State of a Simulation with the Discovered Edge Shown as a Black Line Boundary. The discovered edge is used to find points of interest in the known environment.

With a circular linked list of all the edge nodes, we iterate through the linked list and monitor the node types for specific patterns. Three major scenarios are considered in selecting points of interest.

1. the vector $x_l^* - x_l$ points toward an obstacle that is on the edge of the known environment

2. the vector $x_l^* - x_l$ points toward a region of free space that is on the edge of the known environment

3. x_l^* is in the set of discovered nodes

For item 1. a transition from free space to obstacle boundary (and vice-versa) results in the obstacle boundary cell being selected as a point of interest. This accounts for the target being behind the obstacles because the quadrotor will be navigating towards the edge of the obstacle. For item 2. we introduce the notion of a free space threshold variable. This defines an upper bound on the number of successive free space nodes in the circular linked list before the current free space node is selected as a point of interest. For item 3. we simply add the target cell, x_l^*, as a point of interest if it has been discovered.

Considering the three scenarios mentioned above, Algorithm 2 shows psuedocode for an implementation of that selects points of interest given a circular linked list of the edge of the known environment. To simplify the logic in the psuedocode, Algorithm 2 assumes a transition from N_{i_1,j_1} to N_{i_2,j_2} just occurred while iterating through the circular linked list.

Note that the pseudocode for Algorithm 2 is written for readability and is not optimized for runtime. As noted earlier, in addition to using the points of interest

Algorithm 2: Find Points of Interest

```
1  if $N^T_{i_1,j_1} = \mathcal{F}$ and $N^T_{i_2,j_2} = \mathcal{B}$ then
2      $p_i = N_{i_2,j_2}$                          ▷ Add node $N_{i_2,j_2}$ to points of interest
3  end
4  if $N^T_{i_1,j_1} = \mathcal{B}$ and $N^T_{i_2,j_2} = \mathcal{F}$ then
5      $p_i = N_{i_1,j_1}$                          ▷ Add node $N_{i_1,j_1}$ to points of interest
6  end
7  if $N^T_{i_2,j_2} = \mathcal{F}$ then
8      freeSpaceCount++
9      if freeSpaceCount $\geq$ freeSpaceThreshold then
10         $p_i = N_{i_2,j_2}$                      ▷ Add node $N_{i_2,j_2}$ to points of interest
11         freeSpaceCount = 0
12     end
13 else
14     freeSpaceCount = 0
15 end
```

selected in Algorithm 2, x_1^* is selected as point of interest if it is in the known environment. When Algorithm 2 is run on the environment shown in Figure 3.1 with a free space threshold of 4, the following points of interest are selected $P = \{C_{3,6}, C_{4,11}, C_{7,11}, C_{9,11}, C_{9,9}, C_{9,5}, C_{5,5}\}$.

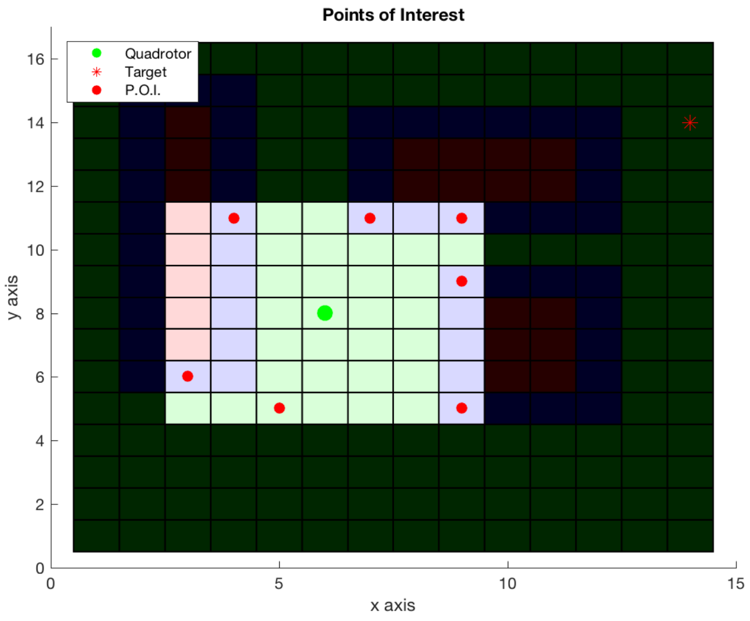

Figure 3.2: Points of Interest in the Known Environment

Shown in Figure 3.2 is the set of points of interest detected in the known environment for this initial condition. For each obstacle, two points of interest are generated (one for each transition to free space). One additional point of interest is generated based on a free space parameter that denotes how many sequential free space nodes there must be before a point of interest is created. Given that the goal of this algorithm is to plan for uncertainty in the undiscovered region of the environment, we can see how the set P can account for a large range of target coordinates and object locations in the undiscovered region.

Chapter 4

Trajectory Planning and Optimization

A standard implementation of Dijkstra's algorithm is run on the directed graph G to find the shortest path from x_l to each point of interest p_i that is stored as a sequence of nodes, or a trajectory, defined as $r_i = \{N_{i_1 j_1}, N_{i_2 j_2}, \ldots, N_{i_n j_n}\}$. The point of interest p_i will always be the last node in the trajectory r_i, thus $N_{i_n j_n} = p_i$ holds true for all trajectories r_i. Dijkstra's algorithm returns the cost associated with the sum of the weights of all edges in the trajectory, however, we are not trying to minimize the cost of reaching a point of interest, instead we are trying to minimize the distance between the quadrotor x_l and the target x_l^*. Thus, we define a custom cost function $J(r)$ where the cost of each trajectory is given by the distance between the target coordinate and the final coordinate in the trajectory.

$$J(r) = \|x_l^* - p\| \tag{4.1}$$

The cost function in Equation 4.1 is a simple cost function that has the effect of favoring trajectories where the end point is close to the target x_1^*. Referring back to the points of interest shown in Figure 3.2, we run Dijkstra's algorithm with the agent's current location, x_1, as the source node and a point of interest, p_i, as the destination node. Running Dijkstra's algorithm on each point of interest results in the set of trajectories shown in Figure 4.1.

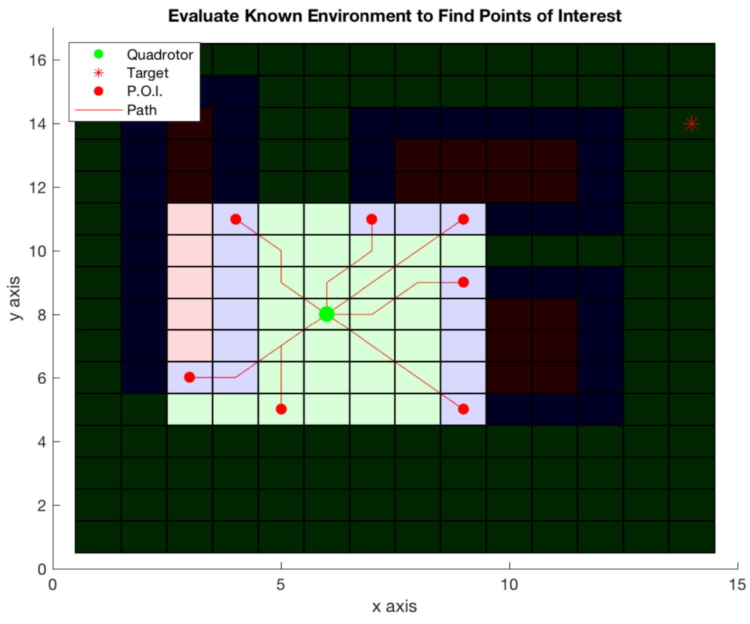

Figure 4.1: Trajectories to Points of Interest

The minimum cost trajectory is found by applying Equation 4.1 to each trajectory. The quadrotor then moves forward to the next cell in the minimum cost trajectory and the path planning process is repeated. Figure 4.2 shows the optimal trajectory being selected and the quadrotor moving to the next node in the trajectory. Figure 4.3 shows the path planning algorithm being repeated on the new known environment.

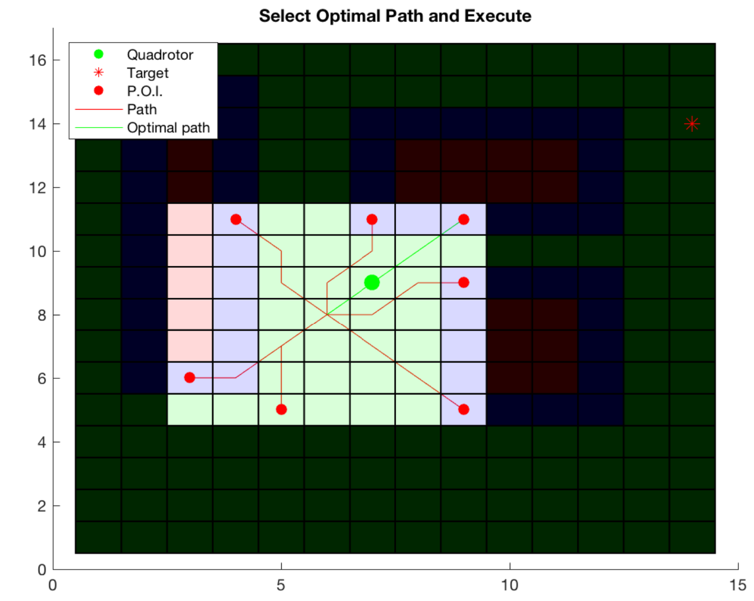

Figure 4.2: Optimal Trajectory and Next Position of UAV

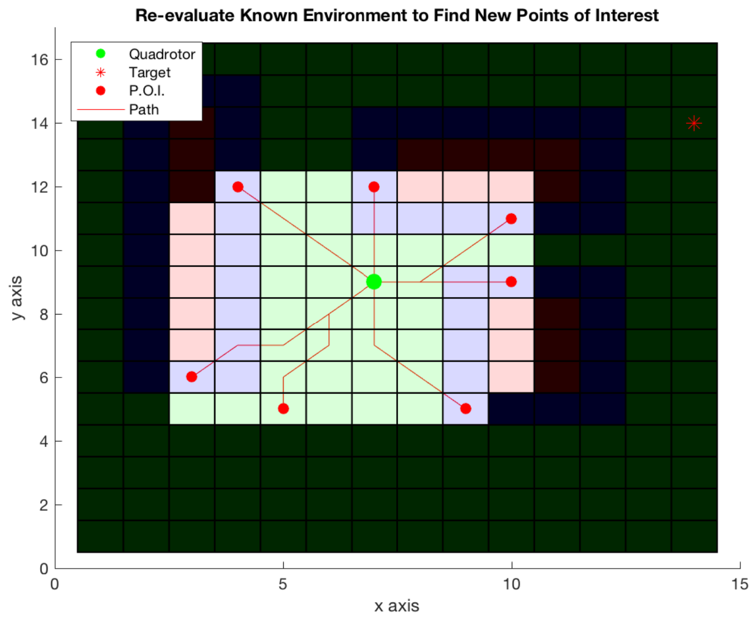

Figure 4.3: Path Planning Algorithm Repeated on New Known Environment

In this application, Dijkstra's algorithm is chosen due to its simplicity and availability. It is one of the most common and best understood shortest path algorithms in graph theory. While Dijkstra's falls short in several scenarios, it is sufficient for finding the shortest path from a source node to a destination node, given a non-negatively weighted graph and enough time.

Dijkstra's algorithm uses a method of breadth-first search to accomplish the task of finding the shortest path. On initialization, all nodes are marked as unvisited and all nodes are given a tentative distance value. The source node is given a distance value of 0, all unvisited nodes are given a distance value of ∞, and the source node is set as the current node. The algorithm then considers all unvisited neighbors of the current node and calculates their tentative distances based on the weight of the edge between the two nodes and the tentative distance associated with the current node. The newly calculated distance is compared with the current tentative distance associated with the neighboring node and the smaller value is assigned to the neighboring node. When all unvisited neighboring nodes have been processed, the current node is marked as visited and the unvisited node with the smallest tentative distance is selected as the current node. This process is repeated until the destination node has been marked as visited. When the destination node has been marked as visited, the algorithm is finished and the distance value associated with the destination node denotes the cost or weight of the shortest path from the source node to the destination node.

As previously stated, any shortest path algorithm can be used in place of Dijkstra's algorithm for this application. In a scenario where runtime is a constraint that should be minimized (i.e. a cyber-physical system with safety concerns), a shortest path algorithm such as A^* should be used as the implementation lends to shorter runtime than Dijkstra's in many cases. Another instance where Dijkstra's algorithm should not be used is the case where the weighted digraph contains negative weights. Instead, a shortest path algorithm such as the Bellman-ford algorithm should be used as it accounts for negative weights being present in the graph. The shortest path algorithm that is selected should be carefully review for each application to make sure the system's requirements are met.

Chapter 5

Simple Environments

We loosely define a simple environment as an environment with several available paths of free space from the initial coordinates to the target coordinates. In this section, we will analyze the algorithm's characteristics with different radii of view in simple environments. Shown in Figure 5.1 is a simple environment that will be used to demonstrate the simulation process. The environment is 10×10 in size ($m = 10$, $n = 10$) with five 1×1 obstacles randomly placed within the bounds of the environment (excluding x_1^0 and x_1^*).

The environment shown in Figure 5.1 has several clear paths of free space that lead from $x_1^0 = (1, 1)$ to $x_1^* = (m, n) = (10, 10)$. In simple environments, we expect the radius of view to have an small effect on the time it takes to reach the target as there is no possibility of ending up in a dead end and having to turn back.

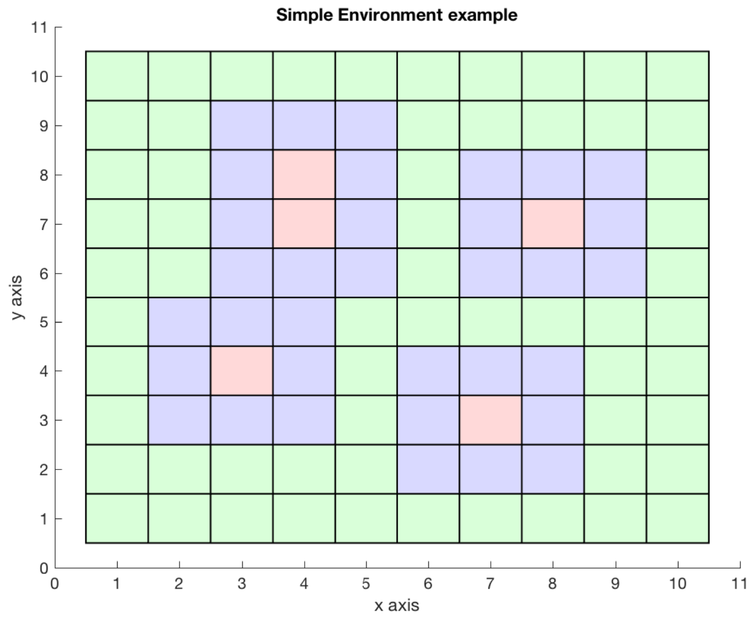

Figure 5.1: Example of a simple environment

To test the correlation between the radius of view and the time it takes for the quadrotor to reach the target, we will generate ten random environments similar to the one shown in Figure 5.1. We will take each environment and run simulations for a range of radii of view $r = 3, 4, \dots, m$. Due to the pattern based logic of this algorithm described in Chatper 3, running simulations with $r < 3$ does not provide consistent results, thus we start with a minimum of $r = 3$. For each environment, the time it takes for the quadrotor to reach the target will be plotted against the radius of view for that simulation. We can expect that increasing the radius of view will lead to a smaller time to reach the target, however, for many simple environments, the path planning algorithm converges to a minimum time at radius of view significantly lower than $r = m$.

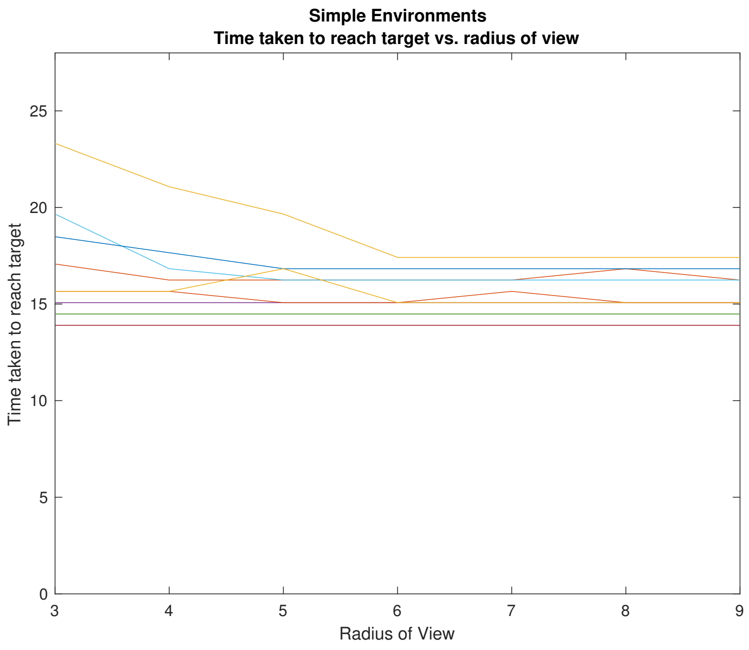

Figure 5.2: Simple environment analysis with an increasing radius of view

Figure 5.2 shows that increasing the radius of view has a positive effect for some environments, while not having an effect at all on other environments. In some cases, we actually see a temporary increase in time take to reach the target at certain radii of view.

We can see that in eight out of the ten environments, by the time the radius of view has reached a value of 6, the time has already converged to the minimum. For the other two environments, there is a small increase in time take to reach the target, followed by a decrease back to the minimum. This can be attributed to the specific layout of the obstacles in the environment and the initial observed points of interest based on the size of the radius of view. Four out of the ten environments achieved the minimum trajectory with all tested values for the radius of view.

Chapter 6

Complex Environments

We loosely define a complex environment as an environment only one viable path to reach the target and one or more dead ends. Creating a complex environment is a more difficult process that creating a simple environment, so the process of creating these environments was not randomized. Instead each environment was created in a custom manner to show the characteristics of this path planning algorithm in such environments. An example of a complex environment is shown in Figure 6.1.

For complex environments, similar initial and target coordinates are used for consistency, $x_1^0 = (1, 1)$ to $x_1^* = (16, 16)$, and increase the size to 16×16 ($m = 16$, $n = 16$) to allow for more possible arrangements of obstacles that attempt to trick the path planning obstacle into guiding the quadrotor down the dead ends. Navigating down a dead end and having to turn back around and attempt a different path adds a considerable amount of time to reaching the target. As the radius of view grows, the dead ends will be discovered sooner, thus decreasing the amount of time taken to reach the target. Shown in Figure 6.1, there is a complex environment with two dead ends. The size of these dead ends is measured by taking the height and width of the free space outlined by red boxes in Figure 6.1. We define the largest dimension of a dead end in an environment as β.

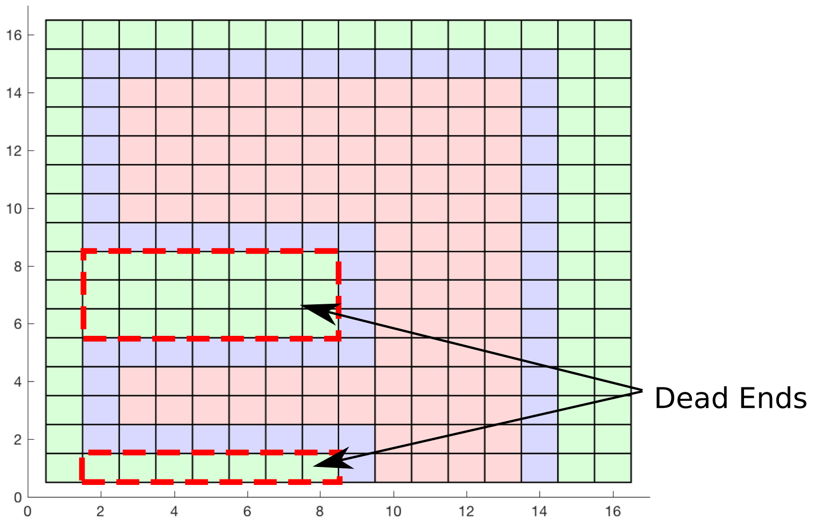

Figure 6.1: Complex environment with two dead ends.

Looking at the whole environment, it is easy to see that the marked dead ends would be ignored since they offer no viable path to safely reach the target. However, with only partial knowledge of the environment, the dead ends can seem like a viable path to reach the target coordinate.

In all environments, the minimum time to reach the target while navigating in free space in this environment can be found by running the simulation with $r = max(m, n)$. This essentially runs Dijkstra's algorithm between the two nodes corresponding to x_1^0 and x_1^*. This results in the minimum possible time to reach the target given the constrains X and U. Shown in Figure 6.2 is the resulting trajectory of four simulations that were run on an environment, where $\beta = 7$, shown in Figure 6.2 with different radii of view $r = 5, 7, 8, 16$. The reasoning behind these values of r for Figure 6.2 is to show a simulation where the radius of view is well below the size of β, a simulation where $r = \beta$, a simulation where the radius of view is just above β, and a simulation where $r = max(m, n)$ (this is guaranteed to show the minimum time trajectory). In the following figures, The black trace is the quadrotor's trajectory with $x_1^0 = (1, 1)$ and $x_1^* = (16, 16)$.

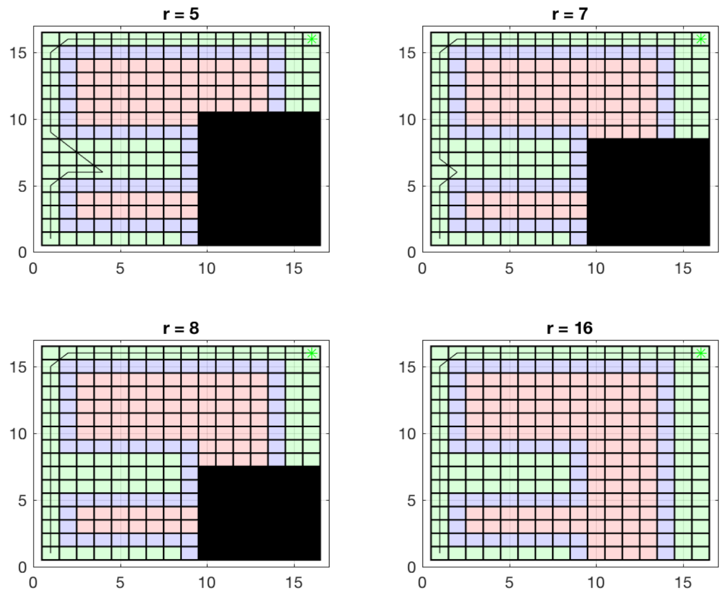

Figure 6.2: Four simulations with different radii of view showing that the optimal trajectory in this environment is achieved when r is greater than the largest dimension of the dead end in the environment. In this case, the largest dimension of the dead end is 7 cells long, so a radius of view of 8 results in the optimal trajectory.

In the first simulation ($r = 5$), the quadrotor navigates 3 cells into the dead end (2×8 cells in size) before turning back around. Once the quadrotor is close enough to see that there the is a continuous sequence of obstacle boundary cells, no points of interest are marked by the discovered edge pattern recognition algorithm. In the next simulation, $r = 7$, and the quadrotor navigates only one cells into the dead end, and when $r = 8$, the quadrotor does not navigate into the dead end. This shows that for environments with dead end features in them, the lowest required radius of view required to each the target in the optimal amount of time is equal to the largest dimension of the dead end. To test this, another environment was created, similar to that in Figure 6.3, however there is only one dead end that is of size 8×9, thus $\beta = 9$.

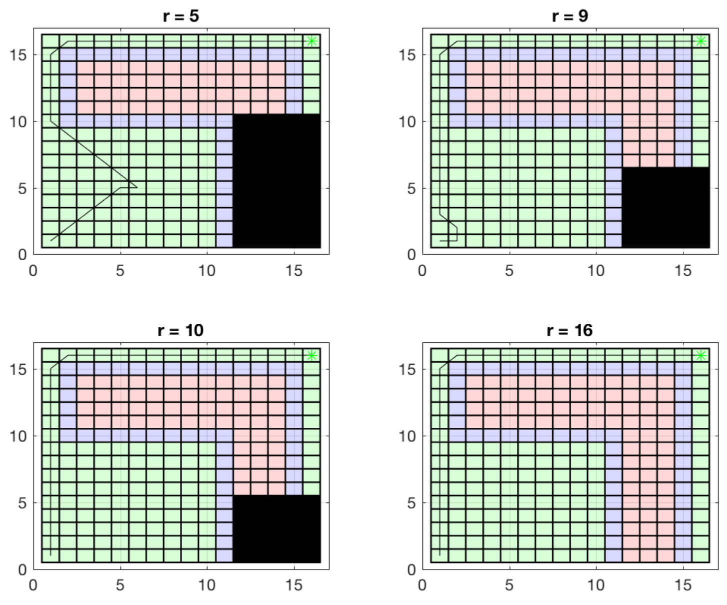

Figure 6.3: Four simulations with different radii of view showing more evidence that the optimal trajectory is achieved when r is greater than the largest dimension of a dead end in the environment. In this case, the largest dimension of the dead end is 9, so a radius of view of 10 results in the optimal trajectory.

To show that the time taken the reach the target converges as the radius of view increases, both environments were simulated with a radius of view $r = 3, 4, 5, \dots, m$. The time taken to reach the target is plotted again the radius of view for the simulation to create Figure 6.4.

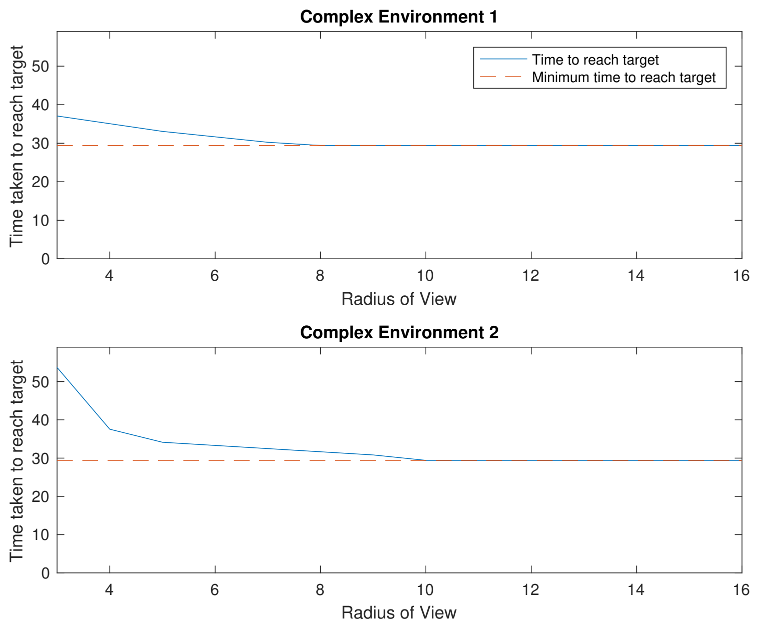

Figure 6.4: Results from simulating complex environments 1 and 2 with an increasing radius of view. The time it takes for the quadrotor to reach the target coordinate decreases until converging to the minimum possible time. The minimum possible time is guaranteed to occur when $r = m$, in this case, when $r = 16$.

In both simulations, the time taken to reach the target converges to the to the minimum possible time at a radius of view significantly lower that $r = m$. To test the algorithm's reliability with different environments, ten complex environments were created and the same simulations, like in Figure 6.4, were run on each environments. The environments used to gather this data are shown in the appendix as Figures A.1, A.2, . . . , A.10. The simulation results were overlaid with each other to show the general pattern of convergence to the minimum possible time.

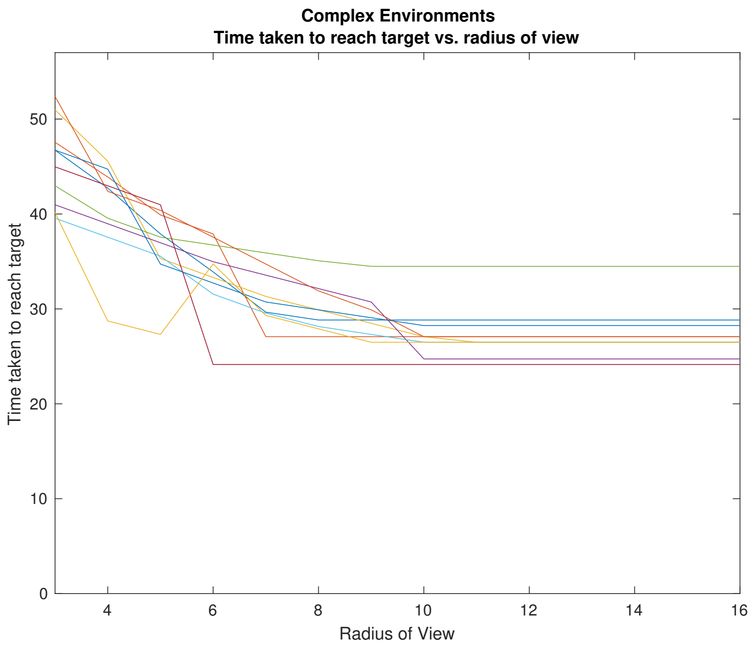

Figure 6.5: Time to reach target converging to a minimum for ten complex environments

This provides evidence that this path planning scheme works for small radii of view and at some values where $r < max(m, n)$ the resulting trajectory taken to get from x_1^0 to x_1^* is the optimal one. With some complex environment, a larger radius of view is required to reach the target in the optimal or near optimal time but for the majority of environments, the time taken to reach the target converges to the minimum possible time when $r < max(m, n)$.

Chapter 7

Future Work

While this algorithm works well for many scenarios, there are several shortcomings that compromise the efficiency of the algorithm in certain scenarios. Three main issues are present that if improved, would increase the reliability of of the algorithm.

1. The cost function is too simple. Only considering the distance from the point of interest to the target can lead to oscillatory nature in select the optimal point of interest. It is difficult to show this problem with figures, instead an simulation with the issue works best. Running the file *OptimizationIssue.m* found in the Github repository[1] will give an example of the oscillatory problem and it becomes clear that the simple cost function given by 4.1 is not effective in certain environment. A proposed solution could be to add a term in the cost function that factors in the length of time for the trajectory. This would cause the quadrotor to favor points of interest that are closer in distance to its location with similar distances to the

[1]https://github.com/JeremyCrowley/CoopNav/blob/master/pathplanning/thesis/OptimizationIssue.m

target.

2. The simplicity of the model used for the quadrotor is not accurate. A more sophisticated model and a more sophisticated trajectory generation algorithm would lead to realistic simulations.

3. Only accounting for one discovered edge in the environment is limiting. Currently, the algorithm cannot handle multiple discovered edges in the environment, causing the simulation to crash or only run the pattern recognition algorithm on one of the discovered edges.

Chapter 8

Conclusion

Path planning in partially known environments is a complex problem that relies heavily on the amount of known information about the environment. Using a cooperative navigation scheme with a high altitude quadrotor to gather environmental information gives the system an advantage in that the ability to identify and localize thoroughfares in urban environments from high altitudes is possible with high quality imaging technology. The benefits of a high altitude quadrotor were taken advantage of and a cooperative path planning scheme was designed around the assumption that given the appropriate imaging technology, an urban environment would be mappable from high altitudes.

In this thesis, a path planning algorithm was designed and simulated to show proof of concept for the proposed cooperative navigation framework given limited environmental knowledge. The algorithm was tested on two varieties of environments defined as simple environments and complex environments. In both types of environments, the time taken to reach the target converged to the minimum possible time, in most cases. with a radius of view significantly lower than the size of the environment. In certain simple environments, the quadrotor took the optimal path regardless of the radius of view. It was also found that the relative size of the dead ends to the radius of view plays a important role in the algorithms ability to select the minimum time trajectory to the target. With larger dead ends, a larger radius of view is required in order to reach the target in optimal time.

Developing path planning algorithms that rely on limited information is a necessary step in the integration of UAVs into our medical and emergency response system. Considerations must be taken to ensure that these algorithms account for possible changes that have occurred in the environment and are able to reliably and accurately map and navigate the environment to complete a mission in potentially dangerous and unknown environments.

Appendix A

Appendix

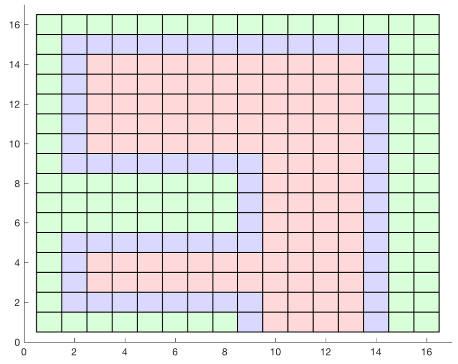

Figure A.1: Complex Environment 1

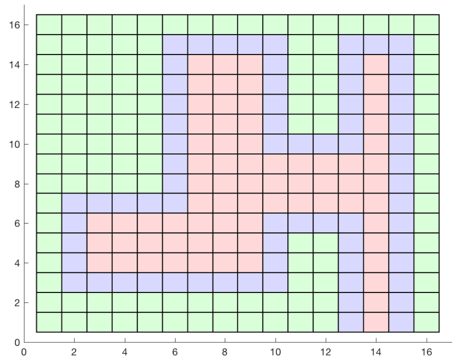

Figure A.2: Complex Environment 2

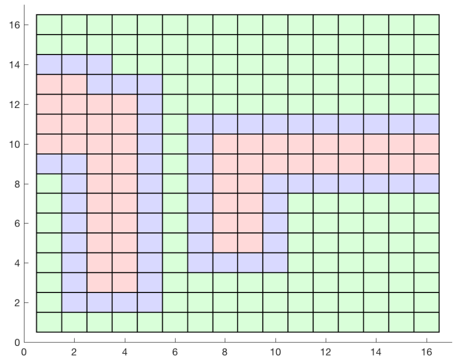

Figure A.3: Complex Environment 3

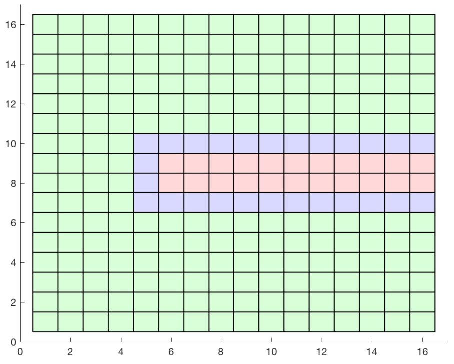

Figure A.4: Complex Environment 4

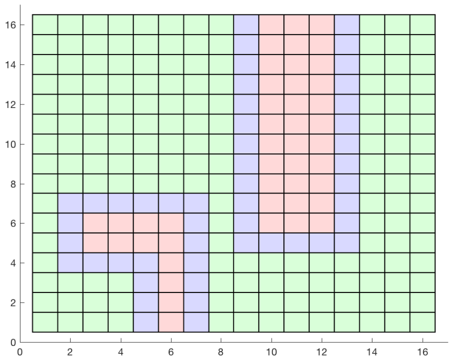

Figure A.5: Complex Environment 5

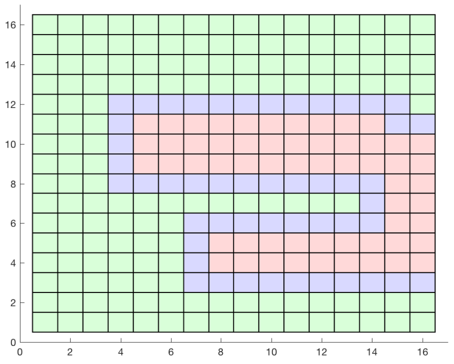

Figure A.6: Complex Environment 6

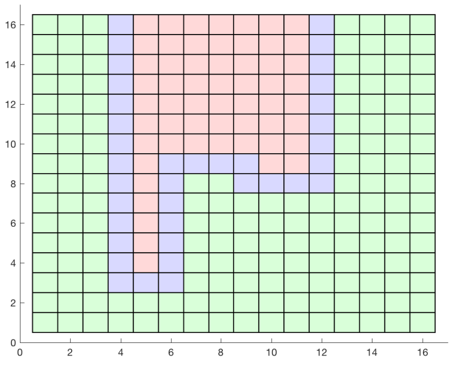

Figure A.7: Complex Environment 7

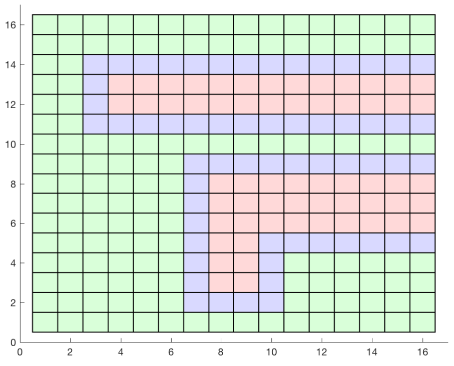

Figure A.8: Complex Environment 8

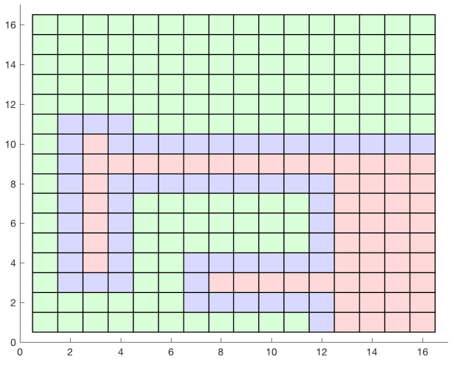

Figure A.9: Complex Environment 9

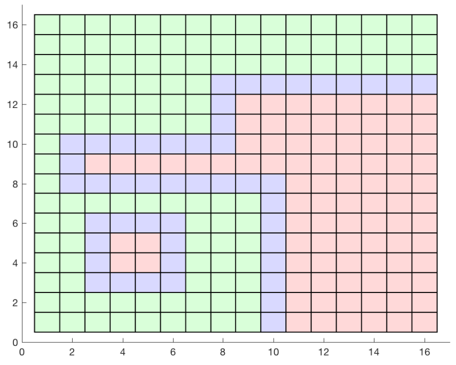

Figure A.10: Complex Environment 10

Bibliography

[1] T. Arney. An efficient solution to autonomous path planning by approximate cell decomposition. In *2007 Third International Conference on Information and Automation for Sustainability*, pages 88–93, Dec 2007.

[2] C. Cai and S. Ferrari. Information-driven sensor path planning by approximate cell decomposition. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 39(3):672–689, June 2009.

[3] P. Casau, R. G. Sanfelice, R. Cunha, D. Cabecinhas, and C. Silvestre. Robust global trajectory tracking for a class of underactuated vehicles. *Automatica*, 58:90 – 98, 2015.

[4] A. Ryan, M. Zennaro, A. Howell, R. Sengupta, and J. K. Hedrick. An overview of emerging results in cooperative uav control. In *2004 43rd IEEE Conference on Decision and Control (CDC) (IEEE Cat. No.04CH37601)*, volume 1, pages 602–607 Vol.1, Dec 2004.

[5] Serkan Ural, Ejaz Hussain, KyoHyouk Kim, Chiung-Shiuan Fu, and Jie Shan. Building extraction and rubble mapping for city port-au-prince post-2010 earthquake with

geoeye-1 imagery and lidar data. *Photogrammetric Engineering and Remote Sensing*, 77(10):1011–1023, 2011.