2022-12-01

CAD-Based Aerial Trajectory Generation And 3D Mapping For Close-Quarter Inspection

Angel Guillermo Ortega Castillo
University of Texas at El Paso

CAD-BASED AERIAL TRAJECTORY GENERATION AND

3D MAPPING FOR CLOSE-QUARTER INSPECTION

ANGEL GUILLERMO ORTEGA CASTILLO

Doctoral Program in Mechanical Engineering

APPROVED:

Angel Flores-Abad, Chair, Ph.D.

Ahsan R. Choudhuri, Co-Chair, Ph.D.

Joel Quintana, Ph.D.

Virgilio Gonzalez, Ph.D.

Stephen L. Crites, Jr., Ph.D.
Dean of the Graduate School

"People think I am crazy.

OPPORTUNITY passes before us all,

but it is only visible to those who SEEK IT."

-JLCH

CAD-BASED AERIAL TRAJECTORY GENERATION AND 3D MAPPING FOR CLOSE-QUARTER INSPECTION

by

ANGEL GUILLERMO ORTEGA CASTILLO, B.Sc.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at El Paso

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

Department of Aerospace and Mechanical Engineering

THE UNIVERSITY OF TEXAS AT EL PASO

December 2022

Acknowledgements

I would like to express my deep-felt gratitude to my advisor, Dr. Angel Flores-Abad of the Mechanical Engineering Department at The University of Texas at El Paso, for his advice, encouragement, enduring patience, and constant support in this endeavor. He has paved roads for international students to aspire to advanced degrees.

I also wish to thank the other members of my committee, Dr. Ahsan Choudhuri and Dr. Joel Quintana of the Aerospace and Mechanical Engineering Department and Dr. Virgilio Gonzalez of the Electrical Engineering Department, both at The University of Texas at El Paso. Their suggestions, comments, and additional guidance were invaluable to the completion of this work.

Additionally, I want to thank The University of Texas at El Paso Aerospace and Mechanical Engineering Department professors and staff for all their hard work and dedication, providing me with the means to complete my degree and prepare me for a career as a mechanical engineer. This includes (but certainly is not limited to) the following individuals whom I consider my mentors:

> Dr. Jack Chessa, Dr. Yirong Lin, Dr. Methaq Abed, Dr. Louis Everett, Dr. Frank Medina, Dr. Cesar Terrazas, Dr. Norman Love.

I would also like to thank my team who have experienced this journey with me: Julio Reyes, Mousumi Rizia, and Noshin Habib. This study was supported by the US Department of Energy (DOE) Award # DE-FE0031655 and The University of Texas at El Paso – Aerospace Center. I would like to thank all the individuals from both institutions that allowed us to conduct this research work.

I would also like to thank Vinton Steel and El Paso Electric for allowing us access to your facilities to conduct our experiments. I would like to thank Jose Bustamante of El Paso Electric for believing in our project and climbing with us.

I must thank my parents and brothers for putting up with me during the development of this work with continuing, loving support. Each of you has made sacrifices that have allowed me to be here today. There are not enough words to express my love and gratitude. Papa, Mama, Ramses, Jose, Uriel, gracias!

To the rest of my family, thank you for guiding me and for all your love. I hope to make you proud.

NOTE: This thesis was submitted to my Supervising Committee on November 15, 2022.

Abstract

Robotic technologies for inspection purposes of large-scale structures have grown in interest. Such technologies are encouraged to reduce the risk in which human operators are involved and to reduce costs due to downtime of the equipment. In the Energy sector, high interest is placed on power plant components where their correct operation is paramount. This work is inspired by the synthetic vision systems for aerial vehicles that use three-dimensional space (3D) to provide pilots with clear and intuitive means of understanding their flying environment. This work can be separated into three main sections: Trajectory Generation from Computer-Aided Design (CAD) Models, Crack Detection using Convolutional Neural Networks (CNNs), and 3D Reconstruction. This work proposes a solution to complete flight missions in GPS-denied environments and a method of obtaining a reconstruction point cloud with segmented cracks.

Human inspectors are the most common way to inspect high-interest structures in the energy sector. That means that such inspectors must be trained to inspect specific structures but also must put themselves at risk to correctly inspect such structures. Some of the common hazards include very high altitudes and some environments can be detrimental to the health of the inspectors due to particulate byproducts from the structures. The inspectors then keep a manual log of their findings and relay that information to maintenance crews to fix or replace the damaged components. This type of inspection practice is prone to miss defects, incorrectly logging, and incorrectly locating defects found by the inspectors. Another approach in the industry is to use robotic systems to inspect like the use of UAVs or drones. Most of the current UAV technologies depend on a stable GPS signal to operate and complete their flight mission tasks. This is a big disadvantage in the inspection industry because some inspections must take place inside big structures that cause heavy signal interference. Also, most of the technologies available require a human pilot in command to be proficient at operating the UAV to prevent collisions and to cover

as much surface as possible. This leads to human error and can prove difficult to even the most experienced pilots to navigate the complex structures.

The objective of this work is to provide a platform to perform inspection tasks on high-interest structures within the energy sector. We propose a CAD-based aerial trajectory generation and 3D mapping platform for close-quarter inspections. With the use of CAD models of structures that are readily available, we can generate an offline trajectory that employs a wall offset and is capable of reaching virtually all exposed surfaces of the structures of interest with a minimum surface offset distance. The system also employs the use of Artificial Intelligence to detect, segment, and localize desired defects within the inspections. This eliminates human error in classifying and documenting the defects while maintaining a record of the defects. This data could then be used to map the environment with the discovered defects to better assess the level of damage. Finally, our system employs photogrammetry and point cloud reconstruction algorithms to accurately reconstruct the inspected environments. This could also be used in instances where an initial CAD model of the inspection structure is not available.

With this work, we hope to streamline the inspection procedures that employ robotic technologies to remove human inspectors from hazardous environments. By utilizing an autonomous UAV platform that does not employ GPS we hope to complete inspections in even the most demanding environments. The system would allow the inspectors to accurately view all the detected defects as soon as each flight mission is completed, thus allowing a more efficient maintenance plan for such plants.

Table of Contents

List of Figures

Chapter 1

Introduction

Power plants are critical components in the energy sector and their proper maintenance and operation are of vital importance. To do so, the industry must constantly inspect the critical components such as the boiler systems in fossil fuel power plants. The percentage of energy produced in the US by fossil fuels is 60.8 percent with 21.8% and 38.3% coming from coal and natural gas respectively. Because of these large percentiles, the correct maintenance of such installations is paramount for the energy supply of the country. The current maintenance approaches for such facilities involve human operators that must personally inspect the structures. A big issue of the current approaches is the need to shut down sections of the plant to inspect and repair for extended periods where the human operators must visually inspect and record their findings. This means that some defects may go unnoticed such as hairline fractures in piping. There are also hard-to-reach sections inside boilers that an inspector might not be able to reach easily and scaffolding must be installed just to reach such areas to inspect. Robotic technologies have also grown and are currently deployed to fulfill such needs. Such robotic technologies include the crawling type of robots that deploy on all surfaces of the structure and Unmanned Aerial Vehicles (UAVs) that must be deployed to visualize specific areas of the structures. However, the use of human inspectors and other robotic technologies like magnetic crawlers still requires a lot of time and planning. Human operators must have scaffolding installed to complete their inspections and they must face hazardous environments inside such structures. Crawlers will take a long time inspecting all the surfaces and must be tethered. Current UAV platforms require strong GPS signals as they are communication-dependent and an operator must maintain a line of sight at all times. Also, a major drawback of manually operating a UAV

in confined spaces is the high chance of crashing while not being able to get too close to structures due to perception errors. In this work, we propose the use of Unmanned Aerial Vehicles (UAVs) to inspect, classify, and map 3D environments.

1.1 Research Objectives

This research presents an improved and streamlined inspection method applicable to diverse infrastructures. The goal is to provide an intelligent autonomous inspection system with a real-time application. The research objectives are as follows:

1. Develop a method to perform a close-quarter trajectory for aerial inspection in GPS-denied areas using available CAD models
2. Create an AI-based framework to automatically detect structural defects
3. Integrate 3D-reconstructed maps using Photogrammetry and Point-Cloud-based methods

1.2 Scope

The use of UAVs allows the inspection to take place in hard-to-reach places where human operators might not be able to go and does so promptly without the use of external structures to be put in place. This work presents a trajectory generation method to perform the close-quarter inspection using readily available Computer Aided Design (CAD) models of structures. This will allow efficient area coverage that will minimize the needed flight time to complete the inspections while allowing the UAV to collect as much detail from structures as possible. Since the system is vision based, we do not need GPS which allows us to go to places where most other drone systems cannot go because of their dependability on GPS. This work also aims to create an AI-based framework to automatically detect structural defects through the use of Convolutional Neural Networks. Thus, removing the

need for a human operator to surf through thousands of images to identify such defects or physically identify such defects while in a hazardous environment. Finally, we aim to integrate 3D-reconstructed maps using Photogrammetry and Point-Cloud-based methods for structures where the CAD models might not be available. This work was validated in different experimental and industrial settings which include the inside of a furnace of a natural gas boiler at the Rio Grande Power Plant operated by El Paso Electric and the cSETR Fabens Facility Hangar where most of the initial experiments were conducted.

1.3 Methodology

This work provides a streamlined system to inspect virtually any structure within a GPS-denied environment. To do so, we can separate the contributions into three main sections: Trajectory Generation from CAD Models (Chapter 3), Automatic Crack Detection using Convolutional Neural Networks (Chapter 4), and 3D Reconstruction (Chapter 5). Chapter 6 includes real-life testing results in experimental and industrial environments of the previously mentioned contributions.

Usually, human operators must personally inspect the structures placing them in a very hazardous environment. Robotic technologies have proved beneficial to remove humans from hazardous environments while still allowing them to reach all corners of the structures they need to inspect. Artificial Intelligence has also allowed human operators to process data more efficiently. Convolutional Neural Networks have allowed the processing of large amounts of images to classify specific details within the images. We are currently interested in finding images with defects such as cracks. Different software and libraries have also made it possible to obtain 3D data from images or video feed along with depth data from specific sensors. This data can then be used to obtain a reconstruction of environments that have never been mapped before.

1. Trajectory Generation Unmanned Aerial Vehicles with this autonomous capability have the potential to minimize human interaction in hazardous environments and significantly reduce the time and cost requirements for such inspections. This system provides an alternative to expensive localization equipment by utilizing CAD models of the inspection structures which are for the most part readily available.

2. Defect Detection The use of Neural Networks will provide a way to easily classify and manipulate acquired data from inspections. This data can help identify the location and severity of such defects promptly. Such defects can also be highlighted by using image segmentation to allow the user to identify them. There is a possibility the image with highlighted defects can be used in reconstructions to highlight the defects in the 3D environments.

3. Reconstruction One major problem our system might face is the lack of an initial CAD model of the structure to be inspected. One way to solve this problem would be to generate a model following the traditional blueprints of the structure but this would be time-consuming and thus inefficient for a single inspection. One way to work around this issue would be to generate an initial inspection and use the initial map to build upon after each inspection. The initial map could provide sufficient data to generate a close-quarters trajectory and after each inspection, the trajectory could be computed again and again as more details are added to the 3D map.

Chapter 2

Literature Review

In the following paragraphs we can find relevant literature related to the robotic technologies developed for similar applications along with concepts related to Artificial Intelligence and 3D environment reconstructions.

Having the CAD model of the structure to inspect enables us to accurately place the UAV in all three dimensions not just latitude and longitude. This work includes documents presented and approved for publication [33, 38, 39, 37] and propose a novel method to generate offline flight trajectories for UAV's to inspect structural components in both indoor and outdoor environments. The robotics community has made advancements in recent years to prevent injury to the operators thus saving time and money. Different robotic technologies have been used to inspect these structures. Examples of robotic technology for inspection purposes include different types of robots and multiple UAV systems and have been studies since the 90's. UAVs, cable climbing robots, and monorail-type robots are some of the technologies developed by researchers. [42, 25, 35, 40, 7, 12, 24, 10, 45, 46].

The robotics field is involved with inspections of multiple structures in different industries. Such industries include, energy, oil and gas, and environmental. Yamamoto *et al* [47] developed a mono-rail type robot for inspection inside a nuclear power plant. Shukla *et al* [44] studied the then (2013) state of the art robotic technologies for inspection in the oil and gas industry. They mainly focus on the use of in-pipe inspection robots (IPIRs) and tank inspection robots (TIRs).

2.1 Trajectory Generation

In order to achieve autonomous inspection, a drone must be able to navigate in complex and intricate environments without the luxury of GPS signal many times. For GPS-denied environments, most of the research is based on hardware such as sensors and cameras used as the primary location system. Using two cameras, Nikolic *et al* [31], demonstrated a successful approach for UAV navigation inside a Coal-fired Power Plant Boiler. Their research tackled the particulate deflection problem successfully but requires mounting of two external cameras to the drone thus limiting the minimum payload requirement of the drone. The system relies heavily on the sensing capabilities of the hardware to avoid contact with the structure. They demonstrated some of the complexities of flying inside a power plant boiler, some of the same issues we would later face as well. Figures 2.1 and 2.2 display the structure where they tested their system along with their drone platform. They prove that vision based systems require state-of-the-art sensors and their performance is heavily dependent on the environment.

Shan *et al* [43] propose a system that compares the current image obtained by the UAV's camera to the Google Maps database to obtain the current position of the UAV. Although their system employs an extensive database to use virtually anywhere in the world, it faces the problem that the database is not updated constantly so it may encounter inconsistencies. Another problem with the approach for our purposes is the fact that aerial images do not contain much information in the third dimension of the structures. Structure elements such as overhangs are not visible and therefore limit the application of the system for inspection purposes.

Ferreira da Silva *et al* [16] used a Quadrotor UAV (Unmanned Aerial Vehicle) to perform aerial inspection of transmission power lines, where a Kalman Filter allowed to mitigate the problems that the power line electromagnetic fields cause on the on-board sensors. They mitigate the environment's negative effects in software rather than hardware.

Artificial Intelligence has also been employed to improve autonomy of UAVs [21]. In

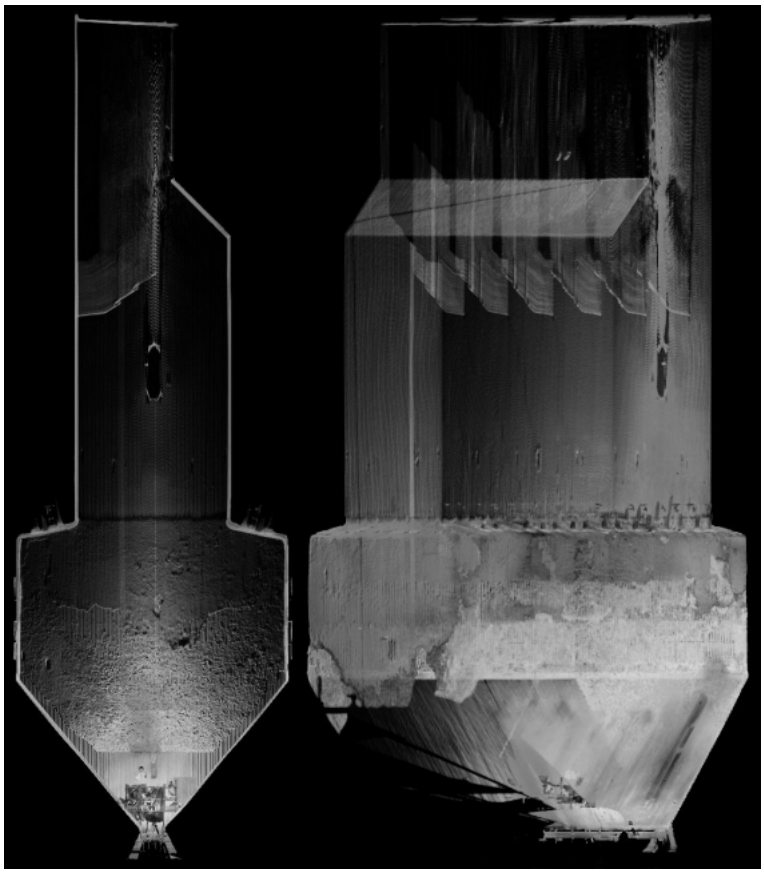


Figure 2.1: Laser Scan of Power Plant [31]

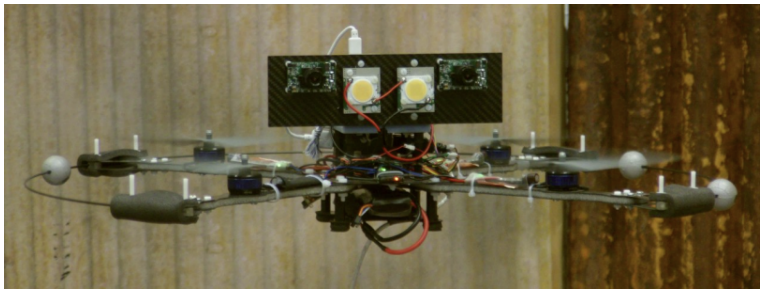


Figure 2.2: UAV Inspection Platform [31]

the area of defense, great advancements have been made with Nano Air Vehicles (NAV) as explained by Zhang *et al* [49], where they developed a Nano-scale quadrotor helicopter weighing less than 50 grams. Such research is supported by agencies such as the Defense Advanced Research Projects Agency (DARPA) for its great potential in the field. Their system uses on-board vision to calculate its current location. This poses a challenge since repetitive structure features can easily confuse the KLT optical flow technique used. To eliminate this problem, they used Parallel Tracking and Mapping (PTAM) algorithm, but it requires an initialization procedure to generate the required initial map.

The system proposed in this work tackles all those issues through the use of a CAD model of the structures in question. The process is based in an Additive Manufacturing concept called slicing as explained by Brown *et al* [9]. Since the trajectory is pre-programmed into the system, additional hardware is only based upon the type of inspection/mission the UAV will tackle. Close contact sensing can then be applied only as a fail-safe mechanism to prevent crashes due to structures not accounted for in the CAD model.

2.2 Defect Detection

Convolutional Neural Networks (CNN's) are commonly used for image classification and for defect detection and it has been very well documented. There are requirements needed to correctly train and operate a CNN. One of the most important is a good training dataset containing images with labels that closely resemble the actual objects that the network will have to identify. There is variation in the size of such datasets with some researchers claiming that the more images in the training dataset the better (20,000-40,000 images per category), while others argue that a more conservative number is enough (100- 200 images per category). For this work we have tested both approaches and found that it is easier to work with smaller dataset while still getting similar accuracy and results. Different studies contain approaches that combine CAD models and CNN's to obtain different results ranging from model classification to pixel comparison to existing road data [34, 6]. In our

stage of the study we have not combined both technologies for results but plan to do so in the future. This work is primarily based on crack detection in structures and the use of CNN's for this purpose is well documented [28, 26, 48, 11].

2.3 3D Reconstruction

2.3.1 Photogrammetry

The last part of our research will encompass the reconstruction of a 3D environment for visualization of the inspection. We use photogrammetry which is the process of obtaining reliable information about physical objects and/or environment through recording, measuring, and interpreting images and patterns. There is a wide range of software capable of doing all the internal calculations needed to produce a point cloud. We employ Alicevision's Meshroom since it allows us to have control over every step in the process rather than having no modification capability like other software. Reljic et al. compare some of those available meshing software options including Meshroom [36]. Since we are dealing with 3D reconstruction from 2D images, we employ a technique within photogrammetry technology called Structure-from-Motion (SfM), which is a technique for estimating three-dimensional structures from two-dimensional image sequences by identifying features [13, 23, 29, 15, 22]. Different researchers have used this technique but have also found drawbacks in the technology that result in distortion of the environments. Some of the proposed solutions include ground and aerial images for a better stitching and employing different camera angles at the same location to obtain more data in every location [50, 5]. We solve the first issue by having only aerial images that cover virtually all the features of the structure in detail and we plan on incorporating a second inspection camera to our system to improve the data acquisition.

2.3.2 SLAM

From a robotics standpoint, one of the most studied problems is Simultaneous Localization and Mapping (SLAM) [18, 8] which asks if it is possible for a robot to navigate its way through an unknown environment while being able to continuously build a map of such environment and be able to locate itself within that map. The tracking camera we employ, Intel RealSense T265, uses a SLAM algorithm internally to track the current location of the UAV through a combination of camera feeds and Inertial Measurement Units. One add-on we hope to employ using SLAM is to start with an initial map of the environment which we already have in the form of a CAD. We intend to use the SLAM algorithms to not only map but also locate itself within the existing environment through a system called ORB-SLAM2 [30]. ORB-SLAM2 allows the use of different sensor types including stereo and monocular. Ruan *et al* [41] compared the results between the two sensor types and found that with better computation capabilities, stereo results outperform monocular results. They also employ results from RGB-D sensor as a benchmark proving that the RGB-D mode yields the best results. This work helped us identify a suitable starting point for our research using ORB-SLAM2 and the different settings. Figure 2.3 displays their camera trajectory obtained by RGB-D camera. Although the systems are very capable and offer good results, it was impossible for us to get ORB-SLAM2 working correctly on the companion computer. We explain more on this in Chapter 5.

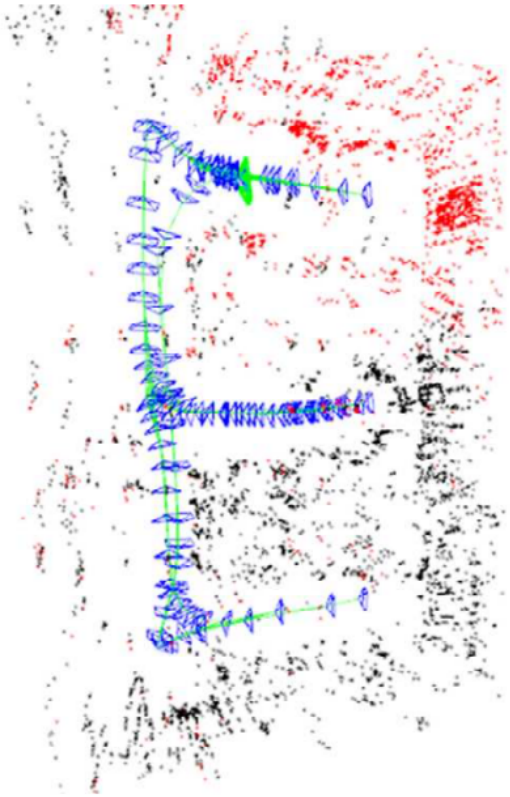


Figure 2.3: Camera trajectory obtained by RGB-D camera suing ORB-SLAM2 [41]

Chapter 3

Trajectory Generation from CAD Models

This chapter is based in its majority on an article presented in the AIAA SciTech Forum 2020, under the name Drone Inspection Flight Path Generation from 3D CAD Models: Power Plant Boiler Case Study. Some hardware changes have taken place but the majority of the software is still the same.

UAV systems allow access to inspection areas that would be hard to reach for other systems and even human operators. Because a flying system has virtually unlimited mobility, this is an idea that has great potential to substitute any current technology applied in the field. The only limitation to these systems is the payload capabilities where the inspection cameras and battery packs are the most critical components, but since hardware advancements are constantly being made, these systems will only get smaller as time advances. The other benefit is that in theory, these systems could fly with more than one UAV at a time thus reducing the battery pack requirements. While UAVs offer multiple benefits, they also carry some difficulties in the implementation such as their navigation and maneuverability. Conventional UAV systems heavily depend on a strong GPS signal to correctly navigate but we study a method for environments where a GPS signal might be too weak or simply non-existent. To be able to do so, a predefined trajectory must be programmed on the UAV to follow.

This system uses a CAD model to produce a trajectory using a MATLAB algorithm. The algorithm utilizes the slicer concept taken from the Additive Manufacturing (AM) field. Just like in AM, the CAD model must first be converted into a Stereolithography (STL) file

which contains only the surface information of a model. The information is derived into a triangular mesh that defines the surface geometry of the model. The STL file also contains the normal vector information for each triangle in the mesh. Once the STL file has been created, it is then arranged into matrices within MATLAB for its further manipulation where it is first "sliced" and then the data acquired is calculated to become a trajectory. In AM, the idea of the slicer irises from the need of defining the layer thickness for the AM technology being employed. For our purposes, we define the layer thickness not from material and extruder needs but from hardware parameters like camera view angle and overlap percentage needed to map the camera feed, we refer to it as slicer height. Keeping this in mind, the system can be easily calibrated to different values by changing a few lines of code. Once the data of the slices is acquired, it is then processed to become a trajectory that the UAV can follow. We are currently using a Raspberry Pi Camera Module v2 for image acquisition for the inspections. Based on the hardware properties we decided to employ a 300-mm (roughly 1-foot) wall offset and slicer height. The model is scaled to 1x104 m to prevent rounding errors within the MATLAB calculations and after the process is scaled back down to meters.

This section is separated into five individual processes as can be seen in Figure 3.1 and each will be explained in detail within their section. There are only three required inputs: Slicer Height Distance, Wall Offset Distance, and an arbitrary Layer Jump Location. The first two variables were explained in the previous section. The third variable is a location the operator must select as the starting point of the trajectory and that same x and y coordinate will be used as the jump location between each layer of the trajectory. This variable is arbitrary since each structure is different but any location the operator desires can be used for this variable. The system will generate two outputs; Yaw Data and Trajectory Coordinates both saved as comma-separated value (.csv) files.

1. CAD Model

2. Slicer Algorithm

3. Outline Algorithm & Wall Offset

4. Matrix Interconnection Algorithm & Trajectory Generation

5. Yaw Generation

6. Final Trajectory

3.1 CAD Model

Because of the advantages that CAD software has, it has now become a fundamental part of the design process in almost every field. That means that CAD models of structures are for the most part readily available. Most CAD packages allow the user to not only design components in a 3D interface but also be able to simulate its operating environment and detect possible flaws in the design way before the components are even manufactured. For that reason, a great amount of resources is destined by companies for the CAD models of components and that translates to models that are extremely accurate when compared to the components once they have been manufactured. This makes the CAD a perfect starting point for the trajectory generation needed for the UAV to complete its mission.

For this research, a simple CAD model was created following literature and online resources since access to power plant components is limited and so are their CAD models. Even though the model lacks detail, it is perfect for testing our algorithm and troubleshooting the code. Since we employ a slicer method in our algorithm, if more detail was provided in the CAD, it would only add more points of intersection per slice, producing an even better trajectory. It is important to note that CAD models exist in almost every engineering discipline including civil engineering meaning that our system is still a viable option for inspection not only on mechanical components but also structures in general.

The CAD model is critical to our method since it is used to limit the chances of collision of the UAV with any of the components. Figure 3.2 shows the CAD model we created and

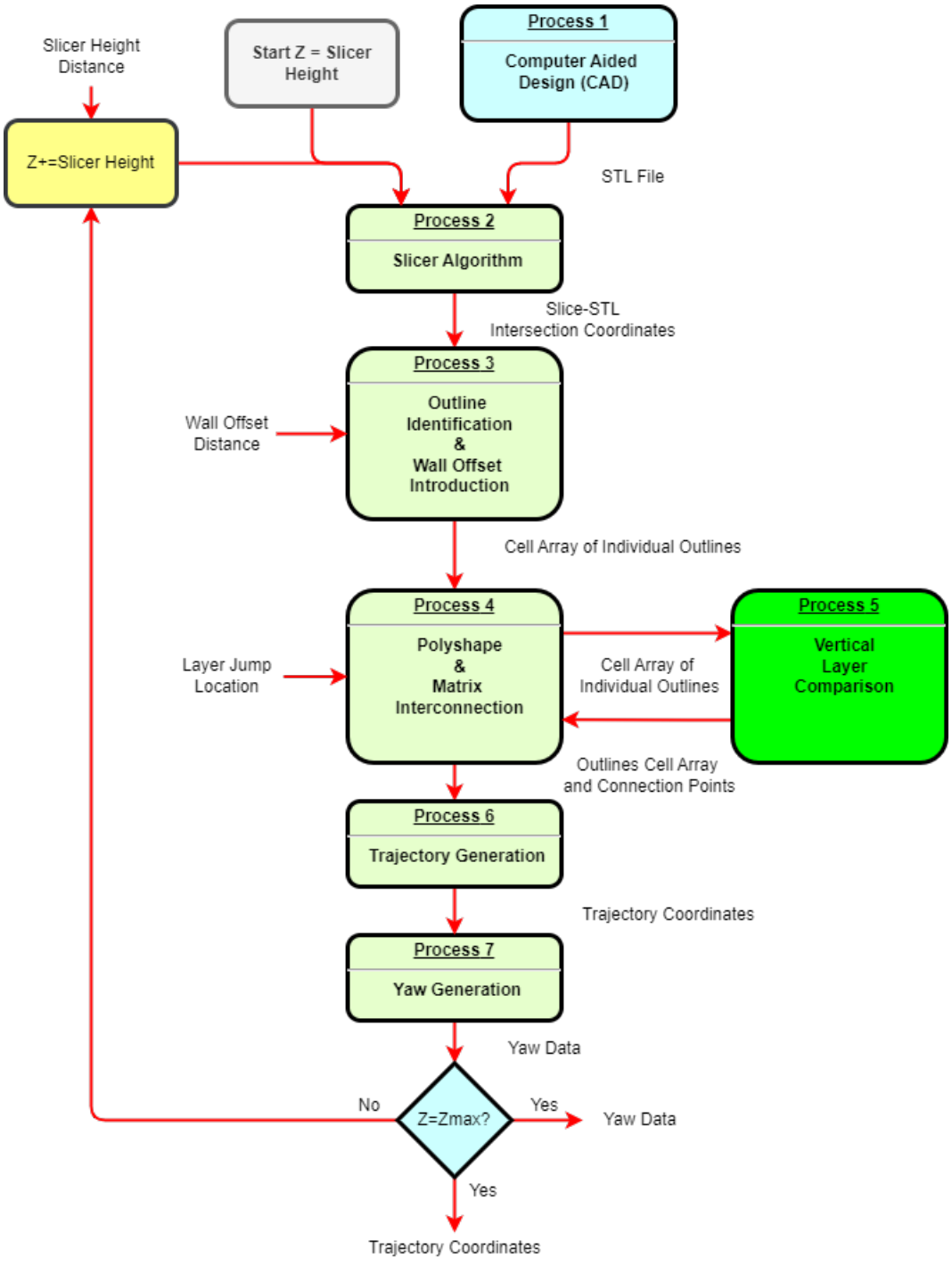


Figure 3.1: Flight Path Generation Algorithm

will be used in this study. The image also shows the different parts commonly found in a Power Plant Boiler. To use the CAD information in our system, the model must be converted into STL to define the surface coordinates. These surfaces are defined by a mesh made up of triangles and each triangular surface also contains a normal vector. Figure 3.3 depicts the STL format of the CAD model used in this effort.

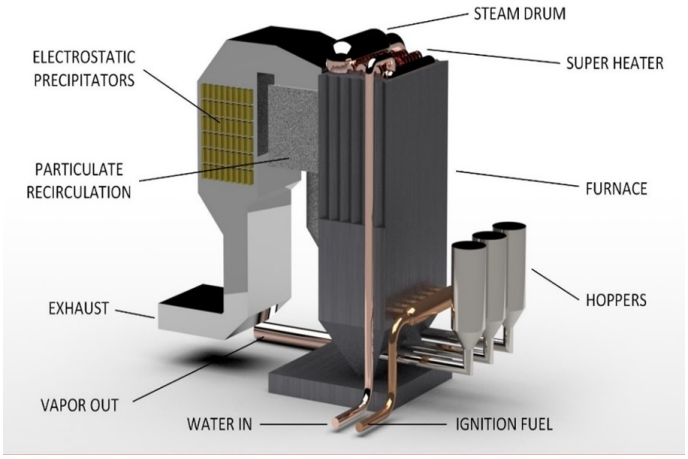


Figure 3.2: CAD Model of a Coal-Fired Power Plant

3.2 Slicer Algorithm

Since the STL file provides a triangular mesh defining the surface of the model, we create an imaginary x-y plane at a set height S_h in the z direction, and calculate the intersection points between this plane/slice and the triangular mesh. We produce a number of these planes which will eventually become the layers of our trajectory. Figure 3.4 depicts the interaction between these desired slices and the existing STL mesh. At this point, the triangle vertices are stored in matrix $\mathbf{T}$ with dimensions n x 9. The number of rows n is equal to the number of triangles present in the STL. Figure 3.5 depicts a single triangle with its coordinates. This provides the data for a single row in the following matrix. A sorting algorithm is used to order the z-values in ascending order. That means that the first vertex has the smallest z-value, and the last vertex has the highest z-value. This makes

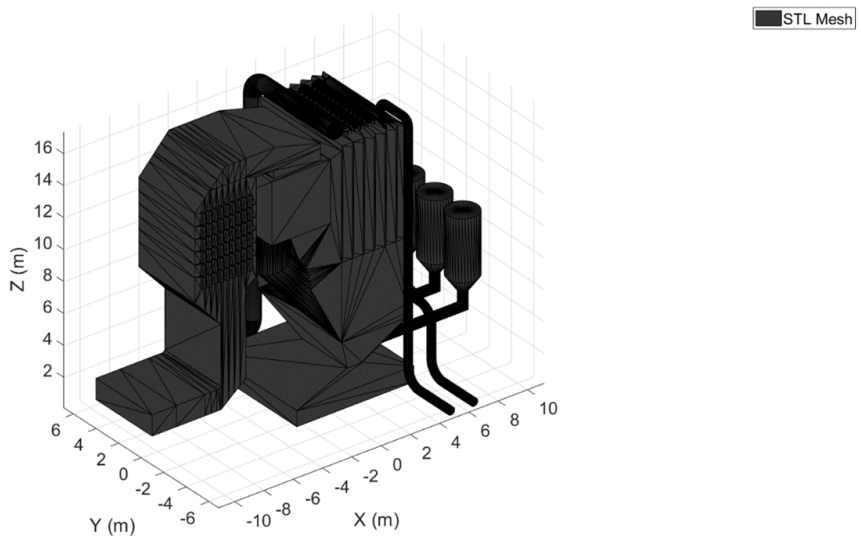


Figure 3.3: STL Mesh of CAD Model

the logic for the next process easier to implement.

$$\boldsymbol{T} = \begin{bmatrix} x_{I,1} & y_{I,1} & z_{I,1} & x_{II,1} & y_{II,1} & z_{II,1} & x_{III,1} & y_{III,1} & z_{III,1} \\ x_{I,2} & y_{I,2} & z_{I,2} & x_{II,2} & y_{II,2} & z_{II,2} & x_{III,2} & y_{III,2} & z_{III,2} \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ x_{I,n} & y_{I,n} & z_{I,n} & x_{II,n} & y_{II,n} & z_{II,n} & x_{III,n} & y_{III,n} & z_{III,n} \end{bmatrix} ; n = 1 : \text{Number of Triangles} \tag{3.1}$$

In this format, the first vertex is composed of the lowest z-value (z_I) and the x and y-values associated with that point, (x_I and y_I). The following vertex is composed of the next biggest z-value (z_{II}) and the corresponding x and y-coordinates (x_{II} and y_{II}). Finally, the last vertex is composed of the biggest z-value of the three vertices (z_{III}) and its corresponding x and y-values (x_{III} and y_{III}). The algorithm then assigns a "Case" number to the triangles depending on how they interact with the slice. The mathematical

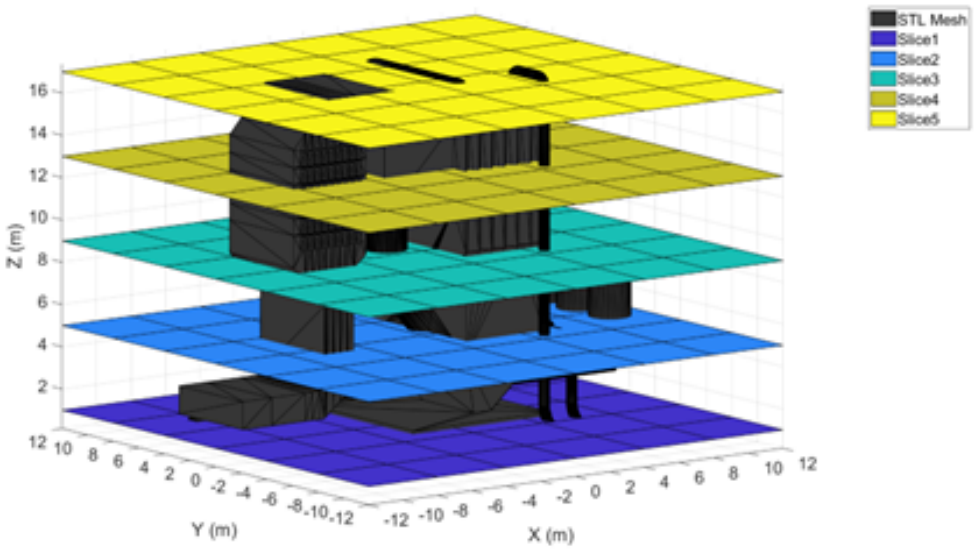


Figure 3.4: Slice - CAD Interaction

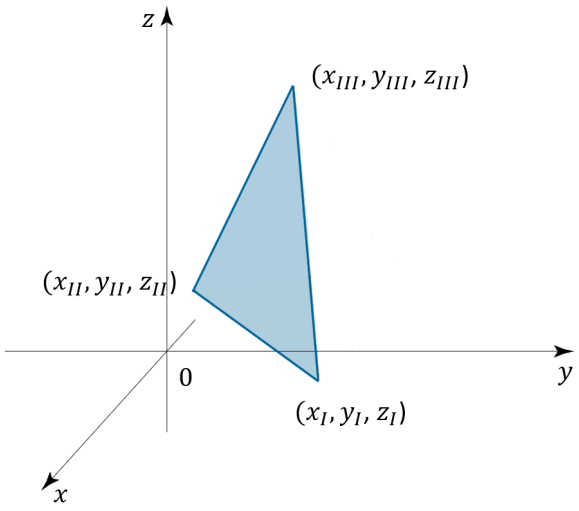


Figure 3.5: STL Mesh Coordinates

relationship of each case can be seen in the list below. This interaction can be seen in Figure 3.6.

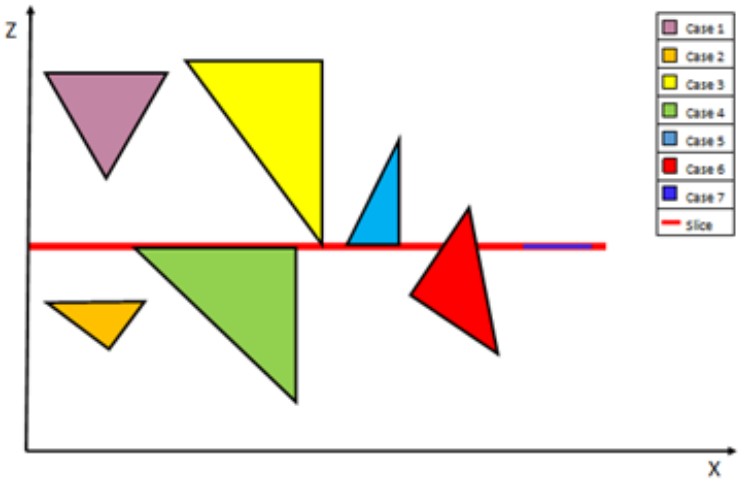


Figure 3.6: Slice - Triangle Interaction

1. $z_I > S_h$
2. $z_{III} < S_h$
3. $z_I = S_h$
4. $(z_{II} = S_h)$ & $(z_{III} = S_h)$
5. $(z_I = S_h)$ & $(z_{II} = S_h)$
6. $(z_I < S_h)$ & $(z_{III} > S_h)$
7. $(z_I = S_h)$ & $(z_{II} = S_h)$ & $(z_{III} = S_h)$

If the triangle is completely above or below the slice height (Case 1 & 2) they are not considered as they don't intersect with the slice. If all three corners of the triangle have a z-coordinate that is equal to the slice height (Case 7), then they are also not considered since they do not provide any valuable information to generate the outline of the body

from the slice intersection. The rest of the cases include triangles that intersect the slice at one or two points. From these, only the cases where the slice intersects the triangle at two points are considered since the other cases would only produce repeated points. From Figures 3.6 and 3.7, only Cases 4,5, and 6 are used to generate the intersection points between the slice and the structure based on the logic expressed earlier.

These points are important since they will define the outline of the body. Our slice algorithm is capable of identifying the boundary/boundaries of single or multiple areas on each slice. If the slice intersects at exactly two corners of the triangle, then the x and y values of those points are added to the outline matrix, but if the slicer intersects the triangle at any other location, an extra calculation is required to obtain the x and y coordinates. To identify those coordinates, linear interpolation is used.

$$x = \left(\frac{x_{II} - x_I}{z_{II} - z_I}\right)(S_h - z_I) + x_I \tag{3.2}$$

In this equation, the known values include the slice height (z), and the point coordinates (x1, x2, z1, and z2). These coordinates are taken directly from the previous step only for triangles that fall into the Case 6 category. This equation must be used to find both the x and y coordinates by substituting y's for all the x's in the equation. This will yield the x and y coordinates of the intersection point between the slice and a single side of the triangle. The process can then be repeated to find the other intersection point for the other line segments in the category. Both points would share the z coordinate with the slice.

The product of this part of the algorithm is a matrix containing all the slice intersection points with their corresponding x, y, and z-coordinates. This data represents the surface coordinates at the specific slice heights, but it does not yet recognize if the points make up a single boundary or multiple boundaries at that specific slice height. The next algorithm will separate the points into outline matrices, and it will introduce the desired wall offset distance to each of the boundaries.

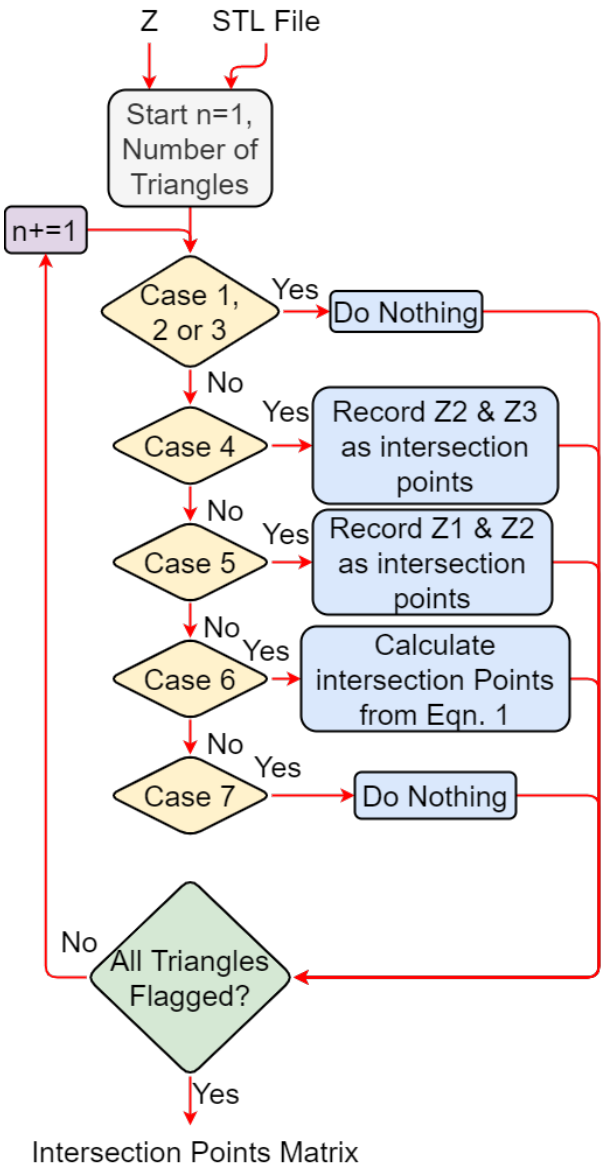


Figure 3.7: Slicer Algorithm

3.3 Outline Algorithm and Wall Offset

The data points obtained in the slicer are arranged in a single matrix of the total number of points by the 3 dimensions (x, y, z). This matrix includes multiple repeated points that will be eliminated in this section. Here, the number of points from the previous section will be cut in half, meaning that a CAD with extreme detail poses no issues for the algorithm and that this data is small enough to transfer to the UAV processor.

$$\boldsymbol{S} = \begin{bmatrix} P_{x,1} & P_{y,1} & P_{z,1} \\ P_{x,2} & P_{y,2} & P_{z,2} \\ . & . & . \\ . & . & . \\ . & . & . \\ P_{x,k} & P_{y,k} & P_{z,k} \end{bmatrix}; k = 1 : \text{Intersection Points @ } \mathrm{S}_h(3.3)$$

The data must first be sorted to generate the desired trajectory. If only a single outline is present, then the trajectory for that slice is simple as the data only needs to be sorted using the repeated values. If the data includes multiple outlines, the process to obtain the trajectory just requires some extra steps. The sorting process starts with the first coordinate in the slicer matrix which is arbitrary and takes the second coordinate as the connection. The algorithm then finds the index i of that repeated coordinate and identifies if the index is even or odd. If the index is odd, then the next connection would be the coordinate of i+1, otherwise, if the index is even, the next connection would be the coordinate of i-1. Once the coordinates are used, they are deleted from the original matrix. Since only the cases where the slice comes into contact with the triangles are used in the slicer algorithm, this logic works to find the repeated coordinates to sort the data into outlines. If the connecting coordinate is the same as the first coordinate that means that the outline has now closed. If there are remaining coordinates after a closed outline, the first index is again selected as the start of the outline, and the process is repeated. This algorithm will yield a cell

array containing the ordered coordinates that make up each outline area of the slice. These matrices have repeated coordinates, a unique command is used to remove the duplicates thus reducing the matrices' size by half. This process is depicted in Figure 3.8. Now that we have the coordinates for each outline, the wall offset distance is introduced as shown in Figure 3.9.

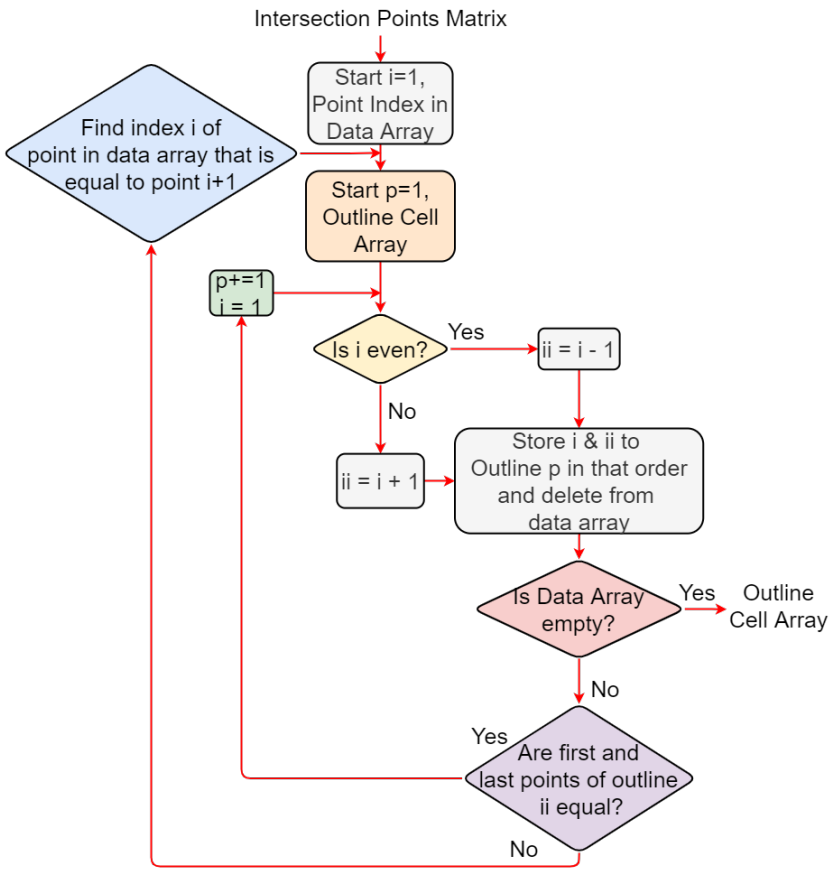


Figure 3.8: Outline Identification Algorithm

Before we introduce the wall offset algorithm, each outline coordinate matrix is converted into a polyshape object, a MATLAB Structure format. This is done so specific commands within MATLAB can be used. Once the data is in polyshape format, the buffer command is used to introduce the desired wall offset, which is a set distance we need from the wall to prevent collision and to be at the correct distance for image acquisition. The

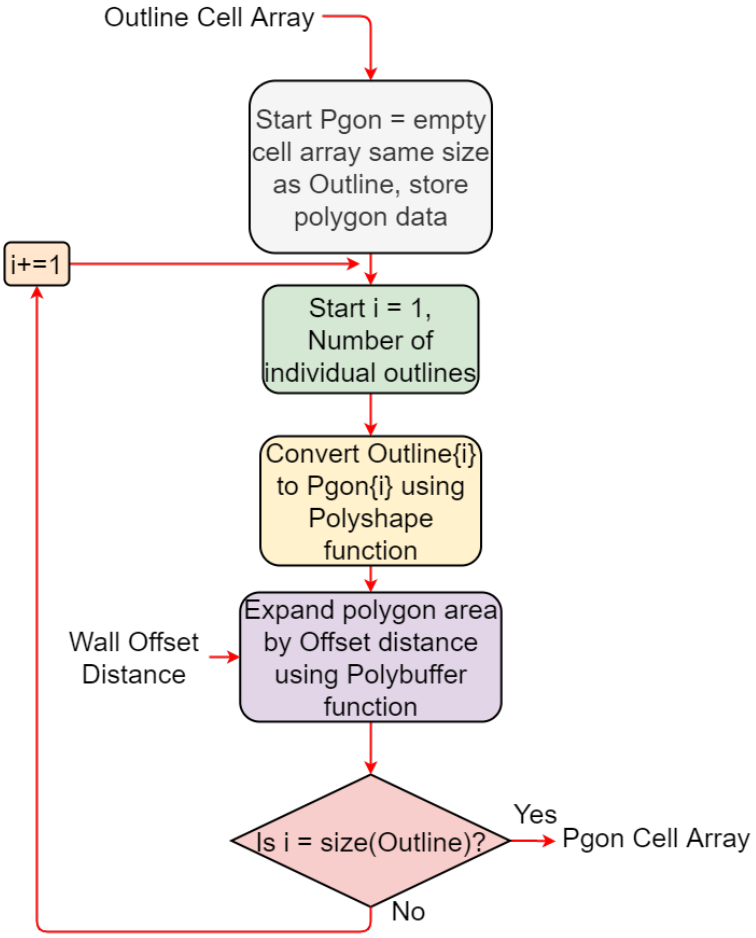


Figure 3.9: Wall Offset Introduction Algorithm

command used a circular area where the radius is the desired wall offset distance and simply traces that circle through the entire outline. The result is an outline that now has the desired offset, but it also rounds off the convex corners similar to what a spline algorithm would do. This is important since the accelerations at those corners are reduced thus generating a better trajectory for the drone to follow. The next planned improvement to the algorithm would be to introduce a spline algorithm for the concave corners to reduce the accelerations required for the drone to complete that corner. This is required since the buffer command only rounds off the convex corners but not the concave ones. Depending on the type of inspection, the wall offset will either move the trajectory inward or outward for internal or external inspection accordingly. For this study, we are focusing only on external inspection.

3.4 Polyshape and Matrix Interconnection

After the wall offset is introduced, the resulting outlines must be checked to see if any interconnection between outlines has taken place. The Polyshape outlines are stored in a cell array containing all the outlines for that layer. Because we are focusing on an external inspection, only three modes of interaction between the outlines may exist and they can be seen in Figure 3.10.

Mode 1 is an outline that fits inside another outline without any intersection (Bodies 1 & 2). For this mode, only the outside outline is taken and the other is eliminated from the trajectory. Mode 2 contains two or more outlines that do not intersect (Bodies 1 & 4). For this mode, the outlines are not modified. Mode 3 consists of two outlines that intersect (Bodies 1 & 3). This means that while the slicer did not encounter any intersections, the wall offset introduction has expanded two outlines to the point where they are now in contact with each other. This means that there is not enough space for the UAV to safely travel between the two outlines. Therefore, only the outside sections of the outlines are used for the trajectory while the sections that intersect are discarded. This type of

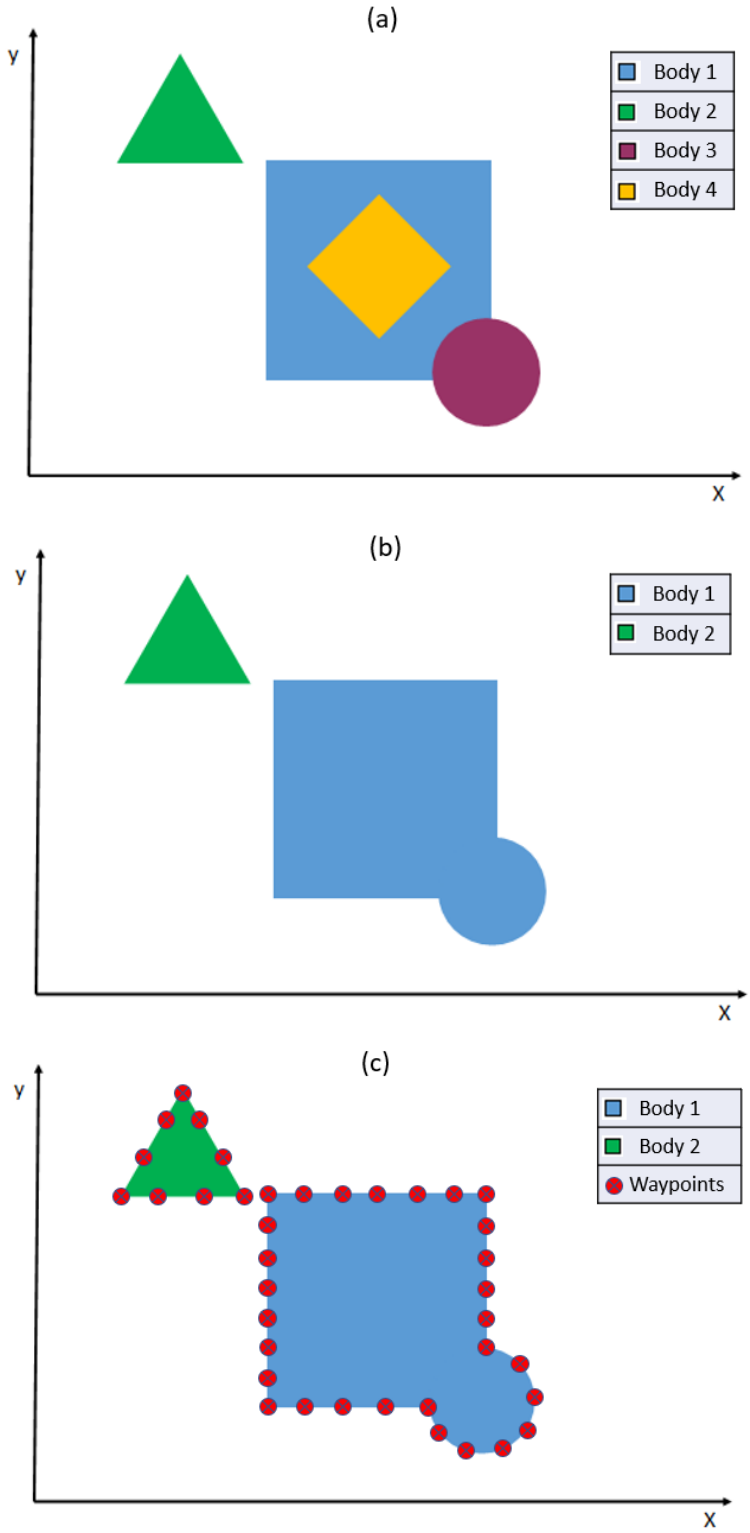


Figure 3.10: Outline Combination

intersection is commonly found in overhang sections but is not common for the outlines to cross if the slicer did not detect it first. The second part of Figure 3.10 depicts the result of these outline combinations. This product is achieved by using a MATLAB command called Union which is used to combine two Polyshape objects. We can also modify this procedure for internal inspections by substituting the Union command with Intersect which produces the internal boundary of the intersection.

3.5 Vertical Layer Comparison

When we tested the original boiler CAD model we created of the power plant we soon realized that the overhang angles were accounted for in the XY plane but not in the Z direction. We also implemented a method to prevent collisions due to such overhangs and tight spaces above and below the UAV's current altitude, we call it a Vertical Layer Comparison. To correctly identify such features a couple of layers are constructed above and below each main layer. The number of layers and the spacing between them is completely dependent on the user and the drone dimensions. It is a simple process that repeats the Outline Combination algorithm but this time instead of doing so for a single main layer it does it for the comparison layers. The algorithm defined the different outlines for all the separate bodies present in the layers to be compared. It then combines all the outlines to achieve the best possible outline for all the layers. If the inspection mode is set to external inspection, then the outlines with the biggest area will take precedence and the overall geometry to be inspected for that layer height would be the larger resulting geometry. If the inspection mode is set to internal, then the opposite occurs, where the smaller outline takes precedence resulting in the inspection taking place around the smaller outlines at the layer height. This produces a safe outline that prevents any type of collision due to overhangs. For this, we created two variables called Comparison Layer Height and Comparison Layer Number. Each of those helps identify the height at which a comparison layer will be calculated and compared to the main inspection layer. The number of layers and the

spacing between them is completely dependent on the user and the drone dimensions. Our values are based on our UAV which measures 30 cm in height. We then assign a value of 50 cm to account for any error in the UAV's flight controller and estimation and we assign a value of 5 to our Comparison Layer Number. This means that the algorithm will compare two layers below and two layers above the Current Inspection Layer and they will vary by 10 cm in height, roughly double the height of the UAV.

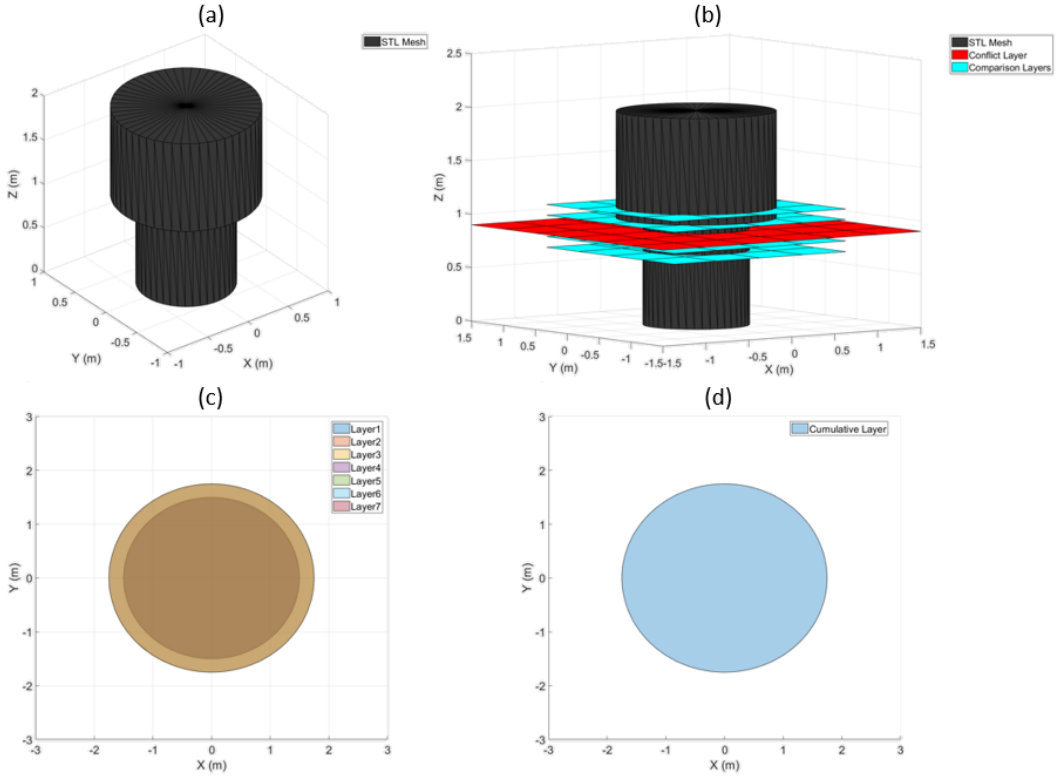


Figure 3.11: Vertical Layer Comparison: (a) Model with overhang. (b) Model with Conflict Layer and Comparison Layers. (c) Outlines of Conflict Layer and Comparison Layers. (d) Resulting Cumulative Layer after combination of outlines.

This then takes into account any other features not present at that specific height but features that could become obstacles for the UAV and a possible collision. Figure figlay depicts this process in a generic structure with clear overhangs. The values for the variables could be changed depending on the UAV and also the level of detail in the comparison, i.e.

make the height lower and the number of comparisons higher to cover more volume. At the end of this process, the data is still stored as matrices in a cell array that will then be used to generate a trajectory.

3.6 Trajectory Generation

A layer jump location must be selected for the next process in the algorithm to take place. It is encouraged to select an area where the jump from layer to layer can be done safely e.g. a place free of overhangs and other components. This jump location is arbitrary but can be adjusted for each inspection.

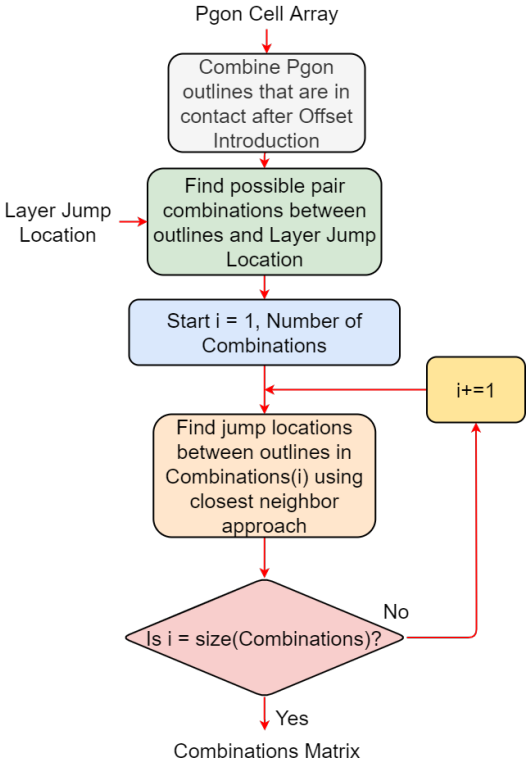


Figure 3.12: Matrix Interconnection Algorithm

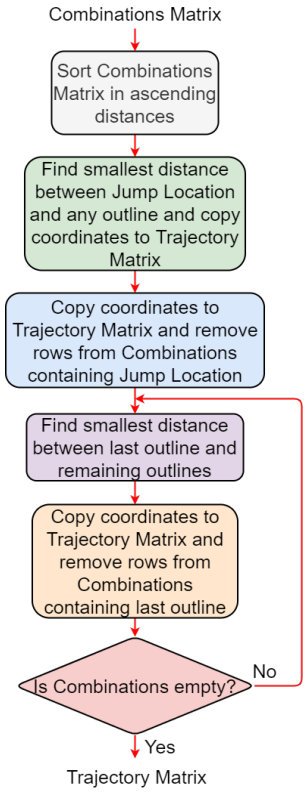


Figure 3.13: Wall Offset Introduction Algorithm

At this point, the data is divided into multiple matrices that would then have to be

combined in a single matrix to be considered a trajectory. The way that data is combined has been researched to obtain results that can be applied to any outline regardless of how complex the shapes are. The process was achieved through a closest-neighbor approach where the closest points between matrices were found and added to a matrix called a Combination Matrix, seen in Figure matrix. This matrix was then used to generate a path using those points to transition from outline to outline. This process can be seen in Figure 3.13. The only remaining process is to generate the trajectory by starting at the layer jump location, going to the closest neighbor in another matrix, and copying the points up until the next closest neighbor is located. At this point, it would repeat the transition to the other boundary and the process repeats until all the points have been read and copied achieving the final trajectory for that particular slice. MATLAB has a boundary algorithm, but it removes a lot of points in the trajectory meaning that the UAV wouldn't fly to places of interest in the structure. A comparison between our boundary algorithm and the MATLAB boundary algorithm can be seen in Figure 3.14.

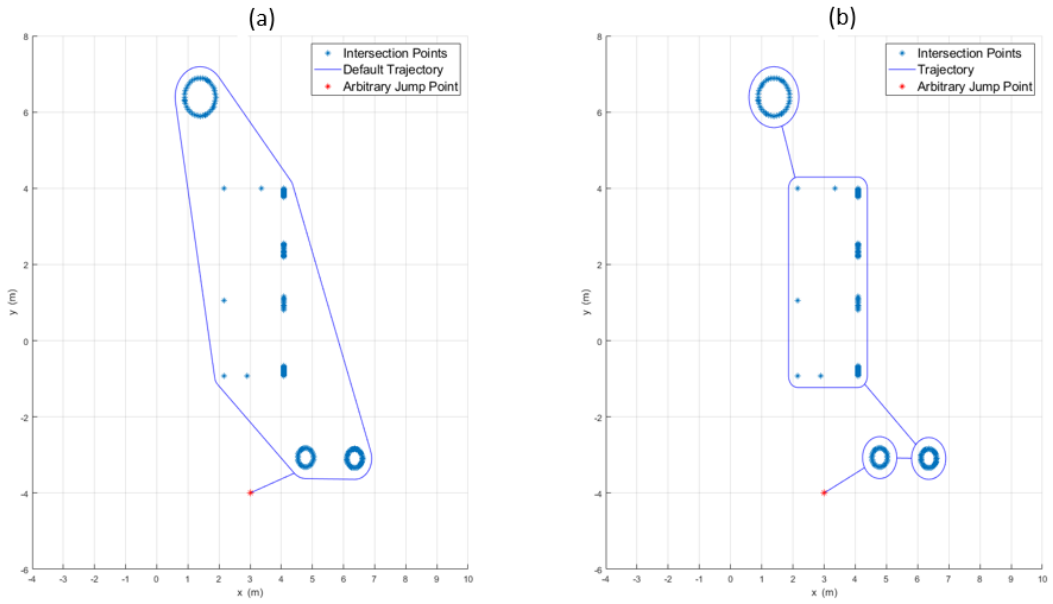


Figure 3.14: a) Trajectory generated using MATLAB's default boundary algorithm b) Trajectory generated using our boundary algorithm.

After the data has been correctly sorted, it can be called a trajectory. The trajectory obtained in this work is extremely similar to G-Code which is often used in other robotic systems like Additive Manufacturing (AM) and Computer Numerical Control (CNC) machines. Figure 3.15 depicts the process that takes place with different algorithms for each step. Once all the processes have taken place, the algorithm loops over to the next slice height until all the layers are completed. Since each layer start and ends at the arbitrary jump location, the jump from each layer height happens at exactly the same x and y-coordinates.

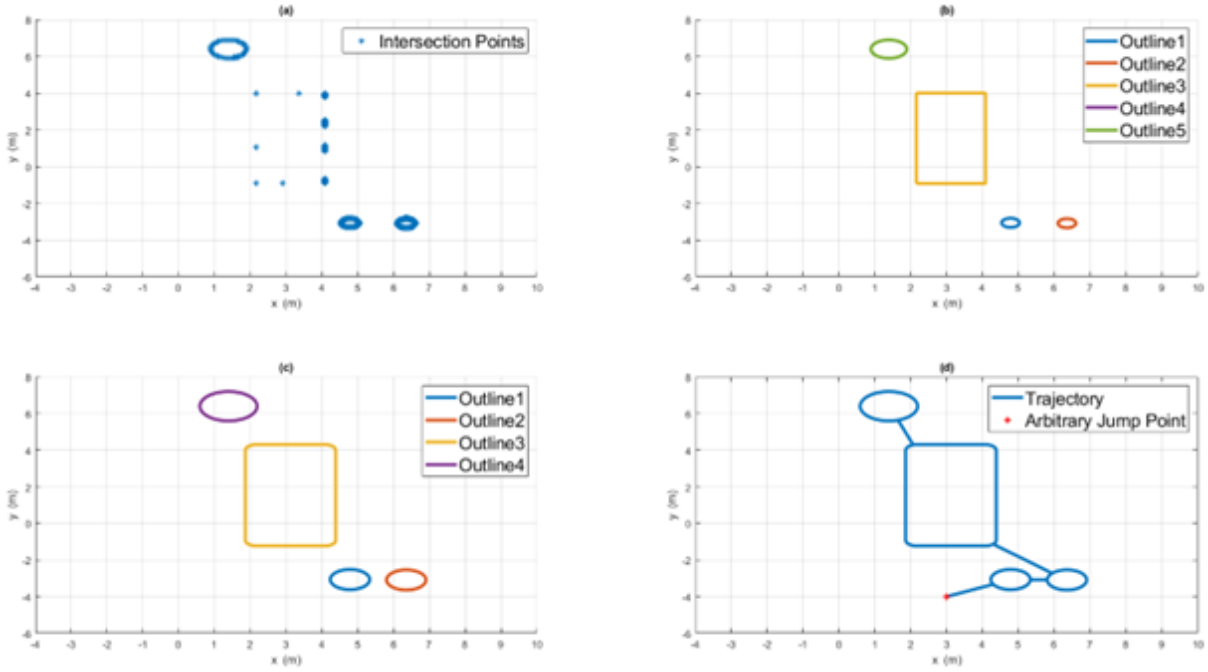


Figure 3.15: Trajectory Algorithm Process using arbitrary slice: (a) Points from Slicer Algorithm. (b) Outlines generated from Outline Algorithm. (c) Wall Offset and Outline Intersection Algorithm. (d) Final Trajectory with Jump Location.

3.7 Yaw Generation for Attitude Control

The system uses a quadcopter as the UAV. This type of aircraft was chosen for its stability and compact size. Another advantage of using a quadcopter is the ability to maneuver with a roll and pitch that is the same. For that reason, the only degree of freedom that requires calculation based on the trajectory points is the yaw. Since we are interested in inspection using the UAV, the attitude of the UAV is important. The system is equipped with EOS and infrared cameras that can be used both for data acquisition and obstacle detection. The yaw was obtained by calculating the normal vectors of the trajectory by using the following formula.

$$\theta = \text{atan2}(y, x) \tag{3.4}$$

The 2-argument arctangent function is defined as the angle between the positive x-axis and the vector (x, y). The function describes the angle in the Euclidean plane and converts the cartesian coordinates (x, y) to polar coordinates (r, θ). Theta then becomes the angle between the target point and the x-axis in the global spectrum, also known as yaw. This function is applied in a loop for each consecutive target point where (x, y) is the difference between the target and the current points. Those yaw values are then stored in a new column for the trajectory matrix. Resulting in a matrix of n by 4; x, y, z-coordinates, and yaw for that specific point given in radians.

3.8 Final Trajectory

A trajectory is successfully achieved from the STL file of the CAD model. Figure 3.16 depicts the final obtained trajectory, and the STL of the Boiler CAD is also provided as a reference. We can see how the wall offset and slice height selected translates to the trajectory generated. A comparison between Figure 3.16 and 3.17 demonstrates the added detail of our algorithm vs a simpler approach using only default commands within the MATLAB

environment. The extra steps yield a trajectory that provides access to most surfaces thus providing a more detailed inspection. There is a reasonable difference in distance between the two Results. The MATLAB Default Outline generations yield a total distance of 1,370 meters of trajectory for the entire structure while our system yields a trajectory of 2,515 meters. The difference of over 1 kilometer of trajectory directly translates to a trajectory that covers much more surfaces allowing more detail in the inspection mission. Future work for this section includes the implementation of more complex CAD models and simultaneous trajectory generation for internal and external inspection. Current work includes the implementation of deep neural networks for image processing and automatic flaw detection along with an improved real-time localization system based on the CAD model for GPS-denied environments.

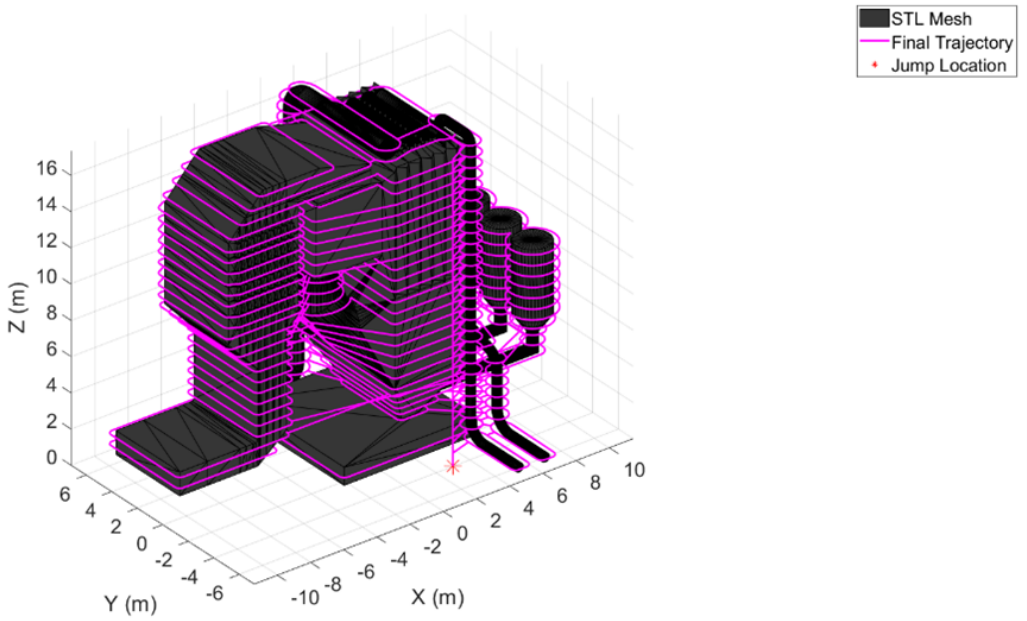


Figure 3.16: Final Trajectory generated using our method with STL present

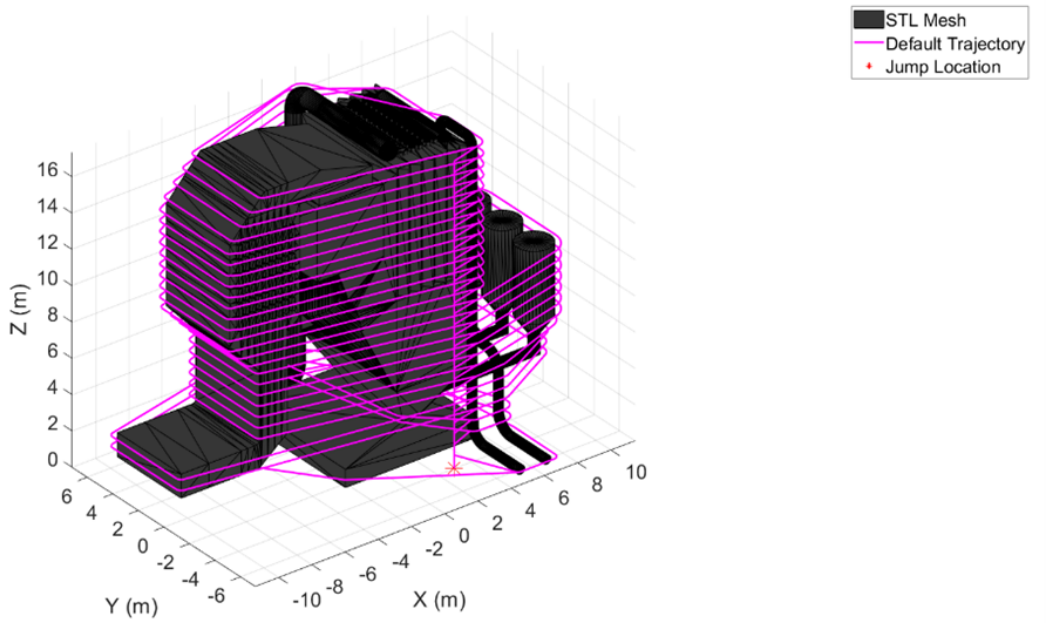


Figure 3.17: Final Trajectory generated using MATLAB's default boundary algorithm

Chapter 4

Crack Detection using Convolutional Neural Networks

In this chapter, we will identify the use of Convolutional Neural Networks (CNNs)for automatic crack detection and the results of the technology. The defect detection system is composed of four sections that will be explained in detail within their section. MATLAB's AlexNet CNN will be used to process images and automatically identify if a crack is present. Although AlexNet already has multiple datasets built in, we must create a new database containing images of cracks in concrete or download an existing database. For this purpose, we decided to use a dataset containing 40,000 images of concrete with and without cracks which is available at https://data.mendeley.com/datasets/5y9wdsg2zt/2.

1. Training Dataset Pre-Processing
2. Neural Network Training
3. Neural Network Validation
4. Neural Network Testing & Image Sectioning
5. Image Segmentation

4.1 Training Dataset Pre-Processing

Once the original dataset is downloaded, it is reviewed to prevent any false classifications. Although this might take some time it is important to do since these images and their

label will be used to train, test, and validate the network. The dataset already contains 40,000 images, 20,000 are true and 20,000 are false for cracks in concrete, but we decided to test augmenting the dataset to expand this number. The purpose of this is to attempt to generate copies of the same dataset with different angles, colors, and brightness to simulate possible environmental conditions. MATLAB already has multiple algorithms that allow to crop, mirror, and shift images but we created an algorithm to recolor the RGB images with three different scales. Since the original images are already clear, two of the scales are below 1 (0.5 & 0.7) and the third is above 1 (1.5). The following equation is used:

$$I = I * s \tag{4.1}$$

In this case, the RBG image I is in the form of a 227x227x3 matrix. The scale simply increases or decreases each pixel value. If the scale s is above 1 it will yield a brighter picture, but if the scale is below 1 then it will yield a darker picture. We believe darker-than-usual environments are more prone to happen than brighter-than-usual ones. Figure 4.1 depicts these recolor results. At this point, we also divide the dataset into three sections, 80% of the images are for training, 10% are for testing, and 10% are for validating.

4.2 Neural Network Training

CNN's work in a layer system. In this work, we worked with MATLAB's AlexNet CNN. The total number of layers in AlexNet is 25, 5 convolutional layers and 3 fully connected layers, making it a Deep Neural Network. Deep learning neural networks are trained using the stochastic gradient descent algorithm. The algorithm requires two variables to be set, the initial learning rate and the momentum. The initial learning rate is the size of each step in the gradient descent while the momentum is the angle difference at each step taken towards the goal. Setting these two variables correctly is important for good training of the network. If the learning rate is too small, there is a chance the network will never reach the level of accuracy that is expected. If the initial learning rate is too big, there will be

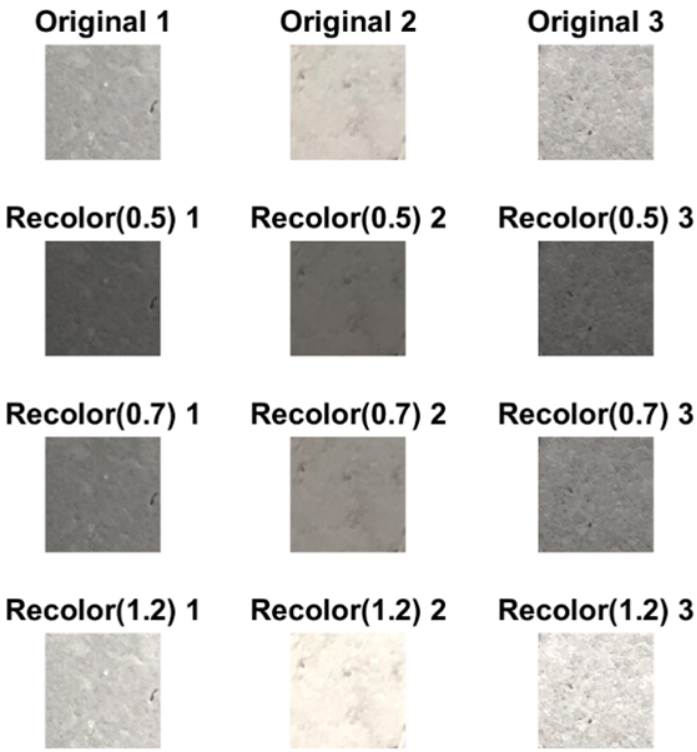


Figure 4.1: MATLAB Dataset Recolor

an overshoot which can translate to the network getting close to its target but never really hitting it. Some systems will run a series of training to automatically find the best values for both variables. In AlexNet this is achieved through trial and error. We had to train the networks following the logic of what a good network would have to look like and adjust our parameters for the next training. Figure 4.2 depicts these behaviors in training.

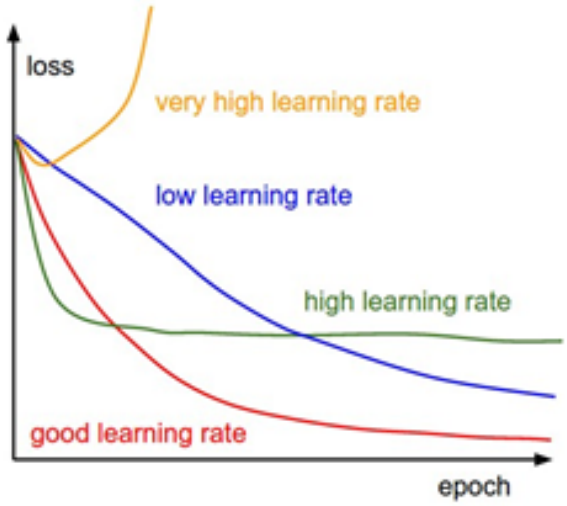


Figure 4.2: CNN Training Behaviors

The biggest difference in training with either small datasets or large dataset is the amount of time required for training. Our first training sessions were done using 20,000 images for crack labeling and 20,000 images of concrete without cracks. Training time for a network with this number of images in the dataset ranged from 50 to 80 minutes. Figure 4.3 depicts a network trained with this size dataset. We also trained a network with 80,000 images for cracks and 80,000 images of concrete without cracks. The training time for those CNN's was roughly 4.5 hours. Figure 4.4 depicts a network trained with this size dataset. Lastly, we trained another series of networks with just 100 images for cracks and 100 images for concrete without cracks. The training times for these series of training range from 5-15 seconds. Figure 4.5 depicts a network trained with this size dataset. Clearly, there is a time advantage in using smaller datasets for training, especially with the fact that multiple training sessions must take place to tune the network to correct initial learning rate and

momentum values. We also started to modify the epoch size and batch size to further tune the network.

The most important variable to set in a gradient descent learning algorithm is the initial learning rate (ILR). For this purpose, we first trained three networks using varying initial learning rates by a factor of 10 (0.1, 0.01, 0.001). Figures 4.6,4.7, and 4.8 depict the learning charts of those three training sets. As shown in Figure 4, the best initial learning rate out of the three is 0.001 since it is the only one that reaches 100% accuracy. The ILR simply depicts the size of the step the model takes while trying to achieve the right classification. The first two ILRs behave poorly. They are most likely overshooting because they are too big.

4.3 Neural Network Validation

As explained previously, the network with the best ILR is 0.001, for that reason we only use the results of that network. The other images that were previously separated for testing and validating will now be used to run the CNN and compare the prediction to the actual label. Figure 4.9 represents a confidence chart with these results. As it can be seen, only 7 images out of 8,000 (2,000 True/False, for Test and Validation each) were labeled wrong while the remainder were correctly identified, 99.95% and 99.88% accuracy respectively.

4.4 Testing Environment and Image Sectioning

After the training has taken place, we created a testing environment consisting of a concrete driveway with areas with and without cracks. This environment is used as a test bed to not only test the trained CNN but also to test the drone's capabilities of capturing images with enough quality to be processed through the network. Figure 4.10 depicts the environment used for the initial tests.

This environment was printed on a poster of size 36 x 48 feet. Consequently, the drone

Figure 4.3: Training Progress with 40,000 Images

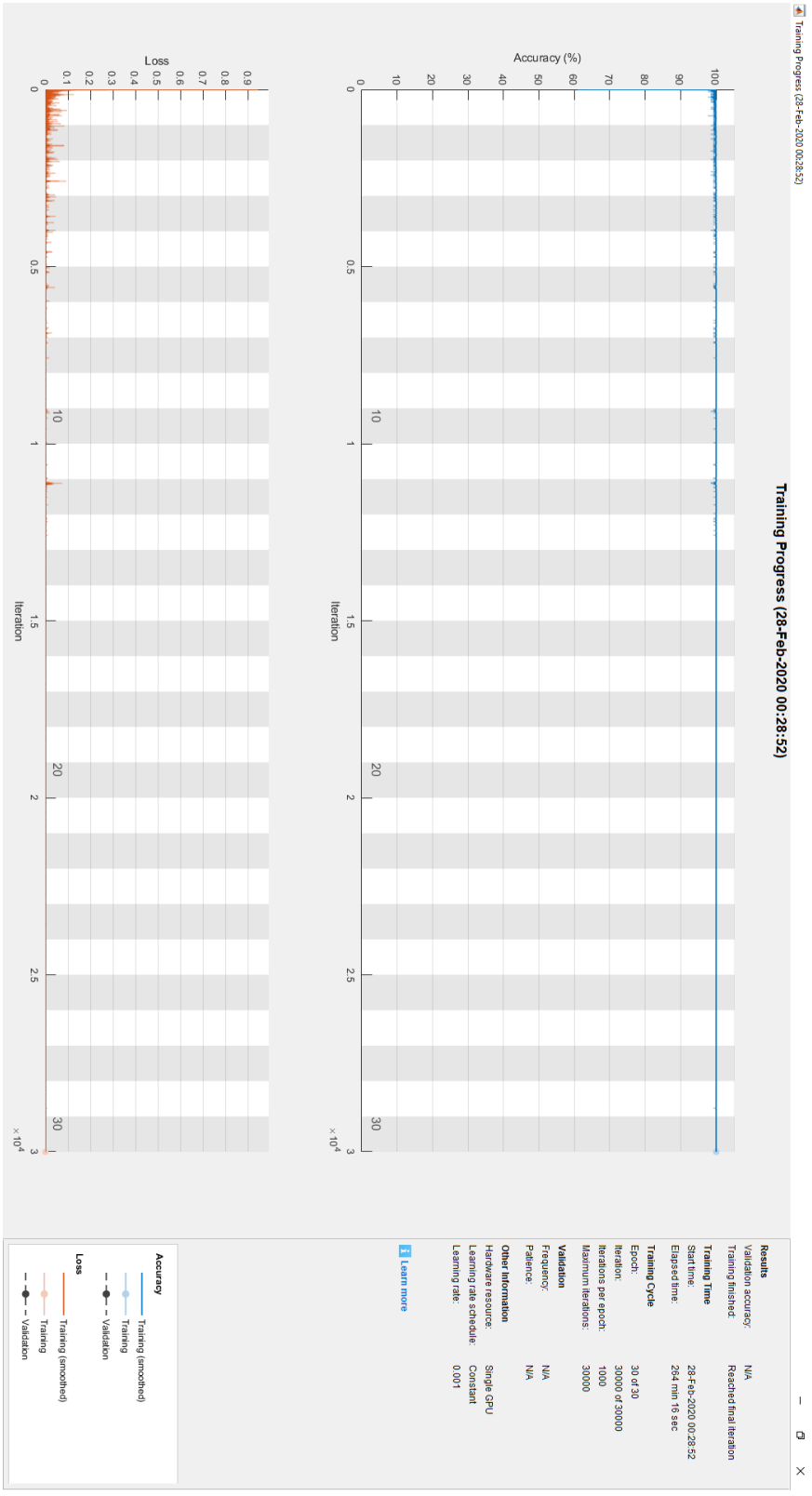


Figure 4.4: Training Progress with 160,000 Images

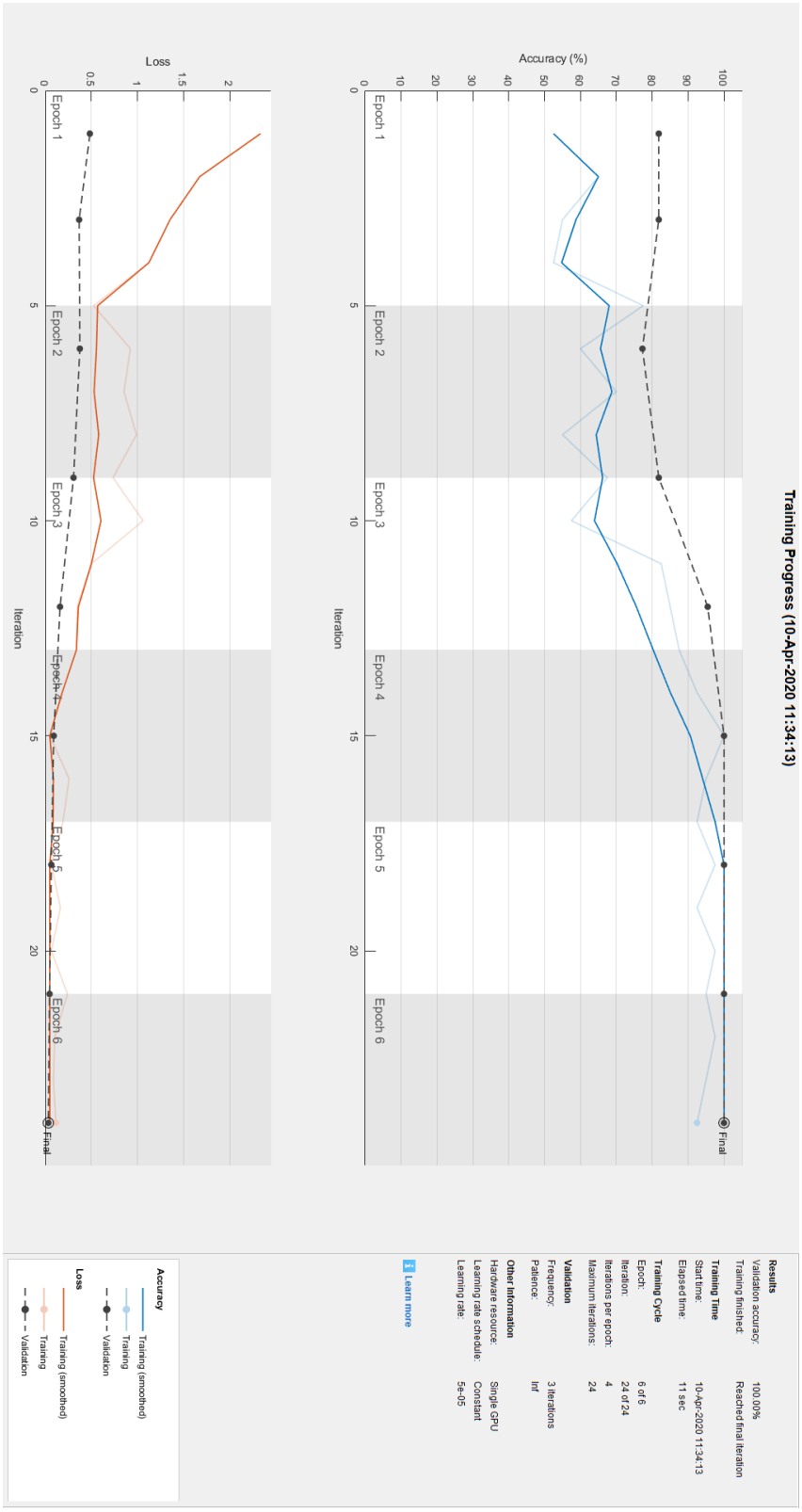


Figure 4.5: Training Progress with 200 Images

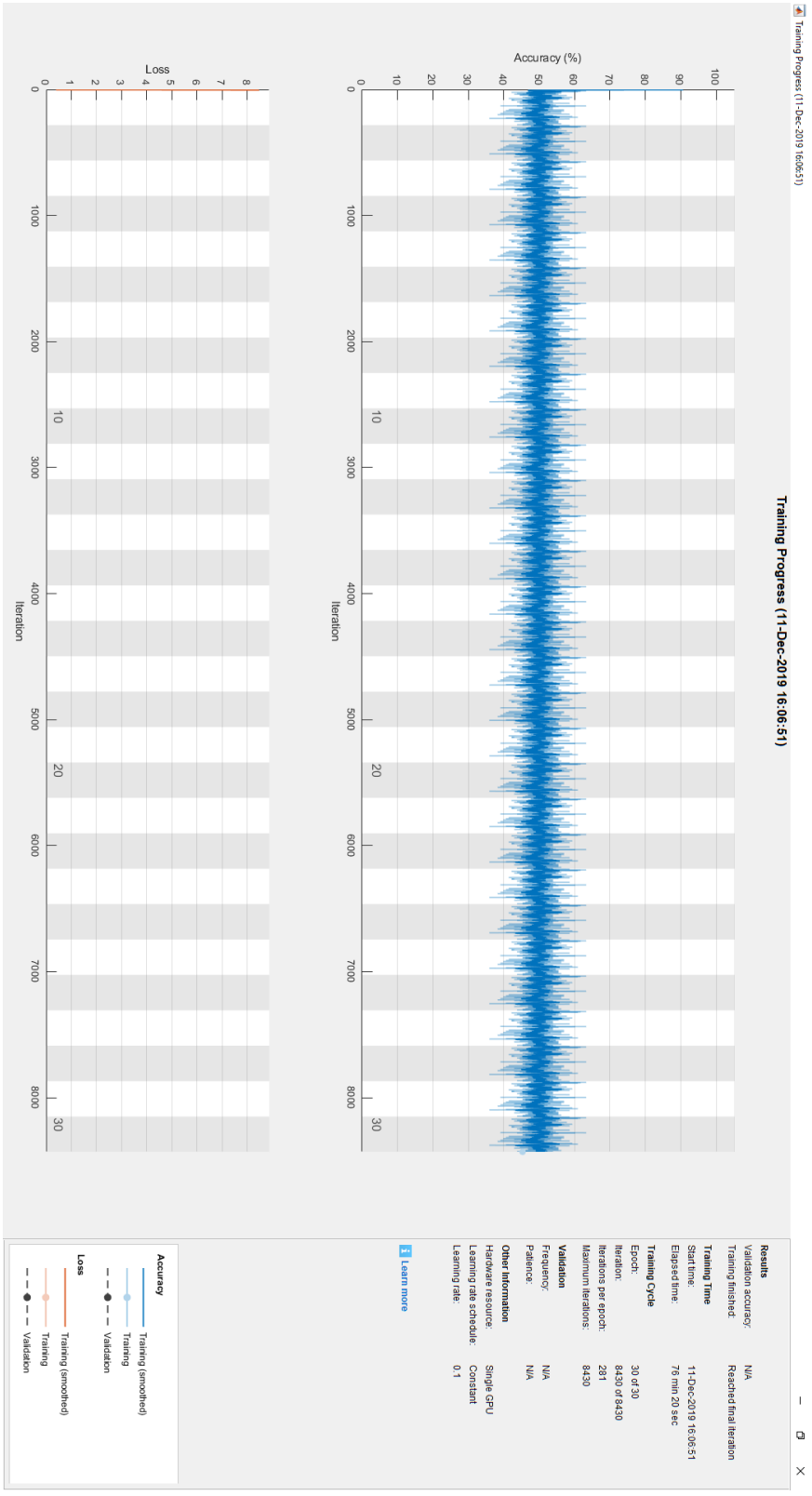


Figure 4.6: Training Progress – ILR 0.1

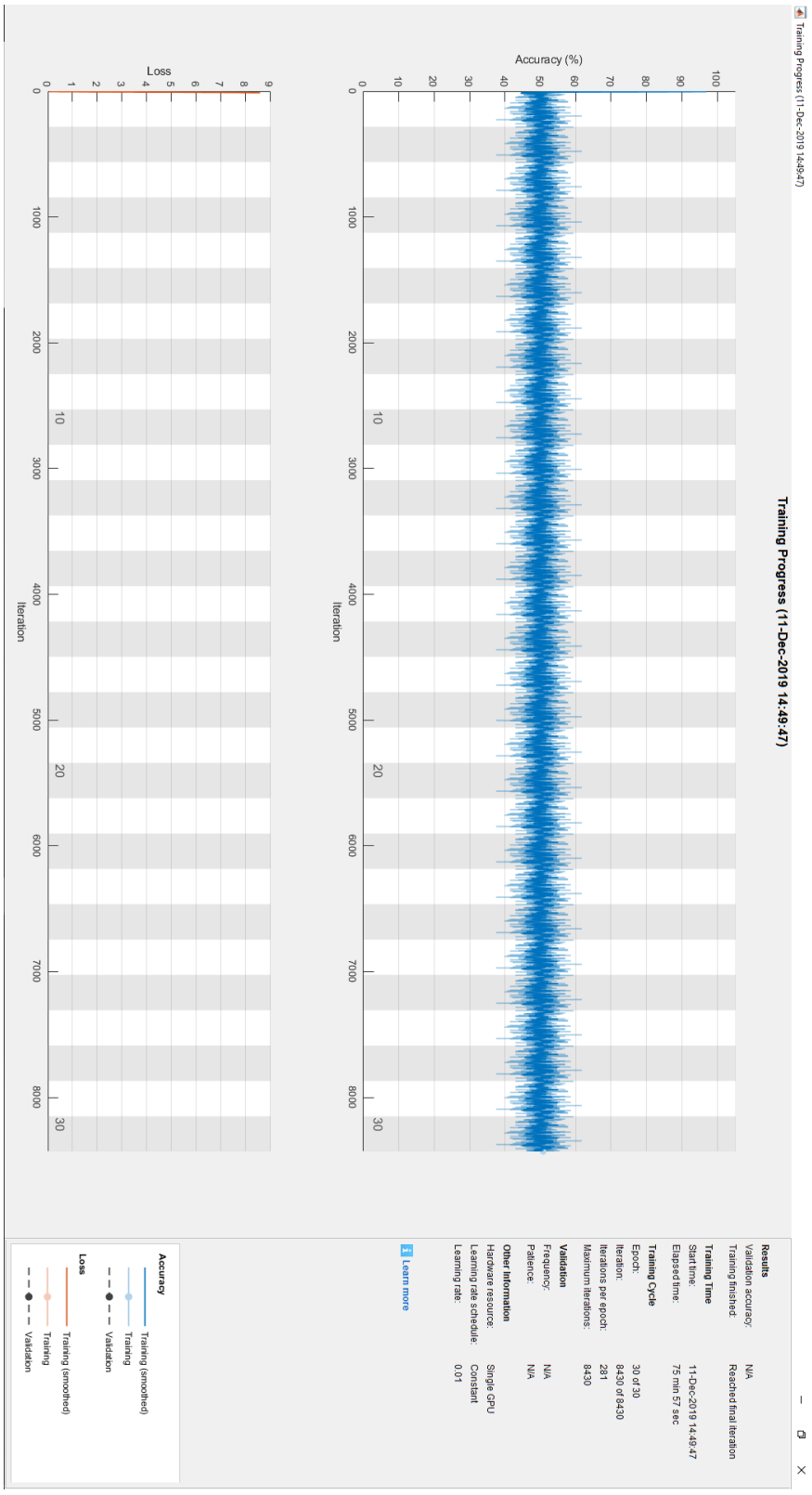


Figure 4.7: Training Progress – ILR 0.01

Figure 4.8: Training Progress – ILR 0.001

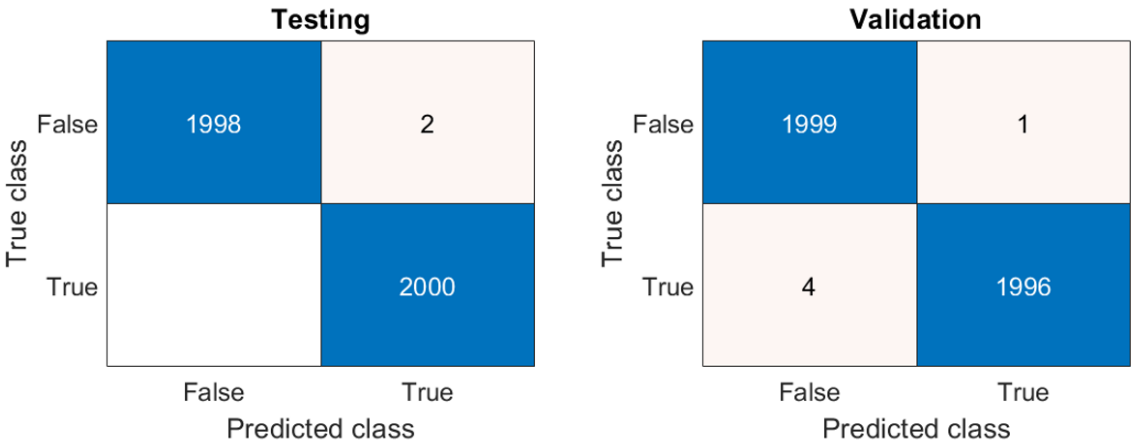


Figure 4.9: Prediction Confidence Charts

Figure 4.10: Concrete Testing Environment

was used to automatically capture images as the drone was moved manually along the environment. Multiple experiments were performed with this setup and the number of images captured ranged from 100-500 for every run. Each of the images was saved and then transferred into MATLAB as a datastore for access by AlexNet. Before the images are processed, a normal CNN classification would have to resize the images to 227 x 227 pixels. Our method originally followed that same procedure, but we realized that the Network would sometimes produce false predictions. To reduce that from happening, we decided to write a code that would first section the original image into n-number of sections where each section would get its own individual prediction and they would then be stitched back together. We experimented with a different number of sections and found that 3x3 is enough for this scenario, but smaller sections could be computed if needed. This decision was also affected by the original image size coming from the drone, which is 480 x 640 pixels, meaning that the images with the current camera setup would yield sections of 160 x 214 pixels if divided into 3 x 3 sections. Each of these sections would then have to be resized into 227 x 227 pixels. Since there is not too much difference in size, the resizing algorithm only produced a marginal loss of quality and focus on each of the images. Figure 4.11 depicts this sectioning taking place in a test image.

After running some tests with this type of image sectioning we found that the location of the crack in reference to the section image yields different results. This is because most of the images from the training dataset contain cracks that either run all the way through the image or are centered within the image. That means that if a crack is present in an edge or corner of one of the sectioned images or is small enough, it might not get picked up by the network and yield a false negative classification. In order to prevent this from happening, we modified the sectioning algorithm to include overlap in each consecutive section in both the x and y dimensions. We are currently working with a 78% overlap but that can be modified easily in the algorithm. We found this number through experimentation, anything bigger is too expensive computationally and anything lower still yields false negatives although at a much lower rate than before. Figure 4.12 depicts this new sectioning with overlap.

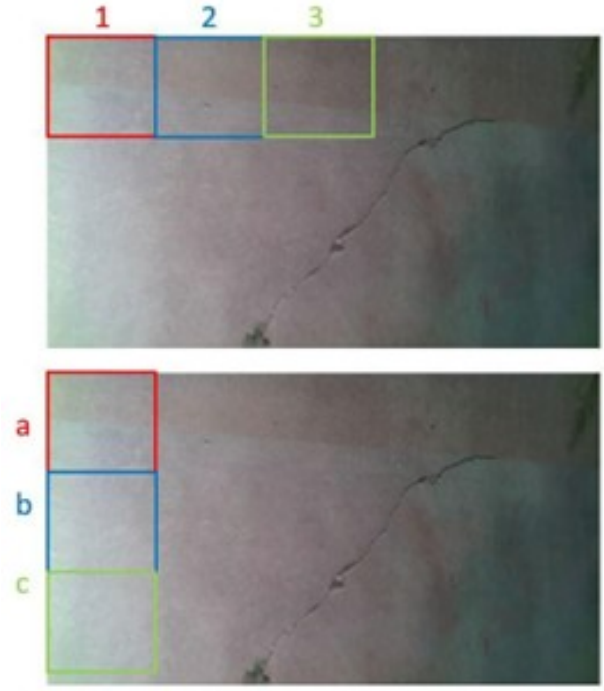


Figure 4.11: Image Sectioning

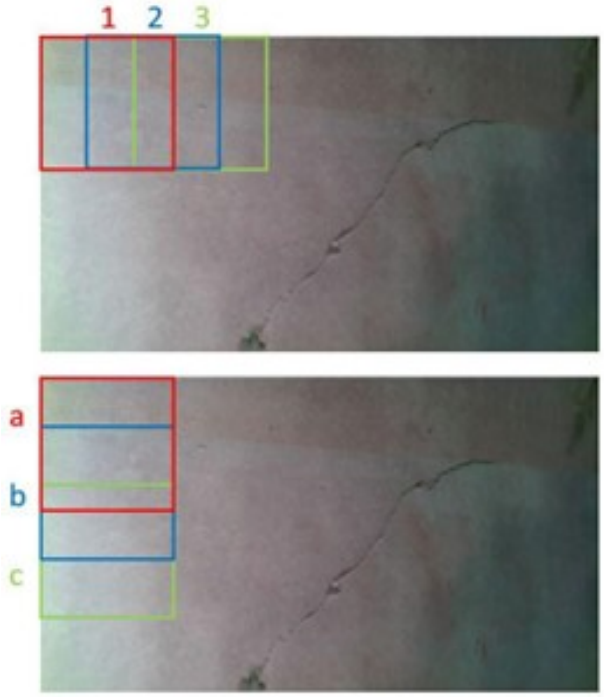


Figure 4.12: Image Sectioning with Overlap

4.5 Neural Network Testing

To test the network, we used captured images from the testing environment. These images were purposely different in brightness and colors from the training images so we could test the actual capabilities of the trained CNN. Figure 4.13 depicts the results obtained and it can be seen that the network predicted false negatives. This is really dangerous since the network is overlooking an area of the structure with damages whereas if it were a false positive it would flag an area as a defect where it does not exist. To prevent this from happening, an algorithm that uses recolor was created. It simply recolors all the actual images using a scale that is obtained by dividing the average Red value of the training images by the average Red value of the actual images and repeating the same process for the Green values and Blue values to obtain three different scales. This scale is then applied to the entire folder of actual images and a new copy is created. Figure 4.14 depicts the color difference between the training and the actual datasets. Figure 4.15 depicts the new prediction using the modified/recolored actual images dataset.

4.6 Image Segmentation

After the original image has been sectioned and each section has been processed through the CNN and given a prediction, they are processed in our segmentation algorithm. The segmentation algorithm is applied only to the sections that have a positive classification for a crack. The algorithm first uses the original section and transforms it into a grayscale image. Then, a binarize command is used to create a binary image by using an adaptive threshold. After the threshold has taken place the result is a grayscale image where the darker features, which are usually cracks, are highlighted. This new binary image is then used as a filter to produce only red on the original section image. At this point, this filter is applied to the RGB image but only to the Red channel. Figure 4.16 depicts this process. Finally, this process is applied to all the sections in all the images that were stored from the

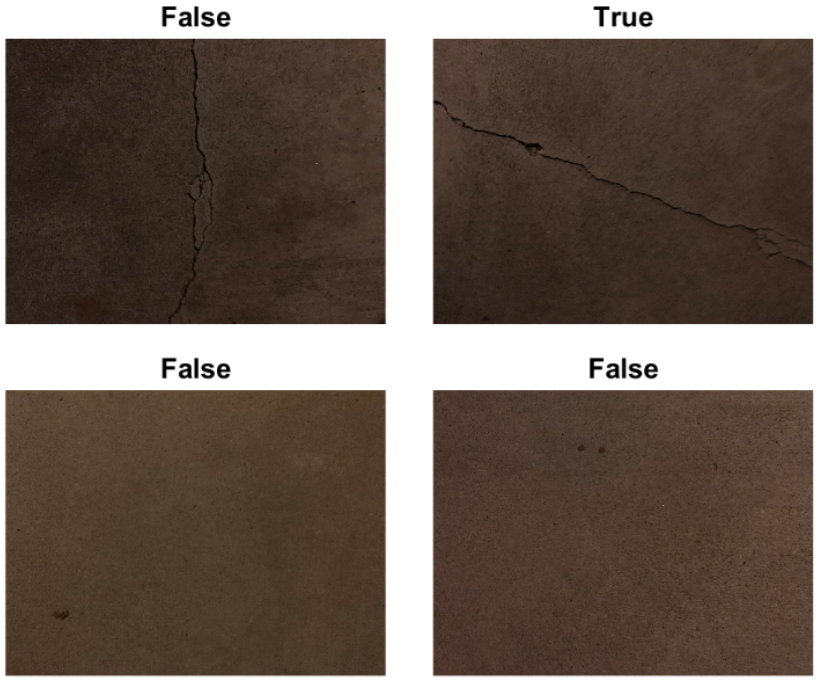


Figure 4.13: CNN Testing with Actual Images (Original Images)

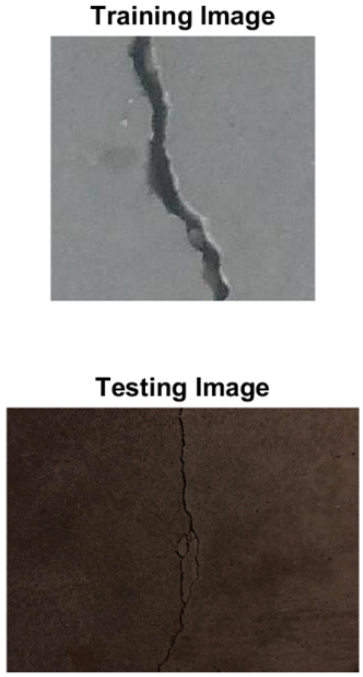


Figure 4.14: Training vs. Testing Images

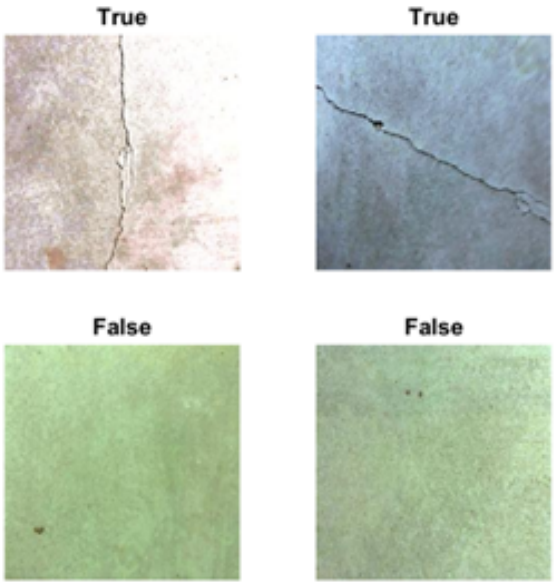


Figure 4.15: CNN Testing with Actual Images (Modified Images)

drone camera. This results not only in a list of classifications for each of those images but also in modified images that highlight the cracks and images that only include the cracks. These three outputs can be used in the future to not only stitch into a single image or 3D environment but the images with only cracks visible could possibly be used as a layer on a such image or 3D environment.

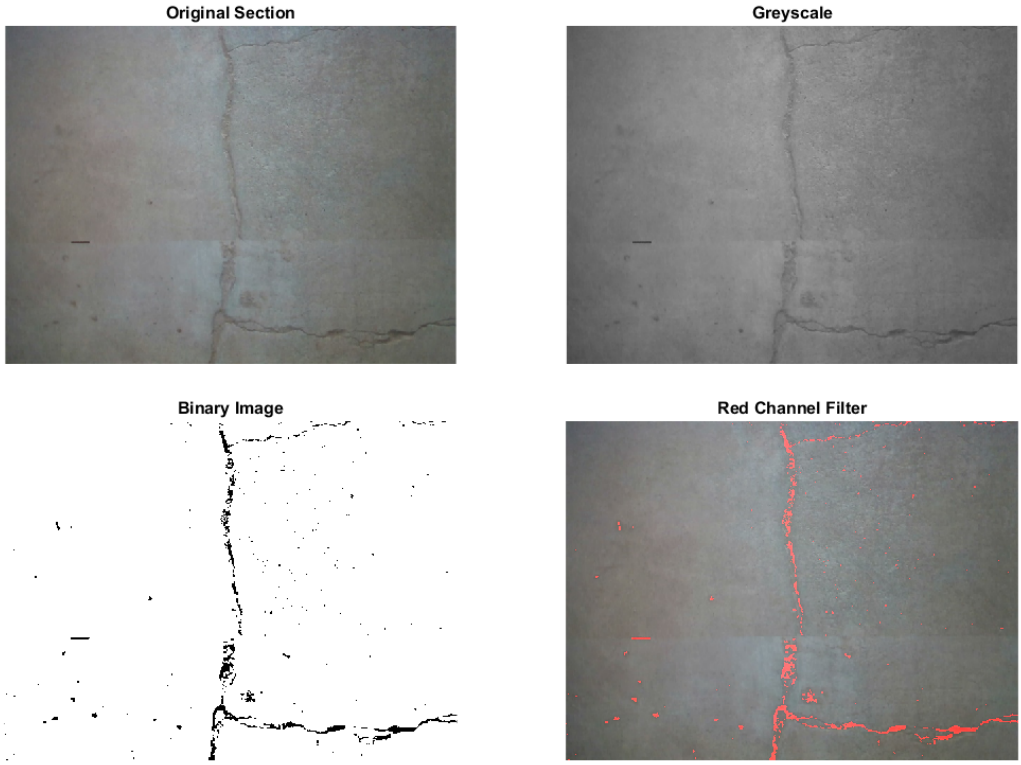


Figure 4.16: Image Segmentation

Chapter 5

3D Reconstruction

In this chapter, we will compare the different methods commonly employed for 3D reconstructions and the methods with the best result for our application. We will compare Photogrammetric Reconstruction, Open3D, and RTAB-Map. Each method uses different sensors from our payload and yields different results.

5.1 Payload Sensor Parameters

The RGB camera that was selected is the Raspberry Pi Camera Module v2 shown in Figure 5.1 and includes G-Streamer capabilities directly compatible with the Jetson Nano. The specs of the RGB camera can be found in Table 5.1.

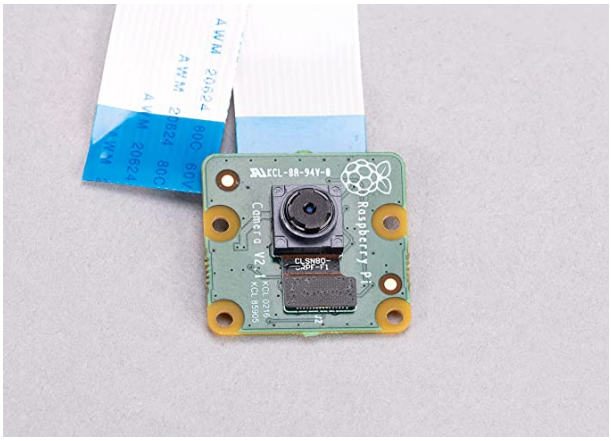


Figure 5.1: Raspberry Pi Camera Module v2

The payload also includes an Intel RealSense D435i depth camera as can be seen in Figure 5.2. The depth camera employs a set of sensors that are identified in Table 5.2.

Table 5.1: Table to test captions and labels.

RGB Camera	
Attribute	Value
Sensor Type	IMX219
Megapixels	8
F-Stop	2.0
Focal Length	3.04
Vertical Field of View	48.8
Horizontal Field of View	62.2
Maximum Resolution	3820 x 2864

Figure 5.2: Intel RealSense D435i Depth Camera

Table 5.2: Table to test captions and labels.

Depth Camera	
Attribute	Value
Sensor Type	Intel RealSense D435
Depth Technology	Stereoscopic
Range	.3 m to 3 m
Depth Field of View (FOV)	87° × 58° (±3°)
Depth Stream Maximum Resolution	1280 × 720
Depth Stream Maximum Frame Rate	90 fps
RGB Maximum Resolution	1920 × 1080
RGB Sensor Resolution	2 MP

5.2 Photogrammetry

5.2.1 AliceVision Meshroom Background

For this work we decided to employ a photogrammetry approach to generate reconstructions from RGB images. Figure 5.3 depicts the framework pipeline of AliceVision's Meshroom. The following list give some insight as to what each of those nodes is doing. This pipeline is based on [1] and [20].

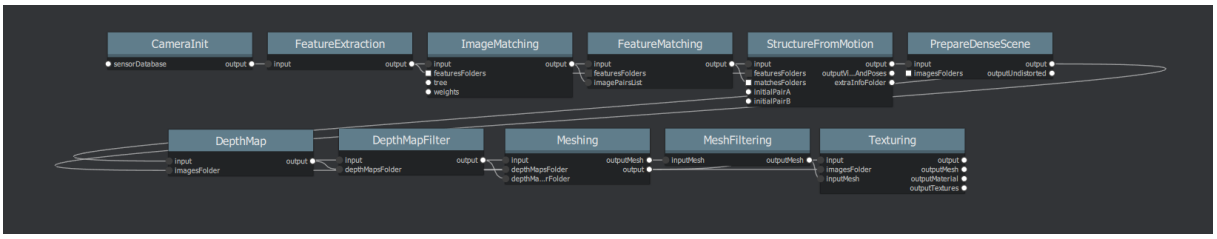


Figure 5.3: AliceVision Meshroom Pipeline

1. Camera Initialization – Initializes the camera parameters from images. Image meta-

data contains camera properties like focal length, camera type, make, model, sensor width, and image height. Creates a virtual camera localization using the images and metadata.

2. Feature Extraction - The objective of this step is to identify and extract specific groups of pixels that are constant even with changing viewpoints from the camera. This means that these pixels make up features that are identified within the camera feed or image dataset.

3. Image Matching – The features found before for each image must then be compared and matched to subsequent features in images.

4. Features Matching – Once all image pairs have been found, this step will identify the feature between such image pairs. Individual distances between corresponding features among two separate images are calculated. The difference between the two pairs will eventually make a 3D structure from corresponding features.

5. Structure-from-Motion – This step tries to identify the geometric relationship between the corresponding features. This step will output an initial 3D scene structure with camera position and orientation, also called pose.

6. Depth Maps Estimation – Once all cameras have been calculated by SfM, the depth value of each pixel in each scene is retrieved. For each image, a specific number of cameras is found that are in the immediate vicinity of the current pose. A small portion of the main image is reprojected into the neighboring cameras and each neighboring projection will accumulate similarities.

7. Meshing – This step generates a dense geometric reconstruction of the scene. All the depth maps from the previous step are merged into a single map also called an octree. A 3D Delaunay tetrahedralization takes place to find interconnection between all point in the point cloud. This then generates a mesh of the structure.

8. Texturing – For each triangle in the mesh previously created, texture candidates are identified from vertex colors. Different pixel values are assigned to each vertex due to the repetitive nature of the previous step. Finally, all of those values are averaged to give each vertex its final pixel value.

5.2.2 Meshroom Implementation

After the images have been classified and post-processed, they are stored in a new folder and they are later used to generate a 3D render of the entire inspection site. We use Alicevision's MeshRoom open-source software since it allows us to control every step in the photogrammetry process. This software uses the concept of Structure-from-Motion to obtain matching features between images. This software does not require GPS information for any of the images to create the render, instead, it calculates the "virtual" location of each image internally. Although it doesn't require GPS, most of the readily available images might already have GPS information stored in the metadata of each image. The software can use the information to skip the camera location calculations but it does require other pieces of information mostly regarding camera properties like focal length and sensor size. One of the challenges we faced in using the datasets from the drone camera is that most of the metadata was missing in both the pre-processed and post-processed datasets. We created another section of code that can override the metadata with the value pertaining to the camera properties we need, listed in Table 5.1. This code was then used on both datasets. Once the images are updated with the metadata, they are then sent to the MeshRoom software for stitching. We have obtained some results that still need improvement but show great potential. Figure 5.4 depicts the 3D render obtained using the pre-processed and post-processed datasets.

Figure 5.4: Photogrammetric Reconstruction

5.3 ORB-SLAM2

The process of installing and running ORB-SLAM2 in the companion computer Jetson Nano proved more difficult than expected. There were a series of installation errors that had to be addressed but were ultimately fixed. Within the ORB-SLAM2 system, there are two main modes, SLAM Mode and Localization Mode. In SLAM Mode, the default mode, the system runs three pipelines in parallel: Tracking, Local Mapping, and Loop Closing. In this mode, the system localizes or "tracks" the camera, continuously builds the map and tries to close loops. In Localization Mode, the system only localizes the camera but deactivates the Local Mapping and Loop Closing pipelines. This mode can be used when there is a good map of the environment, but it does not update the original map. We find this process more convenient since we already have a very good map of the environment but would still want to generate a new map with SLAM. To do so, we reviewed some articles and found an extension available on GitHub that allows us to save the new map even in Localization Mode while still being less computationally expensive than SLAM Mode [32].

5.3.1 Map Saving and Loading Extension

We spent a good amount of time working with the two codes in an attempt to understand and tune the existing software to our needs and hardware. The extension we are currently working with, allows us to run the ORB-SLAM2 algorithm through ROS but

it also publishes real-time ROS Messages from that code. These messages are extremely important since they provide data from ORB-SLAM2 that was otherwise hidden within the code and never published for the user. From this data, we can obtain the live location of the camera, and live feed of the camera images before and after SLAM feature detection and we also have access to a live and constantly updated Point Cloud of the environment it is mapping. This new data gives us access to a map that we did not have before. We ran initial tests with ORB-SLAM2 and the depth camera with the handheld setup at the Vinton Steel factory. Figure 5.5 shows the inside of the structure we initially tested with ORB-SLAM2. Figures 5.6, 5.7, and 5.8 depict a view of the camera feeds and the Point-Cloud being generated, everything is being visualized through RVIZ. Figure 5.6 shows the initial frames and Point-Cloud after a couple of seconds of initializing the system. Note the difference between the two camera feeds in the top left corner. The first feed is the original camera feed in RGB while the second is the output feed after the feature detection of the SLAM algorithm. Figure 5.7 then depicts the updated Point-Cloud after the test was done. Figure 5.8 depicts the feed when the camera might get lost due to rapid movement or external factors, before finding its position again within the environment it has already mapped. Figure 5.9 depicts the Point-Cloud generated being deployed in MATLAB which proves it might help with the already existing Trajectory Generation Algorithm that was developed earlier in the project. Figure 5.9 can be compared to Figure 5.5 in an attempt to understand the perception of the Point-Cloud being displayed. We will continue our work in using this Point-Cloud generated not only as a substitute for the CAD if needed but also as an initial environment for the SLAM algorithms being deployed.

5.3.2 Mesh From Point Cloud Data

Obtaining a mesh from point cloud data is an important task for us since it would allow us to use that mesh in our trajectory generation algorithm and obtain a close-inspection trajectory for our flight missions. This is no easy task since there is no straightforward method to do it. The industry uses different approaches to solve this issue. Some companies

Figure 5.5: Inside View of Vinton Steel Structure

provide the service at a cost to the consumer, but it can be expensive and requires a turnaround time to process. Also, it does not provide the user with any control over the final product. We studied different approaches ranging from algorithms that would identify color patterns from the pixel data in each point from the point cloud to a very simple closest-neighbor approach to connect all the points from the point cloud data.

We realized that the first approach would require a lot of work and we were unsure if the extra work would provide any significant improvement over other more traditional methods. The second approach also had some drawbacks since point clouds do not have a uniform distance between points and a single value for hull parameters is invalid and thus results in a mesh that does not match the scanned environment. This is where the private companies come in, they analyze the entire data and assign different hull parameters to different sections of the point cloud. They of course use proprietary algorithms that are not accessible to the public. The third option we propose is to analyze the problem as a histogram or OctoMap type of problem.

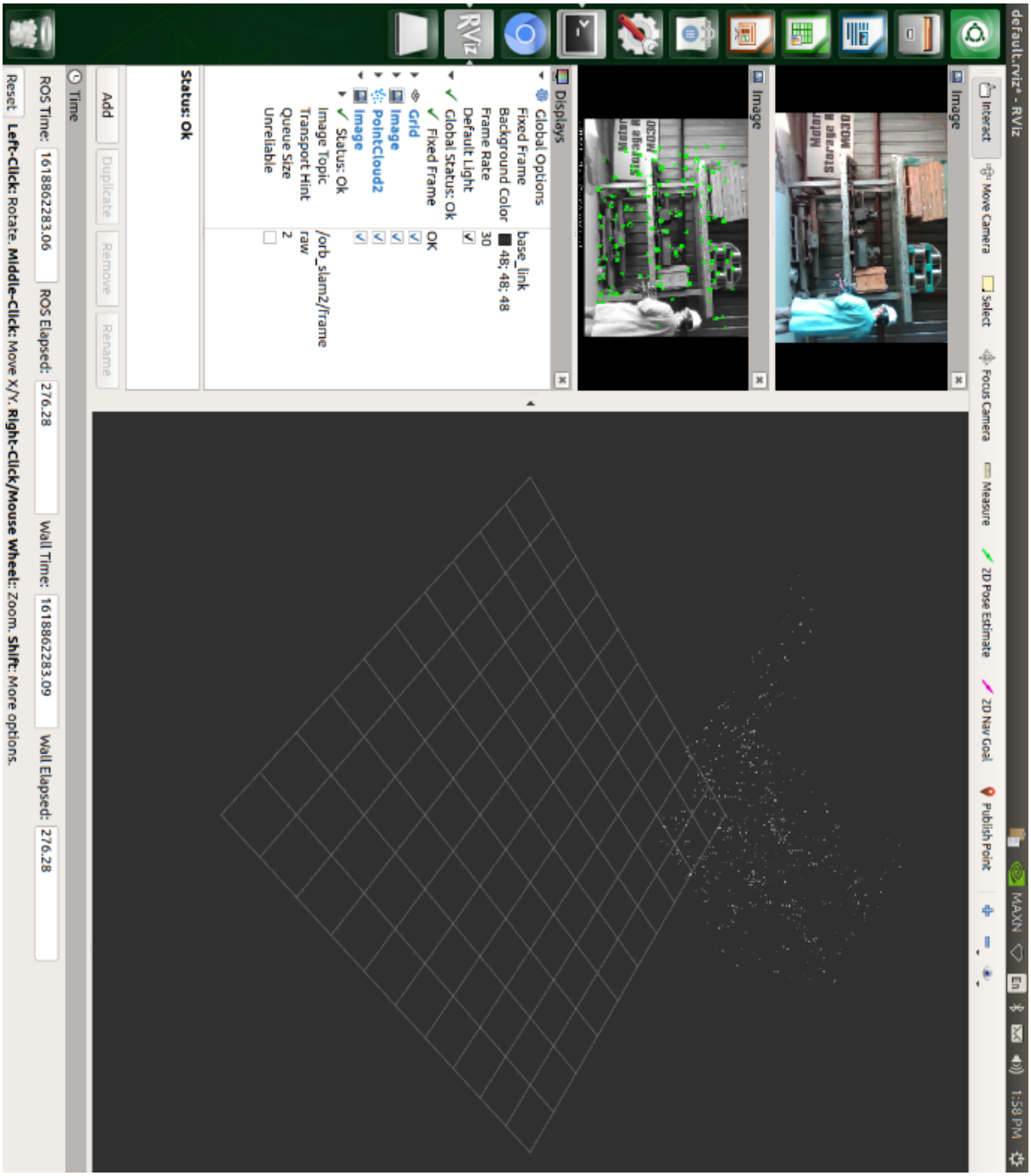


Figure 5.6: ORB-SLAM2 Feed Being Displayed; Includes camera Feeds and Point-Cloud. Initial Seconds of Test

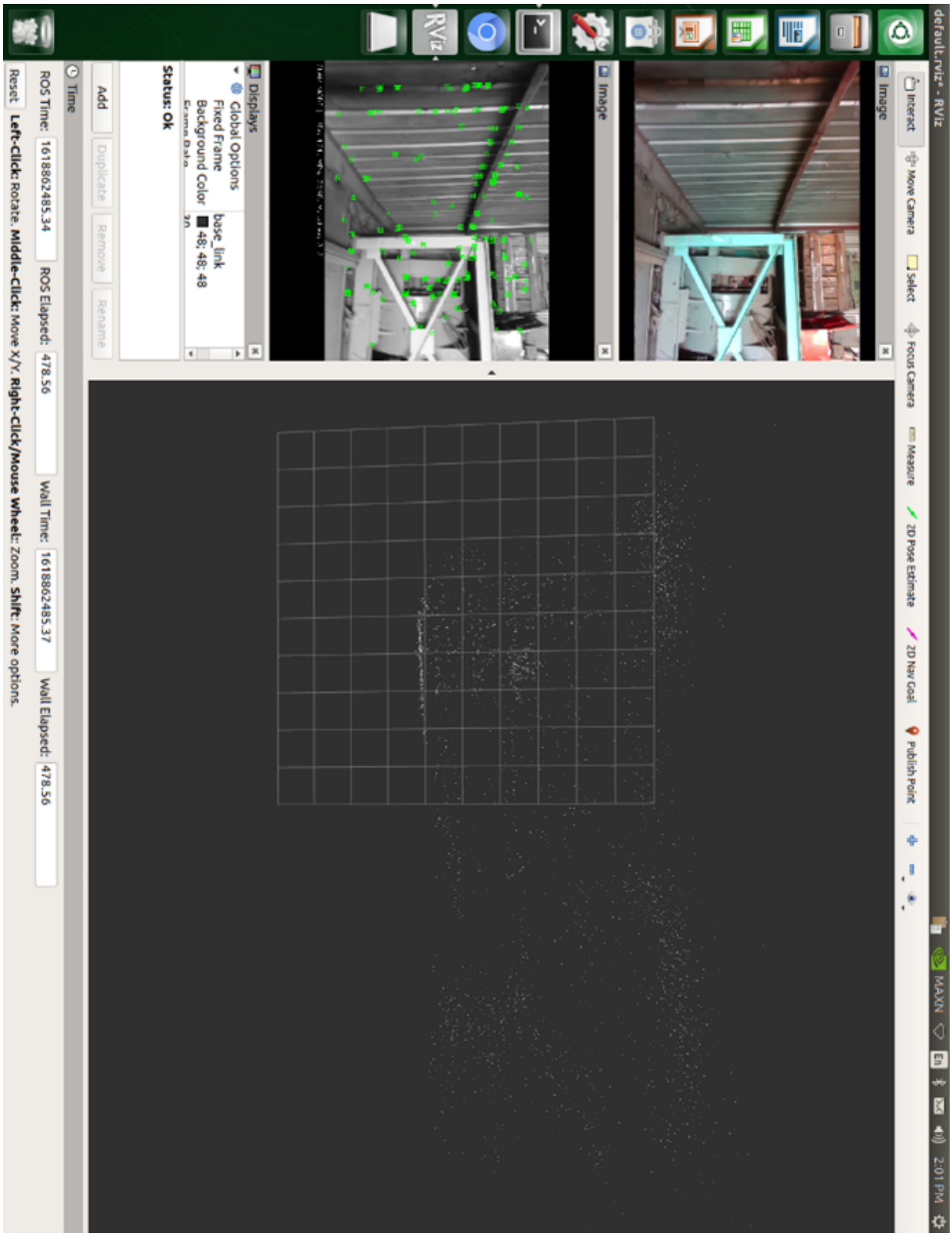


Figure 5.7: ORB-SLAM2 Feed Being Displayed; Includes camera Feeds and Point-Cloud. Final Seconds of Test

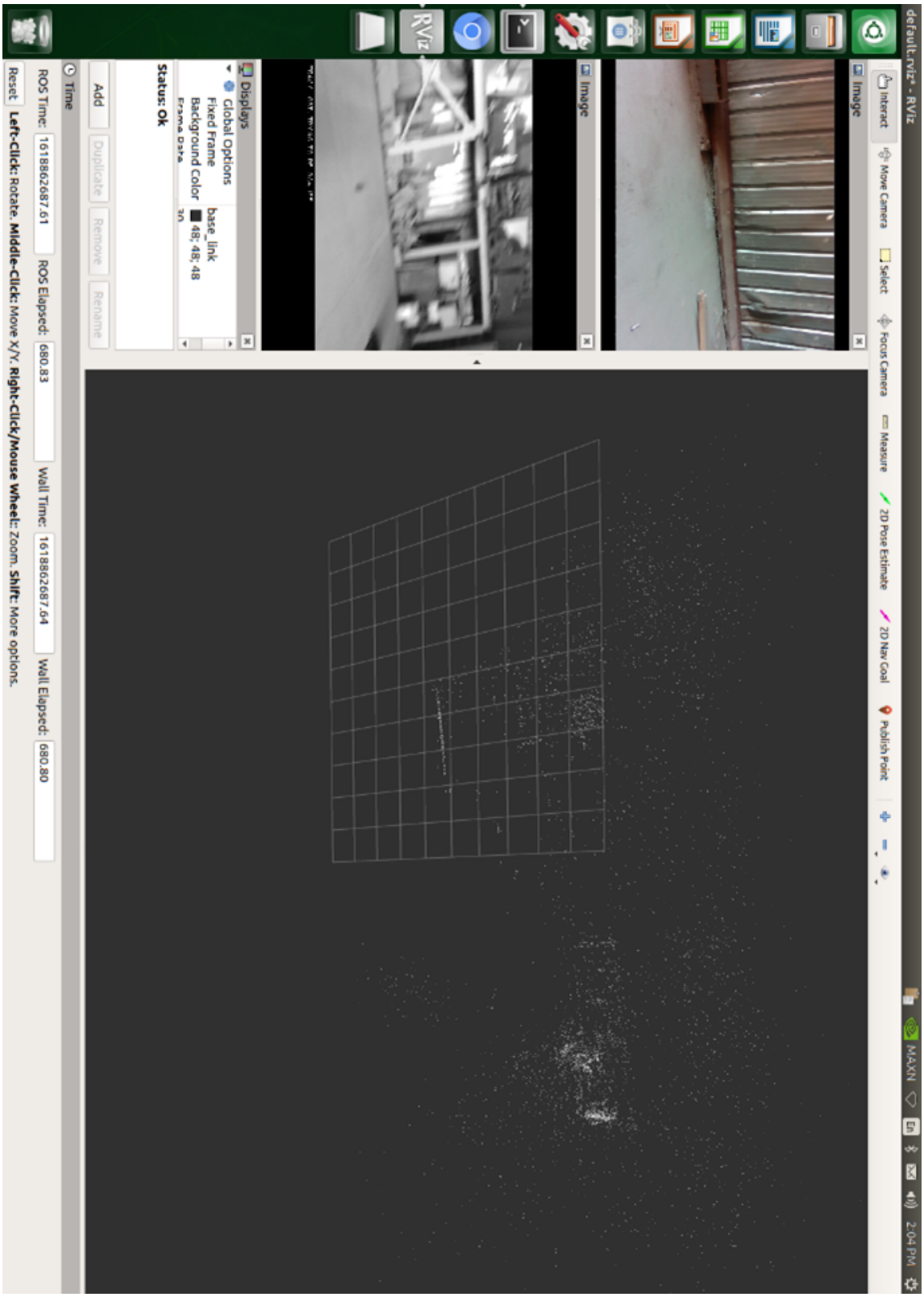


Figure 5.8: ORB-SLAM2 Feed Being Displayed; Includes camera Feeds and Point-Cloud. Camera Loses Track and Tries to Re-localize Itself

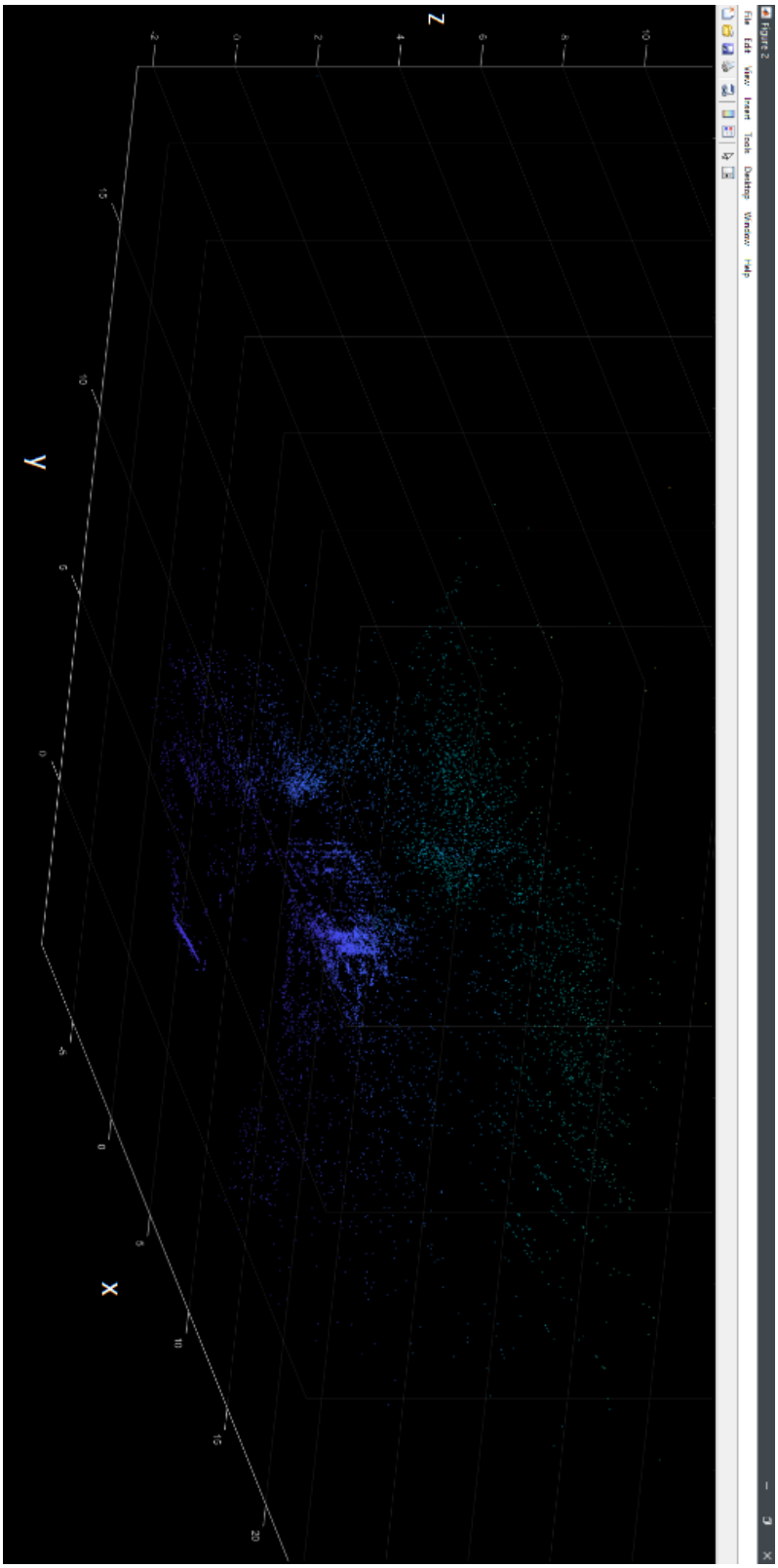


Figure 5.9: Final Point-Cloud generated deployed in MATLAB

We have point cloud data that provides the location of features found from the environment scan. This means that there is a body in each of those points. The approach then turns into an occupied cell within a 3D array with a cell of uniform or variable dimensions. For our initial tests, we are using uniform dimensions that vary in size between each test. The location of each occupied cell is then used to generate a cube mesh around each occupied point. The initial tests prove our approach might work but will still need some refinement. We are currently working on making the algorithm more efficient since the number of points in the point cloud data could increase exponentially depending on the size of the environment we can inspect. The second improvement we are currently working on is to use a dense point cloud scanning method instead of the traditional SLAM algorithm since SLAM greatly reduces the number of points in the dataset to be more efficient. We would like to have as many points as possible to make our method more efficient. Figure 5.10 depicts the occupancy map method with a cube size of 3 cells/meter, the Point-Cloud data is the same as in Figure 5.9. Figure 5.11 depicts the occupancy map method with a cube size of 5 cells/meter. The cubes represent the mesh that could be used to generate the trajectory and the red points represent each of the points in the point cloud data. Notice that some cells might contain more than one point. We could reduce the mesh size even more if we had a denser point cloud from our scans. A resolution of 10-20 cells/meter would be ideal.

Unfortunately, we were unable to get ORB-SLAM2 running correctly on the companion computer without crashing because of its computing requirements. Perhaps if a more powerful computer like the Jetson Xavier was used, it would allow the algorithms to run correctly. This would be complex since the reconstruction system would have to run in parallel with all the flight control algorithms in the UAV. We decided to abandon this research path and focus on alternatives to generate reconstructions.

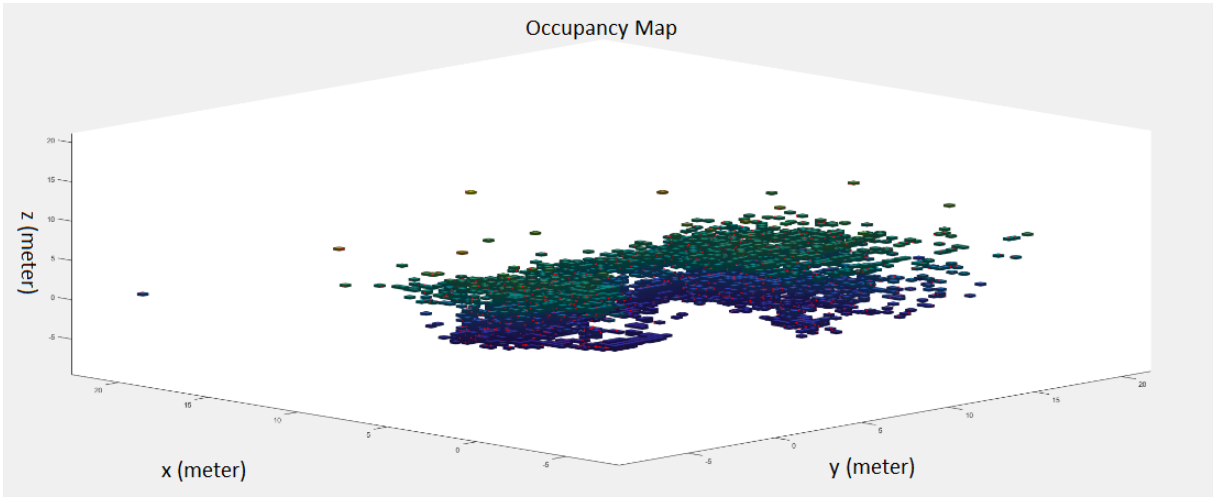


Figure 5.10: Occupancy Map with Mesh Size of 3 cells/meter.

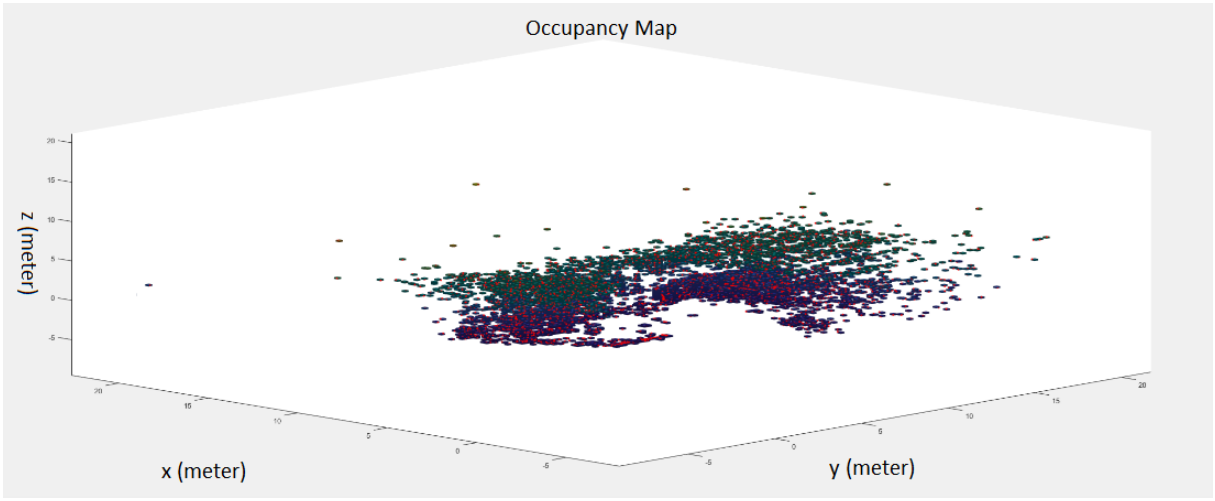


Figure 5.11: Occupancy Map with Mesh Size of 5 cells/meter.

5.4 Open3D

Open3D is an open-source library that supports the rapid development of software that utilizes 3D data. We are currently exploring the option of using the library for three stages in our 3D reconstruction of unknown environments or structures that do not have a readily available CAD model. The first stage is Data Capture of point cloud data using an Intel RealSense D435. We ran initial tests in this mode with modifications in different parameters for the camera in capture mode. One of the main parameters we are focusing on is the depth cut-off distance. We have successfully completed capture with the three distances of 1, 3, and 9 meters. Another aspect we focused on is the recording speed. Our initial tests showed that our data capturing did not match the overlap of benchmark datasets that are common for reconstructions. We identified a big difference in the overlap of the images where the benchmark had much more overlap compared to our dataset. We fixed the issue by making sure we move much slower when we record the images. A speed of 0.25 – 0.5 m/s is ideal.

Initial tests show that reconstructions of small datasets usually compute without many issues. Tests also show that with an increase in the size of the datasets the algorithm will often cause issues due to parameters not being fine-tuned. We ran tests to find the modifications needed to allow this second stage to run correctly. We found that the system did not like datasets higher than 200 images to do scene reconstruction. This has to do with the computing capabilities of the computer used. We are currently using a computer with 16 GB of RAM and an additional swap file of 16 GB to supplement the physical RAM. When we use datasets bigger than 200 images, we tend to have crashes on the system because the RAM is ramped to the max. To fix this issue we opted to keep datasets of 150 images max for reconstructions. This means that we divide the main datasets into subsets of 150 images for depth and RGB each and we usually use the last 50 images in a subset as the first 50 images in the next subset. This is done so the reconstructed scene will have overlapping areas we can use to stitch the two scenes together.

The second stage was obtaining a reconstructed model from the datasets recorded. In this part, we can separate into two main components: individual scenes and integrated scenes. For the individual scenes, we have found that when we use the benchmark datasets, the data is perfect to run individual scenes and the reconstruction does not display any errors. When we use our recorded datasets, we found that the data produced does not yield the best results and it is visible in the reconstructions. We usually have issues with points in the reconstruction that are out of place and some of those points even display as rays in spaces where there should not be any points. We believe some of these issues are because of the lighting conditions of the environment and how the camera behaves in these conditions. Some of these issues will never be able to be fixed since there are limitations to the hardware. We are researching if we could pre-process the dataset to account for some of those conditions. Research has been done to normalize the depth images so they are within a specific range, we plan on working with that to see if this step can help improve our results.

When dealing with the integrated scene, we found some errors in Open3D that we could not fix. Usually, Open3D will do individual scenes and then run an integration model to give a final scene. From testing, we found that although individual scenes might look good, once they pass through the integration step, the resulting scene tends to be wrong. The most common error is the two scenes being stitched in the wrong place with respect to each other. This means that the overlap between the two is wrong, resulting in a scene that might not give any useful information. We have fixed this issue by looking at all the individual scenes and running the integration with an external software called MeshLab. Each individual scene is imported as a point cloud to the system, and each is kept as a layer. The software then allows us to combine two layers by selecting intersection points between each of them. Once that step is done, then the scene will be stitched together into a single integrated scene. We tried to streamline this process to reduce the need for operator manipulation to achieve quality reconstructions but were unable to do so. Figure 5.12 depicts a completed integrated scene with all the previous steps mentioned.

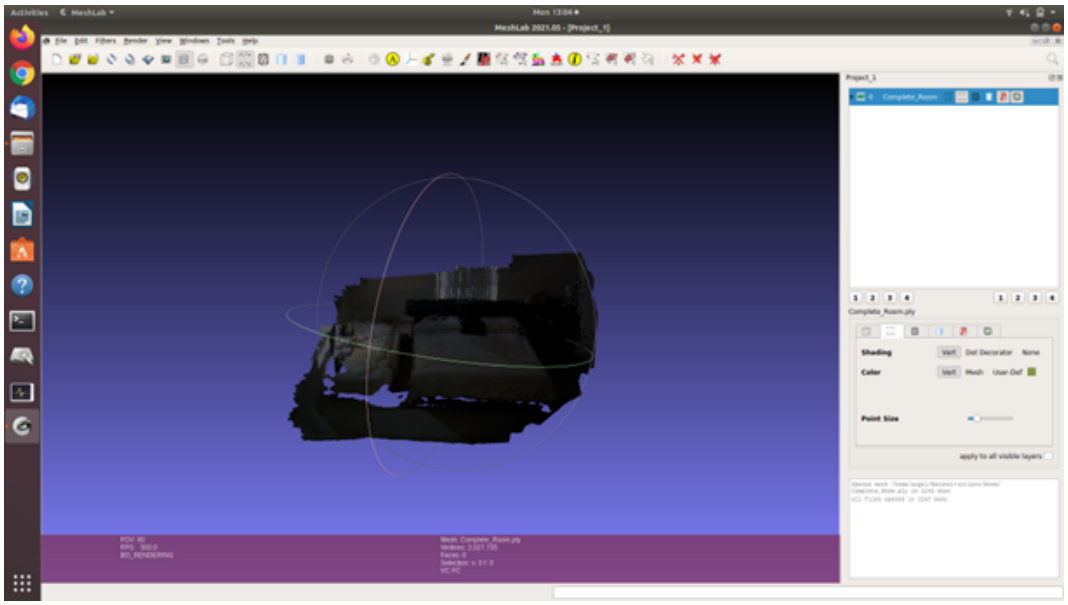


Figure 5.12: Integrated Reconstruction Using Open3D and Visualized in MeshLab

In this stage, the model is also saved as a PLY object. In this format, the model can be manipulated and converted to STL to be used with our trajectory algorithm. We also decided to test the library using our own dataset images. Figure 5.13 shows the reconstruction of a classroom desk. As shown, the reconstruction works partially with better results to be desired.

We decided to leave Open3D as a third alternative but Photogrammetry and RTAB-Map yielded the best results for our systems. but ultimately abandoned the task due to time constraints.

5.5 RTAB-Map

5.5.1 RTAB-Map Background

Real-Time Appearance-based Mapping (RTAB-Map) is a SLAM approach that employs an color-depth (RGB-D) camera feed to map and localize itself within a virtual environment. The system employ a loop closure approach with a bag-of-words approach where it tried

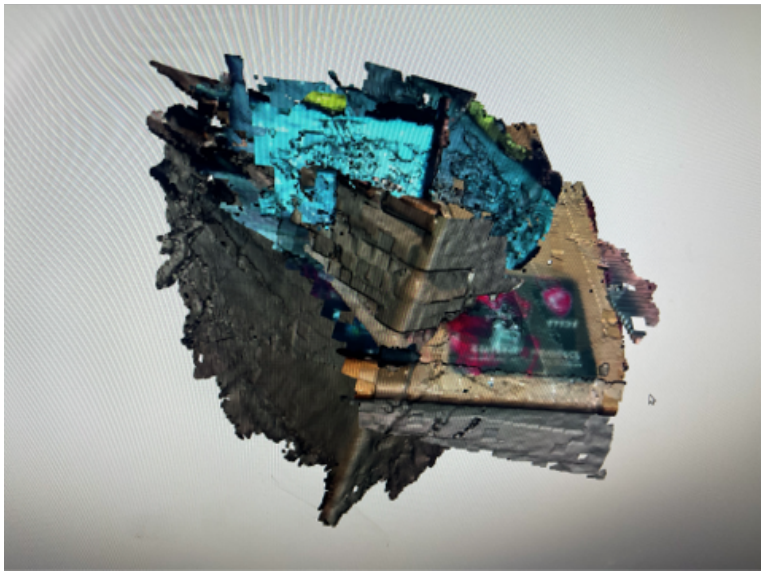


Figure 5.13: Partial Reconstruction using Open3D Library

to determine whether or not the current RGB-d image is part of a new location or if it has already visited that location. Once an existing map has been created, the model can run on Localization mode only, resulting in faster processing since it no longer needs to make the decision of whether or not is a new location. Instead, the system knows to only look for a location with matching point cloud features [4] [3] [17] [27].

5.5.2 RTAB-Map Implementation

After spending some time comparing the different methods we have employed for this research, we have decided on two main methods that work best with our sensors and onboard computer. We concluded that both Photogrammetry and RTAB-Map are the two best options. RTAB-Map is a reconstruction approach that is based on an incremental appearance-based loop closure detector. This system works with different types of cameras such as Color-Depth (RGB-D), Stereo Cameras, and even LiDAR. For this technology, we use the feed from the Intel D435 Depth Camera. This sensor has two RGB imagers along with a depth infrared projector. That means that this camera could be used as an RGB-D

or a stereo sensor. For this approach we decided to take advantage of the depth sensor, so we are running the system with an RGB-D input from the camera.

We are currently doing the process in two steps, we first capture, and then we process the reconstruction. For capturing we are currently using two methods: we employ the Jetson Nano and save bagfiles containing the feed from the multiple sensors; the second method is using an Apple iPhone XS Max which contains a lidar sensor. RTAB-Map can run on the iPhone through an app, the app will automatically generate a database file that is interchangeable through the different operating systems, so we can run those files on Ubuntu either on the onboard computer or an off-board machine.

The iPhone is first used to generate the initial reconstruction which is high quality, but the range of the sensor is limited so objects at big heights or long distances are not recognized. To solve that we then employ the Jetson Nano with the depth sensor to add more detail to the reconstructions. Sometimes the reconstructions are also big so multiple sections must be reconstructed individually. We make sure that there is enough overlap between the different reconstructions to be able to stitch them together. The overlap can be achieved with the feature detector of RTAB-Map or with the point alignment of point clouds using MeshLab. RTAB-Map will work automatically by running code while MeshLab requires manual operation of the software to achieve the alignment. Most of the time the RTAB-Map alignment will work well unless the visual textures of the reconstructions are not uniform due to lighting conditions. In that case, MeshLab is preferred since it does not consider the visual properties of the point cloud but the point cloud density and shape at the connection/overlap sections.

Figure 5.14 depicts the different section reconstructions in the same environment while Figure 5.15 depicts the sections after they are aligned. Once those sections have been aligned, they can be post-processed in MeshLab to convert the PLY format to either STL or OBJ for use in Fusion 360. We pretend to use Fusion 360 to add any details that might have not been captured. After that, the model is ready to be sent to our trajectory generation algorithm.

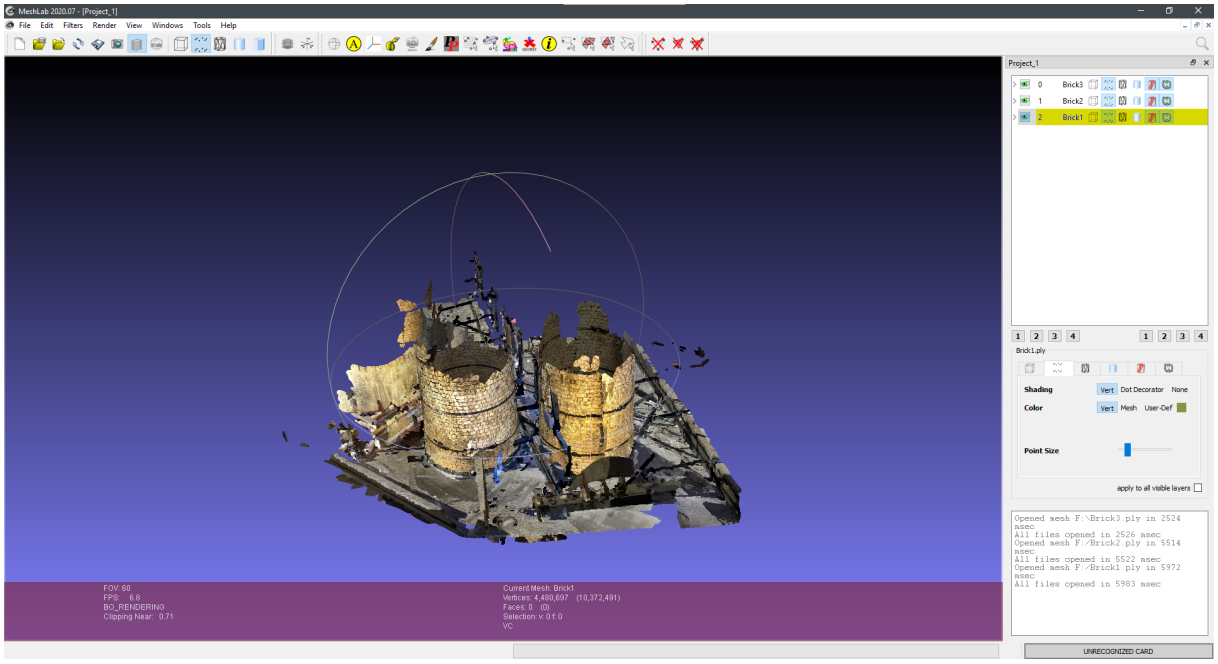


Figure 5.14: Unaligned RTAB-Map Reconstruction Sections Displayed in MeshLab

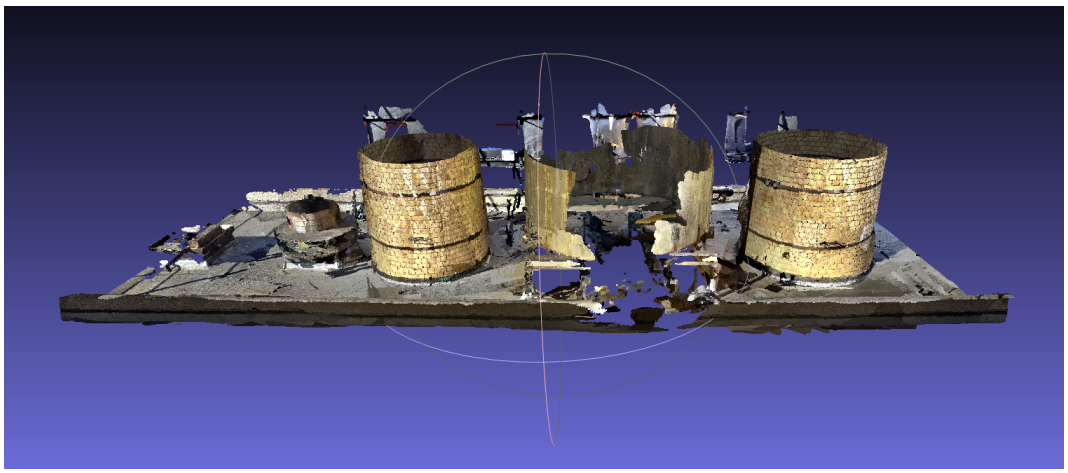


Figure 5.15: Integrated RTAB-Map Reconstruction Displayed in MeshLab

Chapter 6

The UAV Platform

The aerial platform used in this project consists of an off-the-shelf frame with an Nvidia Jetson onboard computer, Pixhawk flight controller, and custom hardware. The drone is depicted in Figure 6.1. All data and parameters are taken from the said UAV system for the corresponding calculations.

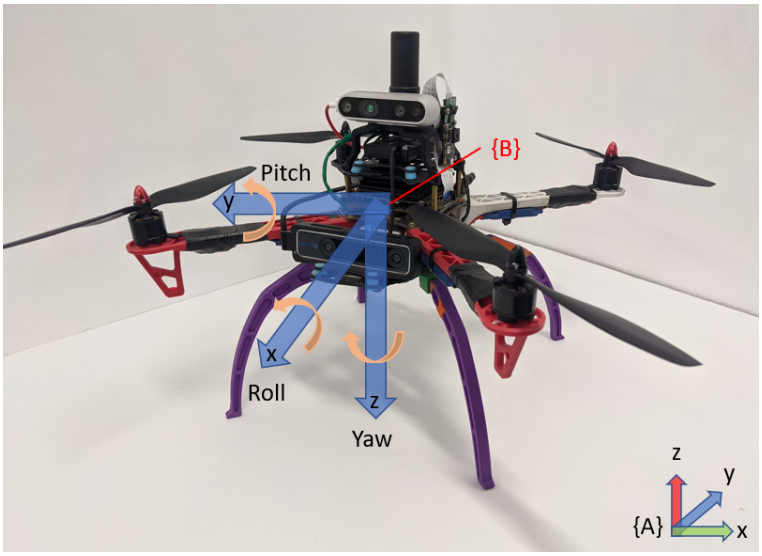


Figure 6.1: UAV Kinematic Forces.

6.1 The UAV System Modeling

To analyze the motion of the drone, an inertial frame of reference {A} (or world frame) and a body coordinate frame {B} are included. {A} is selected at a convenient location that acts as the global reference for the plant to be inspected. On the other hand, {B} has

its origin coincident with the center of gravity of the quadrotor, the z-axis downward, and the x-axis in the direction of the flight, following the aerospace convention (Figure 6.1). Besides, as depicted in Figure 6.2, the position of the drone in inertial frame {A} is given by the vector $\mathbf{r} \in R^3$.

The inspection cameras on the UAV are installed in the chassis and are near the depth sensors; therefore, the camera location can be considered as the point of origin/zero point for the drone. Notably, the UAV propellers expand beyond this point by a_d and from the extreme endpoint of the propeller to the closest point on the inspection surface can be designated as d_s. As such, the total inspection offset from the camera at any time is,

$$d = d_s + a_d \tag{6.1}$$

For safe flight path planning, a minimum drone offset distance d_s is introduced. Also, due to the design of the drone, the rotor blades expand a certain distance from the camera which also needs to be considered along with the camera focal distance itself. Therefore, the offset distance should be, $d \geq d_s + a_d$ and $d > f$ at any time (Figure 6.2). As such, the position vector for the UAV camera is,

$$\boldsymbol{r} = \begin{bmatrix} x + d & y & z \end{bmatrix}^T \tag{6.2}$$

For a quadrotor, the angular motion state about the three rotation axes is represented by six state variables: the angular velocity vector, $\boldsymbol{\omega} \in \boldsymbol{R^3} = [\omega_x, \omega_y, \omega_z]^T$ about the three axes of the body frame and the vector of Euler angles $\boldsymbol{\beta} \in \boldsymbol{R^3} = [\phi, \theta, \psi]^T$, representing the roll, pitch and yaw angles, respectively. The 3-2-1 *(Z-Y-X)* Euler angles are used to model the rotation of the quadrotor in the global frame. The rotation matrix for transforming the coordinates from {A} to {B} is given by the rotation matrices about each axis *(**R**'* (ϕ), ***R**'* (θ), and ***R**'* (ψ)). Therefore, the rotation matrix $\boldsymbol{R} \in R^{3\times3}$ that expresses the orientation of the coordinate frame {B} with respect to global reference frame {A} is obtained.

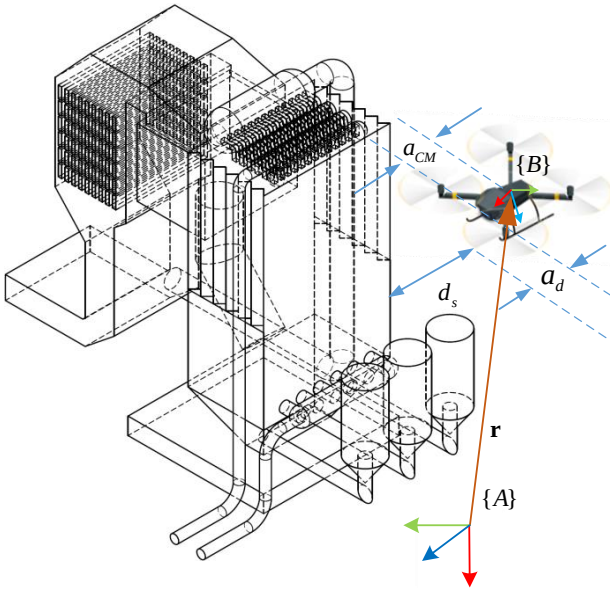


Figure 6.2: Kinematic Diagram: Frames definition and inspection distances.

$$\boldsymbol{R} = \boldsymbol{R}'(\phi)\boldsymbol{R}'(\theta)\boldsymbol{R}'(\psi)$$

$$\boldsymbol{R} = \begin{bmatrix} C_\theta C_\psi & C_\theta C_\psi & -S_\theta \\ S_\phi S_\theta C_\psi & C_\phi C_\psi + S_\phi S_\theta S_\psi & S_\phi C_\theta \\ C_\phi S_\theta C_\psi + S_\phi C_\psi & C_\phi S_\theta C_\psi - S_\phi C_\psi & C_\psi C_\theta \end{bmatrix} \tag{6.3}$$

Where C_ϕ and S_ϕ are abbreviations of $\cos(\phi)$ and $\sin(\phi)$, respectively, similarly for θ and ψ. The components of the angular velocity of the quadrotor, ω_x, ω_y and ω_z are related to the derivatives of the roll, pitch, and yaw angles as,

$$\begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = \begin{bmatrix} 1 & 0 & -S_\theta \\ 0 & C_\phi & S_\phi C_\theta \\ 0 & -S_\phi & S_\phi \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{6.4}$$

The four rotors on the quadrotor are driven by electric motors powered by electric speed controllers. The rotor speed is $\overline{\omega_i}$ and the thrust is an upward vector,

$$T_i = b\overline{\omega_i}, \quad i = 1, 2, 3, 4 \tag{6.5}$$

In the quadrotor's negative z-direction, where b >0 is the lift constant, dependant on number of blades, air density, the cube of the rotor blade radius and the chord length of the blade. Assuming the quadrotor is a symmetric rigid body, its equations of motion can be written as [14],

$$\boldsymbol{f} = \boldsymbol{f}_g + T \tag{6.6}$$

$$\boldsymbol{\tau} = \boldsymbol{J}\dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times \boldsymbol{J}\boldsymbol{\omega} \tag{6.7}$$

where $\boldsymbol{f} \in R^3$ and $\boldsymbol{\tau} \in R^3$ are the force and torque applied to the vehicle, respectively; $\boldsymbol{J} \in R^{3\times3}$ is the inertia matrix of the quadrotor, and $\boldsymbol{\omega} \in R^3$ is the angular velocity vector of the UAV, f_g is the force due to the gravity and T is the total thrust generated by the propellers. The resultant force can be then expressed as

$$\boldsymbol{f} = \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} + \boldsymbol{R} \times \begin{bmatrix} 0 \\ 0 \\ T_1 + T_2 + T_3 + T_4 \end{bmatrix} \tag{6.8}$$

Pairwise differences in rotor thrusts cause the vehicle to rotate. The rolling torque about the vehicle's x-axis is generated by the moments

$$\tau_x = a_{CM}T_4 - a_{CM}T_2 \tag{6.9}$$

a_{CM} is the distance from the quadrotor centre of mass to the rotor axis (Figure 3.6). Converting the equation in terms of rotor speeds using (6.5)

$$\tau_x = a_{CM}b(\overline{\omega_4}^2 - \overline{\omega_2}^2) \tag{6.10}$$

Similarly, for the torque about its y-axis,

$$\tau_y = a_{CM}b(\overline{\omega_1}^2 - \overline{\omega_3}^2) \tag{6.11}$$

The torque applied to each propeller by the motor needs to overcome the aerodynamic drag,

$$Q_i = k\overline{\omega_i}^2 \tag{6.12}$$

where k depends on the same factors as b. From the torque, a reaction torque on the airframe is transmitted that is responsible for the 'yaw' torque about the propeller shaft in the opposite direction to its rotation. The total torque about z-axis is,

$$\begin{aligned} \tau_z &= Q_1 - Q_2 + Q_3 - Q_4 \\ &= k(\overline{\omega_1}^2 - \overline{\omega_2}^2 + \overline{\omega_3}^2 - \overline{\omega_4}^2) \end{aligned} \tag{6.13}$$

where the signs are different due to the rotation directions (counter-clock-wise taken as positive for rotor 1 and 3) of the rotors.Therefore, yaw rotation is achieved by carefully controlling all of the 4 rotor speeds. The applied torque $\boldsymbol{\tau}$ is calculated from (6.10), (6.11), (6.13) and (6.5) obtaining

$$\begin{aligned} \boldsymbol{\tau} = \begin{bmatrix} \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} &= \begin{bmatrix} a_{CM} b(\overline{\omega_4}^2 - \overline{\omega_2}^2) \\ a_{CM} b(\overline{\omega_1}^2 - \overline{\omega_3}^2) \\ k(\overline{\omega_1}^2 - \overline{\omega_2}^2 + \overline{\omega_3}^2 - \overline{\omega_4}^2) \end{bmatrix} \\ &= \boldsymbol{J}\dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times \boldsymbol{J}\boldsymbol{\omega} \end{aligned} \tag{6.14}$$

By combining equation (6.5) and (6.8) the following expression is obtained

$$\begin{aligned} \begin{bmatrix} T \\ \boldsymbol{\tau} \end{bmatrix} &= \begin{bmatrix} -b & -b & -b & -b \\ 0 & -a_{CM} b & 0 & a_{CM} b \\ a_{CM} b & 0 & -a_{CM} b & 0 \\ k & -k & k & -k \end{bmatrix} \begin{bmatrix} \overline{\omega_1}^2 \\ \overline{\omega_2}^2 \\ \overline{\omega_3}^2 \\ \overline{\omega_4}^2 \end{bmatrix} \\ &= \boldsymbol{A} \begin{bmatrix} \overline{\omega_1}^2 \\ \overline{\omega_2}^2 \\ \overline{\omega_3}^2 \\ \overline{\omega_4}^2 \end{bmatrix} \end{aligned} \tag{6.15}$$

The moments and the forces of the quadrotor are function of the rotor speeds. The matrix $\boldsymbol{A}$ is constant and the equation can be rewritten using matrix inversion for obtaining the rotor speeds given the torques and forces [14],

$$\begin{bmatrix} \overline{\omega_1}^2 \\ \overline{\omega_2}^2 \\ \overline{\omega_3}^2 \\ \overline{\omega_4}^2 \end{bmatrix} = \boldsymbol{A}^{-1} \begin{bmatrix} T \\ \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} \tag{6.16}$$

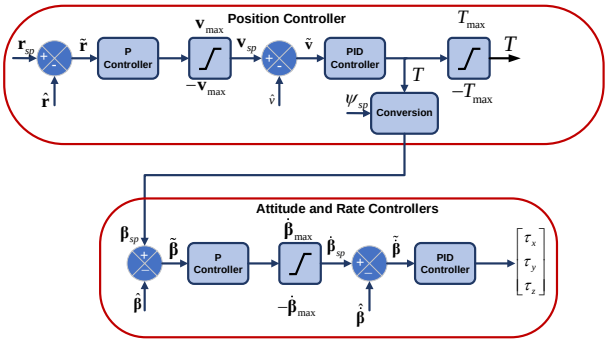


Figure 6.3: UAV controllers for position and attitude control.

6.2 UAV Control System

To achieve autonomous flight in Drones, there is usually a hierarchical scheme with three levels [19], the first is in charge of determining the flight trajectory and it is the main focus of this paper. The second is the trajectory tracking control and the third is the actuator level control. For levels two and three, we are using common control schemes. As depicted in Fig. 6.3, the UAV position and attitude are controlled by a series of controllers built-in in into the system. A position controller receives the desired coordinates to track in the position vector set-point $\boldsymbol{r}_{sp} \in R^3$ that is compared with the actual estimated drone position $\hat{\boldsymbol{r}} \in R^3$ to obtain the position error $\tilde{\boldsymbol{r}} \in R^3$ to compensate for. A Proportional controller calculates then the desired velocity $\boldsymbol{v}_{sp} \in R^3$ that is limited according to the hardware capabilities. Then, a PID controller in cascade will determine the required thrust T to achieve the desired altitude. While, through a conversion stage, the thrust and the desired yaw angle ψ_{sp} determine the desired attitude $\boldsymbol{\beta}_{sp} \in R^3$ that is compared with the

estimated attitude $\hat{\boldsymbol{\beta}} \in R^3$ to obtain the orientation error $\tilde{\boldsymbol{\beta}} \in R^3$ that will be compensated by a Proportional controller, leading to the attitude rate set point $\dot{\boldsymbol{\beta}}_{sp} \in R^3$ that at the same time is related to the estimated attitude rate $\dot{\hat{\boldsymbol{\beta}}} \in R^3$ to determine the attitude rate error $\dot{\tilde{\boldsymbol{\beta}}} \in R^3$ that will be regulated by a PID controller that generates the control torques, that together with the thrust command the drone to the desired position and orientation. Finally, the required motor's angular speed can be calculated by (6.16).

Chapter 7

Experimental Studies

7.1 Trajectory Simulation Study

Our first validation attempt for the trajectory generation system was done through simulation. For this, we employ the Gazebo simulator. This simulator uses the Pixhawk 4 (PX4) autopilot to simulate the real-world implementation of this path with a UAV. PX4 is an open-source flight stack software for controlling UAVs that implements different flight modes and safety features, and it is widely used on several drone platforms today. This software runs either on a flight controller, which is a computer on board the drone or on a personal computer in the case of a simulation. PX4 offers different interfaces for interacting with the user, all of them using the MAVLink protocol for communication: QGroundControl, a graphical interface to setup the drone, change parameters, create and execute flight missions, and visualize telemetry data in real-time; and DronecodeSDK, an Application Programming Interface (API) that provides the libraries for interacting with the UAV. Moreover, ROS is a robotics framework that abstracts common robot functionalities in a modular manner using the concept of independent programs called nodes that publish or subscribe sensor data or computational results to topics. MAVROS is one of these ROS programs for abstracting the transmission of MAVLink messages, so ROS can interact with the PX4 flight stack. Finally, Gazebo is a powerful robotics simulator that offers a 3D environment with a robust physics engine.

The chosen experimental setup consists of a general quadcopter model generated in the Gazebo simulator and running the PX4 flight stack, which shows us how a real drone would behave given the generated coordinates. For the simulation, some of the trajectory layers

were removed to reduce simulation time. This simulation validates our CAD-based method for generating trajectories, as we prove that these paths can indeed be used to generate a real UAV fly mission.

The proposed methodology consists of the following:

1. Use a ROS node to read a CSV file containing the coordinates of the waypoints that compose the generated trajectory.

2. Iterate through each point of the mission array and send position commands using the MAVROS node to follow the path using the simulated UAV in Gazebo.

3. Visualize the mission in real time using QGroundControl, while getting fly logs for an offline analysis of the UAV's trajectory and telemetry information.

The trajectory based on the power plant 3D model was used for creating the UAV's fly mission, and both the ideal generated trajectory and the actual inspection fly path followed by the drone can be compared (Figures 7.1, 7.2, & 7.3). The displayed trajectory was modified from the original trajectory by reducing the total number of layers. This was done only to make the figures easier to read and understand, the original trajectory achieves the same results with the total number of layers. We can see that the trajectories are similar to each other, but the flight path is not as smooth as the ideal trajectory, which can be explained by the inherent physical properties of a real environment and the actual robotic system dynamics. The results show an adequate flight altitude during the test, with some variation but still a monotonically increasing trend. The top view of the flight path shows the general shape of the power plant, while the front and right views of the flight path are close in shape to the generated trajectory, showing only some noise due to the flight controller corrections.

The 3D view of the trajectories shows again that the drone is capable of flying very close to the surface of the boiler. The simulation results prove that the proposed approach for generating inspection paths based on the power plant 3D model can be indeed implemented using a real UAV controlled by a widely used professional flight stack.

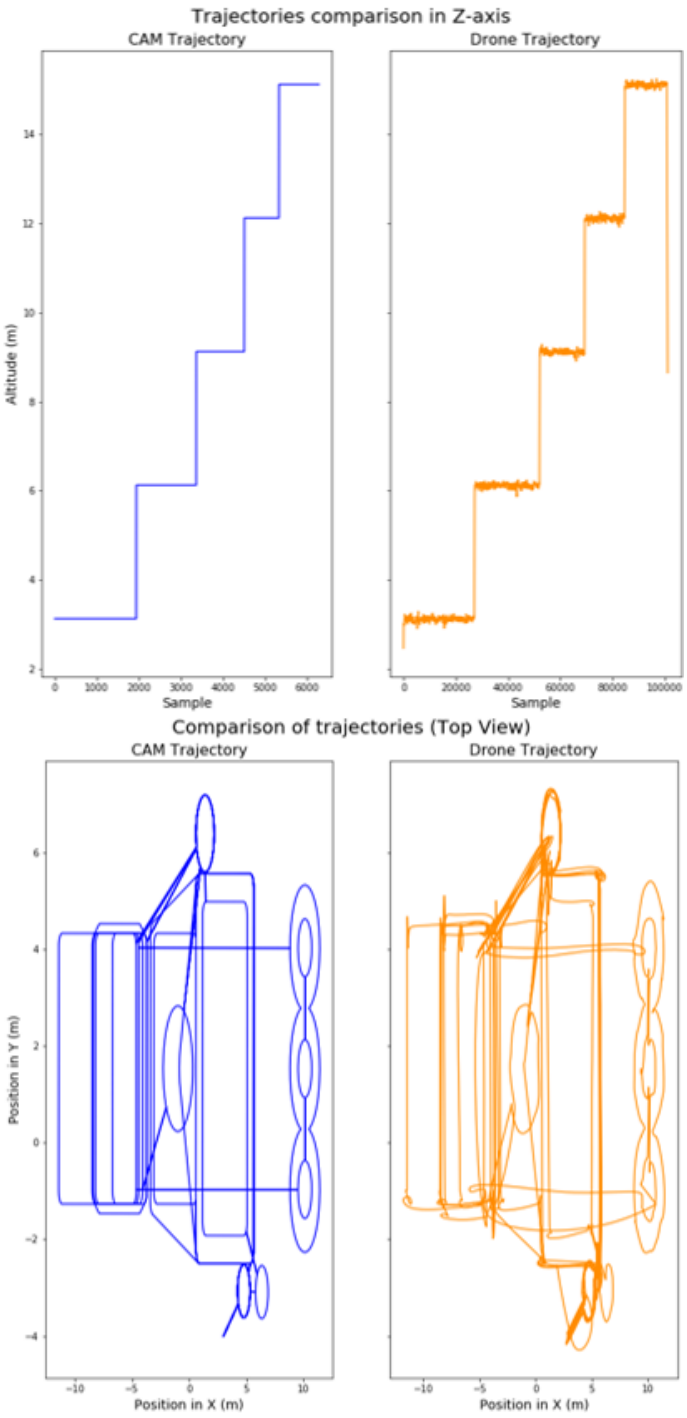


Figure 7.1: Comparison of: 1) The Z position of the tool based on the CAD and the altitude of the UAV during flight (left). 2) The top view of both the CAD generated trajectory and the UAV inspection path (right).

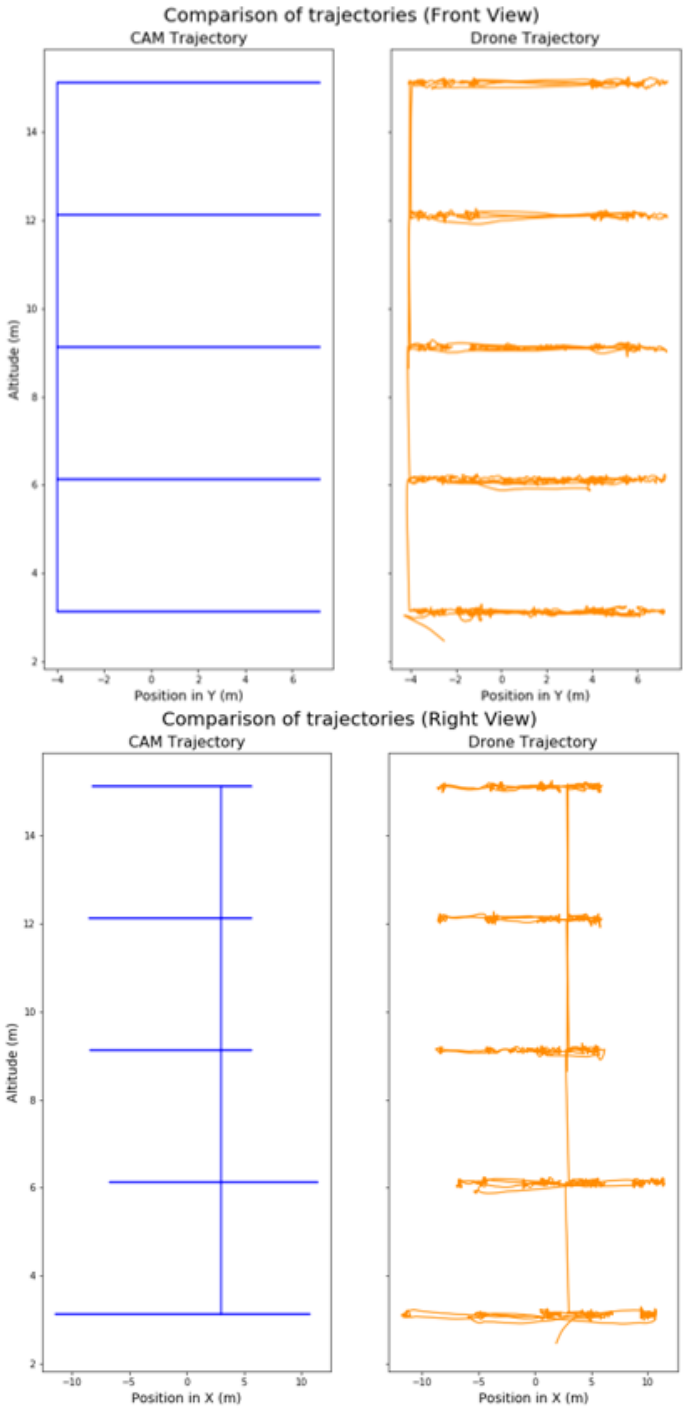


Figure 7.2: Comparison of: 1) The front view of the CAD-generated trajectory and the UAV flight path (left). 2) The right view of the CAD-generated trajectory and the UAV flight trajectory (right).

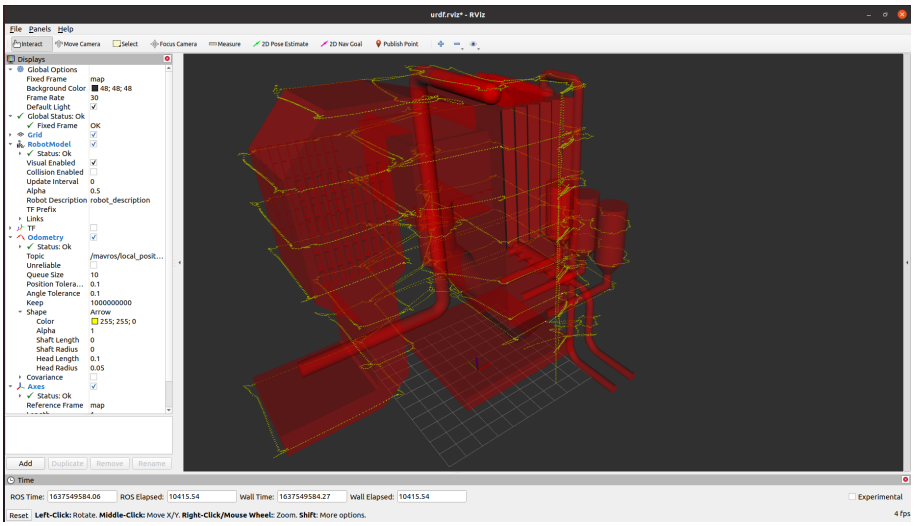


Figure 7.3: Simulated Drone Flight Trajectory

7.2 Experimental Platforms

7.2.1 Experimental Handheld Platform

We created a handheld setup containing the same Jetson Nano companion computer and some of the onboard sensors also present in the UAV platform. The purpose of this setup is to be able to accurately generate reconstructions using the sensors and also be able to test the system without the need to be flying. The handheld setup carries the Intel RealSense d435i camera along with the Raspberry Pi v2 RGB camera. The system also incorporates an Adafruit 5-inch LCD display. The top section of the structure is based on the original design by Devshank published on the hackster.io website [2].

7.2.2 Experimental UAV Platform

For the real-time implementation of the whole system, the UAV platform is based on the DJI F450 platform previously mentioned in Chapter 6. We are employing a Jetson nano as the onboard companion computer and the Pixhawk 4 flight controller. The system also uses an Intel RealSense Depth Camera D435 for mapping purposes, in the case we do not

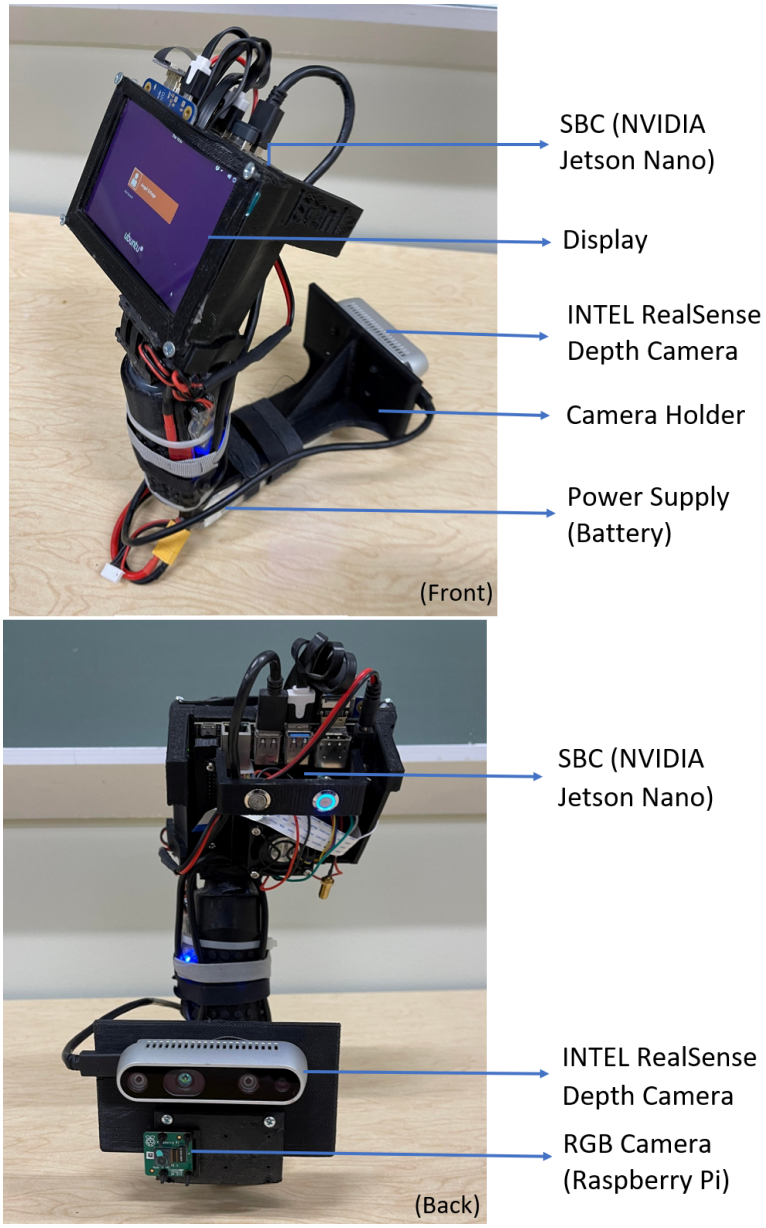


Figure 7.4: Handheld Setup Used for Visual Inspection and Experimentation

have readily available the CAD model; and an Intel RealSense Tracking Camera T265 is utilized for navigation together with a 1D LiDAR. The system also features an RGB 8 MP Raspberry Pi Camera Module V2 and a FLIR Lepton 3.5 IR (Infra Red) camera for inspection purposes. Figure 7.5 depicts the UAV with all the aforementioned components.

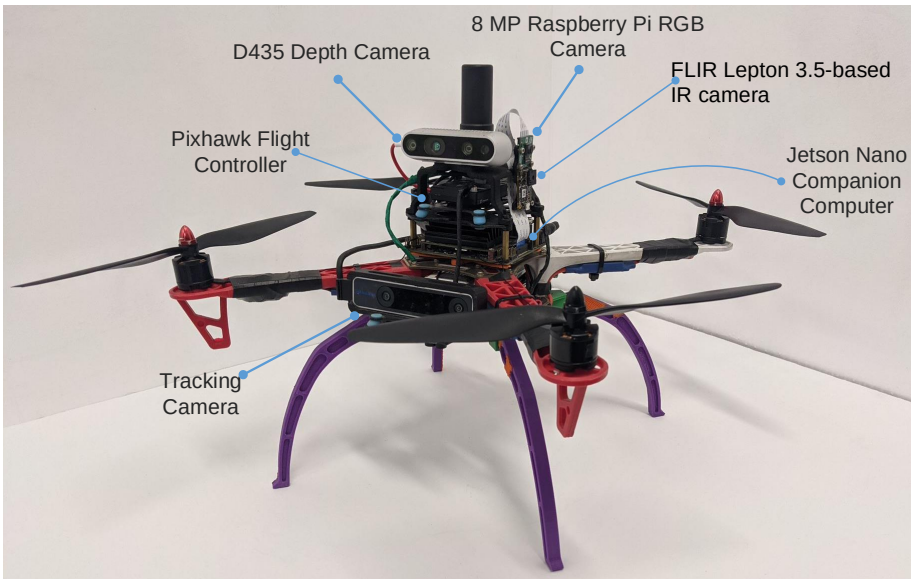


Figure 7.5: The UAV platform used in the experimental validation

7.3 Experimental Environments

7.3.1 Aerospace Center Fabens Facility

Most of our initial experimentation took place in the UTEP's Aerospace Center Technology Research and Innovation Acceleration Park located in Fabens, Texas. The presented system for trajectory generation has been successfully implemented in our UAV platform in an indoor environment. Figure 7.6 depicts the interior of the UTEP Aerospace Center Research Hangar located in Fabens, Texas.

Figure 7.6: UTEP Aerospace Center - Fabens Acceleration Park Hangar

7.3.2 El Paso Electric - Rio Grande Plant

The El Paso Electric (EPE) company allowed our team access to specific parts of their facilities to conduct experiments. We were fortunate enough to not only learn more about the energy sector and its day-to-day operations but also got access to highly guarded structures like the inside of a power unit's boiler. Figure 7.7 shows the first stack we were able to run some experiments around. Figures 7.8 and 7.9 show the outside and inside of the boiler structure we were granted access to. Figure 7.10 shows the stacks and iconic "POWER" sign on top of the original El Paso Electric building. We were allowed to reconstruct and test our systems in that area as well.

7.4 Experimental Results

7.4.1 Fabens - Validation of Reconstruction with Photogrammetry and Trajectory Generation

We employ part of the research hangar for our experimental flights for testing and tuning of our UAV. We generated the reconstruction with photogrammetry methods. A set of images

Figure 7.7: EPE Rio Grande - Individual Stack

Figure 7.8: EPE Rio Grande - Power Unit - Outside of Boiler Furnace

Figure 7.9: EPE Rio Grande - Power Unit - Inside of Boiler Furnace

Figure 7.10: EPE Rio Grande - Main Building - Roof Stacks and POWER Sign

of the environments were taken with slight overlap within one another. Those images were then imported into AliceVision MeshRoom to create the reconstruction. MeshRoom provides an STL file that can then be transferred to any CAD software to clean up by removing unwanted sections of the reconstruction before transferring them to MATLAB. Figure 7.11 depicts the reconstructed hangar using photogrammetry.

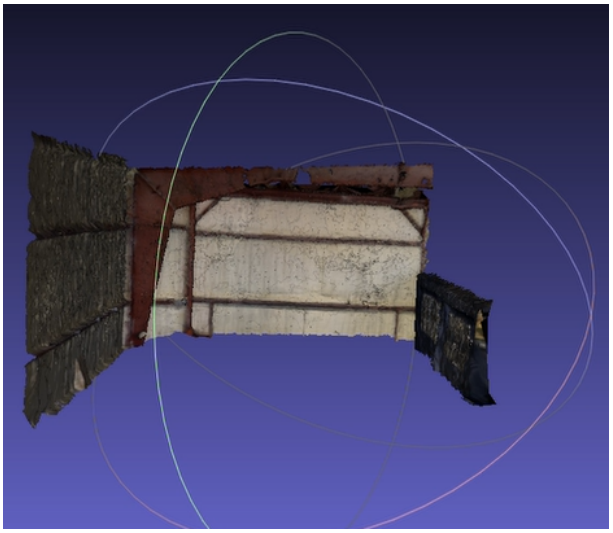


Figure 7.11: UTEP Aerospace Center - Fabens Acceleration Park Hangar - 3D Model Generated with Photogrammetry

Once the STL is in MATLAB, the algorithm completes the flight path generation and stores it as a CSV file. Figure 7.12 depicts the trajectory generated for this environment and Figure 7.13 depicts the UAV flying the generated trajectory.

We are able to log the sensor reading of the drone at all times during the flight mission. Figure 7.14 depicts the log of the mission in x, y, z-coordinates. Figure 7.15 depicts the log in x, and y-coordinates only, while Figure 7.16 depicts the logs in z-coordinates vs time.

Note that in Figure 7.15 there is an area where the drone does not behave as stable as the rest of the flight path. The area in the lower left of the path is the place where the layer jump location takes place, at this point the drone must increase the thrust in order to reach a higher altitude, this translates to some unexpected turbulence. This is the reason

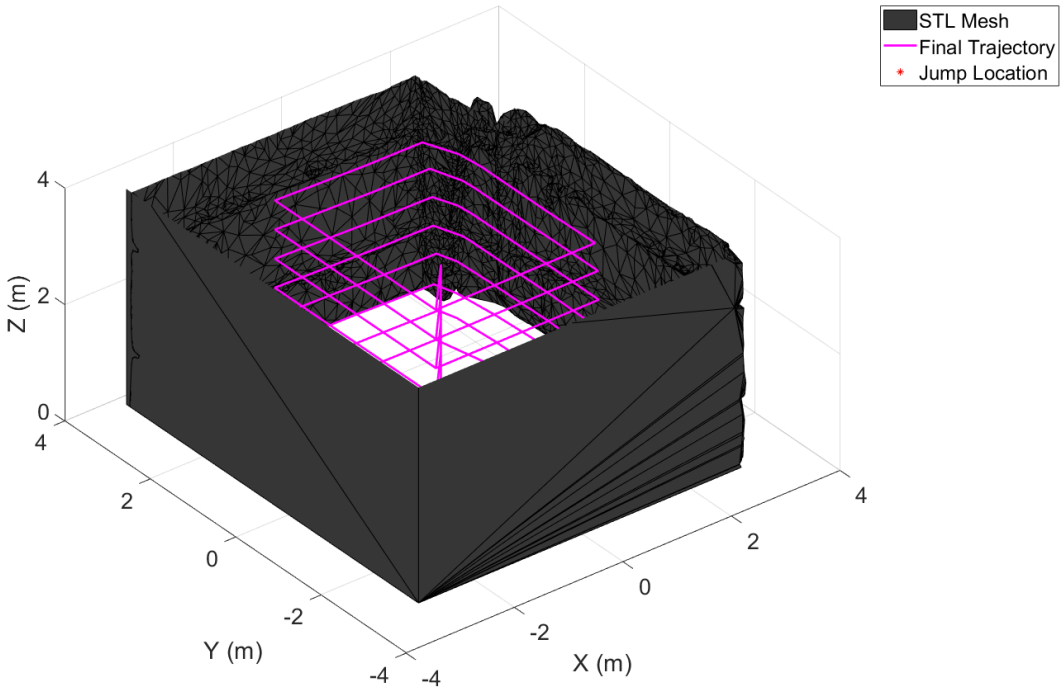


Figure 7.12: Hangar inspection path generated with the proposed approach

Figure 7.13: Drone flying the generated trajectory

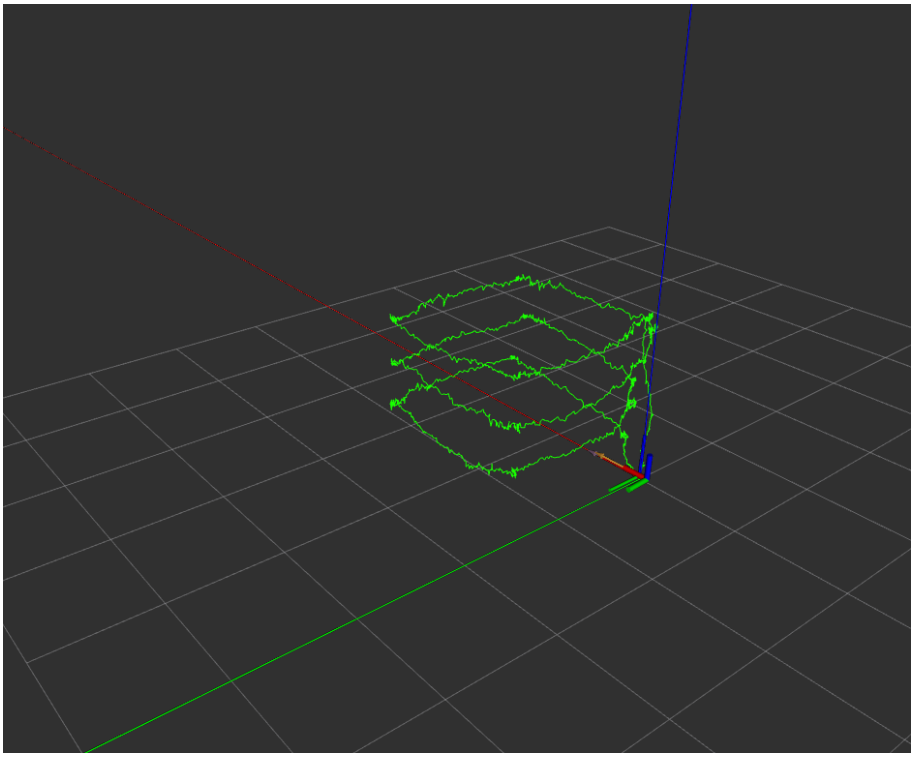


Figure 7.14: Logged Drone Position in x, y, and z-coordinates

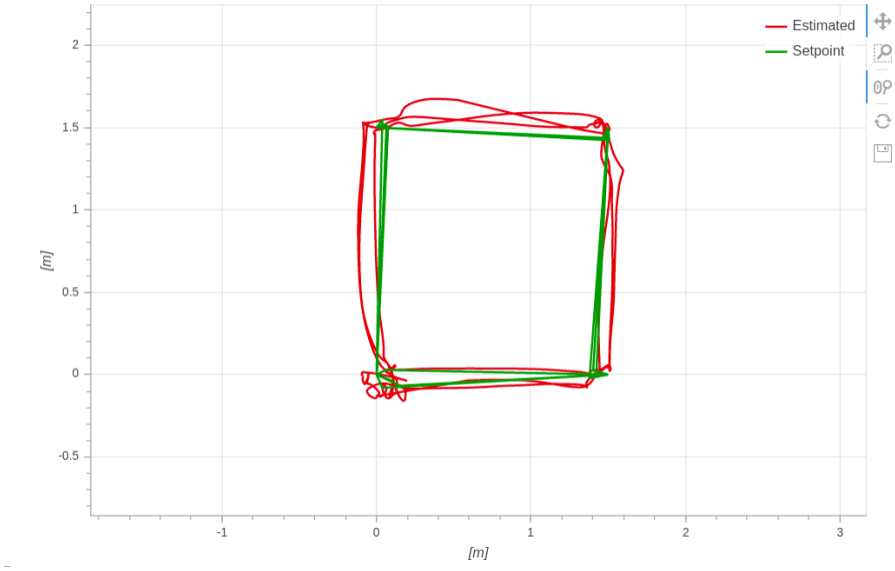


Figure 7.15: Logged Drone x-y trajectory

this jump location coordinate must be identified individually for each structure.

During the flight test discussed above, Figure 7.13, our system always kept track of the yaw set points, so a set of inspection images were acquired. These images had a frame rate of 2Hz, more than enough for obtaining a good overlap between consecutive images, and the timestamp was marked for further processing, see Figure 7.17.

Figure 7.16: Logged Drone Altitude

Figure 7.17: Captured Inspection Images Normal to the Structure Surface

7.4.2 El Paso Electric - Validation of Reconstruction in Industrial Settings with Photogrammetry and RTAB-Map

RTAB-Map - Single Stack - Ground Level

While we were in El Paso Electric, were able to generate a reconstruction of a single cooling stack on the ground level of the facilities. This was our initial test environment and helped us validate some of our systems. We were able to show some of our initial results to the El Paso Electric (EPE) representatives which gave them confidence and allowed further access to their facilities. Figure 7.18 shows the folder containing the images captured to generate a reconstruction using photogrammetry. For this experiment we also generated the reconstruction using AliceVision's Meshroom. Figure 7.19 shows the reconstructed model of the stack. After the model was scaled, we were able to calculate different measurements within the environment and they matched the actual measurements within 0.5 inch.

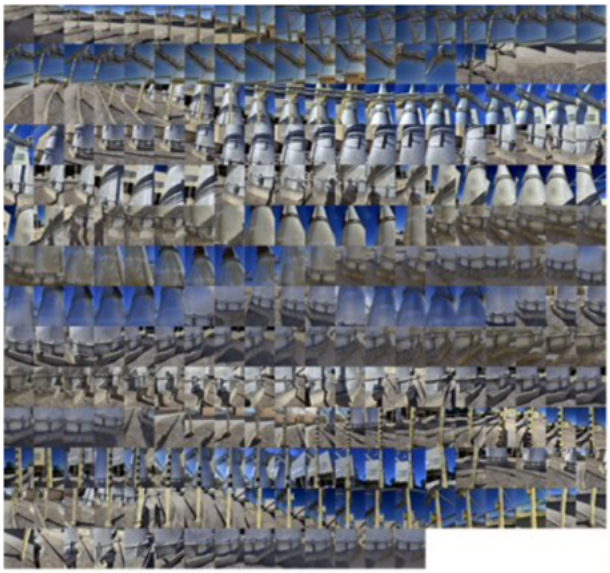


Figure 7.18: EPE - Captured Images of Stack for Photogrammetric Reconstruction

Figure 7.20 shows the reconstructed model displayed as an STL within Fusion 360. It is important to note that although the photogrammetric reconstruction gives good results in large features with enough detail, it tends to have a hard time with thin structures. It

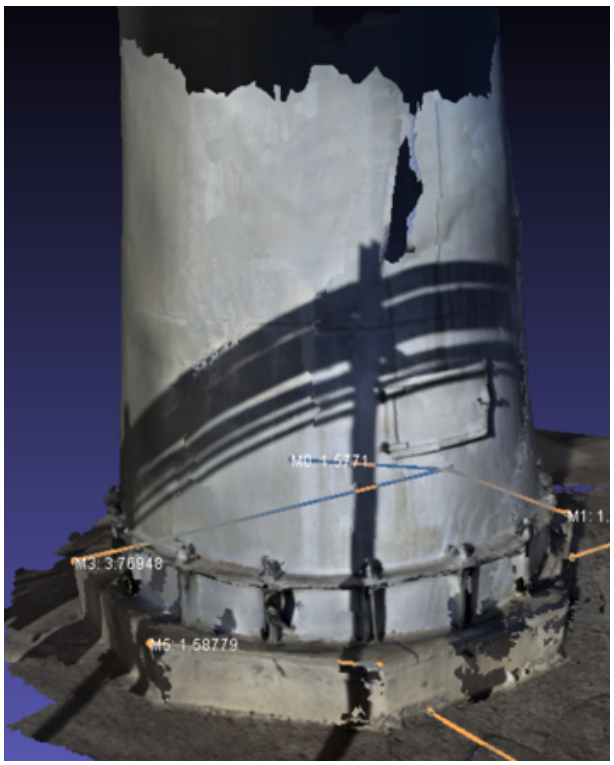


Figure 7.19: EPE - Photogrammetric Reconstruction of Single Stack

can be seen in Figure 7.20 that the pipe structures in front of the stack are only partially reconstructed. A last post-processing step would be required in this case to complete the pipe in the CAD and clean up the rest of the STL before sending it to our trajectory generation algorithm to generate a close-quarter inspection trajectory.

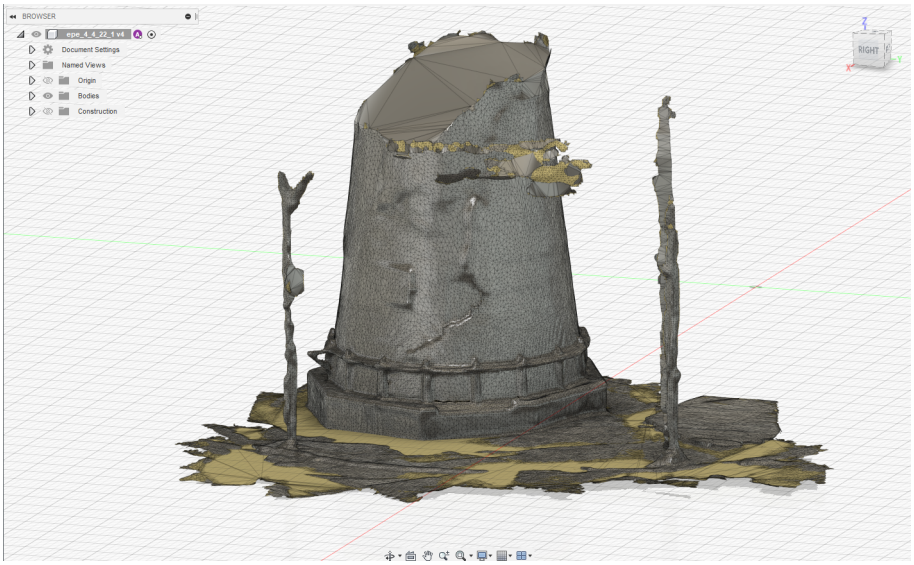


Figure 7.20: EPE - STL Model of the Reconstructed Stack

RTAB-Map - Inside Boiler Furnace

We were able to generate reconstructions in different environments while at the El Paso Electric Rio Grande power plant. One of the most interesting environments was the inside of the boiler of one of the power units. The environment proved to be one of the most complex our systems encountered for multiple reasons. The inside of the structure is covered in dirt, ash, rust, and other debris. Also, the inside of the structure is pitch dark since the entire area is closed off to any outside ventilation and lighting. Figure 7.21 shows the only entrance to the boiler. This makes it a hard environment to reconstruct with the current sensor capabilities. We were provided spotlights to illuminate the inside of the structure but this was still a difficult environment. Figure 7.22 depict the environment inside the boiler. In that image, we were setting up the drone and testing the sensors inside

the structure. Part of those sensors were the RGB and depth cameras we employ for the reconstructions.

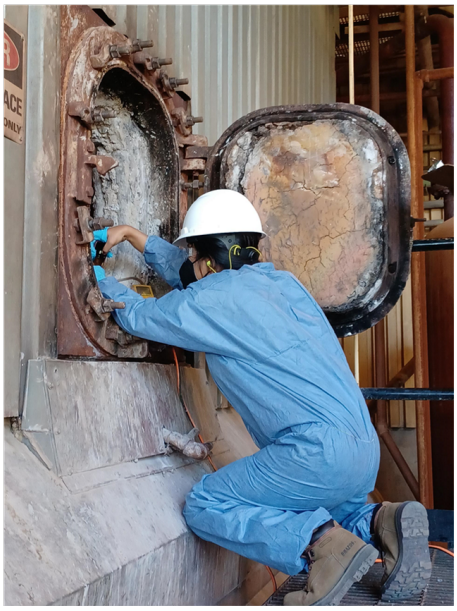


Figure 7.21: EPE - Boiler Entrance

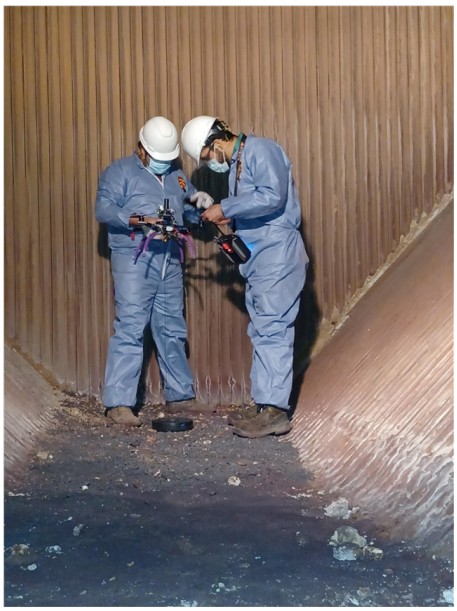


Figure 7.22: EPE - Boiler Environment

We were able to generate some reconstruction on the inside using the RTAB-Map systems in both the handheld platform and the iPhone. Figures 7.23 and 7.24 depict one of the reconstructions of the inside of the boiler. Both reconstructions are the same but one has the color enabled for each pixel while the other has the color disabled. Although we also did some flight tests in an attempt to capture more images at altitude for the reconstruction we were unable to improve our results. This was mainly due to how hard it was for our drone platform to fly correctly and localize itself within the inside surfaces. It was difficult because of the lighting but also because of the debris particulates that would fly as soon as the propellers began spinning. Also, due to the boiler unit having to go back online after the planned maintenance had concluded, we were not able to get access to the boiler again.

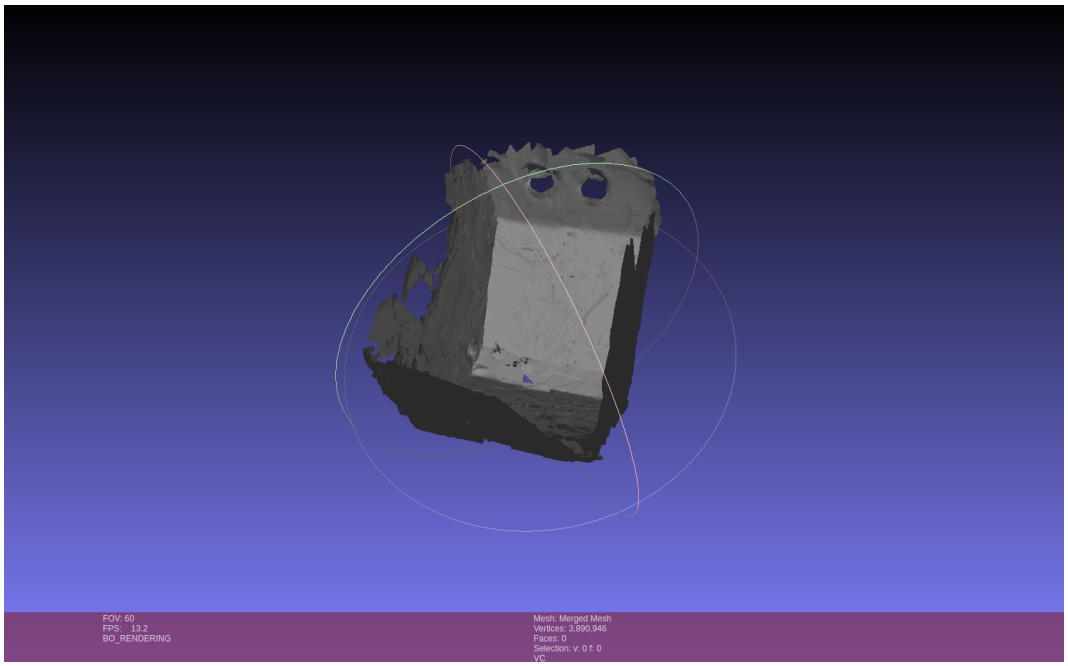


Figure 7.23: EPE - RTAB-Map Reconstruction of Inside of a Boiler - Color Disabled

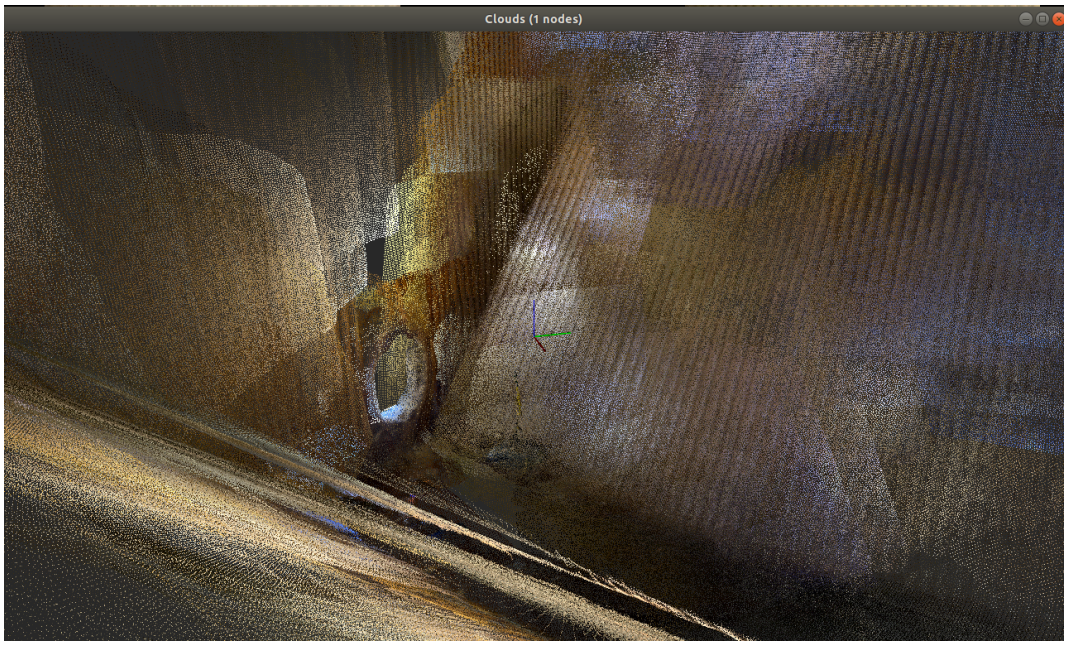


Figure 7.24: EPE - RTAB-Map Reconstruction of Inside of a Boiler - Color Enabled

RTAB-Map - Rooftop Stacks Initial Multi-Session Map

For these experiments, we decided to validate the data-collection system from the drone platform with the map created of the rooftop environment. The map was created with multiple session maps merged into one. Figures 7.25 and 7.26 depict the main map along with the trajectories traveled to obtain the individual point clouds. Those individual point clouds were collected with a variety of setups, which include the LiDAR sensor from an iPhone 12 Pro running RTAB-Map, a handheld Jetson payload running RTAB-Map ROS with an Intel RealSense D435 depth camera, and finally the same payload but on the drone flying. We flew the UAV to make sure it could localize itself within this environment. The red trajectory next to the yellow trajectory visible in Figure 7.26 depicts a trajectory the drone flew in localization mode. The structure that is displayed in these figures in only the point cloud data captured by the sensors and reconstructed by the RTAB-Map system.

The resulting reconstructed model can be seen in Figure 7.27. This model could then be saved as an STL model and sent to the trajectory generation algorithm. Note that the

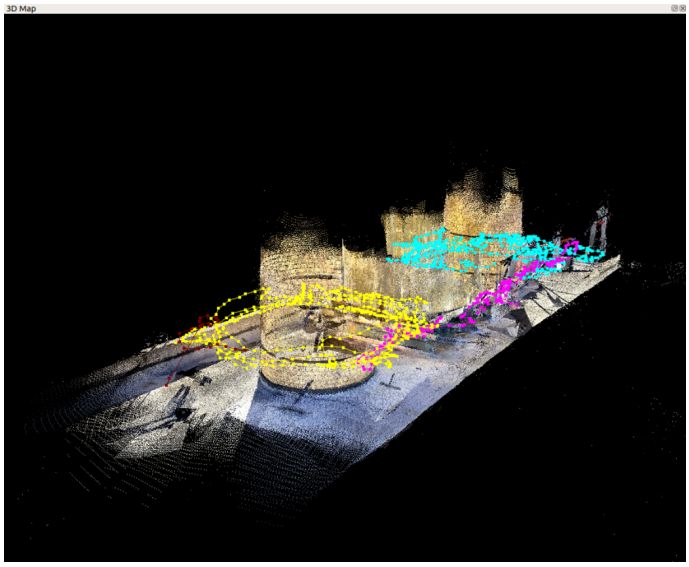


Figure 7.25: Initial Map of Rooftop Environment with Visible Mapping Trajectories (West Heading)

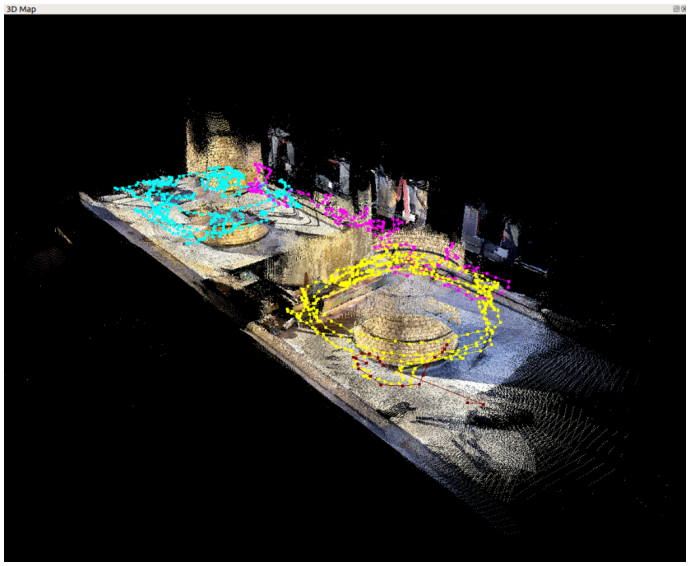


Figure 7.26: Initial Map of Rooftop Environment with Visible Mapping Trajectories (North Heading)

reconstructed model is made up of all the individual inspection paths depicted in Figures 7.25 and 7.26.

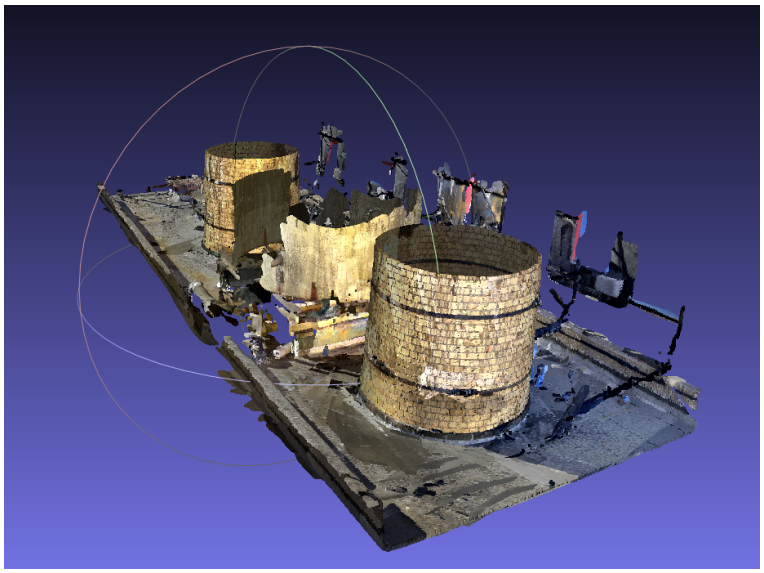
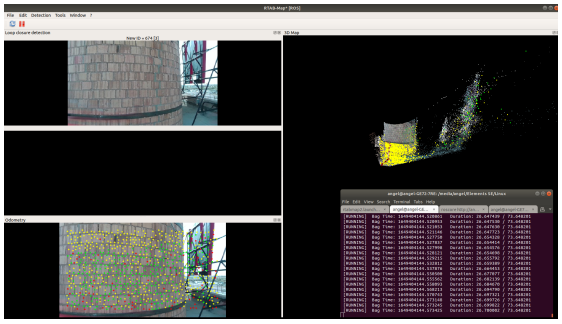


Figure 7.27: Reconstructed Model of Rooftop Environment

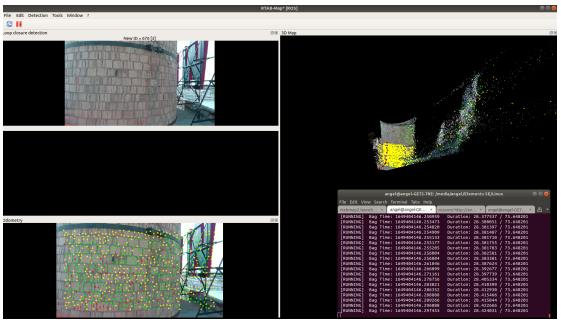
Mapping of New Section of Stack at Altitude

Our last test after seeing that the drone could localize itself within the pre-flight environment was to test if the drone could then map a new section of the environment that had not been mapped before. A decision was made to send the UAV in a vertical path along the entire height of one of the stacks. Figure 7.28 shows the gradual mapping of the UAV with RTAB-Map Ros. Note that the reconstruction was complete offline after the UAV stored the sensor data in a bagfile.

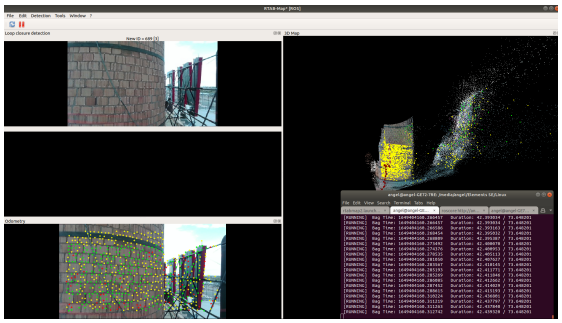
Figure 7.29 depicts the trajectory traveled by the drone with the mapping output at altitude. Here, the system will use odometry and the data of the previous map to first localize itself and then continue mapping. The red line on the right image depicts the path traveled by the drone to create this map. Figure 7.30 shows the UAV flying the same path. Figure 7.31 shows the final map obtained after the mission is complete and RTAB-Map



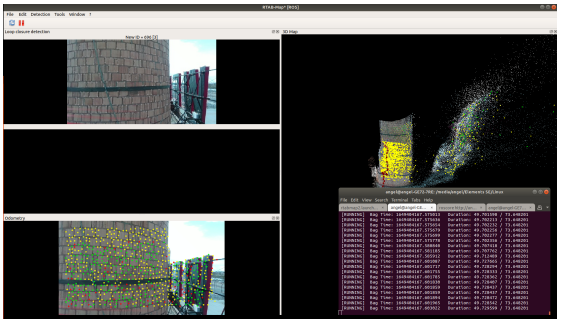
(a) 26 Second Time Stamp



(b) 28 Second Time Stamp



(c) 42 Second Time Stamp



(d) 49 Second Time Stamp

Figure 7.28: RTAB-Map ROS Running in Mapping Mode

has merged the original map with the new section mapped by the Drone. Some post-processing was done in MeshLab to clean up the model. After this step, the model could then be converted into any mesh file for further use like STL for our trajectory generation algorithm.

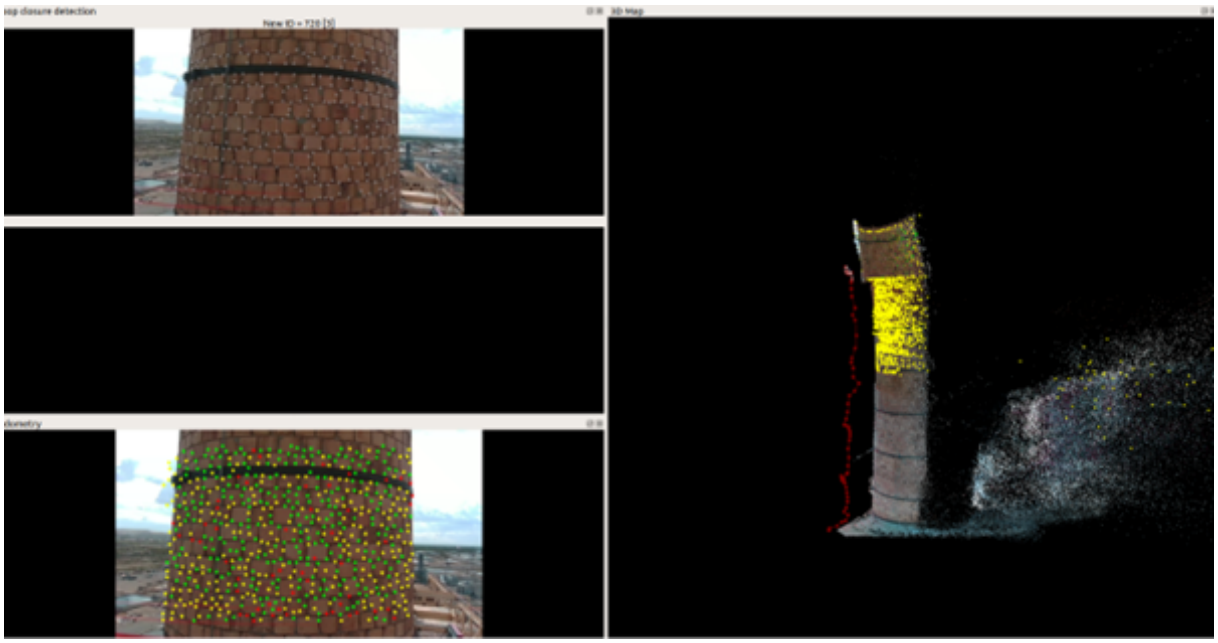


Figure 7.29: RTAB-Map in Mapping Mode from Drone Flying at Altitude

Figure 7.30: UAV Collecting Data at Altitude

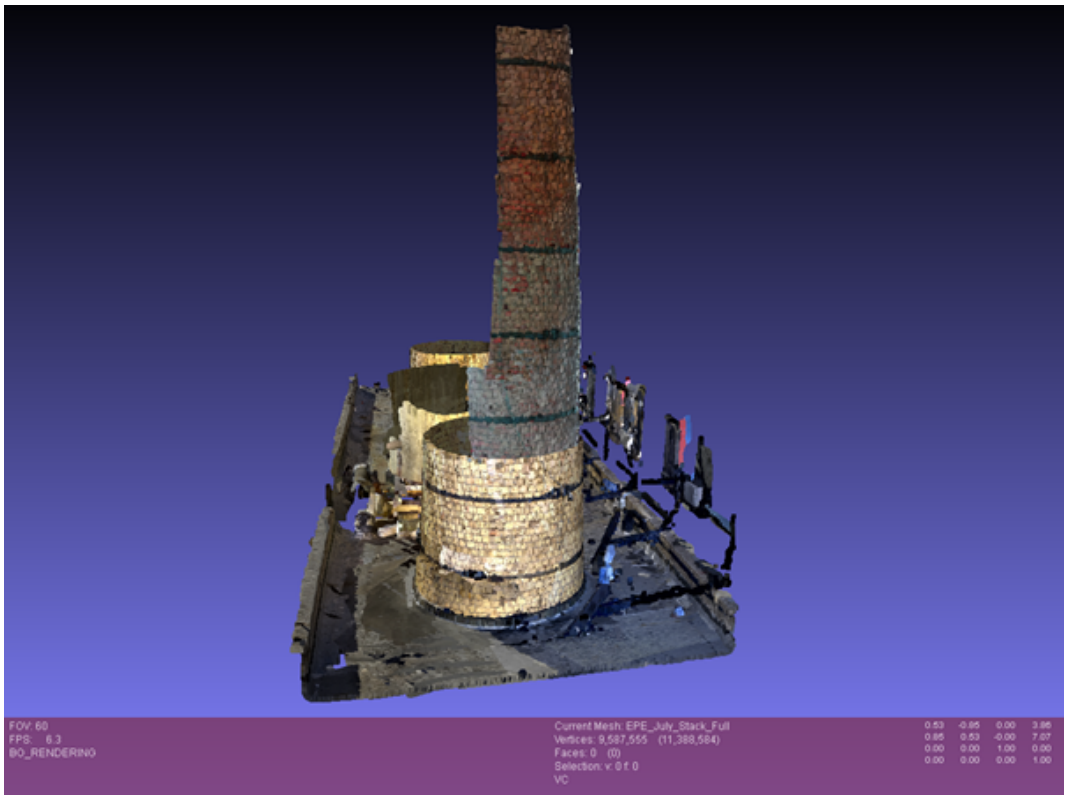


Figure 7.31: Final Map Displayed in MeshLab After Post-Processing

Chapter 8

Conclusions

8.1 Summary

This study presents a means to do close-quarter inspections of structures by using an existing CAD model with a UAV system that does not require GPS to complete its mission. Doing inspections using robotic technology allows workers to not be involved in hazardous environments while saving time and money thus being more efficient. This system can also be complemented with most of the existing developments for collision avoidance making it suitable for almost any mission thinkable ranging from energy to defense applications. The use of Artificial Intelligence provides a means to efficiently classify and segment inspection images to deliver to human inspectors and maintenance crews. Also, structures that lack initial CAD models could be reconstructed in an efficient manner in order to employ our trajectory generation algorithm.

8.1.1 Trajectory Generation

The offline trajectory generation method we present allows the UAV platform to reach virtually all surfaces within a structure. The system also takes into account any overhangs that might be present within the structure and the algorithm uses vertical layer comparison to account for such features and prevent collisions. Furthermore, the system employs a closest-neighbor approach for connecting subsequent outlines within a layer thus resulting in a single pass through each trajectory point. This yields the most efficient approach to travel to all those trajectory points in the shortest and fastest route. In scenarios

where obstacles that were not accounted for are present in the trajectory, the drone must operate with a simultaneous obstacle detection system to maneuver around such obstacles. Once the UAV transmits all the inspection data to the ground station computer, the CAD model could be updated offline with the reconstruction technologies available to us and the trajectory could be calculated one more time. This means that after each inspection the trajectory could be updated to account for any differences between the CAD model and the structure itself. The trajectory system was tested and validated at the cSETR's Technology Research and Innovation Acceleration Park research hangar located in Fabens, Texas. Those results can be seen in Chapter 7.

8.1.2 Defect Detection

The use of Artificial Intelligence in the form of Convolutional Neural Networks proves beneficial to process the large amounts of inspection data collected by such inspections. Our system has the capability to not only classify the defects in real time but also apply image segmentation to the defect images offline. This is beneficial to the operators since it allows the defects to be highlighted. The defect data can then be visualized along with the defect location. This would provide a user-friendly defect report for maintenance crews.

8.1.3 3D Reconstruction

While testing our trajectory generation system at the Fabens Facility we identified the need to reconstruct structures and environments in order to use our system. We identified multiple approaches but settled on two main approaches, each employing different sensors in our platform. Photogrammetry provides a decent reconstruction but must be scaled after the model is generated. Scaling the model is trivial and the only requirement is a comparison between two points of interest in both the actual structure and the reconstructed model. The photogrammetry results depend heavily on the quality of the images provided by the software and the lighting conditions. The second approach we identified and pursued is the

use of the RTAB-Map library which employs the depth information from a depth or LiDAR camera. This approach yields the best results but is computationally expensive. This approach must be executed either in the ground station computer or in the iPhone app which has been fine-tuned to run correctly with the hardware characteristics of the phone. The results from either of the two systems are compatible with the other. This means that even though you can initiate a reconstruction in the phone platform, you can continue mapping in the Jetson platform adding to the initial map. Both approaches provided positive results for the reconstruction under different conditions. The results can be seen in Chapter 7. The Fabens research hangar model was reconstructed using the photogrammetry approach while the El Paso Electric reconstruction was completed with the RTAB-Map approach and the use of the iPhone lidar sensor and the Intel depth camera.

8.2 Future Work

The long-term goal of this work is to make the trajectory generation algorithm available for further advancement by other investigators and to increase its current applications. We plan on upgrading the algorithm to allow the selection of individual components within the CAD assembly to be selected and inspected individually. We also hope to embed the CAD and the planned trajectory to the reconstruction algorithms used to have a better localization framework for the point cloud reconstruction effort. We hope to upgrade to a more powerful onboard computer to allow the reconstruction, defect detection, and localization to happen at the same time within the flight missions. Although defect detection has been refined using other frameworks by other team members, we hope to continue working with AlexNet and eventually generate a user-friendly interface to post-process the inspection data. We want to continue working in capturing high quality images from the inspection camera. We hope by doing this we could then integrate that new data into a reconstruction using photogrammetry and compare the resulting reconstructions from point cloud data and from images. This could help further refine the reconstructed model to have as much detail

present as possible. We hope this technology could help other industries as well.

References

[1] Alicevision. https://alicevision.org/#. Accessed: 2021-5-3.

[2] Jetscan. https://www.hackster.io/devshank/jetscan-16a521. Accessed: 2021-09-30.

[3] Rtab-map. http://introlab.github.io/rtabmap/. Accessed: 2021-12-10.

[4] Rtab-map in ros. https://www.theconstructsim.com/robotigniteacademy_learnros/ros-courses-library/rtab-map-in-ros-101/, note = Accessed: 2021-12-6.

[5] Francisco Agüera-Vega, Fernando Carvajal-Ramírez, Patricio Martínez-Carricondo, Julián Sánchez-Hermosilla López, Francisco Javier Mesas-Carrascosa, Alfonso García-Ferrer, and Fernando Juan Pérez-Porras. Reconstruction of extreme topography from uav structure from motion photogrammetry. *Measurement*, 121:127–138, 2018.

[6] F Farnood Ahmadia, MJ Valadan Zoeja, H Ebadia, and M Mokhtarzadea. The application of neural networks, image processing and cad-based environments facilities in automatic road extraction and vectorization from high resolution satellite images. *The international archives of the photogrammetry, remote sensing and spatial information sciences*, 37:585–592, 2008.

[7] Haruhiko Harry Asada, Anirban Mazumdar, Ian C. Rust, and Jun Fujita. Apparatus and method of wireless underwater inspection robot for nuclear power plant., September 9 2019. US Patent 10,421,192 B2.

[8] T. Bailey and H. Durrant-Whyte. Simultaneous localization and mapping (slam): part ii. *IEEE Robotics & Automation Magazine*, 13(3):108–117, 2006.

[9] Andrew C Brown and Deon De Beer. Development of a stereolithography (stl) slicing and g-code generation algorithm for an entry level 3-d printer. In *2013 Africon*, pages 1–5. IEEE, 2013.

[10] Michael Burri, Nikolic Janosch, Christoph Hurzeler, Gilles Caprari, and Roland Siegwart. Aerial service robots for visual inspection of thermal power plant boiler systems. In *International Conference on Applied Robotics for the Power Industry (CARPI), Switzerland*, pages 70–75, 2012.

[11] Young-Jin Cha, Wooram Choi, and Oral Büyüköztürk. Deep learning-based crack damage detection using convolutional neural networks. *Computer-Aided Civil and Infrastructure Engineering*, 32(5):361–378, 2017.

[12] Brodie Chan, Hong Guan, Jun Jo, and Michael Blumenstein. Towards uav-based bridge inspection systems: A review and an application perspective. *Structural Monitoring and Maintenance*, 2(3):283–300, 2015.

[13] Ismael Colomina and Pere Molina. Unmanned aerial systems for photogrammetry and remote sensing: A review. *ISPRS Journal of photogrammetry and remote sensing*, 92:79–97, 2014.

[14] P. Corke. *Robotics, Vision and Control, Fundamental Algorithms in MATLAB*. Springer Berlin Heidelberg, 2011.

[15] Hainan Cui, Shuhan Shen, Wei Gao, Hongmin Liu, and Zhiheng Wang. Efficient and robust large-scale structure-from-motion via track selection and camera prioritization. *ISPRS Journal of Photogrammetry and Remote Sensing*, 156:202–214, 2019.

[16] Mathaus Ferreira da Silva, Leonardo M Honório, Andre Luis M Marcato, Vinicius F Vidal, and Murillo F Santos. Unmanned aerial vehicle for transmission line inspection using an extended kalman filter with colored electromagnetic interference. *ISA transactions*, 100:322–333, 2020.

[17] Kesse Jonatas de Jesus, Henry Julio Kobs, Anselmo Rafael Cukla, Marco Antonio de Souza Leite Cuadros, and Daniel Fernando Tello Gamarra. Comparison of visual slam algorithms orb-slam2, rtab-map and sptam in internal and external environments with ros. In *2021 Latin American Robotics Symposium (LARS), 2021 Brazilian Symposium on Robotics (SBR), and 2021 Workshop on Robotics in Education (WRE)*, pages 216–221. IEEE, 2021.

[18] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: part i. *IEEE robotics & automation magazine*, 13(2):99–110, 2006.

[19] Daniel C Gandolfo, Lucio R Salinas, Mario E Serrano, and Juan M Toibero. Energy evaluation of low-level control in uavs powered by lithium polymer battery. *ISA transactions*, 71:563–572, 2017.

[20] Carsten Griwodz, Simone Gasparini, Lilian Calvet, Pierre Gurdjos, Fabien Castan, Benoit Maujean, Gregoire De Lillo, and Yann Lanthony. Alicevision meshroom: An open-source 3d reconstruction pipeline. In *Proceedings of the 12th ACM Multimedia Systems Conference*, pages 241–247, 2021.

[21] Lei He, Nabil Aouf, and Bifeng Song. Explainable deep reinforcement learning for uav autonomous path planning. *Aerospace Science and Technology*, 118:107052, 2021.

[22] San Jiang, Cheng Jiang, and Wanshou Jiang. Efficient structure from motion for large-scale uav images: A review and a comparison of sfm tools. *ISPRS Journal of Photogrammetry and Remote Sensing*, 167:230–251, 2020.

[23] Christine A Jones and Elizabeth Church. Photogrammetry is for everyone: Structure-from-motion software user experiences in archaeology. *Journal of Archaeological Science: Reports*, 30:102261, 2020.

[24] N. Kawauchi, S. Shiotani, H. Kanazawa, T. Sasaki, and H. Tsuji. A plant maintenance humanoid robot system. In *IEEE International Conference on Robotics and Automation, Vol. 3, IEEE, Taipei, Taiwan*, pages 2973–2978, 2003.

[25] Ho Moon Kim, Kyeong Ho Cho, Fengyi Liu, and HyoukRyeol Choi. Development of cable climbing robotic system for inspection of suspension bridge. In *International Symposium on Automation and Robotics in Construction*, pages 1422–1423, 2011.

[26] Philipp Krähenbühl, Carl Doersch, Jeff Donahue, and Trevor Darrell. Data-dependent initializations of convolutional neural networks. *arXiv preprint arXiv:1511.06856*, 2015.

[27] Mathieu Labbé and François Michaud. Rtab-map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation. *Journal of Field Robotics*, 36(2):416–446, 2019.

[28] Shengyuan Li and Xuefeng Zhao. Image-based concrete crack detection using convolutional neural network and exhaustive search technique. *Advances in Civil Engineering*, 2019, 2019.

[29] Mario Michelini and Helmut Mayer. Structure from motion for complex image sets. *ISPRS Journal of Photogrammetry and Remote Sensing*, 166:140–152, 2020.

[30] Raul Mur-Artal and Juan D Tardós. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE transactions on robotics*, 33(5):1255–1262, 2017.

[31] J. Nikolic, M. Burri, J. Rehder, S. Leutenegger, C. Huerzeler, and R. Siegwart. A uav system for inspection of industrial facilities. In *IEEE Aerospace Conference, Montana*, pages 1–8, 2013.

[32] Felix Nobis, Odysseas Papanikolaou, Johannes Betz, and Markus Lienkamp. Persistent map saving for visual localization for autonomous vehicles: An orb-slam 2 extension.

In *2020 Fifteenth International Conference on Ecological Vehicles and Renewable Energies (EVER)*, pages 1–9. IEEE, 2020.

[33] Angel Ortega, Julio Reyes Muñoz, Michael McGee, Ahsan R Choudhuri, and Angel Flores-Abad. Drone inspection flight path generation from 3d cad models: Power plant boiler case study. In *AIAA Scitech 2020 Forum*, page 1091, 2020.

[34] Fei-wei Qin, Lu-ye Li, Shu-ming Gao, Xiao-ling Yang, and Xiang Chen. A deep learning approach to the classification of 3d cad models. *Journal of Zhejiang University SCIENCE C*, 15(2):91–106, 2014.

[35] Tarek Rakha and Alice Gorodetsky. Review of unmanned aerial system (uas) applications in the built environment: Towards automated building inspection procedures using drones. *Automation in Construction*, 93:252–264, 2018.

[36] Ivan Reljić, Ivan Dunđer, and Sanja Seljan. Photogrammetric 3d scanning of physical objects: Tools and workflow. *TEM Journal*, 8(2):383, 2019.

[37] Julio A. Reyes-Munoz and Angel Flores-Abad. A mav platform for indoors and outdoors autonomous navigation in gps-denied environments. In *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, pages 1708–1713, 2021.

[38] Mousumi Rizia, Angel Ortega, Julio Reyes Muñoz, Michael McGee, Ahsan R Choudhuri, and Angel Flores-Abad. A cam/am-based trajectory generation method for aerial power plant inspection in gps-denied environments. In *AIAA Scitech 2020 Forum*, page 0858, 2020.

[39] Mousumi Rizia, Julio A Reyes-Munoz, Angel G Ortega, Ahsan Choudhuri, and Angel Flores-Abad. Autonomous aerial flight path inspection using advanced manufacturing techniques. *Robotica*, pages 1–24, 2022.

[40] H. T. Roman. Robotic applications in PSE&G's nuclear and fossil power plants. In *IEEE Transactions on Energy Conversion Conference Vol. 8, No. 3*, pages 584–592, 1993.

[41] Tianshu Ruan, V Amrusha Aryasomyajula, and Nasser Houshangi. Performance of monocular and stereo camera in indoor environment for visual slam using orb method. In *2022 IEEE International Conference on Electro Information Technology (eIT)*, pages 273–278. IEEE, 2022.

[42] J Savall, Alejo Avello, and Leoncio Briones. Two compact robots for remote inspection of hazardous areas in nuclear power plants. In *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C)*, volume 3, pages 1993–1998. IEEE, 1999.

[43] M. Shan, F. Wang, F. Lin, Z. Gao, Z. Tang, Y., and M. Chen, B. Google map aided visual navigation for uavs in gps-denied environment. In *IEEE International Conference on Robotics and Biomimetics (ROBIO),Zhuhai, China*, pages 114–119, 2015.

[44] Amit Shukla and Hamad Karki. A review of robotics in onshore oil-gas industry. In *2013 IEEE International Conference on Mechatronics and Automation*, pages 1153–1160. IEEE, 2013.

[45] Amit Shukla, Huang Xiaoqian, and Hamad Karki. Autonomous tracking of oil and gas pipelines by an unmanned aerial vehicle. In *2016 IEEE 59th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 1–4. IEEE, 2016.

[46] Kevin J Wu, Thomas Stan Gregory, Julian Moore, Bryan Hooper, Dexter Lewis, and Zion Tsz Ho Tse. Development of an indoor guidance system for unmanned aerial vehicles with power industry applications. *IET Radar, Sonar & Navigation*, 11(1):212–218, 2017.

[47] S Yamamoto. Development of inspection robot for nuclear power plant. In *IEEE International Conference on Robotics and Automation, Vol. 2, IEEE, Nice, France*, pages 1559–1566, 1993.

[48] Suguru Yokoyama and Takashi Matsumoto. Development of an automatic detector of cracks in concrete using machine learning. *Procedia engineering*, 171:1250–1255, 2017.

[49] Xu Zhang, Bin Xian, Bo Zhao, and Yao Zhang. Autonomous flight control of a nano quadrotor helicopter in a gps-denied environment using on-board vision. *IEEE Transactions on Industrial Electronics*, 62(10):6392–6403, 2015.

[50] Qing Zhu, Zhendong Wang, Han Hu, Linfu Xie, Xuming Ge, and Yeting Zhang. Leveraging photogrammetric mesh models for aerial-ground feature point matching toward integrated 3d reconstruction. *ISPRS Journal of Photogrammetry and Remote Sensing*, 166:26–40, 2020.

Curriculum Vitae

Angel Guillermo Ortega Castillo was born on August 27, 1994, in Mexico City, Mexico. The fourth son of Salvador Ortega Oropeza and Rosa Isela Castillo Hernandez, he graduated from J.M. Hanks High School, El Paso, Texas, in the spring of 2012. He entered The University of Texas at El Paso in the fall of 2012. He received his bachelor's degree in Mechanical Engineering in the spring of 2016, achieving CUM LAUDE and receiving the Mechanical Engineering Department's Academic Performance Award. For this, he was invited to complete his Doctorate at The University of Texas at El Paso by the then Chair, Dr. Ahsan Choudhuri.

In the fall of 2016, he entered the Graduate School of The University of Texas at El Paso. While pursuing his doctorate degree in Mechanical Engineering he worked as a Teaching and Research Assistant. He instructed over 2,200 students over a span of 5 years, first as teaching assistant lead instructor and then as instructor of record. He participated in research funded by the Department of Energy which employed the use Unmanned Aerial Vehicles for inspection purposes. He is a member of the Pi Tau Sigma UTEP Chapter Honor Society.

Contact information: agortega@miners.utep.edu
orte94oros@gmail.com
Permanent address: 8441 Lasso Cir., El Paso, Texas 79907