2011-04-08

Multi-Resolution Obstacle Mapping with Rapidly-Exploring Random Tree Path Planning for Unmanned Air Vehicles

Brett Wayne Millar
Brigham Young University - Provo

Multi-Resolution Obstacle Mapping with Rapidly-Exploring Random Tree Path Planning for Unmanned Air Vehicles

Brett Wayne Millar

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Tim W. McLain, Chair
Randy Beard
Mark B. Colton

Department of Mechanical Engineering

Brigham Young University

June 2011

ABSTRACT

Multi-Resolution Obstacle Mapping with Rapidly-Exploring
Random Tree Path Planning for
Unmanned Air Vehicles

Brett Wayne Millar
Department of Mechanical Engineering, BYU
Master of Science

Unmanned air vehicles (UAVs) have become an important area of research. UAVs are used in many environments which may have previously unknown obstacles or sources of danger. This research addresses the problem of obstacle mapping and path planning while the UAV is in flight. Online obstacle mapping is achieved through the use of a multi-resolution map. As sensor information is received, a quadtree is built up to hold the information based upon the uncertainty associated with the measurement. Once a quadtree map of obstacles is built up, we desire online path re-planning to occur as quickly as possible. We introduce the idea of a quadtree rapidly-exploring random tree (RRT), which will be used as the online path re-planning algorithm. This approach implements a variable sized step instead of the fixed-step size usually used in the RRT algorithm. This variable step uses the structure of the quadtree to determine the step size. The step size grows larger or smaller based upon the size of the area represented by the quadtree it passes through. Finally this approach is tested in a simulation environment. The results show that the quadtree RRT requires fewer steps on average than a standard RRT to find a path through an area. It also has a smaller variance in the number of steps taken by the path planning algorithm in comparison to the standard RRT.

Keywords: UAV, map building, quadtree, rapidly-exploring random tree, path planning

ACKNOWLEDGMENTS

Though long in coming, I have had the help and support of many individuals in the completion of this thesis. I would first like to thank my advisor, Professor Timothy McLain, for his feedback and patience through the process. I would also like to thank the members of the BYU Magicc Lab. There was always a listening ear available to discuss and talk through whatever was vexing me at the time. I would be remiss without singling out those most helpful. Travis Millet was always willing to help with any problem. Bryce Ready was excellent in helping distill out technical details. Jacob Bishop was instrumental in helping complete this work. I thank him for his willingness to spend many hours to proofread and give feedback. I would also like to acknowledge those outside the Magicc Lab that helped proofread and supported me through the process. I last of all thank my wife Sabrina for her willingness to support me during this effort. She stood beside me, propping me up and pushing me forward. I would not have completed this without her.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1. INTRODUCTION

1.1 Motivation

Unmanned air vehicles (UAVs) have emerged as a feasible tool, capable of performing many tasks in the military and civilian sectors. Roles that UAVs are filling include surveillance, tracking, and search and rescue. Many UAV missions depend on the ability to fly into and out of specific areas, gather data or deliver a payload, and in most cases return to base. Consequently, avoiding potential dangers is of high importance. It is also important to fly routes that optimize mission performance as the mission proceeds.

Often, a UAV sent to accomplish a task will be given a path to fly based upon prior information. This information can include the location of the goal state, a map of the area, or information gathered through previous reconnaissance missions. However, this information can be incomplete, inaccurate, or outdated. Most, if not all, UAV systems depend on GPS for position and navigation. While GPS gives information about the location of the aircraft, it does not give information about possible dangers to the aircraft. Because of the need for the UAV to fly in conditions such as these, there is a need for a method of actively avoiding threats that may have emerged in the area to be flown. While some information about the environment can be known a priori, often other dangers may be present which may not be accounted for on maps. Buildings are not represented on terrain information maps. Terrain information is often only accurate to 3 to 9 meters. Also, if the UAV enters known territory, things may have changed such as the emergence of new obstacles or fallen buildings. Any of these things might lead to potential routes being blocked. The UAV needs to be able to incorporate new information from sensor systems with prior information known about the area in order to produce a path for the UAV to fly.

There are three phases of planning into which planning algorithms can be categorized. These phases are planning, re-planning, and reactively avoiding. The planning phase algorithms are run prior to a mission, and may run for as long as is necessary. These can be run, as necessary,

on powerful computers for extended periods to find optimal paths to fly. The second phase is re-planning. If information about a scenario is incomplete and results in paths planned through areas undesirable areas, this second phase of planning algorithms can be used. The last of the path planning algorithms is reactive avoiding. If the UAV is in a scenario where it does not have the time necessary to run a re-planning algorithm, then it can run algorithms to avoid obstacles in a reactive manner. This type of algorithm often only focuses on getting the UAV out of immediate danger and not on getting the UAV to its goal location.

Since there can be unforeseen obstacles that require deviation from a preplanned route, a method of observing or sensing the obstacle is required. Once a sensor is put on a UAV, there needs to be a method of determining if a newly discovered obstacle lies in the planned path of the UAV. If a previously unseen obstacle is detected, there will be a limited amount of time for the algorithm to re-plan a path before the aircraft strikes the obstacle. Due to this time constraint, sensor readings must be processed efficiently. Most sensor information has uncertainty associated with it, and several sensors, including streaming video and 3D scanning range finders, require high data bandwidth. Using higher bandwidth sensors requires the ability to store and process the information gathered.

Many smaller UAV platforms utilize a camera as a sensor due to its light weight and ability to collect large amounts of information. When implemented on a UAV in a forward facing manner, the camera will return an image stream of those objects which lie in front of the UAV. As the UAV generally is pointed in the direction of its velocity vector, those objects from the image stream can provide information about potential dangers which might lie in the path of the UAV. The problem lies in utilizing the information that the video stream provides to update the path of the UAV. It is a trivial task for a human watching a video stream to identify an object which appears to be growing larger. The human mind interprets this as the object coming closer. While a human might be able to easily identify approaching objects, it is difficult, if not impossible, for a ground station operator to locate obstacles in a video stream and transfer that information into a map of the area.

A computer can also be used to identify objects which potentially lie in the path of the UAV. There are many difficulties associated with obtaining information from video to locate obstacles. It is difficult to accurately determine range information from video when the objects are far away. Storage of information from a sensor is also a problem. Based upon a 30 frame per second video

stream with a 640 by 480 pixel resolution, the video stream can return up to 9.216 million measurements per second. Even if the video stream only returned 5 frames per second, this still results in 1.5 million measurements per second. For these video frames to provide useful information, they must be processed by a depth-from-vision algorithm. Byrne has implemented depth-from-vision using two cameras [1], and is currently implementing depth-from-vision using a single camera. This depth-from-vision approach results in a stream of measurements containing range and bearing information, which can then be used to determine object location. Another difficulty lies in the fact that the potential obstacle is not a single object, it is a collection of measurements, each of which have to be dealt with individually.

These range and bearing measurements represent objects in the real world and need to be mapped so that a path planning algorithm can find and update a safe path between the start and goal locations the UAV must fly between. This mapping must be done quickly and efficiently to give the path planner time to find a safe path for the UAV to fly. This is particularly important when using vision because of the need to process a large stream of data. The method in which the potential obstacles are mapped also influences how quickly the path planner can determine whether or not a portion of the path is safe.

Another problem is then finding a feasible path through the mapped area. As stated previously, the computations must be done in a manner which allows the UAV to continue to fly. Different search algorithms handle different environments in different ways. Some will produce an optimal result, but require more time. Others will return results quickly, but with no guarantee as to their optimality. This thesis focuses on filling the need of an online re-planning algorithm without guaranteeing optimality. An overview of how re-planning relates to the steps of sensing, mapping and actual re-planning can be seen in Figure1.1.

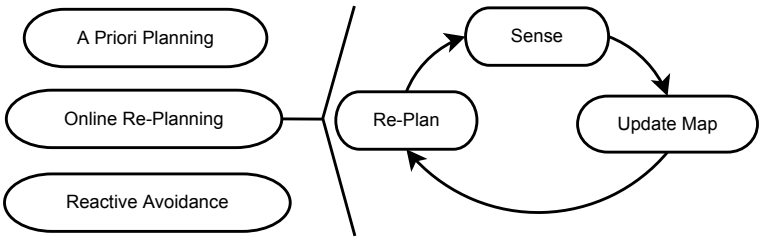


Figure 1.1: Listing of path planners. Sensing, mapping, and planning are subportions of the re-planning step.

1.2 Problem Statement

This research is motivated by the need to fly UAVs in uncertain scenarios and works to develop a path planning methodology for UAVs, based upon range and bearing input from sensors, that will ultimately enable obstacle avoidance during flight. This research develops a mapping solution that will integrate sensor information to create a representation of obstacles that can be used by a path planning algorithm to find and update paths for the UAV to fly.

1.3 Contributions

This research has made contributions in the area of obstacle avoidance, including

- Developing a method for adapting the UAV path with information based upon emergent obstacles using a modified rapidly-exploring random tree with segment length adaptation.
- Creating a mapping strategy to efficiently represent sensed obstacles using a multi-resolution map.
- Demonstrating the validity of the algorithms through simulation testing.

1.4 Outline

The research in this thesis focuses on the re-planning portion of the path planning problem as seen in Figure 1.1. An overview of the sensing portion of this research, along with a discussion of the simulation setup can be found in Chapter 2. Next, Chapter 3 describes the mapping methodology integration of the sensor output to the map environment. Chapter 4 discusses the method of searching for available paths through the map developed in Chapter 3. Chapter 4 also discusses how all three subportions of replanning integrate into one process to help the UAV navigate to its goal locations. In Chapter 5, we will present and discuss simulation results. In Chapter 6, we will re-examine the goals and contributions of this thesis and outline suggestions for future work.

CHAPTER 2. DEVELOPMENT PLATFORM

2.1 Introduction

This research is motivated by the need for a solution to the real-world flight problem of UAV obstacle avoidance. While the thrust of this research is geared towards UAVs, the research is also highly applicable in any area where there is obstacle mapping using sensors and navigating through those areas is required, whether it be unmanned air, underwater, or ground vehicles. Along with the theory that will be developed, the application of the theory helps establish the real purpose for this research. The purpose of this chapter is to describe the simulation environment and how it allows algorithms to be developed. It is then followed up by an overview of the simulated sensing approach known in this research as synthetic vision.

2.2 Simulation and Algorithm Development Environment

While it is beneficial to build and test algorithms in a stand-alone environment, use of a software environment that allows easy deployment of algorithms for simulation implementation enables quick turn around and testing of ideas. The software environment used in this research fills three roles as seen in Figure 2.1. First, it manages a six-degree-of-freedom simulation which can be used to test the algorithms in this research. Second, it manages the flow of state information between the algorithms and the simulation. It also passes command information from the algorithms to the simulation. If desired, this same code could be used to interact with software which controls a hardware UAV to pass state information and commands. The third role which this software environment fills is to provide simulated sensor readings known as synthetic vision. This software environment was created in MATLAB.

The algorithms in this research are dependent upon the state information of the UAV. These states consist of the position and attitude of the UAV. This state information can come via hard-

ware testing or use of a simulation environment. In an effort to standardize the interface of the state information and the simulation, MATLAB is used to create a program which is used as both controller and simulator.

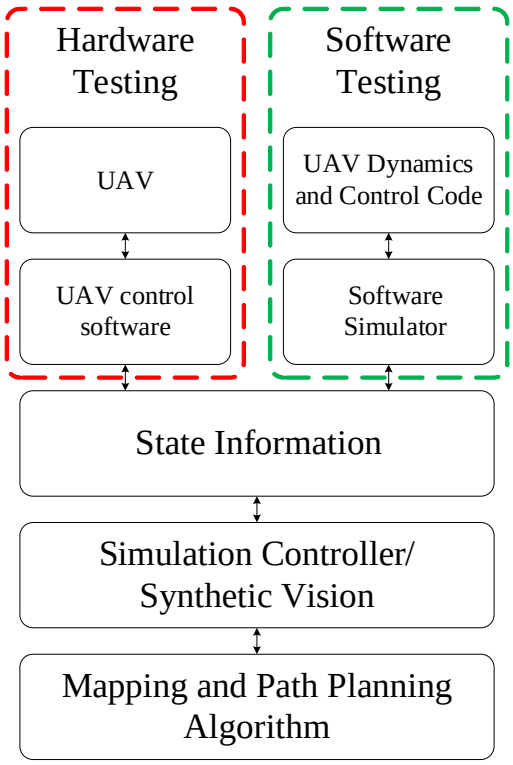


Figure 2.1: Algorithms may be tested using either software or actual hardware to represent the flight vehicle. Because of its relatively low cost and low risk, we use software testing to develop and demonstrate the utility of our re-planning algorithms.

This research utilizes a six-degree-of-freedom model of the aircraft, a description of which can be found in Appendix A. The autopilot code that controls the UAV is compiled into a set of libraries for use in simulation. MATLAB is used to simulate an environment that mimics software to control the hardware UAV and communicate with the autopilot as if it were actually flying. This simulator allows for quick prototyping and implementation of algorithms. The real advantage to this implementation lies in the ability to build algorithms and code and test them in simulation in a manner that allows them to be easily extended to flight testing.

The synthetic vision algorithms used in this project produce a depth map. The depth map, in conjunction with current telemetry information, are input into the mapping algorithm. If the

map is updated, the path is checked to see if it now goes through an unsafe portion of the map. If it does, the search algorithm then finds a new path through non-dangerous space.

2.3 Synthetic Vision

Synthetic vision is a mechanism of simulating depth sensing along rays projected through individual pixel locations of a video camera. As shown in Figure 2.2, sensing is an essential part of the three phase process used in re-planning flight paths. We use synthetic vision to represent the sensing ability of the aircraft.

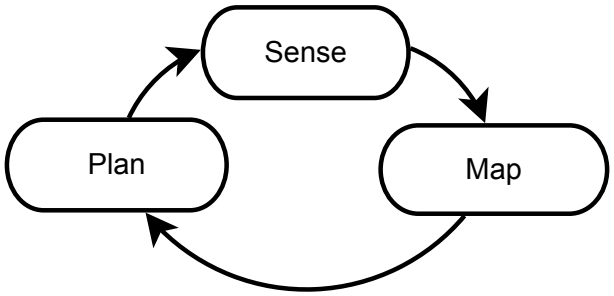


Figure 2.2: Relationship of sensing with mapping and planning

2.3.1 Background

Synthetic vision produces the artificial sensor readings which will be used in this project. It emulates the expected results from depth-from-vision algorithm, not the process normally associated with collecting vision information. In depth-from-vision, algorithms can determine an approximate range to the object in the video image. The algorithms determine the depth based upon how much the video image changes from one frame to the next. Objects far away change little between frames, and as a result, the resulting range is indeterminable. Synthetic vision seeks to simulate that. A typical video frame image is pictured in Figure 2.3. This frame is 640 pixels by 480 pixels, also known as VGA resolution. Since the software written for this project had to both calculate the synthetic vision and then utilize the synthetic vision, a lower 10 pixel by 10 pixel resolution was chosen to reduce computational requirements.

Figure 2.3: An example VGA video frames showing video captured at a resolution of 640 by 480 pixels.

2.3.2 Synthetic Vision Implementation

Synthetic vision works in the following manner. Each pixel x_{ij} within the synthetic vision depth map frame has an associated range measurement. It is by using this range information that the algorithm plans and avoids obstacles. An array of synthetic "pixels" are created via a pinhole camera approximation. The pinhole camera model is a first-order approximation of mapping images in three-dimensional space to a two-dimensional image plane. This requires a set of intrinsic camera parameters, including focal length f and resolution.

In a standard pinhole camera model, a pixel location (u,v) can be computed by starting with the relationship

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{12} \\ r_{21} & r_{22} & r_{33} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}, \tag{2.1}$$

where the rotation matrix R, composed of elements r_{ij}, is used to rotate relative positions X, Y, and Z into positions in the camera coordinate frame x, y, and z. Pixel location (u,v) can then be computed using the focal length in both x and y dimensions $(f_{x,}f_y)$ and another intrinsic camera parameter called a principal point, which is represented by (c_x,c_y) as

$$u = f_x\frac{x}{z}+c_x \tag{2.2}$$

$$v = f_y\frac{y}{z}+c_y. \tag{2.3}$$

This process can be inverted to find the vector located through the center of the pixel location. The following set of equations holds for either the image x or y direction. The field of view α (deg) and resolution E (pixels) can be used to approximate a focal length as

$$f = \tan\left(\frac{\alpha}{2}\right)E. \tag{2.4}$$

The principal point c can be approximated by $c=\frac{E}{2}$. Once we have calculated these terms, we can use them in generating a point on the image plane in three-space using the expressions

$$\frac{x}{z} = \frac{(u-c_x)}{f_x} \tag{2.5}$$

$$\frac{y}{z} = \frac{(v-c_y)}{f_y}. \tag{2.6}$$

When creating measurement in synthetic vision, we are initially unsure of the z component of the location of the obstacle. To find it, we create a unit vector $\hat{e}$ for each pixel in the direction of $(\frac{x}{z}, \frac{y}{z}, 1)$ by normalizing to get

$$\hat{e} = \frac{\begin{bmatrix} \frac{x}{z} \\ \frac{y}{z} \\ 1 \end{bmatrix}}{\left\| \begin{bmatrix} \frac{x}{z} \\ \frac{y}{z} \\ 1 \end{bmatrix} \right\|}. \tag{2.7}$$

In the simulation environment, a ray along $\hat{e}$ is created. The magnitude of the ray is incremented by amount $d_{\text{step}}$and then checked to determine if it has intersected an obstacle. It is by this method that the first obstacle which the ray intercepts is determined. The distance d_{step} can be increased or decreased to change the performance of the simulation. A distance of 5 meters was found to give accurate enough information without increasing the computational load too much.

The ray d is checked over a certain distance d_{ray}. The first time ray passes through an obstacle, the search is terminated and the magnitude of d is returned. If d does not intersect an obstacle before reaching magnitude d_{ray}, the search is terminated. The resulting depth map is an array of values representing distances along rays through the center of the pixels which is called synthetic vision.

2.3.3 Synthetic Vision Example

To help visualize how synthetic vision works, the following example is given. A scenario of the UAV flying can be seen in Figure 2.4, where the UAV is flying towards two buildings. These buildings are located at 450 meters north, 300 meters east and 500 meters north, 550 meters east. Both buildings are 50 meters by 50 meters by 100 meters tall. The UAV is located 500 meters east, 150 meters north. It is pointed straight north with wings level. The lines lines which separate the individual pixels from each other can also be seen.

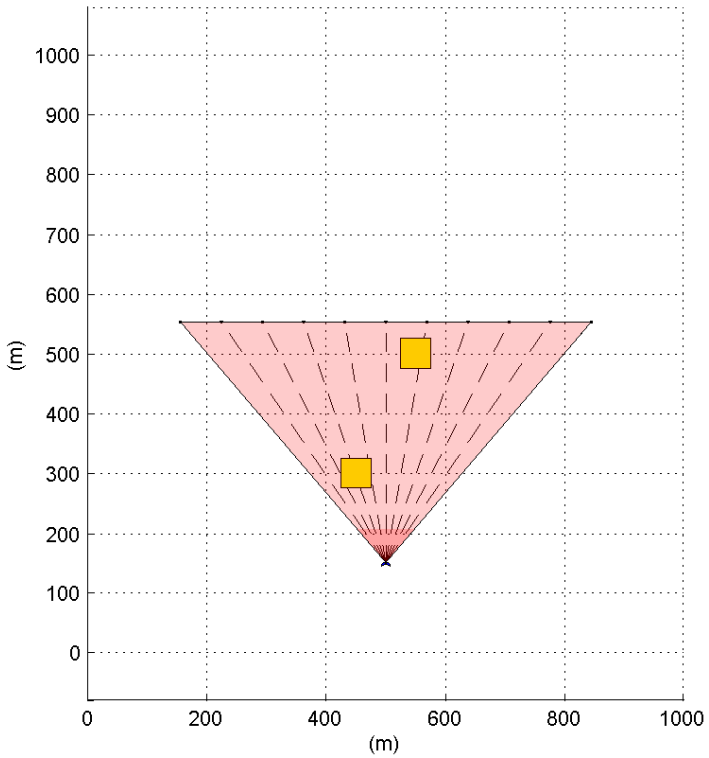


Figure 2.4: Overhead view of environment for synthetic vision example. UAV is shown flying towards two buildings in a 1000 meter by 1000 meter world. The synthetic vision field-of-view can be seen in red.

Figure 2.5 shows the camera view from the UAV location in this example. A ten by ten grid is laid over top to show the location of the synthetic vision pixels. It is along these synthetic pixels lines that a ray is sent out and checked for intersection with an obstacle.

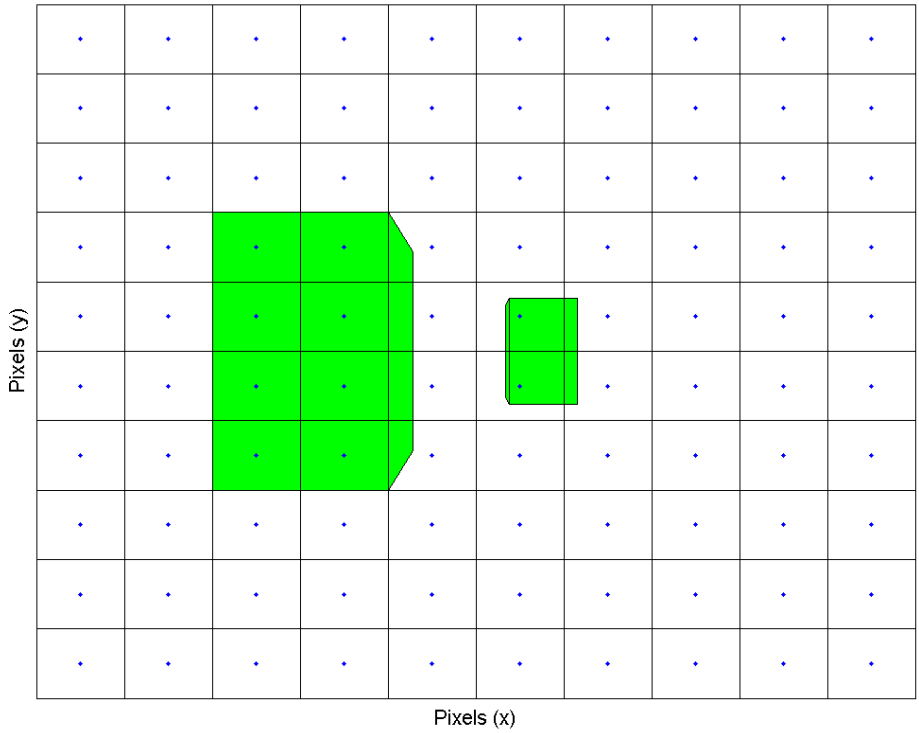


Figure 2.5: World in synthetic vision example as seen from camera perspective. A ten by ten grid of dots is overlaid representing rays at each pixel location.

Figure 2.6 shows the depth map returned from synthetic vision for the view shown in Figure 2.5. The blocks with valid depth measurements are shown in red, and the depth measurements are recorded within the blocks.

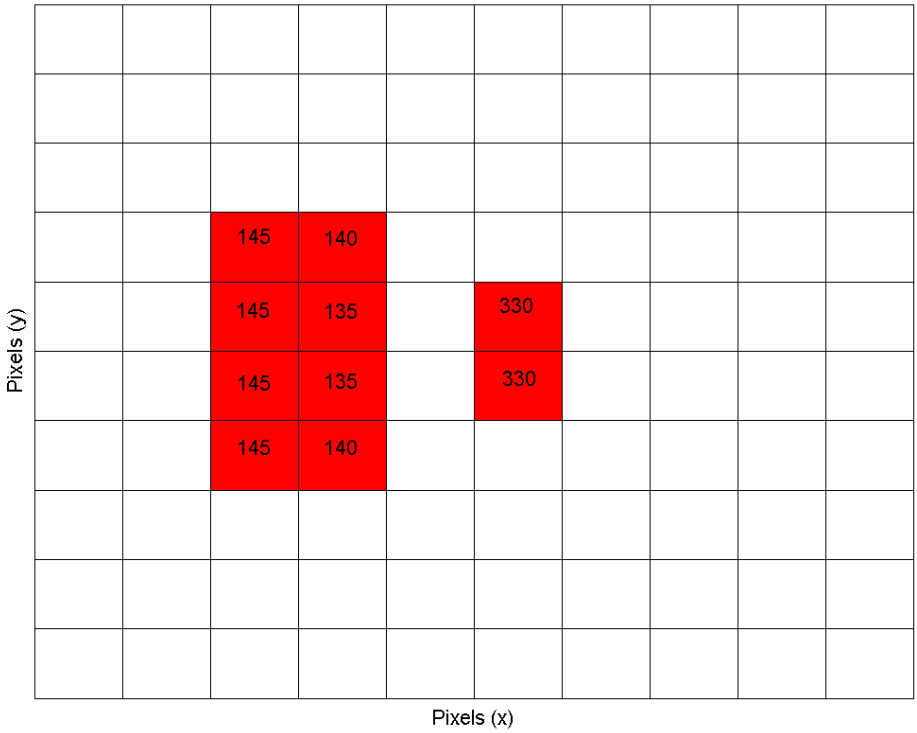


Figure 2.6: Resulting synthetic vision depth map from camera view shown in Figure 2.5.

2.4 Chapter Summary

A software environment has been developed which provides easy integration between stand-alone implementation and simulation testing. This software environment allows for control between the algorithms and simulation as well provide a mechanism for integrating synthetic vision. Synthetic vision has been developed to simulate results from depth-from-vision. These tools will help prove the validity of the algorithm and research.

CHAPTER 3. OBSTACLE MAPPING

3.1 Introduction

UAV mission planning often requires some information about the area where the mission will be flown. Though it may be represented in different ways, often times such information is stored as a map. During the mission, update information may come in the form of sensor readings, data from the mission planner, or other methods. For the purposes of updating the mission map with information acquired during flight, the map storage format should easily integrate new information.

In this chapter is a discussion of the advantages and disadvantages of mapping methodologies, including approaches for incorporating data. Also included is an outline of quadtrees with the reasoning why they were chosen for this research. That is followed by an explanation of sensor measurement integration, including a map update method for input from a vision sensor.

Mapping is a portion of the re-planning cycle and relates to sensing and planning as seen in Figure 3.1.

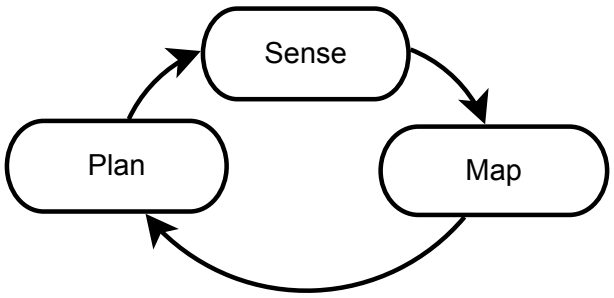


Figure 3.1: Relationship of mapping with sensing and planning

3.2 Background

3.2.1 Literature Review

Research on mapping areas with information obtained from sensors has been pursued for many years. These efforts have yielded various methodologies, each with strengths and weaknesses, resulting in mapping approaches which are suited to some scenarios better than others.

A focus point of current research efforts is on simultaneous location and mapping (SLAM). SLAM is an approach used to map out an environment while calculating the location of the sensor within the environment. The accuracy of SLAM is closely tied to understanding the uncertainties of the location and movement of the sensing unit. SLAM research has been extended to airborne vehicle scenarios [2, 3]. However, as all current implementations of SLAM are based upon trying to localize the observing platform as well as the obstacles to be mapped and avoided, SLAM is not directly applicable to the scope of this research.

Another method of building a map is to use a grid. Pagac, et al. used a grid-based methodology to build up a map using a sensor input [4]. Hsu and Hwang discuss the strategy of using a grid-based mapping method for indoor autonomous robots [5].

A third method of building a map of obstacle information about an area is to use to use a multi-resolution map. One of the main varieties of multi-resolution maps is a quadtree. Ayala, et al. describe the use of quadtrees to represent objects [6]. Yahja, et al. have produced work combining both quadtrees and path planning. They introduce the idea of a framed quadtree [7]. Much of the previous research relating to quadtrees and path planning focused on using the quadtree as a graph, the nodes of which could be connected using a search function. They propose utilizing the quadtree as more than a graph of nodes to search by using it as a map representing space that can be searched. There has also been recent work to optimize the quadtree when using it as a data storage structure [8].

Lingelbach uses a multi-resolution mapping approach that differs from a traditional quadtree. He generates cells of varying dimensions and sizes based upon input information using a method of probabilistic cell decomposition [9]. Burlet, et al. use quadtrees with a Markov decision process path planner [10].

Tsiotras has recently been pursuing work in multi-resolution mapping and path planning for UAVs. He has focused his work on using wavelets to represent the multi-resolution map [11]. Much of his work to date has focused on developing paths through known maps.

Also needed for mapping is a methodology for determining object location. Redding described a method of geo-locating an obstacle on the ground based upon the orientation and location of the aircraft using a flat-earth assumption [12]. Byrne et al. have been producing work using stereo vision for depth perception [1]. Their current work, which focuses on using a single camera for depth sensing, is the basis for the simulated sensor readings for experimentation to be done later.

As mentioned previously, there are three phases of path planning: *a priori* planning, online re-planning, and reactive avoidance as seen in Figure 1.1. This research focuses on online re-planning. Due to the possibility of entering a circumstance under which online re-planning does not have sufficient time to plan a path out, this research anticipates having a secondary reactive avoidance planner running along side the online planner. Sharma, et al. describe the use of a reactive obstacle avoidance [13].

Recently, work was completed by Yu, et al. by approaching this problem using a local coordinate frame [14]. Their focus was obstacle avoidance relative to the UAV and planning paths relative to the UAV. Their approach used a multi-resolution map in a radial coordinate frame to hold information about the obstacles.

3.2.2 Coordinate Frames

Coordinate frames define the environment in which measurements are referenced. A sensor measurement cannot be effectively used unless it is related to a coordinate frame. To build up a map suitable to fly the UAV through, a series of coordinate frames need to be developed. There will be three coordinate frames used in this research, the inertial frame, body frame, and sensor frame.

The first is the inertial frame. It is defined as a north-east-down (NED) frame. The origin of this frame is located at the center of the map with x_i pointed north, y_i pointed east, and z_i pointed towards the center of the earth.

The second frame is the body frame. This refers to a coordinate frame aligned with the body of the UAV. The x_b-axis out the nose, y_b-axis out the right wing, and z_b-axis out the belly of the aircraft. The origin is located at the center of mass (CM) of the aircraft. An image of this can be seen in Figure 3.2.

The third frame is the sensor frame. The sensor frame is defined with the origin located at the sensor. For the purposes of this research, the sensor will be located at the CM of the aircraft and is a forward facing camera. The x_s-axis is defined out the right wing of the aircraft, the y_s-axis is defined positive out the bottom of the aircraft and the z_s-axis is positive out the nose of the aircraft.

An understanding of the coordinate frames allows information from each frame to be used to develop maps and find paths through the environment.

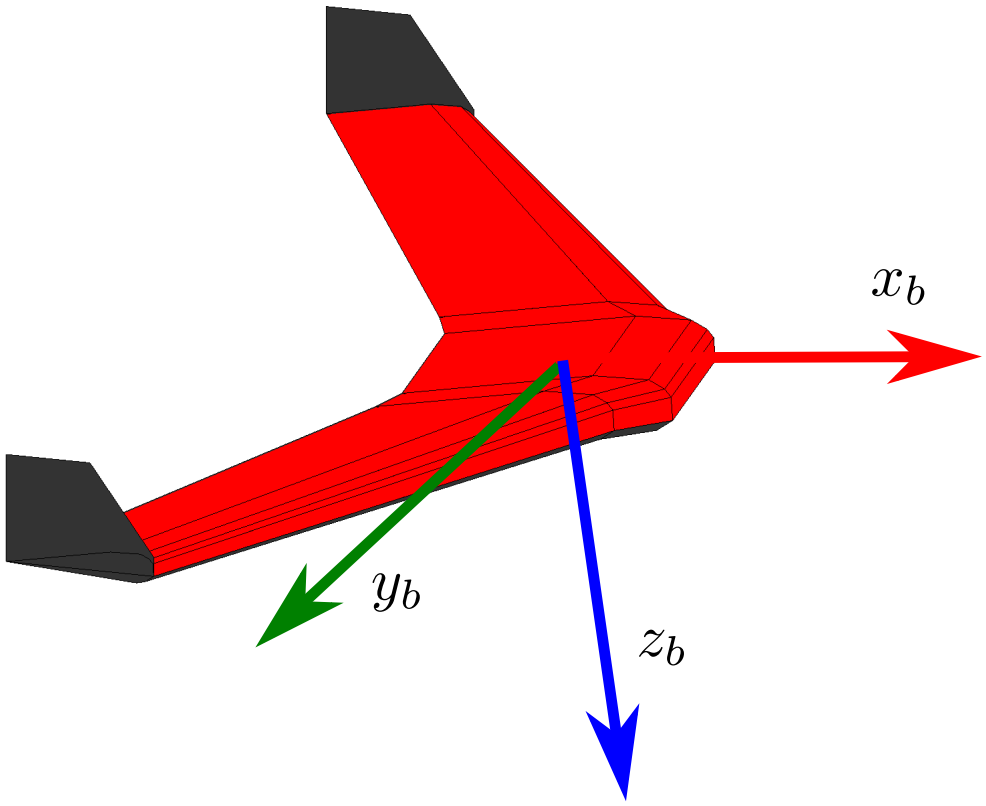


Figure 3.2: UAV coordinate frame

3.2.3 Mapping Methods

This research has several requirements for mapping, the first of which is the ability to incorporate existing information of the area to be flown. It is from this initial information that a path can be planned at the start of the mission. The second requirement of mapping is that the map can be updated with new information throughout the flight. By incorporating updated information, the path planner can re-plan a path should an area be found to be unsafe. The third requirement for the mapping portion of this research is that the map must be numerically efficient to search. By minimizing the memory requirements, this research can be applied to less capable platforms.

Three candidate methods for mapping were considered for use in this research. The first is a discrete map composed of a fixed number of discrete mapping areas. Each of these areas can contain either a true/false value of the existence of an obstacle within the mapping area or a percentage chance of the existence of an obstacle. The second method involves the attempt to map discrete obstacles. The last method discussed will be a multi-resolution mapping approach.

Although we present a method for creating obstacle maps based upon range and bearing extracted from a VGA camera feed, one should not assume this is the only method to obtain this information. Radar and laser range finders represent two viable alternatives.

Flying UAVs requires the knowledge of potential threats in the area of flight. When preparing the mission planning, all obstacles may not be known. To successfully complete a mission, the ability to incorporate new information is critical. Storing this information in an efficient manner is essential to quickly plan and re-plan paths for mission success. In this research, we use a quadtree method to efficiently represent obstacle information. To aid in the presentation of this method and to gain an appreciation of its advantages over other methods, we first introduce fixed-resolution maps, multi-resolution maps, and obstacle collection mapping.

Fixed-resolution Maps

For the purposes of this research, a fixed-resolution map will refer to any map or mapping method describing an area which has been broken down into a fixed number of sub areas, which will be referred to as cells. Generally, these cells will be small enough that an obstacle location and size can be represented to within several meters of accuracy.

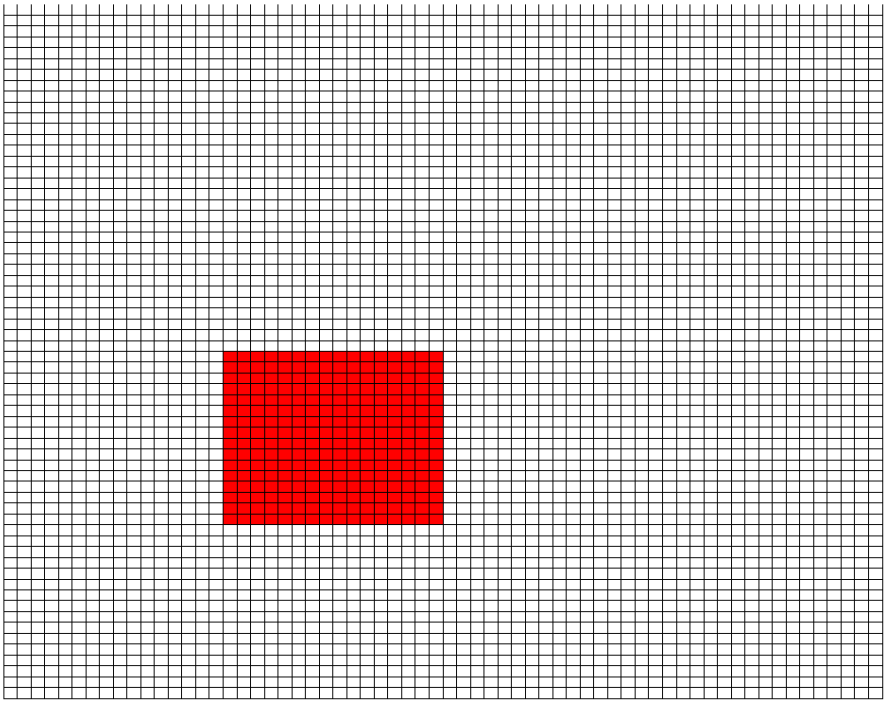


Figure 3.3: A fixed resolution map has a non-changing number of cells to describe an area. This is true whether the area around objects is empty or filled.

Fixed-resolution maps are useful for a number of reasons. The memory for the entire map is preallocated. This means that mapping can proceed without any need to increase the memory. Complete memory allocation occurs with the initialization of the map. Increasing memory allocation for a non-fixed resolution map can take additional time while the algorithm runs, potentially allowing less time for other code to be run.

Locations within the map are accessed for either storing data or retrieving stored data. The use of a fixed-resolution map results in a simple method for accessing specific locations. By simply resolving the desired location on the map into the correct X and Y indices, the data from the sub area of the map which contains that location can be accessed. The use of this method requires little computation.

Not only can the data locations be accessed quickly for data storage or retrieval, the structure of the data storage also lends itself to quick dissemination of information to adjoining cells.

When planning a path through a mapped area, cells need to be checked for obstacles. The grid layout of the data storage means that checking the vacancy of the neighboring area is as simple as incrementing the cell index and checking the area. It is straightforward to implement and requires little additional overhead.

There are also several disadvantages associated with the fixed-resolution mapping method. The first of these is the likelihood of a high number of cells representing open space. This results in a large percentage of allocated memory being unused for storing information about obstacles. This could be a undesirable in memory constrained environments where such memory could be used for other tasks.

Planning a safe path requires that the area which a path passes through be free of obstacles. A fixed-resolution map can be thought of as a single large area composed of a large number of small, fixed areas (cells) to check. The number of areas (cells) to check is equal to the map width (in cells) multiplied by the map height (in cells). Due to the nature of the fixed-resolution map, each of these cells must be checked. This can require the checking of large numbers of cells every time the path planner is called.

As the name "fixed-resolution map" indicates, the resolution is fixed throughout the entire process. While it means that the entire amount of memory can be allocated at the beginning of the run, the strength of a fixed-resolution is also its downside. The map resolution is fixed throughout the process. If the area is not described in sufficient detail, the map resolution cannot be increased.

Multi-Resolution Map

A second method for mapping is the multi-resolution map. A multi-resolution map is similar to a fixed-resolution map in that it contains information covering a large area. This large area is broken down into smaller areas which will be referred to as cells. These cells can vary in size and can be refined while the algorithm is running.

The multi-resolution approach has several abilities which make it useful in certain circumstances. The first of these is faster searching by the path planner due to its ability to represent sparse areas more efficiently than a fixed-resolution map. The path planner searches through each cell of a fixed or a multi-resolution map through which the planned path passes. As the multi-resolution

map is able to represent the sparse area in fewer cells, fewer checks are made by the path planner, thus increasing its execution speed.

Another feature of the multi-resolution map is the ability to represent obstacles with increasing granularity. Due to the uncertainty inherent in sensor readings, the sensed location of the obstacle may not be completely accurate. A multi-resolution map can represent a far away obstacle with high uncertainty in its location with a large cell. It can then refine the shape of the obstacle, increasing the amount of cells which describe its size and shape as the uncertainty in the information about the obstacle decreases.

Due to the uncertain nature of the multi-resolution map, however, the memory required to store the data is increased as the algorithm progresses. This requires memory to be allocated on the fly and also means that the amount of memory required is unknown when the map is created.

While the ability to refine cell size provides great benefits, it also increases the difficulty in identifying which cell adjoins another. The fixed-resolution map provides easy access between adjoining cells. Incrementing the cell index moves the index to the next cell. Because there is no cell index with multi-resolution maps, the structure containing the cells must be searched for the adjoining cell. This adds computational complexity.

Obstacle Collection Mapping

A third method for mapping an area is to build up a collection of obstacles. Upon entering an area sensor data of the location of obstacles can be collected. In a fixed-resolution or multi-resolution map, this data is generally pressed down onto the map and the area in which the obstacle is sensed is considered filled with an obstacle. Both the fixed-resolution and multi-resolution maps require a map area with smaller sub areas where obstacle information can be stored. The first two methods can be thought of as building up a map. Obstacle collection mapping, however, can be thought of as the building up of a discrete set of obstacles.

This third method of mapping obstacles provides some benefits that the other two methods of mapping cannot. As the collection of obstacles is built up, a discrete list of obstacles is produced. This can greatly reduce the amount of checking that must be done by the path planner for a clear path. In large, sparse environments where a fixed-resolution map would have to check each of its

cells along the path, the obstacle collection mapping method would only have a few obstacles to check the path against and would greatly increase the speed of the path planner.

However, collecting obstacles into a set is not an easy task. It is computationally intensive due to the need to combine the series of sensor readings to develop each obstacle. Where the fixed and multi-resolution maps can disregard seemingly redundant information, this second method needs to merge them. Because this method finds discrete obstacles as opposed to filling in a map with areas filled with obstacles, it requires more computation to resolve the individual obstacles.

3.2.4 Quadtrees

Due to its abilities to efficiently represent sparse areas and adapt the resolution of the cells, a multi-resolution approach was selected for this research. There are several major methods of multi-resolution mapping, including quadtrees and wavelets. The quadtree was chosen due to its tree structure and its intuitive representation of area.

A quadtree is a nodal tree structure similar to a binary tree. As the names suggest, a quadtree has four child nodes per parent while a binary tree only has two child nodes per parent node. The representation of a quadtree structure naturally lends itself to represent information on a 2-D plane, such as an image or map.

Figures 3.4 and 3.5 show the same quadtree represented as nodes and also as a map. The quadtree has its root node which has four child nodes. The root node's child node one and child node four also each have four child nodes. Each time a set of child nodes is added to the quadtree structure in the node case, it has the effect of subdividing the map area represented by that node into four smaller areas.

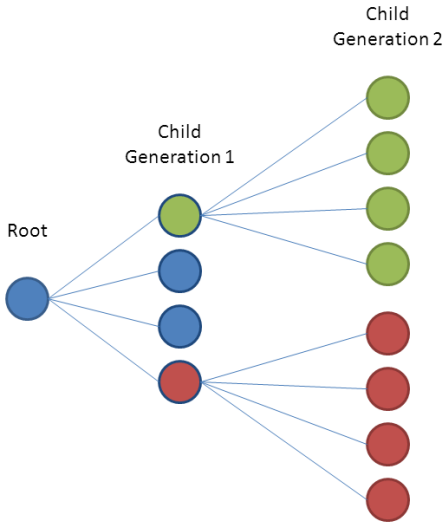


Figure 3.4: Quadtrees can be represented in a tree structure. At the head of the quadtree is a root node. Each node can have four child nodes.

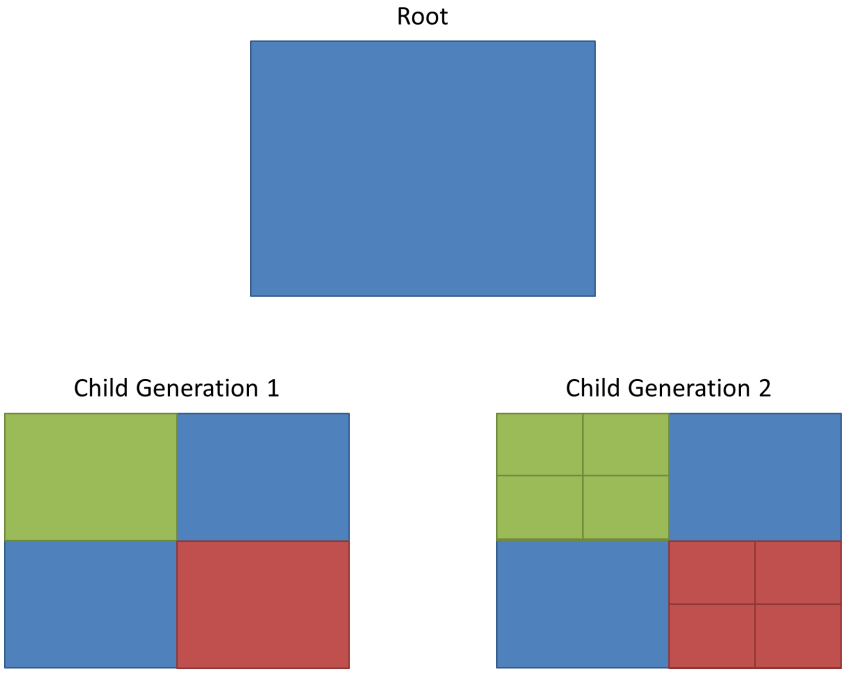


Figure 3.5: Quadtrees can also be represented graphically as a map. A four sided space can be split into four smaller spaces, each with the same ratio of their sides.

3.3 Map Generation

3.3.1 Sensor Input Processing

Processing Method

The approach taken in this research requires a range measurement from the aircraft to an obstacle. Associated with this range must be a bearing estimate. This measurement can be obtained through many different methods. The range information can be returned as a single value, as might be expected in the case of a fixed laser range finder. It can be returned in a vector form, as would be expected from a scanning laser range finder. It can also be returned as an array, as might be expected from the use of vision. For mapping based on a vector or array of sensor readings, each point is individually processed.

Sensor Model Uncertainty

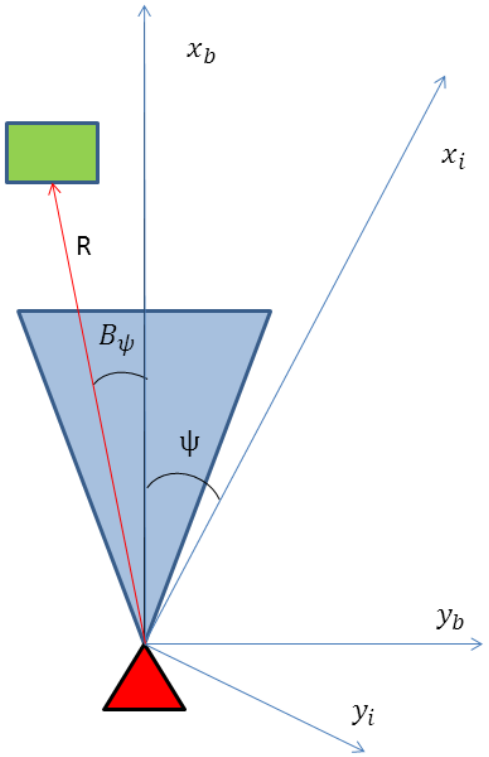
All measurements have uncertainty associated with them. This research is meant to be used in conjunction with depth sensing by vision. Because of this, the greater the distance between the sensor and the object being sensed, the less accurate the range measurement. A sensor uncertainty model that varies linearly with distance to target is used.

In this work, a maximum sensing range R_{max} is assumed. The uncertainty $R_{\text{uncertain}}$ associated with measurement R_{sensed} is calculated as a linear relationship between the range of the obstacle sensed R_{sensed} and R_{max} using the expression

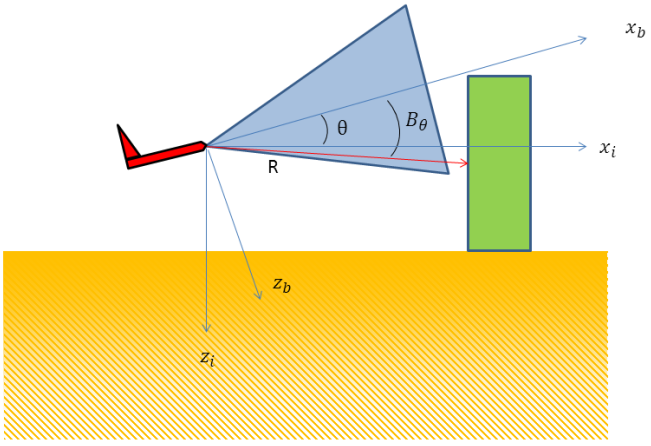
$$R_{\text{uncertain}} = \frac{R_{\text{sensed}}}{R_{\text{max}}}. \tag{3.1}$$

3.3.2 Inertial Frame Transformation

Once a range measurement to an obstacle is obtained from the sensor, the inertial frame coordinate can be calculated. To do so, the range and bearing measurement is used to obtain a location in three space of the point of interest in sensor-frame coordinates. Example images in Figure 3.6 show a top down and side view of the approach discussed next.



(a) Top down view



(b) Side View

Figure 3.6: Sensor measurements are made in the sensor frame. These measurements are first converted to the body frame by $x_b = C_b^s x_s$, then from the body frame into the inertial frame by $x_i = C_i^b x_b$

First the point, as seen in Figure 3.6, with its range R, bearing B_ψ and elevation B_θ, is used to compute the location, p^s, within the sensor frame using

$$p^S = \begin{bmatrix} x^S \\ y^S \\ z^S \end{bmatrix} = R \begin{bmatrix} \cos(B_\theta)\cos(B_\psi) \\ \sin(B_\psi) \\ \sin(B_\theta)\cos(B_\psi) \end{bmatrix}. \tag{3.2}$$

It is then rotated into the body frame from the sensor frame using the direction cosine matrix C_s^b by

$$p^B = C_S^B p^S. \tag{3.3}$$

It is then rotated into the local inertial frame using the attitude of the aircraft ϕ, θ, ψ to generate the transformation matrix C_B^I to rotate it to

$$p^I = C_B^I p^B. \tag{3.4}$$

An expanded form of the transformation matrix for each Euler angle is represented as

$$p^I = C_{roll} C_{pitch} C_{yaw} p^B. \tag{3.5}$$

The vector is then translated using coordinate information T_{gps} gained from GPS to find the real world location of the measurement point returned from the sensor. This can be shown in an expanded form as

$$\begin{bmatrix} x^I \\ y^I \\ z^I \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_\phi & s_\phi \\ 0 & -s_\phi & c_\phi \end{bmatrix} \begin{bmatrix} c_\theta & 0 & s_\theta \\ 0 & 1 & 0 \\ -s_\theta & 0 & c_\theta \end{bmatrix} \begin{bmatrix} c_\psi & s_\psi & 0 \\ -s_\psi & c_\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x^b \\ y^b \\ z^b \end{bmatrix} + T_{gps} \tag{3.6}$$

where $s_x \triangleq \sin(x)$ and $c_x \triangleq \cos(x)$. This calculated point, (x^I, y^I, z^I) can then be used in building the map.

3.3.3 Map Update Routine

Quadtree Node Generation

The map, in its entirety, can be thought of as the root node. Each non-root node within a quadtree will always be associated with a parent node. However, each node will not necessarily have child nodes. When using a quadtree for representing a map or image, creating child nodes for a given node can be thought of as splitting the node. It means that the space is represented by four nodes instead of the one that was used previously. The parent node of the newly created four child nodes is no longer seen in the map, but is used for relationally associating the new child nodes with the rest of the tree. For a quadtree representing a map or image, it is child-less nodes which are used to describe the area.

When a higher resolution is desired, the node is given four child nodes. This corresponds to the whole map being split into four subsections. In turn, if more information about an area is desired the node representing that area can be given four child nodes, and the resulting map has that specific area split into four subsections. For example in Figure 3.7, there is a point of interest to build up a quadtree around. Each successive step creates a higher resolution branch of the quadtree around the point of interest.

With this method, the map can represent obstacles with the resolution desired. When a new set of child nodes is generated, $R_{\text{uncertain}}$ is stored in the node which contains the measured location p.

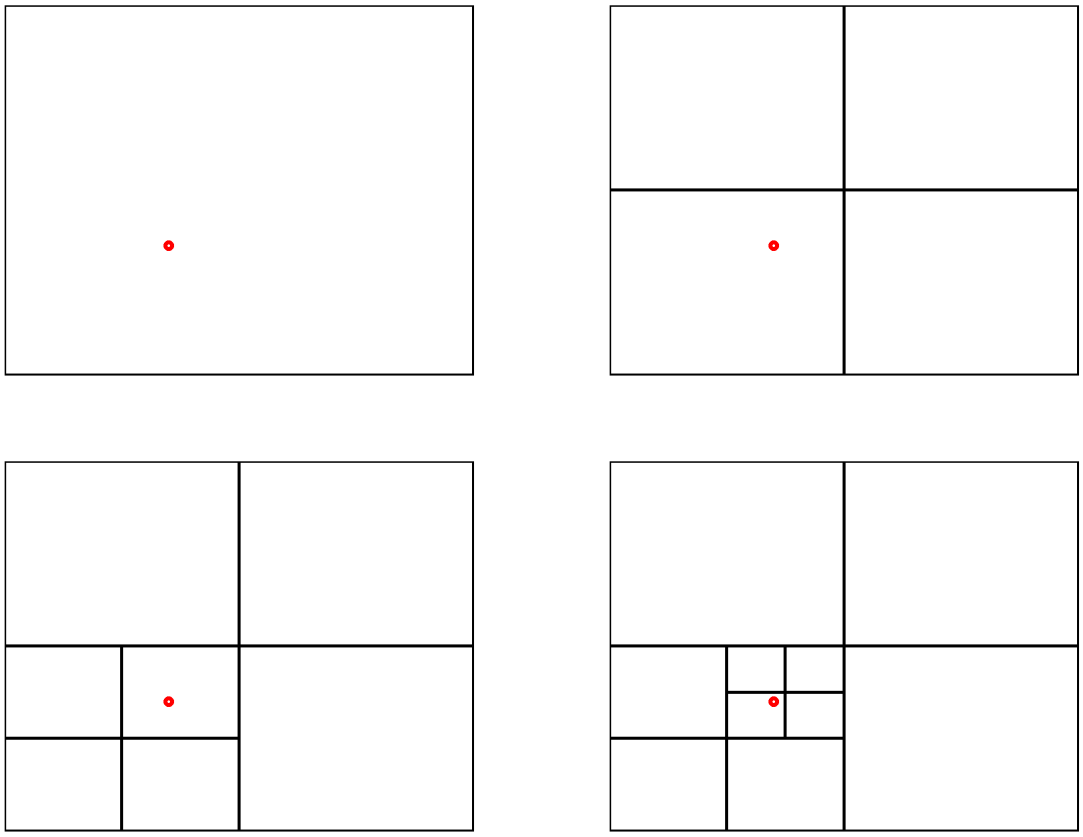


Figure 3.7: When given a point of interest, a quadtree can increase in resolution around the point. As it does so, it continues to split the node containing the point of interest into child nodes.

Locating a Node

Given a sensed object point p, Algorithm 3.1 is used to find the child node corresponding to the highest resolution map node Γ which contains the location of p through a recursive search. Beginning with the root node, the algorithm identifies if the current node has children. If the current node does have children, the algorithm identifies which child contains the measured location. The child node, a higher resolution node than the parent, then replaces the parent node as the current node. This continues until the chosen node has no children. Having located the highest resolution quadtree node containing the point of interest, the node, Γ, is then returned.

Algorithm 3.1 findFinalChildNode(Map,p_{obj})

$while(node_{curr}.hasChild == true)$
{
    $node_{child}$ = FIND_CHILD(p_{obj});
    $node_{curr} = node_{child}$
}
return $node_{curr}$

Measurement Placement

Once a measurement uncertainty $R_{\text{uncertain}}$ is determined, the inertial frame location of the sensed object p calculated, and the node Γ containing p is found, the algorithm can determine if map resolution can be increased via node placement.

The uncertainty $\Gamma_{\text{uncertain}}$ of the the map node Γ is then compared to the uncertainty of the measurement $R_{\text{uncertain}}$. If $\Gamma_{\text{uncertain}}$ is less than $R_{\text{uncertain}}$, and thereby higher resolution than the measurement would provide, nothing is done and the next measurement is processed. However if $R_{\text{uncertain}}$ is less than $\Gamma_{\text{uncertain}}$, the measurement contains a lower uncertainty than map node Γ. The algorithm then recursively generates new nodes Γ_n until the uncertainty associated $\Gamma_{n_{\text{uncertain}}}$ is less than $R_{\text{uncertain}}$ following the node generation method outlined in Section 3.3.3.

3.4 Chapter Summary

Knowledge of an area is essential for a UAV to fly through it without contacting any obstacles. There are many different methods of creating a map to use for this purpose. The quadtree implementation of a multi-resolution map requires less storage space than a standard resolution map would for a more sparsely populated area.

The map is populated by use of range and bearing measurements. The measurements are used to calculate a point in map frame by rotating from the sensor coordinate frame and utilizing known information about the location measurement of the sensor platform. Based on the sensor reading, the uncertainty of the range location can be used to set the resolution of the map at a given location.

CHAPTER 4. PATH SEARCHING

4.1 Introduction

Once a map of the search space containing any known obstacles is developed, the next step is to determine a safe path to the goal location. UAV dynamics preclude the UAV from stopping and waiting until a planning algorithm completes execution. Any path re-planning must be done quickly while the UAV continues its flight. Many path planning algorithms require more time than a UAV would have available to find or re-plan an optimal path. Computation time for many planners increases as more obstacles are added. This chapter will focus on the reasoning behind selecting a rapidly-exploring random tree (RRT) algorithm for path planning as well as the method by which paths are planned and updated.

This chapter is will discuss the planning portion of the re-planning path planner. It is one of the three important steps as seen in Figure 4.1

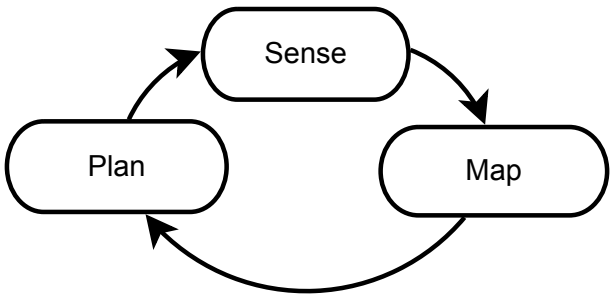


Figure 4.1: Relationship of planning with mapping and sensing

4.2 Background

As UAVs have become more common, research has been done to design optimal paths for multiple UAV teams, for UAVs in areas of potential danger, and for optimal mission effi-

ciency. Many methods of planning have been used, including genetic algorithms, graph-based search methods, and potential fields.

One of the big challenges faced by path planning algorithms is the need to search the entire area of interest for feasible paths. Doing this in an efficient manner is difficult. Several different methodologies have been proposed. Yahja, et al. have produced work combining both quadtrees and path planning [7]. For path planning through a multi-resolution map environment they use a D* algorithm to find and then update the planned path. However, the D* algorithm requires a search of the entire area for the algorithm to find the path. The computation time also increases significantly with increased complexity of the area to be searched. The time required to run these computations can also be significant in terms of the dynamic behavior of a UAV system. They were able to successfully implement it using a ground robot, covering 200 meters in 6 minutes. This would be too slow to implement on a UAV in flight. One of the difficulties with implementing this in a fixed-resolution setting would be the need to visit each node of the map to produce the path.

Randomized potential fields is a method which also produces results in a quick and efficient manner and are used by Barraquand and Latombe in [15]. The downside in using a randomized potential field approach is the use of a random walk. The random walk is used to help find path segments around local minima. However, it is difficult to implement in such a manner that guarantees performance because of the tendency of random walks to search the same area repeatedly. Resulting paths from such an algorithm would typically not be flyable for a UAV.

Path planning can generally be broken into two main categories, single-query planning and multiple-query planning. Single-query planning is more suitable in instances where a given map will be searched only once. This type of path searching does not generally benefit from precomputation. Multiple-query planning utilizes precomputation to build a search graph space which can then be more quickly searched by the planner. However, due to the changing nature of the map and the need to re-plan paths only after the map has changed, a multiple-query approach is not beneficial.

Probabilistic roadmaps were utilized to search for paths by Kavraki, et al. in their work [16]. Though generally a multiple-query approach, they are still usable in a single-query application. To find a path, a random sample is taken from the configuration space and checked to see if it

is unobstructed. It then tries to connect these samples with a local planner. The downside with this approach is the computational overhead associated with using these for a single-query approach.

Rapidly-exploring random trees (RRT) is a method of exploration which biases the search towards areas not previously searched. LaValle first proposed this solution in [17]. RRT searching is a single-query planning method. It combines many of the strengths of both randomized potential fields and probabilistic roadmaps. Probabilistic roadmaps are stable and the probability of not finding a feasible path approaches zero as the search time increases.

In an effort to make RRTs applicable for UAVs, Griffiths, et al. proposed the implementation of a waypoint based RRT in [18]. In their work, they showed how the implementation of waypoints could account for disturbances due to wind. Pruning paths found by RRTs is an important element in creating UAV flyable paths. Hansen, et al. discuss the use of a greedy path refining method in their work [19]. They explain that the use of a greedy path refining methodology leads to efficiency, though not necessarily completeness, in finding the shortest path.

4.3 Rapidly-Exploring Random Trees

Once a suitable map has been established, the process of searching for a viable path can begin. Path planning methods come in many different varieties, all with their own unique strengths and weaknesses. Some have the ability to guarantee an optimal path. Some can compute paths quickly. Others can account for kinematic constraints of the system. A UAV in mid-flight has its own unique set of requirements for a path planner. Primarily, the path planner must be quick and efficient. It also must be able to plan through a changing map area with a potentially high number of obstacles. A proven path planning method which is quick and computationally independent of the number of obstacles is the RRT.

One of the advantages of RRTs is the ability to create paths through higher dimensional spaces such as including attitude in addition to position. However, certain constraints on turn radius and velocity exist that limit the search space. To have robust path planning, the vehicle needs to be able to follow the path found. In addition to searching in a two-dimensional plane, the RRT path planner can also search through aircraft states as well. As the tree is extended, it connects the states that are found to the nearest state in a manner which obeys these constraints.

One challenge with this approach is that the new path is now a series of command inputs given to the UAV. This would work in an ideal world with no disturbances such as wind or atmospheric perturbations provided the UAV was perfect in tracking the course given. However, in a real-world situation, such an approach will not be able to follow the desired path because of disturbances. In lieu of such an approach, waypoint following can be used. In this method, the UAV is commanded to follow a series of points. Between the points, the UAV has a control algorithm which helps it fly level at user-defined speed and altitude. If wind or gusts perturb the UAV off the flight path, the onboard controller will push the UAV back onto the path.

4.3.1 RRT Algorithm

One of the primary characteristics of RRTs is its ability to expand into areas not previously explored. By doing so, the probability of a path being found increases. One of the unique features of an RRT is that regardless of the number of obstacles on the map, the amount of time the algorithm takes to find a path increases minimally with each new obstacle. Many algorithms focus on each obstacle and are required to compute a path in relation to each one. Most path planners of this variety are of order n^2, meaning that computation time increases exponentially with each new obstacle.

As with any path planner it is desired to have a path between two states, x_{init} and x_{goal}. The path should be continuous and lie wholly within the space of interest. This space is a standard configuration space for a rigid two or three dimensional body, C. As the object of this path is to avoid obstacles, it should only pass through states which are free of obstacles $C_{\text{free}} \subset C$. The RRT creates a path which traverses the space between x_{init} and x_{goal} in such a manner as to stay within C_{free}. Included here is a description of the basic RRT algorithm found in [17].

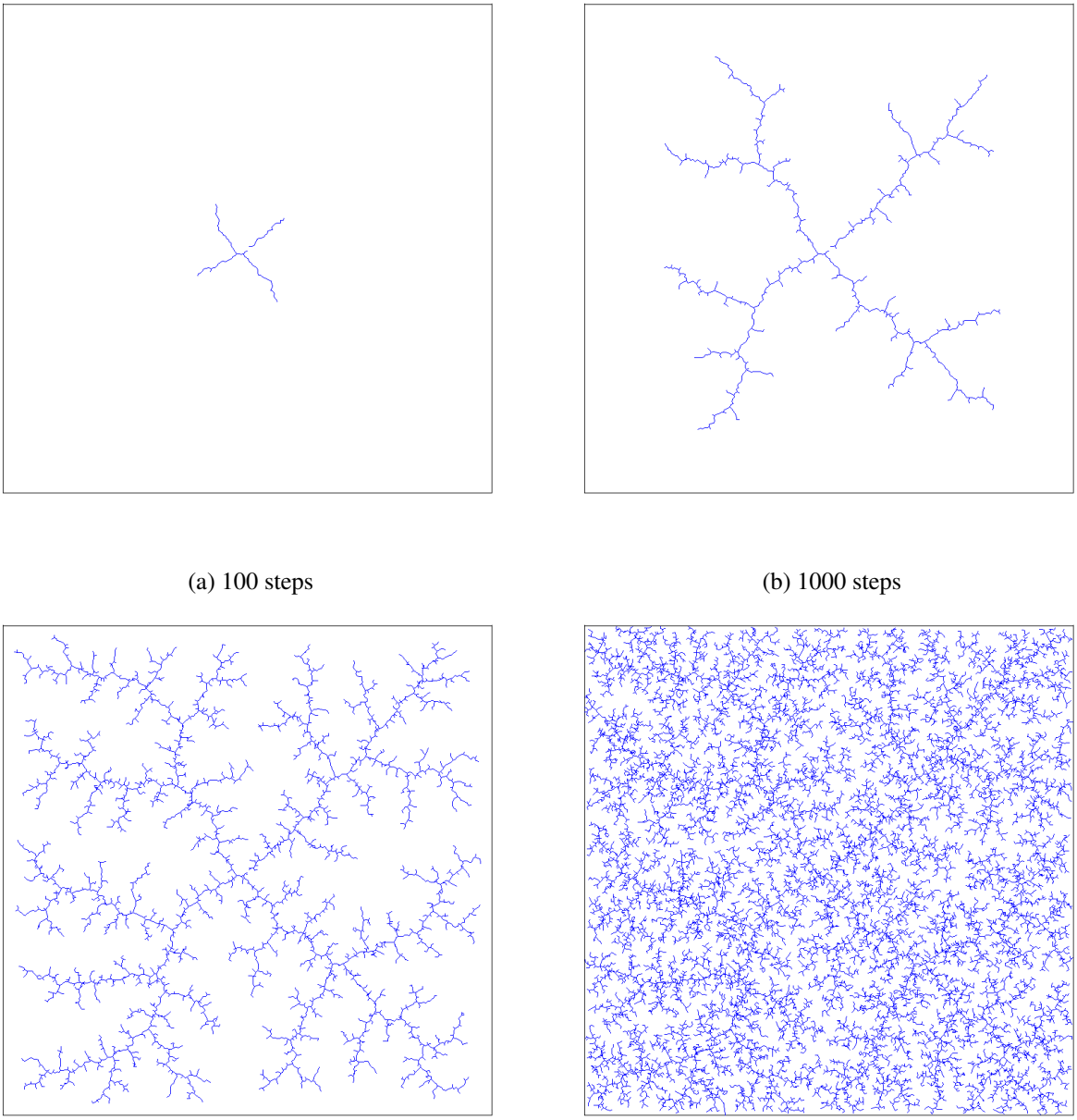
Algorithm 4.1 generate_rrt(x_{init}, K, Δt)

$\tau.init(x_{init})$;
for $k = 1$ **to** K **do**
    $x_{rand} \leftarrow$RANDOM_STATE();
    $x_{near} \leftarrow$NEAREST_NEIGHBOR(x_{rand}, τ);
    $u \leftarrow$SELECT_INPUT(x_{rand}, x_{near});
    $x_{near} \leftarrow$NEW_STATE(x_{near}, u, Δt);
    τ.add_vertex(x_{new});
    τ.add_edge(x_{near}, x_{new}, u);
end

In this algorithm K is the number of vertices to be constructed by the algorithm. To connect states, the state transition equation of $\dot{x} = f(x,u)$ is used, with the vector u selected from a set of inputs. A state x_{new} can be found by integrating $f(x,u)$ over a discrete interval Δt. The tree τ to be constructed starts with point x_{init}. A random point x_{rand} is chosen from C_{free} . The nearest neighbor x_{near} to x_{rand} is located from the tree τ. A control input u which minimizes the distance from $x_{\text{near}}$to $x_{\text{rand}}$is then chosen. A new state x_{new} is then calculated from control input u being applied to state x_{near} for Δt. Once this new state is found, it is added to the tree τ along with the input connecting x_{new} and x_{near}. After K iterations, the tree is returned. An example of a growing RRT can be seen in Figure 4.2. As the number of steps that the algorithm takes increases, it continues to search out into previously unsearched areas. By 20,000 steps in this example, the RRT has searched out almost all available space. The rate at which the RRT covers the space depends on step length and search area size.

4.3.2 Modified RRT for UAVs

RRT generated paths can be defined with as many, or as few, states as are desired. This means that the states can describe not only the position of the aircraft, but also the orientation and velocity. These paths can be created while maintaining dynamic constraints that the UAV might have. However, the paths will not be able to account for disturbances and imperfections in the system. Because of this, any path formed which relies on the ability to perfectly fly the path will result in the aircraft not reaching the destination. The aircraft also has no guarantee that the path will continue to lie in C_{free}.



(a) 100 steps

(b) 1000 steps

(c) 5000 steps

(d) 20,000 steps

Figure 4.2: As an RRT grows, it adds to points previously found. The nature of the RRT forces search into previously unsearched areas as seen when comparing the subfigures.

To account for possible disturbances or flaws in the dynamic model of the system, a method of waypoints has been developed. Waypoints are goal locations for the aircraft to reach that will guide it to its destination. The path of the UAV will be connected by these waypoints. The path

following of the Kestrel autopilot uses a vector field approach. In his paper [20], Nelson details the method. Vector field path following gives the UAV a commanded heading which quickly forces the UAV to the path and helps keep deviation from the path small, even in the presence of large disturbances. By only requiring the UAV to fly between calculated waypoints instead of attempting to fly exact states and control inputs, the UAV can rely on its path following ability to reject disturbances and imperfections. This is important to ensure reliability of the system.

When computing a tree τ using two-dimensional waypoints, the methodology changes slightly. When a random state x_{rand} is generated and x_{near} is found, the RRT generates a new point x_{new} which is located a predetermined distance d along the line connecting x_{rand} and x_{near}. This state x_{new} is then added to the tree τ and the algorithm continues.

4.3.3 Modified RRT for Quadtrees

Quadtree RRT Approach

In a quadtree setting, the RRT algorithm can be used as described in Section 4.3.1. However, when planning a path through an area represented by a quadtree, characteristics of the quadtree can be leveraged to decrease the calculation time required for path planning. Quadtrees make it possible to represent large areas with a single cell. Each cell in the quadtree contains information about whether or not the area described by that cell contains an obstacle. If that area contains an obstacle, planning through that area should be avoided. However, if the area is free of obstacles, the entire distance across it can be traversed without hitting an obstacle.

Implementing this in the RRT algorithm only requires a small change to the RRT algorithm described in Section 4.3.1. The quadtree τ is initialized with with a starting location x_{init} , as shown in line 1 of Algorithm 4.2. On line 4, a random state x_{rand} is generated from C_{free}. Next on line 5, x_{near}, the nearest node currently on the tree τ, is found. On line 6, d_{step}, the distance to cross the cell of the quadtree between x_{rand} and x_{near} is calculated. On line 7, a new point x_{new} is then generated a distance d_{step} from x_{near} along the line between x_{rand} and x_{near}. This state x_{new} is then added to the tree τ via x_{near} on line 8 and the algorithm continues. Once the RRT nodes x_{new} and x_{goal} can be connected by a step across a single cell of the quadtree, the algorithm adds a step between x_{new} and x_{goal}, terminates, and returns the tree τ.

A comparison of paths found the regular RRT and quadtree RRT algorithms using the same example environment can be seen in Figures 4.3 and 4.4. It is important to note that the two paths take roughly the same path between the start and end locations. However, the quadtree RRT takes fewer steps because of its ability to cover greater distance due to its knowledge of empty space in the environment.

Algorithm 4.2 generate_quadtree_rrt(x_{init}, K, Δt)

1. $\tau.init(x_{\text{init}})$;
2. path_complete = FALSE;
3. **while** (path_complete==FALSE)
4. $x_{\text{rand}} \leftarrow$random_state();
5. $x_{\text{near}} \leftarrow$nearest_neighbor($x_{\text{near}}$, τ);
6. $d_{\text{step}} \leftarrow$find_distance($x_{\text{rand}}$, x_{near});
7. $x_{\text{new}} \leftarrow$new_state($x_{\text{rand}}$, x_{near}, d_{step});
8. τ.add_node(x_{new}, x_{near});
9. **if** (adjacent(x_{new}, x_{goal}))
10. τ.add_node(x_{goal}, x_{new});
11. path_complete = TRUE;
12. **end**
13. **end**

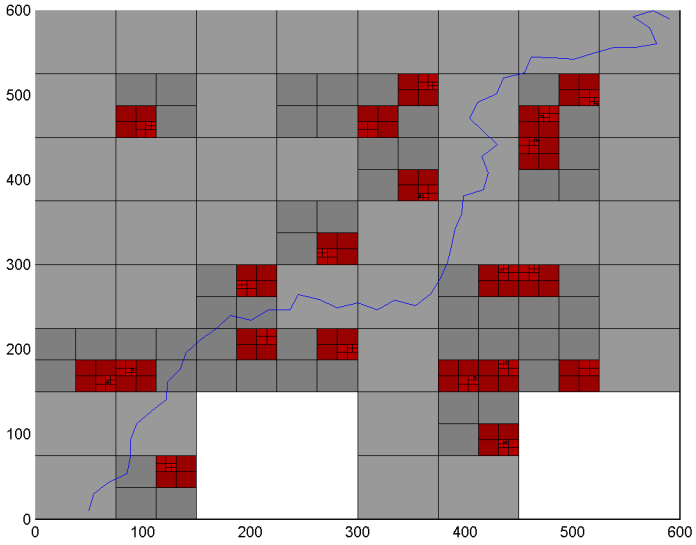


Figure 4.3: RRT found using the regular RRT algorithm through the same example environment as Figure 4.4. The areas in red were considered obstacles and the path was not allowed to pass through those areas.

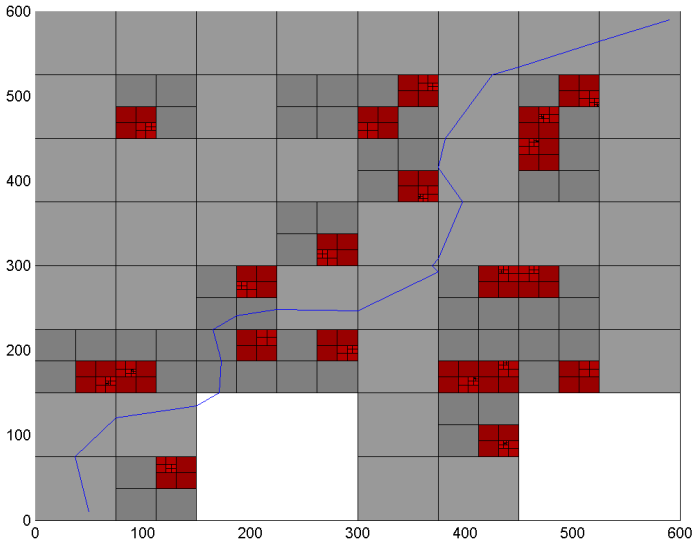


Figure 4.4: RRT path found by quadtree RRT algorithm through the same environment as shown in Figure 4.3. It can be seen in comparing the two that the RRT steps in the quadtreen example step between the edges of the quadtree while the regular RRT path does not take advantage of knowledge of open space.

Quadtree RRT Summary

The quadtree RRT is similar to the standard RRT algorithm. However, it takes advantage of the characteristics of the quadtree. It uses an adaptive leg length to traverse the cells of a quadtree more efficiently. A result of this usage is that the RRT nodes will all be located on the edges of the quadtree cells. The quadtree RRT is a novel approach which is used in this research.

4.4 Path Planning Implementation

In planning paths, there are two scenarios underwhich paths may need to be re-planned. The first is planning a path between two waypoint locations. The other is planning a path between the current UAV location and the next waypoint.

There is a need for a distinction between these two scenarios because when planning between the two waypoint locations, the path is not dependent on the current UAV location. This can lead to the path being such that the next waypoint might be positioned on the opposite side of an obstacle, forcing the UAV to fly into the obstacles unless a reactive avoidance algorithm is used. It can also lead to the UAV flying a very non-efficient path in order to rejoin the path.

Consequently there is a need for a second path which goes from the current UAV location to the next main waypoint which allows the UAV to avoid potential obstacles. This special type of path is referred to as the single-pass path, as it is only to be flown once. Upon arriving at the next main waypoint, the UAV can rejoin the waypoint path. This planning of this single-pass path is dependent upon the heading of the UAV. This avoids major deviations of heading which might cause the UAV to lose time and distance to maneuver.

4.4.1 Refining Paths

Once a path is initially found, the aircraft can continue to fly the path until new information about obstacles emerges. The path of the aircraft then needs to be rechecked for obstructions as outlined in Algorithm 4.3. The algorithm searches through each leg $path_{\text{leg}_i}$ in the path. If the path is found to pass through areas now considered to be unsuitable for flight because of obstacles, on line 3 the path planner plans a new RRT route through free space between the leg endpoints. On

line 4, the path is then pruned as described in Section 4.4.1and smoothed on line 5 with a greedy algorithm as described in Section 4.4.1.

Algorithm 4.3 checkPath($path_{\text{leg}}$)

1. **for** $i = 1$ to K
2. **if** ($path_{\text{leg}_i}$!= clear)
3. $path_{\text{newleg}}$ =findRRT($path_{\text{leg}_i}$)
4. prunePath($path_{\text{newleg}}$)
5. $path_{\text{leg}_i}$ =smoothPath($path_{\text{newleg}}$)
6. **end**
7. **end**

Path Pruning

Path pruning refers to the extraction of the path from the tree found in the RRT algorithm. The RRT algorithm terminates once a node x_k is added to a path that is contained in the same quadtree node which contains x_{goal}. A path pruning algorithm then traverses τ starting from x_{goal} and going backwards through the tree τ until x_{init} is reached. This is the path which will be integrated into the flight path of the aircraft. Figure 4.5 shows a regular RRT which is unpruned while Figure 4.6 shows the pruned path which has been extracted from the tree.

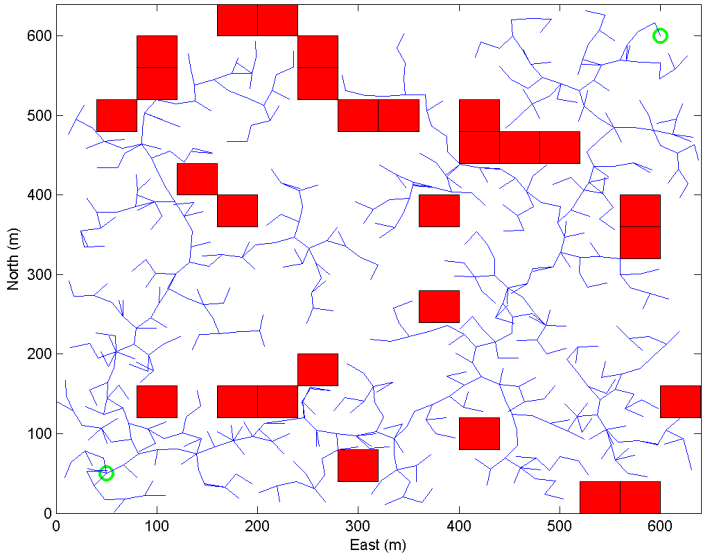


Figure 4.5: Example of unprunned RRT tree. While a branch of the tree has reached the end location, the path must be extracted. The resulting path can seen in Figure 4.6.

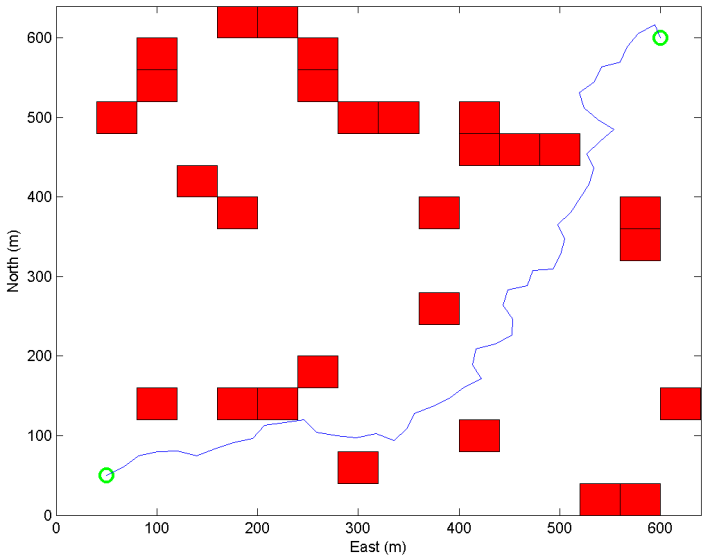


Figure 4.6: Example of pruned RRT path. This is the path extracted from the RRT after the RRT reaches the goal location. The unpruned version can be seen in Figure 4.5.

Path Smoothing

Due to the nature of the RRT, the path found by the RRT may not take the most direct route to the goal. Because of this, a path smoothing routine can significantly decrease the amount of distance the path takes to arrive at the goal location. For this research, a greedy algorithm was chosen due to its ability to smooth the path quickly.

A greedy algorithm, like RRTs, does not guarantee globally optimal results. However, the greedy algorithm is a fast, efficient algorithm which generally yields good results. The greedy algorithm, as outlined in Algorithm 4.4, chooses the locally optimal solution. In the case of the path smoothing function, it is the shortest path from the location currently being evaluated.

Algorithm 4.4 smooth_path(*pathLeg*)

1. $pathLeg_{\text{disc}}$ =discretizePath(*pathLeg*)
2. $n = numPoints$ in $pathLeg_{\text{disc}}$
3. **for** $i = n$ to 1
4. **for** $j = 1$ to $i-2$
5. **if** checkPathIsClear($pathLeg_{\text{disc}_i}$,$pathLeg_{\text{disc}_j}$) $= true$
6. *newPath* =shortenPath($pathLeg_{\text{disc}}$, i, j)
7. restart algorithm with *newPath*
8. **else continue**
9. **end**
10. **end**
11. **end**

The greedy algorithm starts with a smoothed path as seen in an example in Figure 4.7. It discretizes the path into lengths of a specified size d by inserting waypoints between segments as shown in Figure 4.8. The algorithm then begins checking between waypoints to find a shorter path, starting with the destination waypoint. For this example, it is assumed the discretized path

contains n waypoints. The greedy algorithm starts with the last waypoint, x_n, and checks to see if the space between it and the first waypoint, x_1 is free of obstacles as seen in Figure 4.9. It then checks to see if the path is clear between waypoint $x_n$and waypoint x_2 on line 5. It continues until it checks between waypoint x_n and waypoint x_{n-2}. If a clear path is found between waypoints x_i and x_j as shown in line 6, those waypoints are connected, the intermediate waypoints between x_i and x_m are discarded. On line 7, the path is rediscretized, and the process begins again, starting at the x_{n-2} waypoint. This algorithm continues until the path is entirely checked. Any remaining collinear waypoints are then deleted. The result of this process can be seen in Figure 4.10.

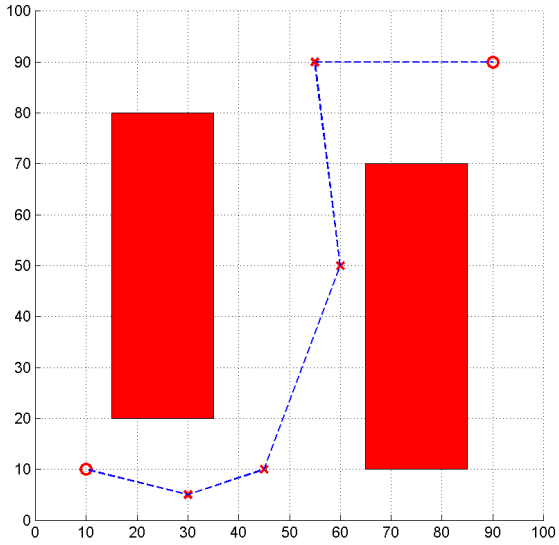


Figure 4.7: Path Smoothing Example: Pruned, unsmoothed path

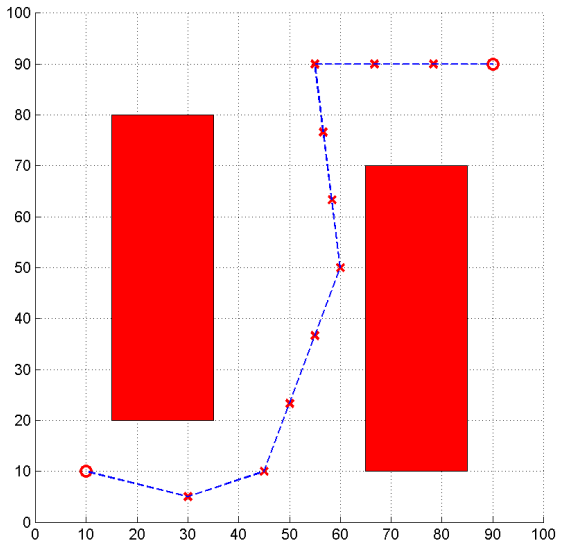


Figure 4.8: Path Smoothing Example: Discretized Path

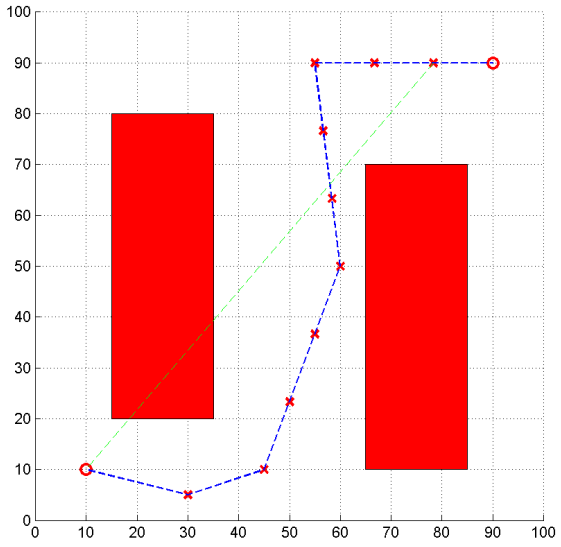


Figure 4.9: Path Smoothing Example: Illustration of checking to shorten path

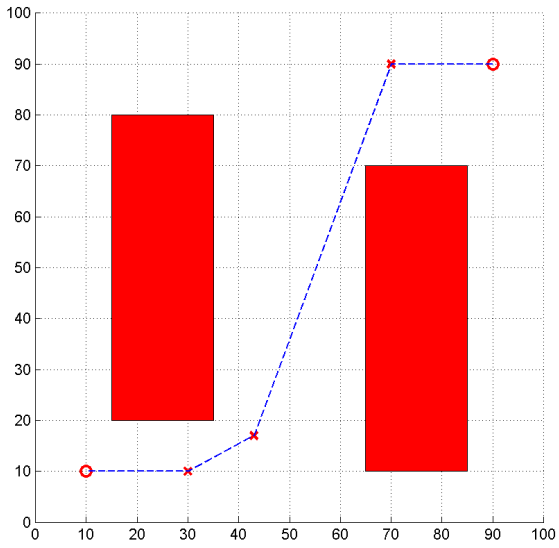


Figure 4.10: Path Smoothing Example: Resultant, shortened path

Traversing the Path

During path smoothing, it is necessary to identify the map cells that a segment of the path crosses. This is done by first identifying the location where the path exits the current cell. Then a new point is found on the path segment a short distance past the first location. The cell containing this point is identified and checked for obstacles. The algorithm then procedes to the next cell boundary and the process is repeated. This continues until the destination point is reached.

4.5 Chapter Summary

Path planning is a process which can be accomplished in many different ways. This research uses a modified RRT approach to plan the path. RRTs have the advantage over many other path planners in that they are computationally fast. They are also efficient when searching an area with large numbers of obstacles. Instead of planning exact paths, a series of waypoints is created, which is flown using the vector field path planning developed previously. The results illustrate the strength of the quadtree RRT when compared to a standard RRT planner. While other mechanisms can be implemented to speed up the standard RRT, many of these could also be applied to the

quadtree RRT. Consequently, a straightforward implementation was used to show the advantage of the quadtree RRT.

CHAPTER 5. SIMULATION RESULTS

5.1 Introduction

This chapter details the experimentation performed for this research. To test our algorithm, and to evaluate its performance compared to existing replanning algorithms and methods, we conducted numerous simulations under a variety of conditions. In this chapter, we present three illustrative simulation scenarios, and discuss the performance of our algorithm under the unique conditions represented by each. We introduce the scenarios in Section 5.2, then present the metrics that will be used for evaluating performance in Section 5.3. Further scenario information is available in Appendix B. Planning results from each of the scenarios are then presented and discussed in Section 5.4.

5.2 Simulation Setup

The research outlined previously generates an obstacle map based upon sensor inputs. As the obstacle map is generated, the path through the obstacle field is checked for collisions. If the path is compromised, a new path is found.

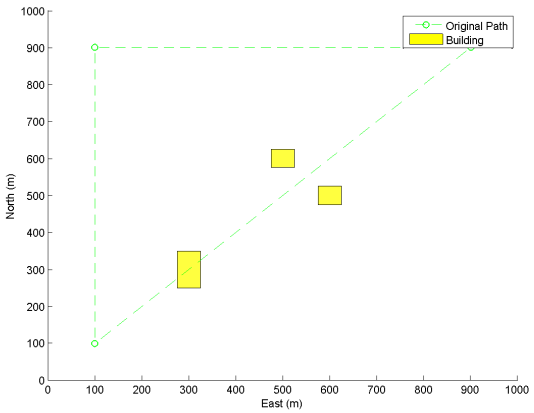
As mentioned in Chapter 2, the simulation environment is coded in MATLAB and runs a control library compiled from C code which runs the autopilot. This is connected to a six-degree-of-freedom model which emulates the dynamics of the aircraft in flight.

For these experiments a 1000 meter by 1000 meter environment was created. Within that environment a series of buildings were distributed. These buildings ranged in size between 50 and 100 meters per side. For the purposes of these experiments, the altitude of all the waypoints is 50 meters. All buildings are 100 meters tall.

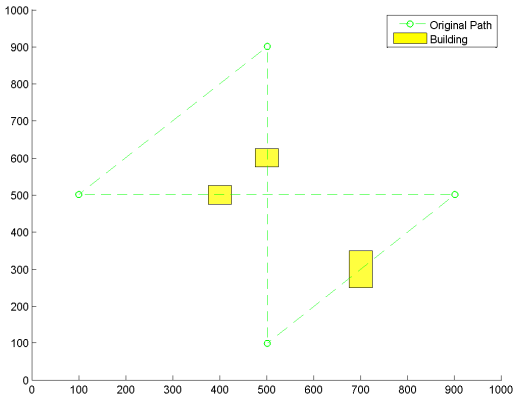
For this experiment, a range and bearing sensor reading, which will be referred to as synthetic vision, is created. Synthetic vision mimics a real video stream by returning a series of frames

as described in Section 2.3.2. These frames consist of an array of range values. Synthetic vision is created by calculating where rays along the lines of the virtual pixels intersect the closest building. The synthetic vision resolution for these experiment will be 10 pixels by 10 pixels.

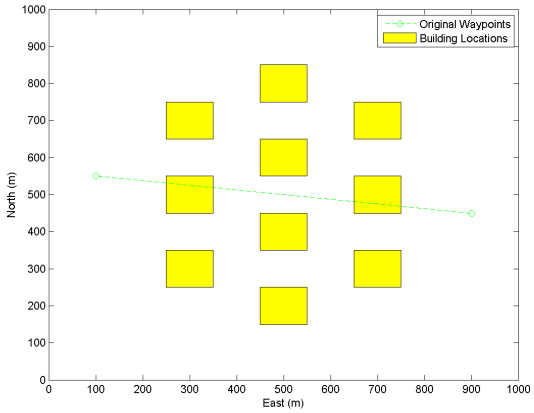
Three scenarios were chosen to test the algorithms of this research as representative examples of performance and behavior. A map of the scenario with the waypoints and buildings is included. The waypoint and building locations are given in tabular form in Appendix B.



(a) Scenario 1



(b) Scenario 2



(c) Scenario 3

Figure 5.1: Scenario maps used in testing the simulation

5.2.1 Scenario 1

Scenario 1 is a triangle shaped route which passed through three buildings on one of the legs as seen in Figure 5.1a. This scenario tests how well the algorithm works along a single leg which passed diagonally in relation to the buildings. The waypoint locations are provided in Table B.1 and the building locations for scenario 1 can be found in Table B.2.

5.2.2 Scenario 2

Scenario 2 is a more complicated scenario with buildings located along three of the four legs of the original path as seen in Figure 5.1b. This scenario tests how the algorithm handles the placement of buildings that are both perpendicular and diagonal to the planned path. The waypoints can be found listed in Table B.3 and the buildings are listed in Table B.4. The legs of this path are numbered according to Figure 5.2.

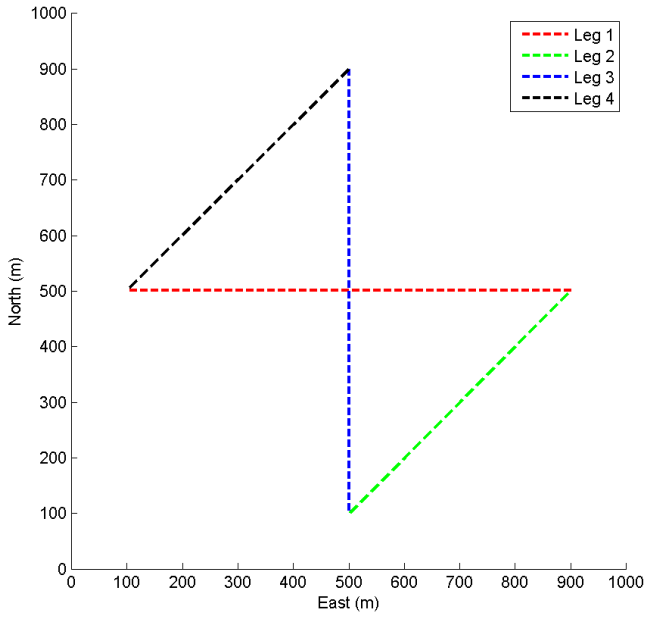


Figure 5.2: Identification of initial path legs for scenario 2. Each of these legs presents a different approach angle to obstacles. These legs are labeled for convience when reviewing the results that follow.

5.2.3 Scenario 3

Scenario 3 is a straight line path across the map. The map includes ten regularly-placed buildings, as seen in Figure 5.1c. This scenario tests the behavior and ability of the algorithm to navigate through multiple layers of buildings. The waypoints can be seen in Table B.5 while the building locations can be seen in Table B.6.

5.3 Result Metrics

5.3.1 Bounded Optimum

One of the metrics of how well the path planning algorithm does is a comparison between the new path length and the straight line distance between the waypoints. This comparison can show how the path length changes as map fidelity increases. However, since the comparison is between the straight line distance and the new path, it is not a comparison to any sort of true best case path. Therefore, we introduce a bounded optimum.

The bounded optimum is a best-case scenario for path distance because it assumes perfect knowledge of the building location and also allows the path to come in contact with the buildings. All error from the mapping portion of this research is eliminated. The bounded optimum is found by exhaustively searching for paths between the waypoints in question by branching out, similar to the tree built by the RRT. The algorithm starts at the first waypoint and sends branches to all building corners which are not blocked by other buildings. The end point of each branch then similarly branches out to all other building corners not blocked by other buildings. The branches are not allowed to branch back towards the first waypoint. The branches terminate when they reach the second waypoint. The shortest distance is then found from the branches. An example of the bounded optimum search showing the buildings and the legs searched can be seen in Figure 5.3.

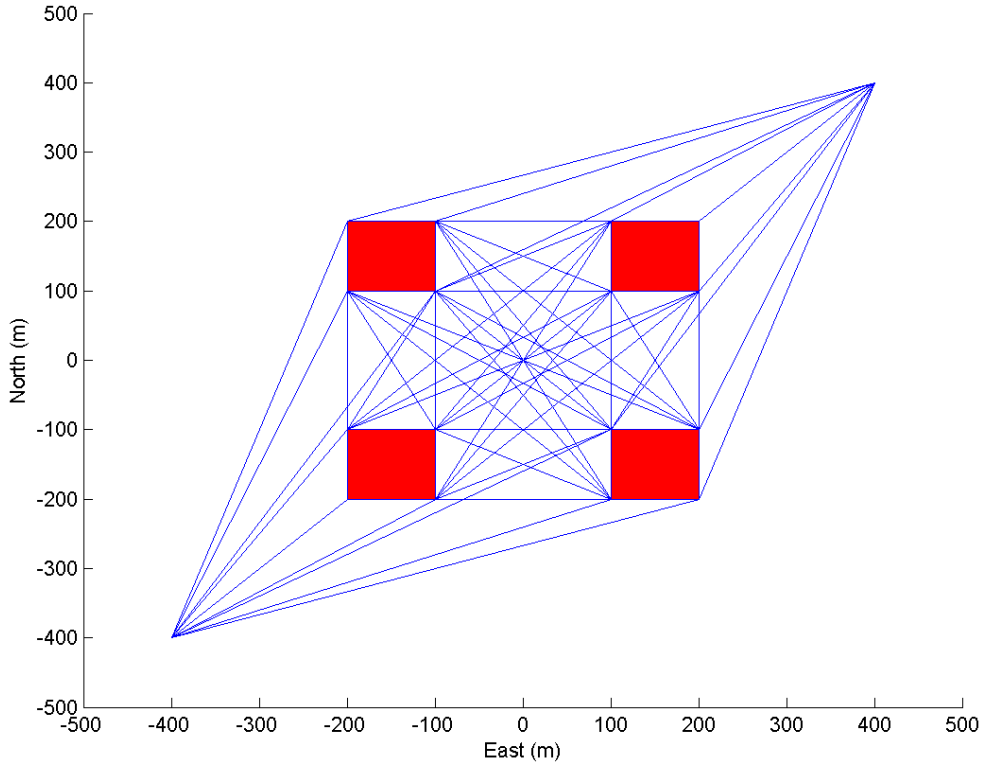


Figure 5.3: Example scenario showing search steps for bounded optimum algorithm. The algorithm searches for valid path segments in the direction of the destination waypoint. It searches between all corners as the bounded optimal path would not change direction in free space.

5.3.2 RRT Comparisons

Another of the metrics considered in this research is the performance of the quadtree RRT compared to the regular RRT. Because Scenario 2 has more legs with obstacles than any other scenario, we will examine the performance there. The performance of the regular RRT is very sensitive. If the step size for the regular RRT is too small, the number of iterations to find a path will increase dramatically. However, if the step size is too large, it can have difficulty in finding paths through highly cluttered environments. For the purposes of this comparison, the regular RRT was chosen to have a step size of 50 meters. This step size is the size of the smallest building in the simulation.

To compare the performance of the regular and quadtree RRTs, two metrics were chosen. The first is the number of iterations it took the algorithm to find a path. The second metric is the final flyable path length of the paths generated by the quadtree and regular RRT algorithms.

The first metric comparing RRT performance was chosen to show effectiveness in finding a path as well as indicate how quickly a path can be found. The time to find a path was not chosen because it is highly dependent on programming implementation of the algorithm and can easily be skewed.

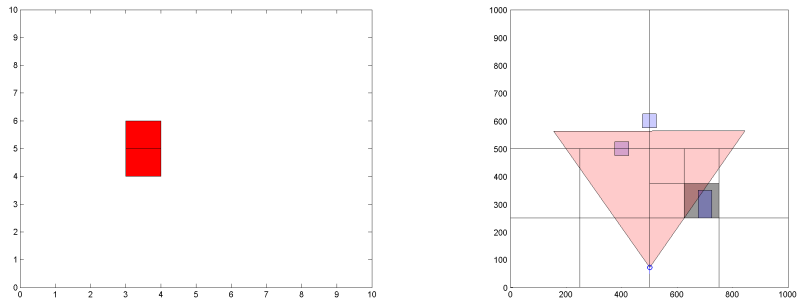
The second metric was chosen to observe if there is a net difference to the UAV when choosing to use the quadtree RRT algorithm over the regular RRT algorithm. If the resulting flyable path from the quadtree RRT is significantly different from the regular RRT it would indicate a strength for one of the algorithms.

5.4 Results

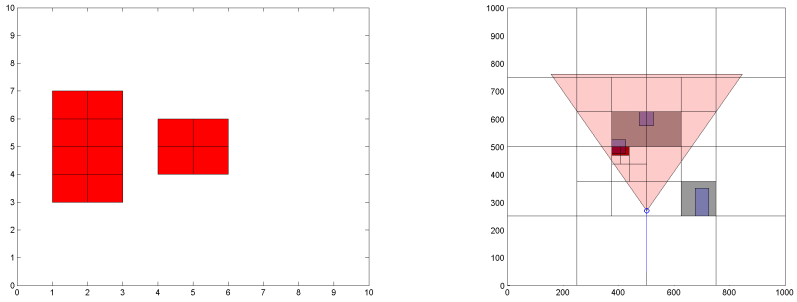
The scenarios were chosen to allow the UAV to approach buildings from different directions and encounter different formations of buildings. The results of these experiments are shown below. Included in these results is a map of the scenario along with the actual path flown. Also shown is a graph showing the different series of waypoints found by the path planning algorithm as the map was updated with new information. Section 5.5 contains an analysis of these results.

5.4.1 Map Evolution

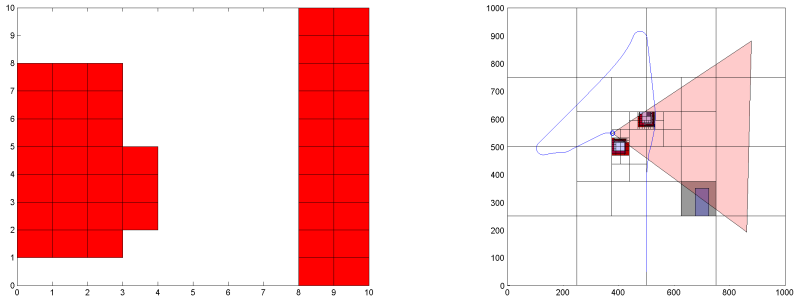
In an effort to assist the reader's understanding of how a map is built up, the images in Figure 5.4 show the synthetic vision view as well as the quadtree map at various times of flight. It can be seen that as the flight goes on, the sides of the buildings visible to the UAV are all mapped.



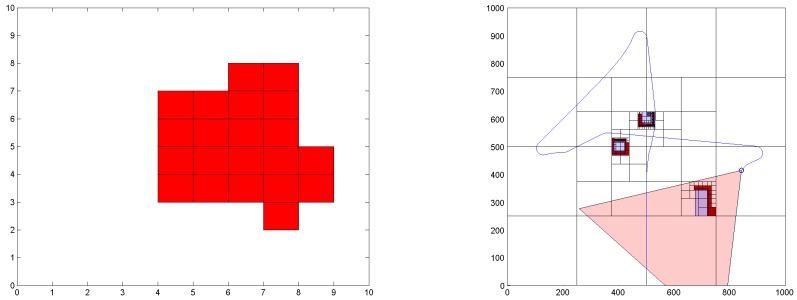
(a) Time = 2 seconds of flight



(b) Time = 20 seconds



(c) Time = 160 seconds



(d) Time = 220 seconds

Figure 5.4: Series of figures showing the synthetic vision at a sequence of times. Also shows the build up of the quadtree map over time.

Figure 5.4a shows the synthetic vision field of view and quadtree map of the obstacles early in the flight. At this particular point in time, only one obstacle is in the field of view and the quadtree map shows low-resolution representations of the three obstacles. The buildings, shown in light blue, are only for use as an aid for the reader.

Figure 5.4b shows the map and vision a short time later. The map surrounding both the building on the middle left and top center have become more refined as the UAV traveled north, keeping both in view.

Figure 5.4c later in flight when the UAV has had a chance to approach both buildings from a different direction. The quadtree mapping comes closer and closer to the actual building locations as more short-range sensor readings are recorded.

Figure 5.4d, even later in flight, gives a good view of how the map is built up on the side of the building visible to the UAV. The side of the building not visible to the UAV remains unknown to the mapping algorithm. It could still choose path points in the unseen area of the building. However, as the UAV would approach that side of the building, it would map that side and replan to the next main waypoint.

5.4.2 Flight History

In scenario 1, the UAV avoided the first obstacle by passing to the north of it. It then passed between the next two obstacles as it progressed to the second primary waypoint. Following that, it looped around to the third primary waypoint and returned to the first. On its second pass, it followed the path found for it on the first pass through the area without having to recalculate a new path. The resulting path along with the developed map can be seen in Figure 5.5.

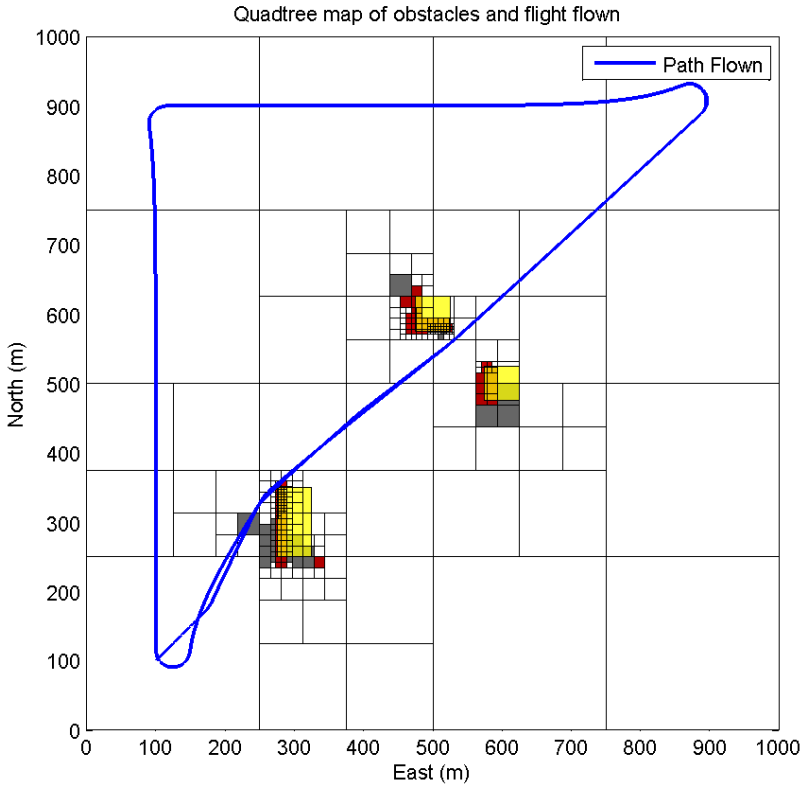


Figure 5.5: Scenario 1 path flown with actual and estimated building locations. The UAV correctly identified the location of the building and built up the quadtree around it. It modified its path and was able to fly its waypoint set multiple times.

In scenario 2, the algorithms had to re-plan three different legs. These are legs 1, 3, and 4 as shown in Figure 5.2. Each of these legs provide a different approach to a building for the UAV. It is important to note that during this scenario the initial leg to be re-planned was leg 1. It was not the initial leg that was flown, leg 3. The algorithm correctly handled replanning when the compromised leg was not the one currently being flown. The resulting path flown can be seen in Figure 5.6.

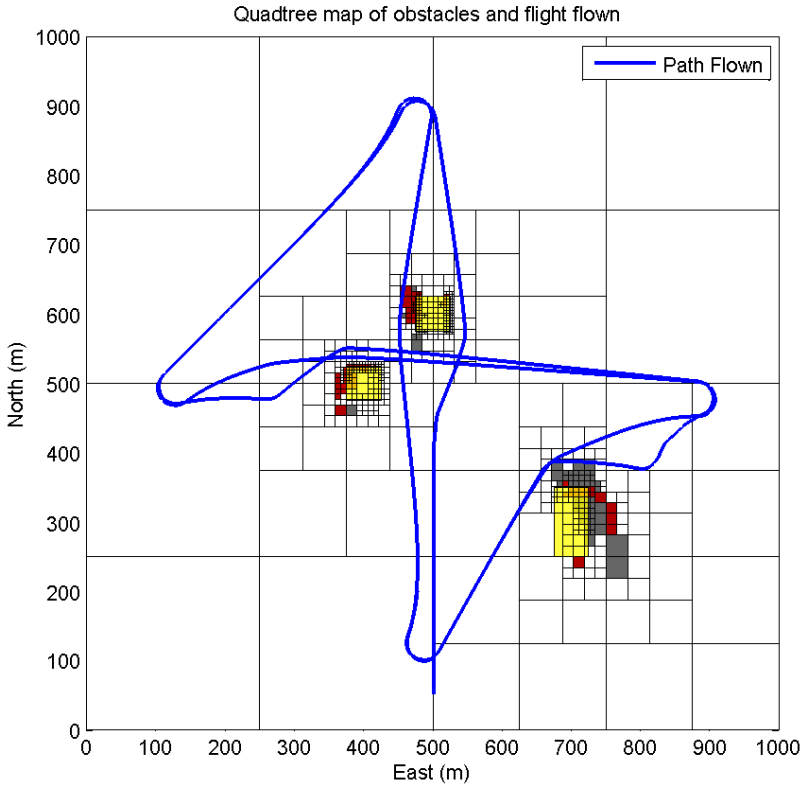


Figure 5.6: Scenario 2 flight and estimated and actual building locations. The algorithms properly mapped the buildings and found paths to avoid the obstacles.

In scenario 3, the UAV passed through a map space cluttered by ten buildings on a mission to fly to a single waypoint on the far side of the map and return to the starting location. The UAV avoided the buildings and found a path through the cluttered environment as seen in Figure 5.7.

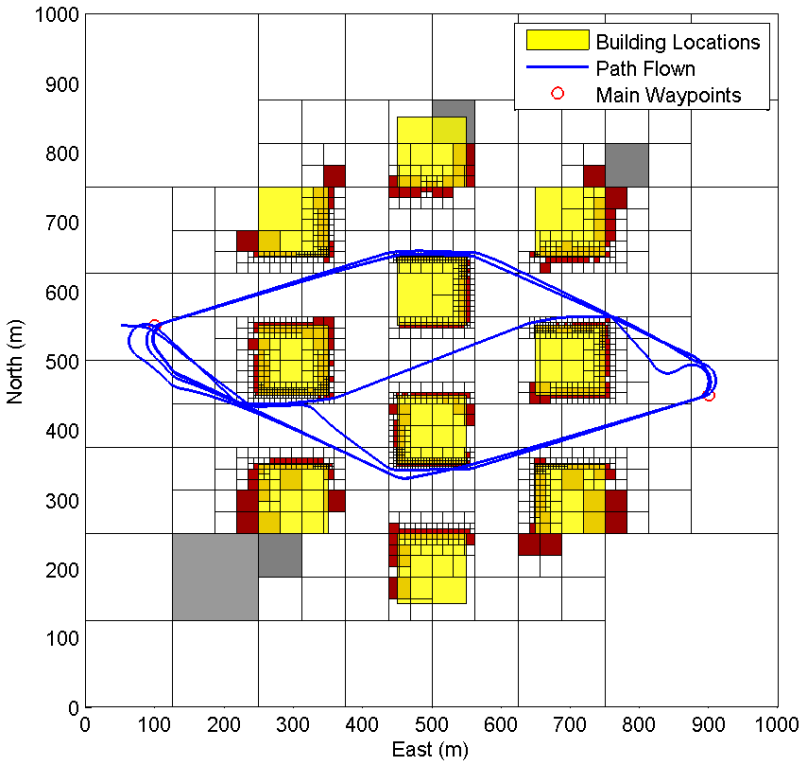


Figure 5.7: Scenario 3 map and resulting flight path

5.4.3 Planned Path Length

An important measure of performance is how closely the planned length of the path compares with the bounded optimum path length. The path lengths for the three scenarios are shown below. Each point on the graphs represents the distance of a single path planned between the two waypoints indicated. Legs will be planned multiple times throughout flight as information about the obstacles in the scenario is sensed and used to build up an obstacle map. The leg is replanned as the obstacle map is built and the path planner finds that the current path crosses a sensed obstacle. This sense, map build, check path, and replan cycle can cause the path to be replanned until the obstacles are completely mapped to the highest resolution available.

Scenario 1 has one replanned leg with obstacles between two waypoints, Scenario 2 has three replanned legs with obstacles between three pairs of waypoints, and Scenario 3 has two legs

between two waypoints. In each of the figures showing path length, the resulting path length settles out a length less than 3% longer than the bounded optimum.

The lengths of the path between the first and second waypoints of scenario 1 is shown in Figure 5.8. It shows a straight line (no obstacles) distance of 1131.4 meters, a bounded optimum of 1137.7 meters and a calculated path length of between 1140.3 meters and 1159.8 meters. As noted previously, the replanning occurs every time the obstacle map is updated and the path is found to pass through an obstacle. Because of this, the last recorded data point on the graph is the path that was found to be flyable and not pass through any obstacles. This distance of 1150.1 meters is an increase of 1.65% over the straight line distance and a 1.09% increase over the bounded optimum.

This figure also shows the effects of the RRT finding a path quickly but without guaranteeing optimality. There are six points which are significantly larger than the rest. These occurred when the path returned by the path planner traveled north of all of the buildings as seen in Figure 5.5. This is a slightly longer path than the path that passes between the northern-most buildings.

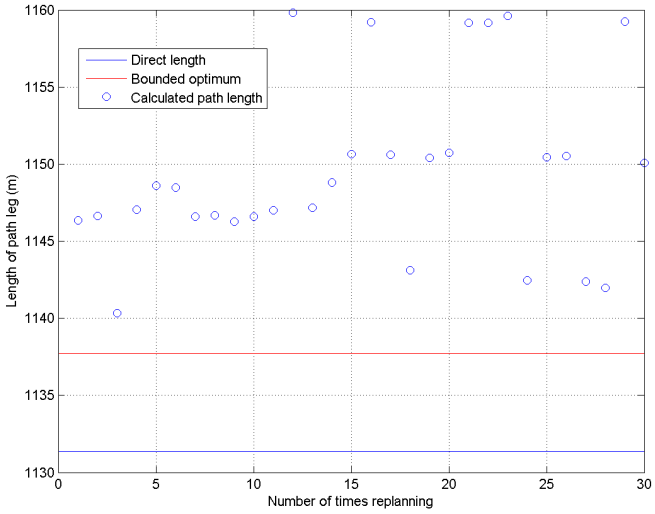


Figure 5.8: Comparison of path lengths for the leg between waypoint 1 and waypoint 2 for scenario 1. The path leg length did not change much after it was initially re-planned. The data points larger than the main body of points occured when the re-planned path went to the North side of all the buildings.

The results from scenario 2 are similar to those in scenario 1. The planned path length tends to increase as the obstacle map is developed. The results for each leg are shown in Figures 5.9, 5.10, and 5.11. These show the path length from each replanning occurance. In leg 1, the straight-line distance was 800 meters. The bounded optimum distance was 801.8 meters and a calculated distance of varied between 800.6 and 806.3 meters as seen in Figure 5.9.

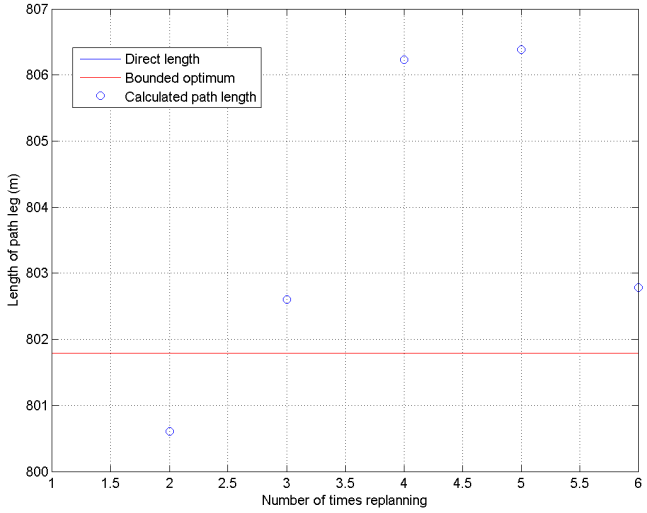


Figure 5.9: Calculated lengths of leg 1 for scenario 2. Shown here is a planned path which is actually shorter than the bounded optimum. This occurs when the mapping information is incomplete and the path is planned through an incomplete map space containing an obstacle.

For leg 3, the straight-line distance was also 800 meters with a bounded optimum of 801.8 meters. The path planner path length varied between 800.4 and 808.2 meters. The path length generally increases as the obstacle map is built up. This is due to the path being allowed to pass through obstacles in unmapped areas. As those previously unmapped areas are mapped, the obstacles are detected and the path is then planned around them. This leads to path lengths that tend to increase with each replanning, as seen in Figure 5.10

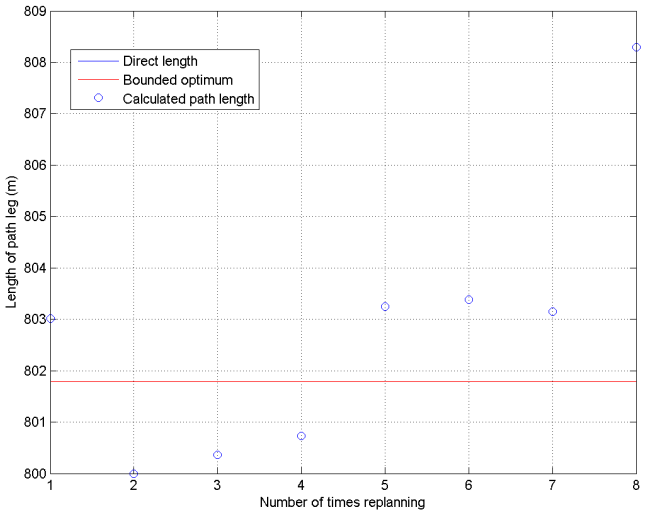


Figure 5.10: Calculated lengths of leg 3 for scenario 2. The final path, which is a path found to be free obstacles after the obstacle map is built up, is often the longest.

Leg 4 had a straight-line distance of 565.7 meters with a bounded optimum of 575.6 meters. The planned path had lengths between 568.8 and 592.1 as seen in Figure 5.11.

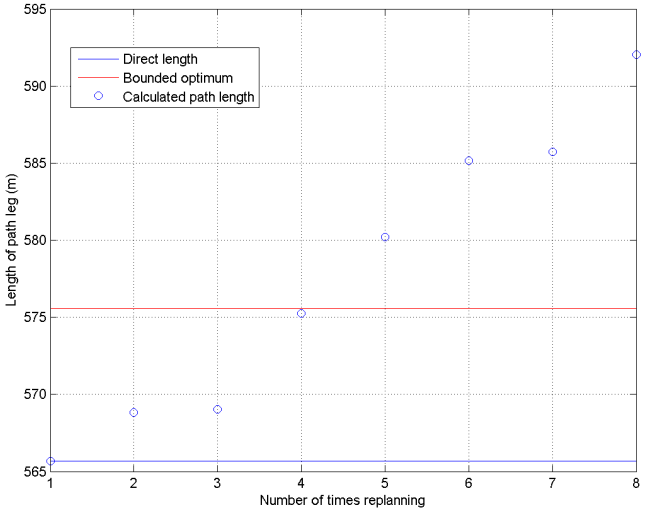


Figure 5.11: Calculated lengths for leg 4 for scenario 2.

During scenario 3, as seen in previous figures, it is possible for the path returned by the planner to be shorter than the bounded optimum. This occurs when the path found travels through an obstacle or part of an obstacle which was unmapped at the time of planning the leg. The progression of the path leg lengths can be seen in Figure 5.12. The final point on the graph was the leg that was actually found to be flyable once the obstacle map was developed in the area of path planning.

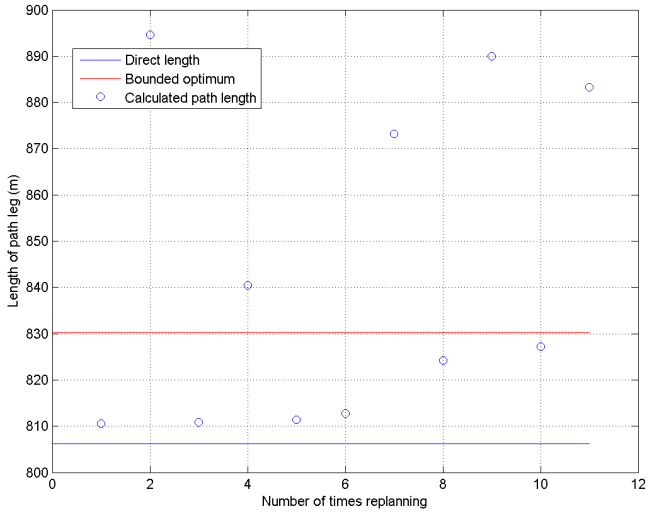


Figure 5.12: Comparison of path lengths for the leg between the two waypoints for scenario 3

5.4.4 Performance Comparison

At this point we examine the performance of the quadtree RRT versus the regular RRT. During the simulation run, the path planner was run using the regular RRT every time the quadtree RRT planner was run. Path planning between main waypoints as well as single path planning was recorded. The Figure 5.13 shows the number of iterations to find a path between two main waypoints. Data points occurred when the current path was compromised. Figure 5.14 shows the number of iterations needed to find a single-pass path. The similar trend for both of these is that the quadtree RRT requires fewer iterations and the number of iterations is much more consistent than that of the regular RRT.

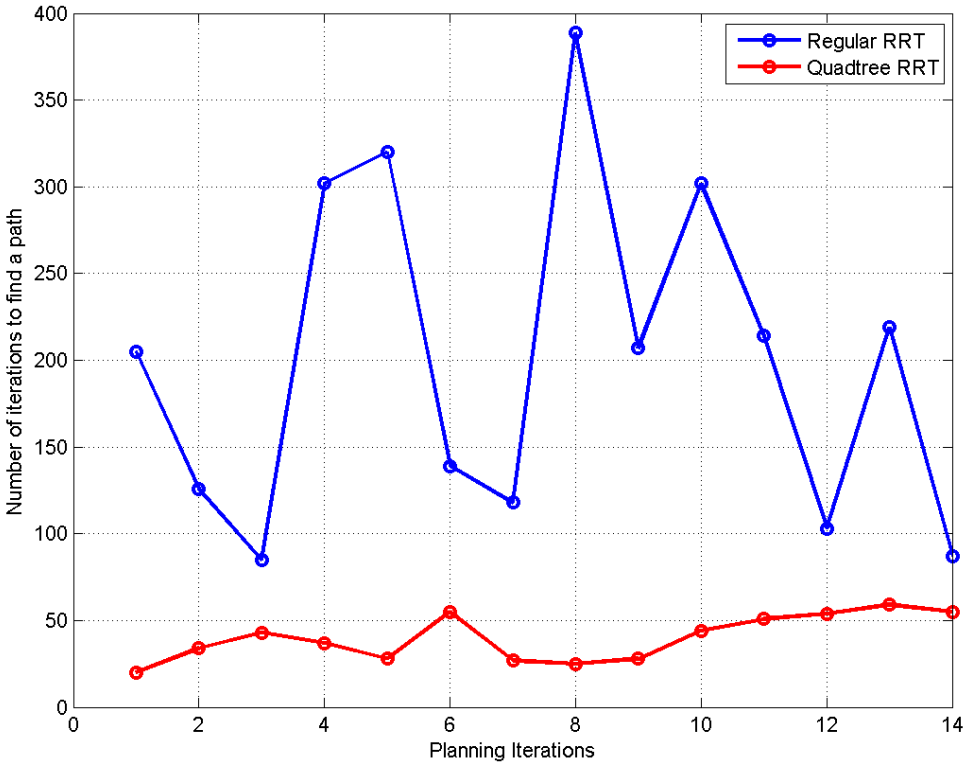


Figure 5.13: Comparison of number of iterations to find path between waypoints during simulation run for scenario 2. Of note is the significant difference between the mean number of iterations for the two RRT algorithms as well as the difference in the variance in the number of iterations.

The number of iterations required to find a path across a map is an important characteristic of the path planner. Figures 5.13 and 5.14 do not represent static maps, but they represent maps that are increasing in complexity. While the quadtree RRT algorithm is not guaranteed to be faster than a regular RRT algorithm, the quadtree RRT has beaten the standard RRT in every representative example it was tested on. Not only did the quadtree RRT find a viable path more quickly than the regular RRT, but it did so with much smaller variance in the number of iterations required.

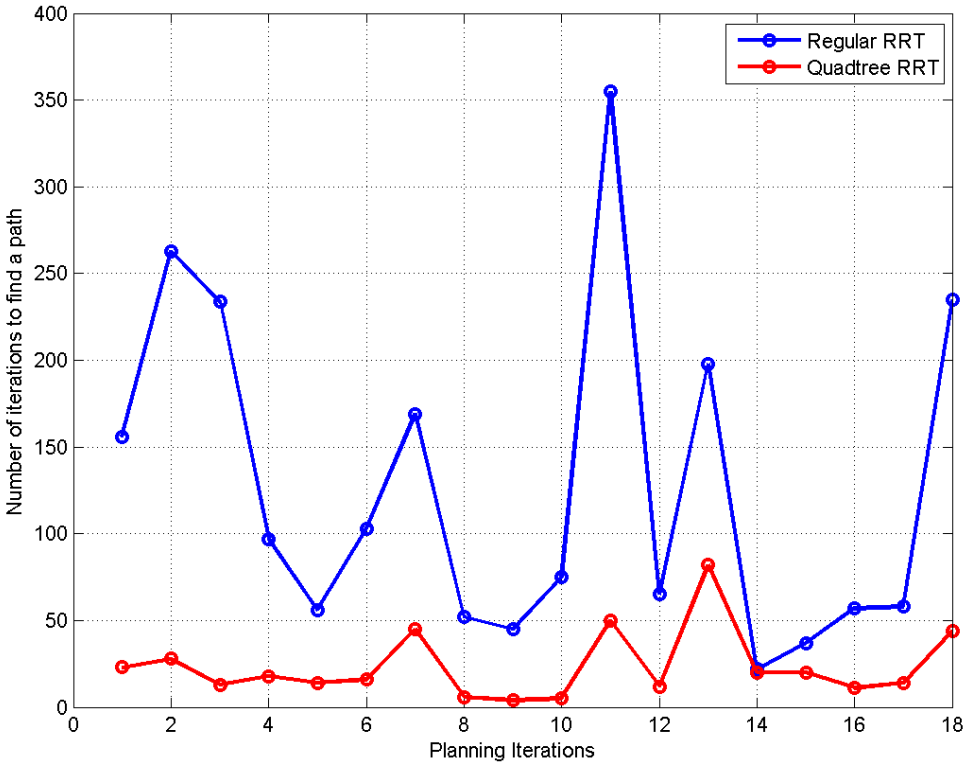


Figure 5.14: Comparison of number of iterations to find a single-pass path during simulation run for scenario 2. It can be seen that the difference in mean and variance for the algorithms seen in Figure 5.13 holds when the algorithm is used for a single-pass path. Again, this favors the quadtree RRT approach.

It is not only important to consider how many iterations the two RRT implementations required, but also to compare the results of the different algorithms. Figure 5.15 shows differences of the smoothed path distances generated by the regular and quadtree RRTs. Figure 5.16 shows the difference in distance when running the algorithms for a single-pass path. Of note in these graphs is that the outlying values do not indicate a significant deviation in performance, but rather show a path traveling the long way around a building.

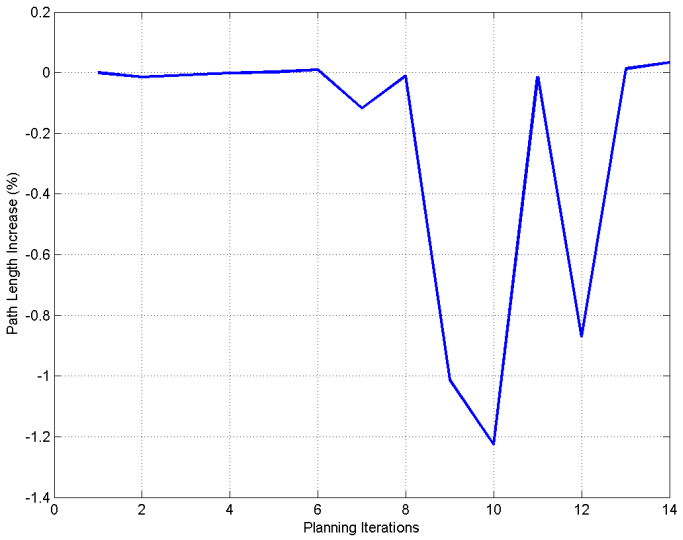


Figure 5.15: Final path distance increase of the quadtree RRT compared to the regular RRT for replanned legs between waypoints for scenario 2. Each data point represents a replanning iteration. This shows that the quadtree and RRT algorithms produce results that are within 1.25% of each other. The outliers on this figure occur when the regular RRT path traverses the longer distance around the obstacle.

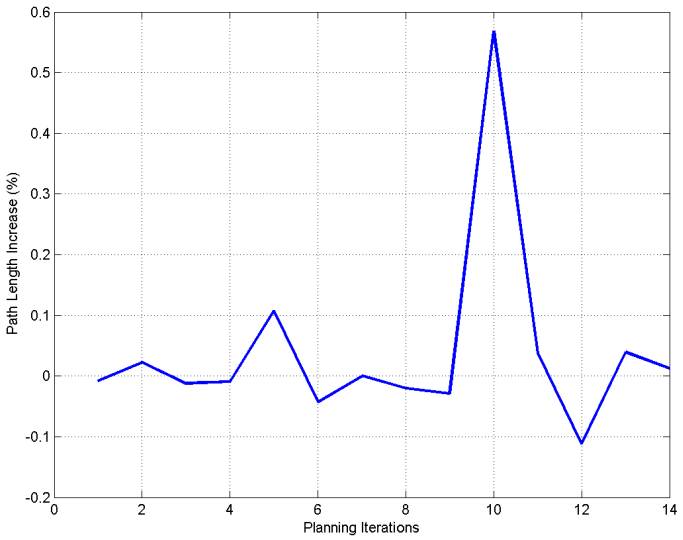


Figure 5.16: Final path distance increase of the quadtree RRT compared to the regular RRT for single-pass path for scenario 2. Each data point represents a replanning iteration. The quadtree RRT algorithm produces paths that are equivalent length to those produced by the regular RRT.

5.5 Analysis of Results

The simulation was able to successfully show the validity of the mapping and path planning algorithms. When comparing the paths found by the quadtree RRT to those paths found by the regular RRT, the quadtree paths were much more consistent in the number of steps it took to find a path. The paths tended to be slightly longer as the quadtree RRT was biased towards more open spaces. However, due to the adaptive stepsize nature of the quadtree RRT, it also had the ability to find paths through highly cluttered environments. The number of steps it takes the regular RRT to find a path is highly dependent upon the step size. When the space is relatively open, larger step sizes are appropriate. A large step size RRT is generally ineffective in a highly cluttered environment, but a small step size is computationally less efficient in low clutter environments. The quadtree RRT is an effective method of choosing an appropriate step size and minimizing variance and mean number of iterations to find a path.

The algorithms were implemented via a vector path following control schemes which allowed the UAV to tightly follow the commanded path. These were tested under different scenarios showing that this method is robust to different situations.

CHAPTER 6. CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

This thesis illustrates the need of a quick, efficient obstacle mapping and path re-planning ability. It also shows that obstacle mapping with a multi-resolution solution in conjunction with an RRT path planner can effectively create flyable paths to avoid obstacles in real time. This work is supported by the knowledge that multi-resolution maps can generally represent a map with less storage requirements than a fixed-resolution map, particularly in sparse environments. Through the use of synthetic vision, the obstacle maps can be updated while taking into account the uncertainty in simulated measurements. These results are shown in the simulation.

This thesis shows that utilizing a multi-resolution map in conjunction with RRT based path planning is not only feasible, but is advantageous over implementing an RRT algorithm on a standard map. By using this implementation, there is a net savings over a standard RRT algorithm alone. This research will help further UAV mapping and path planning research by showing the innovative combination and integration of RRTs and quadtrees.

6.2 Further Work

Although the initial results of this work are quite promising, there is much work that can still be accomplished by improving and extending the methods proposed in this thesis. Several of these ideas include refining the method of building the obstacle map, extending this searching to the third dimension, and integrating enhancements from other research for RRT searches.

The current method of building the obstacle map does not take into consideration clearing areas of interest if there is sufficient information to do so. By integrating a method of validating areas that are clear of obstacles and implementing the refining of the multi-resolution map from that, the planner will have a more refined map to plan through.

One of the biggest additions that could be made is path planning through the third dimension. The map building algorithm and path planning algorithm currently only focus on the two dimensional case. By searching in a third dimension, the path developed for the UAV might be considerably shorter if the obstacles are short enough to not cause significant deviance from altitude.

This research integrated RRTs for path planning for the UAV. The quadtree deviation of the RRT proposed by this research was a variation of a straightforward implementation of the original RRT proposed by Lavelle [21]. There have been some great advancements to optimize the RRT. By implementing these, the path planner should be able to run more quickly and with fewer iterations.

REFERENCES

[1] Byrne, J., Cosgrove, M., and Mehra, R., May 15-19, 2006. "Stereo based obstacle detection for an unmanned air vehicle." In *Proceedings of the 2006 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2830 – 2835.

[2] Bryson, M., and Sukkarieh, S., 2005. "Bearing-Only SLAM for an Airborne Vehicle." In *Australasian Conference on Robotics and Automation*.

[3] Thrun, S., Liu, Y., Koller, D., Ng, A. Y., Ghahramani, Z., and Durrant-Whyte, H., 2004. "Simultaneous localization and mapping with sparse extended information filters." *The International Journal of Robotics Research,* **23**, pp. 693–716.

[4] Pagac, D., Nebot, E., and Durrant-Whyte, H., 1998. "An evidential approach to map-building for automated robots." *IEEE Transactions on Robotics and Automation,* **14**, pp. 623–629.

[5] Hsu, J. Y.-j., and Liang-Sheng, H., 1998. "A graph-based exploration strategy of indoor environments by an autonomous mobile robot." In *Proceedings of the 1998 IEEE International Conference on Robotics & Automation*.

[6] Ayala, D., Brunet, P., Juan, R., and Navazo, I., 1985. "Object representation by means of nonminimal division quadtrees and octrees." *ACM Trans. on Graphics,* **1**, pp. 41–59.

[7] Yahja, A., Stentz, A., Singh, S., and Brumitt, B., 1998. "Framed-quadtree path planning for mobile robots operating in sparse environments." *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on,* **1**, May, pp. 650–655 vol.1.

[8] Eppstein, D., Goodrich, M. T., and Sun, J. Z., 2005. "The skip quadtree: a simple dynamic data structure for multidimensional data." In *SCG '05: Proceedings of the twenty-first annual symposium on Computational geometry*, ACM, pp. 296–305.

[9] Lingelbach, F., 2004. "Path planning using probabilistic cell decomposition." In *Proceedings of the 2004 IEEE Internatitional Conference on Robics and Automation*.

[10] Burlet, J., Aycard, O., and Fraichard, T., 2004. "Robust motion planning using markov decision processes and quadtree decomposition." In *Proceedings of the 2004 IEEE International Conference on Robotics & Automation*.

[11] Tsiotras, P., 2007. "Multiresolution hierarchical path-planning for small UAVs." In *European Control Conference*.

[12] Redding, J., McLain, T. W., Beard, R. W., and Taylor, C. N., 2006. "Vision-based target localization from a fixed-wing miniature air vehicle." In *American Control Conference*.

[13] Sharma, R., Saunders, J. B., Taylor, C. N., and Beard, R. W., 2009. "Reactive collision avoidance for fixed-wing MAVs flying in urban terrain." In *AIAA Guidance, Navigation, and Control Conference*.

[14] Yu, H., Beard, R., and Byrne, J., 2009. "Vision-based local multi-resolution mapping and path planning for miniature air vehicles." In *American Control Conference, 2009. ACC '09.*, pp. 5247 –5252.

[15] Barraquand, J., and Latombe, J.-C., Dec 1991. "Robot motion planning: A distributed representation approach." *The International Journal of Robotics Research,* **10**, pp. 628 – 649.

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

[17] LaValle, S. M., 1998. Rapidly-exploring random trees: A new tool for path planning Tech. rep., Iowa State University.

[18] Griffiths, S., Saunders, J., Curtis, A., Barber, B., McLain, T., and Beard, R., 2006. "Maximizing miniature aerial vehicles." *Robotics Automation Magazine, IEEE,* **13**(3), pp. 34 –43.

[19] Hansen, S. R., 2007. "Applications of search theory to coordinated searching by unmanned aerial vehicles." Master's thesis, Brigham Young University.

[20] Nelson, D. R., Barber, D. B., McLain, T. W., and Beard, R. W., 2006. "Vector field path following for small unmanned air vehicles." *American Control Conference*, June.

[21] LaValle, S. M., 1998. Rapidly-exploring random trees: A new tool for path planning Tech. rep., Iowa State University.

APPENDIX A. SIMULATION BACKGROUND

A.1 Overview

The simulation used for this research was developed by the BYU Magicc Lab to use in conjunction with the hardware used for flight testing. An overview of how it flows together can be seen in Figure A.1. At the heart of the simulation is a 12-state six-degree-of-freedom model for kinematics and dynamics. This is driven by motor thrust and aerodynamic forces and torques acting on the body and fins of the UAV. Low level autopilot control is provided in .dll form from code used in the Kestrel Autopilot. The Kestrel Autopilot originated from BYU. Mission guidance is provided in a waypoint management model. The guidance is fed by waypoint commands from the path planner. The path planner receives an obstacle map from the the mapping algorithm which it uses to check the path and decide if it needs to replan. The mapping algorithm receives a depth map from the synthetic vision which it uses to build up its map of obstacles. Synthetic vision receives the state information and uses that along with knowledge of building location to create the depth map. State information is also fed to the path planner algorithm, mapping algorithm, and autopilot.

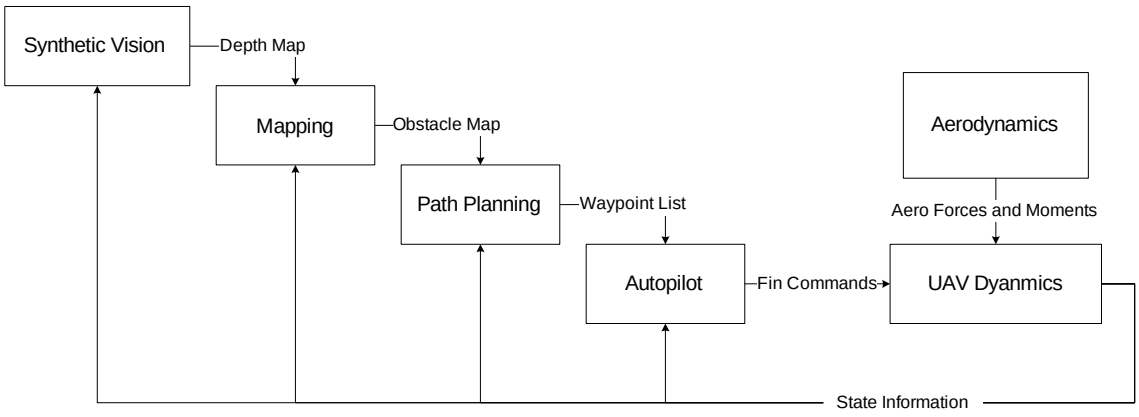


Figure A.1: Simulation block diagram. This shows the flow of information between the different model blocks.

A.2 Dynamics Model

The simulation uses a 12-state six-degree-of-freedom model for kinematics and dynamics as shown below. The model consists of inertial position states p_n, p_e, and p_d, Euler angles ϕ, θ, and ψ. The model also includes body rates p, q, and r, as well as body frame velocities u, v, and w. The system is driven by forces f_x, f_y, and f_z and moments l, m, and n. These forces and moments come from an aerodynamics model based upon commanded fin deflections δ_e, δ_r,and δ_a.

$$\begin{pmatrix} \dot{p_n} \\ \dot{p_e} \\ \dot{p_d} \end{pmatrix} = \begin{pmatrix} c_\theta c_\psi & s_\phi s_\theta c_\psi - c_\phi s_\psi & c_\phi s_\theta c_\psi + s_\phi s_\psi \\ c_\theta s_\psi & s_\phi s_\theta s_\psi - c_\phi c_\psi & c_\phi s_\theta s_\psi + s_\phi c_\psi \\ -s_\theta & s_\phi c_\theta & c_\phi c_\theta \end{pmatrix} \begin{pmatrix} u \\ v \\ w \end{pmatrix}$$

$$\begin{pmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{pmatrix} = \begin{pmatrix} rv - qw \\ pw - ru \\ qu - pv \end{pmatrix} + \frac{1}{m} \begin{pmatrix} f_x \\ f_y \\ f_z \end{pmatrix}$$

$$\begin{pmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix} = \begin{pmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \frac{\sin\phi}{\cos\theta} & \frac{\cos\phi}{\cos\theta} \end{pmatrix} \begin{pmatrix} p \\ q \\ r \end{pmatrix}$$

$$\begin{pmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{pmatrix} = \begin{pmatrix} \Gamma_1 pq - \Gamma_2 qr \\ \Gamma_5 pr - \Gamma_6 (p^2 - q^2) \\ \Gamma_7 pq - \Gamma_1 qr \end{pmatrix} + \begin{pmatrix} \Gamma_3 l + \Gamma_4 n \\ \frac{1}{J_y} m \\ \Gamma_4 l + \Gamma_8 n \end{pmatrix}$$

APPENDIX B. SIMULATION SCENARIOS

The UAV scenarios can be found listed in tabular form below. Table B.1 shows the waypoints for scenario 1. Table B.2 shows the building locations for scenario 1. Table B.3 shows the waypoint locations for scenario 2. Table B.4 shows the building locations for scenario 2. Table B.5 shows the waypoint locations for scenario 3, while Table B.6 shows the building locations for scenario 3.

B.1 Scenario 1

Table B.1: Scenario 1 Waypoint Locations

X	Y	Z
100	100	50
900	900	50
100	900	50

Table B.2: Scenario 1 Buildings

Loc_x(m)	Loc_y(m)	Loc_z(m)	Dim_x(m)	Dim_y(m)	Di_z(m)
300	300	0	50	100	100
600	500	0	50	50	100
600	600	0	50	50	100

B.2 Scenario 2

Table B.3: Scenario 2 Waypoint Locations

X	Y	Z
100	0	50
900	0	50
0	100	50
0	900	50

Table B.4: Scenario 2 Buildings

Loc_x(m)	Loc_y(m)	Loc_z(m)	Dim_x(m)	Dim_y(m)	Di_z(m)
700	300	0	50	100	100
600	500	0	50	50	100
500	600	0	50	50	100

B.3 Scenario 3

Table B.5: Scenario 3 Waypoint Locations

X	Y	Z
100	550	50
900	450	50

Table B.6: Scenario 3 Buildings

Loc_x(m)	Loc_y(m)	Loc_z(m)	Dim_x(m)	Dim_y(m)	Di_z(m)
300	300	0	100	100	100
300	500	0	100	100	100
300	700	0	100	100	100
500	400	0	100	100	100
500	600	0	100	100	100
700	300	0	100	100	100
700	500	0	100	100	100
700	700	0	100	100	100