4-18-2022

A Depth-Based Computer Vision Approach to Unmanned Aircraft System Landing with Optimal Positioning

Nicholas Quattrociocchi
nq9009@rit.edu

ROCHESTER INSTITUTE OF TECHNOLOGY

Master Thesis

Mechanical Engineering Department

Rochester Institute of Technology

A Depth-Based Computer Vision Approach to Unmanned Aircraft System Landing with Optimal Positioning

Nicholas Quattrociocchi

Advisor: Dr. Agamemnon Crassidis

Department Representative: Dr. Schrlau

Defense Date: 4/18/2022

Thesis Signature Page

ROCHESTER INSTITUTE OF TECHNOLOGY

Master of Science in Mechanical Engineering

Program (please check one) MS____ BS/MS__X__ Today's date________________

Student Name: Nicholas Quattrociocchi

Student Signature:

Student ____ Date_________

Committee Signatures:

Faculty Advisor ____ Date_________

Committee Member ____ Date_________

Committee Member ____ Date_________

Committee Member ____ Date_________

Committee Member ____ Date_________

Department Representative ______________________________________ Date_________

Abstract

High traffic congestion in cities can lead to difficulties in delivering appropriate aid to people in need of emergency services. Developing an autonomous aerial medical evacuation system with the required size to facilitate the need can allow for the mitigation of the constraint. The aerial system must be capable of vertical takeoff and landing to reach highly conjected areas and areas where traditional aircraft cannot access. In general, the most challenging limitation within any proposed solution is the landing sequence. There have been several techniques developed over the years to land aircraft autonomously; however, very little attention has been scoped to operate strictly within highly congested urban-type environments. The goal of this research is to develop a possible solution to achieve autonomous landing based on computer vision-capture systems. For example, by utilizing modern computer vision approaches involving depth estimation through binocular stereo computer vision, a depth map can be developed. If the vision system is mounted to the bottom of an autonomous aerial system, it can represent the area below the aircraft and determine a possible landing zone. In this work, neural networks are used to isolate the ground via the computer vision height map. Then out of the entire visible ground area, a potential landing position can be estimated. An optimization routine is then developed to identify the most optimal landing position within the visible area. The optimization routine identifies the largest identifiable open area near the desired landing location. Web cameras were utilized and processed on a desktop to form a basis for the computer vision system. The algorithms were tested and verified using a simulation effort proving the feasibility of the approach. In addition, the system was tested on a scaled down city scene and was able to determine an optimal landing zone.

TABLE OF CONTENTS

List of Figures 3

Nomenclature 4

1.0 Introduction 6
- 1.1 Literature Review 6
- 1.2 Overview of Process 9

2.0 Theory Development 15
- 2.1 Histogram & Feedforward Neural Net 15
- 2.2 Optimization Function 16
- 2.3 Cameras, Monocular Calibration, and Stereo Calibration 17
- 2.4 Image and Depth Map Processing 19
- 2.5 Using the Depth Map to Find a Landing Location 22

3.0 Simulation Results 23

4.0 Experimental Results 25
- 4.1 Experimental Setup 25
- 4.2 Experiment Execution Results 26
- 4.3 Performance Metrics 29

5.0 Conclusion 33
- 5.1 Requirements for Effective Operation [In Current State] 33
- 5.2 Recommendations for Future Work 34
- 5.3 Current State of Mechanical Designs for Aerial system 35

6.0 Acknowledgments 39

7.0 References 40

A. Appendix 43

LIST OF FIGURES

Figure 1	Random Scene, mean of depth: -35.4618 feet, standard deviation of the points around ground level: 2.6809 feet.	10
Figure 2	Histogram of z values from random scene.	10
Figure 3	Thresholded Height map.	11
Figure 4	Optimal landing position marked by red circle (13.7158, -0.0350), with square length of 14.0836.	11
Figure 5	Left: Left camera image. Middle: Disparity Image. Right: Right camera image.	12
Figure 6	Left: “Bird’s eye view” of the depth map. Right: Angled view of the depth map.	13
Figure 7	The histogram of the real height data.	13
Figure 8	“Thresholded” image depth map.	14
Figure 9	Location of optimal landing location.	14

Figure 10	Left column: Randomly generated Scenes. Right column: Randomly Generated Scene's respective Histogram plot.	15
Figure 11	OpenCV 9x6 Checkerboard Calibration Artifact.	17
Figure 12	Left: Left camera image. Right: Right camera image.	18
Figure 13	Checkerboard Identification.	19
Figure 14	Left: Left camera image. Middle: Unprocessed Depth map. Right: Right camera image.	19
Figure 15	MATLAB's Parula scale. Left corresponds to low values, right corresponds to high values.	20
Figure 16	Left: Left camera image. Middle: Processed Depth map. Right: Right camera image.19	20
Figure 17	Salt and pepper noise is removed from an image, image supplied from OpenCV.	22
Figure 18	Depth map converted to XYZ Height map representation.	22
Figure 19	Loss function output during neural net training.	23
Figure 20	Average difference per depth bin.	24
Figure 21	Experimental Setup, the vision system is attached to a robust system of duct tape.	25
Figure 22	Up close view of the web camera system and mounting fixtures.	26
Figure 23	Full view of the vision system.	26
Figure 24	Arrangement 1. Top 3: (Left, Middle, Right) Left Image, Disparity Image, Right Image. Bottom 3: Depth Map, Histogram, Thresholded Depth Map with marked landing point.	27
Figure 25	Optimal positions for a box within a rectangle.	27
Figure 26	Arrangement 2. Top 3: (Left, Middle, Right) Left Image, Disparity Image, Right Image. Bottom 3: Depth Map, Histogram, Thresholded Depth Map with marked landing point.	28
Figure 27	Arrangement 3. Top 3: (Left, Middle, Right) Left Image, Disparity Image, Right Image. Bottom 3: Depth Map, Histogram, Thresholded Depth Map with marked landing point.	28
Figure 28	Arrangement 4. Top 3: (Left, Middle, Right) Left Image, Disparity Image, Right Image. Bottom 3: Depth Map, Histogram, Thresholded Depth Map with marked landing point.	29
Figure 29	Additional starting point, but the same map as Figure 25, yields roughly same point.	29
Figure 30	Average difference for predicted depth vs real depth of real scene data, rescaled between [-10, 0].	30
Figure 31	Average difference for predicted depth vs real depth of real scene data, rescaled between [-15, 0] and shifted down with the X-axis values in the graph.	31
Figure 32	Average difference for predicted depth vs real depth of real scene data, rescaled between [-5, 0] and shifted down with the X-axis values in the graph.	31
Figure 33	Average difference for predicted depth vs real depth of real scene data, rescaled between [-30, 0] and shifted down with the X-axis values in the graph.	32

Figure 34	Subset of images highlighting differences in intensity affecting clarity in depth maps.	34
Figure 35	Angled view of the Aerial System.	35
Figure 36	Side view of the Aerial System.	36
Figure 37	Front view of the Aerial System.	36
Figure 38	Top view of the Aerial System.	37
Figure 39	Second Angled View of the Aerial System; see propeller structure inside cones.	38

NOMENCLATURE

Abbreviations:

FCN = Fully Convolutional Neural Network.

FFN = Feed Forward Network - A neural net wherein the information or data only goes in one direction, this would mimic a non-recursive function.

FOD = Foreign object debris– Objects that inflict onto propellers/engine via debris entering the engine.

Background Concepts:

Binocular Camera System = A stereo computer vision system incorporating strictly two cameras.

Feasible Region = In optimization problems, the feasible region is a subspace of the function to be optimized that does not invalidate the boundary conditions placed onto the problem.

Multimodal Distribution = A histogram of data resulting in multiple distinguished normal distributions.

Sparse Reward Problem = In the context of artificial intelligence and machine learning, the sparse reward problem refers to a high infrequency of positive or matching feedback for the algorithm. In the context of optimization, this would be analogous to having a non-linear and or discontinuous function to minimize, and only decreasing the function’s evaluated value once a boundary condition has been validated. Until the boundary condition has been satisfied, the error/function’s evaluation would remain constant and therefore result in early routine stopping, as the overall routine would think it is at the lowest point of the function.

1.0 INTRODUCTION

Autonomous landing of Unmanned Aircraft Systems (UASs) has received much attention recently; however, there has not been much development in facilitating methods of autonomous landing in highly congested urban environments. Operating within buildings, maneuvering around cars, avoiding debris, mitigating foreign object damage, etc. are constraints for the design of a UAS operating within large urban environments. The overall objective of the research was to develop an autonomous aerial vehicle expediting patient acquisition while avoiding traffic and other obstructions thus increasing the probability of saving lives. The type of autonomous aerial vehicle configuration chosen for this application is a large quadcopter-type autonomous aerial vehicle. In particular, the research addressed the autonomous landing validation mechanism using vision-based algorithms, which have been accomplished by using a stereo computer vision system.

For the grander scope of the aerial system, there will be a set of geolocations (e.g., intersections, wide roads, etc.) deemed as valid initial areas for landing. From these locations, the vision system will deem if and where the aerial system can land. The optimization routine will take over and then the rest of this developed work will begin. The pre-determined landing locations will be generated through Google Maps or some other similar media, and the specific location will be selected via proximity to the location of the emergency. For the scope of this work, these locations are assumed to be exactly known and the aerial system and are within a "reasonable" distance (and can be explicitly determined through testing of the fully-fledged vision system) from the target location.

Once the location is selected, and a position is available to land in, the aerial system will receive commands to move towards the location (within a threshold) via a control system and then descend. While the location is being evaluated, the aerial system will be in place hovering over the location. The control system is outside the scope of this research.

The control system can incorporate a feedback system via the image inputs. Note: (again) development of the control system algorithm is outside the scope of the proposed work. The system will receive the set points of the translational offsets between the current position and the landing location. The feedback signal can be the distance between the center of the current image from the computer vision system and the center of the current image convolved with a filter of the open area deemed as the best point to land. The desired tracking position for this work will be the center of the QR/AprilTags code, and then the distance between the center of the current image and the code.

1.1 LITERATURE REVIEW

In this section, a review of implementations of computer-based vision algorithms for automated landing procedures is outlined. In section 3.1, stereo/binocular vision will be highlighted and how it pertains to the suggested implementation of the proposed vision-based system and other systems. Section 3.2 reviews hardware systems that prior works implemented for vision-based systems.

Section 3.3 provides a review of prior computer vision systems utilizing a form of an artifact to perform the developed tasks. Section 3.4 provides an overview of prior systems using machine learning algorithms to process the information gleaned from the computer vision algorithms. Finally, Section 3.5 provides additional potential methods for object detection to be evaluated deeply, post thesis proposal defense.

1.1.1 Computer Vision Systems

X. Sun et al. [1] highlights the purpose of using a binocular stereo vision system. The authors identify how the disparity of two images can be calculated and then used to determine the depth of each pixel. Brown et al. [2] highlight that solutions implementing stereo vision are not exact and subject to occlusion and error due to problem being ill-posed. Brown et al. also identifies that the error in depth estimate decreases as the distance between cameras increases. Cheng, Wang, and Yang explain the problem worsens when using monocular systems; however, is heavily assuaged by incorporating machine learning to assist in the depth estimation [6]. The prior work addresses why the use of a binocular camera vision is the best method of choice for depth estimation is a binocular vision system.

1.1.2 Hardware Recommendations for Image Processing and Functionality

The work performed by Blachut et al. [7], develops a potential computing platform that could facilitate the operation of the proposed computer-based vision system. Blachut et al. validate a heterogeneous Zynq SoC device as the computing platform to allow real-time calculation speed while reducing the energy cost on the battery to allow for longer operation. Currently, the proposed system and planned algorithms should be able to be implemented on a raspberry pi microcomputer with relative ease. For full system implementation, the computing platform proposed by the authors in Ref. [7] could be a valuable option to be considered as a reduction in the energy cost on the system can be realized. The authors' autonomous landing mechanism discussed utilized a custom landing fiducial/artifact, a standard AprilTags code will be considered in this work due to their resiliency against false positive detections in natural scenes.

For full implantation of an autonomous landing control system, Supriyono and Akhara [8] provide an overview of some key design aspects for the control system. The work included utilizing a GPS to maneuver the UAS to an appropriate range for the vision system to operate appropriately, then transitioning to vision system to reach the artifact. Gonzales [9] utilized a raspberry pi with PiCameras to implement the vision system used in a UAS vision system application with a static landing pad. However, the implementation adds weight to the aforementioned hardware system. The main difference between these implementations and the one proposed here is the constraints under which the UAS is envisioned. The constraints references [8, 9] were operating under have no boundary conditions (such as cars and or debris) impeding landing as the proposed implementation assumes, and therefore describes a similar method for a different problem.

1.1.3 Autonomous Landing Algorithms involving Artifacts

Another implementation of an automated landing sequence was performed in the work shown by Marcon, Janousek, and Kadlec [10]. In this work, an UAS was designed to land using a camera and infra-red (IR) emitter and receiver. The position of the UAS was monitored by a real-time kinetic positioning system, and the UAS tracked the IR emitter and utilized the camera for object detection to land on a general landing pad. By utilizing multiple methods to identify the landing pad, the UAS was able to obtain an accurate positioning on the landing pad. The key point of the authors' approach was the utilization of designated landing pads with IR emitters driving the fine positioning of the system. However, the work leaves a gap regarding landing approaches under constraints onto an AprilTags code or similar implementation.

Static targets for automated landing mechanisms are not the only methods developed in previous works. In the work by Li, et al. [11], the authors utilized AprilTags codes affixed on top of a ground vehicle while the UAS performs image tracking of the code along with landing on the vehicle. The implementation validates the use of AprilTags codes as a viable solution to be used as a tracking target for landing. However, the implementation does not contain constraints of landing in urban areas to be considered here.

The work performed by Lin, Garratt, and Lambert [12] provides a unique procedure for landing on cluttered shipboard environments. By targeting the characteristic landing pad "H", 4 degrees of motion freedom can be estimated via using their procedure. In the work done by Patruno, Nitti, Petitti, et al. [13], a similar approach was studied by using the characteristic "H" landing pad; however, instead of cluttering and covering the area, the mechanism to confound their proposed algorithm was overlaying the "H" target on top of a city carpet play mat, or different testing orientations. In this work, the "H" target is not assumed addressing the gaps described earlier. The main gap in the authors' work (to be considered in this work) is that if the UAS cannot land on the "H" target pad, it must land near it. Thus, a need is identified to allow for a heavily cluttered environment to be the goal, but targets near the goal must also be considered as viable solutions for the problem.

1.1.4 Autonomous Landing Algorithms without Artifacts

Fraczek, Mora, and Kryjak [14] developed a solution to evaluate a large landscape area, utilizing shadow detection, support vector machines, and a depth map to determine if there was a viable landing location based on obtained images. The goal of the implementation was a classification and viable position algorithm (which was based on varying locations including soil, trees, grass, asphalt, water, etc.) to find a location to land in if one is viable. The authors' work overlaps with a few key points of this proposed research; however, the main implementation of the previous work was consideration solely in rural areas. Therefore, the previous work required a much more robust computer vision implementation to validate the locations in question. The identification of fauna, houses, and other obstacles is not important to the implementation proposed here.

Identification of positions of objects will be crucial in the development of the proposed mechanism. Safadinho et al [15] utilized a method of using a Deep Neural Network (DNN) to identify whether or not a “blob” in the image is a person. The authors’ mechanism is quite useful in the overall design of the proposed approach as it developed a method of delivering packages to a person and or their residence. The mechanism, developed by the authors, uses a GPS to maneuver an UAS in the proximity of a desired location and then uses a computer vision algorithm to develop coordinate estimates for the landing position. A vital component for worked previously performed is the lack of an artifact used in the landing sequence. The UAS does however require the presence of a human to deliver the package, which is a major gap not assumed for this proposed work. Therefore, the previous work implementation can only land under these given and extremely constrained conditions for which are not assumed here.

Several works have implemented a crowd detection algorithm and/or counting of persons using a Fully Convolutional Network (FCN) to act as a method to categorize whether an image contains a crowd of people [16, 17]. The FCN was trained using the VisDrone dataset [18] (which is a large dataset of images obtained from UAS camera systems) to be used by future UAS FCN and or Convolutional Neural Nets (CNN). The methodology could potentially follow-on research outside the scope of this work; however, the goal of this research is to identify a region large enough for the UAS land. The granularity allotted by the aforementioned papers is more than what is required to accomplish the given problem. The methodology utilized in these papers was taken further by Castellano et al. [19] to not only determine if a crowd of people existed, but to count them as well. Their method utilized a CNN for the categorization task and another CNN for a regression task. The novelty of identifying if there are humans in the area could potentially be useful; however, for the purpose of this research, if there are any humans in the area, they be avoided, therefore, counting them does not add value.

1.1.5 Object Detection

Since height bucketing was successful, works [3, 4, 5], where object detection has been implemented on a UAS for the purpose of rooftop mapping, architectural mapping, etc. have not been implemented.

1.2 OVERVIEW OF PROCESS

A series of programs were developed to simulate a random street scene, as viewed by a binocular stereo vision system. The Random Scenes (RS) were created to train a neural network to detect the distance from the ground to the vision system, along with an estimate of the standard deviation of the error of the ground prediction. 50,000 random scenes were created and stored. Figure 1 displays an example of a random scene that was created.

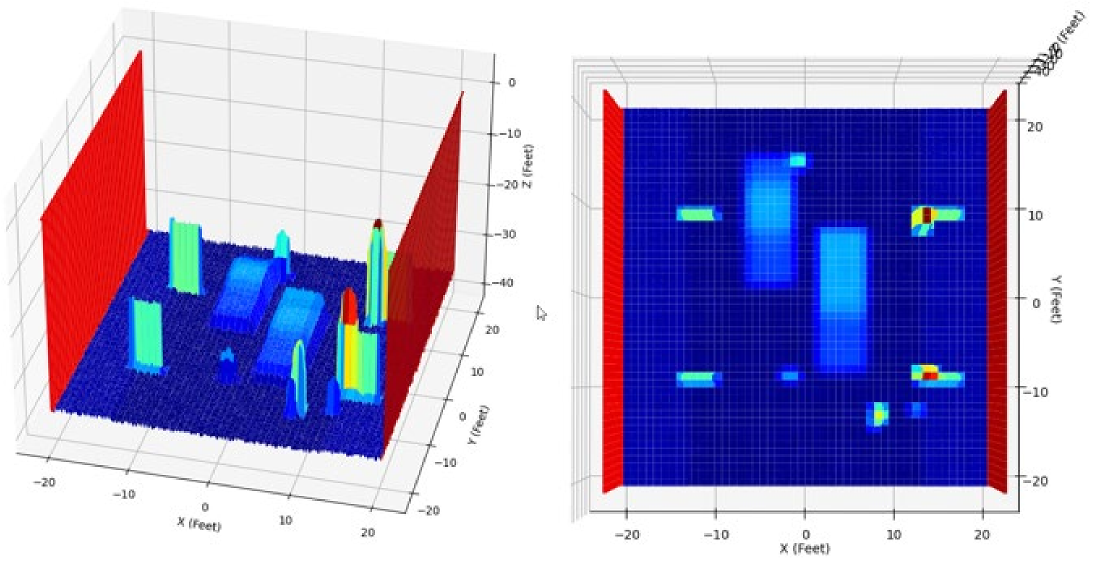

Figure 1: Random Scene, mean of depth: -35.4618 feet, standard deviation of the points around ground level: 2.6809 feet

A histogram of the height (*Z* component of points in Figure 1) was created, shown in Figure 2.

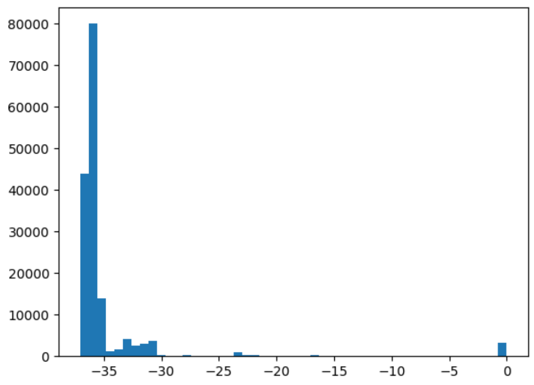

Figure 2: Histogram of z values from random scene

The histogram is fed into a neural net algorithm, and the mean and standard deviation of the ground's height (also referred to as *Z* height or *Z* depth) is output. The NN is able to distinguish the ground height using the bins and counts. Determining the height of the ground is relatively easy for humans by using the histogram: there is roughly a normal distribution around the -36 feet mark and is roughly the average height of the ground. The NN algorithm determined the depth to be -36.847 feet, and the standard deviation to be 1.5514 feet. The NN algorithm is accurate, but not exact.

The values from the neural net are used to determine bounds to threshold the height map of the RS. The "thresholded" random scene is shown below in Figure 3.

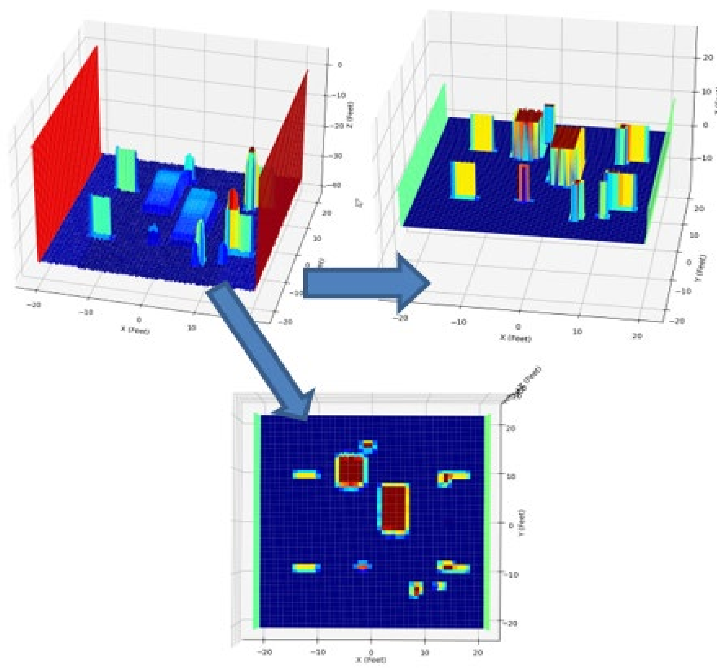

Figure 3: Thresholded Height map

Once the RS is thresholded, an optimization function identifies the largest square bounding box that can be placed within the image, without the box existing on the non-zero points of the thresholded RS. For this scene the optimization function determined the best location to be at coordinates (13.7158, -0.0350). Figure 4 shows the optimal location of the autonomous aerial vehicle based on the applied algorithms. The length of the square that fit was 14.0836 feet.

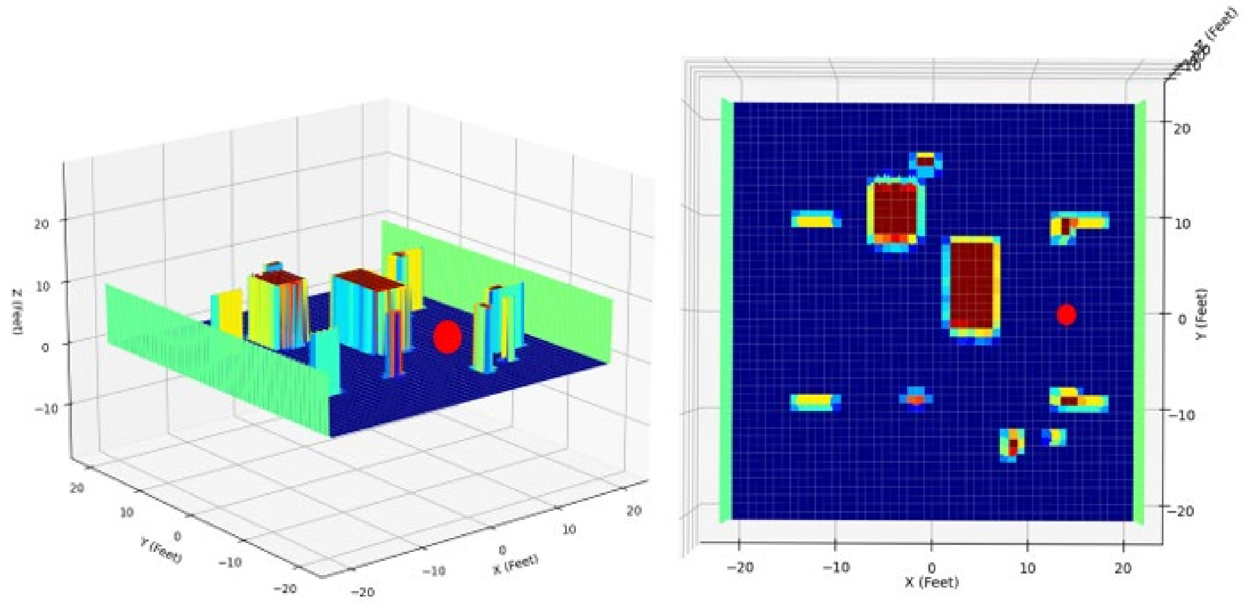

Figure 4: Optimal landing position marked by red circle (13.7158, -0.0350), with square length of 14.0836.

The optimization routine initializes 25 starting points and will allow for more to be selected once implemented on the autonomous aerial vehicle. Initial starting points are arranged in a grid of 5x5 points and are implemented to better guarantee the global optimum rather than just a local optimum.

Hardware has been tested on a smaller-scale set of images. Figure 5 shows the left camera image, disparity map, and the right image respectively from left to right.

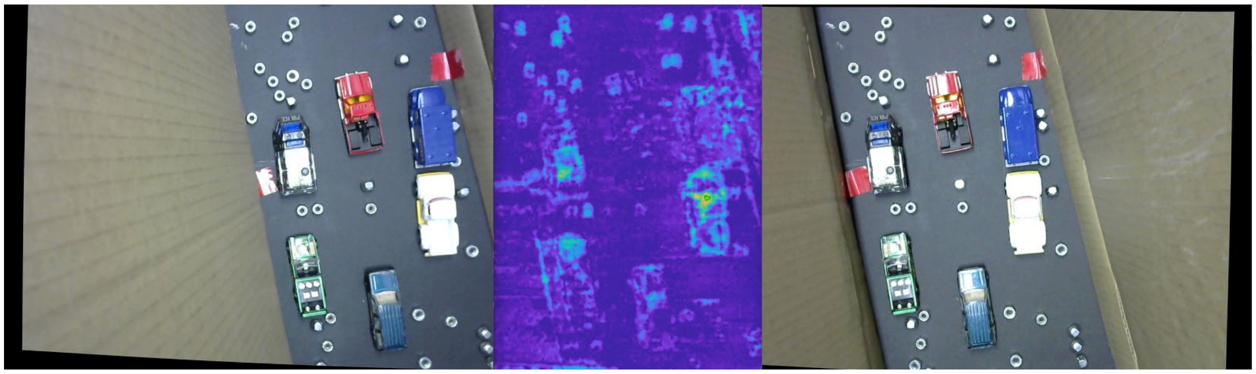

Figure 5: Left: Left camera image. Middle: Disparity Image. Right: Right camera image.

Due to the low quality of the cameras, a vigorous thresholding/penalization function was performed on the image. The function was proportional to the derivative of each pixel over time (10 images over a few seconds). Intensity of the gray scale versions of the images were scaled according to the inverted values of the derivative map. The scaling greatly lowered the intensity of pixels that changed drastically and increased the intensity of the pixels that did not change much. The scaling operation was performed because the cameras were not high-quality causing features to appear and disappear on the initial depth map. The actual features such as the cars and the nuts were static, and therefore with the inverse derivative map applied, it omitted the noise and increased static values. The image was then rescaled and shifted down to resemble the training data. The resulting map is shown in Figure 6 and was also resampled from size (480, 352) to (400, 400).

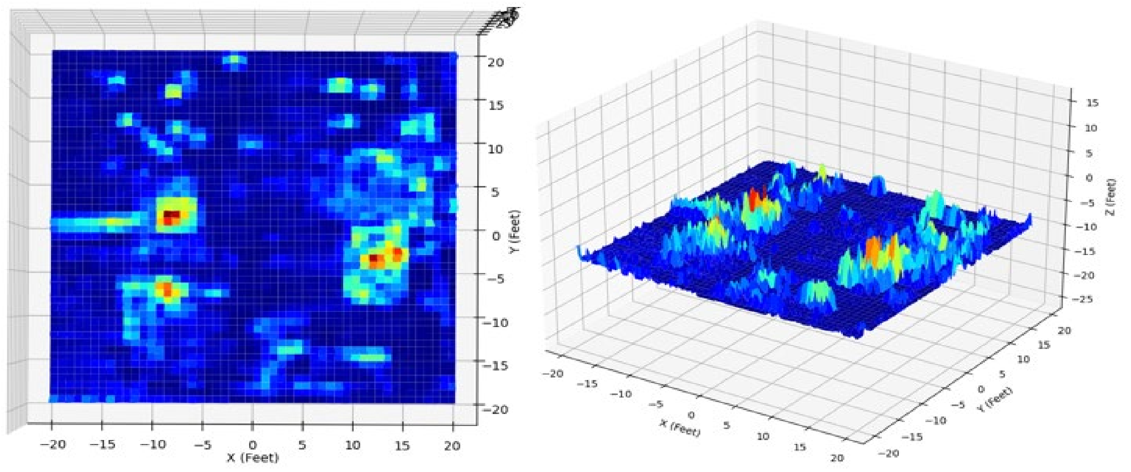

Figure 6: Left: "Bird's eye view" of the depth map. Right: Angled view of the depth map

The height map from Figure 6 produced the following histogram shown in Figure 7.

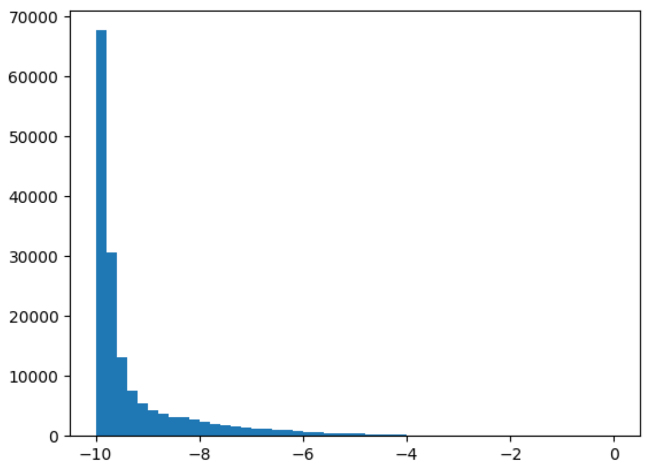

Figure 7: The histogram of the real height data.

The thresholded depth map is shown below in Figure 8, along with the location of the optimal landing location.

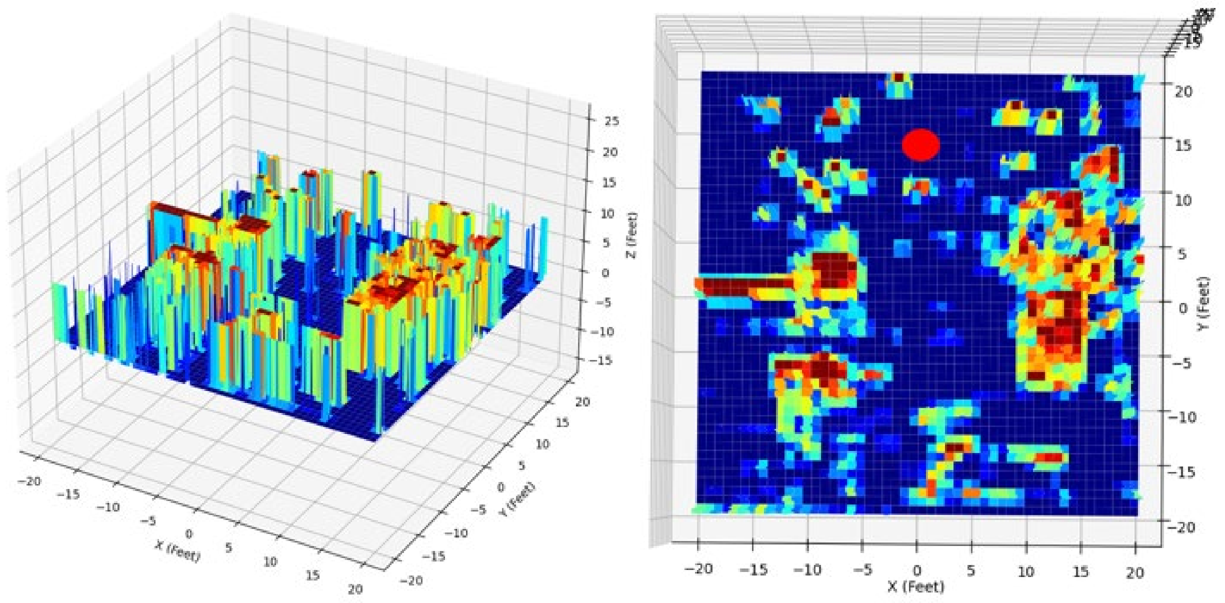

Figure 9: Left: "Thresholded" image depth map. Right: Location of optimal landing location.

As indicated in Figure 9, the optimal landing location is in the top center of the image, i.e., the area of the intended landing location as there were no nuts or cars in this location. Note the histogram in Figure 8 only somewhat resembles the histogram in Figure 2. The discrepancy allows for high overlap between the training data, but not perfect overlap. Lack of multimodal distributions within the histogram is attributed to the amplitude of the noise from the cameras being roughly the same size/intensity of the less robust features within the disparity maps. By forcing the left and right edges of the depth map to be 0 *Z* depth, it allows for a near-perfect prediction (see Section 4.3). In turn, small/precise features were effectively washed out, but thresholding is still possible, and therefore allows the system to function properly. From observing the 8.5 mark on the histogram shown in Figure 7, the graph tends to a plateau slightly. The plateau is the "ghost" of the multimodal distribution. With higher quality cameras, the feature clarity will be much easier to distinguish and therefore decrease error in the prediction of the depth.

2.0 THEORY DEVELOPMENT

2.1 HISTOGRAM & FEEDFORWARD NEURAL NET

Histograms were chosen to feed into the neural network, as it provides an “easy to distinguish for humans” method to identify roughly the average depth of the ground. The method is demonstrated in Figure 10.

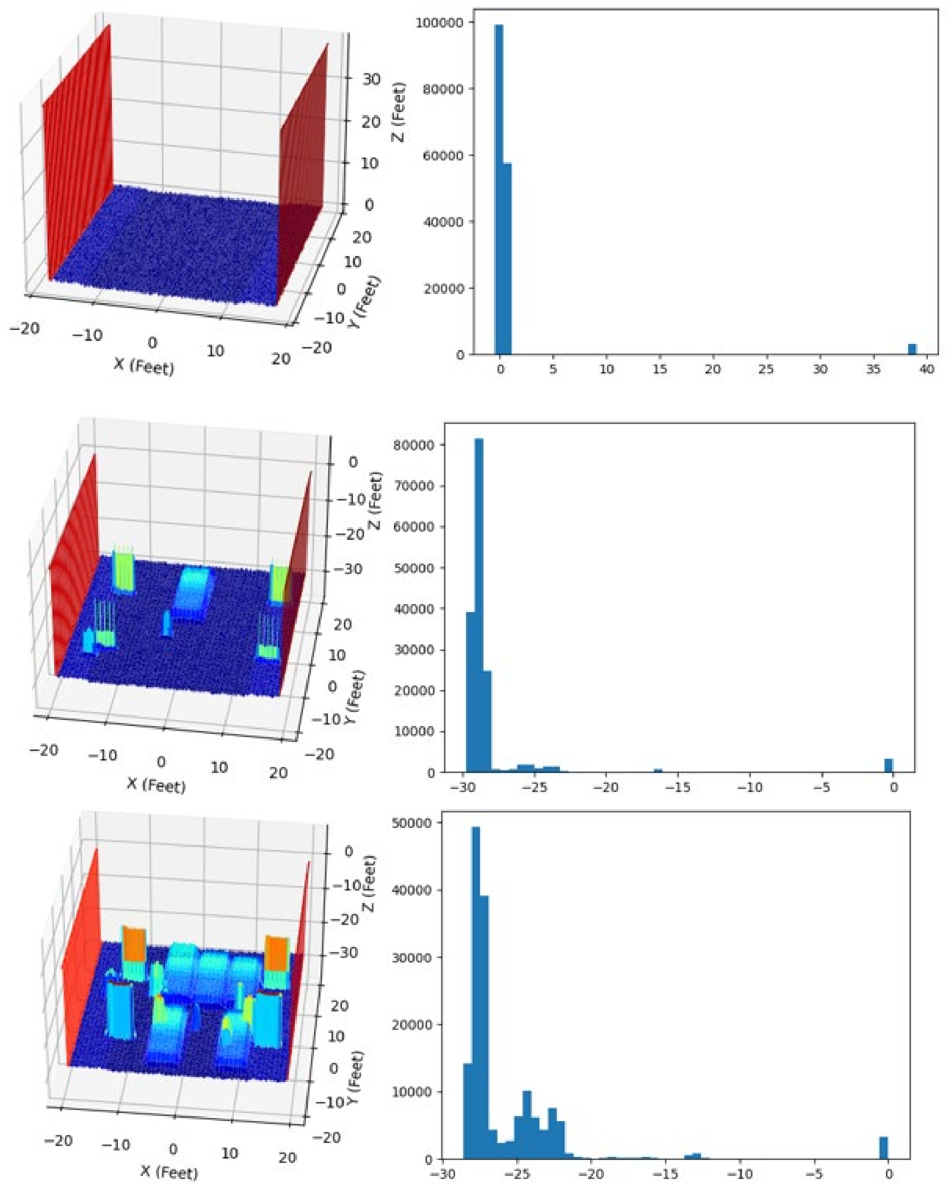

Figure 10: Left column: Randomly generated Scenes. Right column: Randomly Generated Scene’s respective Histogram plot.

There exist a narrow, common distribution, roughly resembling a normal distribution. The distribution is essentially what is the deciding factor for the neural net. The input data to the neural net is a rescaled set of the histogram's counts to exist within [0,1] (data divided by max of data), concatenated with the 51 bins for the histogram. Initially, the net was unable to determine the value of the depth, as just the raw counts from the histogram were input. The input data did not retain the scale that mapped the histogram to the appropriate depth boundaries. Raw count values were normalized due to the large input range of the counts varying too much ([0, >80,000] to [0, 1]). Once the counts were normalized and the scale was incorporated, the net was trainable.

The net was trained using an input shape of 101 inputs as stated, a hidden linear layer of size 50, a hidden linear layer of size 30, and an output layer of size 2. The neural net's development was facilitated by Pytorch (a Python library) [23]. Values for the net are as follows: 20 epochs, learning rate of 0.00003, batch size for the Pytorch data loader is 500 elements, RMSprop optimizer, ExponentialLR scheduler, and a train-test ratio of 80-20. A custom metric function was implemented to better approximate the output from the net and the training labels. The function took the absolute difference between the training data's labels and the output from the net and averaged all the values. The function will be aptly referred to as the "**average difference**", and the average difference between the predicted ground distance and the real ground distance reached 0.379" after 20 epochs. Random scenes were given a random depth between 0 and 40 feet, therefore driving the average difference down to under half an inch demonstrating the net operated well and was a functional method for determining the depth.

2.2 OPTIMIZATION FUNCTION

The objective function implemented was different than what was initially proposed. A bounding box encompassing the aerial system in its entirety is still used to represent the aerial system and its respective footprint within the scene below. The dimensions of the box are as small as possible while still encompassing the aerial system in its entirety. For the purposes of the objective function, the box is only the footprint, such that the face of the box on the ground is being evaluated. The bounding box is brought into fruition after the scene has been thresholded to 0 and 10. Instead of calculating the area of the intersection of the bounding box of the autonomous aerial vehicle and the thresholded scene, it instead sums all the values within/under the bounding square and is used as a penalty function. The thresholded scene only consists of either 0 or 10, corresponding to ground and object (or at minimum, "not ground") respectively. If the summation results in 0 then the square does not intersect with any part of any object. In other words, the square existed only on the ground and not within/over/on objects. The sum under the bounding box is referred to as **Zsum**.

The optimization routine still varies the *X* and *Y* position of the center of the square and length of the square to fit the largest size square into the image. A square bounding box was chosen over a circle to minimize computational complexity and reduce execution time. To maximize the square length while reducing the cost function, the determined length was inverted (refer to **Rf** as the inverted length). Within the code, half of the length is maximized and was used to base the center of the square at the optimized *X* and *Y* values, then all values within +-(length of the square)/2 in the *X* and *Y* direction will be summed to form **Zsum**.

Part of the objective function was penalized if any of the bounding box exited the scene. The value was 0 when the bounding box existed strictly within the scene, and 10E20 when it existed outside the scene. This created a pseudo-exterior-penalty function and pushed the optimization's search back within the scene. The penalty function will be referred to as **Perr**.

Therefore, the function to minimize is as follows:

$$Loss = R_f + Z_{sum} + P_{err} \qquad \textbf{\textit{Eq. 1}}$$

The optimization routine chosen was SciPy's Fmin (Python Library) [22]. The routine was executed at minimum of 25 times, where the starting points were arranged within a 5x5 grid. If the length of a scene was L feet, and all scenes are square, then the length of the grid of 25 initial condition points was $0.90*L$. The grid was centered at the center of the scene. The smaller grid was chosen due to the **Perr** value, since if the initial conditions immediately triggered **Perr** in other words pushed the bounding box outside the scene, the loss function would not change from ~10E20 and increase the likelihood of failure. The lack of change of the function is akin to the sparse reward problem within AI.

2.3 CAMERAS, MONOCULAR CALIBRATION, AND STEREO CALIBRATION

Cameras implemented in the solution were Logitech's C270 Cameras (A1). Each camera only had a 480x640 image and rendered at best 720p resolution. Low-quality cameras caused most of the issues with the vision system, as they were grainy and noisy. By utilizing Python's OpenCV [21] library (reference), calibrating the cameras was straightforward; however, obtaining a solution to the stereovision was dubious. Code from Nicolai Nielson's Computer vision library was adapted for this purpose (A2). The code only operated with the assumption all images for the left and right images contained the checkerboard calibration artifact (Figure 11).

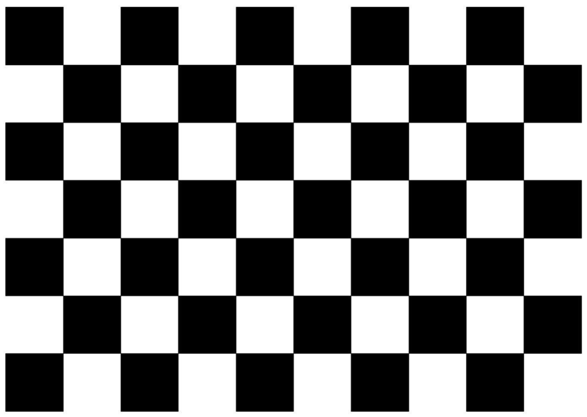

Figure 11: OpenCV 9x6 Checkerboard Calibration Artifact

Calibrating a stereo vision system whose cameras were much further apart than Nielson's setup was not within the scope of his code. The initial system had the cameras 22" apart, as the larger the distance between the cameras, the more accuracy in the depth estimation. The problem with the cameras being 22" apart was the conditions for testing were too close to the cameras to yield viable depth maps, therefore the cameras were brought together to roughly 6 inches. Once the cameras were in this position, calibrating them was much easier as well due to the significantly higher overlap of the images within 4 feet from the camera.

The code from Nielson was modified to calibrate each individual camera with 32 images containing the checkerboard artifact. 34 pictures were taken, 2 images in each set did not yield a full checkerboard within the image, and 29 images contained a checkerboard in both images. The 29-image subset with a checkerboard in each image was used to calibrate the stereovision system (example images are shown below in Figure 12).

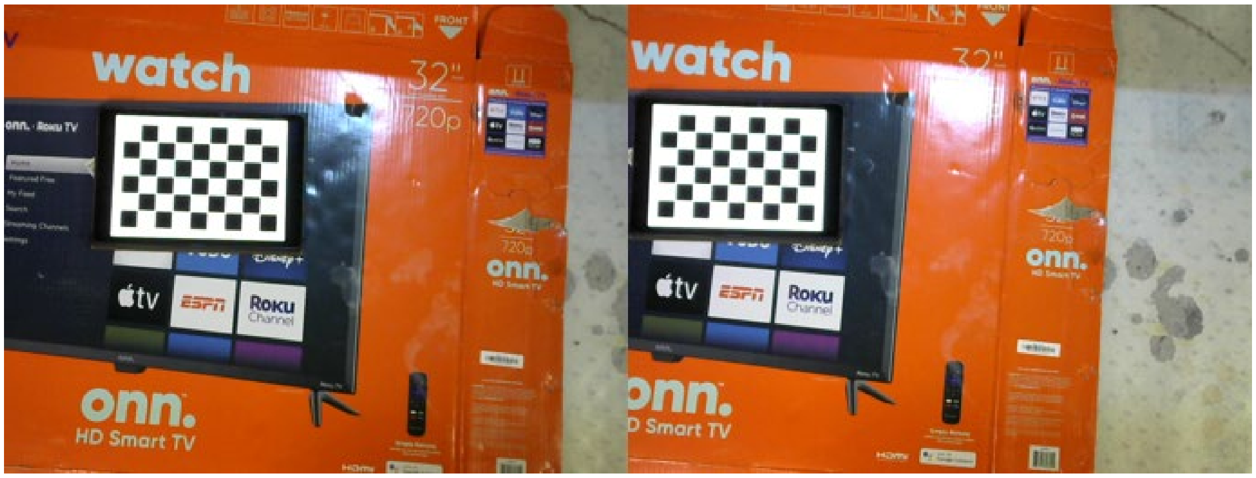

Figure 12: Left: Left camera image. Right: Right camera image.

A Galaxy Tab S6 (reference) was used to display the checkerboard, the size of the length of a square in the checkerboard was 20mm. The calibration routine identified the interior corners of the checkerboard to determine all camera intrinsics and calibrate the camera. Figure 13 shows OpenCV highlighting the checkerboard's internal corners.

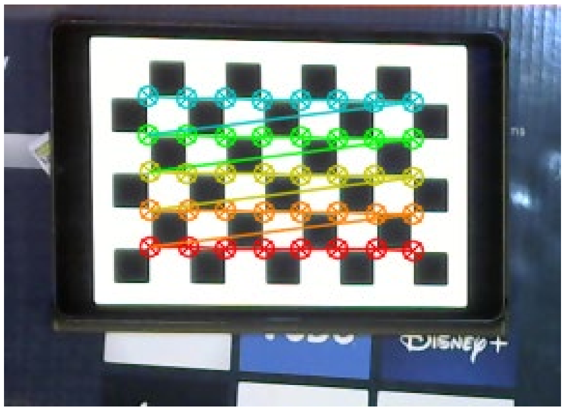

Figure 13: Checkerboard Identification.

After the calibration process was executed, the incoming images were rectified such that the cameras are projected onto a common image plane. Once on the common image plane, the disparity can be determined, and a depth map is developed.

2.4 IMAGE AND DEPTH MAP PROCESSING

Raw images from the implemented cameras were course and contained non-uniform, seemingly random amplitude noise. The results provided a challenge in correctly executing the vision system. Without processing the depth map as described later, the depth map oscillated in intensity and created random shaped "blobs" appearing and disappearing with each passing image. Blobs were the product of the stereovision system detecting noise in each pixel, invisible to the human eye, but visible to a camera system. The resultant flooded the image with such excessive noise, the data was rendered almost useless. The depth map is shown in Figure 14.

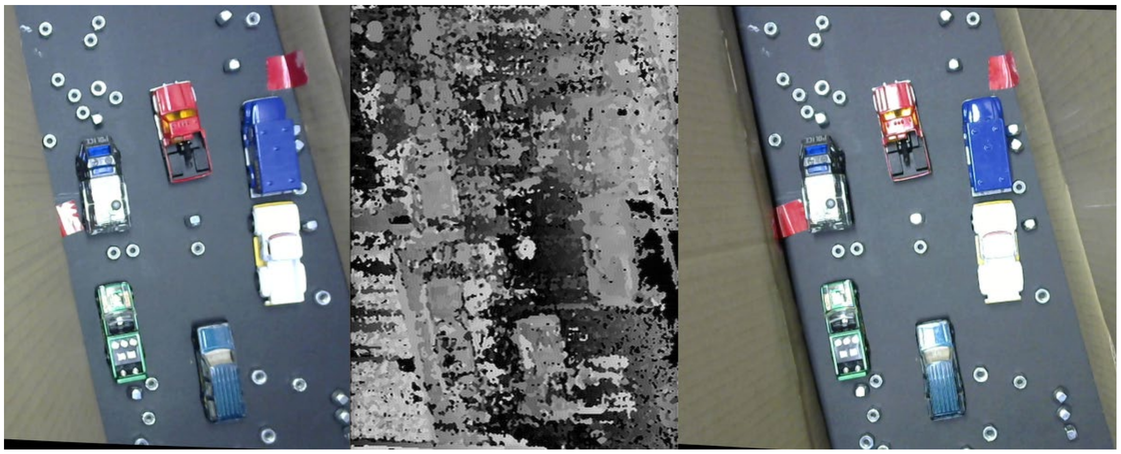

Figure 14: Left: Left camera image. Middle: Unprocessed Depth map. Right: Right camera image.

Pixel intensity changed drastically and rapidly aside from those pixels corresponding to the cars and the nuts. Therefore, a penalty function was implemented such that the intensity of the image was adjusted according to how much each pixel changed over 10 images. The resulting depth map is seen in Figure 16. The grayscale image was modified with a color axis similar to the one traditionally used by MATLAB [24], the scale is shown below in Figure 15.

Figure 15: MATLAB's Parula scale. Left corresponds to low values, right corresponds to high values.

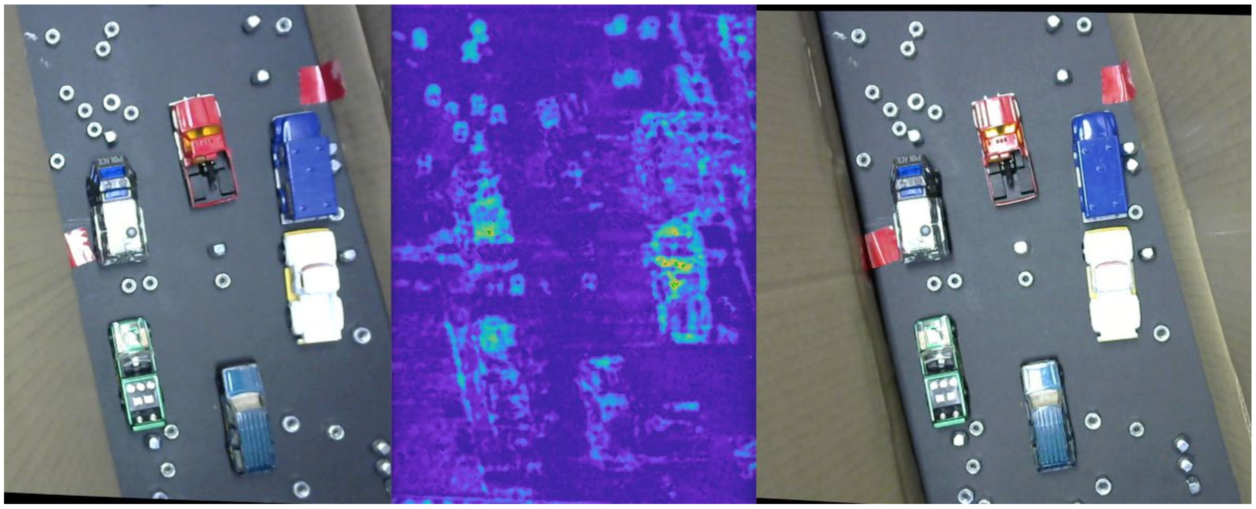

Figure 16: Left: Left camera image. Middle: Processed Depth map. Right: Right camera image.

The process to achieve the following image was as follows:

1) Collect 10 grayscale images
2) Obtain the difference between images
   a) results in 9 difference maps of the size of the input image
3) Obtain the absolute value of the difference map values from Step 2
4) Sum all the difference maps along the time axis from Step 3
   a) results in a single map of the size of the image
5) Invert the values of the summed difference maps from Step 4
6) Rescale the derivative map from Step 5 to be between 0.0001 and 8
7) Element-wise multiply the grayscale image by the inverted difference map

A further Mathematical representation is given below

$$Let\ I\ represent\ a\ set\ of\ Q\ grayscale\ images.$$

$$I\ is\ a\ N\ by\ M\ array\ of\ values, with\ depth\ Q. Thus\ a\ single\ element\ will\ be\ indexed\ by\ I_{nmq}$$

$$Let\ P\ represent\ the\ map\ whose\ elements\ are\ inversely\ proportional\ to\ the\ derivative\ of\ I$$

$$Therefore, P\ is\ M\ by\ N\ by\ 1\ in\ dimension$$

$$P_{nm} = \frac{1}{\sum_{q=0}^{Q-1} \left| I_{nmq} - I_{nm(q+1)} \right|} \tag{2}$$

$$P' = Filter_{Median}(Filter_{Median}(Filter_{Median}(P))) \tag{3}$$

$$P'_{BottomShifted} = P' - MIN(P') + \frac{1}{8} * 10^{-4} \tag{4}$$

$$P'' = \frac{8 * P'_{BottomShifted}}{MAX(P'_{BottomShifted})} \tag{5}$$

$$Let\ X\ be\ the\ incoming\ grayscale\ image\ from\ the\ camera, and\ X'\ be\ the\ processed\ image$$

$$X'_{nm} = X_{nm} * P''_{nm} \tag{6}$$

The map *P* generally contains several extraordinarily large values relative to the mean of the image and provides a new issue. The issue can be overcome by applying 3-3x3-median filters to the map, in later renditions of the approach less or no filters may be needed. Median filters are meant to remove salt and pepper noise within an image, which is shown being removed from an image in Figure 16.

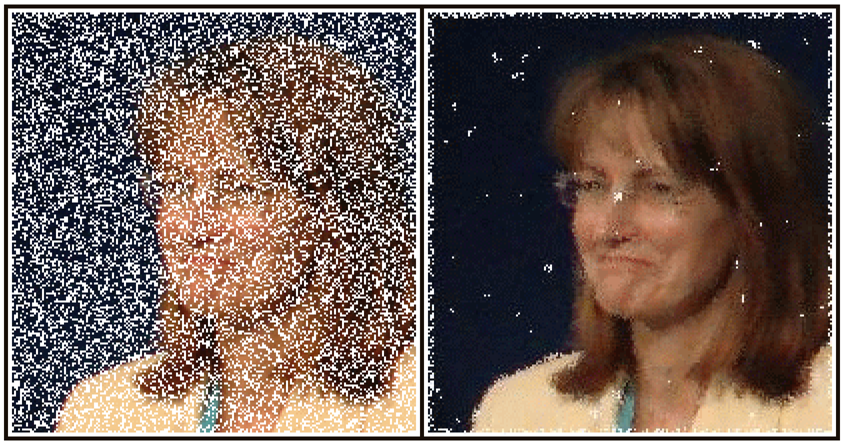
Figure 17: Salt and pepper noise is removed from an image, image supplied from OpenCV.

Median filters work in the scenario as it smooths drastic spikes in arrays. White pixels are represented as 255, and black is represented as 0, therefore with random white spikes in a dark image, filtering would reduce the spike and replace it with the median of the surrounding data. With the spikes removed, the derivative map becomes almost usable. The map is then scaled between 0.0001 and 8, therefore, the low changing pixels correspond to high values, and the high changing values correspond to low values. Note: the desired map does not cause pixel dropout, only causing large/vigorous rescaling. If the minimum of the filter was set to 0, the rescaling would eliminate the pixels that changed aggressively entirely.

2.5 USING THE DEPTH MAP TO FIND A LANDING LOCATION

With the depth map aggressively filtered using the derivative-based penalty function, the incoming image must be resized to 400 by 400 elements to match the training set size. Once resized, the depth map appears as shown in Figure 18.

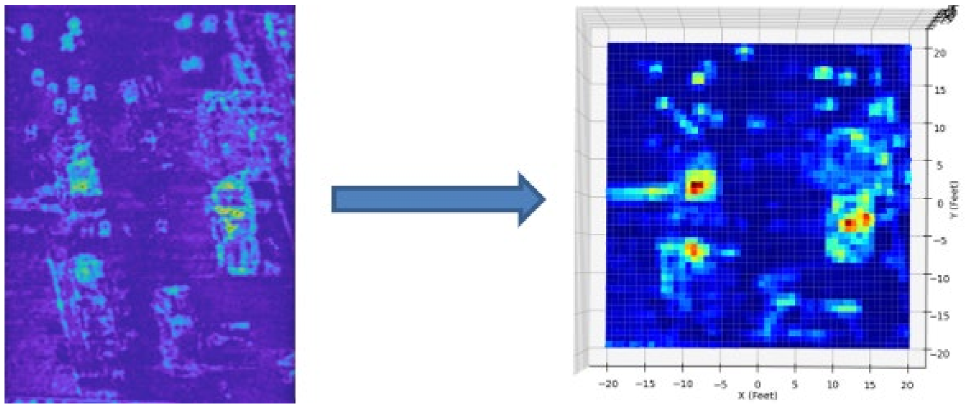

Figure 18: Depth map converted to XYZ Height map representation.

Obviously, akin to the randomly generated scenes, the height data is aggregated into a histogram, and the outputs are fed into the neural network to determine the average ground depth and the standard deviation of the ground depths. Then the depth map is thresholded, and the optimal position is determined.

To map the position in the depth map back to the image, a scaling factor of 1.2 must be applied to the *Y* coordinate, and 0.88 to the *X* coordinate, after shifting the origin to the top left of the image and flipping the *Y* axis. The scaling factor is specific to the currently implemented cameras, where the usable depth map image is 352x480 (*X* by *Y*), and the shape of the test data is currently 400x400. Therefore, 480/400=1.2, and 352/400=0.88. Whatever camera system is implemented for future evolutions of this work will likely have a different scaling factor. 400x400 is arbitrarily chosen in this case to demonstrate the image in the same manner as a fabricated RS. To display the height map easiest, a RS was generated, and the *Z* height was replaced with the vision system's incoming height map. Therefore, the 400x400 was not entirely arbitrary, but merely chosen for ease of use. In practice, since the count from the histogram is normalized, the rescaling of the image would not be required.

Due to the quality and noise in the base images, actual dimensions cannot be determined. Actual dimension determination would be possible with higher quality cameras using the new camera's focal length and other mechanical properties. The general shape and depth relative to objects in the vision system-based scene can be determined, and therefore regardless of the absolute dimensions of the underlying scene, vectors directing the autonomous aerial vehicle towards the desired location can be used to navigate the autonomous aerial vehicle to the optimal landing location. Also, since the final size of the autonomous aerial vehicle is not determined, the largest open area within the scene is returned in place of the viability of landing the autonomous aerial vehicle.

3.0 SIMULATION RESULTS

The neural net was trained and resulted in the following set of values per epoch trained:

```
TRAINING:
        Loss of FFNN: 9.0 - epoch 1 - Average Difference 9.65
        Loss of FFNN: 7.0 - epoch 2 - Average Difference 8.31
        Loss of FFNN: 6.0 - epoch 3 - Average Difference 6.87
        Loss of FFNN: 4.0 - epoch 4 - Average Difference 5.21
        Loss of FFNN: 2.0 - epoch 5 - Average Difference 3.47
                              .
                              .
                              .
        Loss of FFNN: 0.0 - epoch 16 - Average Difference 0.04
        Loss of FFNN: 0.0 - epoch 17 - Average Difference 0.04
        Loss of FFNN: 0.0 - epoch 18 - Average Difference 0.04
        Loss of FFNN: 0.0 - epoch 19 - Average Difference 0.03
        Loss of FFNN: 0.0 - epoch 20 - Average Difference 0.03

TESTING:
        Loss of FFNN: 0.0 - Average Difference 0.0316
```

Figure 19: Loss function output during neural net training

Therefore, the average difference while testing the system on the test set yielded a difference of 0.03162 feet or 0.3794 inches. Most of the error in the prediction is due to scenes where the average depth of the scene is around 3 standard deviations of the max height of the objects within the scenes, like cars, people and or the light poles. As seen in Figure 20, the average difference is shown relative to specific depths.

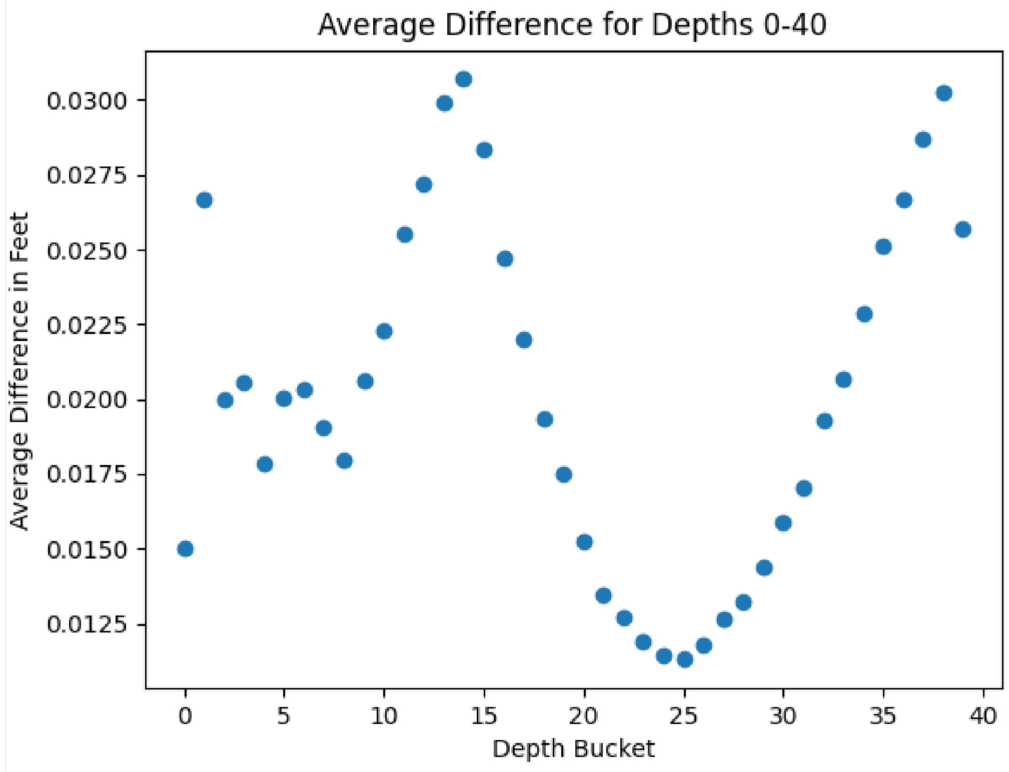

Figure 20: Average difference per depth bin.

Real depth values were truncated, therefore all depths between 20 ≤ Depth < 21 are collected into the 20 bucket and then averaged. The range for cars and SUVs within the random scene code are between 4 and 6.5 feet, which is why the error was around the median for this data. Once the depth was past 14 feet or the max height of the light poles, the error drops significantly. The reason it picks back up is likely due to random distribution being gaussian, and therefore had very few examples nearing 40 feet. The training set was developed to exist between 0 and 40 feet, hence the graph. The data presented within this graph is using all 50,000 random scene histograms.

The range chosen will likely affect real-world scenarios; however, this is a proof of concept for this system and should be sufficient. If more range is desired, then the range of the fabricated set should be increased, and the neural net retrained. Code will be provided to run such a set of calculations/data fabrication.

4.0 EXPERIMENTAL RESULTS

Due to the bad quality of the cameras, the exact dimensions of the objects were not able to be determined along with the depth. The inability to determine size was entirely due to the amplitude of noise being near the amplitude as the features determined. To counteract this, the values of the penalized depth map were scaled between 0 and 1, then rescaled to exist within the training set's range. Where the depth ranged from 0'-40', and vehicle heights ranged between 4'-6.5'.

4.1 EXPERIMENTAL SETUP

Figure 21 shows the precision setup of the camera system and the table on which the experiments were performed. Cardboard walls were implemented to simulate a parallax view of the scene below the vision system and give more depth to the scene.

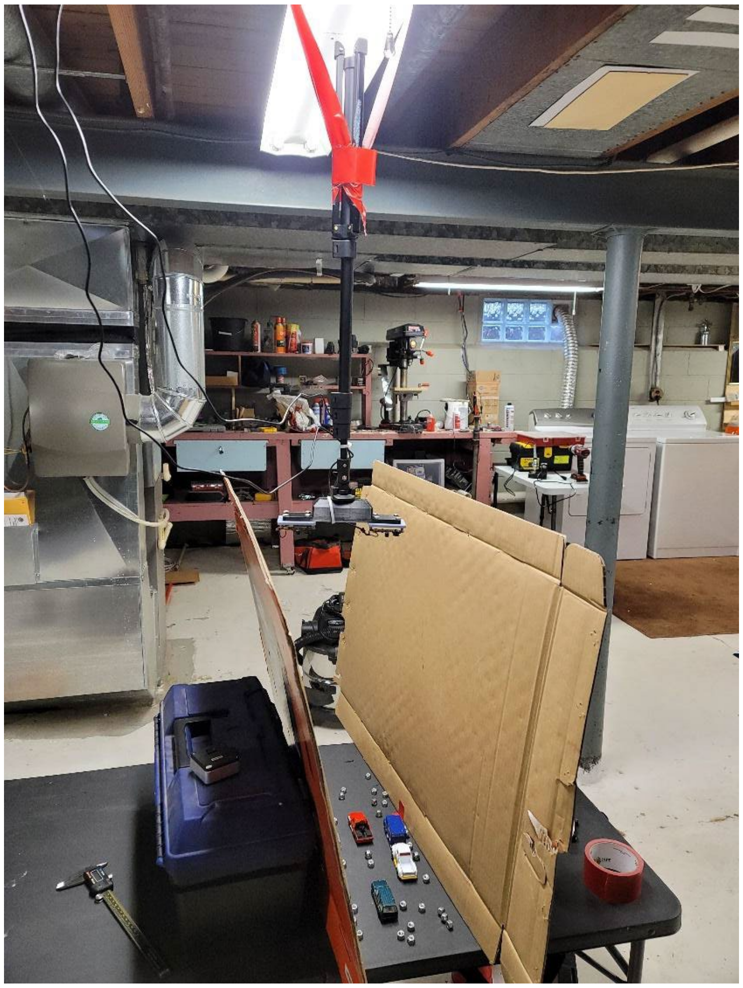

Figure 21: Experimental Setup, the vision system is attached to a robust system of duct tape.

Figure 21 shows the setup in which the two cameras were disassembled and mounted to a 3D printed fixture, and those fixtures were mounted to a 3D printed component to maintain the separation of a specific distance of 5.91 inches, or roughly 150 mm. The component the fixtures were mounted to has an embedded ¼-20 threaded insert, which allowed it to connect to a tripod as depicted in Figure 21. An up-close image of the cameras and mounting of the camera's circuit board is shown in Figure 22.

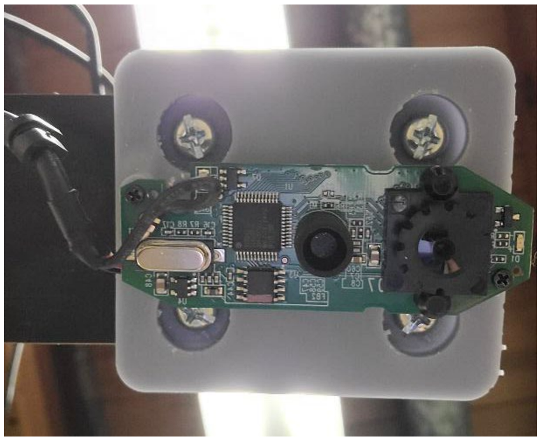

Figure 22: Up close view of the web camera system and mounting fixtures.

Then in Figure 23, the spacing fixture to which the camera mounting fixtures are mounted is shown.

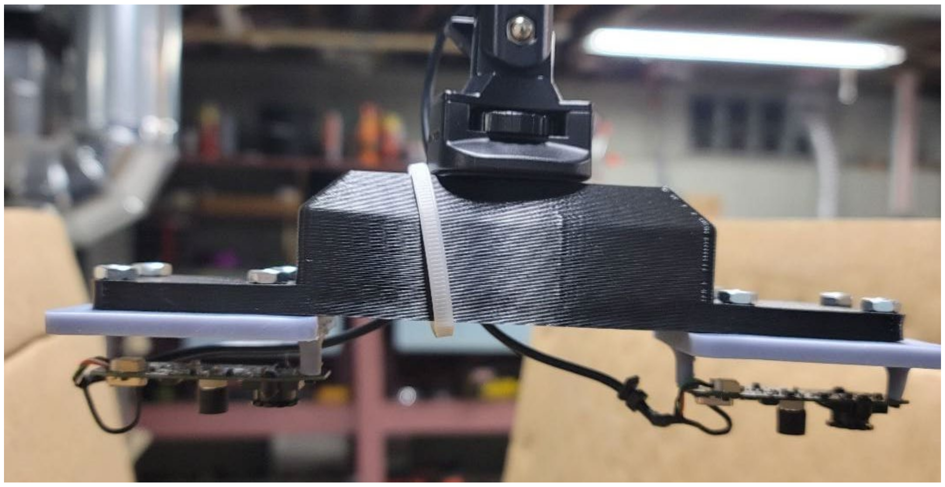

Figure 23: Full view of the vision system.

4.2 EXPERIMENT EXECUTION RESULTS

Below in Figures 24-28, are some experimental results of the system in action. The general guess for the landing position is marked by a red dot in the left and right images. Below the images is the progression from heightmap to histogram to the thresholded height map.

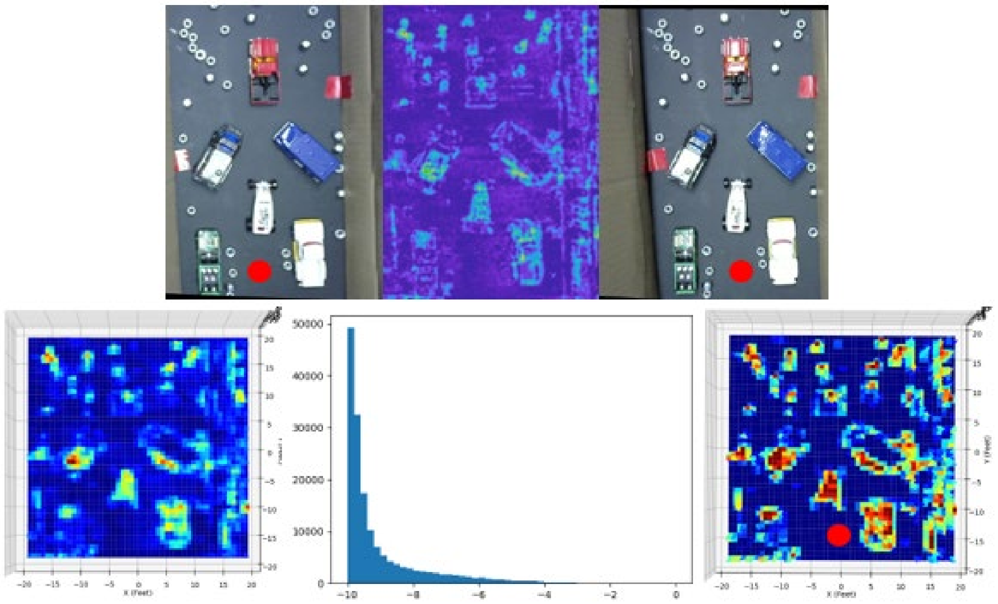

Figure 24: Arrangement 1. Top 3: (Left, Middle, Right) Left Image, Disparity Image, Right Image. Bottom 3: Depth Map, Histogram, Thresholded Depth Map with marked landing point.

Notice in Figure 24, that the dot is slightly to the right compared to the guesses for the landing position in the images. This scenario occurs sometimes since the depth map is mapped to gridded data, most shape edges become approximately flat, and line up with either the X or Y axis, and therefore create multiple solutions for a square to find the furthest distance from objects. An analogy to this is a square inside a rectangle, where one side is of equal length to the side of the square. The square would be able to slide within the rectangle to any number of positions, thus having multiple, equal costing positions. For illustration, see the below Figure 25.

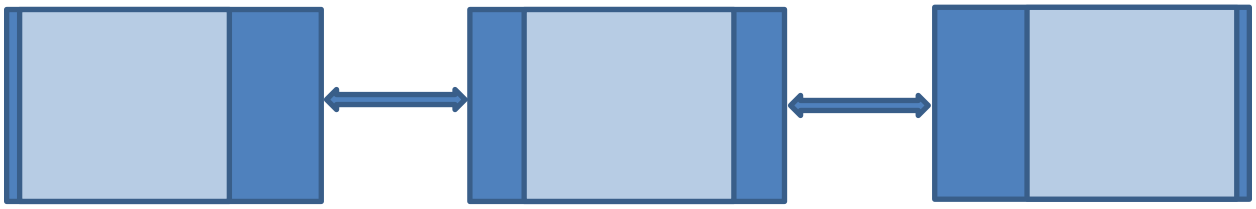

Figure 25: Optimal positions for a box within a rectangle.

It should be noted that for Figures 24, 26, and 27, there was a clear, anticipated position to land in (an area was deliberately cleared of debris). For Figure 28, the whole scene was purposely set up to have debris everywhere and no easy to distinguish position to land in.

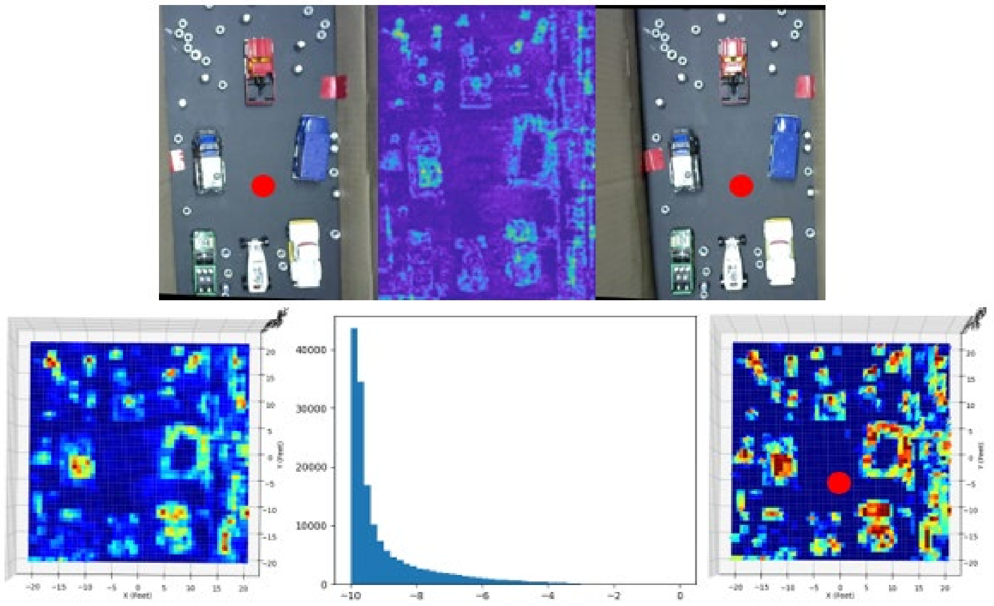

Figure 26: Arrangement 2. Top 3: (Left, Middle, Right) Left Image, Disparity Image, Right Image. Bottom 3: Depth Map, Histogram, Thresholded Depth Map with marked landing point.

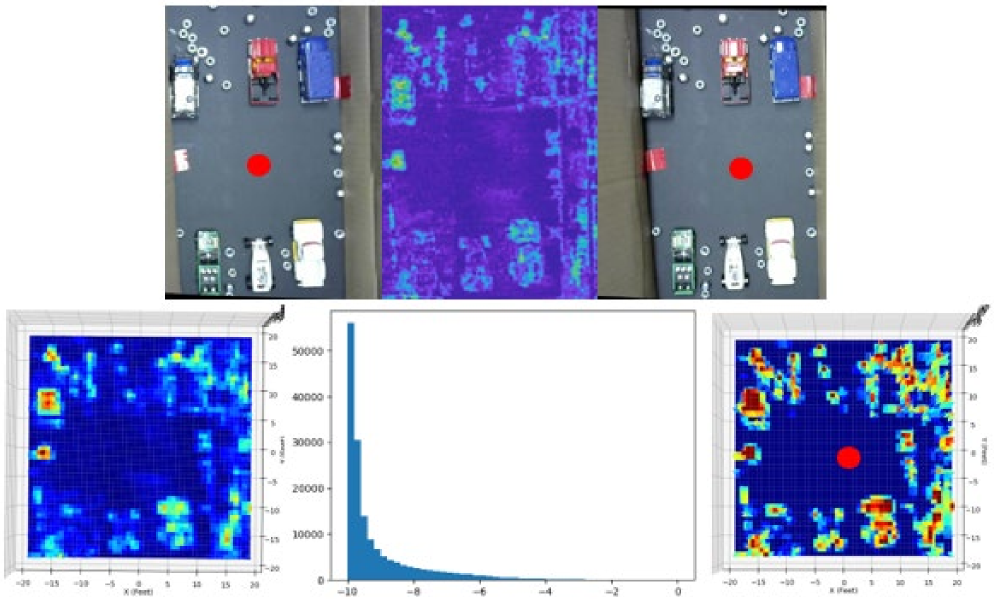

Figure 27: Arrangement 3. Top 3: (Left, Middle, Right) Left Image, Disparity Image, Right Image. Bottom 3: Depth Map, Histogram, Thresholded Depth Map with marked landing point.

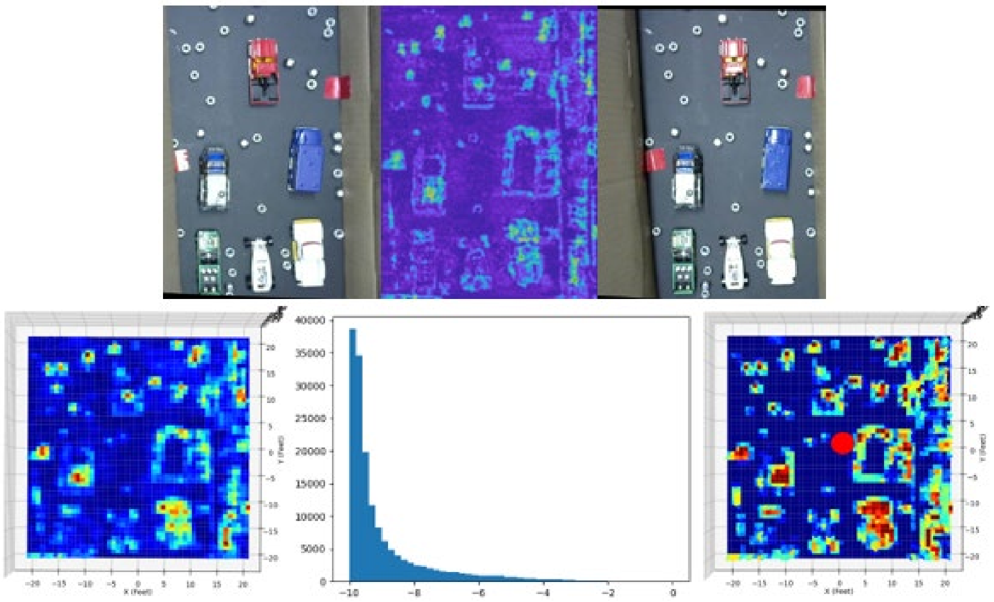
Figure 28: Arrangement 4. Top 3: (Left, Middle, Right) Left Image, Disparity Image, Right Image. Bottom 3: Depth Map, Histogram, Thresholded Depth Map with marked landing point.

This final position in Figure 28 occurred due to the initial condition starting in the middle of the image, and then the rectangle problem stated in Figure 22 occurred. It may appear that since the optimization routine was locked into that small space between the nut and the blue van, it was unable to proceed further. Which would resemble the sparse reward problem since the value of the function did not change during the execution of the routine. However, even after giving an additional starting position in the middle of the 3 nuts in the center and between the two cars, the optimization routine identifies that the resulting location is still the best. As seen in Figure 29; the position did not change much.

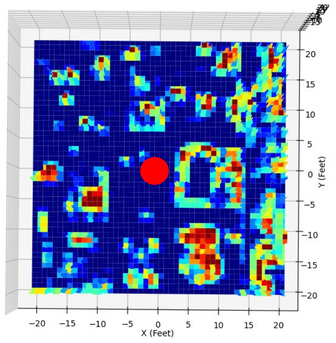

Figure 29: Additional starting point, but the same map as Figure 25, yields roughly same point.

4.3 PERFORMANCE METRICS

Since the cameras were too bad of quality to properly determine dimensions, it should be shown what operating conditions were assumed for testing within section 4.2 and then metrics for deviations from those conditions. For instance, Z depth for the experiments in 4.2 existed between [-10, 0]. It should be noted that to make the data look even closer to the training set, the depth maps were modified to set the left and right edge values to 0. If a depth map is represented by a 400 by 400 grid of data, then the first and second column, along with the last 2 columns has their values set to 0. Once the values were forced to 0, the results are graphs below. Forcing the values to 0 is not a fabrication of data, but rather a loss of values that causes more overlap between the training set and real data. In turn, this loss of data ironically increases the accuracy of the system. Below in Figure 30, a graph illustrating the average difference between the predicted depth and the actual depth is given. The input depth map was scaled between [0,10] and then the depth from the x-axis was subtracted from the depth map.

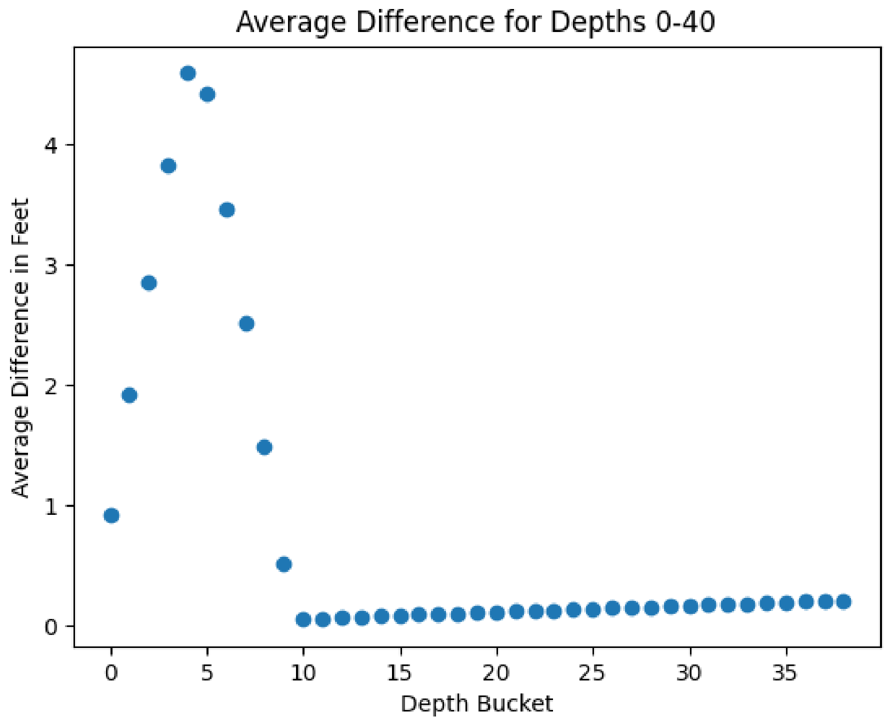

Figure 30: Average difference for predicted depth vs real depth of real scene data, rescaled between [-10, 0].

If you notice, the minimum of the graph is around 10 and deeper, where those predictions were the best or closest to the actual values. The point on the graph is an important point, as the neural net was not accurate when the scene had any positive depth map values. Since the Z depth initially was between [0, 10], any value along the X-axis subtracted from those values less than 10 should make no sense to the neural net, as there would be some positive and negative values. Figure 30 illustrates that the net is accurate on data where the map is at minimum touching the camera and further away until at least a depth of 39 feet. Further examples are shown below (Figures 31-33) for different scales of the depth scene.

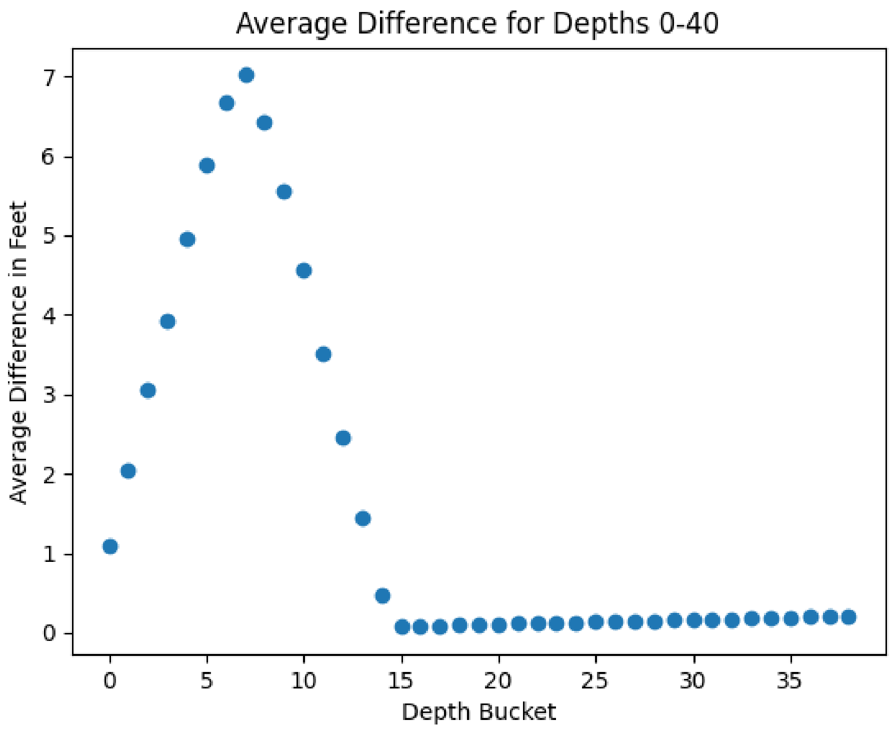

Figure 31: Average difference for predicted depth vs real depth of real scene data, rescaled between [-15, 0] and shifted down with the X-axis values in the graph.

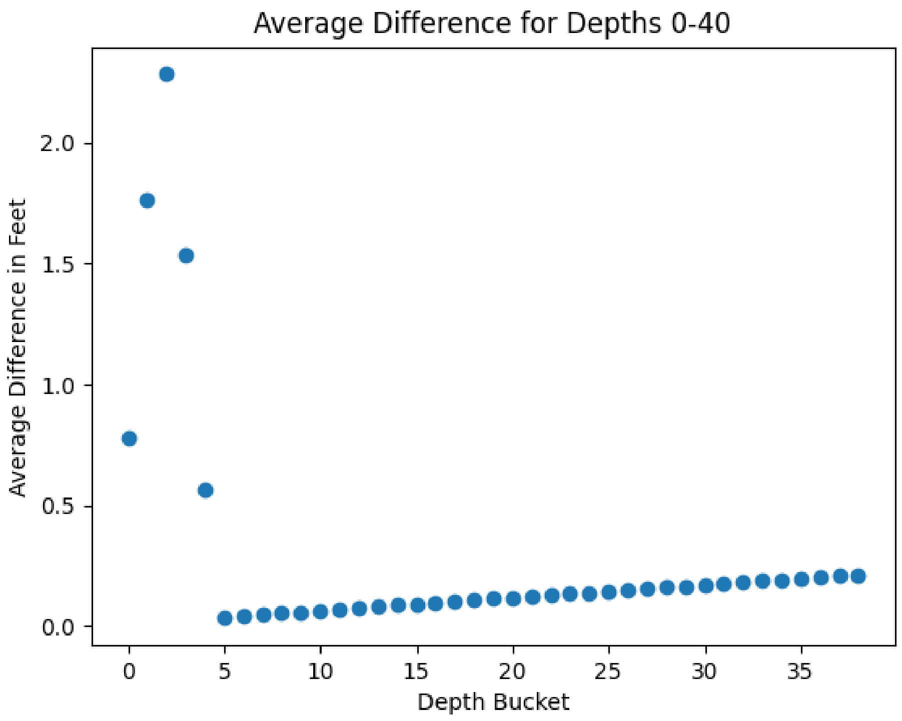

Figure 32: Average difference for predicted depth vs real depth of real scene data, rescaled between [-5, 0] and shifted down with the X-axis values in the graph.

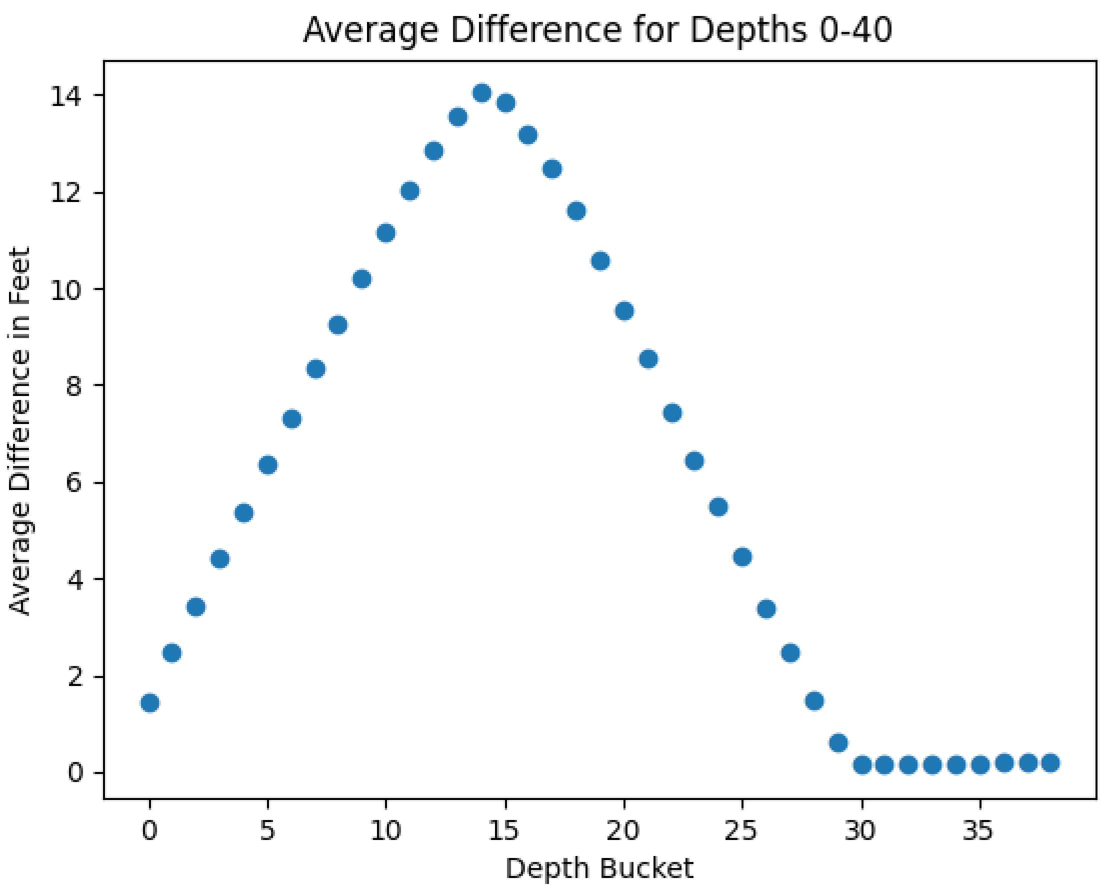

Figure 33: Average difference for predicted depth vs real depth of real scene data, rescaled between [-30, 0] and shifted down with the X-axis values in the graph.

As seen in Figures 31-34, the error of the system increases after the depth of the map but is still significantly lower than when the map is effectively inside the camera. The scenario where there are positive values on the depth map should never happen and therefore that data is omitted. Thus the system is operational in determining open areas, but not in exact depth due to the quality of the cameras.

At the current state, the routine takes roughly 12.3 second to execute (using [A3]): remapping the depth map to a random scene structure, taking the histogram of the height data, feeding the histogram into the neural network, thresholding the random scene structure according to the neural network, and then finally (the bulk of the time) determining the largest open area within the scene to land within. The list of steps being measured does not consider the camera image acquisition component, as that is a function of the hardware itself. Including time for the image acquisition would skew the execution time as it is not reflective of the system as a whole. Currently, however, for the system to stabilize with the low-quality cameras being utilized, it takes around 5 seconds to acquire appropriate images for the scene underneath the aerial system. This is directly a function of the roughly 9 frames per second acquisition rate the camera retains currently. With faster cameras, it would execute significantly faster.

5.0 CONCLUSION

In this work, a computer vision-based system was developed for using in fully autonomous aerial vehicle operation. The work focused on solving a current gap in the technology by considering a system for autonomous landing with in highly congested urban environments. The environment was solely constrained to urban type applications and did not consider landing in highly variant scenarios such as forest, on hills, rivers, etc. The algorithm used two images and determined the disparity of the images to develop a depth map relative to the location of the two cameras. A histogram of the depth of objects within the depth map was then developed. Once the histogram was finalized, the data was fed into a neural net and the output of the neural net algorithms was used to threshold the depth map. An optimization routine was then used to determine the largest "optimal" landing area. Simulated images were first used to test the feasibility of the approach. The algorithms were successful in identifying an optimal location for the aerial system to land. In addition, hardware testing was performed using two cameras to image a parking area. The algorithms were then tested using depth map resulting from the two obtained images and a resulted in obtaining an optimal landing location.

5.1 REQUIREMENTS FOR EFFECTIVE OPERATION [IN CURRENT STATE]

For this system to operate effectively, the optimal position returned from the optimization routine should be interpreted as a vector rather than a point to move the autonomous aerial system. If the vector is scaled by the inverse of its norm, it could be effectively interpreted as a velocity vector. By utilizing the system as a method to determine the direction to land, a control system could minimize the distance from the current position to the identified position by changing the velocity according to the direction and magnitude of the developed vector. The approach would omit the need for absolute position and therefore arrive above the identified position regardless of exact units.

For the camera system to function properly, a camera with a high rate of image acquisition must be employed. This caveat and a gimbal system would prove ideal, as the derivative-based penalty function would prove ineffective with excessive movement/shifting of images. Therefore, with a short burst of images encompassing the same position, the system would process multiple images (e.g., greater than two used here) as the same location with little change after collecting the pictures. The little change in position would incur little penalty.

The optimization routine could be used as feedback for the flight control system regardless of the coordinate system chosen. The controller would be able to work to minimize the difference between the current position and the desired position since the *X* and *Y* direction are the same. Once the magnitude of the vector is small enough, the autonomous aerial system is able descend.

If the autonomous aerial system descends before directly above the landing zone, the cameras may not capture debris on top or on the side of buildings, therefore only a direct downward descent would be recommended.

5.2 RECOMMENDATIONS FOR FUTURE WORK

Further tuning of the neural net could yield greater results in its current state, as the input data does not exactly match the shape of the test data. This measure would yield more accuracy if a transfer learning mechanism was employed and tweaked according to the real results. As stated prior, the accuracy should increase drastically when using higher quality camera with much better resolution.

Significant testing and tuning could yield a less computationally costly method of thresholding if a fully convolutional neural net was implemented. The input data would be the raw depth map, and the output would be the thresholded map. The approach would gain little additional advantage over time but would still potentially quicken processing once the images are taken.

Lighting effects generated quite a difference in depth maps as well. Intensity of light of one pixel or set of pixels in one camera compared to the other caused some dropout in the depth map as observed in Figure 35.

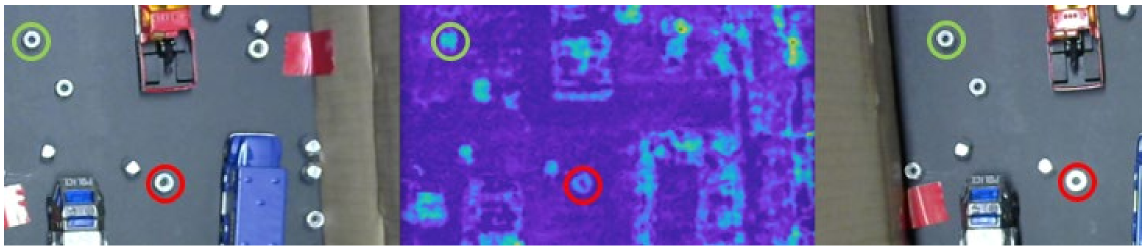

Figure 34: Subset of images highlighting differences in intensity affecting clarity in depth maps.

The nuts shown in Figure 35 within the green circles are roughly the same intensity within the left and right images and are clearer/higher in the depth map. Conversely, the nuts in the red circles have a much larger difference in light intensity, which results in a lower clarity of the nut in the depth map. Handling the light disparity could potentially be a very useful improvement for the system and would yield better results in the depth map.

One drastic improvement that is relatively capable with the current system is translating the code to machine code and executing almost entirely on a graphics card. Python has libraries that can facilitate this, and most of the code has been implemented to allow for seamless use of Python's library Numba, which would allow for the code to transfer to machine code, and then be executed on a graphics card. Through some cursory research, NumPy can be easily converted to machine code, whereas SciPy cannot. An alternative gridded interpolation function has been implemented strictly utilizing NumPy, which was one of the only issues in the improvement. The alternative interpolation function implemented a bilinear interpolation of gridded data omitting one of the issues of SciPy libraries being implemented on a graphics card via Numba. The only other function from SciPy libraries is from the optimize section, where "fmin" was employed as the optimization routine of choice. If an alternative to the function is implemented, there should be no issue with a full conversion from Python to machine code. If "fmin" is implemented by hand, using strictly operation from NumPy, then the whole system should in theory be relatively easily transferable to machine code via Numba. The whole sum of code could also be translated to a lower language manually as well.

5.3 CURRENT STATE OF MECHANICAL DESIGNS FOR AERIAL SYSTEM

The mechanical design of the Aerial system is demonstrated in Figures 35 through 39. The system has not changed much since the proposal, however after consultation with experts regarding the FOD mitigation cones mounted around the propellers, the cones would operate best when the larger end pointed down.

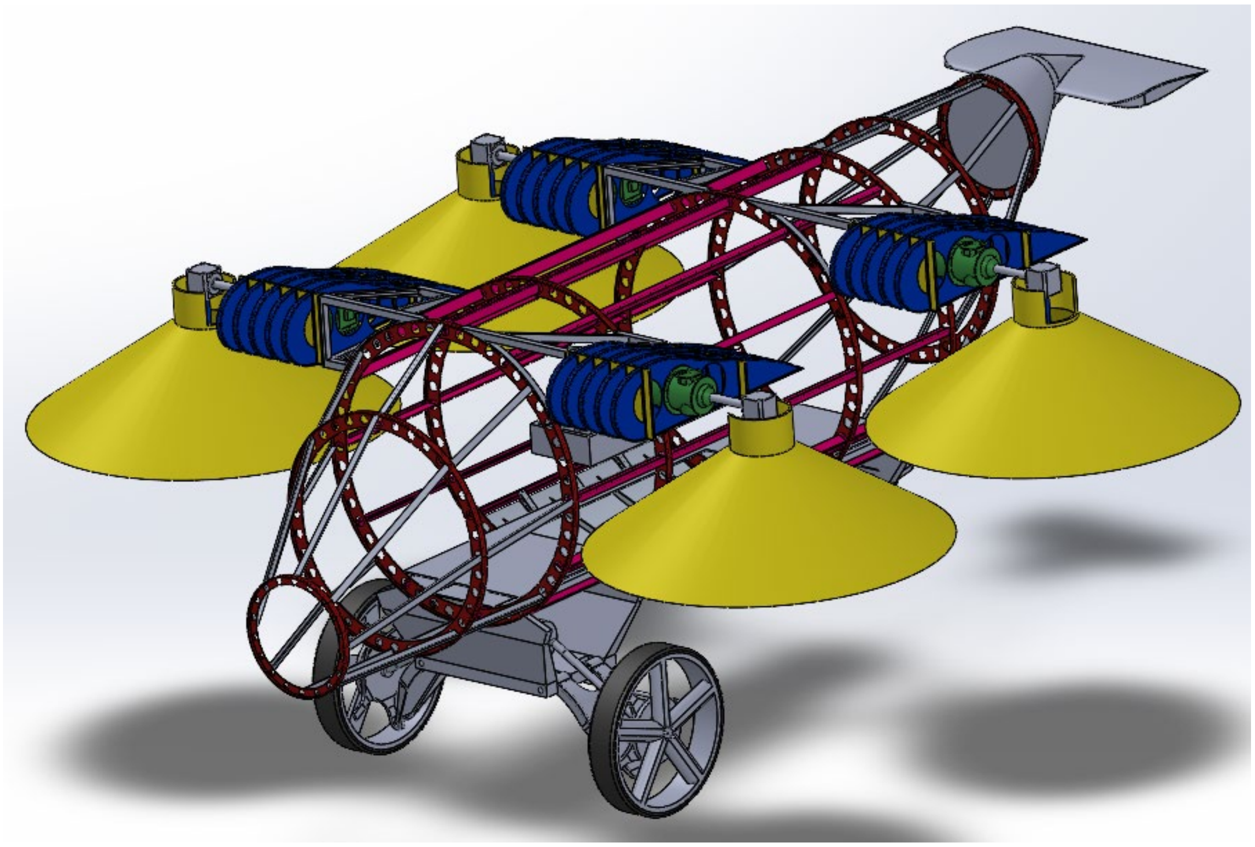

Figure 35: Angled View of the Aerial System.

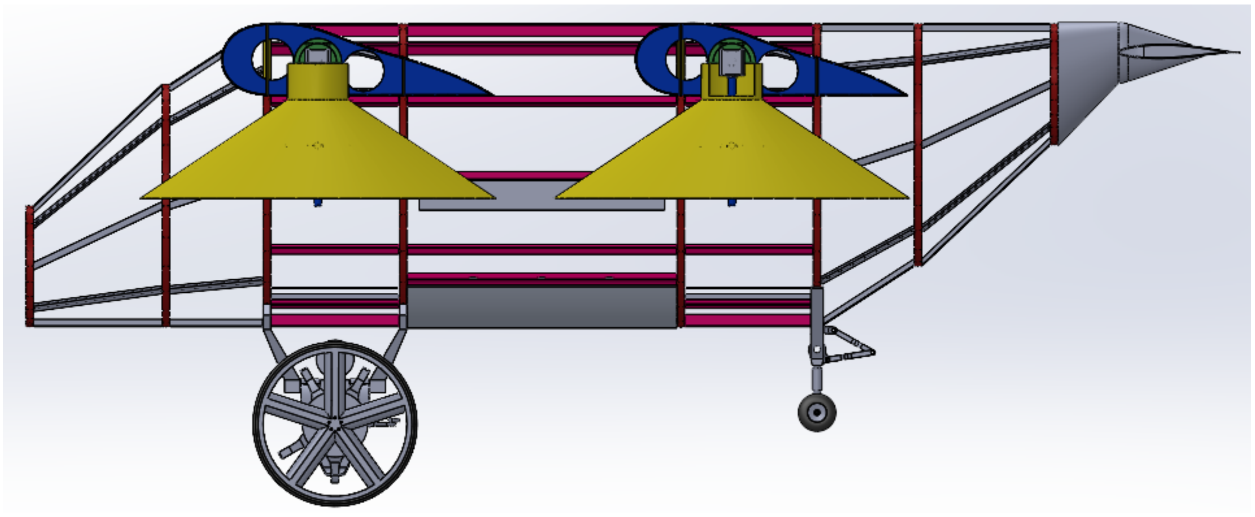

Figure 36: Side View of the Aerial System.

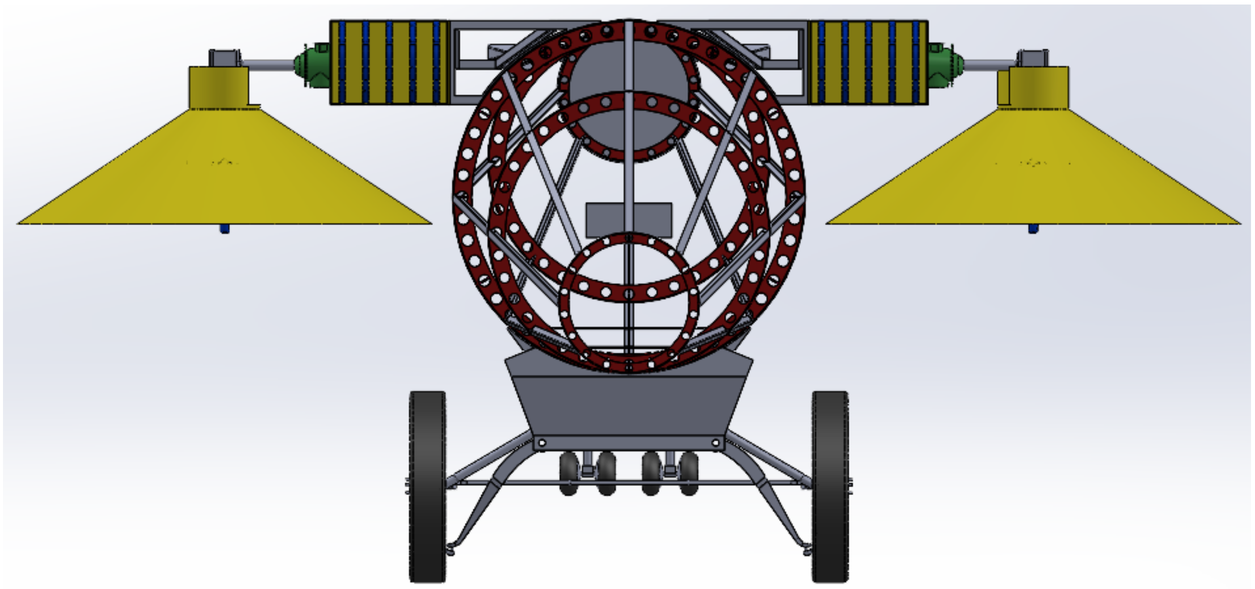

Figure 37: Front View of the Aerial System.

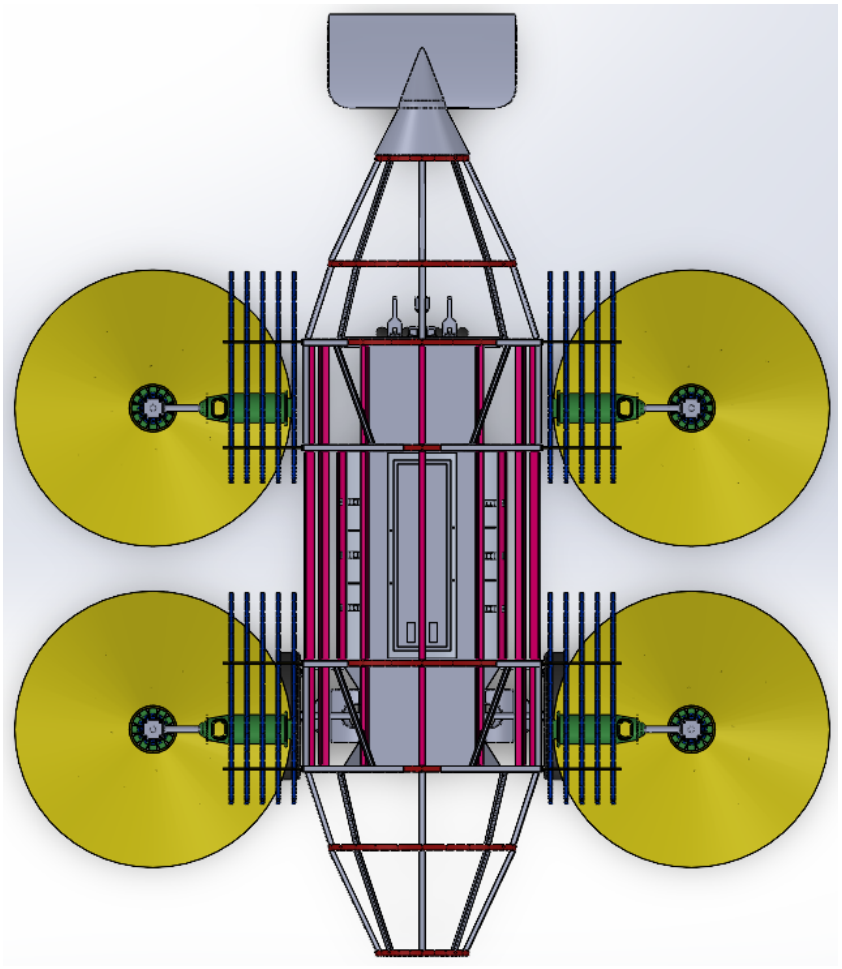

Figure 38: Top View of the Aerial System.

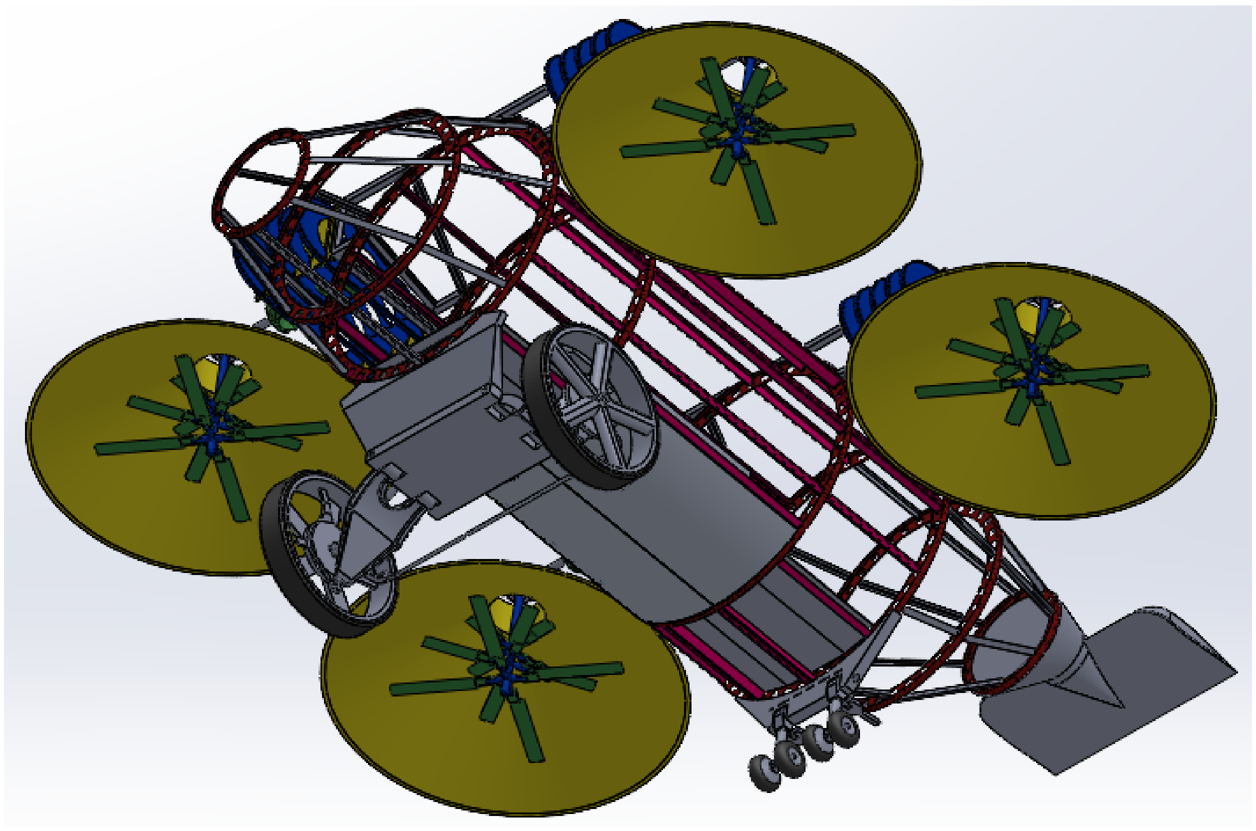

Figure 39: Second Angled View of the Aerial System; see propeller structure inside cones.

The mechanical components that still need to be implemented, at minimum, to be bearing reasonable semblance to the final design are as follows:

- The back wheels and wheel system require a full redesign.
- The exterior shielding plates for the entire plane.
- The joining structures for the FOD cones to the main body.
  - An aesthetic exterior shielding for this area.
  - A structural support section for this area. This component should mount to the 90-degree gear box directly mounted to the propellers and attach to the cone as well.
- A cockpit of sorts, this would include control mechanisms, gauges, etc.
- The interior currently is non-existent, and therefore would require a selection of items that a patient may need and or may find available within a standard ambulance.
- Proper design of a raising and lowering mechanism. This would include doors at the bottom of the Aerial system itself.

Some beneficial additional components to the mechanical design would include, but are not all encompassing:

- The current front wheels only appear functional; they should only be utilized as a visual representation of what the system should look like.

- The FOD cones could require some aerodynamical analysis using CFD to better chose a shape than just a simple cone.
- The overall shape has not been tested to validate an aerodynamic shape, only a general shape has been selected.

6.0 ACKNOWLEDGMENTS

I would like to thank Dr. Agamemnon Crassidis and Kody Viele for giving me the opportunity to assist in the development of the proposed autonomous aerial solution for use in medical emergencies and in particular, an autonomous landing mechanism and associated algorithms. I want to thank Dr. Salvaggio for guiding me and helping me debug many of the issues that came up with the vision system. I would also thank and acknowledge the help of Joey Streib in aiding in the development of the CAD models for the autonomous aerial system design. I would also like to thank the following people who have assisted me in the process of developing this Thesis: Amy Quattrociocchi, Brent Carlson, Ben Kennard, and Connor Mallo.

7.0 REFERENCES

[1] X. Sun et al., "Distance Measurement System Based on Binocular Stereo Vision", *vol 252, no 5, bl 052051*, Jul 2019. https://iopscience.iop.org/article/10.1088/1755-1315/252/5/052051/pdf

[2] M. Z. Brown, D. Burschka and G. D. Hager, "Advances in computational stereo," *in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 25, no. 8, pp. 993-1008,* Aug. 2003, doi: 10.1109/TPAMI.2003.1217603.

[3] Sun, Shaohui; Salvaggio, Carl, Aerial 3D building detection and modeling from airborne LiDAR point clouds, *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 6, 3, pp. 1440-1449* (June 2013)

[4] Sun, Shaohui; Salvaggio, Carl, Complex building roof detection and strict description from LiDAR data and orthorectified aerial imagery, *Proceedings of IEEE, IGARRS 2012, Analysis Techniques: Image Processing Techniques, Feature Detection in Images, Munich, Germany* (July 2012)

[5] Nilosek, David R.; Sun, Shaohui; Salvaggio, Carl, Geo-accurate model extraction from three-dimensional image-derived point clouds, *Proceedings of the SPIE, SPIE Defense and Security Sensing, Algorithms and Technologies for Multispectral, Hyperspectral, and Ultraspectral Imagery XVIII, Modeling and Simulation, 8390, Baltimore, Maryland, United States* (April 2012)

[6] Cheng X., Wang P., Yang R. (2018) Depth Estimation via Affinity Learned with Convolutional Spatial Propagation Network. *In: Ferrari V., Hebert M., Sminchisescu C., Weiss Y. (eds) Computer Vision – ECCV 2018. ECCV 2018. Lecture Notes in Computer Science, vol 11220*. Springer, Cham. https://doi.org/10.1007/978-3-030-01270-0_7

[7] Blachut K., Szolc H., Wasala M., Kryjak T., Gorgon M. (2020) A Vision Based Hardware-Software Real-Time Control System for the Autonomous Landing of an UAV. *In: Chmielewski L.J., Kozera R., Orłowski A. (eds) Computer Vision and Graphics.* ICCVG 2020. Lecture Notes in Computer Science, vol 12334. Springer, Cham. https://doi-org.ezproxy.rit.edu/10.1007/978-3-030-59006-2_2

[8] H. Supriyono and A. Akhara, "Design, building and performance testing of GPS and computer vision combination for increasing landing precision of quad-copter drone," *Journal of Physics: Conference Series, vol. 1858, (1), 2021*. Available: https://ezproxy.rit.edu/login?url=https://www.proquest.com/scholarly-journals/design-building-performance-testing-gps-computer/docview/2518768208/se-2. DOI: http://dx.doi.org/10.1088/1742-6596/1858/1/012074.

[9] R. C. Gonzalez, Digital Image Processing 2nd Edition, New Jersey: Prentice Hall, 2002.

[10] P. Marcon, J. Janousek, and R. Kadlec, "Vision-Based and Differential Global Positioning System to Ensure Precise Autonomous Landing of UAVs," *2018 Progress in*

Electromagnetics Research Symposium (PIERS-Toyama), pp. 542-546, 2018, doi: 10.23919/PIERS.2018.8598179. https://ieeexplore-ieee-org.ezproxy.rit.edu/document/8598179

[11] Z. Li, Y. Chen, H. Lu, H. Wu and L. Cheng, "UAV Autonomous Landing Technology Based on AprilTags Vision Positioning Algorithm," *2019 Chinese Control Conference (CCC), 2019, pp. 8148-8153,* doi: 10.23919/ChiCC.2019.8865757. https://ieeexplore-ieee-org.ezproxy.rit.edu/document/8865757

[12] Lin, S., Garratt, M.A. & Lambert, A.J. "Monocular vision-based real-time target recognition and tracking for autonomously landing an UAV in a cluttered shipboard environment". *Auton Robot 41, 881–901* (2017). https://doi.org/10.1007/s10514-016-9564-2

[13] Patruno, C., Nitti, M., Petitti, A. et al. "A Vision-Based Approach for Unmanned Aerial Vehicle Landing". *J Intell Robot Syst 95, 645–664* (2019). https://doi.org/10.1007/s10846-018-0933-2

[14] Fraczek P., Mora A., Kryjak T. (2018) Embedded Vision System for Automated Drone Landing Site Detection. *In: Chmielewski L., Kozera R., Orłowski A., Wojciechowski K., Bruckstein A., Petkov N. (eds) Computer Vision and Graphics*. ICCVG 2018. Lecture Notes in Computer Science, vol 11114. Springer, Cham. https://doi-org.ezproxy.rit.edu/10.1007/978-3-030-00692-1_35

[15] D. Safadinho et al, "UAV Landing Using Computer Vision Techniques for Human Detection," *Sensors, vol. 20, (3), pp. 613, 2020*. Available: https://ezproxy.rit.edu/login?url=https://www.proquest.com/scholarly-journals/uav-landing-using-computer-vision-techniques/docview/2550450574/se-2. DOI: http://dx.doi.org/10.3390/s20030613.

[16] Tzelepi, M., Tefas, A. "Human crowd detection for drone flight safety using convolutional neural networks". *In: 2017 25th European Signal Processing Conference (EUSIPCO), pp. 743–747*. IEEE (2017). https://ieeexplore-ieee-org.ezproxy.rit.edu/document/8657776

[17] Tzelepi, M., Tefas, A. "Graph embedded convolutional neural networks in human crowd detection for drone flight safety". IEEE Trans. Emerg. Top. Comput. Intell. (2019).

[18] Zhu, P., Wen, L., Bian, X., Ling, H., and Hu, Q., "Vision Meets Drones: A Challenge", *arXiv e-prints*, 2018. https://arxiv.org/abs/1804.07437

[19] Castellano G., Castiello C., Mencar C., Vessio G. "Crowd Detection for Drone Safe Landing Through Fully-Convolutional Neural Networks". *In: Chatzigeorgiou A. et al. (eds) SOFSEM 2020: Theory and Practice of Computer Science. SOFSEM 2020. Lecture Notes in Computer Science, vol 12011*. Springer, Cham. https://doi-org.ezproxy.rit.edu/10.1007/978-3-030-38919-2_25

[20] Harris, C.R., Millman, K.J., van der Walt, S.J. et al. Array programming with NumPy. Nature 585, 357–362 (2020). DOI: 10.1038/s41586-020-2649-2. (Publisher link).

[21] Bradski, G., 2000. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.

[22] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E.A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. (2020) **SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python**. *Nature Methods*, 17(3), 261-272.

[23] Paszke, A. et al., 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., pp. 8024–8035. Available at: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[24] MATLAB, 2010. *version 7.10.0 (R2010a)*, Natick, Massachusetts: The MathWorks Inc.

APPENDIX

Link for extraneous items

1. Logitech C270 Webcam
    a. https://www.amazon.com/Logitech-Desktop-Widescreen-Calling-Recording/dp/B004FHO5Y6/ref=asc_df_B004FHO5Y6/?tag=hyprod-20&linkCode=df0&hvadid=241890262974&hvpos=&hvnetw=g&hvrand=13663382025040501500&hvpone=&hvptwo=&hvqmt=&hvdev=c&hvdvcmdl=&hvlocint=&hvlocphy=9005653&hvtargid=pla-405660788128&psc=1
2. Nicolai Nielson’s GitHub Page
    a. https://github.com/niconielsen32/ComputerVision/tree/master/stereoVisionCalibration
3. Computer Setup used for testing:
    a. Processor:
        i. Intel(R) Core(TM) i9-10850K CPU @ 3.60GHz   3.60 GHz
    b. RAM:
        i. 32.0 GB
    c. System Type:
        i. 64-bit operating system, x64-based processor