©Copyright 2014
Cody O. Deacon

Autonomous Collision Avoidance and Mapping for a Quadrotor Using Panning Sonar Sensors

Cody O. Deacon

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science in Aeronautics & Astronautics

University of Washington

2014

Reading Committee:

Kristi Morgansen

Juris Vagners

Program Authorized to Offer Degree:
Department of Aeronautics & Astronautics

University of Washington

Abstract

Autonomous Collision Avoidance and Mapping for a Quadrotor Using Panning Sonar Sensors

Cody O. Deacon

Chair of the Supervisory Committee:
Associate Professor Kristi Morgansen
Aeronautics & Astronautics

Recent research and development of autonomous vehicles has expanded their capabilities in both military and civilian applications. A variety of missions including military and police surveillance, scientific reconnaissance, and search and rescue in large or hazardous search areas can now be done more efficiently and effectively through the use of autonomous vehicles. In order to decrease workload and reduce risk of further human life, the research here focuses on implementation of an autonomous navigation algorithm to allow a quadrotor to successfully maneuver through its environment. Using two ultrasonic sonar sensors and implementing a Vector Field Histogram (VFH) algorithm in conjunction with an A* algorithm, a technique is created that demonstrates successful navigation through a dense environment to a prespecified waypoint without any prior environmental knowledge. Furthermore, using a voting and devoting scheme on a Cartesian grid provides robustness to spurious or inconsistent sensor data. The algorithm is tested in a simulation for a quadrotor.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

First, I would like to thank my advisor, Kristi Morgansen, for her assistance and guidance throughout my research and thesis. I would also like to thank Juris Vagners for his assistance and for funding me through the Andris Vagners Memorial Fellowship. Thank you to the entire Nonlinear Dynamics and Controls Laboratory for their inviting and team oriented mindset. They were always happy to assist in any way they could. Finally, I would like to thank the Office of Naval Research for funding a portion of my research through ONR grant N000141010952.

DEDICATION

To my wife and best friend, Janée.

Chapter 1
INTRODUCTION

The utility of unmanned aerial vehicles (UAVs) has been realized in recent years for a variety of applications both military and civilian. UAVs provide a cost effective alternative to manned vehicles in situations where direct human action would be unnecessarily expensive, tedious, or dangerous. The Northrop Grumman RQ-4 Global Hawk flies into hurricanes to collect environmental data on the recent increase in hurricane intensity [3]. For the first time in 2008, the MQ-9 Reaper fully replaced the U.S. Air Force 174th Fighter Wing F-16 fleet because of the advantages the UAV offers over manned aircraft. Aside from the cost effectiveness, the Reaper is better suited for dangerous missions and can remain on site for more than 14 hours, able to respond at a moment's notice [4]. Police and fire departments in Europe use quadrotor platforms to provide cost effective aerial views for evidence gathering as well as for search and rescue [5].

All of these organizations are moving towards UAV integration because they are able to accomplish missions with less risk to human life and in a more cost effective manner than with manned alternatives. Since UAVs are not constrained by human limitations, they can be smaller, lighter, and more maneuverable than traditional manned aircraft. These greater design freedoms allow for aircraft that are more suited for specific missions, meaning that tasks can be successfully accomplished more efficiently than manned aircraft. Surveillance missions that once required vehicles large enough to carry a human pilot, all the required life support systems, and a human control interface can now be accomplished by a small, electrically powered UAV and a camera.

Another anticipated advantage of UAVs is their potential to reduce the required man-hours while still accomplishing the same mission. First, maintenance of an unmanned system can be less rigorous than of a manned system because the cost of equipment failure is much lower. Not only is an unmanned system generally less expensive than a manned system, but failure is also less devastating. For these reasons, maintenance schedules are more flexible, and parts are held to less stringent criterion. The second way that UAVs could reduce required man-hours is that they conceivably do not require an operator during all mission phases. If designed for sufficient autonomy, a UAV can perform many missions independent of user input. While the workload decrease is less significant for the second reason, the fact that the workload decreases at mission critical moments can potentially increase safety and success rates. A UAV that can autonomously perform intercontinental travel means pilots are not at risk to fatigue that could lead to crashes. Likewise, a quadrotor that can autonomously enter a burning building and search for victims frees its operator to focus on other tasks.

No matter the application, UAVs must be able to successfully interact with their environments in order to work autonomously. Successful interaction means being able to fly around restricted airspace, avoid collisions with buildings or other aircraft, or navigate in close quarters in a cluttered environment. Successful environment interaction is particularly difficult in indoor environments because sensors such as GPS and compasses will likely face interference. Furthermore, indoor environments are inherently more densely occupied, requiring more accurate localization and mapping data.

1.1 Previous Research

Much previous research has been performed involving autonomous environment interactions with small unmanned vehicles. Early research involved wandering algorithms with local obstacle avoidance. The simplest example would be an algorithm that commanded a vehicle to turn away from any sensed obstacle. Aside from pure collision avoidance, the next step was the development of simultaneous localization and mapping (SLAM) as first introduced in [6] and [7]. The premise of SLAM is to build a map of an environment while simultaneously localizing a robot's position based on obstacle association with previously detected obstacles. This method allows the robot to correct for odometry error and successfully travel through an unknown environment gathering sensor information while simultaneously mapping the robot's surroundings. SLAM methods have continued to evolve with a multitude of different types of sensors and vehicles to fit specific needs, and each combination of sensors and vehicles presents unique benefits and challenges. The most common sensors used for SLAM include RGB sensors (video cameras), laser sensors, or LIDAR systems. Using video and laser systems has the advantage of low uncertainty, making the data analysis more precise. The work in [8] used a SLAM algorithm with a SICK laser scanner and [9] tackled the SLAM problem using a Microsoft Kinect camera to map indoor environments. These research projects proved that both laser scanners and RGB sensors are viable options for SLAM, but due to weight constraints, computational power constraints, and cost effectiveness, some systems have turned to the use of sonar sensors. Sonar sensors have many advantages for a fully on-board autonomous navigation system. They are light, inexpensive and not data intensive (a single sonar ping only returns a distance). All of these assets are important when developing a system that is weight constrained and designed to enter potentially hazardous areas as damage to, or loss of, equipment is a possibility. Two of the first publications on SLAM using sonar sensors were [10] and [11]. In both cases, a ring of 16-24 sonar sensors was mounted on a mobile ground robot. The methods used for feature detection were different, but both projects successfully mapped indoor environments using sonar.

Upon developing SLAM algorithms, autonomous vehicles can utilize the map information to navigate to specified locations within their environments. Early research involved purely local obstacle avoidance as described in [12]. However, trying to navigate with only local information leads to vehicles frequently becoming trapped as obstacles disappear and reappear from view. As a result, algorithms such as those described in [13] and [14] were developed to take advantage of global map knowledge through use of algorithms such as the A* search method and the rapidly-exploring random tree (RRT) path planner.

1.2 Research Contributions and Chapter Organization

The contributions of this research project are: (1) a new mapping algorithm that functions with only two sonar sensors, and (2) implementation of a collision avoidance and path planning algorithm on a flying vehicle without prior knowledge of the exploration space. This research is different from previous works because sonar based collision avoidance, mapping, and path planning is being utilized on a UAV as opposed to a ground robot, and it is being performed using only two sonar sensors, as opposed to an entire ring of sensors. Navigation and mapping with two sonar sensors is lighter, less expensive, and less data intensive than previous research to perform the same task, albeit less robust.

Chapter 2 discusses the specific hardware on which this project is based. In Chapter 3, the approach to mapping indoor environments using only two sonar sensors is discussed. The map is created using a Cartesian grid with voting and de-voting algorithms to determine the existence of obstacles. In Chapter 4, the VFH* navigation algorithm that is implemented is discussed. Chapter 5 demonstrates the success of the VFH* algorithm in MATLAB simulation. Chapter 6 discusses the conclusions from the current research as well as opportunities for research in the future.

Chapter 2
HARDWARE

Although SLAM and autonomous path planning have been researched extensively in recent years, the hardware being considered here makes this project unique. Aside from being a test platform for experimentation on path planning algorithms, the quadrotor has been modified to incorporate an actuating mass pendulum for pitch control. The idea to control pitch from an actuating mass is biologically inspired from hawk-moths, who use their abdomens for pitch control as described in [15]. However, the addition of the actuating mass makes weight constraints very important for the rest of the hardware components. Therefore, designing the lightest possible navigation system was key to the research project's success.

2.1 Vehicle Hardware

The test platform for this project is an Ascending Technologies (AscTec) Hummingbird quadrotor as shown in Figure 2.1. According to AscTec, the maximum payload is 200 grams. The extra hardware that has been added to the quadrotor includes a processing stack shown in Figure 2.2 with a Gumstix Overo Firestorm 800 MHz processor, 1 GB of RAM, and a wireless internet antenna. The processing stack is 180 grams.

The actuating mass pendulum has not been implemented yet and will utilize the quadrotor battery for the mass, but the pendulum structure is more than 200 grams. Therefore, the quadrotor payload with the actuating pendulum will be greater than the published maximum payload from AscTec. The quadrotor will still be able to function with degraded performance because the rotors have a measured maximum

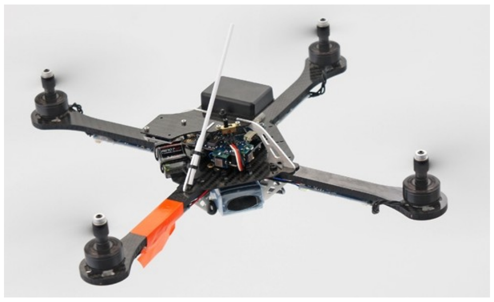

Figure 2.1: A photograph of the AscTec Hummingbird [1]. The Hummingbird is approximately 500 grams and generates a maximum thrust of 15 Newtons.

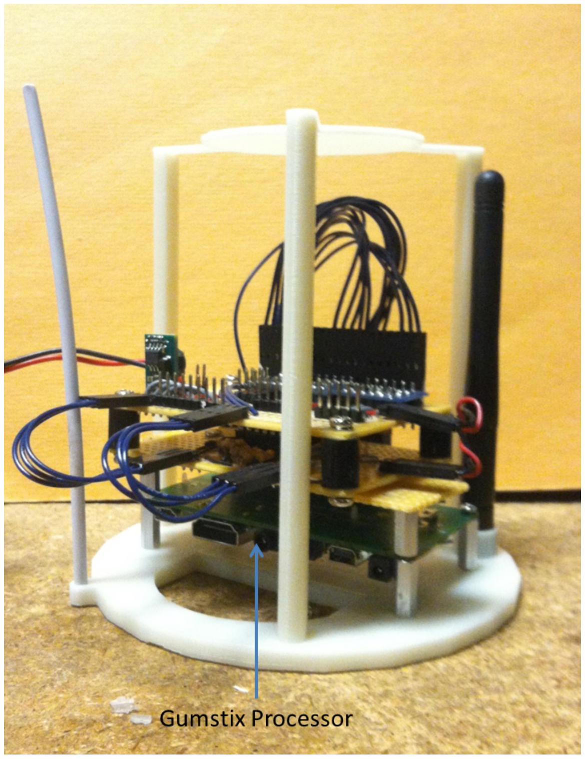

Figure 2.2: The processing stack that mounts on the top, center of the quadrotor.

thrust of 15 Newtons, equating to the ability to lift a maximum gross weight of 1,529 grams. However, adequate maneuverability requires significant excess thrust, and weight is a precious commodity. Therefore, the hardware for navigation needs to be as minimalistic as possible.

2.2 Sensor Hardware

The quadrotor has five total sensors for state estimation. While experimental tests were not performed during this project, these physical hardware constraints were used to set up the simulation parameters. The sensors that the collision avoidance algorithm primarily uses are three sonars: one fixed downward for altitude awareness, one fixed forward to detect imminent collisions, and one that pans 180^o in front of the vehicle to gather additional environmental information. The idea of a panning sonar sensor is different from previous research. The work in [10] and [11] utilized a ring of 16-24 sensors, but the panning sonar is more suited for flying aircraft. Fewer sonars presents a challenge because less environmental information can be gathered in a discrete amount of time. Therefore, fixing a forward facing sensor is essential to prevent collisions in the case of insufficient data from the panning sonar. The specific sonar being used on the quadrotor is the LV-MaxSonar-EZ4 sonar range finder. The beam characteristics, represented in Figure 2.3, describe the detection patterns for different sized objects. Image (A) represents the detection pattern for a 0.25 inch diameter dowel, image (B) represents the detection pattern for a 1 inch diameter dowel, image (C) represents the pattern for a 3.25 inch diameter rod, and image (D) represents the pattern for an 11 inch wide board. One should note that the beam width shape represented in Figure 2.3 is the effective beam width (i.e. the area in which an obstacle will elicit a return) due to the physical shape of the sensors, not the actual beam width resonating from the sensor. Furthermore, due to the nature of a sonar sensor, there are physical limitations to detecting extremely small objects until the objects are nearly within rotor length of the vehicle. However, for larger obstacles

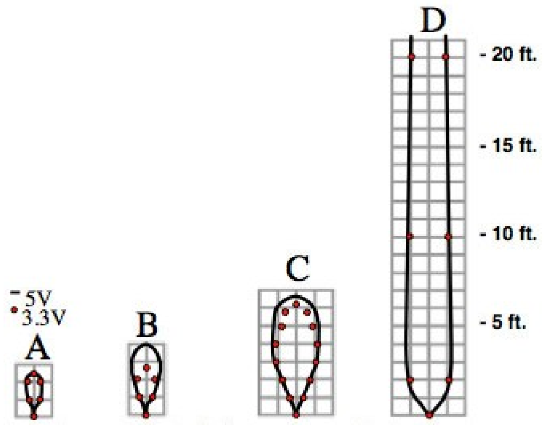

Figure 2.3: Sonar beam width characteristics of the LV-MaxSonar-EZ4 sonar range finder as described in [2]. (A) represents the detection pattern for a 0.25 inch diameter dowel, (B) represents the detection pattern for a 1 inch diameter dowel, (C) represents the pattern for a 3.25 inch diameter rod, and (D) represents the pattern for an 11 inch wide board.

at effective distances, the beam width can be conservatively modeled as expanding 30^o from center for the first two feet, then remaining a constant width of two feet out to a maximum range of 20 feet. This beam width estimate is conservative because the uncertainty is overestimated at all ranges, marking more potentially clear areas with obstacles than if the beam width were modeled exactly. An important note is that there are no upward facing sensors on the vehicle. As a result, the quadrotor must work under the assumption that any time it increases altitude, the area overhead is clear. In order to mitigate the risk of colliding with overhead obstacles, the quadrotor will only hold a single altitude after initial take-off, essentially eliminating obstacle avoidance in the third dimension. Upon mounting upward facing sensors, the algorithm can be reconfigured to perform obstacle avoidance in three dimensions.

Other sensors include an inertial measurement unit (IMU) for roll/pitch/yaw data and a downward facing optic flow sensor that estimates velocity and can be integrated for position. These sensors are primarily used in conjunction with an Unscented Kalman Filter as part of a dead reckoning position estimator. With all of these sensors, the system is partially observable, requiring prior position knowledge to estimate the quadrotor's current position. As a result, the quadrotor's position estimate is prone to accumulation error. The consequence of the accumulation error will be unknown until the quadrotor undergoes real-world testing. However, if the accumulation error is too great to successfully navigate, the quadrotor will require an external localization technique to provide position information until an on-board localization algorithm is developed.

Chapter 3
MAPPING

Assuming that the vehicle has a way to localize itself within its environment, the main dilemma when performing obstacle avoidance and mapping is creating an accurate enough map of the environment to then implement an algorithm to successfully navigate the map.

While sonar provides multiple benefits over other sensor types, creating an accurate map using sonar presents a larger challenge than mapping using sensors such as video and lasers. Sonar often produces false returns or fails to receive a return when an object is present. Furthermore, sonar has a very large angular uncertainty, making it difficult to determine an obstacle's exact location. To overcome these challenges, a room is mapped on a Cartesian grid as done by [10], [11], [12], [17], [13]. The individual cells of the grid accumulate votes based on sonar returns that indicate the presence (or absence) of an obstacle occupying that cell.

3.1 Spurious Data Challenge

To overcome the spurious data challenge, the environment here is mapped using a voting scheme similar to those used in [10], [11], and [12]. The approximate size of the environment is estimated, and a Cartesian grid is created to represent the room. When estimating an environment, overestimation is preferred because the quadrotor will discover and avoid natural boundaries as long as the boundaries fit within the confines of the grid. As the quadrotor explores the environment, the individual cells of the grid accumulate votes based on sonar returns. After cells surpass a threshold value, the cell is assumed to be occupied by an obstacle. The threshold value used here was obtained experimentally and seemed to function well with a threshold of 10 votes to signify a cell was occupied with the maximum value a cell could reach being 20 votes. The purpose of setting a maximum value will be discussed more thoroughly shortly. By requiring multiple returns to determine the existence of obstacles, the algorithm is naturally robust to spurious data and false returns. However, with such a large uncertainty arc, creating a navigable map required further data analysis.

3.2 Overcoming Large Sonar Uncertainty

There are many different approaches to creating a map despite the sonars' large uncertainty. The research in [10] divided the returns into two types, point returns and line returns, then found point features where multiple uncertainty arcs intersected and found line features using a Hough transform. The work in [11] used the same method for point features, but determined line features by strategically setting up the sensors on the vehicle and using their orientations with one another. Since Hough transforms are computationally expensive and we were limited by size and weight constraints on the quadrotor, neither of these options were feasible for our system. Instead, a de-voting algorithm was developed to reduce the certainty of cells which failed to produce a return. The premise of this algorithm is that an obstacle cannot be detected if it is behind another obstacle. Therefore, when an obstacle is detected, the algorithm lowers the certainty value of all cells within the beam width and in front of the detected obstacle (with a minimum certainty value of 0). After multiple iterations from slightly different quadrotor positions, the de-voting algorithm allows the sensors to effectively locate the position of an obstacle despite the wide beam width. The algorithm, however, is potentially sensitive to the quadrotor remaining stationary for a prolonged period of time. Without a maximum cell value, as discussed previously, a stationary quadrotor could accumulate a large number of votes in an unoccupied cell that is within the uncertainty arc. By placing a maximum certainty value on the cells, the de-voting algorithm is able to function properly even if the quadrotor does not move for a long period of time. Figures 3.1 and 3.2 represent a simulated environment and the map that the voting algorithm created, respectively. In Figure 3.2, the dark areas represent cells with high certainty that an obstacle is present, and the gray areas represent unexplored portions of the map. The reason that unexplored areas are represented as gray is because initially all cells are given a certainty value slightly less than is required to confirm the existence of an obstacle. Early simulations assumed all cells were unoccupied and required positive identification of an obstacle before a cell was considered occupied. Requiring positive obstacle identification led to collisions in situations where the quadrotor initialized near a wall or when the quadrotor attempted to make sharp turns around corners because the quadrotor sensors were unable to acquire enough samples to prove an obstacle's existence before the quadrotor collided with the obstacle. By initializing all cells just below the certainty threshold, the quadrotor was still able to move freely without assuming all cells were occupied, but it could quickly identify when it was beginning near obstacles or maneuvering around corners.

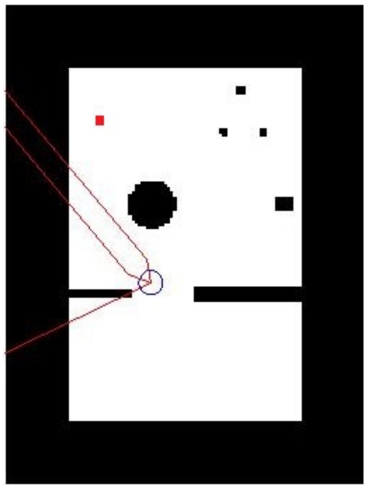

Figure 3.1: A full map of the simulated environment that the quadrotor was navigating. In the image, black areas represent obstacles, and the red square is the target destination. The quadrotor is facing in the direction of the double wide lines, representing the forward facing sonar beam. The single line extending to the left of the quadrotor represents the panning sonar. Although the panning sonar is represented in the image as a single line, the beam characteristics match the beam characteristics of the forward facing sonar.

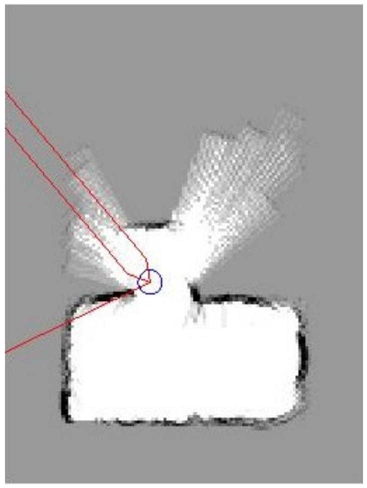

Figure 3.2: A certainty map of the simulated environment that the quadrotor is navigating. In the image, black areas represent obstacles that the sonars have detected and white areas represent places the quadrotor has declared obstacle free. All cells are given an initial certainty value slightly less than what is required to deem an obstacle present. Therefore, gray areas represent places that the quadrotor has yet to explore.

Chapter 4
OBSTACLE AVOIDANCE

Once a reliable map is created, the next challenge is developing an algorithm to efficiently navigate to a target position. Furthermore, the map is constantly updated as new information becomes available from the sonar sensors. Effectively, a receding horizon control problem presents itself because the quadrotor must make navigation decisions based on current data, then reevaluate after moving only slightly.

4.1 Vector Field Histogram Algorithm

The first navigation approach in this research was the vector field histogram (VFH) algorithm, which was developed by [12] for a rolling vehicle. The VFH method is based on the Vector Force Field (VFF) potential field navigation method similar to the potential field navigation methods used in [16]. In VFF, the vehicle is represented as a particle that is acted on by attractive and repellent forces. A constant attractive force, F_a, draws the quadrotor towards the target location, and repellent forces push the quadrotor away from obstacles. The repellent forces are a Gaussian function of the obstacle's distance from the vehicle, multiplied by the certainty value of the obstacle's existence as determined by the voting scheme. However, only obstacles within an active window centered around the vehicle and of predefined size $w_s \times w_s$ (discovered experimentally) emit a repulsive force. The repulsive force is defined by

$$F_{i,j} = e^{-\frac{d_x^2+d_y^2}{w_s}} c_{i,j}^*, \tag{4.1}$$

where d_x and d_y are the x and y distances from the quadrotor, respectively, and $c_{i,j}$ is the certainty value of an obstacle existing within cell (i, j) of the active window.

Furthermore, $c^*_{i,j}$ indicates only cells within the active window. The total repellent force is equal the sum of the repellent forces of each individual cell,

$$F_r = \sum F_{i,j}. \tag{4.2}$$

The final resultant force acting on the vehicle, normalized for maximum velocity, is:

$$F = F_a + F_r. \tag{4.3}$$

While the VFF method is a sound theory, there are many challenges with application. First, success or failure is highly sensitive to force gain tuning. If the attractive force is too strong, the vehicle tries to move directly through obstacles, and if the repulsive force is too strong, the vehicle often fails to pass between obstacles. The VFH algorithm uses a two stage data reduction to address the VFF navigation problems.

VFH uses a similar process to VFF, except rather than calculating force vectors for each individual cell using (4.1) within the active window, VFH calculates cell magnitudes,

$$m_{i,j} = {c^*_{i,j}}^2(a - bd^2_{i,j}), \tag{4.4}$$

where,

$$a,\ b = \text{positive constants} \qquad d_{max} = \frac{w_s - 1}{2}$$

$$a - bd^2_{max} = 1.$$

Upon calculating the cell magnitudes, two Cartesian grids are determined. The first, the global grid, shows the environment knowledge based on sonar returns as shown in the example in Figure 4.1. The second grid, the local active window, shows a $w_s \times w_s$ grid centered around the vehicle with obstacle cells of different magnitudes as represented by the color plot in Figure 4.2. The second stage of data reduction creates a polar histogram from the active window. The active window is split into K sectors of angular width α. In these experiments, the histogram contained 72

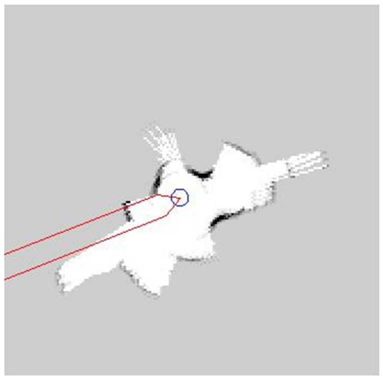
Figure 4.1: The global certainty map of the simulated environment that the quadrotor is navigating.

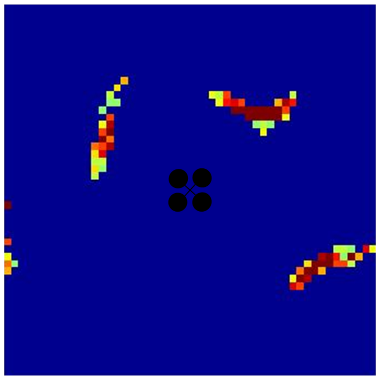
Figure 4.2: The active window centered around the quadrotor. Near obstacles corresponding to the dark areas on the global map are represented on the color plot with the colors representing each cell's respective certainty magnitude as defined in (4.4).

sectors with angular width $\alpha = 5^o$. Within each sector, a polar obstacle density h_k is calculated as defined by

$$h_k = \sum_{i,j} m_{i,j}. \tag{4.5}$$

Then, since data from the sonar is spurious and the possibility of empty cells within an obstacle is likely, a smoothing function

$$h_k^{'} = \frac{h_{k-l} + 2h_{k-l+1} + \cdots + lh_k + \cdots + 2h_{k+l-1} + h_{k+1}}{2l+1} \tag{4.6}$$

is applied to eliminate small gaps in obstacle data. Finally, each of the 72 sectors is deemed clear or obstructed based on a threshold value. Looking at Figure 4.3, any sector that has a density value extending above the horizontal threshold line is considered obstructed and not a viable direction of travel, whereas any sector with a density value below the horizontal line is a safe direction of travel, leaving the quadrotor with information analogous to Figure 4.4.

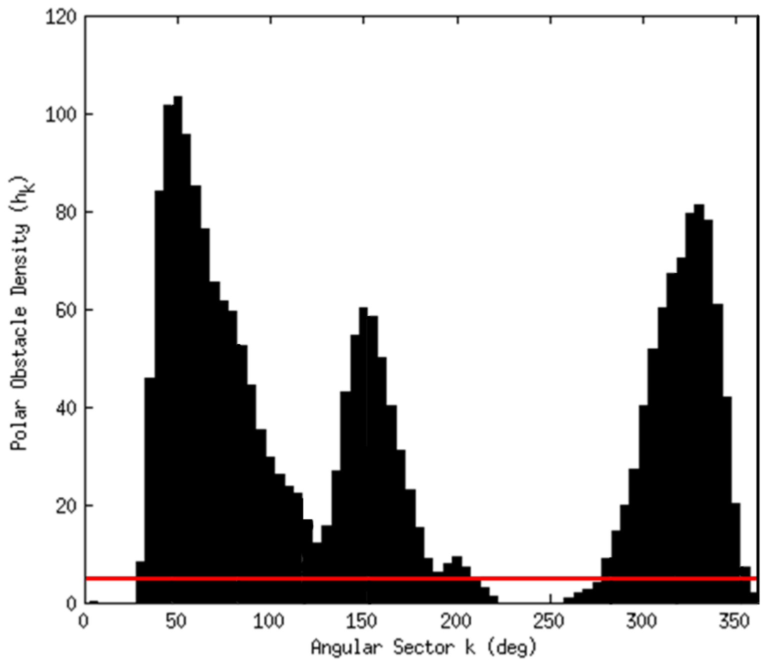

Figure 4.3: Histogram representing the Polar Obstacle density (y-axis) in each angular sector relative to the map horizontal axis (x-axis). The bars that extend beyond the red threshold line indicate that a specific sector is blocked, whereas the bars that do not reach the red threshold line indicate that a specific sector is clear.

Upon discovering clear and blocked areas, the vehicle needs to analyze specific candidate directions to travel. Therefore, each clear area is classified as either a narrow or a wide opening, then treated accordingly. If an opening has at least s_{max} consecutive sectors clear, then the opening is considered wide, otherwise the opening is considered narrow. In these experiments $s_{max} = 18$, meaning that a wide opening is at least 90^o. If an opening is classified as narrow, then that opening provides only one candidate direction, directly down the middle, in order to keep acceptable spacing from obstacles. However, if the opening is classified as wide, then the opening provides up to three possible candidate directions: one direction near the left boundary, one direction near the right boundary, and if the target direction is contained within the opening, directly towards the target direction is also an option. The purpose in having

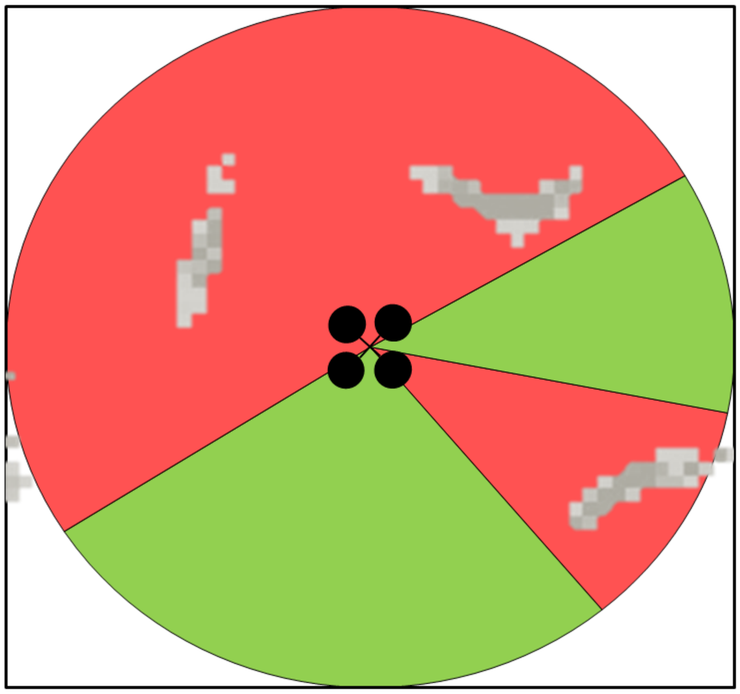

Figure 4.4: A graphical representation of the polar histogram in Figure 4.3. The green colored areas indicate clear path and are represented in Figure 4.3 by bars below the threshold line. The red colored areas indicate blocked paths and are represented in Figure 4.3 by bars extending beyond the threshold line.

directions that stay near to the boundary is that wide openings usually exist when only one side of the vehicle is blocked. By forcing the vehicle to move along one side of the opening, the vehicle remains a constant distance from the blocking feature, resulting in a smoother, faster path than if the vehicle simply moved directly through the center of the open space.

After identifying all possible candidate directions of motion, the VFH algorithm chooses the direction that least deviates from the target direction and travels for one time step before re-analyzing and choosing a new direction. A flow chart describing the VFH algorithm shown in Figure 4.5.

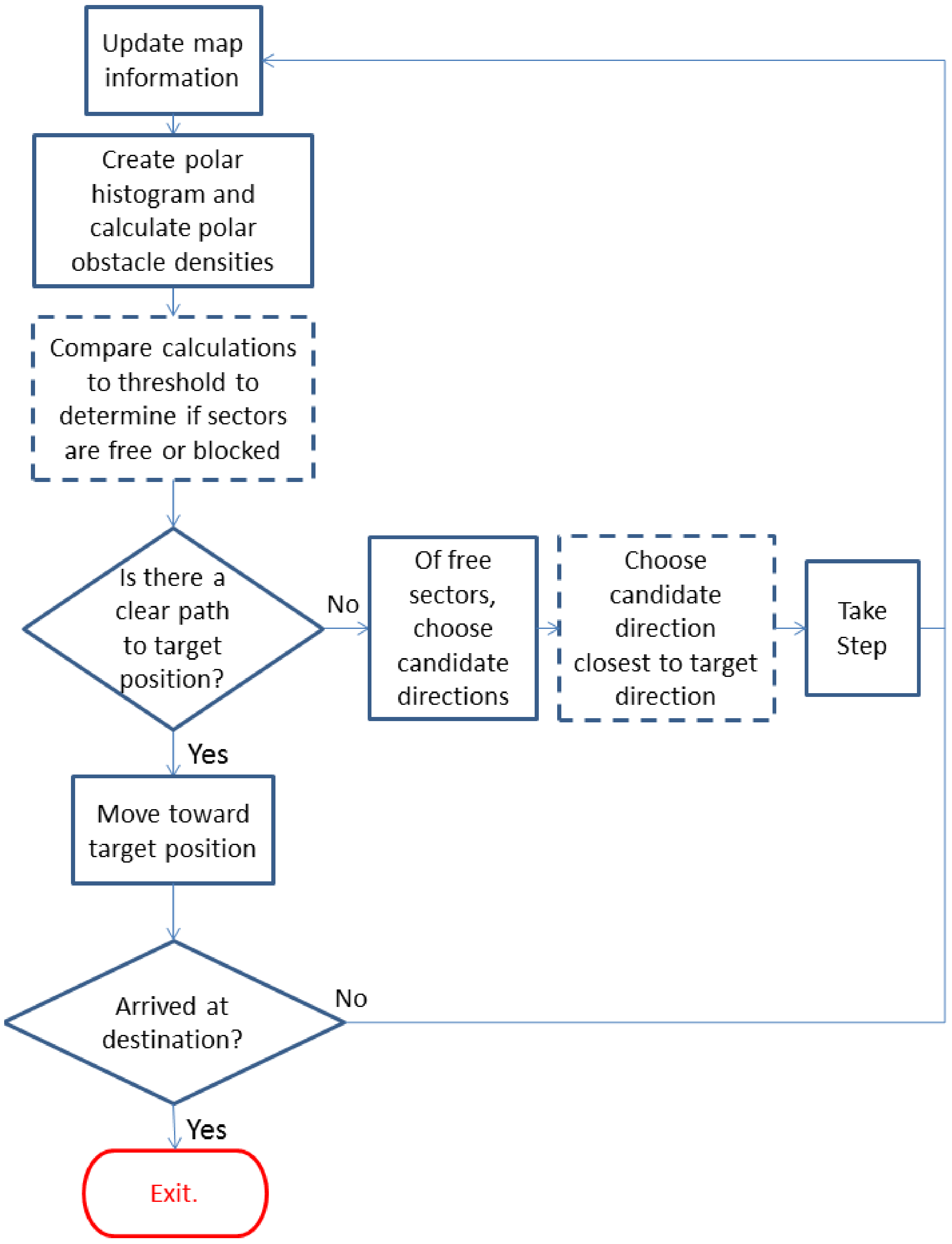
Update map information
Create polar histogram and calculate polar obstacle densities
Compare calculations to threshold to determine if sectors are free or blocked
Is there a clear path to target position?
No
Of free sectors, choose candidate directions
Choose candidate direction closest to target direction
Take Step
Yes
Move toward target position
Arrived at destination?
No
Yes
Exit.

Figure 4.5: Flow chart describing the original VFH algorithm. The red box indicates exiting the algorithm once the quadrotor has reached its goal destination. The dashed boxes indicate differences from the VFH* algorithm in Figure 4.9

4.2 VFH Algorithm*

The VFH algorithm proved to be moderately successful, but it had enough shortcomings that someone familiar with the algorithm could predictably cause the algorithm to fail. Most notably, the VFH algorithm often became trapped in corners due to the algorithm only making decisions based on the local active window information. Additionally, the algorithm suffered greatly due to indecisiveness. If two candidate directions were nearly the same distance from the target direction, the quadrotor had a tendency to chatter. Chattering can be a serious problem if the two candidate directions are significantly different from each other or if an obstacle happens to be in between them. With the VFH algorithm, the quadrotor occasionally collided with obstacles because the decided direction of travel kept changing, effectively sending the vehicle directly towards the obstacle.

The VFH* algorithm developed, and more thoroughly explained, by [17] and [13] mitigates the problems that the VFH algorithm encountered with several modifications. First, the smoothing function described in the VFH algorithm was cumbersome, difficult to tune, and often blocked areas that did not have obstacles. Instead, as described in [17], in order to fill potential gaps in obstacles and still provide a safe distance for the quadrotor (that is represented as a point object), each obstacle cell is constructed to be larger than the radius of the vehicle, thereby creating a buffer around each obstacle cell large enough to prevent any collisions. In order to enlarge each cell, an enlargement angle, $\gamma_{i,j}$, is defined relative to its proximity to the quadrotor, $d_{i,j}$, and the radius of the enlarged obstacle, r_{obs}. These three terms are defined as,

$$d_{i,j} = \sqrt{|y_j - y_0|^2 + |x_i - x_0|^2} \tag{4.7}$$

$$r_{obs} = r_{quadrotor} + r_{buffer} \tag{4.8}$$

$$\gamma_{i,j} = \arcsin(\frac{r_{obs}}{d_{i,j}}), \tag{4.9}$$

where (x_i, y_j) are the x and y coordinates of each obstacle relative to the lower left hand corner of the active window, (x_0, y_0) are the x and y coordinates of the quadrotor in the active window $(x_0, y_0 = \frac{w_s}{2} + 1)$, $r_{quadrotor}$ is the radius of the quadrotor, and r_{buffer} is the minimum distance that the quadrotor should remain from obstacles at all times. Next, the polar obstacle density is computed similar to the original VFH algorithm with the equation:

$$H_k^p = \sum_{i,j} m_{i,j} h'_{i,j}. \tag{4.10}$$

In this equation, $h'_{i,j}$ is a binary operator indicating the existence of each obstacle in the specified sector as defined by:

$$\begin{aligned} h'_{i,j} &= 1 \text{ if } k\alpha \in \left[\beta_{i,j} - \gamma_{i,j}, \beta_{i,j} + \gamma_{i,j}\right] \\ h'_{i,j} &= 0 \text{ otherwise,} \end{aligned} \tag{4.11}$$

where $\beta_{i,j}$ is the angular direction from the quadrotor to an obstacle,

$$\beta_{i,j} = \arctan\left(\frac{y_j - y_0}{x_i - x_0}\right). \tag{4.12}$$

Once the polar obstacle density is calculated for each sector, threshold values are again applied in a similar manner to the VFH algorithm to decide whether or not a sector is clear or blocked. However, rather than selecting a single threshold value and basing the status of the sectors on their relation to that value, a binary polar histogram is created based on a high threshold value, τ_{high}, and a low threshold value,

τ_{low}. The binary polar histogram is then formed by:

$$\begin{aligned}
H^b_{k,n} &= 1 \text{ if } H^p_k > \tau_{high} \\
H^b_{k,n} &= 0 \text{ if } H^p_k < \tau_{low} \\
H^b_{k,n} &= H^b_{k,n-1} \text{ otherwise.}
\end{aligned} \tag{4.13}$$

The benefit of using two thresholds is apparent when navigating through narrow passages. With a single threshold, a narrow passage can alternate between blocked and open from time step to time step. By being able to set a low and a high threshold, the user is more able to ensure that obstacles remain marked as blocked, and clear zones are not falsely obstructed.

After the binary polar histogram is complete, the final stage in the histogram building process is implemented to further reduce the available directions of motion. Unlike most rolling robots, the quadrotor can hover and yaw in place, but coming to a complete stop for every turn is extremely inefficient. As a result, the quadrotor has a specified maximum velocity that is limited by how quickly the sonars can collect and make sense of environmental data. In simulations, the maximum velocity that the quadrotor could travel without collision issues due to lack of data was 5 ft/sec. Furthermore, in order to maintain reasonably accurate values for the dead reckoning controller, all rotational rates are kept small. The maximum turn rate was set to 60 deg/sec, but it can easily be changed to suit experimental discoveries. With a set maximum velocity and turn rate, a turn radius, r_t, can be established. Knowing the turn radius of the quadrotor becomes important when an open candidate direction to the left or right requires that the quadrotor maneuver past an obstacle as shown in Figure 4.6. Based on turn radius, the VFH* algorithm checks both left and right hand turns to see if either direction is blocked.

In order to calculate whether or not a direction is blocked, one must first calculate

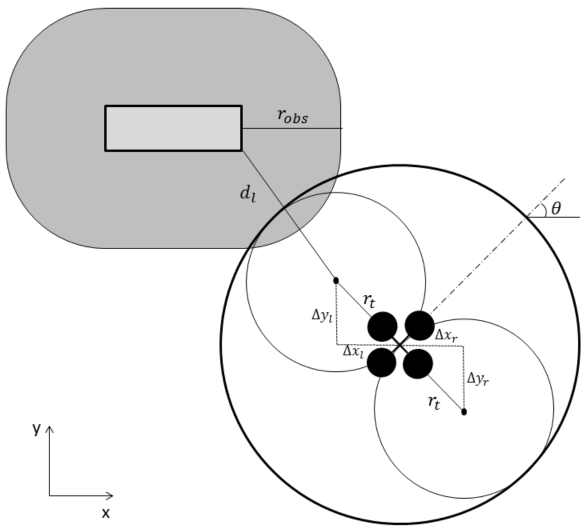

Figure 4.6: A representation of a quadrotor navigating near an obstacle to the left and free to the right. Based on the quadrotor's turn radius at its current velocity, if the quadrotor made a hard left turn, it would within r_{obs} of the obstacle. Therefore, all motion to the left of the obstacle is unattainable at the quadrotor's current speed.

the relative locations of the left and right turn radius centers using:

$$\begin{aligned}
\Delta x_r &= r_t \sin(\theta) \\
\Delta y_r &= r_t \cos(\theta) \\
\Delta x_l &= -r_t \sin(\theta) \\
\Delta y_l &= -r_t \cos(\theta),
\end{aligned} \tag{4.14}$$

where θ is the quadrotor heading relative to the horizontal axis. Next, the distances from the left and right centers, $(\Delta x_l, \Delta y_l)$ and $(\Delta x_r, \Delta y_r)$, to each occupied cell in the active window are calculated by:

$$\begin{aligned}
d_r^2 &= (\Delta x_r - \Delta x(j))^2 + (\Delta y_r - \Delta y(i))^2 \\
d_l^2 &= (\Delta x_l - \Delta x(j))^2 + (\Delta y_l - \Delta y(i))^2.
\end{aligned} \tag{4.15}$$

A left turn is blocked if, for any obstacle,

$$d_l^2 < (r_t + r_{obs}), \tag{4.16}$$

and a right turn is blocked if, for any obstacle,

$$d_r^2 < (r_t + r_{obs}). \tag{4.17}$$

If obstacles are detected to the left or right that will interfere with the quadrotor's turn, a limit angle must be determined. First, an angle ϕ_b is calculated, indicating the opposite direction of the quadrotor's current heading. Upon calculating ϕ_b, two limit angles ϕ_l and ϕ_r are defined and initially set equal to ϕ_b, indicating the turn limit is 180^o in either direction. Next, if an obstacle's angle, $\beta_{i,j}$, is to the left of the quadrotor heading, θ, and to the right of ϕ_l, then (4.16) is checked. If the condition holds, then ϕ_l is set to $\beta_{i,j}$. Conversely, if an obstacle's angle, $\beta_{i,j}$, is to the right of the quadrotor heading, θ, and to the left of ϕ_r, then (4.17) is checked. If the condition holds, then ϕ_r is set to $\beta_{i,j}$. With a new value for ϕ_l and ϕ_r, the final polar histogram can be constructed with the boolean:

$$\begin{aligned} &H_k = 0 \text{ if } H_k^b \text{ and } k\alpha \in \big\{ \big[\phi_l, \theta\big], \big[\phi_r, \theta\big] \big\} \\ &H_k = 1 \text{ otherwise.} \end{aligned} \tag{4.18}$$

To understand the process graphically, Figure 4.7 shows ϕ_l and ϕ_r based on an obstacle blocking a left turn. As shown, ϕ_r is directly behind the quadrotor's direction of travel, indicating all turns to the right are safe. However, ϕ_l is limited to the point where the left turn circle intersects with the enlarged obstacle. Therefore, all paths between ϕ_l and ϕ_b, regardless of whether or not an obstacle is present, are marked as blocked paths.

Finally, with the completed polar histogram, the candidate directions are chosen in the same manner as the VFH algorithm. However, the VFH* algorithm chooses a best path in a manner that minimizes the indecisiveness and trapping issues found

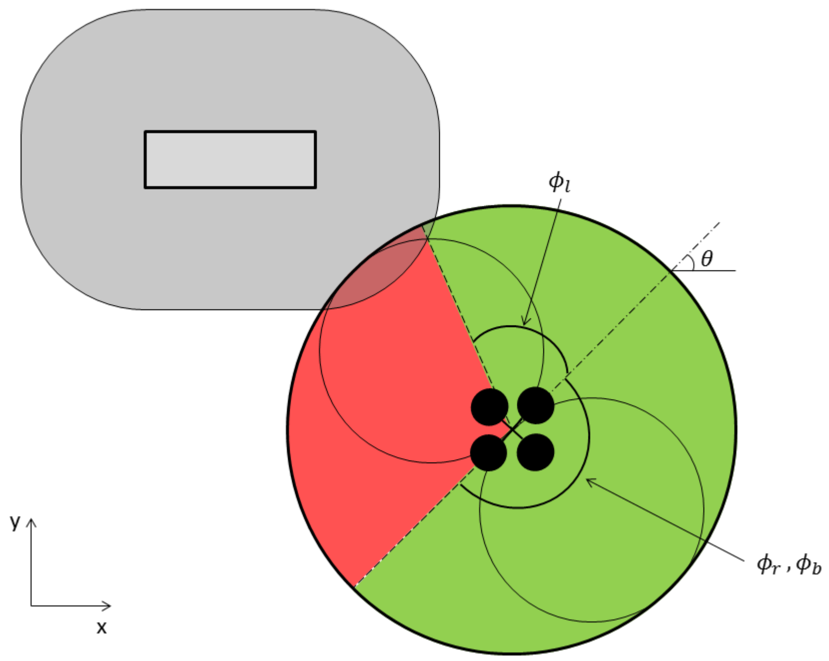

Figure 4.7: A demonstration of the quadrotor's viable options when navigating around the obstacle to the left. Any direction between ϕ_l and ϕ_b is unattainable at the quadrotor's current velocity. Therefore, the candidate directions to the quadrotor's left and behind the obstacle are marked in red. Viable candidate directions are marked in green.

in the original VFH algorithm. Simply moving in the direction closest to the target direction made the quadrotor too goal focused, so in order to allow the quadrotor to move in any direction, even if that direction is opposite from the ultimate goal, VFH* utilizes a variation of an A* search algorithm. As described in [18], A* is a greedy best-first search algorithm, meaning that the algorithm first searches the path of lowest expected total cost from the starting point, then deviates only if another path is estimated to have a lower cost. The A* algorithm considers global map information, not just the data contained in the active window. The A* search algorithm does not produce a globally optimal path; finding an optimal path is an NP-complete problem as explained in [18], meaning that there is no known polynomial solution. Furthermore, since the known environment map changes with each time step, calculating an optimal path at each time step is unnecessary. In order to decide the best path to follow, the A* algorithm is compromised of two separate functions: a cost function that represents the actual cost for the UAV to travel from its initial position to the position of the node n that is currently being analyzed, and a heuristic function that estimates the cost from node n to the goal destination. The cost function is computationally intensive, whereas the heuristic function is generally a "quick-and-dirty" estimate of the remaining cost to reach the goal. Choosing a proper heuristic function is the key to the success of the A* algorithm. A poorly chosen heuristic can result in exponential computation time, while a well chosen heuristic can result in polynomial computation time. The sum of the cost and heuristic functions,

$$f(n) = g(n) + h(n), \tag{4.19}$$

where $g(n)$ is the cost function and $h(n)$ is the heuristic function, estimates the lowest cost to travel from an initial point to the goal destination passing through node n. Initially $f(n)$ is dominated by $h(n)$, but as n increases towards the target node, $h(n)$ decreases and $f(n)$ becomes dominated by $g(n)$.

The A* algorithm utilized in the VFH* algorithm is slightly different from traditional A* because rather than basing the cost and heuristic functions on nodal distance from the target, the VFH* algorithm bases its cost and heuristic functions on direction. Secondly, due to the dynamic nature of the algorithm's map building process, calculating an A* path all of the way to the target destination would be wasteful because an obstacle will very likely block the initial path. As a result, the VFH* algorithm only calculates the lowest cost path for the first n_g steps before executing and reanalyzing. The cost of each candidate direction at the starting point (i.e. when $n = 0$) is defined by the initial cost function

$$g_0(c_0) = \mu_1 \Delta\big(c_0, k_t\big) + \mu_2 \Delta\big(c_0, \frac{\theta_n}{\alpha}\big) + \mu_3 \Delta\big(c_0, k_{d,n-1}\big), \tag{4.20}$$

with

$$\Delta(c_1, c_2) = \min\left\{|c_1 - c_2|, |c_1 - c_2 - s_{max}|, |c_1 - c_2 + s_{max}|\right\}. \tag{4.21}$$

In the cost function, c_0 is the candidate direction. The first term is the cost associated with the candidate direction's deviation from the target direction k_t. The second term is the cost associated with the candidate direction's deviation from the quadrotor's current orientation, θ_n. The final term is the cost associated with the candidate direction's deviation from the previously chosen direction, $k_{d,n-1}$. Effectively, the first term penalizes the quadrotor for deviating from the target direction, and the latter two terms penalize the quadrotor for control changes because changing course is cost prohibitive. Furthermore, μ_1, μ_2, and μ_3 are weighting factors for each term. In order for the quadrotor to remain goal oriented,

$$\mu_1 > \mu_2 + \mu_3.$$

Since A* is a best-first search method, the candidate direction with the lowest initial cost function proceeds to the next step.

At subsequent steps beyond the starting position, a slightly modified cost function is used. The modified cost function is defined as:

$$g_n(c_n) = \lambda^i\left[\mu_1'\max\{\Delta(c_n, k_t), \Delta(k_e, k_t)\} + \mu_2'\Delta(c_n, \frac{\theta_i}{\alpha}) + \mu_3'\Delta(c_n, c_{n-1})\right]. \tag{4.22}$$

The modified cost function is similar to the initial cost function, but it does have minor changes. First, λ^i, $0 < \lambda^i \leq 1$, serves as a weighting factor that allows the costs associated with steps later in the quadrotor's trajectory to be weighted less. Since the quadrotor will likely choose a new A* path in the next time step, weighting later cost functions with lower values allows for the initial cost function to have more importance in deciding the quadrotor's ultimate trajectory. Furthermore, the first term still represents the cost associated with deviating from the target direction, but a new term, k_e, defined as

$$k_e = -\arctan\left(\frac{y_{i+1} - y_i}{x_{i+1} - x_i}\right) \tag{4.23}$$

is called the effective direction of motion. While c_n represents the candidate direction that the quadrotor would ideally be traveling, the quadrotor requires a finite amount of time to move from its current direction of travel to the candidate direction of travel. The variable k_e represents the quadrotor's current direction of travel at any given time step based on the chosen candidate direction. In the projected cost function, one must account for current direction of travel, not just the ideal direction of travel. Otherwise, the candidate direction, c_n, that is closest to the target direction may be selected even though in reality the quadrotor may take an excessive number of time steps to reach c_n. The time delay required to reach c_n might make the candidate direction less appealing. Figure 4.8 shows an example in which the quadrotor might choose the worst candidate direction if the effective direction of travel was not considered. If the quadrotor just looked at c_n, then the cost would be lowest to choose the direction directly towards the target. However, since the quadrotor would not be able to effectively turn to fit through the gap in obstacles, one of the directions to the outside of the obstacles would be a better choice.

Instead of causing the quadrotor to be more decisive like the initial cost function, the last two terms in the projected cost function cause the quadrotor to choose a smooth path by relating the current candidate direction c_n to the current orientation $\frac{\theta_i}{\alpha}$ and the previous candidate direction c_{n-1}. Again, in order to make the cost function goal oriented,

$$\mu'_1 > \mu'_2 + \mu'_3.$$

Summing (4.20) and (4.22) for each projected step i produces the total $g(n)$ to estimate the cost.

Choosing a proper and admissible heuristic function is key for both accuracy and computational efficiency. The heuristic must allow for quick calculation, but it must also provide an accurate estimate of the lowest cost path to follow. As a rule, the heuristic should produce a value close to the actual cost of a specific path, but it

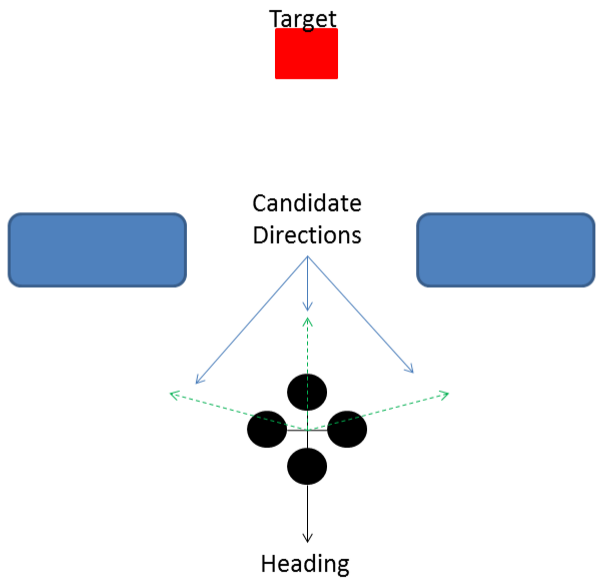

Figure 4.8: A graphical explanation of the importance of considering the effective direction of travel in the projected cost function. If the projected cost function only considered the candidate directions, c_n, then the lowest cost direction would be directly towards the target. However, since the quadrotor in facing the opposite direction, the best choice would probably be to pass around the outside of one of the obstacles.

must never overestimate that cost. With an optimal, but inefficient, heuristic the computational complexity is exponential in comparison to the input. However, with a well chosen heuristic, an admissible path can be calculated with only polynomial computational complexity. Since performing real time calculations was key to the research project, optimal path planning was sacrificed for computational efficiency. A simplifying assumption was made to the heuristic function that the quadrotor would always be able to head directly towards the target direction at the following node.

With this assumption, c_n was set equal to k_t, resulting in the heuristic

$$h(n) = \lambda^i \Big[\mu_2' \Delta\big(k_t, \frac{\theta_i}{\alpha}\big) + \mu_3' \Delta(k_t, c_{n-1})\Big]. \tag{4.24}$$

Since the simplifying assumption always assumes a best case scenario, the heuristic never overestimates the cost. Furthermore, this heuristic only estimates the cost of moving to the next branch, not all of the way to the goal destination. Summing the heuristics from all future branches would result in a more realistic cost estimate, but would also be much more computationally expensive. Since these two simplifications never overestimate the cost to reach the goal destination, the result is an admissible and computationally efficient heuristic function. Inserting $g(n)$ found above and $h(n)$ into (4.19) provides the estimated lowest cost for each path in the A* algorithm. The lowest value of $f(n)$ is then chosen as the designated path for the quadrotor to follow. A flow chart describing the VFH* algorithm is shown in Figure 4.5. The differences between the VFH and VFH* algorithms are indicated by the dashed boxes in the charts.

Since the A* algorithm analyzes the global map, VFH* usually steers the quadrotor away from situations where the entire polar histogram is blocked, but the nature of a constantly changing map can lead the quadrotor into unforeseen problems. However, the quadrotor is always able to escape from trapped situations due to its ability to hover and yaw in place. The goal is to keep the quadrotor moving at a reasonable velocity, so the quadrotor does not slow down unless all viable directions are blocked. In the case that the quadrotor does become trapped, the algorithm commands a full stop and a 180^o turn in place. This maneuver allows the quadrotor to back out of a tight space and approach again from another position. Simulations show that the quadrotor consistently reaches its goal destination even when initially getting trapped when using this method.

While the A* algorithm is effective in finding a successful path to the goal destination, A* is, at best, a polynomial computational cost. Therefore, the A* algorithm is only implemented when an obstacle appears in the active window between the quadrotor's current position and the target destination. As a result, the vehicle reacts more slowly to upcoming obstacles than it possibly could, leading to suboptimal paths, but at the benefit of a more computationally efficient path planning algorithm.

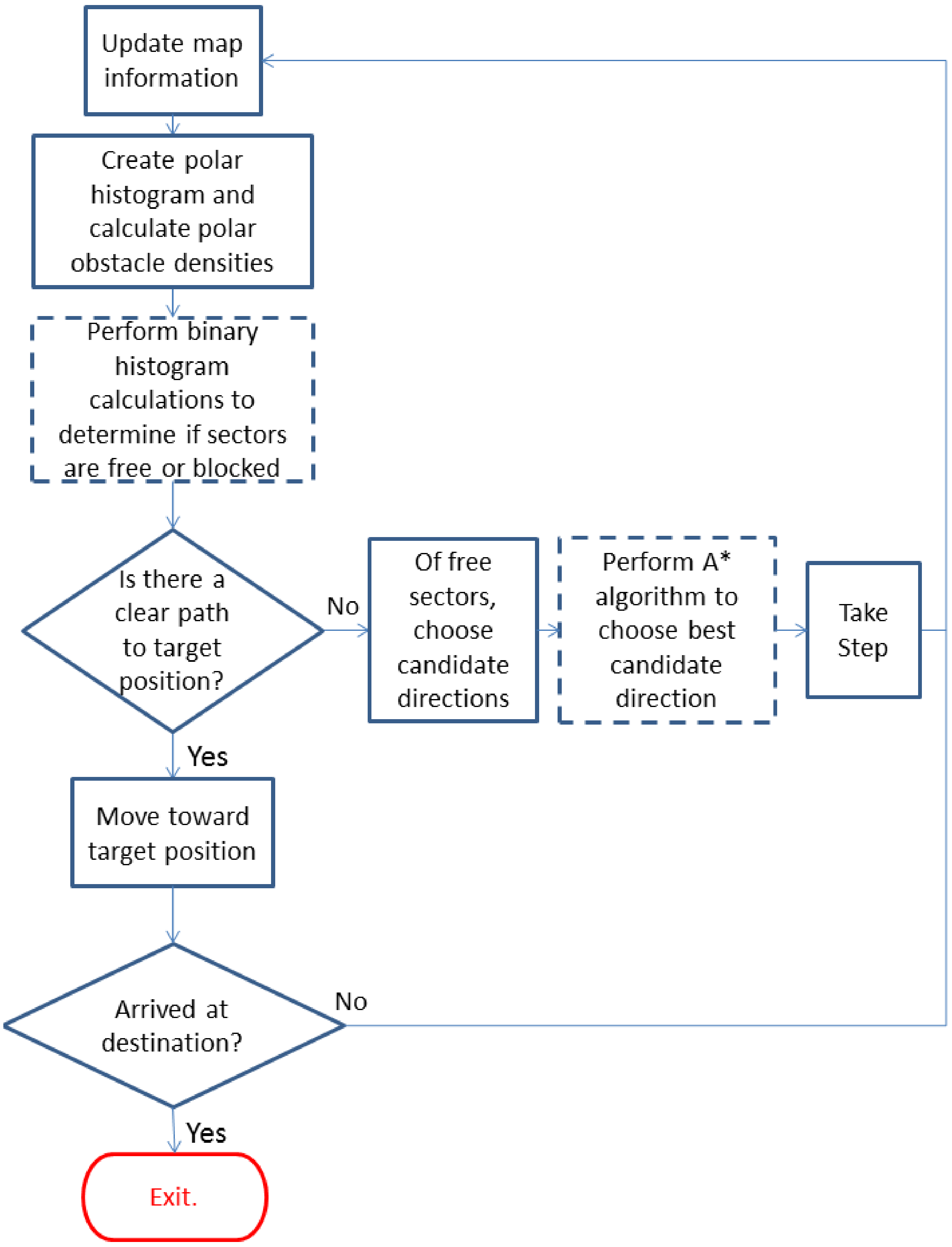
Update map information
Create polar histogram and calculate polar obstacle densities
Perform binary histogram calculations to determine if sectors are free or blocked
Is there a clear path to target position?
No
Of free sectors, choose candidate directions
Perform A* algorithm to choose best candidate direction
Take Step
Yes
Move toward target position
Arrived at destination?
No
Yes
Exit.

Figure 4.9: Flow chart describing the VFH* algorithm. The red box indicates exiting the algorithm once the quadrotor has reached its goal destination. The dashed boxes indicate differences from the original VFH algorithm in Figure 4.5

Chapter 5
SIMULATION AND RESULTS

The VFH and VFH* algorithms were tested in seven different simulated environments, both structured and unstructured. A Monte Carlo method was used in selecting starting and ending locations in order provide a wide variety of unique situations to both algorithms, and their performances were analyzed.

5.1 VFH Simulations

The VFH algorithm was initially simulated to test its viability as a simple path planning algorithm. Figure 5.1, shows an unstructured course for the vehicle to navigate through. Although suboptimal, the path is admissible, especially for a purely local collision avoidance algorithm. However, the quadrotor simulation actually collided with two obstacles during the journey to the target destination. The collision was caused by the quadrotor turning too sharply around obstacles and catching their edges. This problem was generally a result of lack of data directly around a corner and was the most prevalent of those that the VFH algorithm experienced. Figures 5.2 and 5.3 are further examples of the quadrotor attempting to maneuver too closely around corners.

Another area in which the VFH algorithm struggled was starting in close proximity to an obstacle. The VFH algorithm often failed to gain the needed certainty value in time to avoid obstacles quickly. Figure 5.4 shows the quadrotor starting in the northeast corner of the map, between three small obstacles. Aside from the collision near the end of the path due to the problem discussed previously, the quadrotor also collided with the small obstacle near its starting position.

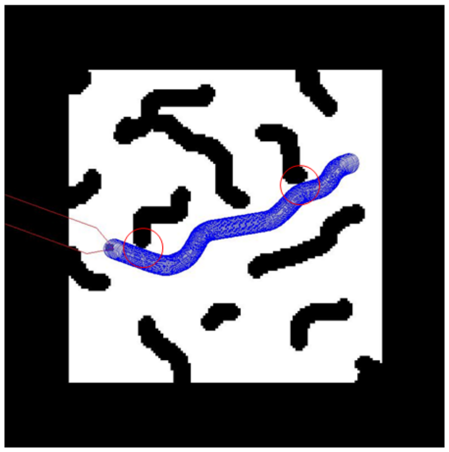

Figure 5.1: A simulation of a quadrotor navigating through an unstructured environment towards a goal destination. The quadrotor takes a relatively efficient course to the goal, but it partially collides with two walls in the process.

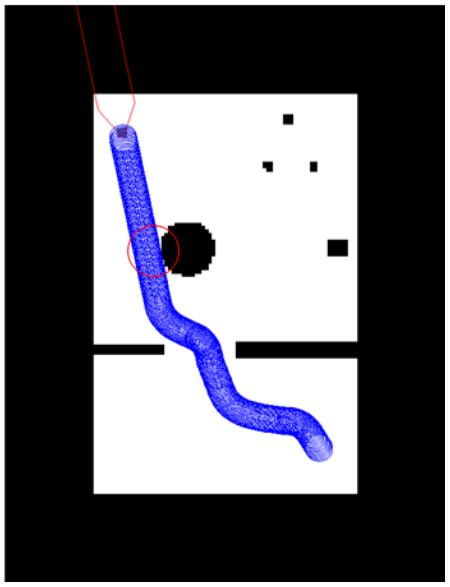

Figure 5.2: Navigating around a room type environment, the VFH algorithm failed to successfully navigate around the round object because the quadrotor maneuvered too close around the obstacle edge.

In another example, the quadrotor was placed in a situation where it approached a wall perpendicularly, as shown in Figure 5.5. Although the wall was short and

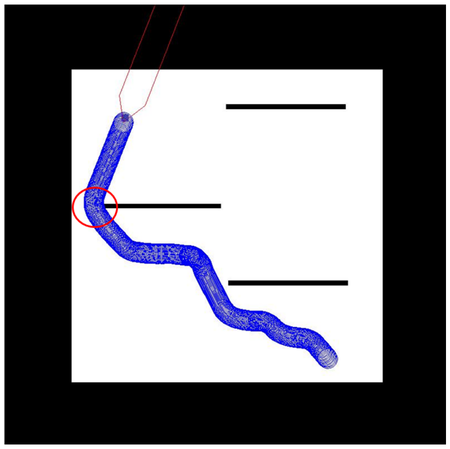

Figure 5.3: A simulation in which the quadrotor needed to navigate around three walls to a target destination. The VFH algorithm tried to pass the wall on the left despite not having enough room and collided with the corner of the obstacle.

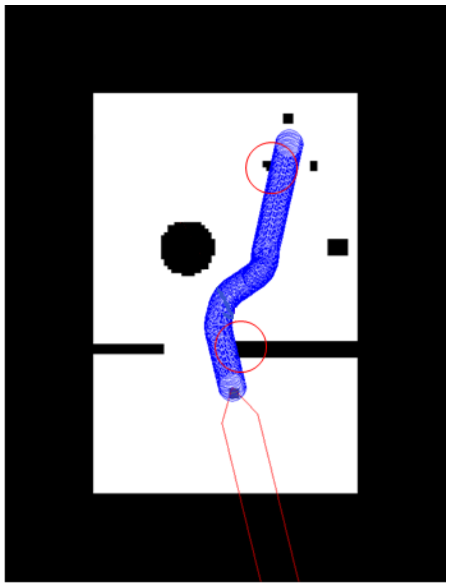

Figure 5.4: Starting in close quarters, the VFH algorithm failed to react quickly enough to avoid the imminent collision.

seemingly easy to navigate, the fact that there were two equally viable paths presented a chatter problem. The VFH algorithm is designed to choose the direction that most closely matches the target direction. Since both the left and right choices were nearly the same angular distance from the target direction, VFH showed indecisiveness, causing the quadrotor to oscillate between both directions. Once the quadrotor moved close enough to the wall, the vehicle did a full revolution before moving enough to one side to safely pass around the wall. The same problem arose in multiple situations.

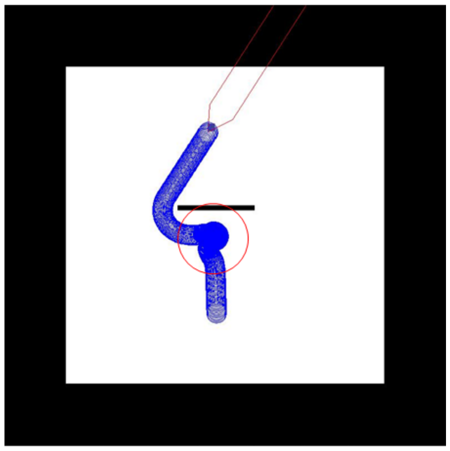

Figure 5.5: A simulation of a quadrotor initially placed on a trajectory perpendicular to a wall. The algorithm displays chatter on whether to pass to the left or right of the wall. The quadrotor eventually does one full circle before moving enough to one side to safely pass around the wall.

On several occasions, the algorithm remained indecisive up to the point of collision with the obstacle.

A final simulation in Figure 5.6 shows the quadrotor's behavior when confronted by a long wall. The path shows that the quadrotor moves back and forth along the wall, retracing its own path. As a result, the quadrotor failed to ever reach either end of the wall and could not safely proceed to the goal destination. Similar situations also appeared when the quadrotor moved into concave corners. The vehicle would initially turn to move out of the corner, but it often returned to the same path, causing the quadrotor to become trapped.

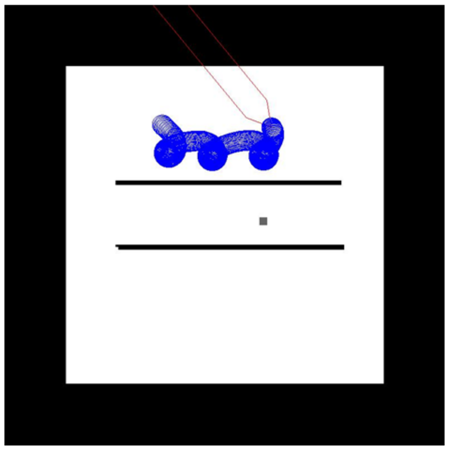

Figure 5.6: A simulation of a quadrotor attempting to pass around a long wall. The quadrotor turns in towards the wall several times due to lack of environmental information before realizing that the wall continues. Due to a very target oriented decision algorithm along with the availability of only local data, once the quadrotor passes the vertical plane of the target in the photo, the algorithm orders a change of direction. As a result, the quadrotor never discovers the path to the right of the wall.

5.2 VFH* Simulations

Upon analyzing the results from the VFH algorithm, it was determined that a more robust path planning algorithm would need to be utilized in order to increase the success rate of reaching the target destination. The changes made to the VFH* algorithm alleviated the problems discovered in VFH. The following simulations were performed with $n_g = 5$ steps to ensure adequate consideration of the global map to successfully navigate around the simulated environments without sacrificing excessive computational efficiency. Figures 5.7, 5.8 and 5.9 show the quadrotor beginning in the same positions in their respective maps, with the same goal destinations as the VFH algorithm. The quadrotor takes similar paths to the target destinations as VFH, but maintains proper spacing to safely maneuver around the corners that into which the original VFH collided.

Secondly, the VFH* algorithm performed well when starting in confined spaces.

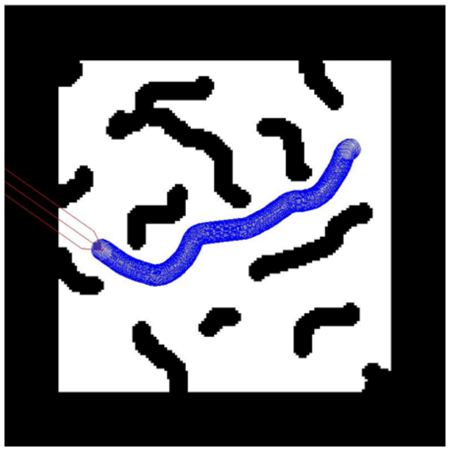

Figure 5.7: A simulation of a quadrotor navigating an unstructured environment using the VFH* search algorithm. Unlike the VFH algorithm, the quadrotor safely passes around the obstacles without cutting corners.

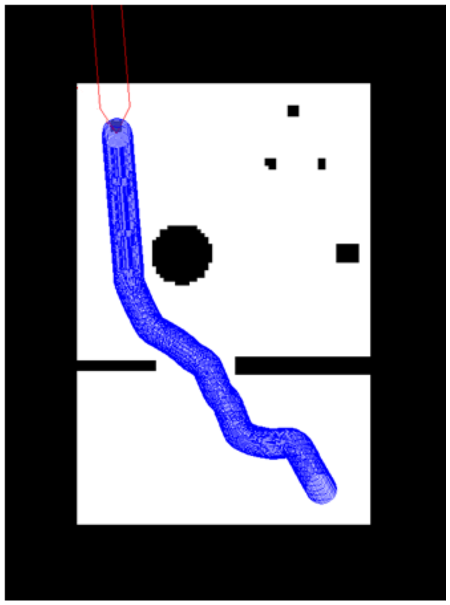

Figure 5.8: Navigating around a room type environment, the VFH* algorithm successfully navigated around all obstacles with proper spacing.

Even when beginning very close to the small obstacle in the northeast corner of Figure 5.10, the quadrotor reacted appropriately and moved through the gap.

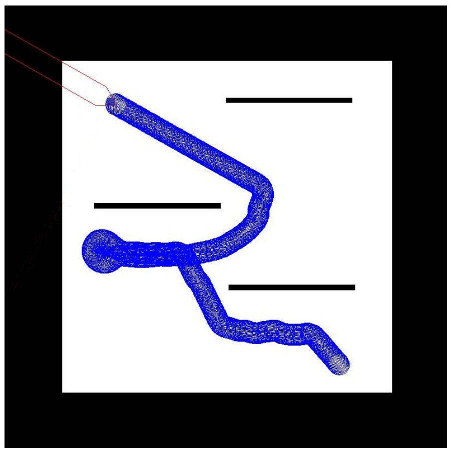

Figure 5.9: A simulation in which the quadrotor needed to navigate around three walls to a target destination. The VFH* algorithm made one pass towards the opening to the left before deciding that the gap was too small proceeding around the right side of the wall.

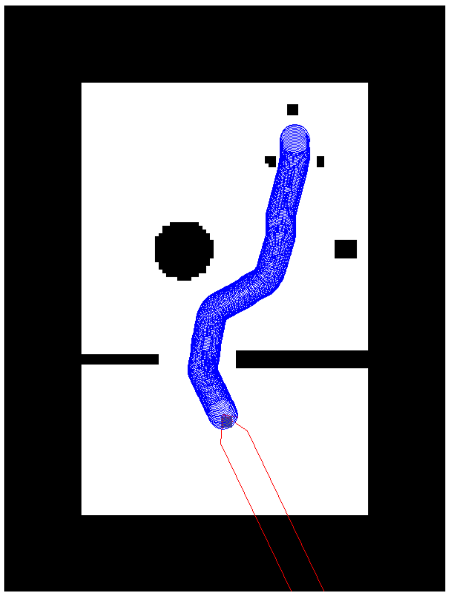

Figure 5.10: Even when beginning in close quarters, the VFH* algorithm is able to react quickly enough to safely pass around the obstacle immediately in front of it.

When analyzing the VFH* algorithm's behavior after initializing the vehicle to begin perpendicular to a wall, the quadrotor follows a much more appropriate path than the original VFH algorithm. Although the quadrotor did not begin to evade the wall until after the obstacle appeared in the active window, once a course correction was made to move around the wall, the quadrotor decisively maintained that direction. The result was an efficient maneuver around the wall as shown in Figure 5.11.

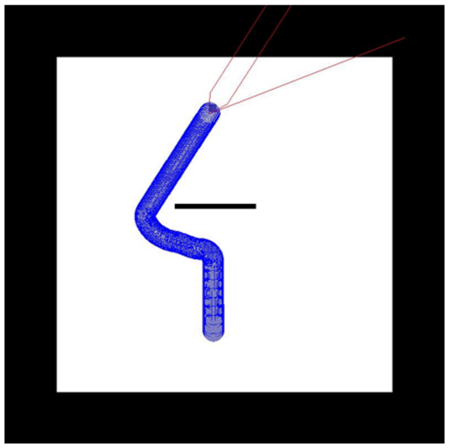

Figure 5.11: A simulation of the VFH* algorithm with a quadrotor initially placed on a trajectory perpendicular to a wall. The algorithm displayed much more directional decisiveness than the original VFH algorithm and efficiently passed around the wall before proceeding to the target destination.

In the final simulation, the VFH* algorithm maneuvered around a long wall to reach the target destination. While the original VFH algorithm never reached the opening to the side of the wall, VFH* was able to navigate around the wall. Figure 5.12 shows the quadrotor's path using the VFH* algorithm (highlighted in red to follow more easily). Using VFH*, the quadrotor did double back once. However, when the quadrotor returned from a slightly different angle, the opening was discovered and the quadrotor successfully navigated around the wall.

5.3 VFH/VFH Comparison*

The VFH algorithm was successful in sparsely populated environments and environments without concave corners. However, even when presented with relatively

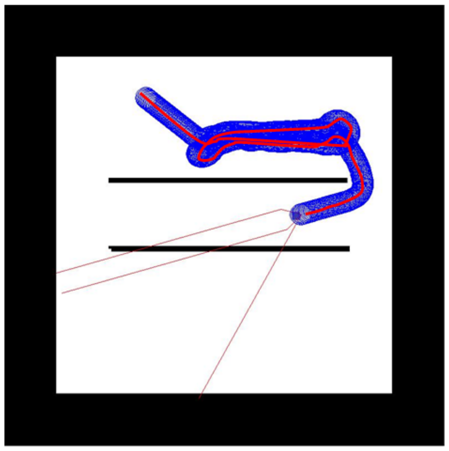

Figure 5.12: A simulation in which the quadrotor needed to maneuver around a long wall. An additional outline of the path is marked in red to better understand the quadrotor's movement. The quadrotor initially followed the wall until reaching the corner, then it doubled back. Upon realizing how far the wall extended to the left, the quadrotor returned to the right from a different angle and was able to proceed around the wall.

few obstacles, the VFH's shortcomings became noticeable. The VFH* algorithm addressed many of the problems associated with the original VFH algorithm, and the improvements are evident in looking at their respective paths. When comparing their paths, both algorithms were suboptimal, but optimality comes at the cost of greater computational complexity, and a truly optimal path is unattainable without full map knowledge. Therefore, both algorithms are admissible for their purpose as a real time collision avoidance and mapping algorithm when they reach their goal. However, as becomes apparent when comparing maps such as those shown in Figures 5.5 and 5.11, the VFH* algorithm provides a less costly path more often than the VFH algorithm.

The first problem clearly shown in Figure 5.1 is the VFH algorithm's tendency to pass too closely around obstacles and to frequently collide with the edges of walls. The problem stems from the smoothing function in (4.6). Since the quadrotor is simulated as a point, the algorithm relies on enlarging the perceived size of the obstacles to prevent collisions. However, since the smoothing function relies on the polar obstacle density of neighboring sectors, if the sectors' magnitudes are not great enough, the quadrotor tends to "clip" the corners of obstacles. By removing the smoothing function, and using the obstacle enlargement equation defined in (4.9), the quadrotor has an easily adjustable buffer around obstacles, which prevents the frequent collisions due to cutting corners in the VFH algorithm.

The smoothing function also leads to the quadrotor's inability to avoid obstacles when starting in close quarters. Because the smoothing function requires high certainty values in order to provide adequate spacing for the point modeled quadrotor, the vehicle often failed to completely avoid immediate threats. Since (4.9) takes effect immediately after an obstacle passes the threshold value, the VFH* algorithm was able to react to obstacles more quickly than the original VFH algorithm.

Another problem that is demonstrated in Figure 5.5 is the VFH algorithm's indecisiveness when presented with multiple paths. Since passing to the left or right of the wall in Figure 5.5 was nearly equivalent, the VFH algorithm made multiple changes in direction approaching the wall, almost leading to a collision. The VFH* algorithm addresses the vehicle indecisiveness concern with the second two terms of the cost function defined in (4.20) and (4.22). Regardless of the direction that is chosen initially, the VFH* algorithm will continue on a similar path unless presented with a compelling reason to change paths (e.g. the sensors detect another obstacle).

Finally, the problem that is most inhibitive to the long term success of VFH is the algorithm's inability to use global map data. As demonstrated in Figure 5.6, the algorithm never allowed the vehicle to deviate far enough from the goal direction to find the openings on the far ends of the map. In contrast, VFH* implements the A* algorithm to incorporate global map data as well as the local active window in order to find viable paths to the goal destination. While the VFH* algorithm did retrace the path towards the center of the map once in Figure 5.12, the reason for this motion was lack of environmental information. On the first pass, the sonar did not gather enough data near the corner of the wall and the wide sonar arc made the opening

Table 5.1: Table showing the success rates of the respective algorithms in each course.

Course	**VFH**	**VFH***
Course A	14/25	24/25
Course B	14/25	24/25
Course C	16/25	24/25
Course D	6/25	25/25
Success Rate	50%	97%

appear too small. Furthermore, the lack of environmental data on the far side of the wall made the algorithm unaware that the wall continued for a long distance on the other side as well. Upon further inspection, the quadrotor ultimately reached the goal destination. Although the path was not desirable, VFH* succeeded where VFH failed.

In comparing the success rates of the VFH and the VFH* algorithms, Table 5.1 shows the results of each algorithm in the four different environments shown in Figure 5.13 with randomly selected starting and ending positions. Success is defined as arriving at the goal destination without any collisions. The table shows moderate success with the VFH algorithm, and very high success with the VFH* algorithm. Of the three failed cases, two were a result of the target being very close to an obstacle, and one was due to an obstacle's perfect placement between sensor angles.

Although the VFH* algorithm is more computationally complex, the algorithm is able to perform one iteration in simulation in approximately 0.15 seconds on the desktop computer with a 3.4 GHz processor using MATLAB. This means that the 3.4 GHz processor can run the simulation in real time at 6.4Hz. While the processor is more powerful on the desktop, the computational efficiency of C++, the coding language used on the quadrotor operating system, is significantly greater than the computational efficiency of MATLAB. Furthermore, the majority of the MATLAB algorithm is

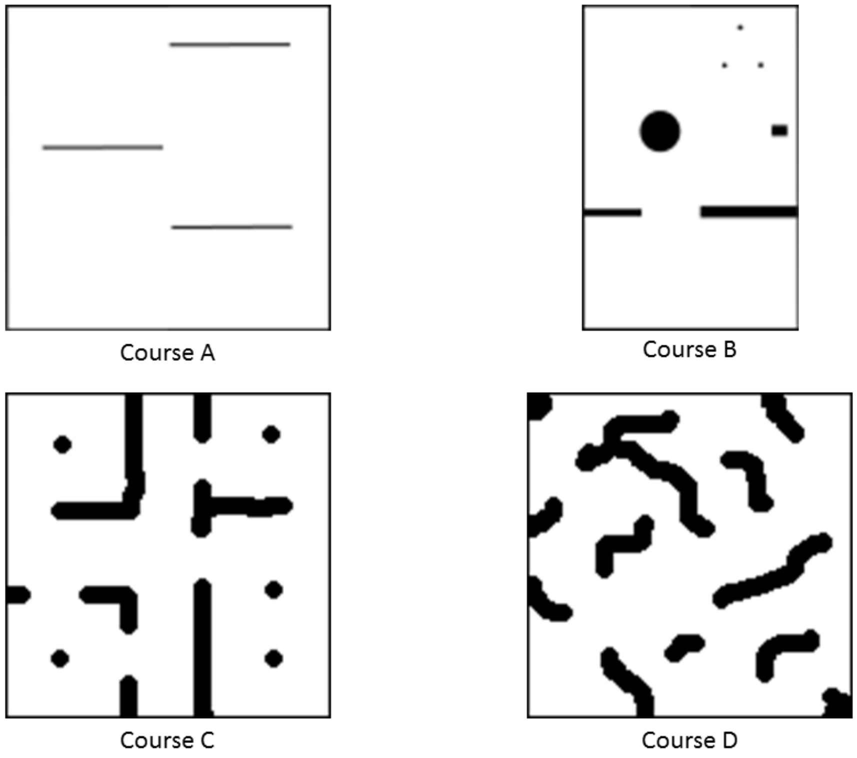

Figure 5.13: The simulated course environments through which the quadrotor navigated through.

dedicated to creating and maintaining the simulation environment. Therefore, the anticipation is that the quadrotor will be able to perform real time calculations at 5Hz.

Chapter 6
CONCLUSIONS AND FUTURE WORK

The results of the VFH algorithm showed a collision avoidance and path planning algorithm that reached a goal destination only in conducive environments. Since the algorithm only utilized data within a window, the path planning algorithm was, at best, inefficient and, at worst, resulted in the quadrotor becoming trapped or colliding with obstacles. With the addition of the enlarged obstacle smoothing method, the introduction of the cost and heuristic functions, and a receding horizon A* algorithm, the resulting VFH* algorithm solves many of the problems discovered in the original VFH algorithm. Furthermore, while less efficient than the VFH algorithm, the VFH* algorithm is efficient enough to perform real time obstacle avoidance and mapping. Therefore, VFH* is a viable option for a path planning algorithm in unknown environments.

The next step is to implement the algorithm on a quadrotor. Implementation involves translating the algorithm language from MATLAB to C++, then testing the effects of real world data in the system. Although noise was added to the simulation data, real world data can have a significantly different impact on the system performance.

An additional algorithm that needs to be implemented on the quadrotor in order to reduce error in the data is a localization algorithm. If the system does not rely solely on a dead reckoning localization process, the odometry error can be accounted for, making mapping easier and more accurate. A possible method for localization is to implement an object association algorithm such as nearest neighbor that associates previously discovered obstacles with obstacles currently being seen, then inputs a pose correction so that the obstacles are aligned.

Another future area to explore would be to expand path planning to three dimensions. Currently, the quadrotor maintains a constant altitude during all transverse maneuvering due to a lack of upward facing sensors to detect potential overhead obstacles. However, if the quadrotor is fitted with upward facing sensors, the path planning algorithm could be expanded to three dimensions, allowing for potentially faster routes to target destinations.

Adding another dimension to the search space would add to the computational consumption of the algorithm, so finding more efficient methods of path planning is also important. The final potential item for future work would be to explore some other methods of path planning. One method that has become increasingly popular recently is the Rapidly exploring Random Tree (RRT) algorithm. This algorithm is potentially more computationally efficient than the VFH* algorithm and merits more research.

As UAVs continue to be incorporated into everyday activities, increasing autonomy will continue to expand their uses. Currently, many critics question the safety of machines operating without human oversight. However, autonomous systems have the advantage of being predictably reliable. This quality gives UAVs the potential to be more successful and safer than human systems with continued advances in technology.

BIBLIOGRAPHY

[1] "Ascending Technologies." http://www.asctec.de/uav-applications/research/products/asctec-hummingbird/.

[2] MaxBotix, "LV-MaxSonar-EZ4 High Performance Sonar Range Finder." http://www.maxbotix.com/documents/MB1040_Datasheet.pdf/, 2012.

[3] T. Eichenseher, "A Global Hawk Turns Hurricane Hunter." http://news.nationalgeographic.com/news/2012/09/pictures/120921-hurricane-drones-nasa-usgs-environment-science/, September 2012.

[4] S. Page, "The Rise of Droids." http://www.strategypage.com/htmw/htairfo/-articles/20080811.aspx, August 2008.

[5] J. Villbrandt, "The Quadrotors Coming of Age." http://illumin.usc.edu/printer/162/the-quadrotors-coming-of-age/, October 2011.

[6] R. Smith, M. Self, and P. Cheeseman, "Estimating Uncertain Spatial Relationships in Robotics," in *Uncertainty in Artificial Intelligence*, vol. 2, (New York), pp. 435 – 461, Elsevier Science, 1988.

[7] P. Moutarlier and R. Chatila, "Stochastic Multisensory Data Fusion For Mobile Robot Location And Environment Modelling," in *Fifth International Symposium on Robotics Research*, pp. 85 – 94.

[8] S. Fu, H.-y. Liu, L.-f. Gao, and Y.-x. Gai, "SLAM for Mobile Robots Using Laser Range Finder and Monocular Vision," in *International Symposium on Robotics Research*, 2007.

[9] A. S. Huang, A. Bachrach, P. Henry, M. Krainin, D. Maturana, D. Fox, and N. Roy, "Visual Odometry and Mapping for Autonomous Flight Using an RGB-D Camera," in *International Symposium on Robotics Research*, 2011.

[10] J. D. Tardós, J. Neira, P. M. Newman, and J. J. Leonard, "Robust Mapping and Localization in Indoor Environments Using Sonar Data," in *The International Journal of Robotics Research*, vol. 21, pp. 311 – 330, April 2002.

[11] J. Choi, S. Ahn, and W. K. Chung, "Robust Sonar Feature Detection for the SLAM of Mobile Robot," in *Intelligent Robots and Systems, 2005. IROS 2005. 2005 IEEE/RSJ International Conference on*, pp. 3415 – 3420, 2005.

[12] J. Borenstein and Y. Koren, "The Vector Field Histogram - Fast Obstacle Avoidance for Mobile Robots," in *IEEE Transactions on Robotics and Automation*, vol. 7, pp. 278 – 288, June 1991.

[13] J. Borenstein and Y. Koren, "VFH*: Local Obstacle Avoidance with Look-Ahead Verification," in *IEEE International Conference on Robotics and Automation*, vol. 3, pp. 2505 – 2511, April 2000.

[14] A. Bry and N. Roy, "Rapidly-exploring Random Belief Trees for Motion Planning Under Uncertainty," in *Proceedings of the 2011 IEEE International Conference on Robotics and Automation (ICRA 2011)*, May 2011.

[15] J. Becker, "Modeling and Control of a Quadroter with Dynamic Inertia," Master's thesis, University of Washington, 2013.

[16] A. Rahmani, K. Kosuge, T. Tsukamaki, and M. Mesbahi, "Multiple UAV Deconfliction via Navigation Functions," in *AIAA Guiadance, Navigation and Control Conference and Exhibit*, August 2008.

[17] I. Ulrich and J. Borenstein, "VFH+: Reliable Obstacle Avoidance for Fast Mobile Robots," in *IEEE International Conference on Robotics and Automation*, vol. 2, pp. 1572 – 1577, May 1998.

[18] T. Crescenzi, A. Kaizer, T. Young, J. Holt, and S. Biaz, "Collision Avoidance in UAVs Using Dynamic Sparse A* Technical Report # CSSE11 - 02." http://www.eng.auburn.edu/programs/csse/research/pubs.html, August 2011.