Real-time roadway mapping and ground robotic path planning via unmanned aircraft

Scott Carson Radford

Thesis submitted to the faculty of the Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science in
Mechanical Engineering

Kevin Kochersberger, Chair
Alexander Leonessa
David C. Conner

August 11th, 2014
Blacksburg, VA

Keywords: UAV, Computer Vision, Image Mosaicking, Path Planning

Real-time roadway mapping and ground robotic path planning via unmanned aircraft

Scott Carson Radford

ABSTRACT

The thesis details the development of computer vision and path planning algorithms in order to map an area via UAV aerial imagery and aid a UGV in navigating a roadway when the road conditions are not previously known (i.e. disaster situations). Feature detection was used for transform calculation and image warping to create mosaics. A continuous extension using dynamic cropping based on newly gathered images was used to improve performance and computation time. Road detection using k-means segmentation and binary image morphing was applied to aerial imagery with image shifting tracked by the mosaicking to develop a large road map. Improvements to computation time were developed using k-means for calibration at intervals and nearest neighbor calculating for each image. This showed a greatly reduced computation time for a series of images with only 1-2% error compared to regular k-means segmentation. Path planning for the UAV utilized a traveling wave applied to the traveling salesman genetic algorithm solution to prioritize close targets and facilitate UGV deployment. Based on the large map of road locations and road detection method, the Rapidly-exploring Random Tree (RRT) algorithm was modified for real-time application and efficient data processing. Considerations of incomplete maps and goal adjustments was also incorporated. Finally, aerial imagery from an actual UAV flight was processed using these algorithms to validate and test flight parameters. Testing of different flight parameters showed the desired image overlay of 50% to give accurate mosaics. It also helped to develop a benchmark for the altitude, image resolution and frequency for flights. Vehicle requirements and algorithm limitations for future applications of this system are also discussed.

DEDICATION

I would like to dedicate this to my mother, Aimee Rogers. All the work I put into this pales in comparison to all the things she has done to support and encourage me throughout the good times and the bad. Thank you, Mom. I love you.

ACKNOWLEDGEMENTS

First, I would like to thank my advisor, Dr.Kochersberger and the Unmanned Systems Lab members for all their help in helping conduct this work and giving me experience in many different facets of robotics, flight mission planning and many others. Dr.Kochersberger helped guide me in making and justifying many strong decisions for my thesis despite my arguments and debating. He also gave me many responsibilities in the lab, which helped me grow and develop the necessary skills I now utilize every day to aid in lab operations and advise others. Justin Stiltner has been an amazingly valuable asset to the lab and myself with all the work that I have done during graduate school. I sincerely thank him for all his help and advising. Matt Frauenthal, Joey Lang and Gordon Christie have also been great friends and colleagues during this whole process. I hope to stay in touch and work with you all again someday.

Second, I would like to thank all of Virginia Tech. The past 5 years at this university have been a great experience. The sense of community, pride and comradery at VT is amazing and I will truly be sad to leave this wonderful place. The strong education, life lessons and great atmosphere that I have had the pleasure of being a part of will leave a lasting mark on me. I am, and forever will be, a Hokie! Go Tech!

TABLE OF CONTENTS

LIST OF FIGURES

1 INTRODUCTION

Unmanned Aerial Vehicles (UAV) and Unmanned Ground Vehicles (UGV) are becoming much more accessible for many applications. Due to their low cost and easily available software, these autonomous systems allow for design and implementation of systems that can aid in many tasks. The use of these systems can reduce human labor costs, risk in dangerous environments and speed of response in disaster situations. The goal of this thesis will take advantage of all these aspects with the goal of developing methods for an autonomous aerial and ground system to collaborate and share information. The major goal is to map unknown or disaster ridden terrain using a UAV in order to guide a UGV through the area to a specific goal. For example, roads and pathways can easily be obstructed after a hurricane, and emergency response vehicles need to know the quickest path that is unobstructed in order to help victims of the storm.

Computer vision systems have been a great focus of research recently and have proven to be an effective tool for evaluating images of environments or objects. Applications have been developed for object identification, image mosaicking, facial recognition, object tracking and much more. Because computer vision is so versatile and many algorithms can be developed, it can aid in the autonomous applications of aerial and ground vehicles by gathering and processing information about the environment. For this system, the use of computer vision will help in mapping the terrain in 2D using image mosaicking as well as road detection for means of UGV navigation. The development of these algorithms and processing of this information is a major focus of this work. Reducing computation time and efficient timing of image processing will be required to share information with both the UAV and UGV.

The major motivation of this thesis is to aid in disaster situations including storms, flooding or even military applications. By using UAVs, aerial imagery can be quickly acquired to map terrain while UGVs or emergency response vehicles can be deployed to navigate to areas in need of aid. By using autonomous systems, the speed of response can be greatly reduced by continually gathering and sharing information in real time to update goals and paths. Real time application should allow for computation during the flight mission without undue delays. This will allow the ground vehicles to be deployed almost immediately after the UAV begins imaging without waiting for data collection to complete.

Below is a list of the goals of the project and requirements for successful development of this system. Although a ground vehicle is required, this research is more focused on the aerial vehicle with development of the computer vision system and information processing. Certain assumptions about the capabilities of the vehicles are discussed and used to aid in the system development.

Project goals include:

Hardware and vehicle considerations: The major means of information gathering is through the use of cameras which are limited by the resolution requirements. The vehicles have certain limitations on speed, control, and autonomy depending on the system chosen. Understanding of these limitations will be factored into the development of algorithms. Defining some of these parameters for imagery and flight plans will be one goal of the project and should help develop future real work systems.

Computer vision development: Image mosaicking and road detection are the two major focuses of computer vision in order to map terrain and determine paths. Low computation time for both of these algorithms and efficient data processing will be the goals for an effective computer vision system.

Path planning and vehicle control: Aerial vehicle flight plans will be developed for efficient information gathering based on prior mapping information. After collecting and processing information, path planning will be used on the ground vehicle to quickly develop a path to the goal.

Integration of multiple systems and facilitating data transfer: Algorithm development involve the integration of all of the algorithms in order to ensure the final program efficiently gathers and processes data in order to map and give path planning information.

Validation and testing of final algorithms: Final work will involve the testing of these algorithms using test flight information and imagery. This should help validate the functioning algorithms and help to define parameters for future flights and research.

2 LITERATURE REVIEW AND BACKGROUND

For the design of this autonomous collaborative system, there is research needed in many different aspects. Computer vision algorithms for road detection and image mosaicking, path planning, and UAV hardware research are the major research topics for this project. By covering all these topics, a comprehensive understanding of the needs for this system can be developed. This next section details the main findings of recent research in all these applications. For road detection and 2D mapping, focus was put on the segmentation methods for images as well as image mosaicking and reducing the computation times of these algorithms. Path planning research is focused on an effective method based on the input map data and also considers the plan for UAV control to map areas. Based on all this information, decisions will be made to begin developing this system and will be detailed in future sections.

2.1 Computer Vision Algorithms

In order to gather and organize information from an aerial vehicle, computer vision is a useful tool. Algorithms can be used to complete many different tasks and process information effectively. In order to effectively map areas, image mosaicking can to be employed to take successive images and combine them into a larger map using a low cost system. Road detection methods are also discussed in order to identify passable paths for the ground vehicles. Image segmentation is one of these methods and has its own section because of the various options for implementation. Reasons for focusing on segmentation are also discussed. The first section below details the background for image mosaicking procedures.

2.1.1 Image Mosaicking

Feature-based alignment is used for 2D mosaicking. It takes automatically detected features in an image and matches them with the same features found in a second image. These key-point features are usually defined by corners and the surrounding pixels at that location [1]. The importance of these features is they are usually invariant of the position of the camera. Information can be gathered relating to these corners to help with matching and correlating the features in multiple images. Other features like edges (straight line segments) can be used for determining things such as vanishing points to extract camera pose and position, although this is not of interest for this research.

Key-point feature detection is useful because they allow for correlation regardless of orientation changes, size differences, or occlusions in the image. Two methods of feature matching are popular. First is finding features that are can be tracked using local search techniques; the other is independently detecting features and matching them based on local appearance. The second method is better suited for larger amounts of motion between images [1], and seems more advantageous for this project's applications. Although images could be taken in more rapid succession, in order to manage computation speed and information timing, the images may need to be more spread out.

David Lowes's work using Scale Invariant Feature Transform (SIFT) in 2004 is an effective method that detects features, calculates descriptors, and matches the features. The features are invariant to scale, location, and orientation. This is accomplished using an array of image gradients calculated (with Gaussian blur chosen based on scale for invariance) around the feature point to make it rotationally invariant. A 4x4 array is created for these gradients with 8 orientation bins resulting in a 128 element feature vector, which was proven to be the most effective sizing for key-points based on accuracy and low limitations on speed [2]. Figure 1 below shows a basic graphic of these bins.

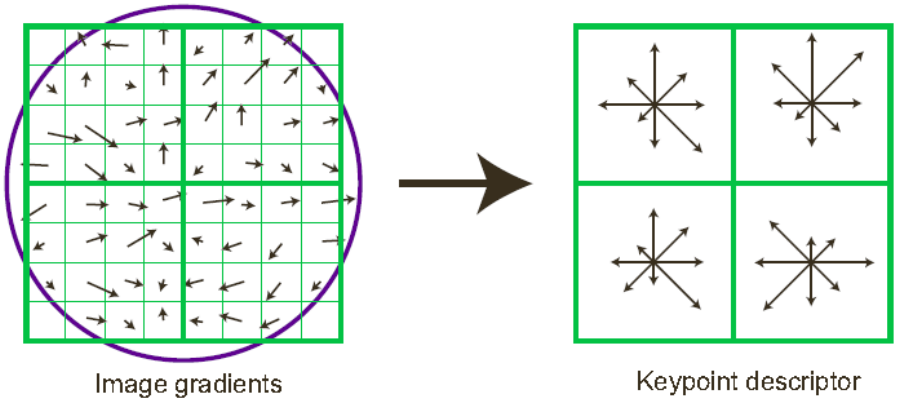


Figure 1: This figure shows the basic layout of a key-point descriptor with 2x2 array with 8 orientation bins (although the proposed SIFT method uses as 4x4 array) [D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *Internation Journal of Computer Vision,* vol. 60, no. 2, pp. 91-110, 2004]. Used under fair use, 2014.

In 2006, Bay, Tuytelaars and Van Gool used a similar descriptor method called Speeded-Up Robust Features (SURF) for this feature detection matching. It uses a similar descriptor with the small difference of using Haar-wavelet response in horizontal and vertical orientations

instead of gradients values at varying orientations. This limits the bin size to 4 and the feature vector to 64 elements to increase speed of detection and processing [3].

Many other different methods of feature detection have been developed and are optimized for different properties. Based on the comprehensive survey of these feature detectors done by Tuytelaars and Mikolajczyk [4], the SIFT algorithm would be ideal for this image mosaicking application. Although it shows slightly lowered speed, SIFT has higher accuracy and can still be implemented at speeds suitable for real time applications. Also, MATLAB has open source computer vision algorithms that utilize these features and can be easily implemented with the next step for image processing.

After feature detection and matching have been completed, a simple homography matrix can be calculated relating the feature points in each image to homogeneous coordinates. A homography matrix represents projective transformations of images based on the geometric properties of the camera. It assumes a pin-hole camera model. The goal is to warp one images coordinates onto another based on their camera's relative positions. The basic equation for image transformation is shown below, which applies a transformation in order to match image points with a second image's features [5].

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \quad (1)$$

The x and y variables are the coordinates of the pixels while z is used for normalizing to homogeneous (projective) coordinates. The transformation of the image is performed with simple multiplication, although image padding and shifting is required to overlay the images correctly. That is the basic procedure for image mosaicking in order to give a basic background understanding, but other work has been done to develop these calculations [6]. The overall goal was to collect image sequences and assemble them into a larger 2D map of the area.

Continuous image mosaicking in real time has not been greatly researched, although Laplacian pyramids and image registration have been shown to be used for image alignment. Work using real-time scene stabilization has used a motion estimation technique for affine (2D

projection) transformation in order to align these images, although highly accurate SIFT feature detection was not used [7]. Image mosaicking work that has been done using these SIFT features is further detailed later in the paper (see 3.1 Image Mosaicking) which focuses on continuous mosaicking strategies using SIFT descriptors and methods for reducing computation time.

2.1.2 Road Detection Methods

Road detection is the next important method to be researched. After successfully imaging the environment, it is important to extract information from the larger map. The goal is to develop road detection methods in order to aid in guiding the ground vehicle to its target and identify changes to known road locations. There are major methods that have been implemented before including line segment detection and segmentation. Segmentation has multiple methods that are described in the next section, but for now, the applications of segmentation are discussed.

Segmentation is a means of grouping pixels based on given parameters [1]. For image segmentation this can be anything from RGB pixel values, pixel location, image gradient values, and others. However, the end result is having each pixel in an image labeled as a specific group membership. By identifying a single road location in the image (or previous images), the group that matches road locations can then be used as the main roadway detection method. Use of connected components and image morphing are also used in order to refine the selected pixels to remove noise, or areas not near the roads that have similar properties [8] [9].

Line segment detection is another method that can be used. Many different methods have been developed including using Canny edge detection, Hough transform, or mathematical morphology. From these algorithms, each line is identified usually by their start and end points. Lines can be compared based on slope, location, length and other properties. In terms of road detection, the line segments need to then be filtered based on their slopes and locations due to major overlapping segments that can occur. Methods for connecting line segments based on distance thresholds or intersections should also be performed to develop a full map and road skeleton. This method is much more applicable at higher altitudes because roads are more likely to appear as line segments, although the edges of roads can be defined at lower altitudes [10].

For this application, segmentation seems a more appropriate method for road detection. It gives versatility for varying altitudes and the simplicity of processing after extracting road locations. Also, the path planning algorithms could use this higher resolution data more effectively or vehicle parameter analysis could be performed on the map. Therefore, investigation of the types of segmentation was conducted in order to optimize it for road detection and this application.

2.1.3 Segmentation Methods

Some of the major segmentation methods include active contours, level sets, region splitting and merging, mean shift, k-means, and normalized cuts [1]. Based on these methods, mean shift, k-means and normalized cuts were chosen as major methods for investigation due to their clustering output and efficient methods based on accuracy and computation times [1] [11] [12]. Some common segmentation methods (k-means and mean shift) are essentially mode finders in a large dimensional space. By finding the local maximums, centers (groups) for the segmentation can be identified with neighboring values included in that segment. Below are some explanations for these chosen algorithms.

Mean shift attempts to model the set of data points non-parametrically. By convolving a kernel and derivative kernel function on subsets of the data, a resulting density and derivative can be estimated (illustrated in Figure 2). By looking at the distance from the density function to data points, a mean shift vector can be calculated to shift the kernels and reapply the algorithm [1]. This is a “brute force” method that can be computationally intensive especially as the dimensionality of the data increases. The advantages to this method are there is no need to know the number of centers as a distribution of points can be applied with similar convergent points eliminated, and the precision can be tuned based on the threshold for the mean shift.

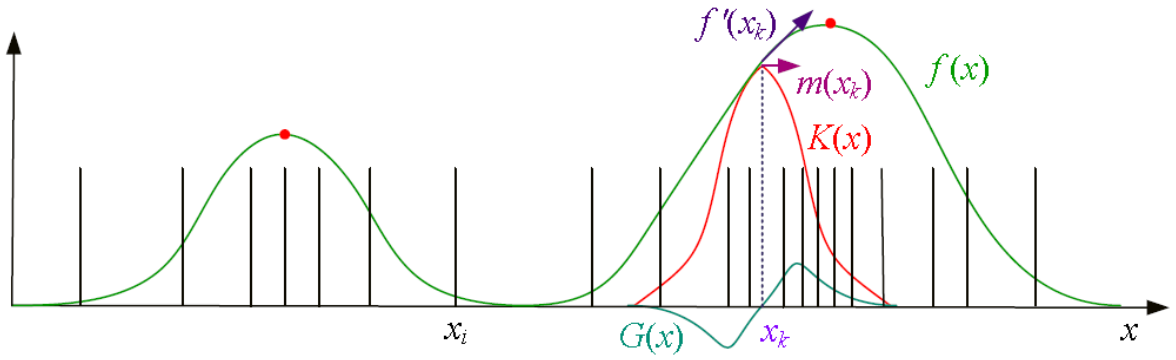


Figure 2: Mean shift method illustrated. The kernel function (K(x)) and derivative kernel (G(x)) are convolved on the subset of data to calculate the density and derivative functions respectively. The mean shift ($m(x_k)$) can then be calculated based on the distances of the kernel to the data points [R. Szeliski, Computer Vision: Algorithms and Applications, Springer, 2010]. Used under fair use, 2014.

K-means segmentation takes a prescribed number (k) of centers and uses a sum of a spherically symmetric distribution on the data points. The mean location of this distribution is used as the new center until convergence is met. The center locations can be randomly distributed or specifically chosen in order to reduce convergence time. Splitting or merging centers has also been implemented for improved performance [1]. Overall, from research this seems to be the most widely used method and has been implemented in many algorithms involved in machine learning and computer vision.

Graph cuts uses a different method by looking at affinities, or similarities, of pixels. It attempts to remove affinities that are weak showing low correlation. The goal is to minimize the cost of "cuts" through the affinities [1]. This method is much more computationally intensive, but can result in very accurate segmentations because there is no limitations for convergence and all pixel correlations can be compared. A minimum cost function can be developed and many cuts tested. By putting limits on the affinities calculated, timing can be improved but accuracy is sacrificed.

Overall, for this application, k-means seems the most appropriate due to its optimized computation time and extensive use currently in computer vision, although there will be concerns with the large amount of points for high quality images. Mean shift is a good second option, especially if the dimension of parameters in the image pixels remains low. Graph cuts seems to be too computationally intensive for this application. Investigation of the computation and

application of k-means for road detection will be addressed. This projects research focused on how to apply k-means segmentation for road detection applications. Selection of segments and tracking points based on image mosaicking were used to develop a larger map across a wide sequence of images.

2.2 UGV and UAV Path Planning

Unmanned aerial and ground vehicles will be used in collaboration for this system. Pathway planning for both systems needs to be considered as well as application limitations. The aerial vehicle needs to effectively map the area and cover all the known passable roads within a range. For controlling the ground vehicle, effective paths avoiding obstacles and reaching goals in a timely manner are important. Also, this is a real time system that should constantly update maps and paths as more information is gathered by the aerial vehicle. Extensive research has been done into path planning for both of these systems, but the focus of this work will be on effectively integrating the two so information can be shared. The algorithms need to be efficient and quick for real-time application in order to now have delay in path updates during the flight mission.

2.2.1 UAV Control for Mapping

The UAV will be used to effectively map and detect roads in a given area. Algorithms have been developed to effectively map an area [13]. They use some basic rules to create large sweeping motions to reduce costs of changing directions (see Figure 3 below). Boustrophedon cellular decomposition is another method well suited for completely mapping an area that utilizes sweeping motions to plan an exhaustive path [14].This is more suited for fixed wing UAV and when extensive mapping is important. For the sole goal of situation awareness and full area information this would be a good algorithm for planning aerial paths. However, it can be rather time consuming to fly. Work has been done with using multiple UAVs with this algorithm to reduce cost, but this research is focused on using a single UAV at this time.

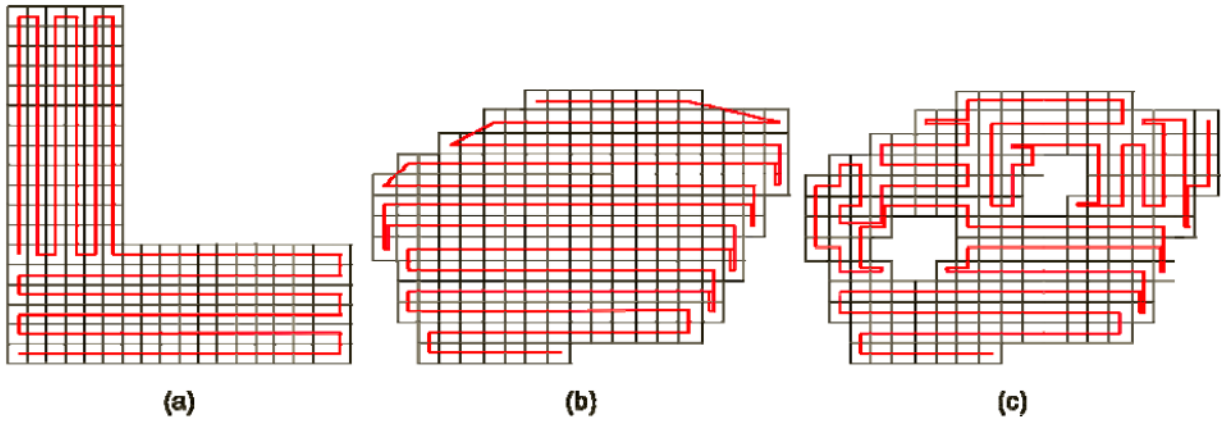


Figure 3: Sample paths generated from an algorithm developed to fully map terrains with large sweeping paths [E. Santamario, F. Segor and I. Tchouchenkov, "Rapid Aerial Mapping with Multiple Heterogeneous Unmanned Vehicles," in *10th International ISCRAM Conference*, Baden-Baden, Germany, 2013]. Used under fair use, 2014.

If areas of interest are already known a more optimized path to inspect those points can be developed (i.e. known road locations). The Traveling Salesman Problem (TSP) has been proposed for solving a path planning method for reaching a given set of "city" locations with the minimum cost. Work has been done to use the algorithm developed to solve this for road network searching. By having the road locations and using evenly dispersed points, the TSP solution can be applied to ensure all locations of the road are visited.

Another solution for route inspection is the Chinese Postman Problem (CPP), where paths are limited to the given road locations. This is more applicable for ground vehicles, but can shed some insight into control schemes for road detection. Comparisons between using the TSP algorithm, the CPP algorithm, and the simple "Zamboni" weaving pattern described above have been done with a timing and cost analysis [15]. Different densities of road networks were tested and the TSP performed the best for most densities, although if high density road networks are present, the Zamboni pattern is much more effective because the whole area needs to be mapped anyways. Below, Figure 4 shows a comparison of these methods on a suburban-style road network. It should be noted this also took into account costs of turning with a fixed wing vehicle, but should still help provide means of improving other UAV applications.

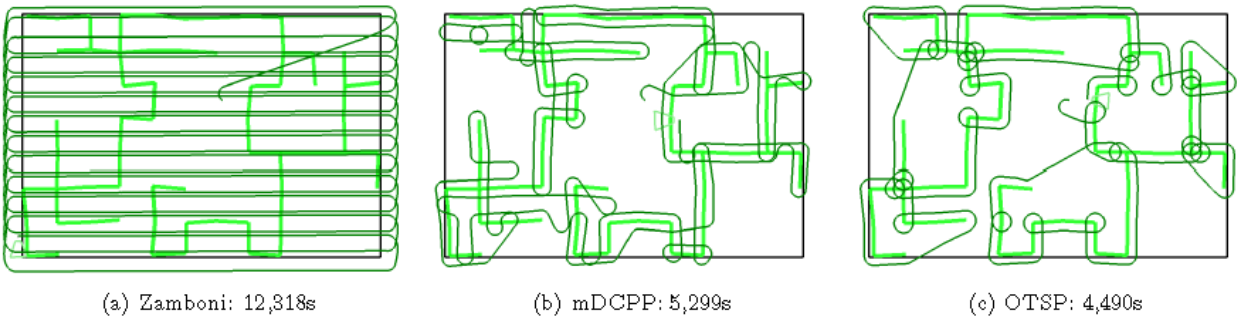


Figure 4: Test simulations of algorithms for route inspection using multiple methods to minimize flight time costs. (a) Zamboni Pattern, (b) Modified CPP algorithm, and (c) TSP algorithm [M. Dille and S. Singh, "Efficient Aerial Coverage Search in Road Networks," in *AIAA Conference on Guidance, Navigation, and Control*, Boston, Massachusetts, 2013]. Used under fair use, 2014.

As stated the TSP solution appears to be the most effective method for efficiently mapping roads and performing road detection for disaster situations. Implementation of this solution will be performed to maximize mapping efficiency. Different methods for solving this TSP exist, but focus was not put on this for this application. Improvements and opportunities to improve this solution are discussed in the improvements section (6.2.1 Aerial Vehicle Planning). The next goal is to take the data obtained from these road inspections to update maps for the purpose of developing path planning for ground vehicles as well.

2.2.2 UGV Path Planning Algorithms

From the proposed road detection methods, a road map that selects areas deemed passable would be created. Connecting paths based on starting points and goals can be generated with optimal paths found. Several methods can be used including artificial potential fields, probabilistic roadmap (PRM), Rapidly-exploring random trees (RRTs), A-star (A*), and Support Vector Machining (SVM). These algorithms are discussed in this section with focus on optimal methods to use for this application. Many other methods are available, but this survey helps to describe a range of possibilities and considerations made.

Artificial potential fields uses a field of gradients that add high potential to areas where obstacles are found and low potential near targets [16]. The basic model for this system resembles a gravitational model with a ball resembling the vehicle and letting it roll downhill to the target. This allows for creating potentials based on geometric knowledge of the environment, constraints on the vehicle, and priority of targets. However, control issues arise as local minima

areas are reached [17]. Areas of low potential can cause the ground vehicle to get stuck in points where there are no strong forces directing it towards the goal. Adjustments have been made to this algorithm to avoid these problems, but increase the need for application development [18] [19]. Figure 5 shows below some basic potential maps that are created for paths [16] [20].

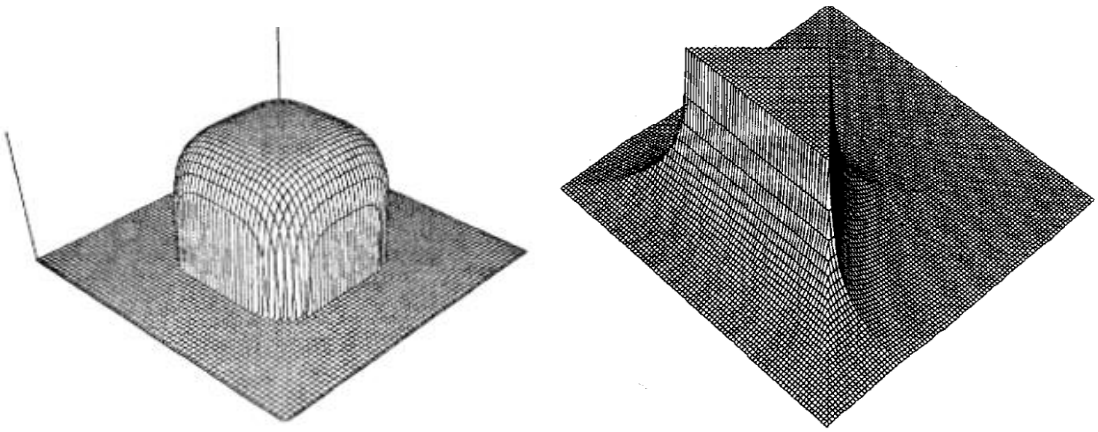


Figure 5: Potential fields based on the artificial potential field algorithm that show vectors and gradients leading away from obstacles and towards goals [H. Safadi, "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots," Stanford Univerity, Standford, VA, 1985. Used under fair use, 2014] [Y. K. Hwang and N. Ahuja, "A Potential Field Approach to Path Planning," *IEEE Transactions on Robotics and Automation,* vol. 8, no. 1, pp. 23-32, 1992]. Used under fair use, 2014.

Probabilistic road maps are categorized by two phases, the learning and query phases. Random paths and vehicle states are repeatedly generated and a simple motion planner creates paths rapidly. A set of edges and nodes are created from these random states to develop a graph space to later query results. The training has proved to be quite fast (on the order of ~30 seconds) and allows for queries to be performed in a fraction of a second afterwards. The query phase uses the current vehicle and a goal location and to create a path. Graph search is then applied to find edges that were created to connect this path [21]. Similar to the artificial potential field method above, this is advantageous when detailed information about the entire environment is known to find the most efficient path.

Support Vector Machining has more recently been used in many applications for path planning. It is known as a strong classifier and can generate smooth separating surfaces. Positive and negative obstacles are created and a smooth surface is found to separate the two classes [22]. Prior known map information can also be used in order to develop boundary points for faster

map processing [23]. Figure 6 below shows a basic example of the SVM algorithm being applied. Although it shows good performance, poor handling of multiple paths and the need for training make it undesirable method for this application.

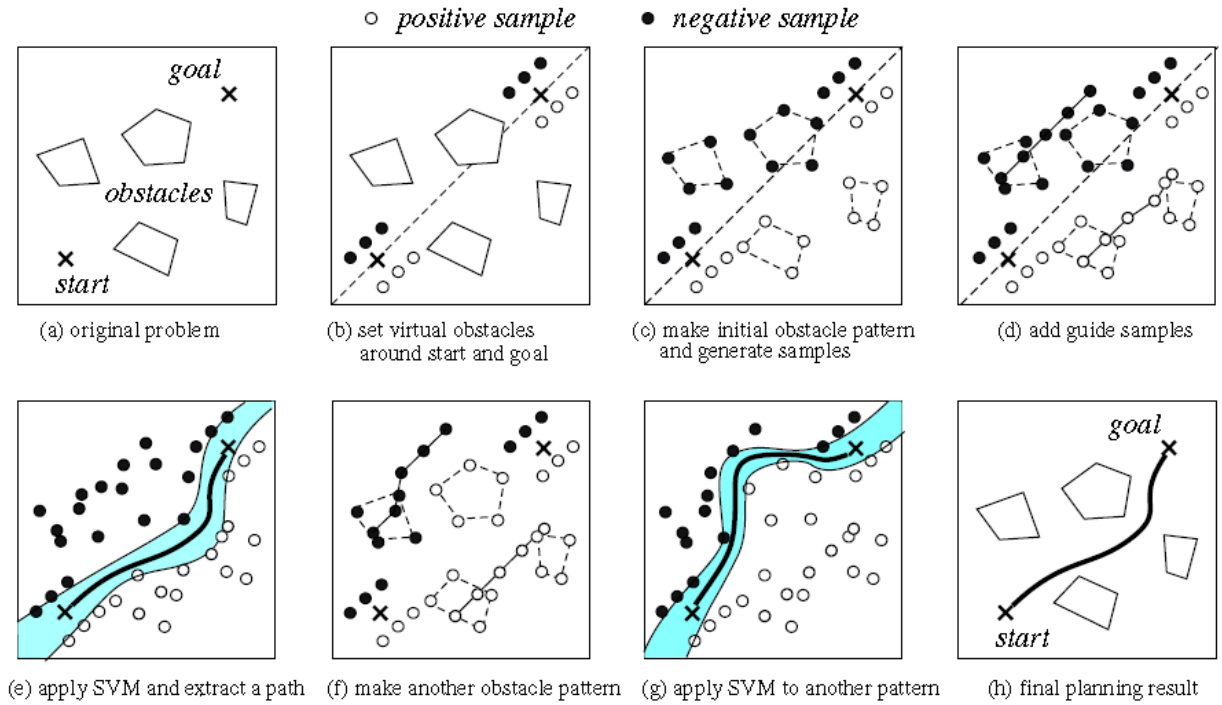


Figure 6: General steps for SVM algorithm and path planning [J. Miura, "Support Vector Path Planning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Beijing, China, 2006]. Used under fair use, 2014.

A* is a widely used method for path planning and graph traversal without pre-processing the maps. Its widely used applications optimizations including Anytime Dynamic A* [24] and Lifelong Planning A* [25] show that many vehicle and environment parameters can be considered. Use of heuristics (experience-based) techniques also have been applied to these systems in order to improve speed and accuracy of path planning methods [26]. By using previous map information or calculating path costs, heuristics are used to help refine the calculation of a future path. This requires preprocessing of the map information, but can yield an optimized result as paths are constantly being adjusted. Focus on ground path planning optimization were not major goals for this work, but this algorithm's application is further discussed in the future work section because of its potential (6.2.2 Ground Vehicle Path Planning).

The final path planning algorithm researched was the Rapidly-exploring Random Tree (RRT) algorithm. Similar to probabilistic road maps, it is a relatively simple algorithm, but does not require a large number of connected edges or training time on a generated space. RRT is focused on exploring unknown or unvisited portions of the map, which can be advantageous for this application to ensure the ground robot is guided towards areas the UAV will be exploring (assuming the UAV is progressively heading towards the goal). Consistency of resolving is also a strong advantage because the distribution of points approaches the sampling distribution [27]. The figure below (Figure 7) shows how the rapidly-exploring tree can be used to quickly find efficient paths and connect the necessary edges [28]. Randomly selected nodes create a branching tree structure to search for the goal. Because it is focused on exploring the unknown terrain, it quickly makes decisions and leave margins around edges for a quick path calculation without over focusing on optimization. Vehicle parameters are not included in this algorithm at this time, which means the map must only include traversable points. However, this allows for a simple operation of the algorithm for quick path planning.

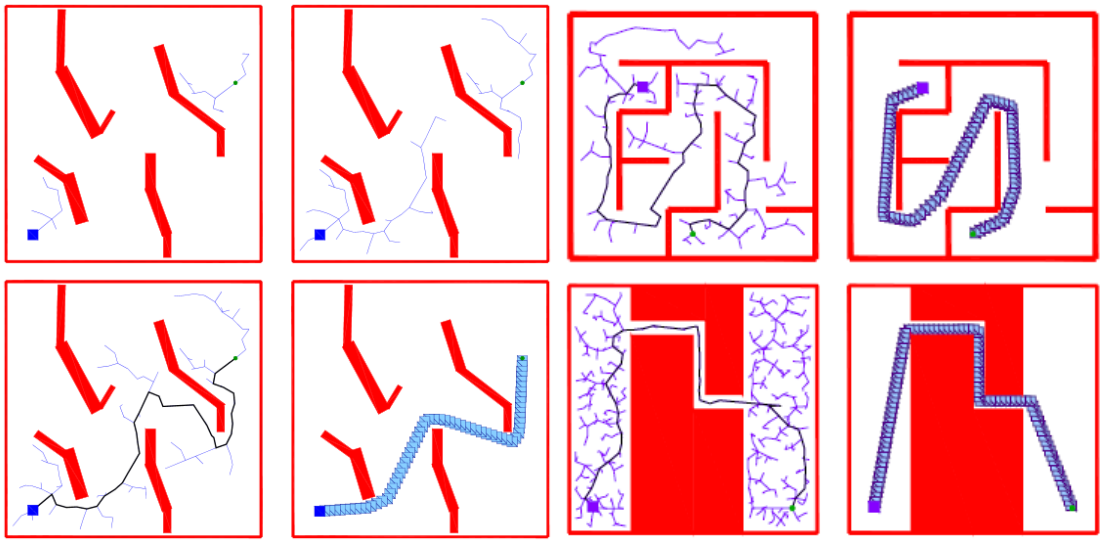


Figure 7: RRT Algorithm search for a single query path to a given goal in a 2D generated map [J. J. Kuffner and S. M. LaValle, "RRT-Connect: An Efficient Approach to Single-Query Path Planning," in *IEEE International Conference on Robotics and Automation*, San Francisco, CA, 2000]. Used under fair use, 2014.

From this section, an overview of the path planning algorithms has been developed. Understanding the limitations of some of these including training time and overall computational problems for real time systems shows that the RRT algorithm should be promising in this application. Work was done with implementing the RRT path planning algorithm with the discussed road detection method. Improvements to this method were performed and show this simple method can still be sufficient for waypoint navigation. As stated, focus was not placed on optimization of the ground path planning at this time, although the A* path planning and use of heuristics could improve the path planning efficiency for future applications. Real time handling of image mosaicking, road detection and the UAV and UGV path planning algorithms should prove to create an effective combined system for mapping an guiding an UGV towards a goal target in unknown terrain. For this real time application, the goal was to avoid any undue delays in order to do all information processing during the flight mission for ground vehicle path planning.

3 COMPUTER VISION ALGORITHMS

Below details all the work done with computer vision algorithms and the conclusions reached for final applications. Image mosaicking and road detection are the two major topics. The image mosaicking uses a sequence of images taken from a UAV (or simulated images) and creates a larger mapped terrain by warping and overlaying them. Analysis of using SIFT descriptors for the homography calculations was performed, and improvements made to the system for reduced computation time. The road detection section discusses the use of k-means and other methods involved in determining road locations. A continuous extension that takes advantage of previously found road locations was also developed. Connection of road locations and overall mapping was performed for use in path planning in the next section.

3.1 Image Mosaicking

3.1.1 Descriptors and Homography Calculation

As stated in previous research, use of SIFT descriptors should prove effective for finding feature points while still remaining efficient for timing [4]. The descriptors are found based on areas of interest in the image, mainly corners and edges. These points then need to be correlated between separate images where the camera position and/or pose has changed. By using the 128 parameter array created in the SIFT descriptor, the error between the descriptors can be minimized to find the matches. There are open source codes used for SIFT feature matching that have been ported to many different platforms to aid in implementation. If lighting conditions are not strong or consistent, image brightness might need to be normalized in order to accurately find matching features. For this application, all code was done in MATLAB a modified version of optimized SIFT mosaic code [29]. Below, Figure 8 shows SIFT features identified in two different images and the correlation between them.

Figure 8: Top SIFT Features identified and correlated between two overlapping images [29] [National Oceanic and Atmospheric Administration, "Hurricane SANDY Response Imagery," National Geodetic Survey, 2012]. Used under fair use, 2014.

It should be noted that this only shows the top 15 matches based on vector distance. A total of 860 descriptors were found and correlated, but many are inaccurate or poorly correlated. Therefore, only high scoring matches should be used in the next step of calculating the homography (fundamental) matrix. Homography matrix transforms are an extension of epipolar geometry that maps point transfer from one plane to the next [30]. Camera positions are correlated using epipoles (lines connecting the center of the cameras) to map the planes of images created and the correlation between these points. Epipoles are created from the intersection of all epipolar lines, which connect points from one image to the other. Below Figure 9 shows a basic depiction of this correlation. The epipoles can be represented algebraically using the fundamental matrix, which gives the parameters of the epipolar lines projected to points in the second image.

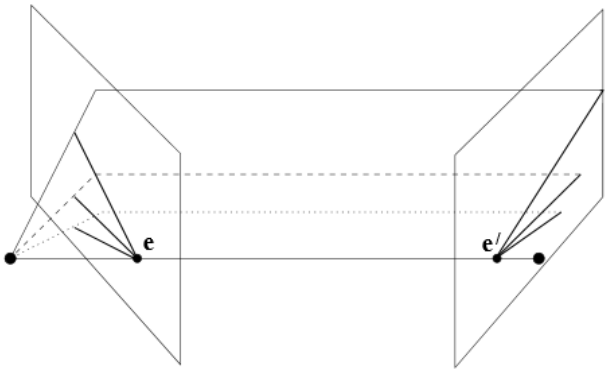


Figure 9: Epipolar geometry correlating the line between 2 images and their camera's position changes. Line e-e^{l} is the epipole connecting camera centers [P. P. Singh and G. R. Garg, "Automatic Road Extraction from High Resolution Satellite Image using Adaptive Global Thresholding and Morphological Operations," *Indian Society of Remote Sensing,* vol. 41, no. 3, pp. 631-640, 2013.]. Used in fair use, 2014.

As stated, the homography, or fundamental matrix, is used to transform the points from one image to the next by correlating the geometries. In order to calculate this, at least 4 points must be used. More points can be used for these calculations, since the final solution focuses on minimizing error. The equation below shows the relation of the homography matrix to correlate the points between two images.

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{1}$$

Using homogeneous (projective) coordinates makes calculations of transformations more accurate. Therefore, the equations should be normalized by one coordinate (z coordinate in these equations), which yields 8 equations (given 4 points). These equations can be re-written in matrix form shown below

$$\boldsymbol{A}\vec{h} = 0 \tag{2}$$

where,

$$A = \begin{bmatrix} x_i & y_i & 1 & 0 & 0 & 0 & -x_i x_i' & -y_i x_i' & -x_i' \\ 0 & 0 & 0 & x_i & y_i & 1 & -x_i y_i' & -y_i y_i' & -y_i' \end{bmatrix} \tag{3}$$

Solving this system of equations can be achieved using the eigenvectors calculated from the smallest eigenvalue. This finds the minimum error for the transformation because it shows the strongest correlation between points (see [31] for the derivation). Single vector decomposition (SVD) is an effective method that can be employed in MATLAB to quickly solve this. Using 4 points is all that is needed and gives an exact solution, however improvements to the calculation by using more features can help give a more accurate transformation. One of these methods is called RANSAC or RANdom Sample Consensus and was specifically develop for use in computer vision [32].

RANSAC attempts to remove outliers from the data by starting with a small data set and includes more data when it improves the results [33]. This is perfect for this application because only 4 points are required for an accurate transformation and many descriptors found are poorly correlated and do not match. Although the algorithm summary can be found here [32], the basic idea is using a randomly sampled set of points using the minimum required points and determining parameters to model this. Then, the number of points in the full data set that are within an error bounds based on this model is found as a fraction of the whole. Once this fraction gets high enough, remodel the system with all the points found within this threshold. In order to minimize computation time, the number of iterations can be limited and still yield good results. With enough iterations, an accurate homogeneous model should be found using the most number of points from the SIFT descriptor detector. This should result in the best homography transform to apply to the images for warping.

3.1.2 Warp Image and Overlay

Once the homogeneous solution is found, the homography matrix can be used to transform all the points. However there are some challenges with processing pixel information when transforming images. Because the transformation is not limited by pixel locations, points can be warped to areas in between pixels. In order to preserve the image, there are 2 major methods, either “splatting” which distributes color among neighboring pixels or inverse warping which interpolates color values from the original image [34].The inverse warping method is better for avoiding holes in the data. The transformation matrix can be inverted and used to take the final image locations and get its corresponding previous location. The color values of all the neighboring pixels in the original image can be interpolated to the single pixel. Below in Figure

10, shows two images with one (inversely) warped to overlay on top of the other to create a mosaic. This was done using SIFT descriptors and illustrates how clean images can be mosaicked together.

Figure 10: This shows a sample image mosaic between two images with features determined by SIFT descriptors creating a clean image. Photos by author, 2014.

After calculating the image transform and getting pixel locations using inverse warping, the second image is warped and added on top of the first image. It is possible to either replace pixel values or make some interpolation of the two images where there is overlap. Image padding and position shifting is also required in order to fit the images together. By using the homography matrix and applying it to the corner dimensions of the image to be warped, the size of the final image can be found. Based on how much the image is warped and comparing it to the other image should give the max dimensions and also determine how much the non-warped image should be shifted. This shifting and positioning process will also be important for dynamic cropping developed in the next section as continuous mosaicking is discussed.

3.1.3 Continuous Extension

For this application, continuous image mosaicking is a major goal in order to continually map the terrain and use road detection. Sequences of images (and determined road locations) need to be stitched together as the UAV moves across terrain. Use of the previously discussed image mosaicking seems to be a promising method in order to develop these maps due to their

accurate arrangement and lack of artifacts between images. Because there will be a large amount of information coming in from the high resolution images being taken, there will need to be ways of only processing vital information for the mosaicking methods. Timing will become important in order to facilitate road detection and path planning as well. This section discusses effective methods of continuous mosaicking as well as all the ways the computation time is reduced for real time applications.

For a basic image mosaic between two images, the steps include:

- Detect features of both images
- Match the features together
- Calculate a homography transform based on these points
- Warp one image onto the other by padding and shifting the image positions
- Average or replace pixel values at overlapping locations

It should follow that for a basic continuous mosaicking system to work, using this final mosaicked image and another image, the previous steps could be repeated indefinitely. However, there are a few problems that occur when making this assumption. By using the full mosaicked image with each iteration, the size of the mosaic can quickly get quite large as more and more terrain is imaged. Also, SIFT descriptors for a large portion of the mosaic might not even match the incoming image and can give errors for homography calculation. Figure 11 below shows an example with this basic algorithm. It functions quite well for the first few images, especially because the distance between images is low, with only slight alignment errors. RANSAC was used with only a small number of iterations to conserve computation time. However, after the 5^{th} iteration, homography calculations become irregular and accurate warping fails (images were 5MP).

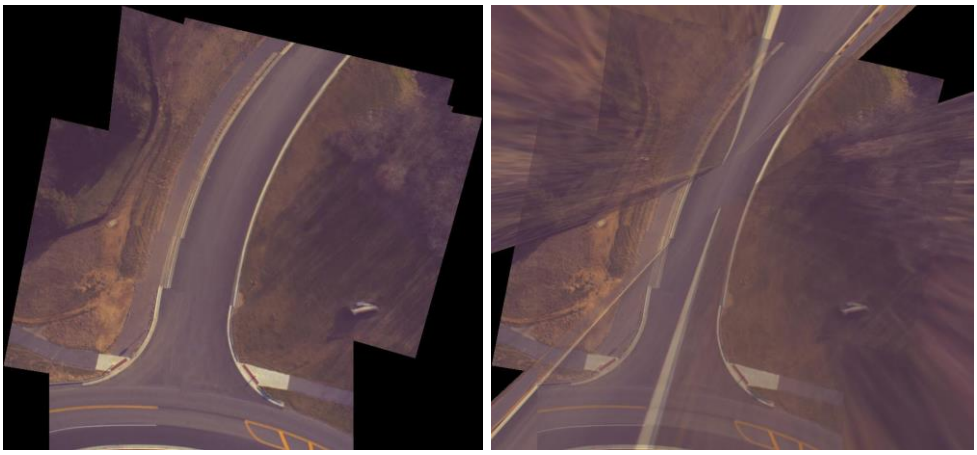


Figure 11: Basic results from continually mosaicking (left) and the error from poor homography calculation (right). Photos by author, 2014.

Another test was done with lower resolution images to see if it was simply the large amount of data points. However, this also yields errors at a higher iteration count (around 15-20 images for 2.5MP resolution). As said, these errors can be occurring for multiple reasons. The size of the image continues to grow in order to preserve all pixel information, so warping onto the large image becomes more drastic and slight deviations in the homography calculation give larger skews. The number of descriptors is getting larger with each iteration as well so matching can become more difficult with more poor correlations being found.

In order to try to reduce the number of descriptors being calculated and give accurate correlation, another method was hypothesized and tested. By saving H (homography) matrices that are calculated, it should be possible to compound transformations onto each other. Therefore, homography matrices can be calculated between image 1 and 2, then image 2 and 3. Then the morphed image of 2 and 3 can be transformed onto 1 using the H-matrix from 1 and 2. This eliminates the need to calculate descriptors on a full mosaicked images that are growing inside. An algorithm was developed and implemented to test this method. However, the final mosaics still reached very large sizes and computation time is still rather slow for final mosaic realization (compounding all homography matrices together). Figure 12 shows an example of this methodology run on 6 images (5MP) and shows better correlation than the previous method, but still has shifting errors due to the compounded homography matrices.

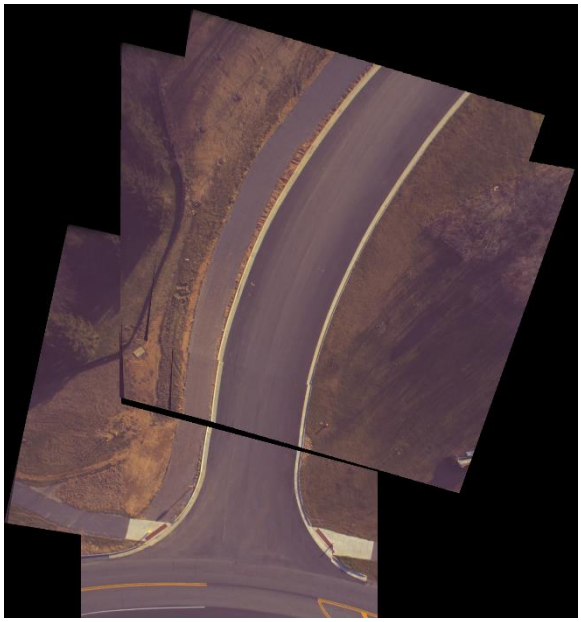


Figure 12: Mosaicked images using saved homography matrices compounded on top of each other. Photos by author, 2014.

Unfortunately, processing these 6 images took too long for real-time applications. Although the descriptors can be correlated better and resolution was preserved, a stronger focus on real-time application is important in order to have this system be functional. It would be ideal to preserve all the details of the image and create a large precise map. However, unless it is assumed there are strong graphics processing capabilities, the data processing will have to be severely reduced. Improvements were be made to reduce the time of image mosaicking with focus on road detection since it is the essential data for path planning. Therefore, more focus was put on on the effect mosaicking had on road detection than on the full map resolution preservation.

3.1.3 Improvements and Computation Time Reduction

Although much work has been done using higher processing capabilities, for this application, basic computation systems (personal laptops) are assumed for ease of access. This allows for an affordable portable ground station system. Therefore, this work should show how a low-cost, effective system can easily be applied in many computer vision and mapping situations. All algorithms are run on a basic duo-core laptop without utilizing the graphics processor for assistance.

In order to reduce computation time and improve overall performance, a dynamic cropping system was developed. Essentially, it was assumed that keeping a mosaic image slightly larger than the captured image should be enough for continuous mosaicking to function, assuming images are captured at a high enough frequency. This frequency will be dependent on camera view angle, altitude of the vehicle and its air speed while mapping the area. Different applications may have different parameters, but essentially there needs to be enough overlap between the stored image and the next one so strong descriptor correlation is found. Also, it was assumed the camera would always be facing straight downward. Although this isn't factored into homography calculations, the extracted road locations and shifting of images are expected to be in a single plane, called an affine transformation. This affine transformation is commonly used in aerial imagery application and should help with processing road information. This requires a nadir view, which means the camera is facing straight down, which can be accounted for by using a camera gimbal or transforming the image based on camera pose information (discussed in Chapter 5).

For this application, simulation of data was used for testing and fixed image sizes were chosen to analyze computation time. Based on a larger resolution image, flight plans can be simulated and video/images created based on the locations of these points. Some natural random drift is assumed in the path as well to more accurately simulate images the vehicle would actually capture. Figure 13 shows a sample view of this flight plan on a larger map where the images are simulated. APPENDIX A: *Flight Simulation from Input Flight plan* shows the code for developing this flight plan and controlling the parameters. The larger map is a compilation of images taken from the aftermath of Hurricane Sandy by the National Oceanic and Atmospheric Administration (NOAA) for the National Geodetic Survey to more accurately evaluate post-disaster environments [35].

Figure 13: Sample flight plan for simulated images based on a larger resolution image and taking sample images at and between the given points (red). [National Oceanic and Atmospheric Administration, "Hurricane SANDY Response Imagery," National Geodetic Survey, 2012]. Used under fair use, 2014.

The basic algorithm described above (simple continuous mosaicking onto a single image) was first tested on images given a simulated flight path. Optimized code that uses SIFT descriptors for image mosaicking was used [29]. Image resolutions were 640x480 and no image data was removed. The chart below (Figure 14) shows the timing of calculating SIFT descriptors and mosaicking two images for this method. As can be seen, the time increases incrementally as more images are added and the resolution of the image increases. The more dramatic increases are during times when vehicle is changing position, since the total image resolution is increasing. Also, 375 iterations were supposed to be completed, but the code timed out after 310 due to homography errors.

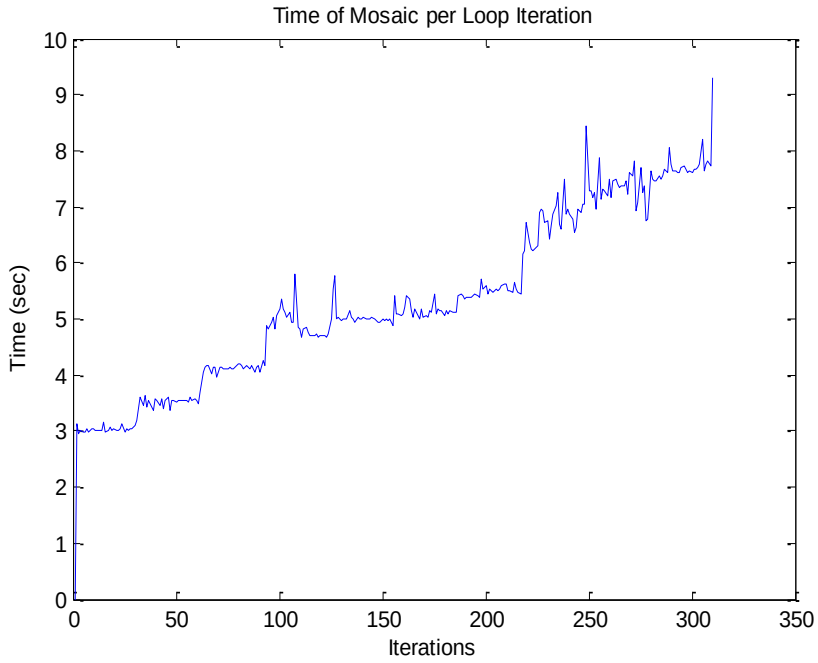


Figure 14: Times of iterations for mosaicking images with simple mosaicking algorithm. Time increases as the mosaicked resolution increases until failure occurs due to too much data processing.

These results confirmed the need for dynamic cropping in order to eliminate data not immediately pertinent to the road mapping. This dynamic cropping method was developed and applied to the current mosaicking method to improve performance. As images are mosaicked onto each other, the resolution of the final image is checked to ensure it has not reached this limit. When the resolution is higher than the set limit, it is cropped to this maximum size. However, in order to ensure the newest image added is not lost, the shifting of the images is tracked in order to crop in the correct direction. This results in a traveling window that follows the images being added to the map. Although the processing uses cropped image, information omitted can still be stored with shifting information for full map generation to maintain higher resolutions. Figure 15 below shows a sample taken from a simulation completed using the old optimized code with this dynamic cropping extension added. The main driver code has been included in APPENDIX B: *Continuous Mosaicking Code* to understand this algorithm. The blurring of certain sections is from averaging over many images and is acceptable because there is less certainty on the mapping of the old images.

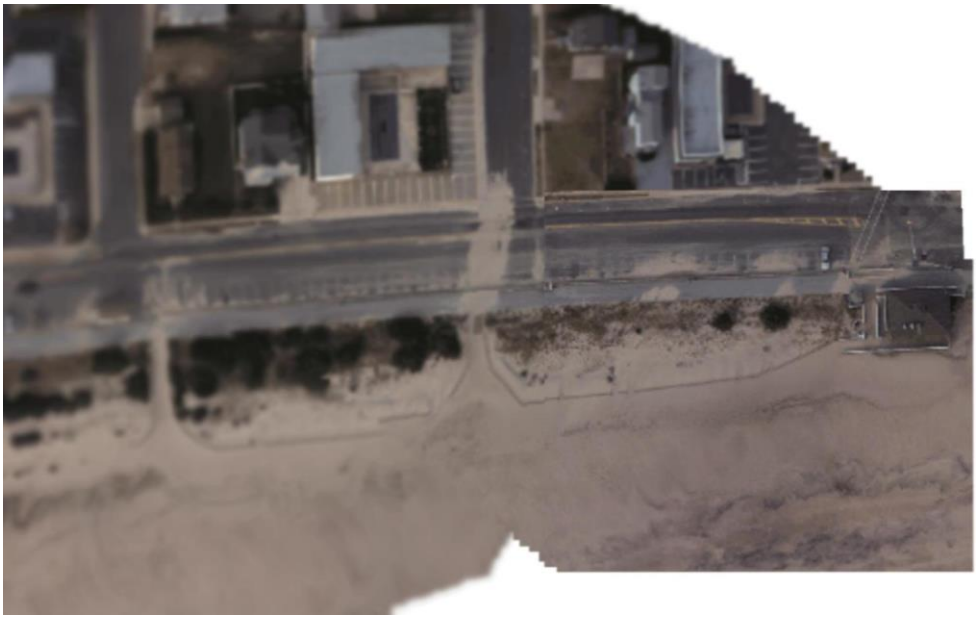


Figure 15: Image mosaicking with dynamic cropping shows a traveling window to remove less recent data for easier mosaicking. Newest image is in the bottom right, and blurring is due to averaging since images to the left are older [29] [National Oceanic and Atmospheric Administration, "Hurricane SANDY Response Imagery," National Geodetic Survey, 2012]. Used under fair use, 2014.

After this was developed a time study similar to the previous was conducted showing that the time was greatly reduced over all by limiting the resolution (dynamically cropping). This kept the iteration times pretty consistent to the initial loop times. The graph shown below shows this trend and that the code was able to complete all 375 iterations and successfully mosaic all of the images given.

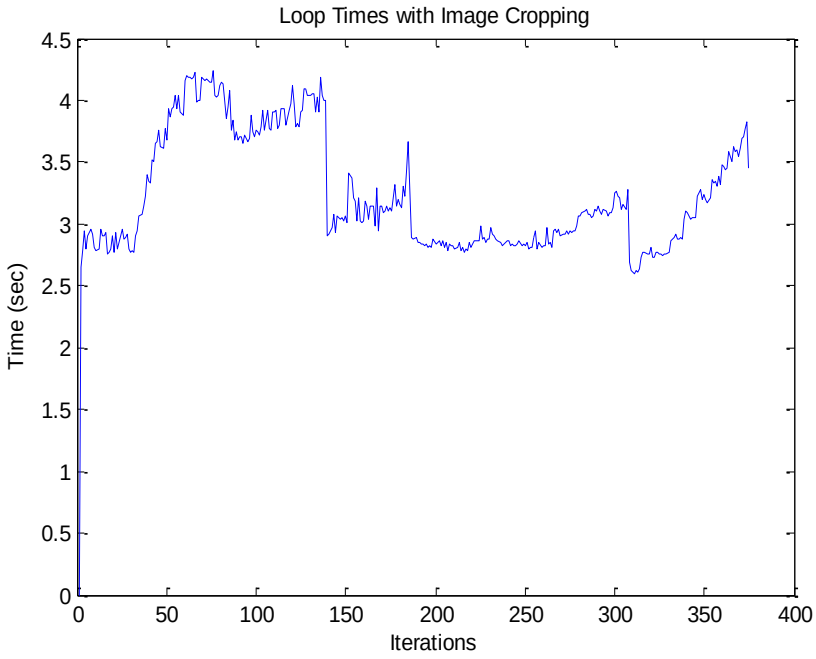


Figure 16: Timing of iterations for continuous mosaicking that utilize the dynamic cropping to reduce computation times are shown. Mosaicking holds consistent throughout the whole simulation.

Although the average iteration time of 3-4 seconds is not extremely fast, it proves feasibility for this application. Because there are means for compiling code for improved performance, final implementation could have significantly lower times. By using this dynamic cropping method, continuous image mosaicking is feasible for real-time applications. This image processing frequency should be enough in order to map an area and perform road detection without causing delays in ground vehicle navigation. As stated before, careful consideration of vehicle speed, altitude and camera viewing angle should be conducted to ensure this. Now that accurate terrain mapping can be created using image mosaics, road detection will be run on these maps in order to find clear paths for the vehicle to travel and run path planning algorithms.

3.2 Road Detection

As stated in the literature review, two methods of road detection seemed the most popular: segmentation and line segment detection. For this application, higher resolution images will be taken at a lower altitude, so image segmentation should give better results to develop maps for path planning. K-means is a widely used algorithm in computer vision and machine learning applications and yields fast and accurate responses. It can process images at high frame rates with configured hardware [36]. Also, it shows good performance for tight, distinct clusters

even with image noise [37]. Therefore, k-means was used in order to process mosaicked images for this application. The major method used is similar to the work done by Singh and Garg [9], where high resolution images are used with morphological operations performed in order to smooth boundaries and connected components used to identify road sections. The road detection method in this paper was developed independently from Singh and Garg's work with considerations included for this specific application. More considerations for overall mapping as well as implications for path planning and real-time application are also discussed.

3.2.1 K-means and Image Morphing

Image segmentation by using k-means is a rather basic process where a certain number of clusters are given to segment the data. Cluster centers are first determined either by random assignment, image intensity peaks or user input (RGB values for image pixels). Then, using averaging of values, centers are shifted until a threshold for minimum shift is reached. Each data input (pixel in an image) has its distance to the centers calculated based on RGB pixel values. The membership of each pixel is determined by the center with the lowest distance (or closest RGB value) [38]. This basic algorithm can easily be implemented using MATLAB and their built in k-means function. Although other aspects of the image can be clustered from pixel location or gradients, this just uses RGB values for clustering. The more centers used, the more noise and errors that can occur with segmentation (i.e. empty clusters with no members). Since coloring of roads and the areas around them are usually distinctly different, four centers were used as an initial start, although a different number could be used based on specific applications. Figure 17 shows this k-means clustering on a sample image.

Figure 17: K-means (4 centers) image clustering performed on a single image based on RGB values [National Oceanic and Atmospheric Administration, "Hurricane SANDY Response Imagery," National Geodetic Survey, 2012]. Used under fair use, 2014.

As can be seen from this image, it is clear that areas of road are well segmented by the darkest gray areas, with obstacles such as cars identified as different groups. However, noise and pixilation of areas makes it difficult to clearly define. By using image morphing, these areas can be smoothed out and connected components can be determined in order to more accurately define these locations. A binary image must be created for image morphing by thresholding the cluster image based on the k-means value of a known road location. Determination of this road detection will be discussed in the next section.

Image morphing allows sections to be eroded, dilated, shrunk, thickened, and more. MATLAB has a built in 'bwmorph' function that details many of these operations [39]. By tuning a series of image morphing operations, the road limits can be smoothed out. Examples of these effects on a binary image are shown below (Figure 18). Depending on the environment may, the morphing may need to be tuned for that area. The goal for this algorithm was to tune it for a specific case, but robustness to varying environments should be tested. Chapter 5 (5 REAL SYSTEM TESTING AND ACCURACY) tests this method on another environment to look at variations.

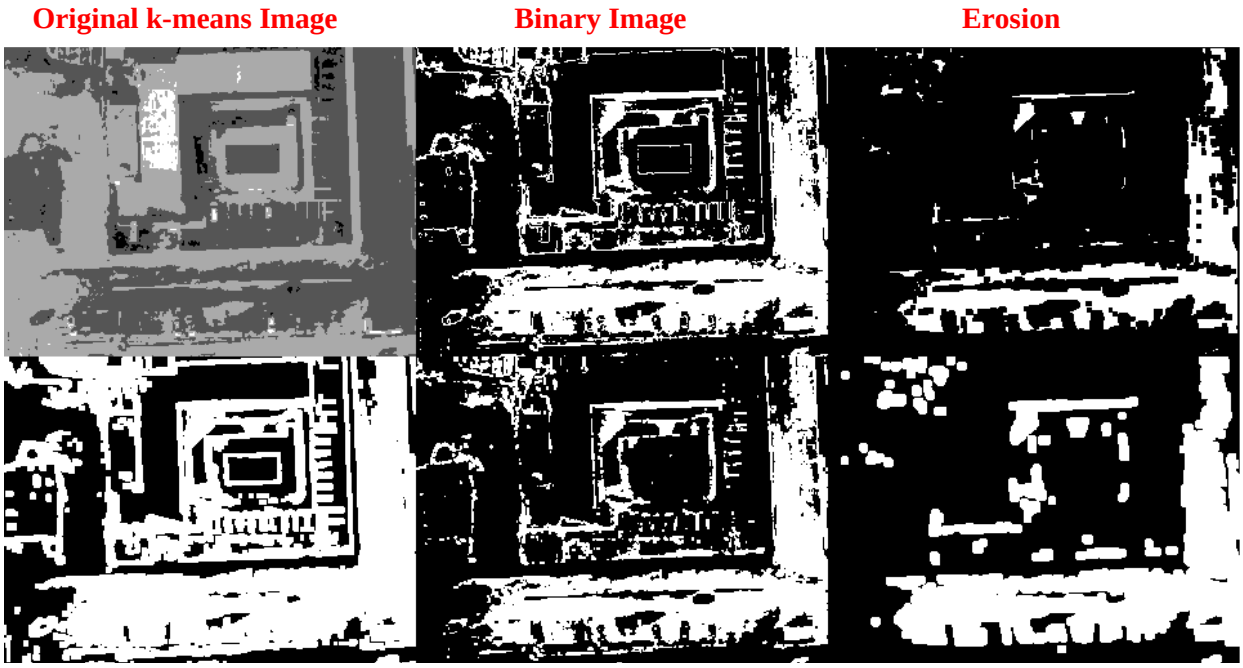


Figure 18: Morphological tests were performed on the k-means image pictures in the top left, in order to see the effects on road locations. The combination (clean, erode x2, spur, open, thicken x2 then dilate x3) in the bottom left shows a good result where the road locations are simplified with less noise [National Oceanic and Atmospheric Administration, "Hurricane SANDY Response Imagery," National Geodetic Survey, 2012]. Used under fair use, 2014.

Based on these tests, the effects of morphological operations can be seen. The combination performed in the bottom left comprised of clean, erode (X2), spur, open, thicken (X2) and dilate (X3) and shows that it is possible to smooth and simplify the complexity of the image and identify good road locations for the path planning algorithm. The code used to test this and calculate k-means memberships can be found in APPENDIX C: *k-Means and Image Morphological Testing*. Unfortunately, there are a lot of sections found in this image that match the parameters of the road, but are not connected and obviously not roads. Therefore, a connected components filter can be applied to the image in order to ensure only areas connected to known road locations are considered. This makes sense with this application because even if

other roads are found, if they are not connected to the current roadway, the vehicle could not reach them.

Connected components is another easy process in MATLAB where it can use parameters of connectivity (either 4 or 8 point arrangement around the pixels in question, depending on the application) and the known points for testing. These points would be the locations determined from k-means and image morphing operations. There are a few different built in connected component commands, but the ‘bwlabel’ command is ideal for this situation because although it uses more memory, the speed is higher and it is better suited for image analysis [40]. Below shows the final image morphing and the results from connect components overlaid on the original image (Figure 19).

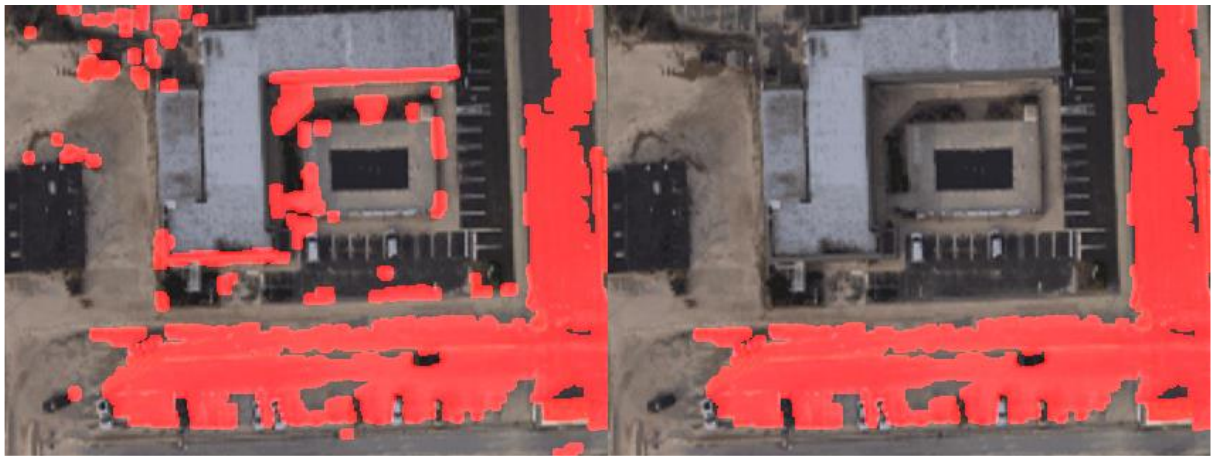


Figure 19: Binary Image morphing (left) and connected component (right) are shown in order to identify road locations [National Oceanic and Atmospheric Administration, "Hurricane SANDY Response Imagery," National Geodetic Survey, 2012]. Used under fair use, 2014.

The images above show that it is possible to determine road locations based on an image and RGB values. Through the use of k-means segmentation, it should be noted that it is still required that at least one road location needs to be known for this algorithm to work. The next section details ways the road could be identified and image parameters updated.

3.2.2 Road Identification

As the previous section explained, after a known road location is determined, the segmentation method developed can easily track road locations. Based on k-means clustering, a group of average RGB values can be determined, called centers. With the first road location used to identify which center is the road, image morphing and connected components can find all the

road locations in an image. However, verification of these points and determination of a starting road location are important for a robust system to function in many different environments.

For this application, it is assumed the ground vehicles starting location will be known, which gives a simple method for determining a starting road location to assign the k-means center. Changing of lighting conditions or road characteristics overtime or in different applications can easily skew the pixel values and therefore pre-determined pixel values are not possible. For these simulations, k-means centers will continuously be calculated and shifted, but at a lowered rate to reduce computation time in order to update parameters for road locations. This is discussed in the next section, but should prove to effectively update road parameters as the aerial vehicle goes through varying terrain and lighting conditions.

Also, given previously known road locations due to GPS coordinates, more locations can be verified despite uncertainties due to disaster situations. These previously known road locations will be used to prioritize aerial imagery path planning, detailed in the next section, so it is reasonable to use them to verify road locations. Although some of the locations may have obstacles, a running average of pixel values for known road locations can help to ensure the image parameters are known. This is not currently being developed, but can be considered in the future. It is proposed that the aerial vehicle also have a range finder of some kind in order to verify altitudes of road locations, which is important for image scale. Essentially, if areas have slightly varied image parameters, the range finder can determine if it is large obstacle or deep hole that is not traversable. Although the development of the final system is not in the scope of this project at this time, it could be developed in the future to ensure a robust system and handle more complex environments.

Continuous road mapping for real-time applications is the ultimate goal for this work and therefore, the handling of these determined road locations from a series of images needs to be discussed. The next section talks about how the road locations found from the segmentation algorithm can be used to create a larger map. Real-time goals and attempts to reduce computation time are also discussed.

3.2.3 Continuous Road Mapping and Computation Considerations

This section details the specific considerations for road detection to be continuous and run efficiently. Continuous image mosaicking was developed in the previous section; however

due to computation, data, and timing constraints, a dynamic cropping system was developed which removes less recent image data from the system. Therefore, it is not possible to use the developed road detection on the entire map at any given time to easily determine all road locations. K-means segmentation timing is entirely dependent on the number of data points (pixels). Therefore, by limiting this data (size of the image) it also helps to allows for the segmentation to remain quick and efficient. Also, the varying of terrain and visible objects will make a segmentation less robust. This gives good reasoning for only running segmentation on sections of images (partial mosaics) or individual images themselves.

However, it is now important to consider options for processing the road location data for a continuous application. Road detection can be used on the cropped mosaicked images, which have a larger resolution than captured images, or it can be ran on individual images. In terms of accuracy, doing road detection of each road image will prove more accurate, but at the expense of overall computation time (because of the number of iterations). Therefore, road detection was tested on the larger mosaicked images first. Examples were run using the final mosaic image from before to show the effect of the blurring and image mosaicking on road detection (Figure 20).

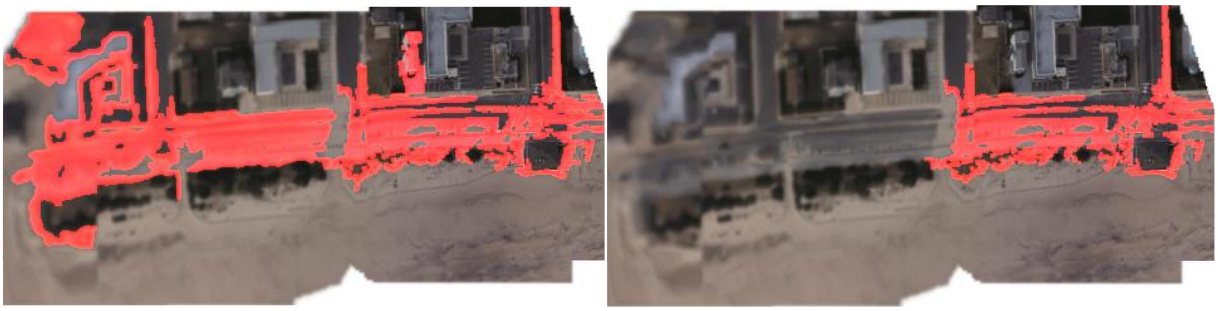


Figure 20: Road detection algorithm used on a full mosaicked image (2 iterations). This does not appear to perform as well as road detection on smaller images and is not as repeatable [National Oceanic and Atmospheric Administration, "Hurricane SANDY Response Imagery," National Geodetic Survey, 2012]. Used under fair use, 2014.

Road detection on the large image mosaic proves to not be as smooth and reliable. Although this uses the same morphological operations of the small image, different combinations did not seem to improve the results much. There is quite a large amount of pixilation, as well as incorrectly selected sections. It should also be noted the time for k-means calculation increased

by a factor of 5 (from ~2 secs to ~10 secs with image sizes from 640x480 to 1280x1024). Although the road detection would not need to be run as often, this lack of consistency makes it not ideal for this application. The inaccuracies were caused by two major factors, the blurring of the image over time and the variability of the k-means centers. The image is blurred over time due to slight variations in the image mosaic and causes clearly defined sections to lose their certainty. Also, because the k-means segmentation starts from randomly placed centers, the membership of each center fluctuate greatly, making the detection not very repeatable. For these reasons, it was decided to use road detection on each image (size 640x480). This increases repeatability because k-means is run on smaller data sets and the resolution of the image is not diminished. As stated, if k-means is run on every image, it could increase over all computation times; however, there was a method developed to reduce the time needed for each road detection conducted on each image.

Because there is only small variations between each image, the use of k-means segmentation should result in similar centers (mean pixel values) from image to image. In k-means, the membership to each center is determined using the "nearest-neighbor" method that finds the center with the shortest vector distance to each pixel. Therefore, the full method of segmentation does not need to be conducted every time, which accounts for a majority of the computation time. By only conducting nearest-neighbor calculations for most images, the computation time can be greatly reduced (on the order of 0.25 secs per image for 640x480 images). K-means can still be run at a lower interval in order to reconfigure the center locations and ensure continued accuracy, but nearest-neighbor should be enough for most iterations. A study running K-means on every image and using nearest-neighbor to reduce time was conducted (Figure 21, Figure 22, and Figure 24). Comparison of the difference between the k-means image and its nearest-neighbor image is shown below as well as the computation time for each looping structure (Figure 23).

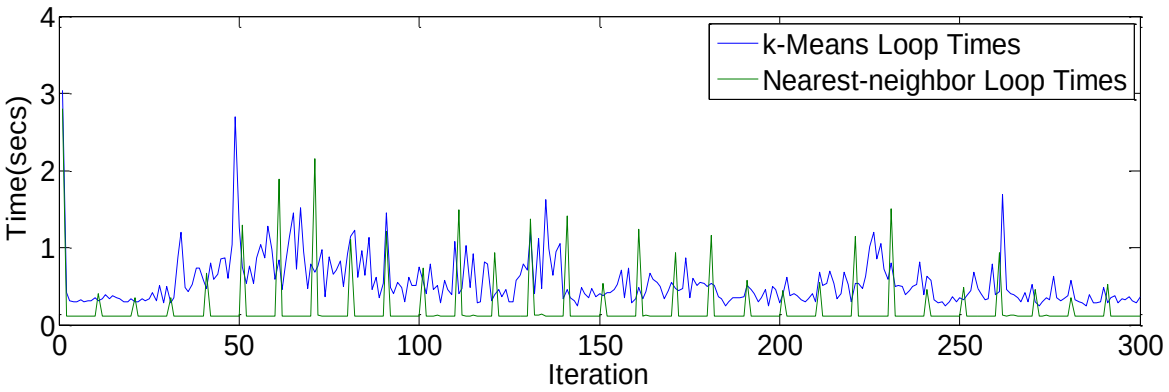


Figure 21: K-means and nearest neighbor (with k-means at an interval for calibration) were run on 300 images showing the loop times.

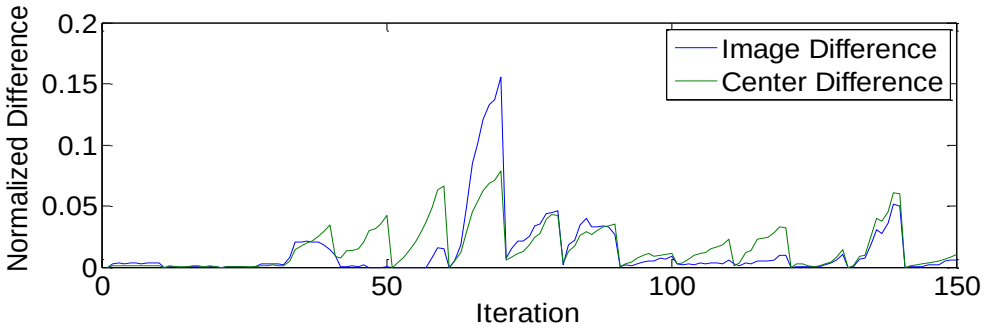


Figure 22: Normalized differences for the k-means centers calculated and the image pixels grouped to those centers with nearest-neighbor. The peaks drop off as k-means recalibrated the centers.

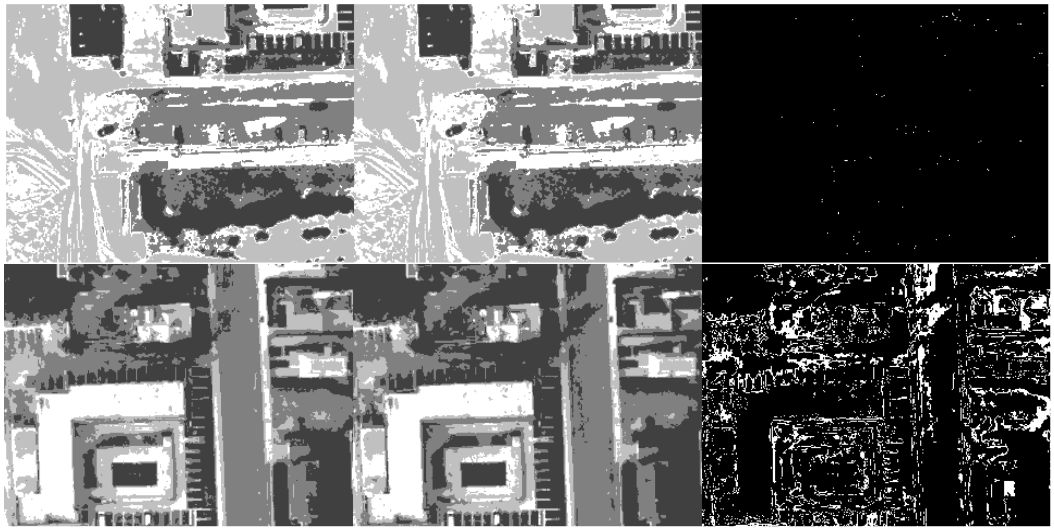


Figure 23: Sample comparisons between the k-means image (left) and nearest-neighbor grouping (middle) with the difference between the images calculated (right). A small difference is shown in the top and maximum difference (16% at the bottom) [National Oceanic and Atmospheric Administration, "Hurricane SANDY Response Imagery," National Geodetic Survey, 2012]. Used under fair use, 2014.

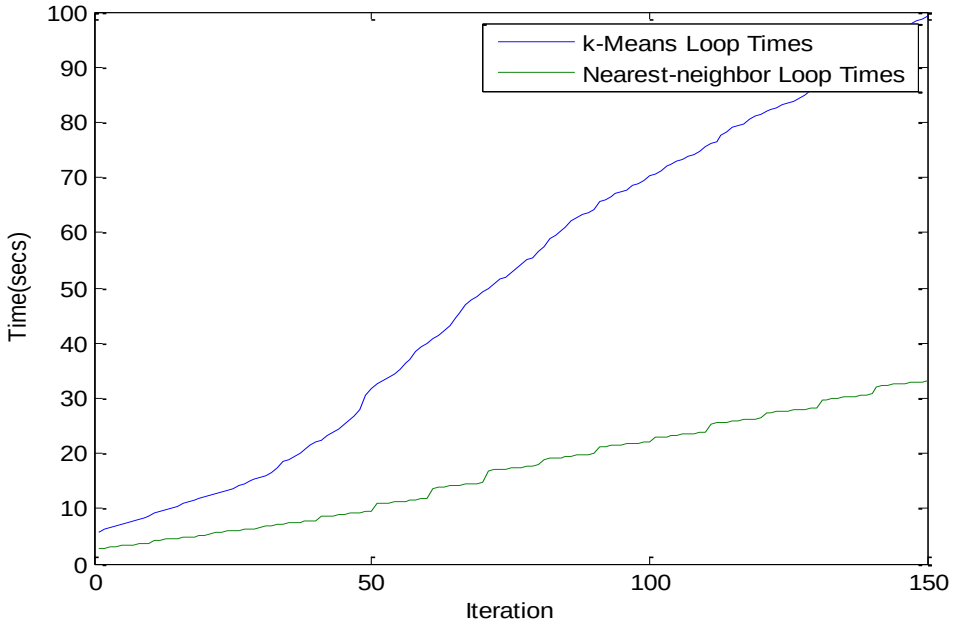


Figure 24: Overall loop times for both segmentation methods as more images are tested. This shows a time save of nearly 70 sec. for 150 images.

From this study, it can be seen that the use of nearest-neighbor with previously calculated k-means centers is an effective method to reduce computation time. After running 150 images, the time difference was 70 sec. with less than 33% of the total computation time. These images were taken with a majority of overlap and more study could be done with the necessary limit for overlap, but this can greatly depend on changing of terrain, desired speed of vehicle and accuracy of segmentation needed. For this study, an error no greater than 16% for an image was found with averages around 1-2% error when k-means was only performed every 10 images. This error is for the whole image, but comparison to road detection accuracy should not change by much.

Now that lists of road points can be extracted from this segmentation method with a greatly reduced computation time, the road locations can be saved overall in a larger map with relative ease. In order to save road locations from individual images into a full map, they should be warped using the transforms calculated in the mosaic section in order to match up with the final map. For simplicity, the road detection is simply ran on the transformed image that is to be added to the mosaic.

As stated before, it is assumed the camera will always be pointing straight down resulting in a single (x-y) plane for all these images. At higher altitudes with a smaller viewing angle, this should not prove to be an inaccurate assumption and helps simplify the road detection mapping. However, adjustments based on known sensors to get camera pose are discussed in Chapter 5 (5.1.1 Image Transforms based on IMU input). Based on the image mosaicking algorithm, shifts in the x-y plane are known according to how the images are shifted. These shifts correlate to how the aerial vehicle travels and collects the new data assuming the downward nadir view. If this information is exported from the mosaicking function, it can easily be applied to the found road locations in a single image (post transformation) in order to store all road locations from a sequence of images. This also allows road detection to not be run on every image if needed because the shifting is constantly tracked. Therefore, based on the speed of the vehicle and amount of road locations mapped in an area, road detection could be run at lower interval than the images captured if it is desired. It is important to ensure there is enough overlay in the images to ensure connectivity of roads, but this can be tuned based on flight characteristics or even monitored in the algorithm. For this section, road detection was ran on every image, but more development of effectively timing this could be done with a specific application.

Integration of road detection into the image mosaicking algorithm was performed in order for them to focus as one loop per image captured. As stated, the shifting of the images as they are warped was tracked along with the transform parameters. The road detection was run on each transformed image separate from the overall mosaic with the shifting information applied afterward where the road locations were stored in a final list. After running this on a large data set of images, a final road map can be quickly created. A few checks were included in the continuous section to ensure the extracted road locations does not vary by much, which can be assumed since the images not separated by much motion.

These large road maps can be used for path planning specific needs. Ideally, to limit data usage, the image mosaic would not need to be stored and simple road images could be extracted. This would only be done if image storing was limited by speed or memory space. Smaller embedded processing systems might consider this alternative if on board processing was possible. Below shows a few sample images from the developed continuous mosaic and road

detection (Figure 25). Also, the developed map is shown as more areas are mapped until the final map is found (shown in Figure 26).

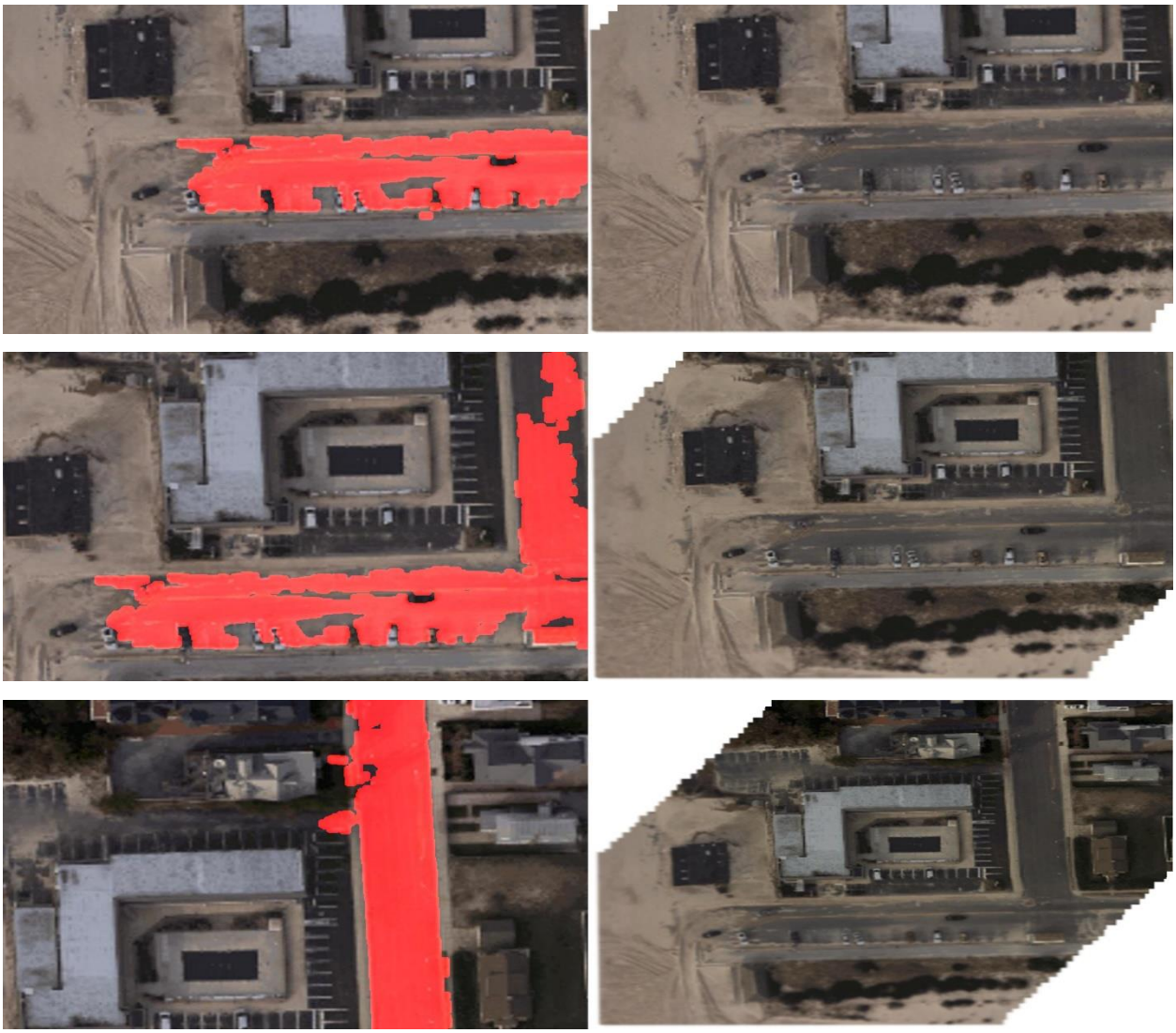


Figure 25: This sequence of images show the road detection (left) and image mosaicking(right) running together. Road locations are stored between images so a larger map can be created [National Oceanic and Atmospheric Administration, "Hurricane SANDY Response Imagery," National Geodetic Survey, 2012]. Used under fair use, 2014.

Figure 26: Final large map created from continuous mosaic and road detection algorithms. Areas selected correspond to determined road locations [National Oceanic and Atmospheric Administration, "Hurricane SANDY Response Imagery," National Geodetic Survey, 2012]. Used under fair use, 2014.

This developed system is not free from errors at this time, k-means can throw errors of empty cluster generation and selection of road locations can be improved. Computation time can be reduced even more, especially on the image mosaicking side. However, this appears to show an effective method and opens up many areas to pursue for future optimization. Now that road maps can be developed using image mosaicking and segmentation methods, path planning can be used to efficiently map the terrain and guide the ground vehicle. The next chapter will discuss both the aerial vehicle algorithm for efficiently mapping as well as the ground vehicle path planning to successfully navigate this map to reach the goal.

4 PATH PLANNING

This chapter details the path planning algorithms developed and implemented for this project. UAV path planning focuses on efficiently mapping the area of road locations, while considering the needs of the ground vehicle at the same time. The ground vehicle path planning algorithm is a modification of the previously developed RRT method for processing data from the developed road detection algorithms. It is assumed these vehicles can take given GPS coordinate commands in order to navigate.

4.1 Aerial Vehicle Control

Path planning for the aerial vehicle is important in order to effectively and efficiently map all of the roads in a given area. Although mapping of the entire terrain may be advantageous for situational awareness and target identification, for this problem only road locations to achieve a path to a predetermined target is of interest. Although previous research shows the "Zamboni" pattern can still be effective in highly dense areas of mapping [15], the use of the Traveling Salesman Problem (TSP) is the focus for this application. The goal is to use prior road locations to develop an efficient mapping plan. However, this path plan can be updated as obstacles are found to reduce the need for further mapping of that specific path. First, the path planning of the previously known map information is discussed.

4.1.1 Prior Map Information and Initial Plan

Given a network of roads, an initial flight plan can be developed. By prioritizing roads, an array of points can be determined in order to plan all roads to be mapped and imaged. Algorithms to determine connected networks and shortest paths can help with this prioritization and limit the area needed to map. Dijkstra's algorithm is an effective method for pre-planning the shortest path given a set of road points [41]. This can be used to find the shortest path for multiple locations as well as multiple shortest paths. Therefore, if multiple targets are needed or planning for multiple paths and they can be considered before mapping occurs. This algorithm was easily implemented using MATLAB and was applied to a large map with road points already determined. This would be similar to having GPS coordinates of roads, which are easily available and could be used for this pre-planning method. Figure 27 below shows this sample map and Dijkstra's shortest path calculated.

Figure 27: A set of known road points on a given map with Dijkstra's algorithm calculating the shortest path. This can be used to prioritize mapping locations for the UAV [National Oceanic and Atmospheric Administration, "Hurricane SANDY Response Imagery," National Geodetic Survey, 2012]. Used under fair use, 2014.

This path was not used in actual ground vehicle path planning, but knowing the quickest path or paths can help to determine the initial path for the UGV as well as where the UAV needs to map to ensure there are no obstructions. Other parameters can also be used to determine more road locations to map based on any predetermined information. For this application, Dijkstra's shortest path was used to define the points within a distance away from the vector between start and goal (basically creating a box in the direction of the goal). This is to ensure that other paths are included for mapping other than the shortest. Other methods could be developed including using multiple paths created by Dijkstra's method or other geometric requirements, but a simple distance was used at this time. Figure 28 shows the points select based on our determined method.

Figure 28: All points in blue were selected for mapping purposes. They are currently determined by Dijkstra's algorithm for shortest path (shown in green) and a given distance from a vector between

start and goal [National Oceanic and Atmospheric Administration, "Hurricane SANDY Response Imagery," National Geodetic Survey, 2012]. Used under fair use, 2014.

Now that a set area to be mapped has been determined, the method for effectively reaching all these points needed to be determined. This next section talks about adopting the solution to the Traveling Salesman Problem (TSP) and using a traveling wave to prioritize points based on distance.

4.1.2 Traveling Wave Applied to TSP

From the given set of locations that need to be mapped based on the initial information of the area, use of a solution to the TSP can easily reach all of these points while minimizing the flight path length. However, in order to quickly deploy a ground vehicle before the entire area is mapped, priority should be given to the closest locations. Therefore, by applying a traveling wave of increasing distance and running the TSP solution on each segment, closer areas can be fully mapped to help make decisions for the ground robot quickly and minimize the total travel time. This assumes the aerial vehicle has a higher speed and can map these sections quick enough to be effective. As stated many times now, the speed of the aerial vehicle and other flight characteristics are important considerations for this application. A visual representation of this traveling wave is shown below (Figure 29) with darker points being prioritized later in the flight plan. However, this is just a model system to use in understanding UAV flight plan limitations.

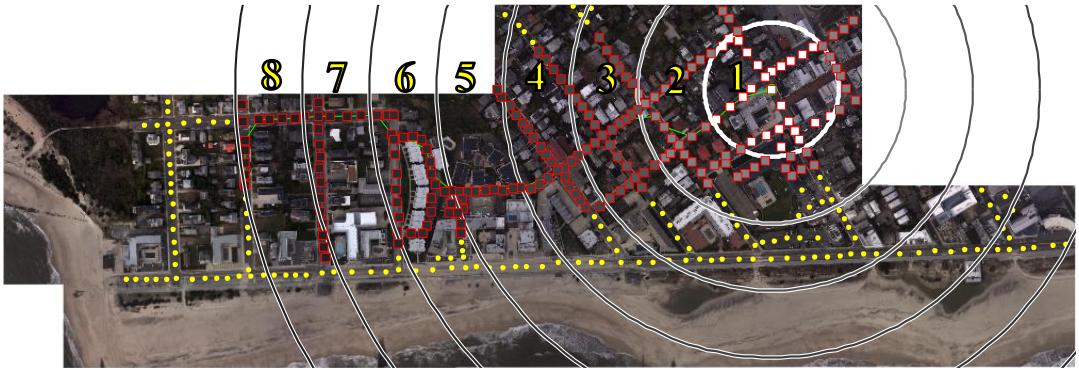


Figure 29: Visual representation of the traveling wave being applied to the points determined for mapping [National Oceanic and Atmospheric Administration, "Hurricane SANDY Response Imagery," National Geodetic Survey, 2012]. Used under fair use, 2014.

The chosen solution method for solving the traveling wave TSP utilized the Genetic Algorithm. Although many methodologies exist for solving the TSP, this solution was chosen for simplicity and its availability in MATLAB. The specific code written by Joseph Kirk uses a fixed-starting point, open solution method which searches for the shortest route from the specified starting point with no limit on the end location [42]. Based on the traveling wave, the TSP solution is calculated iteratively for each section of the wave in order to prioritize the closer positions and create an effectively mapped terrain. Based on the road locations determined before, an example of the TSP is shown below in blue (Figure 30) with the path turning white as it progresses.

Figure 30: A flight path (blue-white line) determined in order to map areas with an aerial vehicle using a TSP genetic algorithm with a traveling wave applied to prioritize locations based on distance [National Oceanic and Atmospheric Administration, "Hurricane SANDY Response Imagery," National Geodetic Survey, 2012]. Used under fair use, 2014.

From the figure, a flight path was developed using the genetic algorithm solution. This used 1000 iterations for each wave section. This flight plan attempted to view how this method developed a flight plan based on this concept. It can be seen that certain areas are rather complex and inefficient, but this is due to the applied limitation on wave distances, tight density of points and need to visit every road location. The sectioning aspect of the traveling wave does nothing to consider using more optimized locations or sections to improve the cost of flight. Some consideration for improving this system needs to be addressed. Because the distance threshold for the traveling wave is not optimized, each section is independently calculated. Wave front algorithms could help to improve this issue and are discussed next. Also, camera parameter information should be factored into the algorithm such as field of view of the camera and

altitude. By including this information, an area that the camera can capture could be used to minimize the need to reach every road location. At this point, this algorithm requires the vehicle to reach all chosen road locations, but the field of view allows the camera to see multiple road locations at a time. A filter could be applied to this flight path that considers the viewable area at each point and removes surrounding points from consideration in the path planning. Therefore, the actual flight plan would be able to sweep over areas more quickly and with less restrictive control. However, there are also considerations needed based on the overlapping of images for image mosaicking. This is discussed in the next chapter for adjustments needed for actual flight plans based on camera field of view (5.1 Accuracy of Aerial Imagery and Mosaic).

After completing this developed traveling wave TSP solution, research was done into wave front path planning algorithms because of the similarity to this application. Wave front algorithms apply a cost function based on distance from the start to all points on a map. The implementation of wave front propagation uses stages of path completion with Dijkstra's algorithm to optimize the path and distances for stages [43]. This is essentially what the traveling wave TSP method is doing, by using the iterations of distance and calculating stages for the full flight plan. However, use of the more developed wave front algorithm could prove to yield a more efficient path than choosing distances for separate stages, because the stages are determined by the cost of paths calculated. The current traveling wave solution has sections at equal fixed distances and each section path is calculated separately. Research has been done in using the wave front algorithm to help fully map areas [44] and could be used in future work. Future application of wave front propagation other path planning algorithms with the given flight plan goals are discussed in future sections (6.2.1 Aerial Vehicle Planning).

Although all of this is used to pre-plan the aerial mapping flight, some consideration of computation time is also important because there may be a need to expand search patterns if blocked roads are found. Fortunately, it took 5.6 seconds to run this algorithm for all 8 sections of the traveling wave, therefore it should be easy to quickly recalculate paths due to needs based on road detection information. The next section details possibilities for updating flight plans in order to continue mapping should the initial plan not be successful.

4.1.3 Map Checking and Flight Path Changes

During the actual flight, as information is captured, road blockages and other factors might exhaust all the terrain that has been mapped with actual connected roads. Therefore, by checking for a fully connected path from goal to start using ground path planning, it can be determined if the goal is reachable with the currently mapped area. This ground path planning is discussed in the next section. However, if there is not a connected path, more mapping needs to be done to help guide the vehicle. Based on the current ground vehicle's position when this is determined can be used as the new starting location for mapping. The same method described above could easily be used again to determine another flight path to map more areas. Figure 31 shows a newly generated flight plan based on an updated ground vehicle location.

Figure 31: This figure shows how the TSP Traveling Wave algorithm can be re-run to determine a new flight plan (blue-white line) when the area mapped is insufficient [National Oceanic and Atmospheric Administration, "Hurricane SANDY Response Imagery," National Geodetic Survey, 2012]. Used under fair use, 2014.

It should be noted that areas determined to be blocked roads need to be marked on the aerial planning map (similar to those above). By carefully keeping track of coordinates for points in the road mapping, they could easily be correlated to the flight plan coordinates to mark blocked paths and all points mapped can be removed from future flight plans. Because waiting is not ideal for rapid response situations, the ground vehicle could be directed to an unmapped area closer to the target with hopes of finding an unblocked path, but allowing the aerial vehicle time to map would be ideal. The areas determined to be blocked should be utilized as well in order to ensure the ground vehicle does not try to approach those at a later time. For this work, not much

focus was put into coding algorithms to handle this based on the scope of the project. However, there is a lot of customization of case handling that could be developed for different environments or applications.

4.2 Ground Vehicle Control

After capturing images based on the pre-determined flight path described above and using road detection, the ground vehicle should have a finalized road map as found in the previous section. This road map can then be used as the path planning map where non-road areas are defined as obstacles. From the literature review, RRT planning seems to be a promising method for path planning. The work in the following section describes testing of the RRT method, adjustments required for incomplete maps to continually adjust path planning, and effects of constantly updating the map.

4.2.1 RRT Planning Algorithm and Improvements

In order to run the Rapid-exploring Random Tree (RRT) planning, different algorithms were tested, including algorithms by University of Illinois' Control Systems Lab [45], toolboxes and systems developed by Peter Corke [46] and the ARMS Lab at Nazarbeyev University [47], but all proved to be ineffective for real-time application. Using Univ. of Illinois' algorithms, preliminary testing of the RRT method was conducted to see calculation times, understand the results and operation and to know the limitations. Using the basic algorithm and sample data given, the figure below shows the output of path along with the obstacles identified. Circles are placed on obstacles in order to give a buffer area and the final path is smoothed out to simplify and shorten the ground vehicle path.

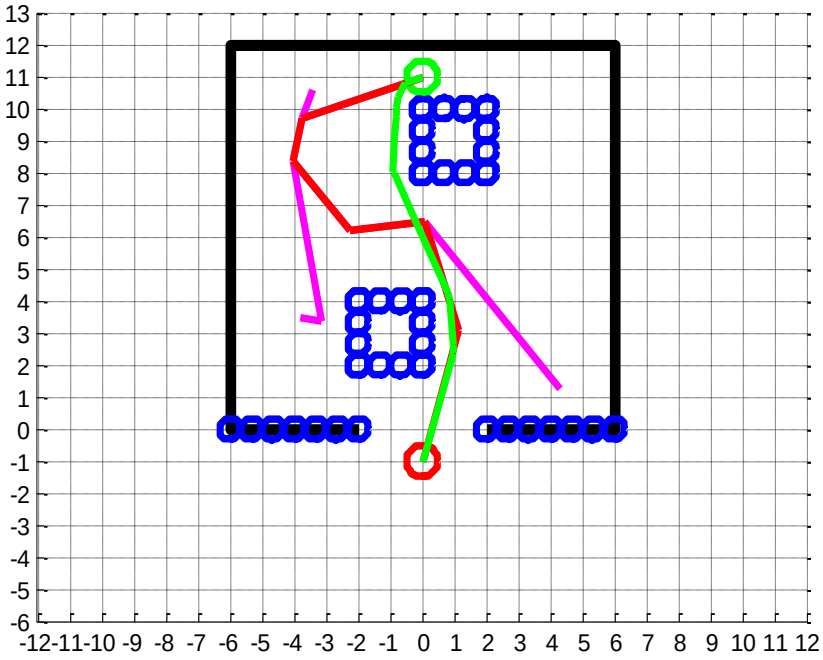


Figure 32: This shows the basic results from an RRT run of the sample data for obstacles (blue circles) with the RRT based tree (pink), the calculated path (red), and the smoothed path (green) [University of Illinois Control Systems Laboratory, "Rapidly Exploring Random Tree (RRT) Path Planning," [Online]. Available: http://coecsl.ece.illinois.edu/ge423/spring13/RickRekoskeAvoid/rrt.html. [Accessed May 2014]] Used under fair use, 2014.

The basic operation of this algorithm is as follows:

- Import obstacle locations and plot circles around obstacles for buffer
- Plan the path (Iterative)
    - Generate random point (new node)
        - Check for node overlap with obstacles
        - Skip iteration if true
    - Connect point to its closest matched node
        - Check for intersection of connected nodes with obstacles
    - Check if point is close enough to goal
        - If true, check for intersection of node connected to goal for obstacles
            - If true, end loop
- Once path is planned, use a function to smooth points and shorten distance.

- Exit Function and plot resulting path

Upon testing this algorithm with other data and reviewing other codes, it was determined that a better optimized method needs to be developed. Because points are randomly generated, there are many failed iterations in a large area with high density of obstacles (common for roadway maps). Also, the searching through all obstacle points for intersections takes a lot of time. Therefore, a new code was written similar to the above described algorithm, but with focus on reducing failed iterations and identifying path/obstacle intersections quickly.

In order to reduce failed iterations, points are only selected randomly from known road locations. This is how the info is output from the road detection code, so this is ideal. Also, this forces a limited resolution of the path nodes. This eliminates the need for a buffering system to avoid obstacles (like the blue circles plotted above) as the road detection code should return all points that are possible to be occupied. Therefore, the only check required is for intersection of the connecting path with known obstacles or non-road locations. Also, another criteria of distance to the previous node was placed on node creation. This ensures the points placed are less like to be around corners or obstacles in the denser road maps. After developing this code, improvements on iteration times were greatly increased. Figure 33 below shows test runs on a fabricated road map with both high and low iteration values. Because the expanding tree is randomly generated, iteration totals can vary and show more widely spread exploration of the map. This either allows for faster computation (low iterations) or more efficient control for the ground vehicle (higher iterations).

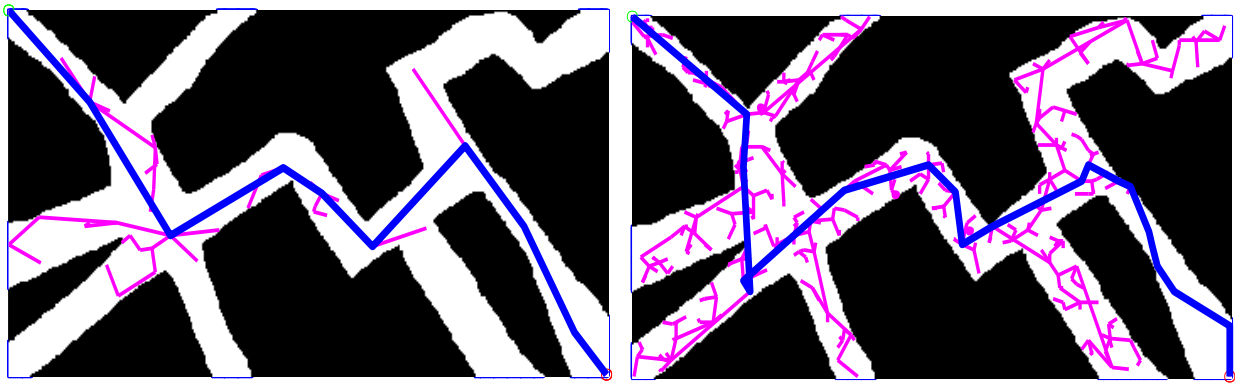


Figure 33: RRT path planning results for low iteration (left: 43 iterations) and high iteration (right: 399 iterations) .

After developing this method, it can be seen that some paths generated are not very smooth or direct in terms of reaching the goal. As pictured above on the right, the randomly generated tree finds a path that makes a wide, unnecessary turn. This poor path is because each path point is only determined based on its shortest distance, not the total connected bath. The necessary search to check all possible paths instead of all possible points add a larger complexity to the search algorithm, so computation time would increase. Therefore, to improve the path performance a basic path smoothing algorithm was developed similar to the path smoothing implemented in University of Illinois' RRT path planner [45]. It iterates through the path attempting to move or add points to reduce the overall path distance. Figure 34 shows this smoothing algorithm run on a generated path. It shortens the overall path, but does sometimes bring it closer to edges of road to achieve this goal.

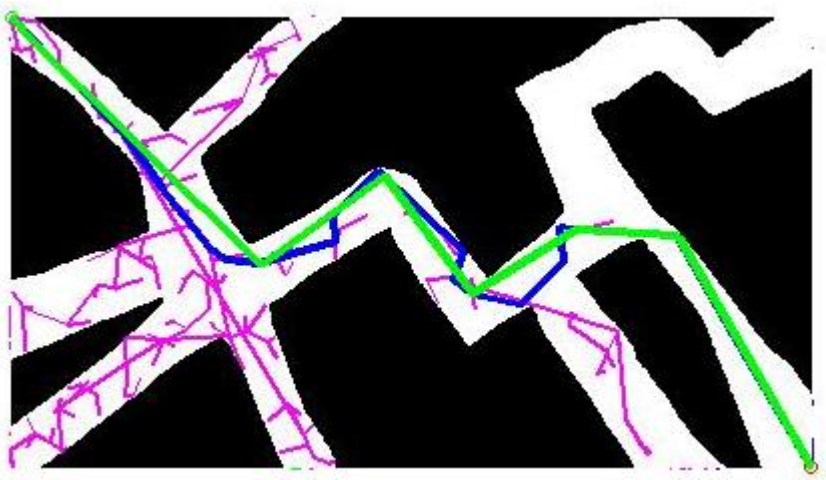


Figure 34: After running the RRT algorithm, a path smoothing algorithm helps to create a shorter more efficient path (green).

There are many extra parameters to begin considering now that this algorithm has been developed. Ground vehicle characteristics including speed, turning radius, and size all affect whether it can navigate through the terrain. As stated from the beginning problem definition, it was assumed that ground vehicle would have some basic obstacle avoidance on board so the generate path should be sufficient. However, if more optimization to this path planning was to be developed, it should consider the vehicle's needs and characteristics. The RRT method does have some inherent built in sizing parameters. If obstacles or tight spaces constrain the path,

generating a random tree that completes the path could take more iterations. Looking at Figure 35 below shows the effect of tight spaces.

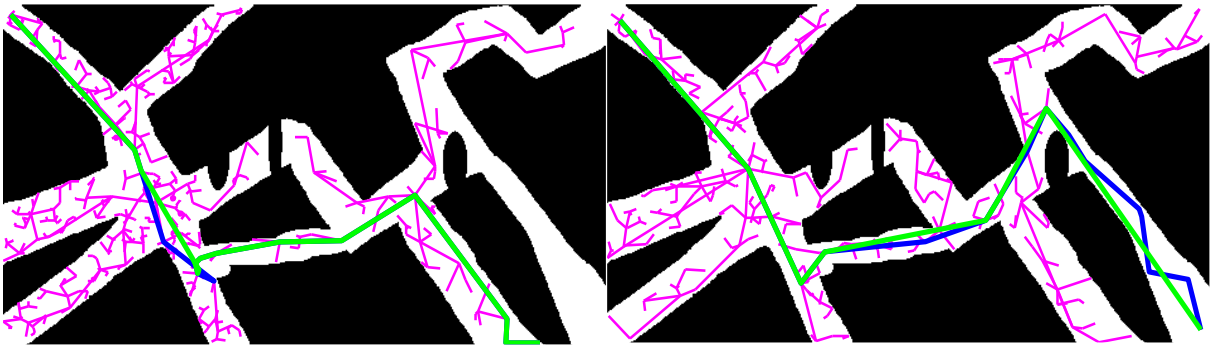


Figure 35: Tight spaces and obstacles can constrain the path and make it difficult to find a final path. Although it sometimes fails to navigate obstacles (left image), it can still reach a goal close to the target. Different iteration counts can still solve the path though (right image).

Although it can find difficulty in navigating these paths, different iteration counts can still yield results. It should also be noted that it can take a different path that gets the ground vehicle close to the goal despite the obstacles. The next focus of this project was in processing the map information at intervals during the data collection to develop intermediate paths when the map is still incomplete. This is detailed in the next section.

4.2.2 Incomplete Maps and Goal Adjustments

In order to facilitate ground vehicle control, the path planning must be performed while road map info is being collected and updated. Therefore, incomplete maps would be input into the path planning algorithm to give the ground vehicle an initial path. In order to complete path planning, an intermediate goal must be determined. For this purpose, using the known road point closest to the final goal would be appropriate to choose. However, if it is a dead end, a path should not be mapped. Therefore, only checking roads that extend outside of the current map should be evaluated. By checking edges of the mapping area for road locations, blocked or dead end paths can be identified and ignored. The RRT algorithm was also modified to adjust these goal locations, and incomplete maps were tested on the algorithm (see Figure 36 below).

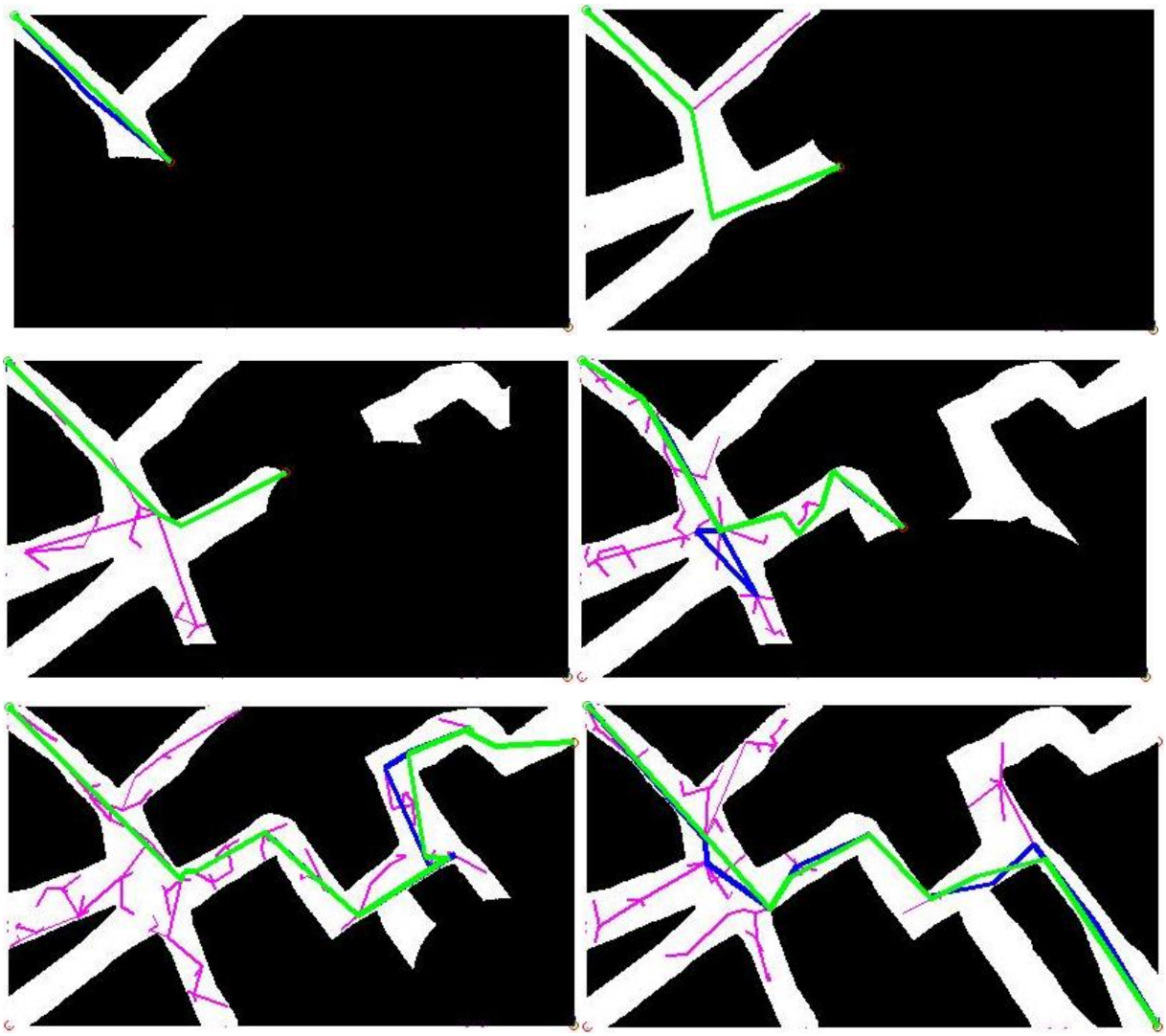


Figure 36: These images show a sequence of incomplete (complete on bottom right) images with the developed RRT path planning algorithm. It shows good performance with some misdirects, that are quickly remedied as the map is completed.

As can be seen, the RRT algorithm functions well with incomplete maps as well. Certain maps lead to some miss-directions, but are quickly remedied as more information is gathered. More criteria could be used to determine intermediate road locations, or to limit mapping to a given range in order to improve performance. Research into this area using other decision based methods might be of interest for future projects. Overall, this system appears to function and prove that the method is viable for real time path planning with mapping information being updated. All code developed for this path planning, has been included in APPENDIX D:

Modified RRT Path Planning Algorithm for reference and to aid in future improvements. Now that all the systems have been developed, the next chapter discusses real data tests on and actual system to evaluate the developed algorithms. For reference, a flow chart of the entire developed system is also included to show the connection between image mosaicking, road detection and path planning (APPENDIX E: *System Flow Chart*).

5 REAL SYSTEM TESTING AND ACCURACY

This section details a test conducted using a real system using aerial imagery and adjustments for real system limitations. A study of varying parameters from image frequency/overlap, image resolution and syncing GPS locations. Adjustments based on sensor input were also considered and an explanation of flight plan considerations.

5.1 Accuracy of Aerial Imagery and Mosaic

Aerial imagery at this time is assumed to be geotagged in order to relate pixel values to locations useable by the ground vehicle for navigation. Although image resolution and capture frequency can be extremely high to improve accuracy, processing of this information is limited by computer vision algorithms. A flight conducted by the Unmanned Systems Lab (USL) collected imagery for radiation detection that gave geotagged images at a frequency of 1Hz. The Yamaha RMAX UAV (Figure 37) is used by USL for many different missions. It has a stereo vision system that collects images while flying a given flight plan and captures information from a GPS and inertial measurement unit (IMU). These images were used for stereo data analysis, but could also help test the parameters for real aerial imagery and the given mosaic method. Comparison of different image parameters can show the effects that image resolution and timing can have on the image mosaic and GPS correlation.

Figure 37: The Yamaha RMX UAV is used for many different missions for the USL and has captured aerial imagery that can be analyzed for these algorithms. Photo by author, 2014.

5.1.1 Image Transforms based on IMU input

One important assumption in the simulation testing was that the camera was in nadir view (straight down). This can be achieved using a camera gimbal if that hardware is available. However, using an IMU can help to estimate the camera's pose given the roll, pitch and yaw of

the aerial vehicle. Based on the projective area of the image, the IMU and altitude information can help to accurately transform the image. Figure 38 below shows the basic geometric principles for estimating the view. Although slight errors can occur based on the position of the camera to the IMU, the most important feature is the camera's altitude. The altitude defines the exact scale of the image and can give large deviations based on the camera's actual height and pose. Therefore, the chosen final vehicle should be equipped with a laser range finder to accurately measure altitude information. Other work has been done using feature detection to also help estimate pose-based on known feature locations and could be useful to improve or validate information from sensors on board the vehicle [48].

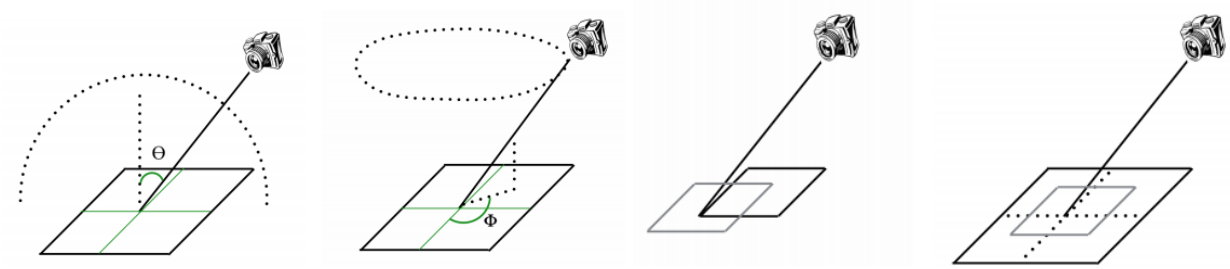


Figure 38: This figure illustrates the basic effect how camera pose defines the projective plane the image will actually show. Camera pose and altitude can help estimate the area of the image [A. Geniviva, J. Faulring and C. Salvaggio, "Automatic georeferencing of imagery from high-resolution,," Rochester Institute of Technology, Rochester, NY, 2014]. Used in fair use, 2014.

Based on the roll, pitch, and yaw information from an IMU on board and altitude information, the imagery of the RMAX flight was transformed to nadir view. Essentially a rotation matrix was calculated based on the IMU angles and applied to the image with altitude defining the scale. Below Figure 39 shows a sample of these transformed images. Although the transformation is not significantly large due to the vehicle maintain level flight, it helps with improving the accuracy of the nadir view assumption. This was developed as a MATLAB code and was used on all images from RMAX flights for a mosaic and road detection test performed in future sections. Image parameters were also modified to test the mosaic algorithm based on resolution/altitude and frequency/image overlap. These are detailed in the next two sections.

Figure 39: Images were transformed based on IMU information to closely approximate a nadir view for image mosaicking. Photos by author, 2014.

5.1.2 Image Resolution/Altitude Effects

Images were down-sampled to varying levels of resolution to see how it affect the image mosaic. This essentially simulated different camera resolutions or different flight altitudes to test flight parameters. Error of pixel values were calculated based on comparison to full resolution images at the lower resolution's pixel points. Obviously lower resolution would show quadratic growth in error if all points in original image were compared. The compared images are shown below (Figure 40) with their calculated error and image resolution. The plot for different resolutions' accuracies is shown below in Figure 41. Images were captured at 1728 x 2034 pixels (3.5MP) and down-sampled to relate to different altitudes. At an altitude of 30m, the camera used (Canon Powershot A810) on this flight had a ground resolution of 1.60cm/pixel (37 x 27.7m area).

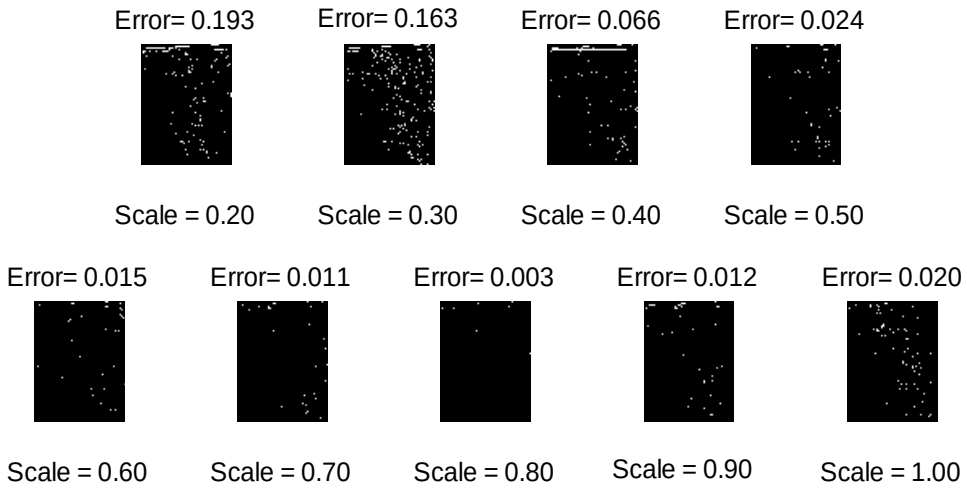


Figure 40: Binary image differences show the error of mosaicking at different resolutions. The images are compared to a downscaled version of the full resolution mosaic.

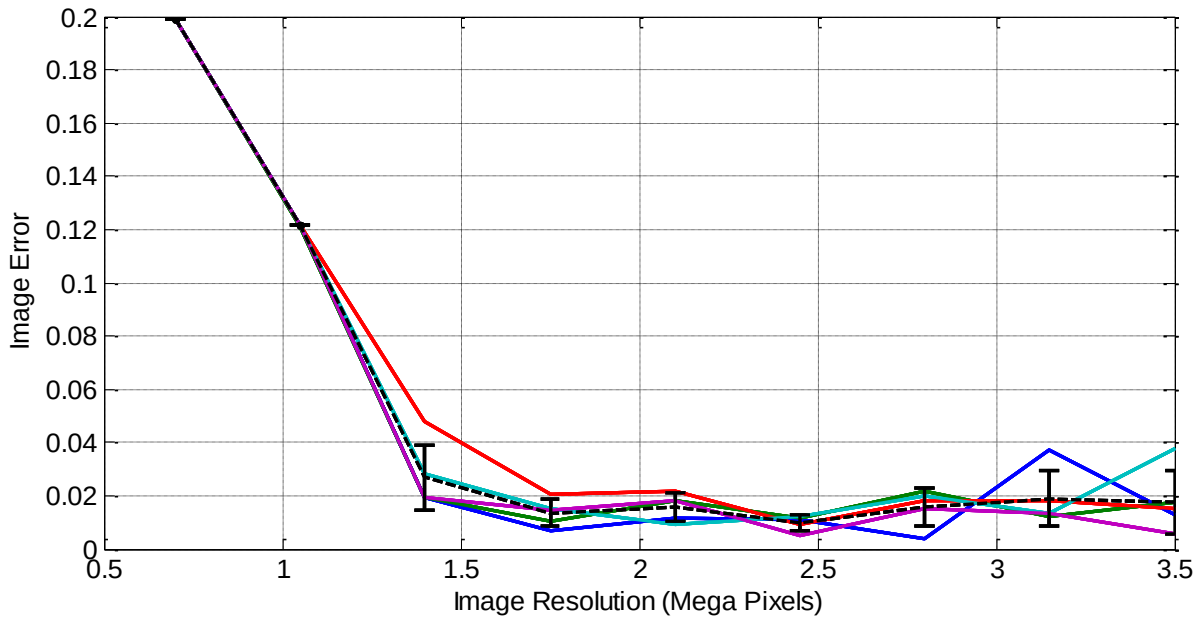


Figure 41: Multiple trials (colored lines) of mosaicking at different resolutions with standard deviation calculated (black bars). After reaching around ~0.5 of full resolution, error approached minimum. For this aerial imagery, the original resolution was 2304x1728 with an altitude of 30m (1.6 cm/pixel ground resolution).

From this imagery, it can be seen that around 0.5 the original resolution, mosaicking accuracy approaches a minimum. Based on the flight altitude of 30m and an image resolution of 2304x1728, the ground resolution for the original mosaic had a ground resolution of 1.6cm/pixel. However, at the 0.5 downscaled image, the ground resolution becomes 3.2 cm/pixel, which is a

60m altitude for the given camera. This gives a good design point for flight parameters, although the image features affect image mosaicking as well. Ground resolution can be achieved by controlling flight altitudes or camera resolution. Based on the camera system used on the RMAX, it is possible to fly at 60m and still achieve a similar mosaic. Again, using a laser range finder to more accurately measure altitude will be important for adjusting images based on pose to give accurate mosaics.

The other notable trend is that as the ground resolution is increased, the error stops improving greatly and does not reach zero at the same original resolution. The average value remains below 5%. Also, the variation of the error seems to increase. This is most likely due to the function of feature detection and the number of features found. As the resolution of images are increased, more features can be found and matched to create the homography. Based on the discussed RANSAC method, as more features are found, it is possible to get slightly different mosaics for different runs. Therefore, higher resolution imagery is good to improve ground resolution, but does not improve the overall map mosaic. However, this is good for real time applications because a lower resolution image can be used to calculate the mosaic and homography transform, which should improve computation time. The transform could then be applied to higher resolution images later to create a more accurate road map (based on road detection) or even a full terrain map.

Although ground resolution is an important design point and can help to define the parameters of the camera and flight altitude. Flight speed, image frequency and overlap are other design parameters. The next section discusses a frequency test conducted to understand those effects on mosaicking accuracy.

5.1.3 Image Frequency/Overlap Effects

During the RMAX flights, images were taken at a frequency of 1Hz. Based on how often the images are taken, more overlapping of areas can be used for mosaicking. In order to simulate lower frequencies (and therefore less overlap), images were removed from the sequence. Image mosaicking was run with these removed images. Similar to the resolution test, image differences were calculated and shown below (Figure 42) for 0.5, 0.33 and 0.25 Hz. Overlap based on the image mosaic was also calculated to plot the error against (see Figure 43). This can be used to define the relationship between frequency and airspeed.

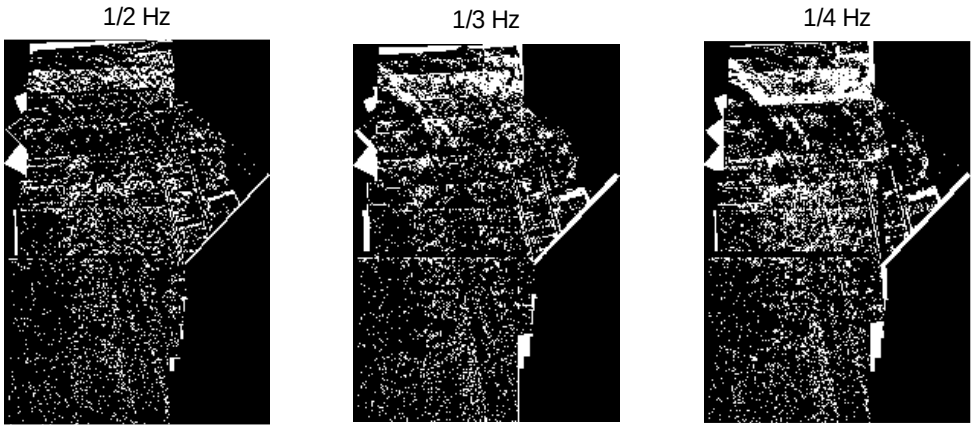


Figure 42: Image differences from decreased frequency (from 1Hz) were calculated to see the effect on mosaicking.

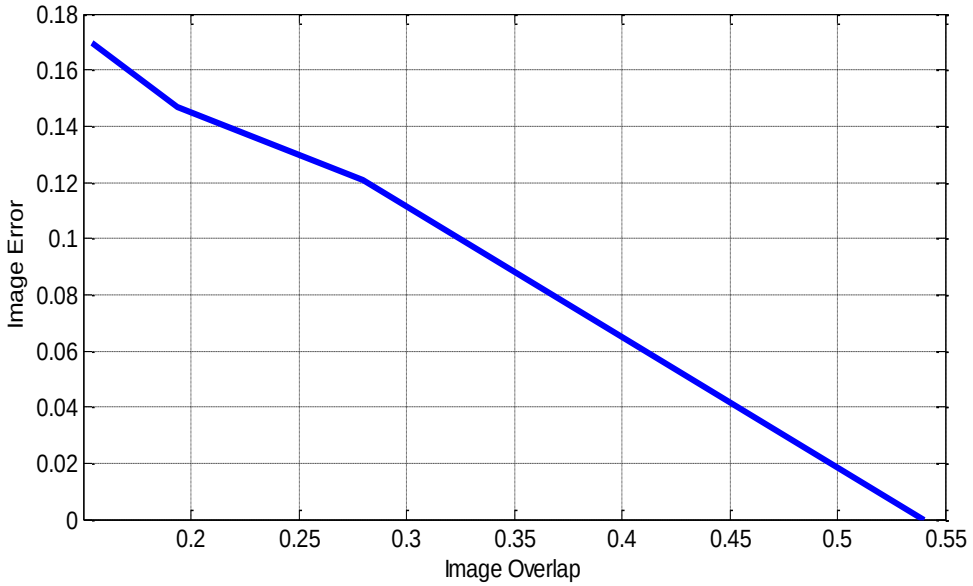


Figure 43: The image error of reduced frequency mosaicking shows that as overlap increases, the error decreases. Image error of zero is due to the image difference was compared to the original image (with overlap calculated for it).

As can be seen from the plot above, the mosaic image error decreases as overlap increases. An overlap of 50% shows the good performance that meets the below 5% error expected from image mosaic variance. This overlap value matches up with other studies conducted using image stitching [49]. This image overlap can be used for planning image frequency based on the altitude and speed. Different platforms have been developed for planning this aerial mapping and can even use camera parameters to design triggering speed.

Ardupilot has a built-in survey program for setting up camera triggers to map an area [50]. This is currently used for surveying areas, but allows for automatic trigger based on distance traveled verified by GPS and airspeed sensors. Figure 44 shows how triggering can be automated for planned missions. This gives a simple way to control image triggering based on image overlap given the camera parameters, flight speed and parameters.

Figure 44: Ardupilot's Mission Planner allows for designing camera triggering based on image and flight parameters to ensure overlap based on altitude and flight speed M. Osbourne, "Ardupilot," 3DRobotics, 2014. [Online]. Available: http://planner.ardupilot.com/. [Accessed 2014]]. Used in fair use, 2014.

Now that the important flight parameters have been investigated, it should be possible to design flights for different requirements. Understanding the need of ground resolution based on altitude and image resolution as well as the overlap required for certain levels of accuracy for the mosaic can help to plan flight parameters for real systems. The current developed TSP traveling wave system can use these parameters for an improved flight plan. It currently outputs a flight plan based on a set of known road locations and a basic model of a traveling wave applied to TSP. However there are important considerations for improving this system. As stated in the

previous chapter, the current UAV flight plan is not optimized and gives a flight plan this is difficult for vehicles such as the RMAX to navigate. There are many course changes and inefficient sectioning. It also does not accurately factor in the field of view of the camera for waypoints. The Mission Planner system developed by Ardupilot (shown above) takes in camera information for accurate triggering based on the field of view of the camera. Given a known altitude and camera resolution, the field of view area is easily determined and could be applied as a filter to improve generated flight plans. By using the determined field of view area, multiple road locations can be imaged simultaneously. Essentially, using the field of view area can eliminate intermediate planned locations from the TSP traveling wave solution because they would be imaged by the previous location. This should give a simpler output to the autopilot and less harsh path commands to the vehicle. However, this overlap study shows that for accurate mosaicking, image frequency should still remain high enough to capture enough features (~50% image overlap) and not miss changes in course if the vehicle speed is high. These considerations for image overlap, field of view area, and image frequency should give good criteria for UAV flight planning applications. As a final study, image mosaicking was compared using geo-tagged images to determine the accuracy of correlating this information to measured ground GPS points.

5.2 GPS correlation to Image mosaics

From the RMAX flight, images were geo-tagged based on the GPS information from the vehicle. In order to help determine GPS locations of roads and areas of interest, ground truth GPS points were measured and compared to GPS points interpolated from the geo-tagged images. The center point of the image before transformation was assumed to be the GPS location from the sensor. Although this could be untrue for other camera angles, this should be a safe approximation based on the nadir view assumption/adjustment due to IMU information.

Based on the calculated mosaic, dimensions between points can be calculated for pixel distance. Then, the distance between two image center points (after transformation) can help define the GPS scale of the image (Figure 45). By dividing the GPS distance by the pixel distance between image centers, the GPS scale can be calculated and applied to the corners of the image in order to create a meshed grid of GPS points across the full mosaic. By defining the points of the mesh by the pixel coordinates, the goal would be to geo-reference each pixel location to a GPS coordinate.

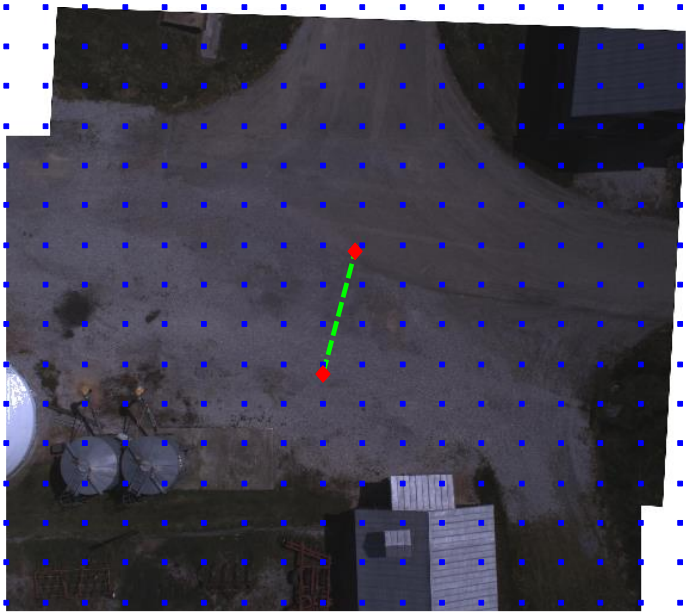


Figure 45: A GPS meshed grid (blue) can be calculated based on GPS tagged image center (red) by calculating the distance (green) between points and applying the scale to the corners of the image. Photos by author, 2014.

As stated, 5 ground truth GPS points were measured in order to compare to the generated GPS mesh. A GPS error was then calculated based on the distance between the ground truth and GPS locations for each corresponding pixel. Figure 46 below shows the comparison of ground truth to calculated GPS locations. Averaging the mesh over multiple points for a sequence of images was hypothesized to aid in reducing GPS drift effects. Figure 47 shows that averaging over 5 points does not noticeably reduce the variance. This is most likely due to the combined GPS variance in the vehicle as well as ground truth points.

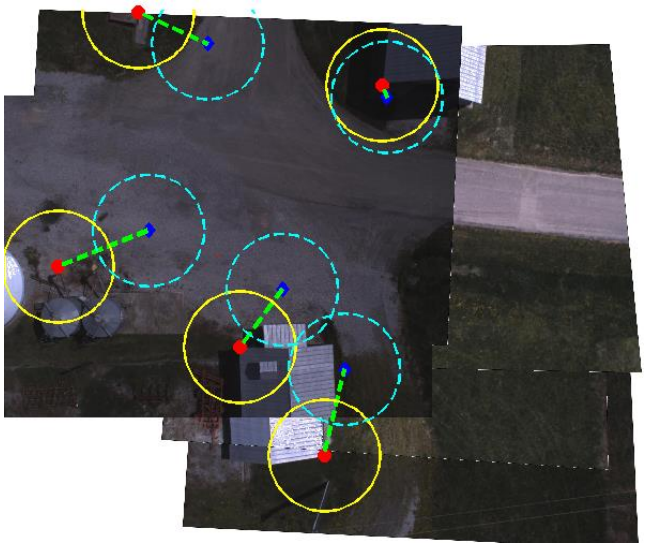


Figure 46: GPS ground truth points (red) were compared to calculated GPS locations (blue) from geo-tagged images (uncertainty shown). Distances measured between points show an average error around 15 feet. Photos by author, 2014.

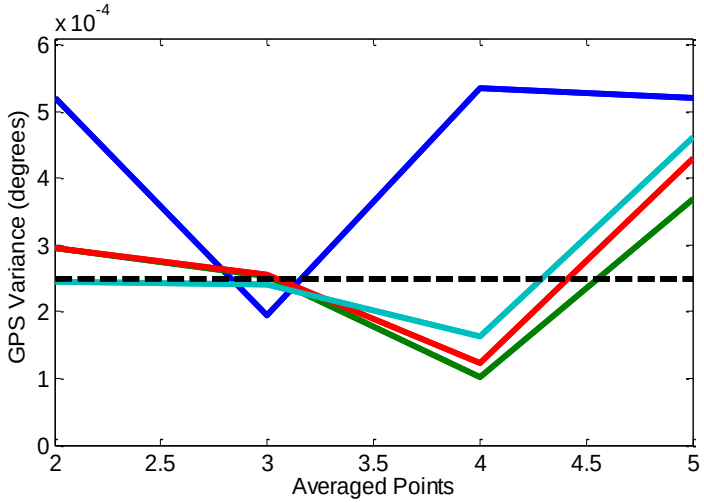


Figure 47: GPS difference to ground truth was calculated based on total point averages for the GPS mesh. Average variance was found to be 2.5E-4 degrees, which correlates to ~10ft with ranges reaching ~20ft.

From this study, it shows that the GPS error based on mosaics is only slightly worse than current GPS resolution. The ground truth points had an uncertainty of ± 10 feet (pictured above with red points and yellow circles). If the mosaic system was used, it would be expected that the error would also be ± 10 feet as well. The calculated points from the UAV GPS and meshing

grid are shown to have errors (blue points with turquoise circles of uncertainty) outside the ground truths uncertainty. This shows there are more improvements to be done with calculating ground GPS from UAV imagery and sensor data. From the above image, it appears that there is possibly some scaling errors based on the direction of the calculated errors. This could be due to poor averaging of the GPS to pixel scale or a factor of the image scaling, but needs for research.

From the measurement of distances and averaging of GPS variance, an error of 10 to 20 feet was found. These values matched up with another study that used ground control points to improve accuracy of aerial imagery [51]. They got a similar error distanced based on 5 ground truth points. This shows that the aerial GPS could be used as a means to create control points if more points are used. In the previous study, error has been reduced down to 3-4 feet; however, this required 21 ground points. This would require a high number of overlapping images, but could be the trade-off for increased accuracy. If computation time and flight time were not as important, more accurate GPS correlation could be achieved. More flights could be designed to measure increased accuracy and test if more GPS points could improve accuracy. Although road detection has much more accurate values, GPS accuracy seems to be the limitation on relaying road locations and controlling the ground vehicle. In a future section, the needs for the ground vehicle to still effectively navigate the roads are discussed.

5.3 Final Real System Testing

In order to finally test the validity of the developed algorithm, other imagery from a different RMAX flight was used to mosaic, perform road detection and conduct path planning. Images were transformed based on the IMU data to give the nadir view before being run through the algorithms. Figure 48 shows a complete mosaic and developed path for rounding a corner. It effectively maps and creates a path for the vehicle to navigate. Although the scale of this test was limited, this illustrated the robustness of the algorithm to new data sets without recalibrating the k-means or image morphing parameters. These parameters should still be considered for other flights and adjusted if necessary. However, this shows that flights with the goal of image mosaicking and path planning on a large scale are viable and should be one of the next steps to understand the effects of larger areas and flight plans. Also, the accuracy of GPS is still limited

but can be used as basic waypoint generation for path navigation. The next section details what is still required for the ground vehicle to navigate the terrain.

Figure 48: A simple test incorporating all facets of the developed algorithm shows the robustness of the algorithm to data without calibrating parameters. The images are still effectively mosaicked, road detection and path planned for a ground vehicle. Photos by author, 2014.

5.4 Ground Vehicle Considerations

Based on the known errors for GPS location correlation, more accurate GPS waypoints would require too many mosaicked image. Therefore, the aerial mapping and road detection are currently only effective for the ground vehicle by accurately determining passable paths and generating a simple path planning algorithm. Essentially, it will determine the best route for the vehicle to take. As stated before, more work can done to verify the path based on different vehicle parameters for control and size. However, there is still plenty of validity in using mosaicking and road detection for high resolution mapping means that could be used in other applications such as processing information for identifying areas of interest in disaster situations.

This would be a way to update goals of the ground vehicle path planning depending on the situation.

In order to navigate the generated path, the ground vehicle will need to receive aerial images, GPS waypoints and perform its own sensing to avoid obstacles and reach the waypoints. Use of a RF frequency connection to the ground vehicle needs to be maintained to update waypoints and share imaging information. Work on using UAVs as a radio-relay was conducted by another member of the USL to share this information even in highly variable terrain to maintain communication [52]. As of right now, based on the computation requirements, the image processing should be conducted on the ground vehicle based on weight limitations for the aerial vehicle. This means it will require a computer on-board. Currently all the algorithms were run on a simple duo-core processor using a laptop, so cheap mobile systems could easily be developed. Also, the vehicle will need some obstacle avoidance systems using Lidar, computer vision or sonar measurements to perform SLAM (simultaneous localization and mapping) methods. This is a well-researched topic in robotics and ground vehicle operation. This local mapping can be used to help validate the 2D road and obstacle detection if desired, or can be kept local to the ground vehicle.

As stated, the design of the ground vehicle was not a major focus of the thesis, but more optimization could be researched in the future. The consideration of vehicle capabilities and hardware systems could help to improve the path planning and optimize the image processing algorithms. In the next section, the algorithms and methods are discussed and future work planned in order to advance this system to a real world application and conduct more testing.

6 DISCUSSION AND FUTURE WORK

6.1 Computer Vision Discussion

6.1.1 Image Mosaicking

After developing the image mosaicking algorithm and continuous extension, areas were able to be effectively mapped with little artifacts or errors. More focus could be put on quantifying and improving image warping accuracy in the future. Although the SIFT method is one of the most effective methods currently used, more validation of mosaics created could be compared back to ground truth data. Developing methods to compare incoming data to known satellite imagery, or linking road locations could also help validate maps and quantify performance. GPS correlation shows some inherent error but could improve with more imagery and testing. Because GPS drift is such a common problem, this would be an interesting challenge as a small number of images might not yield accurate GPS points, but a large group of images could help to average and smooth out drifting errors.

The most useful improvement for image mosaicking will definitely be in increasing computation time. Although the system developed had a loop time of 3-4 seconds to process an image, the need to do road detection and path planning as well, means reducing its individual components' computation time would greatly improve the system. Also, if the algorithm is improved, on board processing could become possible on the UAV, which would reduce the hardware requirements. One major method of improving the speed is using "MEX-files" in MATLAB in order to compile the code to an executable that can run in C or C++ [53]. This allows subroutines or functions to be run more quickly and possibly on embedded systems instead of the MATLAB software itself. Computation improvements are always important and should be a future focus of this project. This can also help with the GPS correlation because processing more images quickly could give more control points to reduce errors of GPS drift.

6.1.2 Road Detection

Based on the road detection testing and algorithm, an effective method using nearest-neighbor has been developed which reduced the computation time by over 70% and remained accurate (average of 1-2% error from regular k-means). Tracking of road points in between sequential images also worked effectively based on the affine transformation assumption. The major concerns are with the k-means errors that occur due to empty cluster or poor calibration

with image morphing operations. Studies of ways to improve this and tune parameters for different environments could improve accuracy of road location detection. Also, work has been done using segmentation to extract more features and prioritize areas in a map to help identify areas of interest like roads, buildings, and vehicles in urban settings [54]. Although this level of detail was not a goal for this work, this improved segmentation and identification could give more information for future path planning goals such as following targets or exploring an area of interest. Also, the road tracking shifting using affine assumptions or IMU adjustments could yield small errors that could translate to large deviations if larger areas are mapped. Studies on the effects of using these methods on even larger scale mapping could help improve performance and allow for wider capabilities.

6.2 Path Planning Discussion

6.2.1 Aerial Vehicle Planning

The traveling wave TSP solution that was developed was a very quick and simple model solution. Its distance separation for sectioning was arbitrary and did not always yield optimal connections between sections. The developed flight plan would be difficult for a UAV to navigate and was inefficient. Using known camera parameters and flight altitudes the areas of images could be defined. This could remove some complex maneuvers in the system by simplifying the flight plan and reducing the waypoints required. As stated in the literature review, better systems could be developed that take advantage of prior map information. Wave front algorithms are similarly developed systems with higher resolution for distance criteria. Essentially, wave front propagation uses Dijkstra's algorithm with a number of stages or checkpoints to reach the goal [43]. This is similar to the traveling wave TSP solution developed because it iteratively calculates the path based on the stages developed. However it optimizes the number of stages and factors changes of stages into the total cost of the path [43]. This puts greater focus on optimizing the section transitions. Therefore, a smoother connection between each stage could minimize the flight time and organize the flight plan.

More recent applications of wave front propagation include level sets and fast marching. Level sets create a gradient map used for calculating shortest paths. It factors in topology, corners and obstacle curvatures to generate propagating fronts based on gradients and partial differential equations [55]. Application of the fast marching find the shortest path similar to

Dijkstra's algorithm based on the gradient maps generated by level sets. This method has shown promise in tight, complicated environments, long optimal paths and dynamic situations [56]. It can be applied in both 2D and 3D maps if altitude adjustments needed to be performed for the aerial vehicle [57]. Also, it has been applied in time dependent flow fields that could be used to factor in environmental factors such as wind for vehicle control [58]. Overall, from this further research and understanding of flight plan goals, more advanced methods can be used to take advantage of known environmental information and give more focus on prior planning of the area. Efficiency of mapping and development of a quick flight plan can help to allow faster deployment and navigation of the ground vehicle because information is gathered quicker and aerial vehicle controlled more effectively.

The given solution also made a lot of assumptions about knowing road locations and prioritizing areas to map as well as the vehicle capabilities. In terms of road location extraction, OpenStreetMap (http://www.openstreetmap.org) is an open source mapping tool that uses aerial imagery and GPS devices to keep updated maps. Integration of this software to develop path plans, and simulate missions could prove useful. Extraction of GPS coordinates and features from a previous map to develop the flight plan would be an interesting project and help to aid with syncing newly captured aerial imagery to those same GPS locations. This could be applied to the discussed methods of wave propagation above because more advanced map information would be extracted and utilized. Information such as road widths, building sizes and possibly topography could be found from this source and processed. These methods require this higher level of terrain analysis and could benefit from vehicle parameter definitions as well to ensure path planning analyses are complete and optimized.

As stated multiple times, consideration of vehicle capabilities for a specific vehicle will be important for implementing this in a final solution. However, performed studies on real data and varied flight parameters showed design needs for flight plans. A 50% overlap should give accurate mosaicking. Image frequency and speed are a function of this overlap. Also, a ground resolution around 3 cm/pixel worked well for a test case at 30m, but this is dependent on the features and matching involved with the scene. It does show that a lower resolution can be used for image mosaicking than the full mapped area. Because the developed simulation allows for customization of discretizing the flight plan and image resolution, it is possible to still control

these parameters for future studies and customize it to a specific vehicle and application should they be chosen.

Another important study that should be considered is the lead time required for the UAV to begin mapping the area and yield ground vehicle path planning. Although these vehicle parameters are not known at this time, developing a real world system should consider these. Scale of the map, complexity of the road paths, and hardware used are all important factors. A design problem of vehicle speed and navigation characteristics compared to the computation speed of the hardware can be weighed and optimized in order to process information efficiently, and begin ground vehicle deployment at the right time. By carefully planning and estimating the time for flight plans to be conducted, the UAV and UGV can positioned and deployed at optimized times to minimize total ground vehicle traversal time. This would be an interesting optimization problem based on the vehicles characteristics and environments involved.

6.2.2 Ground Vehicle Path Planning

The RRT path planning algorithm was chosen for its quick computation and efficient planning. It was modified in order to process data from the road detection and mosaicking developed. Overall, it was a well-developed code, but more information could be used to improve the paths. The vehicle requirements for turning, size and other parameters could greatly affect areas that are currently evaluated as roads. However, at this time, there is no way to ensure these vehicle parameters could be met, because they could vary for a specific application.

A* path planning would be a great improvement once vehicle characteristics are defined. One could take advantage of A* heuristics, or experience-based methods, using prior known mapping information. Many examples of different A* heuristic methods have been developed and an overview of these methods has been detailed [26]. There are methods of using A* with a dynamic or incomplete map for quick re-planning of a ground vehicle path [59]. This could be useful for this application due to the desire for real time mapping with incomplete information being used. Because Dijkstra's algorithm is used for the aerial vehicle planning, and prior road locations are already known, this information can also be used in the heuristic functions in order to prioritize the shortest distance, but still focus on quick computation time. Essentially, because a shortest path is already calculated, this can help guide the path planning and improve efficiency of ground vehicle control.

Development of a system to feedback information to the aerial vehicle in order to know when mapping of the area was insufficient should be one of the next focuses. A basic system to handle this for the aerial flight plans could be restarting the algorithm with a new ground vehicle position. However a system for clearly identifying and feeding back this information has not been developed. This could also help aide with prioritizing targets that change in the middle of a mission. Areas of interest could change while images are being collected. Different image analysis would need to be done on this, but could re-direct ground vehicles as needed. Based on the intermediate goals developed, the RRT path planning algorithm would not even need to be adjusted given this change, but other path planning strategies could also factor in this adjustment.

6.2.2 Vehicle and Hardware Integration

In order to improve and understand the limitations of the algorithms, more studies with real hardware and processing capabilities should be pursued. Currently, Ardupilot is a widely used open source autopilot system that could be used as a vehicle platform for both the aerial and ground vehicles. It utilizes GPS for its major control of waypoints and could be implemented fairly inexpensively. Depending on the platform used however, the formatting and utilization of the algorithms developed in this paper would need to be modified. Ultimately, the final goal for the project would be to deploy two real systems (UAV and UGV) that can communicate and share information similar to our simulation to reach a goal. The UAV should be capable of communicating and relaying imagery/GPS information to the ground vehicle. Also, the ground vehicle needs to be developed in order to use SLAM with sensors and computing hardware for image processing. Most of the work done at this time proves the effectiveness of the method and algorithms, but the integration will be a great challenge in terms of communication and data transfer as well as hardware limitations for control.

7 SUMMARY & CONCLUSIONS

The major goals for this work were focused on mapping terrains to determine road locations based on aerial imagery. Computer vision algorithms were developed for continuous image mosaicking and road detection to meet these mapping needs. Path planning was developed for both the aerial vehicle and ground vehicle to efficiently map the area and calculate paths quick for navigation.

For image mosaicking, SIFT feature detection proved to be an accurate method to stitch images. Image resolution was kept relatively high and a dynamic cropping method was developed in order to efficiently process information and improve computation time. More improvements to reduce computation time and validate mosaics based on ground truth information could be major focuses for future work. Based on these mosaicked images, road detection using k-means segmentation and binary image morphing was developed. Careful tuning of these parameters based on the scene should be factored into applications. Also, improvements to computation time were included by using nearest neighbor calculations with k-means only used to calibrate at a given interval. This reduced the speed of road detection by 70% for our given resolutions. The continuous extension for road detection tracks road locations and uses X-Y plane shifting calculated from the image mosaic in order to generate a larger road map based on the entire environment.

Path planning was also developed for the aerial and ground vehicle. It was assumed the vehicles could handle basic GPS coordinate commands at this time, although more work could be done to refine these systems for optimal control. The aerial vehicle utilized previously known road locations as a means to develop a flight plan to efficiently cover the whole area of interest from the start to a goal. In order to prioritize locations for ground vehicle use, a traveling wave was applied to the known locations with limitations on range towards the target location. A fixed-start open genetic algorithm was then applied to each wave iteration. This put focus on mapping closer targets to give initial path planning goals for the ground vehicle. Other path planning methods were discussed that utilize wave front propagation in order to improve this system. RRT path planning was used for the ground vehicle because it can quickly develop paths for real-time applications. Modifications were made to this algorithm in order to more efficiently develop paths and process data from the specific road detection method chosen. Handling of

incomplete maps was also considered to select intermediate goals. More work for the ground vehicle path planning should include consideration of vehicle parameters and restrictions.

Real flight testing data was used to validate this work. It showed a mosaic error <5% with repeatedly run algorithms. Altitudes of 60m for a 3.5MP camera at a frequency of 1HZ showed good performance with 50% overlap based on a test flight using a Yamaha RMAX UAV and this benchmark can be used for baseline parameters to design other systems. Based on all of this work, an effective method for data collection and image processing has been developed. Continuous image mosaicking and road detection algorithms were developed and gave insight into the important considerations for real system use. Computation time was improved for these algorithms and show promise for real-time applications. Focus on the hardware and vehicle parameters should be a major focus for future work on this project in order to allow for actual applications in real world systems.

REFERENCES

[1] R. Szeliski, Computer Vision: Algorithms and Applications, Springer, 2010.

[2] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *Internation Journal of Computer Vision,* vol. 60, no. 2, pp. 91-110, 2004.

[3] Herbert Bay, A. Ess, T. Tuytelaars and L. Van Gool, "Speeded-Up Robust Features (SURF)," *Computer Vision and Image Understanding,* vol. 110, pp. 346-359, 2008.

[4] T. Tuytelaars and K. Mikolajczyk, "Local Invariant Feature Detectors: A Survey," *Foundations and Trends in Computer Graphics and Vision,* vol. 3, no. 3, pp. 177-280, 2007.

[5] R. Szeliski, "Video Mosaics for Virtual Environments," *IEEE Computer Graphics and Applications,* vol. 16, no. 2, pp. 22-30, 1996.

[6] E. Dubrofsky, "Homography Estimation," The University of Britsh Columbia, Vancouver, 2009.

[7] M. Hansen, P. Anandan, K. Dana, G. van der Wal and B. P, "Real-time Scene Stabiliaztion and Mosaic Construction," in *Applications of Computer Vision*, Sarasota, FL, 1994.

[8] Y. Lin and S. Saripalli, "Road Detection from Aerial Imagery," in *IEEE International Conference on Robotics and Automotion*, Saint Paul, Minnesota, 2012.

[9] P. P. Singh and G. R. Garg, "Automatic Road Extraction from High Resolution Satellite Image using Adaptive Global Thresholding and Morphological Operations," *Indian Society of Remote Sensing,* vol. 41, no. 3, pp. 631-640, 2013.

[10] Y. Cao and L. Yan, "Automatic Road Network Extraction from UAV Image in Mountain Area," *Image and Signal Processing (CISP),* vol. 5, pp. 1024-1028, 2012.

[11] F. J. Estrada and A. D. Jepson, "Benchmarking Image Segmentation Algorithms," *Internation Journal of Computer VIsion,* vol. 85, pp. 167-181, 2009.

[12] G. Gajanayake, R. Yapa and B. Hewawithana, "Comparison of Standard Image Segmentation Methods for Segmentation of Brain Tumors from 2D MR Images," in *Fourth International Conference on Industrial and Information Systems*, Sri Lanka, 2009.

[13] E. Santamario, F. Segor and I. Tchouchenkov, "Rapid Aerial Mapping with Multiple Heterogeneous Unmanned Vehicles," in *10th International ISCRAM Conference*, Baden-Baden, Germany, 2013.

[14] H. Choset and P. Pignon, "Coverage Path Planning: The Boustrophedon Cellular Decompition," in *International Conference on Field and Service Robotics*, Canberra, Australia, 1997.

[15] M. Dille and S. Singh, "Efficient Aerial Coverage Search in Road Networks," in *AIAA Conference on Guidance, Navigation, and Control*, Boston, Massachusetts, 2013.

[16] H. Safadi, "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots," Stanford Univerity, Standford, VA, 1985.

[17] K. Uyanik, "A study on Artificial Potential Fields," Middle East Technical University, Ankara, Turkey, 2011.

[18] J.-O. Kim and P. Khosla, "Real-time obstacle avoidance using harmonic potential functions," *Transactions on Robotics and Automation,* vol. 8, no. 3, pp. 338-349, 1992.

[19] J. Guldner and V. Utkin, "Sliding Mode Control for Gradient Tracking and Robot Navigation Using Artificial Potential Fields," *Transactions on Robotics and Automation,* vol. 11, no. 2, pp. 247-254, 1995.

[20] Y. K. Hwang and N. Ahuja, "A Potential Field Approach to Path Planning," *IEEE Transactions on Robotics and Automation,* vol. 8, no. 1, pp. 23-32, 1992.

[21] L. E. Kavraki, J.-C. L. Švestka and M. H. Overmars, "Probablistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces," *IEEE Transactions on Robotics and Automation,* vol. 12, no. 4, pp. 566-579, 1996.

[22] J. Miura, "Support Vector Path Planning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Beijing, China, 2006.

[23] C. Qingyang, S. Zhenping, L. Daxue, F. Yuqiang and L. Xiaohui, "Local Path Planning for an Unmanned Ground Vehicle Based on SVM," *International Journal of Advanced Robotic Systems,* vol. 9, pp. 246-259, 2012.

[24] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz and S. Thrun, "Anytime Dynamic A*: An Anytime, Replanning Algorithm," Canergia Mellon University, Pittsburgh, PA, 2005.

[25] S. Koenig, M. Likhachev and D. Furcy, "Lifelong Planning A*," *Artificial Intelligence*, vol. 155, no. 1-2, pp. 93-146, 2004.

[26] D. Ferguson, M. Likhachev and A. Stentz, "A Guide to Heuristic-based Path Planning," in *International conference on automated planning and scheduling (ICAPS)*, Monterey, CA, 2005.

[27] S. M. LaValle, "Rapidly-Exploring Random Trees: A New Tool for Path Planning," Iowa State University, Ames, IA, 1998.

[28] J. J. Kuffner and S. M. LaValle, "RRT-Connect: An Efficient Approach to Single-Query Path Planning," in *IEEE International Conference on Robotics and Automation*, San Francisco, CA, 2000.

[29] E. Wiggin, "Image Mosaic using SIFT," MathWorks, 24 March 2011. [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/30849-image-mosaic-using-sift. [Accessed 27 February 2014].

[30] R. Hartley and A. Zisserman, Multiple View Geometry in Computer Vision: Second Edition, Cambridge, UK: Cambridge University Press, 2004.

[31] P. Rosengren, "Calculating Homography," Royal Institude of Technology, 2 May 2007. [Online]. Available: http://www.csc.kth.se/~perrose/files/pose-init-model/node17.html. [Accessed April 2014].

[32] K. G. Derpanis, "Overview of the RANSAC Algorithm," York University, Toronto, Canada, 2010.

[33] M. A. Fischlers and R. C. Bolles, "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography," *Communications of the ACM,* vol. 24, no. 6, pp. 381-395, 1981.

[34] S. Lazebnik, "Image Warping and Morphing," Univeristy of North Carolina, Durham, 2008.

[35] National Oceanic and Atmospheric Administration, "Hurricane SANDY Response Imagery," National Geodetic Survey, 2012.

[36] T. Maruyama, "Real-time K-Means Clustering for Color Images on Reconfigurable Hardware," in *International Conference on Pattern Recognition* , Hong Kong, 2006.

[37] D. Budden, S. Fenn, A. Mendes and S. Chalup, "Evaluation of Colour Models for Computer Vision using Cluster Validation Techniques," in *RoboCup 2012: Robot Soccer World Cup XVI*, Berlin Heidelberg, Springer, 2013, pp. 261-272.

[38] T. Kanungo, D. Mount, N. Netanyahu, C. Piatko, R. Silverman and A. Wu, "An Efficient k-Means Clustering Algorithm: Analysis and Implementations," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 24, no. 7, pp. 881-892, 2002.

[39] MATLAB Documentation, "bwmorph," MathWorks, 2014. [Online]. Available: http://www.mathworks.com/help/images/ref/bwmorph.html. [Accessed 10 2013].

[40] MATLAB Documentation, "bwlabel," Mathworks, 2014. [Online]. Available: http://www.mathworks.com/help/images/ref/bwlabel.html.

[41] J. Morris, "Data Structures and Algorithms," 1998. [Online]. Available: https://www.cs.auckland.ac.nz/software/AlgAnim/dijkstra.html. [Accessed 9 May 2014].

[42] J. Kirk, *Fixed Start Open Traveling Salesman Problem - Genetic Algorithm,* MathWorks, 2008.

[43] S. LaValle, "Planning Algorithms: Wavefront propagation algorithms," Cambridge University Press, 20 4 2012. [Online]. Available: http://planning.cs.uiuc.edu/node372.html. [Accessed 8 2014].

[44] A. Zelinksy, R. Jarvis, J. Byrne and S. Yuta, "Planning Paths of Complete Coverage of an Unstructured Environment by a Mobile Robot," in *International Conference on Advanced Robotics*, Atlanta, GA, 1993.

[45] University of Illinois Control Systems Laboratory, "Rapidly Exploring Random Tree (RRT) Path Planning," [Online]. Available: http://coecsl.ece.illinois.edu/ge423/spring13/RickRekoskeAvoid/rrt.html. [Accessed May 2014].

[46] P. Corke, "Robotics Toolbox," February 2013. [Online]. Available: http://www.petercorke.com/robot/. [Accessed 10 January 2014].

[47] Nazarbayev University ARMS Lab, "MATLAB Toolbox of RRT, RRT* and RRT*FN algorithms," 18 October 2013. [Online]. Available: http://arms.nu.edu.kz/research/matlab-toolbox-rrt-based-algorithms. [Accessed May 2014].

[48] A. Geniviva, J. Faulring and C. Salvaggio, "Automatic georeferencing of imagery from high-resolution,," Rochester Institute of Technology, Rochester, NY, 2014.

[49] V. Rankov, R. Locke, R. Edens, P. Barber and B. Vojnovic, "An algorithm for image stiching and blending," *SPIE: Image Aquisition and Processing,* vol. 5701, pp. 190-199, 2005.

[50] M. Osbourne, "Ardupilot," 3DRobotics, 2014. [Online]. Available: http://planner.ardupilot.com/. [Accessed 2014].

[51] D. Vericat, J. Brasington, J. Wheaton and M. Cowie, "Accuracy assessment of aerial phototgrpahs aquired using lighter-than-air blimps: low-cost tools for mapping river corridors," *River Research and Applications,* vol. 25, no. 8, pp. 985-1000, 2009.

[52] B. Krawiec, "A*-based path planning for an unamnned aerial and ground vehicle team in a radio repeating operation," Virginia Tech, Blacksburg, VA, 2012.

[53] MathWorks Inc, "Introducing MEX-Files," MathWorks, Boston, MA, 2014.

[54] S. Hinz and A. Baumgartner, "Automatic extraction of urban road networks from multi-view aerial imagery," *ISPRS Journal of Photogrammetry and Remote Sensing,* vol. 58, no. 1-2, pp. 83-93, 2003.

[55] J. Sethian, "A fast marching level set method for monotonically advancing fronts," *Proceedings of the Nation Academy of Sciences,* vol. 93, pp. 1591-1595, 1996.

[56] W. Cheng-Yuan and L. Jing-Sin, "Level Set and Fat Fast Marching Method for Normal and Dynamic Path Planning of Pursuit-Evasion Problem," in *International Conference on Autmation and Logistics*, Hong Kong and Macau, 2010.

[57] L. De Sanctis, S. Garrido, L. Moreno and D. Blanco, "Outdoor Motion Planning Using Fast Marching," in *Twelfth International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines*, Instabul, 2009.

[58] T. Lolla, M. P. Ueckermann, K. Yigit, P. J. H. Jr. and P. F. J. Lermusiaux, "Path Planning in Time Dependent Flow Fields using Level Set Methods," in *International Conference on Robotics and Automation*, Saint Paul, Minnesota, 2012.

[59] S. Koenig and M. Likachev, "Fast Replanning for Navigation in Unknown Terrain," *IEEE Transactions on Robotics*, vol. 21, no. 3, pp. 354-363, 2005.

APPENDICES

APPENDIX A: ***Flight Simulation from Input Flight plan***

APPENDIX B: ***Continuous Mosaicking Code***

APPENDIX C: ***k-Means and Image Morphological Testing***

APPENDIX D: ***Modified RRT Path Planning Algorithm***

APPENDIX E: ***System Flow Chart***

APPENDIX A: *Flight Simulation from Input Flight plan*

```
%This related to the completion of Scott Radford's Master's
% Thesis at Virginia Tech in the Unmanned Systems Lab
%"Real-time roadway mapping and ground robotic path planning via unmanned
aircraft"
%Scott Radford @ VT USL - scottyr@vt.edu
%Date: 04/2014
%Description: Used to create a simulated flight given a large image file by
%             imoorting a flight path, generating a smaller sample image
%             and augmenting extra frames based on random drift

%% Customize Image Size and Frame Info

close all; clear all; clc;

flightPoints = load('flightPoints.mat'); % flight points determined (Nx2 matrix
of points)
pointRoute = flightPoints.pointRoute;

imageWidth = 640;
imageHeight = 480;

numFrames = length(pointRoute); %note limit is set 999,999 frames currently
(change sprintf value below to increase)
simulatedFrames = 1; %Interpolate images in between points (value of 1 means no
interpolation)
baseImageUpScale = 3; %can be used to increase resolution of largers image by
interpolation
bufferingSize = 1000; %to ensure images don't overlap an edge
drift = 3; %pixels in each direction assuming flight variation

imageWrite = true; %write all images including simulated if any
videoWrite = false; %write all images to video .avi format
videoName = 'video';
imageName = 'image';
folderName = 'Final_Images';
frameRate = 15;

picNameLarge = 'final_map_big.png';

%% Setup info
width = imageWidth/2;
height = imageHeight/2;
x = baseImageUpScale*pointRoute(:,1)+bufferingSize;
y = baseImageUpScale*pointRoute(:,2)+bufferingSize;
image = cell(numFrames,1);


imageDisp = imread(picNameLarge);
imageDisp = imresize(imageDisp,baseImageUpScale);
imageDisp = padarray(imageDisp, [bufferingSize bufferingSize] ,[255], 'both');

%% Loop for creating images

for i = 1:(simulatedFrames):numFrames

%Ensures points are fixed pixel locations
```

```
    x(i) = floor(x(i)); y(i) = floor(y(i));
%Set range of window coordinates
    windowY = (y(i)-height):1:(y(i)+height-1);
    windowX = (x(i)-width):1:(x(i)+width-1);
%Create image based on coordinates
    image{i} = imageDisp(windowY,windowX,:);

%Based on simulated frames number, create a line between
    if simulatedFrames == 1
        %do nothing
    else
        if i < 2 %First set remains stationary to simulate drift
            lineX = x(i)*ones(simulatedFrames,1);
            lineY = y(i)*ones(simulatedFrames,1);
        else %Draw line between points and create simulated points
            clear lineX lineY
            lineX = ceil(linspace(x(i-1),x(i),simulatedFrames));
            lineY = ceil(linspace(y(i-1),y(i),simulatedFrames));
        end
%Create random drift based on value to aid in flight simulation
        for a = 1:simulatedFrames
            x(i+a) = lineX(a)+randi([-drift,drift],1);
            y(i+a) = lineY(a)+randi([-drift,drift],1);

            windowY = (y(i+a)-height):1:(y(i+a)+height-1);
            windowX = (x(i+a)-width):1:(x(i+a)+width-1);

            image{i+a} = imageDisp(windowY,windowX,:);
        end
    end
end

%% Writes all images to the given folder name
if imageWrite
    mkdir(folderName)
    for i = 1:numFrames

imwrite(image{i},fullfile(folderName,sprintf(strcat(imageName,sprintf('%06d.jpg
',i)))));
    end
end

%% Writes a video based on all images captured and given frame rate
if videoWrite
    outputVideo = VideoWriter(fullfile(folderName,strcat(videoName,'.avi')));
    outputVideo.FrameRate = frameRate;
    open(outputVideo);
    for i = 1:numFrames
        writeVideo(outputVideo,image{i});
    end
    close(outputVideo);
end
```

APPENDIX B: *Continuous Mosaicking Code*

```
function mosaicContinuous
This related to the completion of Scott Radford's Master's
% Thesis at Virginia Tech in the Unmanned Systems Lab
%%"Real-time roadway mapping and ground robotic path planning via unmanned
aircraft"
%Scott Radford @ VT USL - scottyr@vt.edu
%Date: 05/2014

clc; clear all; close all;
%Import first images
img1 = imread(fullfile('images','img000001.jpg'));
img2 = imread(fullfile('images','img000002.jpg'));
imageNum = 450; %Total number of images to mosaic
% Performs first mosaic
[img0,~] = imMosaicCont(img2,img1,0,true);
a = 3;
% Loop for continous mosaic (while condition is total number of images)
while a < imageNum;
    % File name is formated for increasing number label to import
    % sequential images
    fileName = sprintf('img%06d.jpg',a);
    img1 = imread(fullfile('images',fileName));

    % Performs actual mosaic - img0 is the combined image - dynamic
    % cropping is included
    [img0, crop] = imMosaicCont(img1,img0,0,true);

    a = a+1;

    imwrite(img0,fullfile('mosaic',sprintf('mosaic3_%05d.jpg',a)),'jpg');
    imappend = appendimages(padarray(img1,[1081-size(img1,1) 1921-
size(img1,2)],[255],'post'),...
                            padarray(img0,[1081-size(img0,1) 1921-
size(img0,2)],[255],'post'));
    imwrite(imappend,fullfile('append',sprintf('append3_%05d.jpg',a)),'jpg');
end

figure,imshow(img0)
save('loopTime3.mat','loopTime');
imwrite(img0,['final_mosaic_' 'img3.' 'jpg'],'jpg')
end

function [ imgout, cropType] = imMosaicCont(img1,img2,adjColor,crop)
%[ imgout, cropType] = imMosaic( img1,img2,adjColor, crop )
%   img1 and img2 can (both) be rgb or gray, double or uint8.
%   If you have more than 2 images to do mosaic, call this function several
%   times.
%   If you set adjColor to 1, imMosaic will try to try to adjust the
%   color(for rgb) or grayscale(for gray image) of img1 linearly, so the 2
%   images can join more naturally.
%   Yan Ke @ THUEE, 20110123, xjed09@gmail.com

%   Adjustments Made by Scott Radford @ VT USL scottyr@vt.edu 2014
%   Updated transform and image warping functions and added dynamic
%   cropping

% use SIFT to find corresponding points
```

```
[matchLoc1 matchLoc2] = siftMatch(img1, img2);

% use RANSAC to find homography matrix
[H corrPtIdx] = findHomography(matchLoc2',matchLoc1');

% tform = maketform('projective',H'); % old command
tform = projective2d(H'); % More accruate tform calculation
%img21 = imtransform(img2,tform); % old command
img21 = imwarp(img2,tform, 'FillValues',[255]); % reproject img2

% Display purposes
%figure,imshow(img1)
%figure(1),imshow(img21); drawnow;

% adjust color or grayscale linearly, using corresponding infomation
[M1 N1 dim] = size(img1);
[M2 N2 ~] = size(img2);
if exist('adjColor','var') && adjColor == 1
    radius = 2;
    x1ctrl = matchLoc1(corrPtIdx,1);
    y1ctrl = matchLoc1(corrPtIdx,2);
    x2ctrl = matchLoc2(corrPtIdx,1);
    y2ctrl = matchLoc2(corrPtIdx,2);
    ctrlLen = length(corrPtIdx);
    s1 = zeros(1,ctrlLen);
    s2 = zeros(1,ctrlLen);
    for color = 1:dim
        for p = 1:ctrlLen
            left = round(max(1,x1ctrl(p)-radius));
            right = round(min(N1,left+radius+1));
            up = round(max(1,y1ctrl(p)-radius));
            down = round(min(M1,up+radius+1));
            s1(p) = sum(sum(img1(up:down,left:right,color)));
        end
        for p = 1:ctrlLen
            left = round(max(1,x2ctrl(p)-radius));
            right = round(min(N2,left+radius+1));
            up = round(max(1,y2ctrl(p)-radius));
            down = round(min(M2,up+radius+1));
            s2(p) = sum(sum(img2(up:down,left:right,color)));
        end
        sc = (radius*2+1)^2*ctrlLen;
        adjcoef = polyfit(s1/sc,s2/sc,1);
        img1(:,:,color) = img1(:,:,color)*adjcoef(1)+adjcoef(2);
    end
end

% Morph the corners of the image
pt = zeros(3,4);
pt(:,1) = H*[1;1;1];
pt(:,2) = H*[N2;1;1];
pt(:,3) = H*[N2;M2;1];
pt(:,4) = H*[1;M2;1];
x2 = pt(1,:)./pt(3,:);
y2 = pt(2,:)./pt(3,:);

% Check size of morphed corners to shift image placement
up = round(min(y2));
Yoffset = 0;
```

```
if up <= 0
    Yoffset = -up+1;
    up = 1;
end

left = round(min(x2));
Xoffset = 0;
if left<=0
    Xoffset = -left+1;
    left = 1;
end
% Perform shifting on images and overlay in one composite image
[M3 N3 ~] = size(img21);
imgout = 255*ones(max([up+M3-1 Yoffset+M1]), max([left+N3-1 Xoffset+N1]),3);
imgout(up:up+M3-1,left:left+N3-1,:) = img21;
    % img1 is above img21
imgout(Yoffset+1:Yoffset+M1,Xoffset+1:Xoffset+N1,:) = img1;
imgout = uint8(imgout);

%Dynamic Cropping to crop old data - tracks newest images added and removes
%old for efficient processing for continuous mosaicking
%Modifcation added by SR
cropType = 'no crop';
    if crop
        [M4 N4 ~] = size(imgout);
        width = 1920;
        height = 1080;
        if N4 > width | M4 > height
            if Xoffset >= (N4-width) & Yoffset >= (M4-height)
                cropType = 'crop xy';
                imgout = imcrop(imgout, [N4-width M4-height width height]);
            elseif Xoffset > (N4-width) & Yoffset < (M4-height)
                cropType = 'crop 2';
                imgout = imcrop(imgout, [N4-width Yoffset width height]);
            elseif Xoffset < (N4-width) & Yoffset > (M4-height)
                cropType = 'crop 3';
                imgout = imcrop(imgout, [Xoffset M4-height width height]);
            else
                cropType = 'crop 4';
            end
        end
    end
end
```

APPENDIX C: *k-Means and Image Morphological Testing*

```
% K-means Algorithm, Image Morphing and Connected Components Testing
% This code goes through the testing and timing of all the above
% algorithms. This related to the completion of Scott Radford's Master's
% Thesis at Virginia Tech in the Unmanned Systems Lab
%%"Real-time roadway mapping and ground robotic path planning via unmanned
aircraft"
%Scott Radford @ VT USL - scottyr@vt.edu
%Date: 04/2014

clc; close all; clear all;
%% Initialize everything including image
tic
k = 4;
frames = 450;
centers = zeros(k,3,frames);
i = 1;
image = imread(fullfile('images','img000045.jpg'));

%% Reshape pixels and run k-means (times it as well)
[M, N, d] = size(image);
points = double(reshape(image,M*N,d));
tic;
[idx, centers] = kmeans(points,k);
kmeansTime = toc

%% Reshape and plot kmeans image beside normal imag
kmeansImage = uint8(reshape(idx, M, N)); %255/4 is for scaling purposes
kmeansImage(:,:,2) = uint8(reshape(idx, M, N));
kmeansImage(:,:,3) = uint8(reshape(idx, M, N));
figure(1);
imshow([image 255/4*kmeansImage],[]); drawnow;

%% BWMorphing of images
%Gets a known road location
title('Click Road Location (left image)');
[x,y] = ginput(1);
[x,y] = deal(ceil(x),ceil(y));
[M2,N2,d2] = size(kmeansImage);
%Thresholding for known road location pixel value
kmeansBW = kmeansImage(:,:,1);
bwImage = kmeansBW == kmeansBW(y,x);
bwImage = reshape(bwImage,M2,N2);

%Creation of different bwmorphed images
erode = bwmorph(bwImage,'erode',3);
dilate = bwmorph(bwImage,'dilate',3);
open = bwmorph(bwImage,'open',3);
combination = bwmorph(bwImage,'clean');
combination = bwmorph(combination,'erode',2);
combination = bwmorph(combination,'spur');
combination = bwmorph(combination,'open');
combination = bwmorph(combination,'thicken',2);
combination = bwmorph(combination,'dilate',4);

%plots examples - multipliers are only for scaling purposes
figure(3);
```

```
imshow([k/255*rgb2gray(image) 3*bwImage 3*erode; 3*dilate 3*open
3*combination],[]);

%% Connected components
%cast the logical as a unit8 for display purposes mostly
kmeansMorphedRoad = uint8(combination);

%find road locations based on connected components and the
%known road location's value from bwlabel's output
tic
connectedRoad = bwlabel(kmeansMorphedRoad,8);
blobValue = connectedRoad(y,x);
connectedRoad = uint8(connectedRoad == blobValue);
[yRoad,xRoad] = find(connectedRoad==1);
bwlabelTime = toc

%Displays road locations over the image used for both bwmorphed (left) and
%connected components (right)
disp1 =
imlincomb(1,image,1,cat(3,255*kmeansMorphedRoad,zeros(M2,N2),zeros(M2,N2)));
disp2 =
imlincomb(1,image,1,cat(3,255*connectedRoad,zeros(M2,N2),zeros(M2,N2)));
figure(4); clf;
imshow([disp1 disp2])

toc
```

APPENDIX D: *Modified RRT Path Planning Algorithm*

```
function path = RRTpath_planningSR
% This related to the completion of Scott Radford's Master's
% Thesis at Virginia Tech in the Unmanned Systems Lab
%%"Real-time roadway mapping and ground robotic path planning via unmanned
aircraft"
%Scott Radford @ VT USL - scottyr@vt.edu
%Date: 07/2014

% Import road locations and map
roadFilename = 'roadLocations.mat';

roadLocations = load(roadFilename);

% Use if map is a jpeg instead of list of roadLocations
% map = im2bw(imresize(imread('incompleteMap3.jpg'),0.25));
% [roadLocations(:,2),roadLocations(:,1)] = find(map);

%% Check road location size
if size(roadLocations,2) ~= 2 | size(roadLocations,1) < 1
    if size(roadLocations,1) == 2
       roadLocations = roadlocations';
    else
        size(roadLocations);
        error('Road Location Matrix Size Incorrect')
    end
end
fprintf('Road Locations Checked');

%% Create and display road map
[N,~] = max(roadLocations(:,1))
[M,~] = max(roadLocations(:,2))
roadMap = zeros(M,N);
size(roadMap)
linearInd = sub2ind(size(roadMap), roadLocations(:,2), roadLocations(:,1));
roadMap(linearInd) = 1;
imshow(roadMap,[]);
drawnow;
%% Start and Goal Locations
start = [1 1];
goal = [N M];

%% Check Connection and Adjust Goal if necessary
valid = 0;

    [valid, roadMap] = checkConnect(roadMap, start, goal);
    figure(3); imshow(roadMap,[]); drawnow; figure(2)
    clear roadLocations
    [roadLocations(:,2),roadLocations(:,1)] = find(roadMap);

    if ~valid
        [goal, origGoal] = adjustGoal(roadLocations, start, goal);
    end


% Plot start and goal
plot(start(1),start(2), 'og', 'MarkerSize', 5);
```

```
plot(goal(1),goal(2), 'or', 'MarkerSize', 5);

%% Parameters
param.resolution = 1;     %relative pixel resolution
param.distThresh = 100;    %distance threshold to goal location
param.maxIter = 500;     %maximum number of RRT iterations

%% Path Planning
fprintf('\nStarting RRT')
[path, tree, found] = RRTplanning(start, goal, param, roadLocations);

%% Plot RRT full tree
hold on

    for a = 2:length(tree)
        x = [tree(a,1); tree(tree(a,3),1)];
        y = [tree(a,2); tree(tree(a,3),2)];
        plot(x,y,'-m','LineWidth',2);
    end

%% Plot final path generated
hold on
plot(path(:,1),path(:,2),'-b', 'LineWidth', 3);
path
%% Smooth Path
iterations = 50;
smoothedPath = SmoothPath(path,roadLocations,tree,iterations);

hold on
plot(smoothedPath(:,1),smoothedPath(:,2),'-g', 'LineWidth', 3);
smoothedPath

end

function [path, tree, found] = RRTplanning(start, goal, param, roadLocations)
    %Initialize
    path = [];
    tree = [];
    tree = addNode(tree,start,0);
    found = false;
    %Number of road points
    roadSize = length(roadLocations);

    % Main loop exits when goal is found of interations are maxed
    i = 1;
    while i <= param.maxIter
        fprintf('\nIteration %01d',i);
        point = roadLocations(randi(roadSize,1),:);

        treeSize = size(tree,1)
        dist = sqrt((tree(:,1)-point(1)).^2 + (tree(:,2)-point(2)).^2);
        [minDist, prev] = min(dist);

        if minDist > 800
            fprintf('\nNew point too far');
            continue
        end
```

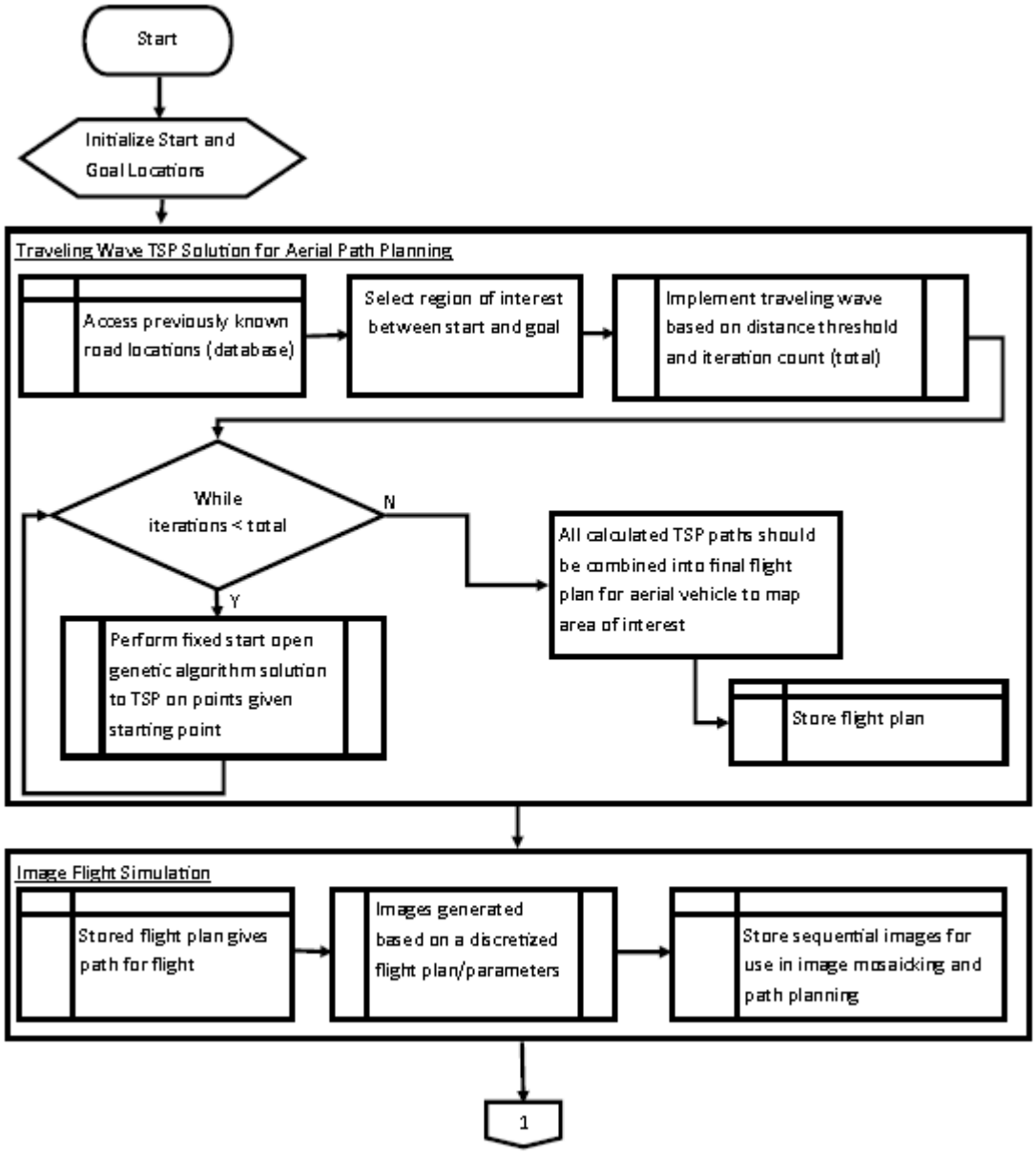
```
        intersect = intersectionTest(prev,point,tree, roadLocations);
        % Iterate loop if point intersects obstacle
        if intersect == 1;
            %i = i+1;
            continue
        end
        % Create node on tree
        [tree] = addNode(tree, point, prev);

        % Check for distance to goal and intersection
        distGoal = sqrt((point(1)-goal(1))^2 + (point(2)-goal(2))^2);
        if distGoal < param.distThresh
            dist = sqrt((tree(:,1)-goal(1)).^2 + (tree(:,2)-goal(2)).^2);
            [minDist, prev] = min(dist);
        intersect = intersectionTest(prev,goal,tree, roadLocations);
            if intersect == 1
                i = i + 1;
                continue
            else %Create final node to goal and generate full path
                [tree] = addNode(tree, goal, prev);
                a = size(tree,1);
                path = tree(a,1:2);
                while 1 % Generates path from initial to final point
                     a = tree(a,3);
                     if a == 0
                         return
                     end
                     path = [tree(a,1:2); path];
                end
                i = parameter.maxIter; % exits final loop
                found = true;
            end
        end
         fprintf('\nNodes:   %d, Distance: %.1f, Iterations:
%d/1000',length(tree),distGoal,i)
        i = i + 1;
    end

    if ~found;
        fprintf('\nGoal not found, path to closest point shown')

        treeSize = size(tree,1);
        dist = sqrt((tree(:,1)-goal(1)).^2 + (tree(:,2)-goal(2)).^2);
        [minDist, closest] = min(dist);

        path = tree(closest,1:2);
        while 1 % Generates path from initial to final point
             closest = tree(closest,3);
             if closest == 0
                 return
             end
             path = [tree(closest,1:2); path];
        end
    end
end

function [tree] = addNode(tree, point, prev)
    % point = New point to add
    % prev = Closest node in tree (index)
```

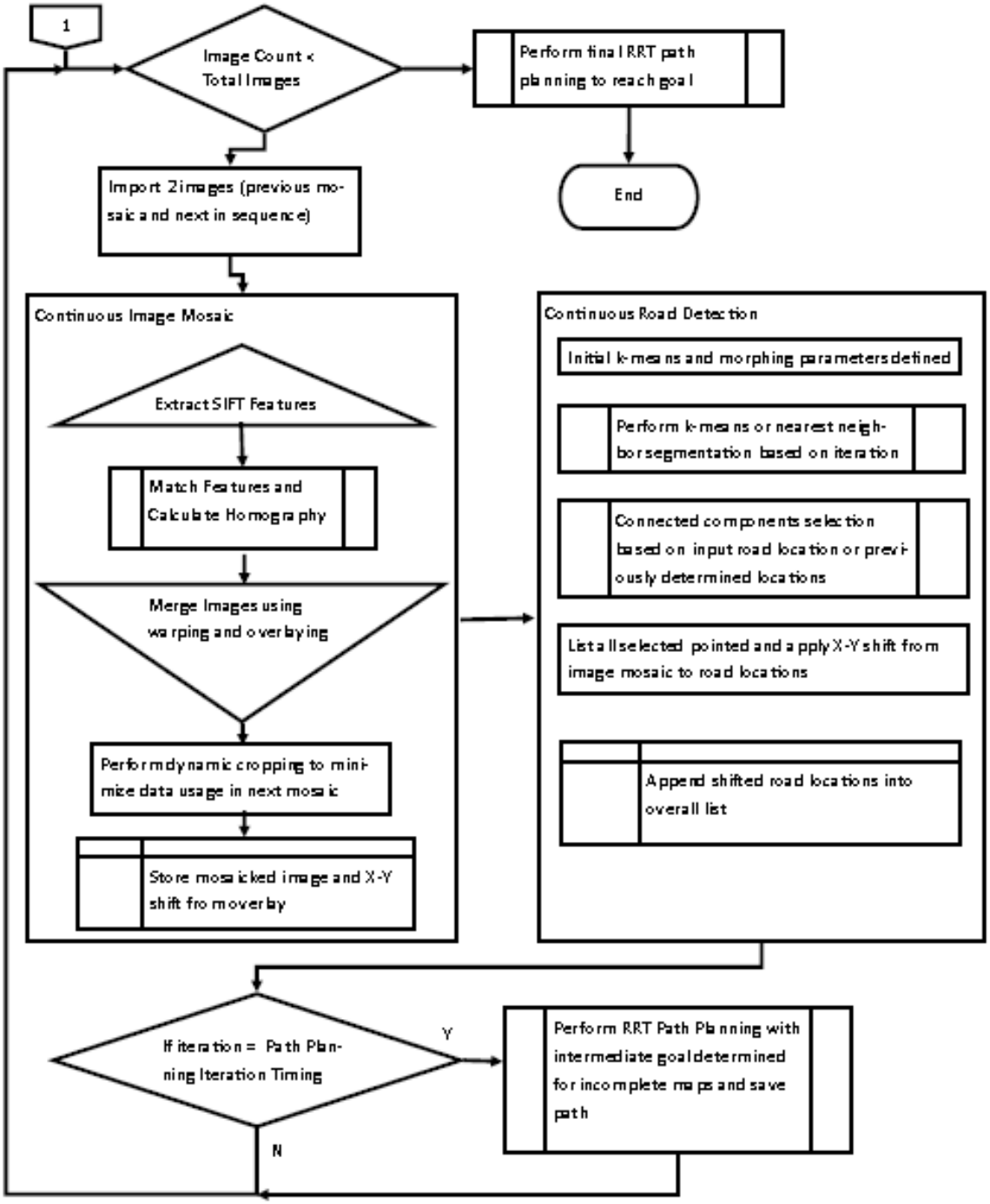
```
    tree(end+1,:) = [point prev]; % Mx3 matrix of points with linked node
end

function [intersection] = intersectionTest(prev,point,tree, roadLocations)
    % Test intersection of new point to previous point (prev is row index
    % number of the point in the tree matrix)
    x = round(linspace(point(1),tree(prev,1),100))';
    y = round(linspace(point(2),tree(prev,2),100))';
    line = [x y];
    line = unique(line,'rows');

    check = ismember(line, roadLocations,'rows');

    intersection = sum(check == 0);
    intersection = intersection > 0;
    if intersection
        fprintf('\nIntersection Detected Selecting New Node');
    end
end

function [valid, roadMap] = checkConnect(roadMap, start, goal)
    % Checks Connectivity of start to goal
    valid = 0;

    connectedRoad = bwlabel(roadMap);

    if connectedRoad(start(2),start(1)) == connectedRoad(goal(2),goal(1));
        valid = 1;
    else
        roadMap = connectedRoad == connectedRoad(start(2),start(1));
    end

end

function [goal, origGoal] = adjustGoal(roadLocations, start, goal)
    % Changes the goal to the closest goal on the map that is on an edge.
    % Consider adjusting criteria to include an area of known mapped
    % regions
    origGoal = goal;

    [N,~] = max(roadLocations(:,1))
    [M,~] = max(roadLocations(:,2))

    edges = roadLocations(roadLocations(:,1) == N,:);
    edges = [edges; roadLocations(roadLocations(:,2) == M, :)];

     dist = sqrt((edges(:,1)-goal(1)).^2 + (edges(:,2)-goal(2)).^2);
     [~, ind] = min(dist);

     goal = edges(ind,:);
end

function [smoothedPath] = SmoothPath(path,roadLocations, iterations)
    %Smooths/shortens path. Checks given road locations for intersection or
    %obstacles. 'iterations' is total number of attempts not actual
    %reduction/addition of points
    smoothedPath = path;
    [num,dim] = size(smoothedPath);
```

```
    if dim ~= 2
        fprintf('Path Dimension Incorrect');
        return
    end
    if num == 2
        fprintf('Path is only two points. No smoothing capable');
        return
    end

    lines = zeros(num,1);

    for i = 2:num
        lines(i) = norm(smoothedPath(i,:)-smoothedPath(i-1,:)) + lines(i-1);
    end

    line_init = ceil(lines(num));
    i = 1;
    while i <= iterations
        s1 = randi(line_init,1);
        s2 = randi(line_init,1);
        if s2 < s1
            temps = s1; s1 = s2; s2 = temps;
        end
        for k = 2:num
            if s1 < lines(k)
                x = k - 1;
                break;
            end
        end
        for k = (x+1):num
            if s2 < lines(k)
                y = k - 1;
                break;
            end
        end
        if (y <= x)
            i = i + 1;
            continue;
        end
        t1 = (s1 - lines(x))/(lines(x+1)-lines(x));
        gamma1 = round((1 - t1)*smoothedPath(x,:)+t1*smoothedPath(x+1,:));
        t2 = (s2 - lines(y))/(lines(y+1)-lines(y));
        gamma2 = round((1 - t2)*smoothedPath(y+1,:)+t2*smoothedPath(y+1,:));
        tree = [smoothedPath(x,:) 1; smoothedPath(y,:) 1];
       col = intersectionTest(1,smoothedPath(x+1,:),tree, roadLocations) + ...
            intersectionTest(1,smoothedPath(y+1,:),tree, roadLocations);

        if col >= 1
            i = i + 1;
            continue;
        end
        smoothedPath = [smoothedPath(1:x,:); gamma1; gamma2;
smoothedPath(y+1:num,:)];
            lines = zeros(num,1);

        [num,dim] = size(smoothedPath);

        for i = 2:num
```

```
            lines(i) = norm(smoothedPath(i,:)-smoothedPath(i-1,:)) + lines(i-
1);
        end

        i = i + 1;
    end
end
```

APPENDIX E: *System Flow Chart*




1
Image Count < Total Images
Perform final RRT path planning to reach goal
End
Import 2 images (previous mosaic and next in sequence)
Continuous Image Mosaic
Extract SIFT Features
Match Features and Calculate Homography
Merge Images using warping and overlaying
Perform dynamic cropping to minimize data usage in next mosaic
Store mosaicked image and X-Y shift from overlay
Continuous Road Detection
Initial k-means and morphing parameters defined
Perform k-means or nearest neighbor segmentation based on iteration
Connected components selection based on input road location or previously determined locations
List all selected pointed and apply X-Y shift from image mosaic to road locations
Append shifted road locations into overall list
If iteration = Path Planning Iteration Timing
Y
N
Perform RRT Path Planning with intermediate goal determined for incomplete maps and save path