OREGON STATE UNIVERSITY

HONORS THESIS

3D Specialty Crop Model Rendering with the Xbox Kinect v2

Author:
Nick ALVEY

Supervisor:
Dr. Joseph DAVIDSON

A thesis submitted to Oregon State University Honors College in partial fulfillment of the requirements
for the degree of Honors Bachelors of Science

Honors Scholar

Presented March 5, 2019
Commencement June 2019

February 27, 2019

Declaration of Authorship

I, Nick ALVEY, declare that this thesis titled, "3D Specialty Crop Model Rendering with the Xbox Kinect v2" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

OREGON STATE UNIVERSITY

Abstract

Dr. Joseph Davidson
Honors Baccalaureate of Science in Mechanical Engineering presented on March 7, 2019

Honors Bachelors of Science

3D Specialty Crop Model Rendering with the Xbox Kinect v2

by Nick ALVEY

A relatively new field of robotics, robotics for automated harvesting and pruning, has gained recognition in the last 10 years. Several studies have been conducted in the harvesting and pruning of various plant types, including citrus fruits, various pepper families, cucumbers, grapes, and more. One of the most critical components of a robot is its arm's ability to reach and manipulate the selected target with proper control and dexterity. To optimize the effectiveness of the arm, it is extremely helpful to run optimization simulations of different arm shapes, sizes, and allowable rotation and extension in an attempt to execute their function in a world-like environment. One method of creating this world-like environment is to scan crops with a depth camera and create a virtual 3D model of them. The project objective was to create a virtual environment in which optimization simulation can be run. A library of various plant models including raspberries (red and black), blueberries, and blackberries of various kinds was created using a low-cost Xbox Kinect v2. From all research conducted into similar endeavors, an attempt to use a low-cost sensor of this kind to create usable, 3D simulation models of complex specialty crops was not found. Scanning techniques were compared between data collections, with results showing a marked improvement in scan density (between 3.1 and 14.7 times) and mesh quality (up to 50x higher resolution) when scanning sections of a crop closely, compared to scanning the entire crop from a larger distance. It was concluded that, despite greatly improved model rendering in the second data set, the Kinect did not produce a model with sufficient feature accuracy for simulation testing of harvesting or pruning techniques. Despite this, however, it was recommended that certain models be used in general dexterity simulations as they presented good surface approximations of the crop.

Acknowledgements

I would like to thank Professor Joe Davidson for giving me the opportunity to develop this project and a special thanks to those working at the North Willamette Research and Extension Center for allowing me to collect data from their crops. I would also like to thank my fiance, family, and friends for encouraging, pushing, and supporting me through this process...

Contents

List of Figures

List of Abbreviations

GDP	**G**ross **D**omestic **P**roduct
LTR	**L**imb to **T**runk **R**atio
2D	**T**wo **D**imensional
3D	**T**hree **D**imensional
LIDAR	**LI**ght **D**etection **A**and **R**anging
TOF	**T**ime **O**f **F**light
IR	**I**nfra - **R**ed
RGB-D	**R**ed **G**reen **B**lue - **D**epth Camera
DoF	**D**egrees **o**f **F**reedom
ROS	**R**obot **O**perating **S**ystem
VREP	**V**irtual - **R**obot **E**xperimentation **P**latform
NWREC	**N**orth **W**illamette **R**esearch and **E**xtension **C**enter

Chapter 1

Introduction

1.1 Need for Automation in Agriculture

The average consumer knows the importance of agriculture and the provision of household crops like corn, potatoes, apples, grapes and other common foods. This is reinforced in the fact that a 12.9% share of the sum of household expenditures were spent on food in 2017. Economically, agricultural production makes up a significant portion of the U.S. Gross Domestic Product (GDP), contributing $992 billion in 2015, or about 5.5%. Farms specifically contributed $136.7 billion, or about 1% to this overall value. By 2017 the farming industry employed about 2.6 million people, 1.3% of the overall U.S. employment [1].

A large portion of agricultural production - about 42% of the variable cost - is labor [2]. Many of these positions are filled by seasonal and migrant laborers, a group that has been in significant decline over the past decade [3]. Labor is availability is generally uncertain, and its costs have been steadily rising. Additionally, more stringent labor regulations have been implemented as of late, as working environment safety regulations continue to improve. This inhibits the labor output allowed per worker [4]. This is coupled with a steady increase in crop production [2] and an even more marked growth in specialty crop production and consumption [5]. This has led to a growing disparity between required production from farmers and a shrinking pool of labor.

1.1.1 Automation for Crop Production

To address the disparity between production demands and labor available, several automation attempts have been made in various agricultural environments. For example, there have been many noticeable improvements in automated tractor guidance. These improvements focus on allowing tractors to autonomously localize crop-rows to allow for intelligent tillage, planting, cultivating, and splaying [6, 7, 8]. There have also been endeavors into the use of drones to monitor crop growth and quality [9], sophisticated chemical compounds and automated distribution methods of herbicides and pest control [10], and automated shake and catch mechanisms for harvesting [4].

While the push for automation in crop quality and production has been steady, it has become apparent that automation in specialty crops has lagged behind that of other agricultural endeavors. For clarification, specialty crops are defined as "fruits and vegetables, tree nuts, dried fruits, horticulture, and nursery crops (including floriculture)", and are generally represented by fruits (with a categorization of tree-fruit and non tree-fruit), nuts, vegetables,

and herbs and spices [11]. This is due largely to the complexities in the harvesting and maintaining of the crop, which includes the need for seasonal human labor to execute difficult, time intensive processes, and a found shortage of laborers to conduct these tasks [2, 4, 6, 12].

Despite these complexities, there have been endeavors into both automated harvesting and pruning of specialty crops, including into the automated harvesting of cucumbers, apples, and capiscum and sweet peppers, as well as automated pruning of grape vines [13, 14, 15, 16, 6]. These endeavors have provided an insight into many of the specific challenges that roboticists face when developing the technology to automate these tasks. Challenges are divided into two main categories: feature detection and robot ability to execute the pruning or harvesting technique.

Detecting the desired feature is considered the greatest challenge to the automated harvesting and pruning industry. Initial attempts struggled to accommodate variability of crops within a certain crop type and challenging lighting conditions. This led to significant inaccuracies in produce detection. One such endeavor by Vision Robotics used a canopy to shut out sunlight in favor of their controlled light, while other attempts only functioned at night to avoid sunlight. However, due to advancement in camera technology and computer vision algorithms, there have been significant improvements in feature detection [14].

To address the challenge of harvesting deformable, soft produce, some endeavors have created novel end-effectors to imitate a worker's hand. In the harvesting of sweet peppers, the roboticists designed a coupled suction and grasping end-effector to ensure a secure grasp without puncturing the soft outer layer of the sweet pepper [15]. It was found that in the apple-harvesting industry, certain technique is utilized to break the apple off of the tree from the stem by a certain grasping and twisting technique based on the length/size of the stem. To address the difficulty of grasping an apple efficiently, the roboticists designed a robot that could detect stem length of the apple and an end-effector that could effectively break off the apple using the technique of a professional harvester [17].

To address the challenge of pruning, different end-effector kinds have been tested. One such experiment tested an end-mill style end-effector and found it provided difficulties by often ineffectively cutting branches and requiring a 10-cm-long motion to cut, a significant distance in the densely packed environment of use. This led to difficulties in creating no-collision paths. Due to these challenges, a secateur - shear styled - end-effector was recommended due to its compact size and ability to cut exactly in the target area [14]. As one expects, however, the design of a specific end-effector is highly dependent on the environment in which it will be utilized.

Within the apple industry, there have been movements to accommodate robotic harvesting and pruning, as 22% of the total production variable cost of an apple orchard is attributed to pruning costs and 30% to harvesting costs [18]. These movements have been both in standardizing crop layout and initializing robotic endeavors in apple harvesting and pruning. Crop training systems have been adopted to restrict rootstock and branch shapes and sizes, leading to a smaller, simplified canopy. This also reduces the variability in canopy shape and position. Fruit systems are ideally trained to a two-dimensional (2D) planar wall allowing easier access to sunlight, and has been doubly beneficial in its allowance of simplified robotic vision and detection algorithms. Additionally, standardization of pruning criteria has been implemented through the use of a limb-to-trunk ratio (LTR) specification. Endeavors using machine vision and Lidar and camera-based detection schemes have made progress, however there have been no machines developed for the robotic pruning of tree fruit crops [6].

1.2 Optimization of a Pruning or Harvesting Robot Through Simulation

In the design of both harvesting and pruning robots, there is a need for optimization for commercial viability of proposed solutions. One emerging way of optimization is the creation of a simulation space in which large studies can be conducted to test variations of the robot's features. This optimization simulation is initiated by creating three-dimensional (3D) models of the produce to be harvested or pruned and inserting the designed robot to virtually execute its path planning and pruning or harvesting technique. This allows many scenarios to be tested with no added cost to production and in relatively little time in comparison to physical testing.

There are several software packages available to do this kind of optimization testing. Common ones include Virtual - Robot Experimentation Platform (V-REP), Gazebo, and ARGoS. Each has its own strengths and weaknesses as it pertains to usability with other programs, CPU usage to conduct simulation, built-in models, and various other features [19]. In the undertaking of this project, only V-REP and Gazebo were considered due to their ability to work with the software Robot Operating System (ROS), the software used as the basis of this project. These have been compared for their viability in various scenarios, and determined that Gazebo provides easier integration into ROS, while V-REP contains more fully designed models to be added directly into the simulation world [20].

Due to the general design of a harvesting or pruning robot, the feature that has the most variation, and can cause the most dramatic change in function, is the robotic arm. For this reason, this is the main focus of optimization in these endeavors [13, 16].

1.2.1 Testing Degrees of Freedom

To optimize a robot arm's harvesting or pruning ability, an objective function is used to combine the two desired characteristics of the robot arm: its ability to reach a certain location in its workspace, and the manipulation possible in that target location, defined as dexterity. These are expressed as functions of robotic arm geometries, including possible joint angles, arm lengths, and changing arm lengths (in the case of prismatic joints). Results are found by testing different robot arm styles through the course of their possible orientations within a certain scenario and the objective function plotted. This has provided very useful information for the endeavors in which they are used, displaying direct better/worse comparisons and providing a sound basis for optimization [13, 16].

To properly run these scenarios, it is important to import life-like models into the simulation. The most common way this is done is through the use of a depth sensor of some kind (camera or other) and recreating the crop as a 3D model in a simulation world.

1.3 Scan-based Model Creation Techniques

The use of machine vision has been prevalent in agriculture for several decades, and there have been many endeavors to recreate 3D models of crops. For example, in the apple industry, attempts have been made to use 3D modeling techniques for branch detection for pruning. To do this, roboticists will employ the use of cameras or Light Detection and Ranging (LIDAR) sensors. LIDAR sensors work by sending focused beams of light to illuminate

its surroundings and measuring the return to recreate a map of its surroundings. It uses three factors to determine position:

1. The time difference between the output of light and the return.
2. The angle at which the pulse was sent.
3. The absolute location of the LIDAR sensor [21].

LIDAR sensors have been used for applications in tree reconstructions, branch length studies, and diameter identification [6].

Similarly, cameras, specifically RGB-D sensors, provide not only color feedback of the scene (RGB - red, green, and blue, the primary colors of light), but also a depth measurement. These are significantly lower priced than LIDAR and are divided based on their image acquisition method, which is specified as either passive or active. Passive methods use reflected natural light on a given target to measure its shape, while active methods will use an external lighting source that provides additional shape information. One type of active sensor is a time-of-flight (TOF) sensor, which measures depth by estimating the time delay from light emission to light detection. Others, like structured-light sensors, will combine projected light patterns with a standard 2D camera and measure depth via triangulation [22, 23]. These cameras have been used increasingly in projects related to tree reconstruction, produce and branch detection, and many other applications in agriculture due to their increasing availability. [6].

1.3.1 Common Software

The data collected by RGB-D sensors, is displayed as a series of points containing color and depth information, referred to as a point cloud. Many programs have been developed to manipulate point clouds, and one software that allows such manipulation is called Robot Operating System (ROS). ROS is a flexible framework that allows users to write robot software and multilingual communication across platforms [24, 25]. This is useful as it allows programs written in different programming languages to run together.

Another common software is Meshlab, which is an open-source system designed for processing and editing 3D triangular meshes. It provides a set of tools for editing, cleaning, healing, inspecting, rendering, texturing and converting meshes [26]. Meshlab allows the user to stitch the points gathered in a point cloud to create a meshed solid model. It also allows for RGB data to be stored such that color, texture, and measured size are maintained throughout the process. There are several methods to create 3D models from point clouds in Meshlab [27, 28, 29]. These techniques utilize Meshlab's surface meshing features, texturing features, and more.

1.3.2 Limitations of These Processes

While this is a robust method of model rendering and optimization testing, there are potential limitations to the accuracy and effectiveness of a study of this kind. The main shortcoming is that the simulation can only be as accurate as the models inputted into the system. If either the model of the robot or the crop is too simple, does not have distinguishable features, or is inaccurate to a known standard, the simulation will be ineffective. Therefore, the quality of model rendering is very important.

There are many potential limitations to scan quality including the the quality of camera or LIDAR used (correlating to the density of point cloud generated), the ambient light in the scene which can create shadows or glare, and calibration inaccuracies in the camera or LIDAR. Within this project, calibration was conducted under ideal lighting, as the program would not operate under non-ideal conditions (insufficient or too much ambient light, direct light on the camera or target, non-flat target surface, etc.). This left some unknowns in the effectiveness of the calibration in the field.

Within Meshlab there are limitations in the accuracy of the meshing and face rendering, which is dictated by the density of point cloud. Since Meshlab generates a best-fit mesh, there are always some inaccuracies, even if only miniscule, as the process is inherently just an approximation. Finally, these methods can be quite computationally expensive, sometimes requiring very long run times and powerful computers to run. This has been partially addressed by limiting the ranges of variation that are tested [15, 13].

Chapter 2

Process Initialization, Setup, and Initial Testing

2.1 Initialization of Xbox Kinect v2 in ROS

For this project, the Xbox Kinect v2 was utilized within ROS for data capture. The Kinect v2 was chosen because of it was on-hand, has been effective in other modeling endeavors [22, 30, 31], and has a great literature base (the Web of Science displays over 2000 publications involving the Kinect). While there are numerous options for operating an Xbox Kinect v2 for the creation of point clouds, ROS was used because of its robust support and literature, and because it allows multilingual communication. Python was used (from personal preference) as the language to develop the point cloud manipulation program. The program was loosely based on the Open Source Calibration Vision Library (OpenCV) process of saving and manipulating point clouds [32]. OpenCV is an open-source software that has C++, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. OpenCV was designed for computational efficiency and with a strong focus on real-time applications [33]. Gazebo was chosen as the simulation software for this project.

2.1.1 Interfacing With the Kinect v2

There were two packages utilized for this process. One package, called libfreenect2, was used to download and install the proper drivers to interface with the Kinect camera. It was first developed in January of 2016 and is open-sourced [34]. It allows direct connection with the Kinect v2 in Windows, MacOS, and Linux, and publishes a live-stream of the Kinect color and depth outputs.

The other package, also open-sourced, is called iai_kinect2 and was used to publish the camera streams into ROS [35]. This package was developed by a researcher at the Institute for Artificial Intelligence at the University of Bremen. It includes a bridge from libfreenect2 to ROS, calibration tools for the Kinect, and a viewer for the generated point clouds. It is operated by running a bridging node that initializes the program and the Kinect. The viewer can then be used and details of the Kinect including the framerate and serial number are shown. This can be viewed in Figure 2.1.

Utilization of these packages allows ROS to interpret the camera streams as nodes that will publish camera information. The nodal view of the Kinect's information streams between the bridge and the viewer can be seen in Figure 2.2.

This publishing style allows freedom for the user to create their own programs to manipulate output from the Kinect. Since ROS is multilingual, the programmer can use any

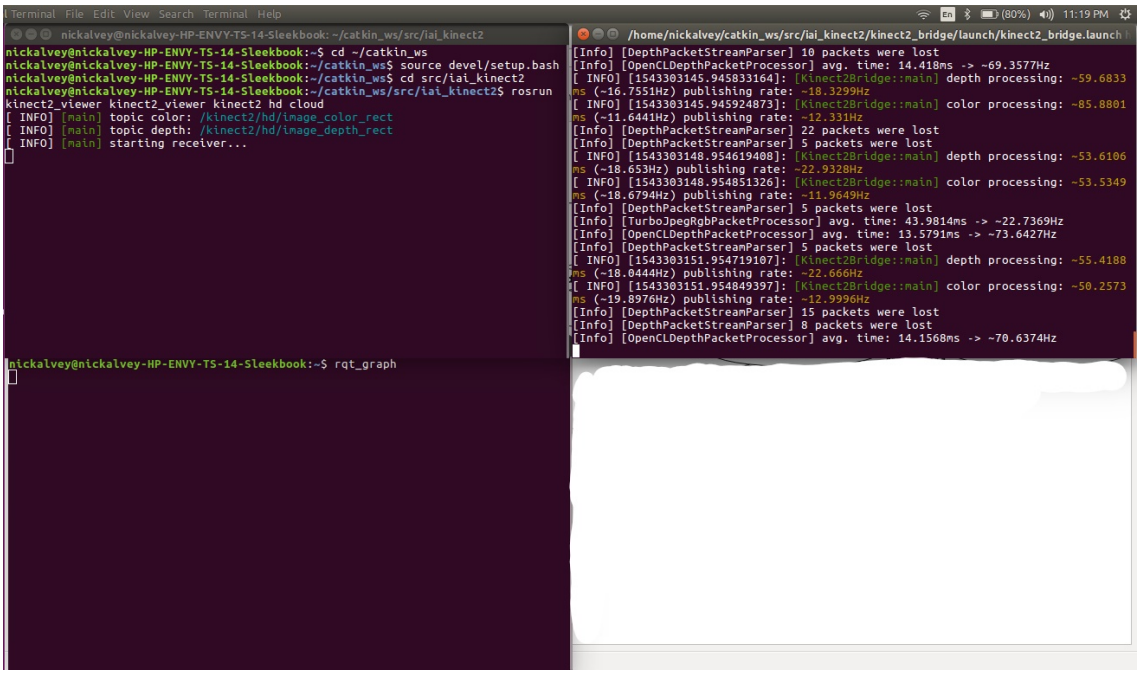


FIGURE 2.1: The viewer node and launch node running in conjunction using the aforementioned packages.

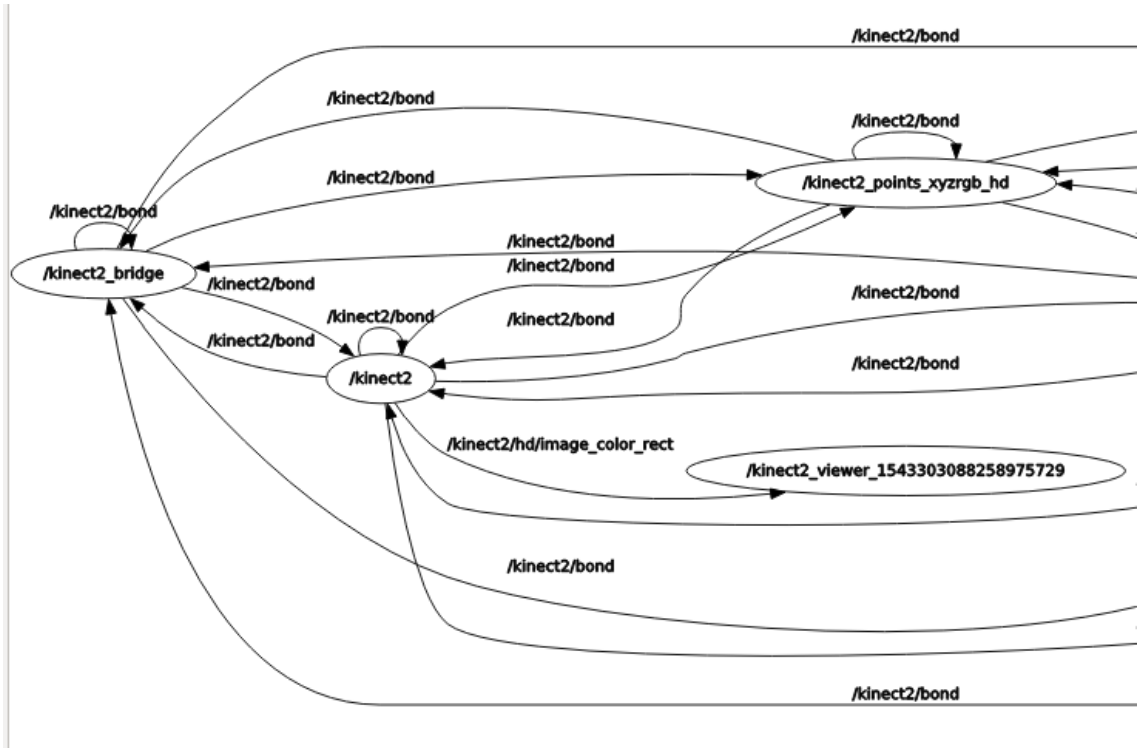


FIGURE 2.2: A depiction of the nodes of camera information publishing to and from the Xbox Kinect v2 and the viewer.

language that is convenient. For this reason, Python was used to complete the point cloud storage aspect of the project.

2.1.2 Depth and RGB Image Capture

As mentioned previously, the Kinect data is presented in a point cloud. Without manipulation, the pixels are returned in respect to their position in the 512x424 matrix of the Kinect camera without regard to their actual X,Y, and Z position in space. An established way to overcome this is to utilize the known focal properties (displayed in Figure 2.3) and the known camera matrix of the Kinect to create a projected position of the pixel (displayed in Figure 2.4). It is worth noting that in this method c_x and c_y are the principal points of the camera, f_x and f_y are the focal lengths of the camera, quantified as pixel units, and u and v are the pixel location in the 512x424 pixel matrix [22].

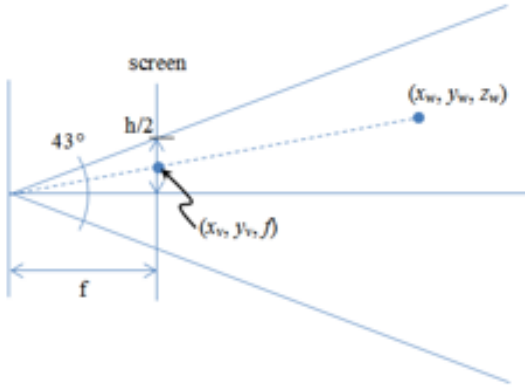


FIGURE 2.3: A display of the focal view of the Xbox Kinect v2.

2.1.3 Implementation of Calibration and Filtration Methods

For any use of a depth and RGB camera, calibration is vitally important to ensuring the accuracy of a model. The uncalibrated Kinect v2 contains distortion due to the offsets between the two on-board cameras (the IR camera and color camera). Calibration techniques are run to layer the depth measurements over perceived depth measurements, as well as measure color offsets and IR camera offsets. One common technique is to use a simple image of known size and geometry and compare the known dimensions to the calculated results by the camera. Generally, several scans are taken of the image and a standard offset calculated [37, 38]. One method is to use a checkered image as shown in Figure 2.5.

This method was developed by OpenCV [38], and maps both edge distortion and tangential distortion parameters in the camera. This is done by finding the edges of the overall image and corners in the checkered images. Since the distances between the corners of the checkers are known, as the features of the image are defined beforehand, the difference between the measured RGB and depth values and known RGB and depth values are recorded. The distortion returns two matrices of values, one of which is the calculated intrinsic properties (focal lengths and principal points) of the camera, and the other of which is the distortion values of the camera. It then provides steps to un-distort the image and fix projection errors. A walk-through to utilize this method was developed at the University of Bremen in the

With known camera matrix P,

$$P = \begin{bmatrix} f_x & & c_x \\ & f_y & c_y \\ & & 1 \end{bmatrix}$$

projecting a 3D point $\mathbf{v} = (x, y, z)^\top$ to depth space to get a 2D point $\mathbf{u} = (u, v)^\top$ with depth value d is quite easy, because depth value is exactly z value, i.e. $d = z$. Therefore,

$$\begin{bmatrix} f_x & & c_x \\ & f_y & c_y \\ & & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} ud \\ vd \\ d \end{bmatrix}$$

$$\begin{aligned} x &= \frac{(u - c_x)d}{f_x} \\ y &= \frac{(v - c_y)d}{f_y} \\ z &= d \end{aligned}$$

FIGURE 2.4: Camera matrix for the conversion of pixel number and calibration to real-world x, y, and z values [36].

FIGURE 2.5: Setup used in the University of Bremen in the Institute for Artificial Intelligence [37].

Institute for Artificial Intelligence [37]. This method is highly effective in determining distortion parameters of the Kinect. The initialization of the calibration program can be seen in Figure 2.5.

The OpenCV program was incorporated into the iai_kinect2 package as downloaded, and the calibration run per the described method within the package description. Initialization can be seen in Figure 2.6. There were two nuances to the calibration technique which provided some level of uncertainty in field data collection. The first and most significant

nuance was that the lighting was critically important to successful calibration. If additional light was shined toward the target or toward the camera, the Kinect was unable to undertake the calibration process. Since the environment changes significantly in most agricultural environments, the Kinect's inability to calibrate in changing environments contributes to error. The other nuance was that the target would not be identified if it was not sufficiently flatly displayed. This was to ensure accuracy in the known distance between the corners of the checkers. The successful attempts are contrasted by the unsuccessful ones in Figures 2.7 and 2.8. Conveniently, once the calibration data was found, it could be saved into the camera initialization program so that the calibration was applied thereafter.

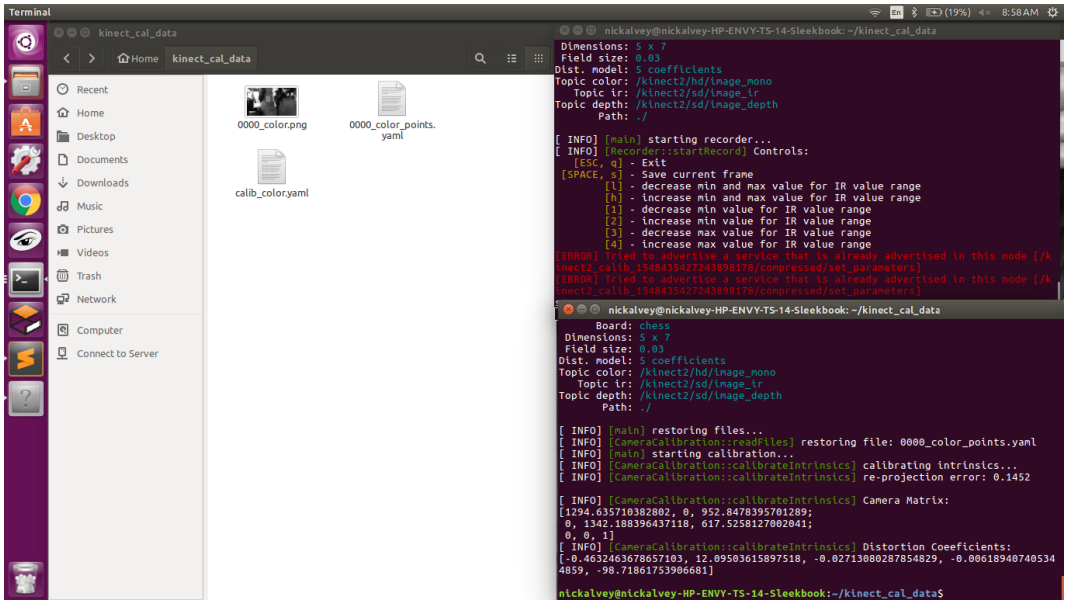


FIGURE 2.6: The initialization of the iai_kinect2 package running the OpenCV calibration.

For calibration, 100 scans were taken of the checkered target. The scans were taken at 15 degree increments ranging between 0 degrees and 30 degrees in either direction from the normal vector of the target. Five scans were taken at each increment, as vertical orientation was varied between scans. This method was reproduced four times at depths of 0.5m, 1m, 1.5m, and 2m. The setup is displayed in Figure 2.9, and raw calibration data can be found in Appendix C. The general conclusions were found:

- The camera had an average depth offset of +7.685cm with a standard deviation of 0.00245, meaning that the the camera perceived the target to be about 7.7cm closer than in reality. This was unconditional in respect to orientation, angle, and distance from the target, as indicated by the low standard deviation.

- Error parameters were calculated for the IR and color offsets (displayed in Appendix C) and inputted into the camera initialization program.

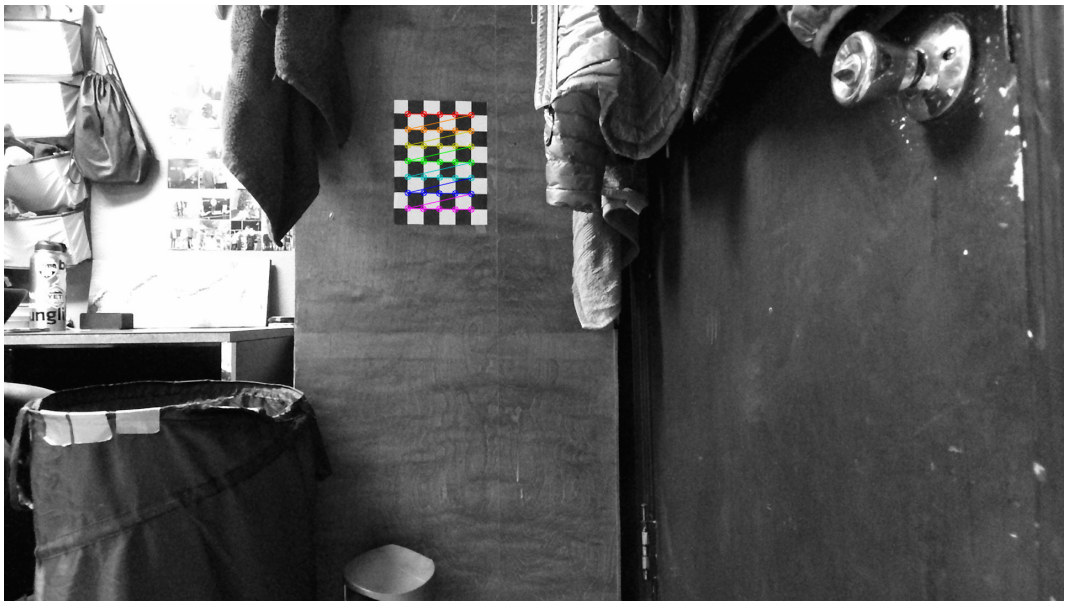


FIGURE 2.7: A successful OpenCV calibration measurement.

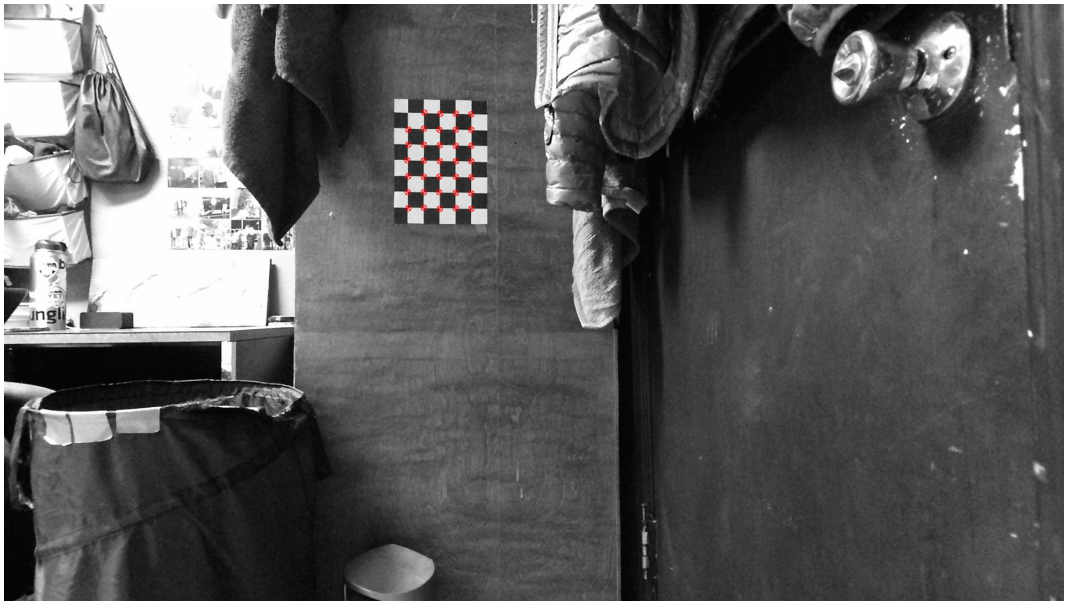


FIGURE 2.8: An unsuccessful calibration attempt.

2.1.4 Development of Code

Once calibration parameters were calculated and properly incorporated into the camera publishing stream, a program was written to subscribe to the camera and manipulate the point cloud. Figures 2.10, 2.11, and 2.12 display the code written in three sections. The first section (Figure 2.10) displays the code used to properly incorporate OpenCV and ROS packages to subscribe to the camera publisher, the second section (Figure 2.11) displays the code used to gather data from the camera publisher and apply the pixel location formulas described in Section 2.1.2, and the third section (Figure 2.12) displays the formatting and writing of

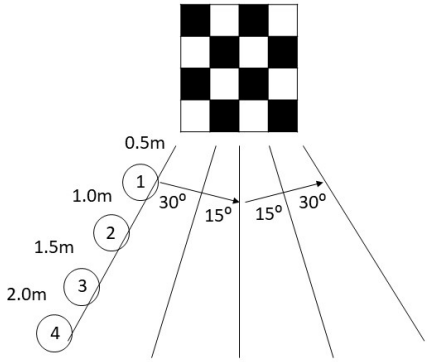


FIGURE 2.9: The calibration setup.

the text file which stores the information gathered from the point cloud manipulation. This was done because Meshlab reads in a text file with X, Y, Z, R, G, B data and projects it in a workable space.

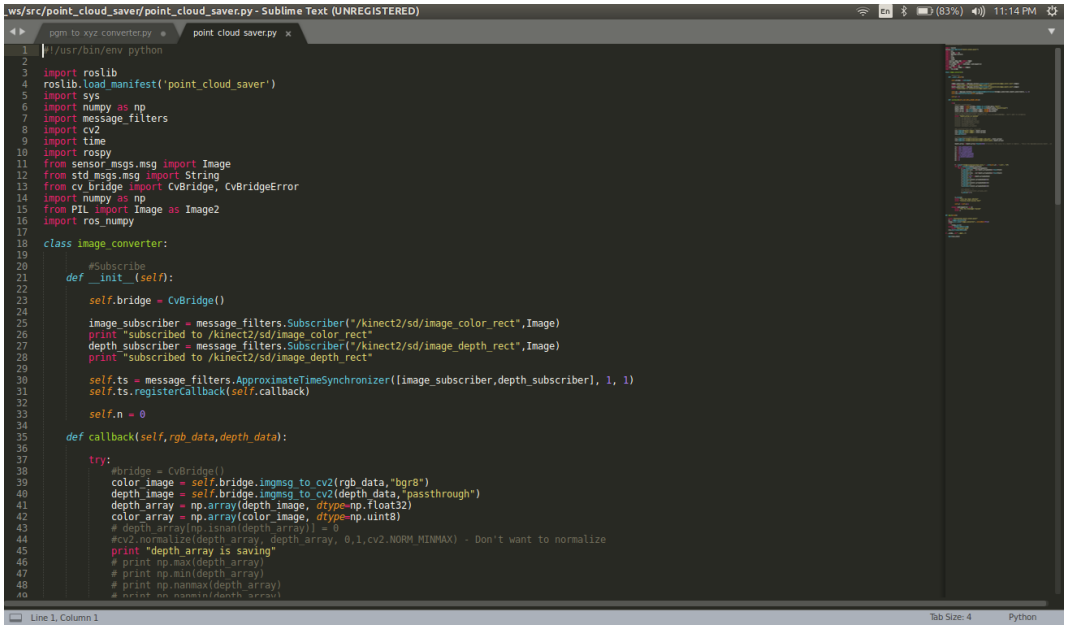


FIGURE 2.10: The initialization section of the developed code.

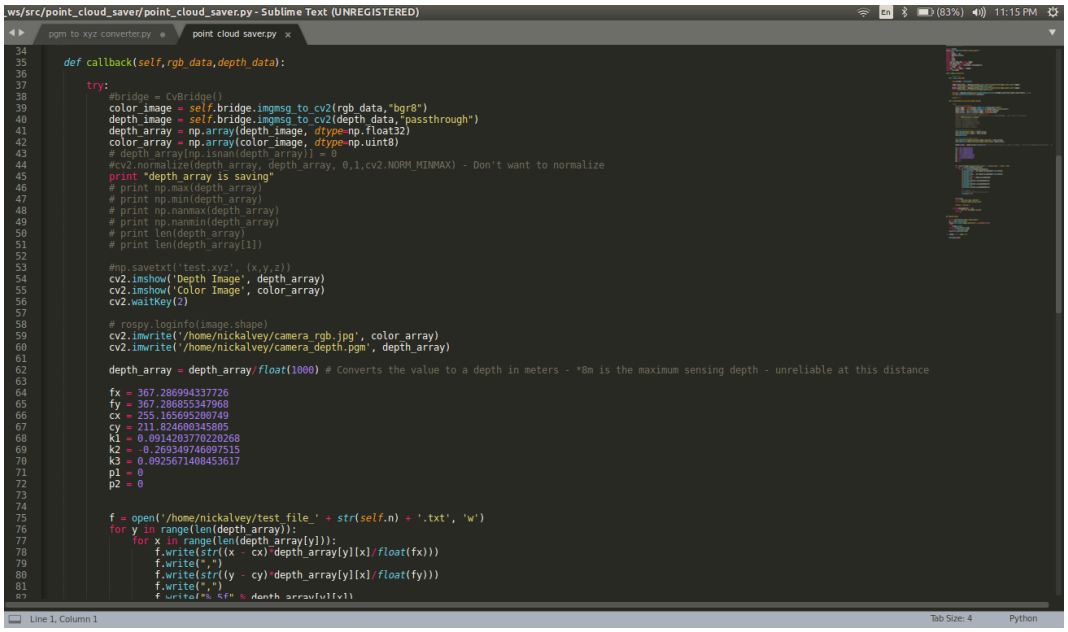


FIGURE 2.11: The camera properties section of the code.

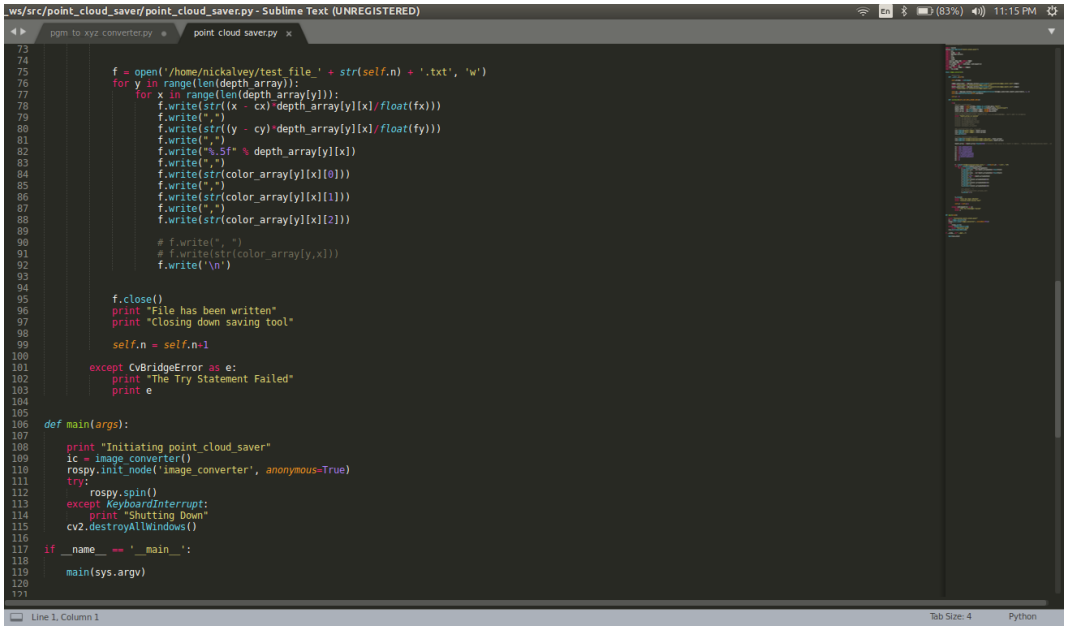


FIGURE 2.12: The data organization and storing section of the code.

2.1.5 Definition of Success

Due to a lack of accessible computing power and expertise, a feature-by-feature comparison to the real crop was not used as the success criteria for this project, unlike many other endeavors [22, 23, 31, 30]. The hypothesis was that even the best scans, or combination of scans, with a Kinect could not provide sufficient resolution to capture the complexity of the many thousands of individual features in any large specialty crop, so any scan could not qualify

as a success in that regard. Instead, it was decided that this process would investigate the Kinect's ability to create the proper shape and texture of a large specialty crop, and to see what level of detail could be provided in a complex environment by a low-cost sensor of this kind. Success is therefore defined as the continuity and resolution of the mesh, as well as the feasibility of the geometry displayed and color returned. Feasibility in this sense signifies that the model had a color and shape that, when subjected to a subjective visual analysis, were deemed accurate to the original object. Measurements were done of the overall size and shape of the crop in Meshlab using its measuring tool to compare measurements in the field. Further improvement to this model creation technique, an analysis of the partially subjective success criteria, as well as shortcomings to this definition of success will be discussed in the concluding section.

2.2 Testing Developed Program and Incorporating Meshlab and Gazebo

Initial setup of the Kinect 2, ROS, and Meshlab yielded fairly mixed results. Tests were done indoors with uniform lighting, entirely static features, and performed in an enclosed room that was entirely within the scanning range of the Kinect. The camera took data from a single location, and only a single scan was used for the initial trial. The "kinect2_viewer" node was launched such that the output could be estimated before a scan was taken. Before the initial scan, an example output is displayed in the image shown in Figure 2.13.

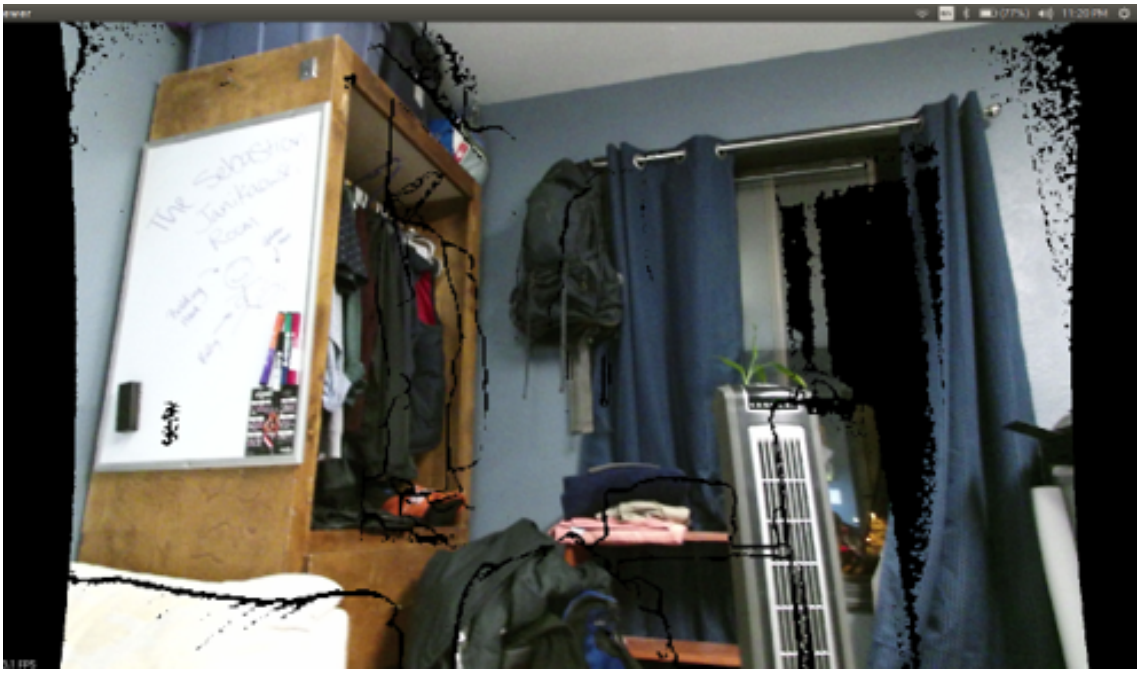


FIGURE 2.13: Camera stream provided by the "kinect2_viewer" node.

It is worth noting that the example view was not used for the first scan, as a wider view was used to capture more data within the testing environment. The scan was saved as a text file and imported into Meshlab. The Meshlab results are shown in Figure 2.14.

FIGURE 2.14: Initial scan result in Meshlab

There were several obvious takeaways from this initial scan, namely the wide variety of open spaces between noticeable features. It provided the first insight that multiple scans would be needed to collect accurate, meaningful data. Another takeaway was the amount of noticeable pixels that registered non-negligible depth and RGB values, but could not be meshed during the surface rendering process. These are primarily seen around the edges of the model and their effect on the meshing process was unknown. For the sake of testing, they were left uncleaned and untouched throughout the process, though they can be cleaned manually if desired. Examples of these areas were highlighted in Figure 2.15.

FIGURE 2.15: A few areas of scan error highlighted.

The next important step was to understand the impact of 1) the open spaces of no pixel return and 2) the pixels that were disconnected from the densely populated areas. This led to the next step of creating faces, setting minimum and maximum vertex length, and eventually meshing the faces together to create a cohesive model. To begin the initial testing, the an established process of rendering a 3D model was followed [28, 27, 29]. This walked through the following eight sequential steps to create a cohesive model:

1. First, the model was imported into Meshlab. While not utilized in the initial test, multiple data entries may be combined and merged into one, aligned, larger point cloud.

2. Normal vectors were computed for the point set. This was done by creating a surface over a set square of pixels and computed the normal vector. There is one changeable parameter available which allows a normal to be calculated by a variable surrounding pixel number. Though the standard given is a 10x10 square, through testing the optimum was found to be a 5x5 square pixel area around each normal. This was found to provide the best balance between smoothing error pixels and providing as many normals as possible.

3. Next, the surface was reconstructed according the the algorithm known as Poisson Surface Reconstruction. This is done by solving the point cloud to a voxel grid (a grid of 3D boxes in which the points of the point cloud are approximated to the box's centroid [39]) with a maximum resolution of $2^d x 2^d x 2^d$ where d is the inputted depth. Inputting a higher depth corresponded to a higher resolution mesh, however each integer increase approximately doubled the computation time. Through testing, an integer increase of 4 from the standard value of 8 was found to provide the best resolution while taking a reasonable time to process (less than 10 minutes).

4. Error was then deleted by eliminating faces larger than a certain size, which depended on the model type and resolution. There was no set optimization of this parameter, as faces were deleted until the model was subjectively evaluated to be optimum. Several overshoots were committed in which too many faces were deleted, resulting in incomplete surface meshes. Improvements could not converge to an ideal minimum face allowance.

5. Next, the texturing and color mapping process was initiated through the Vertex Attribute Transfer feature. The color and texture was saved as a 512x424 pixel matrix, matching the point cloud output of the Kinect. This process was divided into the following two steps.

6. Next, the Parametrization: Trivial Per-Triangle feature was utilized. This generated the faces from which the mesh texture was created. Size was calculated from the Kinect's pixel matrix.

7. The feature known as Vertex Color to Texture was then used. This ensured that the texture file was saved as an applicable .PNG addition to the model in Gazebo.

8. Finally, the model was exported as a .DAE file into Gazebo, as this allowed for the .PNG file of the model color to remain associated with it.

Utilizing this method led to the output result displayed in Figure 2.16.

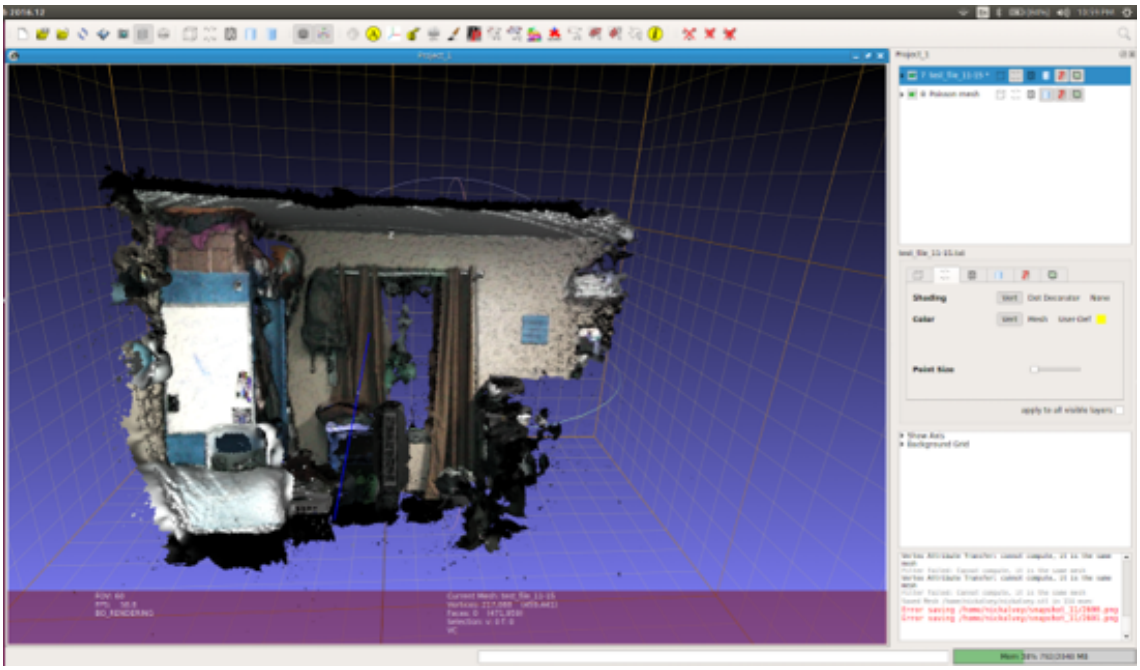


FIGURE 2.16: A model view of the initial scan in Meshlab.

Figure 2.16 clarified the main initial questions about the ability of Meshlab to deal with error. It is clear from the image that the singular pixel values were vastly reduced, with only a few remaining at the bottom of the model. These were easily manually erased afterward. It is also clear that the vast majority of empty spaces were filled with what appeared to be reasonably accurate renderings of the features actually present in the scene. This left two remaining questions: how accurate was the surface approximation applied over known gaps in raw data, and what would the effect of the approximation be when the model was imported into Gazebo for simulation.

The scan in Figure 2.16 was imported into Gazebo and used to create a new world in which it could be verified for both completion and accuracy in its final form. Figure 2.17 shows the result. It was noted that the edges of this scan could have been manually cleaned such that the model appeared cleaner, however it was determined that understanding the changes that occurred to a raw model throughout the process was more valuable to the future improvement.

It quickly became apparent that there was more error present in the Meshlab depiction of the raw model than had originally been assumed. It was noticed that in the transition from Meshlab to Gazebo, error had been introduced to the model, as the surface projections in Meshlab that approximated the surfaces marred by occlusions were removed. This generated questions of how best to overcome this fact. For the purpose of this project, physical data was not attributed to models within Gazebo, except to define the model as a rigid body. The utilization of the process for this single scan led to a few important takeaways for necessary process modifications.

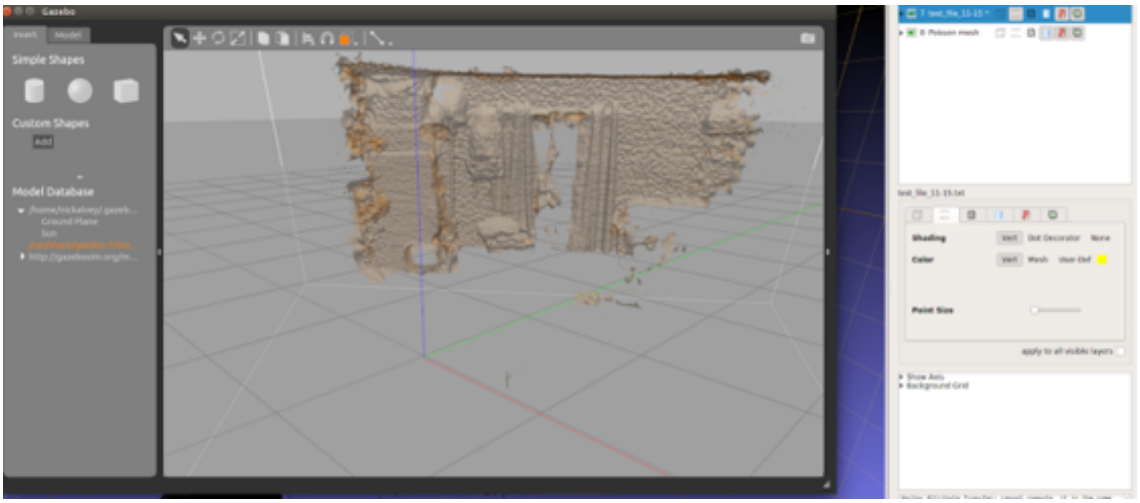


FIGURE 2.17: A view of the model within its Gazebo world.

2.3 Takeaways from Initial Process Validation

The important takeaways are summarized as:

1. There was a significant amount of error present in the scan taken - visually detected as gaps between cohesive surfaces of the model - which led to cascading effects of error through the rest of the process.

2. While Meshlab was able to accommodate some error, a majority of it could not be overcome. This meant that point clouds would need to be significantly more accurate to generate the meaningful results.

3. Due to the Kinect's inability to produce a sufficient number of points for a point cloud in one scan, multiple scans were going to be needed to ensure that accuracy was preserved.

Chapter 3

Data Collection 1

3.1 Process Improvement

Using the takeaways from the initial assessment, the process was modified before data was collected. It was found in initial testing that only one scan was insufficient to generate an accurate 3D model, as the Kinect generated between 50,000 and 200,000 (whereas the average number the software expects is between 1 - 20 million points [26, 28, 27, 29]). This led to the need to take many scans from many angles and aligning them in Meshlab. In order to ensure ease of this added process step, three adjustments were added:

1. It was specified that each scan within a series would be the same distance away from the target. This was specified as a radial distance, as scans were moved along an arch to capture different angles of the target. Scans taken from multiple distances from the target resulted in a need to scale the point clouds differently between scans when aligning. This created many difficulties in assuring accuracy and consistency.

2. It was specified that each scan within a series would be taken from the same height. It was found that scans taken from multiple heights in relation to each other were much more difficult to align properly.

3. It was specified that scans within a series would be taken with the camera in the same vertical orientation (tilt angle of the camera). It was found that meshing scans of different orientations of camera was very difficult.

It was at this point that it was determined that the process was viable to take data.

3.2 Data Collection Protocol

To ensure worthwhile and accurate data collection, a data collection protocol was developed. This protocol was created based on difficulties faced in initial testing of the scanning program, mentioned previously. The form included these specifications:

1. Plant type, age, location, and important/notable information will be recorded before scan

2. Each plant will be scanned a determined number of times across either a 180 degree or 360 degree circle of consistent radius.

(a) The determination between 180 and 360 degree scan will be determined based on planar or non-planar growth pattern of plant

(b) The number of scans will be determined based on size of plant

(c) Generally, there will be between 5-10 scans for each plant

(d) Sets of scans will be kept at an equivalent scan height and each height will be recorded within the file name of the scan file

(e) Multiple levels of scans may be combined depending on size of plant being scanned

(f) Multiple records of each plant type may be recorded to verify accuracy and consistency

3. Scan sets of different distance from the plant may be taken for study of effect on simulation accuracy

4. Scans of multiple heights and orientation (vertical angle) of sensor may be taken for study of effect in simulation

The protocol gave guidelines to address the difficulties mentioned previously, namely that different orientations of the camera, a changing distance from the desired object, and scans from multiple heights greatly increased the difficulty of making a cohesive model. Also created was a form included in Appendix B which gave required fields to be filled in of important information regarding the plant being recorded, the distance the camera was away from the plant (to allow a potential correlative study between distance of scan and accuracy of model), and the height at which it was being scanned.

3.3 Data Collection: North Willamette Research and Extension Center

Data collection was done at the North Willamette Research and Extension Center (NWREC). The NWREC is associated with Oregon State University and focuses on developing crop systems including nurseries and greenhouses, vegetables and specialty seed crops, berries and small fruits, and small commercial farming crops. It is located in Aurora, OR and conducts research to assist farmers in Oregon by running experimental crop trials and experimental growth/maintenance techniques. There are 26 faculty that work there to maintain the facilities and conduct the research [40].

3.3.1 Crops at the NWREC

For the purposes of this experiment, berry crops were focused on at the NWREC. The NWREC produces blueberry, different subtypes of blackberry, black and red raspberry, strawberry, cranberry, and kiwi fruit within the berry crop family. They conduct two main kinds of research: production and physiology research, and breeding and genetics research [41].

There were five kinds of berries scanned for the experiment: marionberry, blackberry, red and black raspberry, and blueberry. The first crop scanned, nicknamed Columbia Star, is classified as a trailing blackberry. This is because it is oriented on a trellis such that the canes of the crop can be trained in a contained manner. It was planted in 2010 and considered to be

a mature crop. The subsequent scan occurred on a marionberry crop that was also planted in 2010.

Next, red raspberry plants were scanned. They are known as Primocane Fruiting Red Raspberry and were planted in 2015, an age that is considered mature. Similarly, the blackberry plants scanned afterward are Primocane Fruiting Black Raspberries, and were planted during the same time. They are also considered mature.

Later, more blackberry crops were scanned. These were planted in 2017 and have its first full crop later in 2019. It had a light "baby" crop in 2018 and will be pruned in early 2019.

Finally, blueberry crops were scanned. They were planted in 2003 and considered mature. They are normally pruned between December and February and are considered ripe in mid to late August. The variety scanned is referred to as "Elliott" [42].

3.3.2 Data Collection

The setup for the data collection included a cart to maneuver and elevate the Kinect, the Kinect and computer to execute the software, a portable battery package for power, and measuring tape to verify distance computed by the kinect and actual distance from the object. Figure 3.1 displays the setup.

FIGURE 3.1: The camera setup for data collection at the NWREC.

Scans were taken of each of the crops described in Chapter 3.3.1

3.3.3 Data Processing

Data was processed following the outline described in Chapter 2. A walk-through of the described process as applied to data collected is shown in Figures 3.2-3.15.

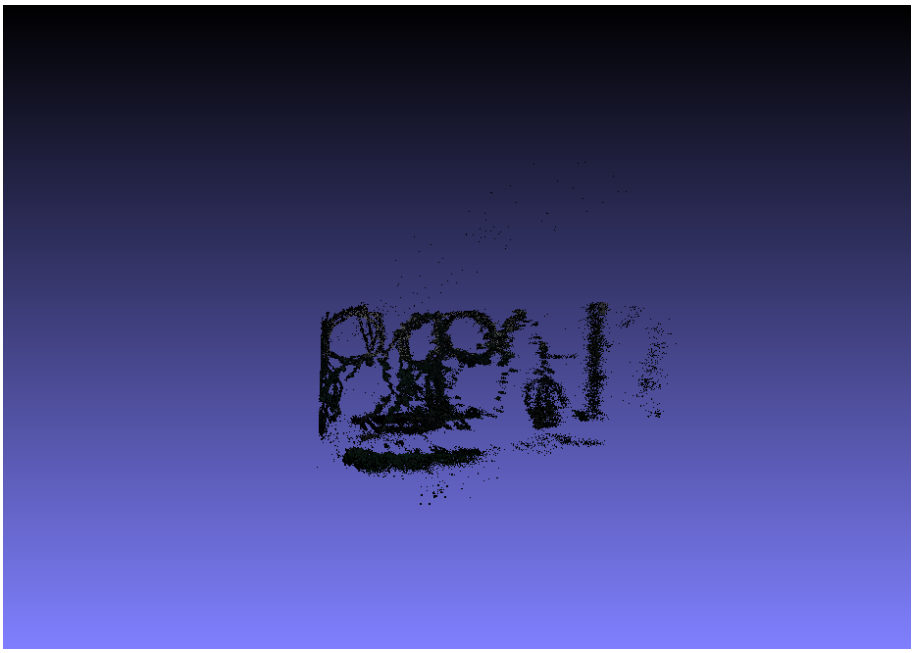


FIGURE 3.2: A raw point cloud of a measured marionberry bush nicknamed "Marion" [42].

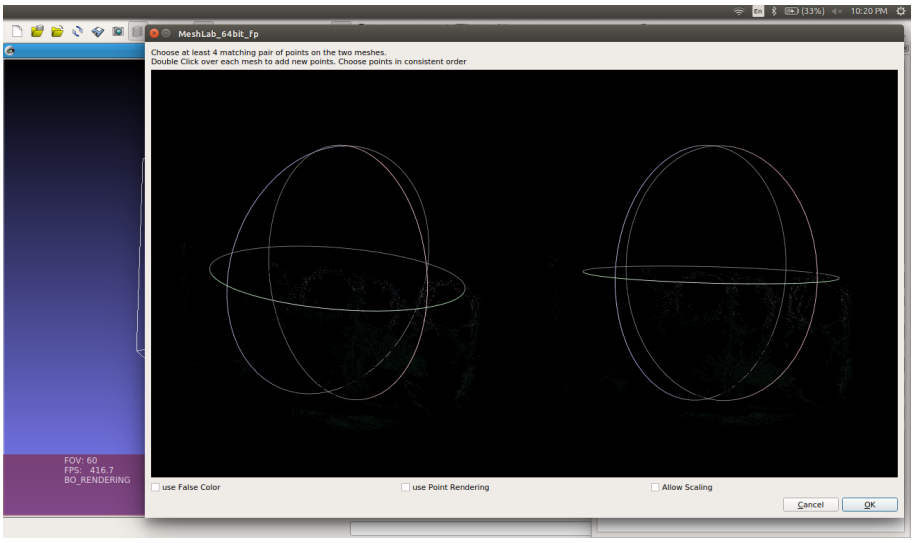


FIGURE 3.3: An example of orienting the point clouds for initial orientation programs to run.

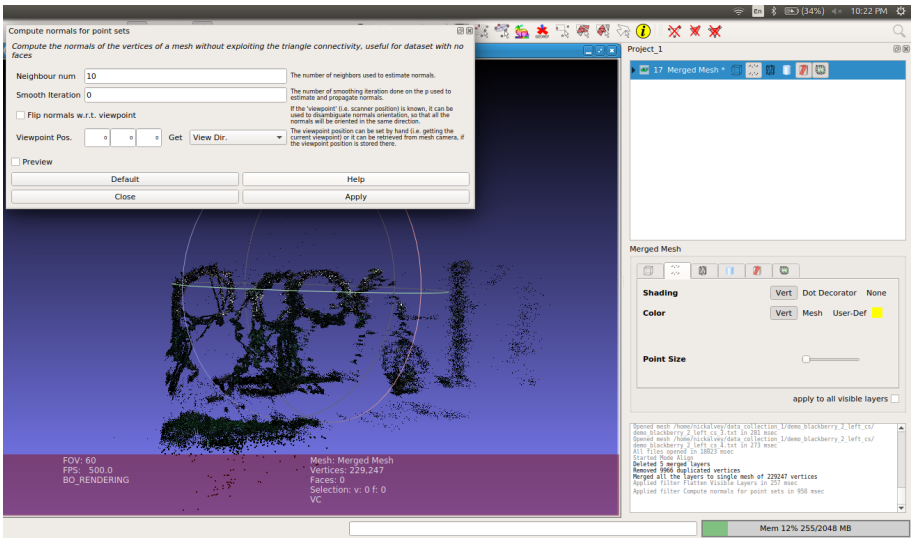


FIGURE 3.4: Computing the normal vectors of the vertices.

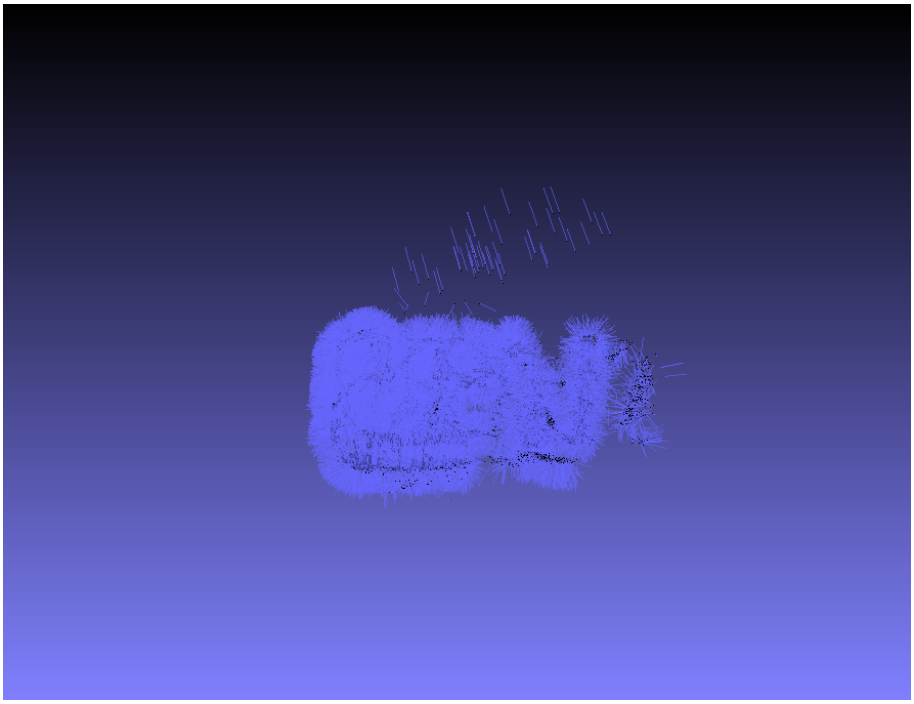


FIGURE 3.5: A rendering displaying the normal vectors computed.

3.4 Results

The results are represented in a series of figures of the point clouds, meshes created in Meshlab, and their turnout in Gazebo. Each is displayed as a subset of three or four images together showing the progression through the process. Alternate views are included if they give insight into surface rendering continuity around the model. One model (the black raspberry crop shown in Figure 3.18) does not have a Gazebo display as there was insufficient processing power to run the simulation environment. The images are displayed in order

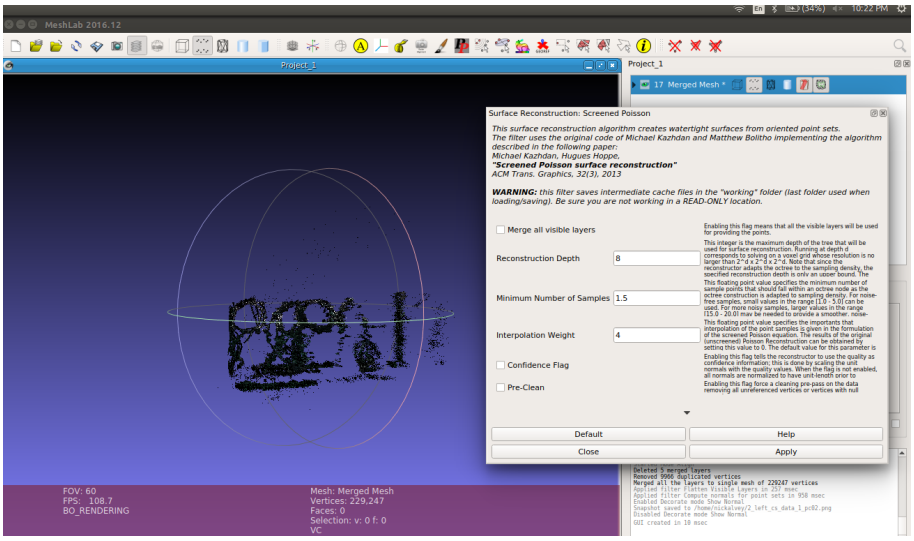


FIGURE 3.6: Executing the meshing program.

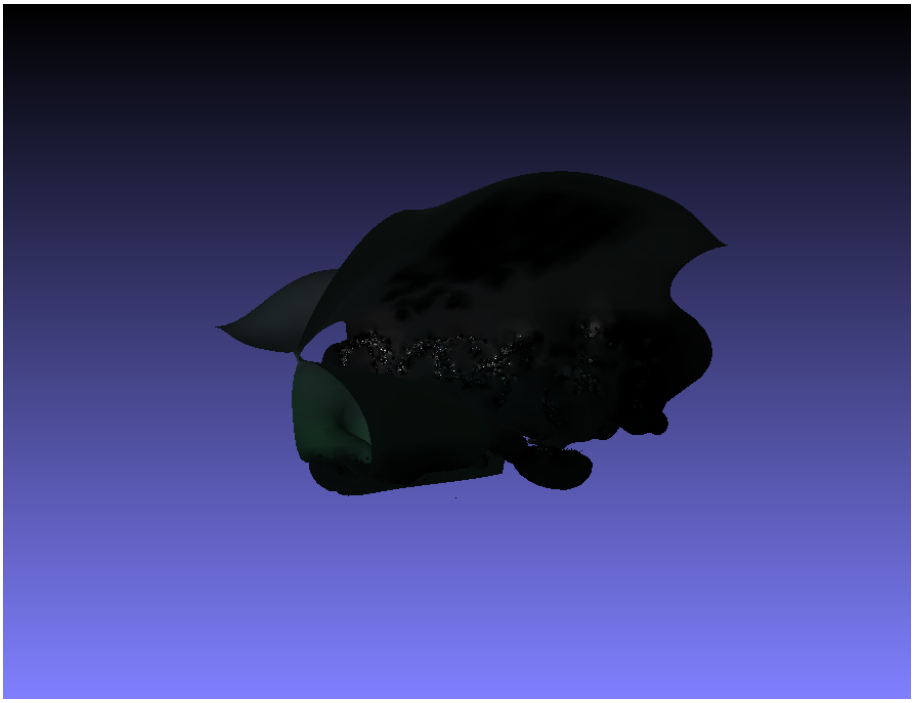


FIGURE 3.7: The outcome of the large faces created by initial meshing.

taken, starting with the Columbia Star scan and ending with the "Elliot" blueberries scan. Results span from Figures 3.16-3.20.

3.5 First Data Collection Initial Takeaways

There are numerous takeaways from the first data collection that were both predicted and unforeseen. Among the predicted takeaways were the difficulties resulting from a low point

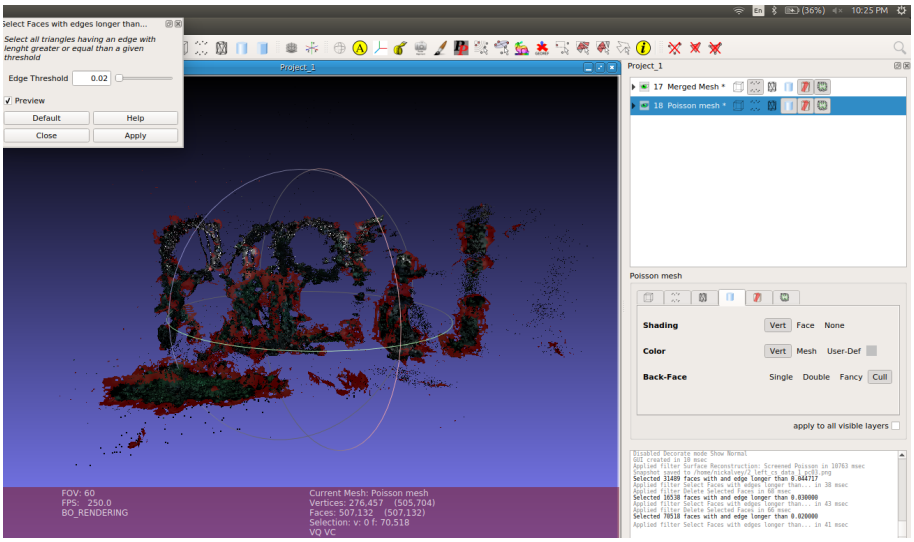


FIGURE 3.8: Eliminating edges to a determined threshold.

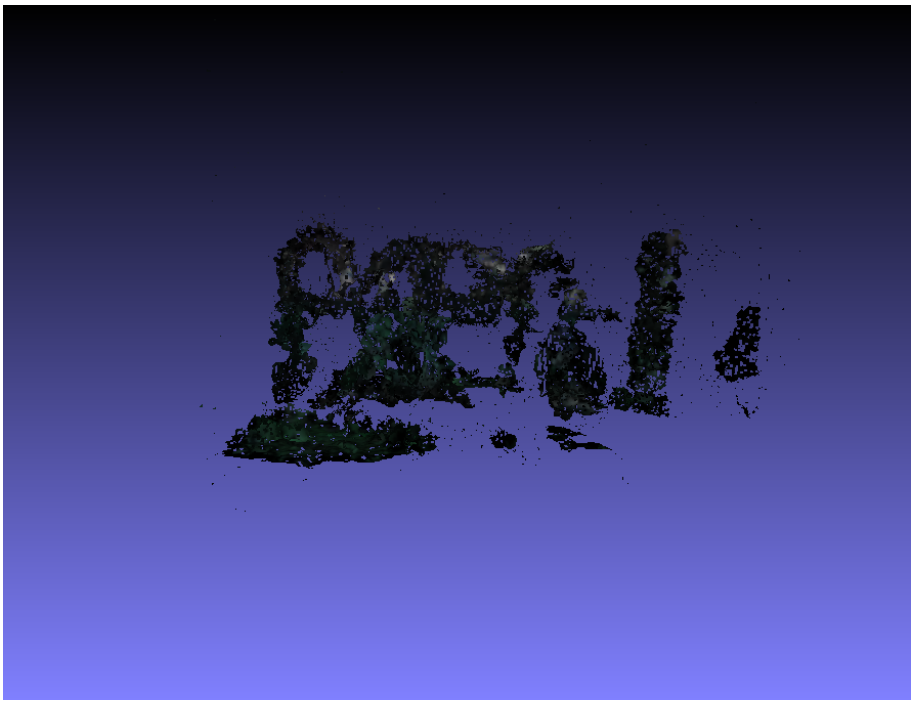


FIGURE 3.9: The outcome of the model in its standard Meshlab view.

cloud density. Some models displayed, like the red raspberry crop in Figure 3.17, returned only 15,000 out of 217,088 possible pixels each scan. Since there were so few points, meshing techniques allowed for massive distortions, and for refinement techniques to cause large, irreparable errors in the models. The effect of this shortcoming is discussed in further detail in the review of the rendered models in Section 5.1.

Another takeaway was the inability to verify the accuracy of the models based on the calibration data of the Kinect. One purpose of this technique was to understand if Kinect

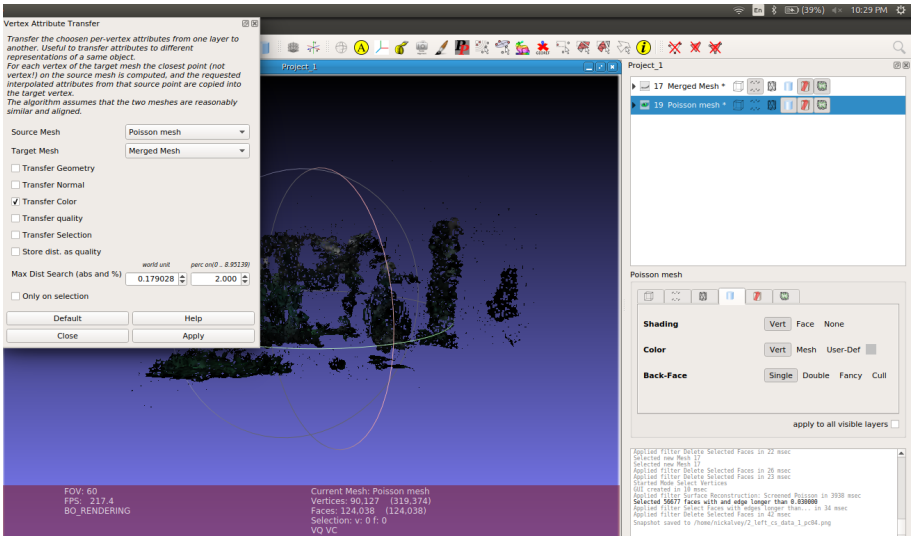


FIGURE 3.10: The computing and saving of texture to allow color to be transferred to the Gazebo simulation.

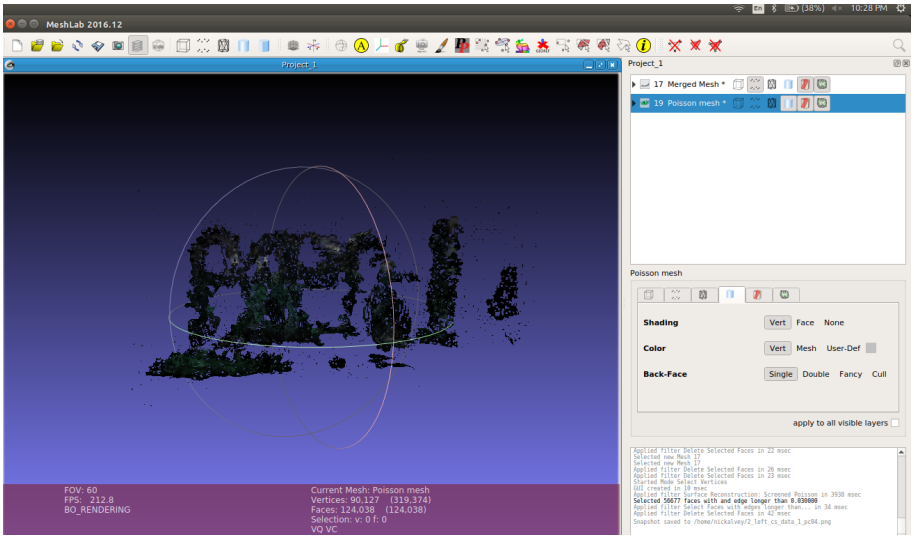


FIGURE 3.11: The finished Meshlab model with its projected surface view.

scans that span the entire area of the crop were accurate (defined by a consistent and visually reasonable surface rendering) and produced sufficiently dense point clouds to result in meaningful data. In addition to it being clear that this technique did not produce a sufficient dense point cloud, the surface meshing error in the remaining parts of the crop make the accuracy verification near impossible.

What was unforeseen was the extent of the auto-culling feature found in transition between Meshlab and Gazebo. Meshlab projects a computed surface based on assumptions of the orientations of the points in the point clouds, which accounts for errors due to occlusions of features in the crop. This projection, however, is not applied in the transfer from Meshlab to Gazebo, and models are culled when they are converted between software. Because of this, occlusions became a very significant factor in the accuracy of the model. This can be

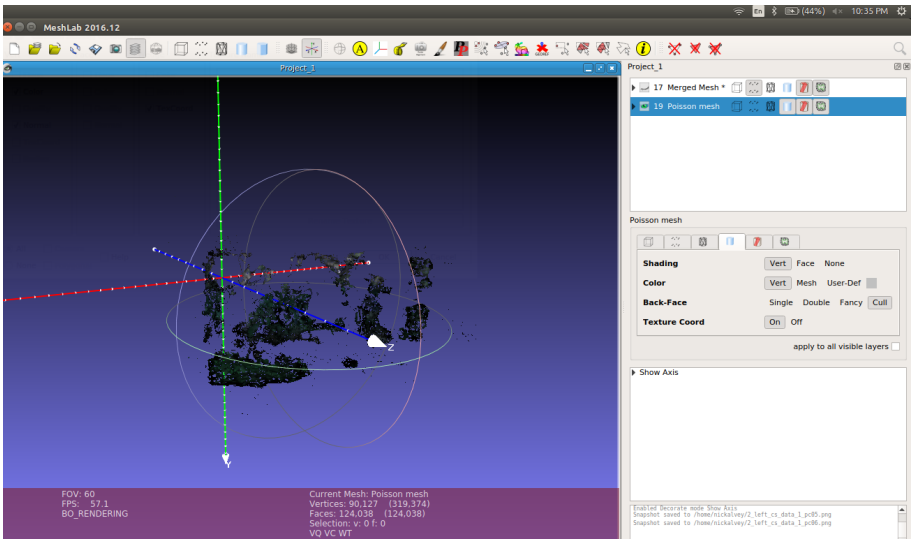


FIGURE 3.12: The culled view of of the resulting model.

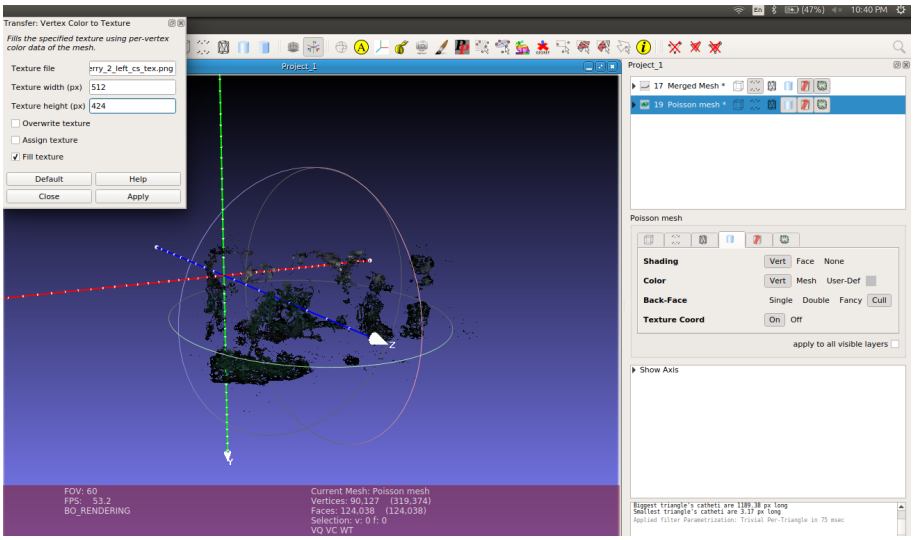


FIGURE 3.13: The texture applied to the culled view.

seen best in the difference between Figures 3.11 and 3.12, as well as in the difference between Figure 3.16 image (B) and image (D). Overall, the models were very inaccurate, and would not be helpful in simulation testing.

The main takeaway for future data collection was that a significantly higher number of point cloud points would be needed to properly account for occlusions and meshing inaccuracies. Additionally, the emphasis on scanning from multiple angles was found to be less helpful than initially thought. This was because the number of points was not increased significantly enough between each scan (due to each scan providing so few returns) to make the combination of multiple scans worthwhile. For the next data collection, a different approach was be taken. The Kinect was be placed much closer to the desired section to be scanned and instead of attempting to capture the entire crop in one scan, sections were combined to create a much more densely populated point cloud.

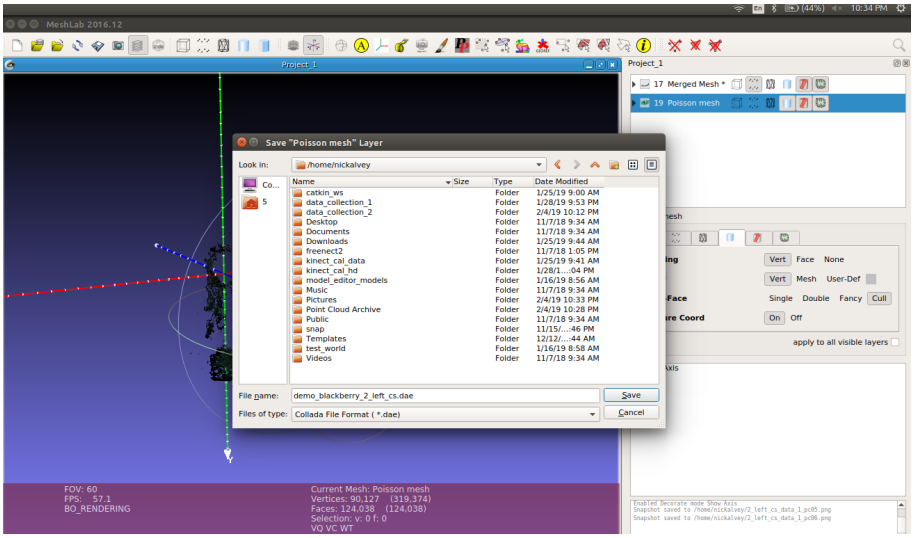


FIGURE 3.14: The model being saved to be placed in Gazebo simulation.

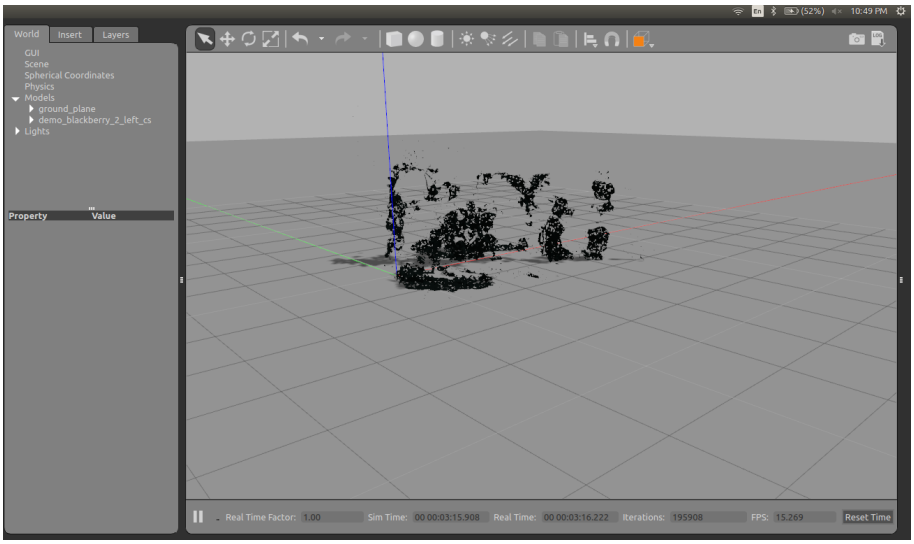
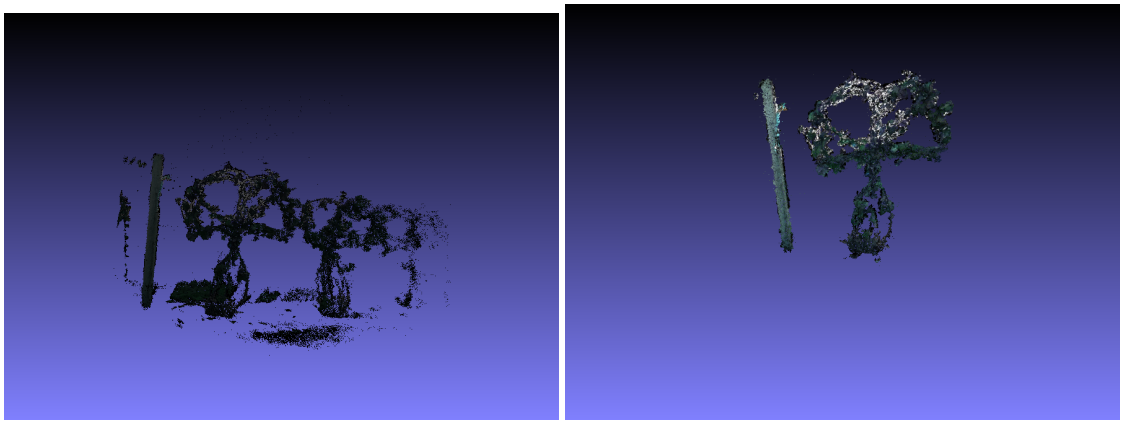


FIGURE 3.15: The model within Gazebo.



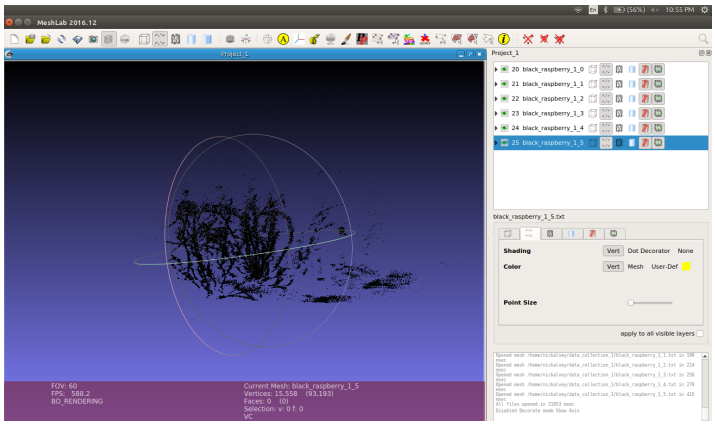
(A) The initial point cloud of the Columbia Star blackberry crop.

(B) The meshed model of the Columbia Star blackberry.

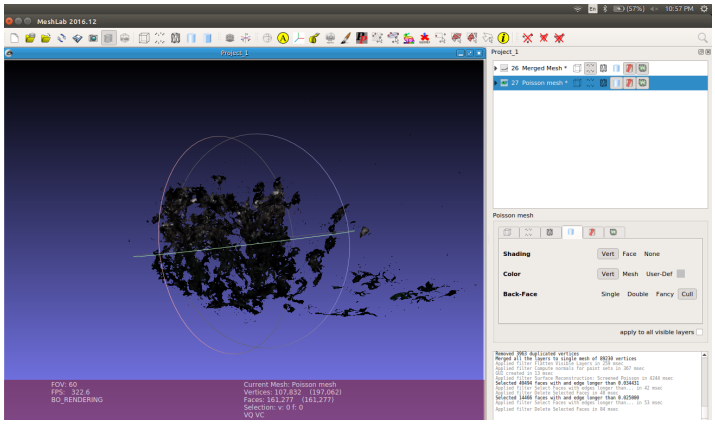
(C) The Columbia Star placed in Gazebo without color or texture association.

(D) The Columbia Star in Gazebo with color and texture.

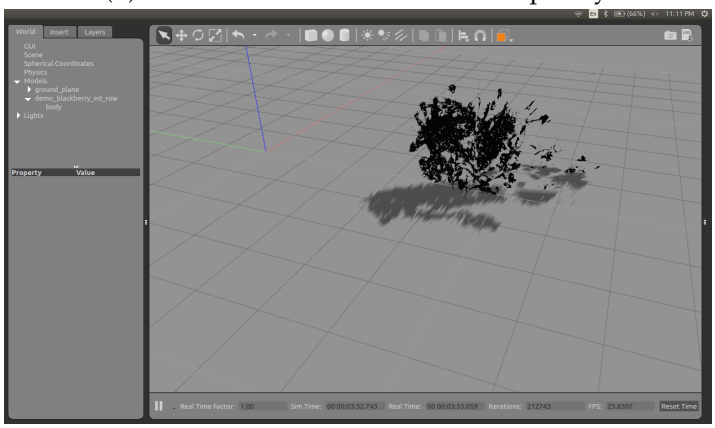
FIGURE 3.16: The Columbia Star turnout.



(A) The initial point cloud of the red raspberry crop.

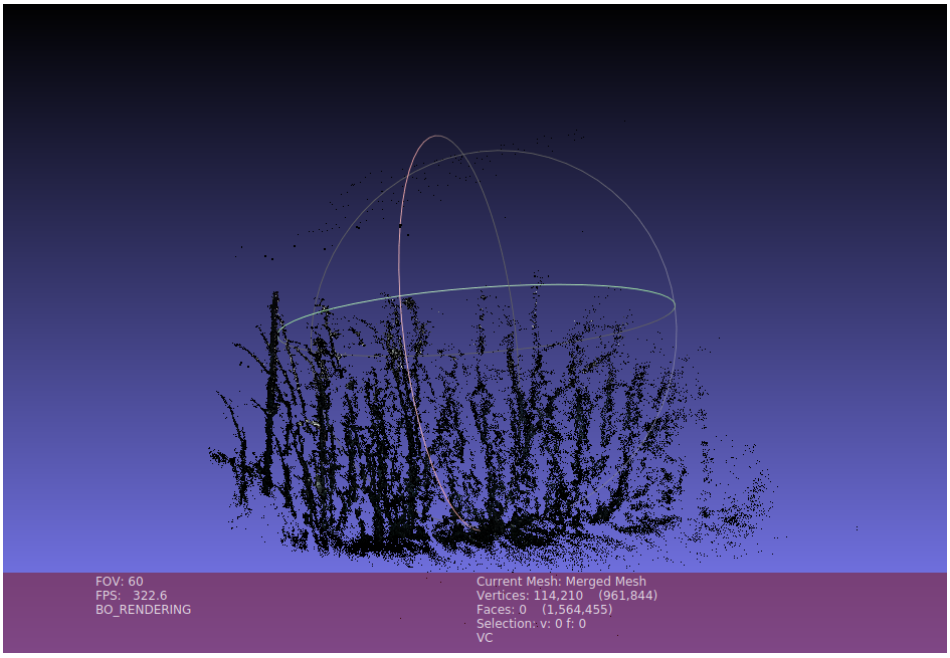


(B) The meshed model of the red raspberry.

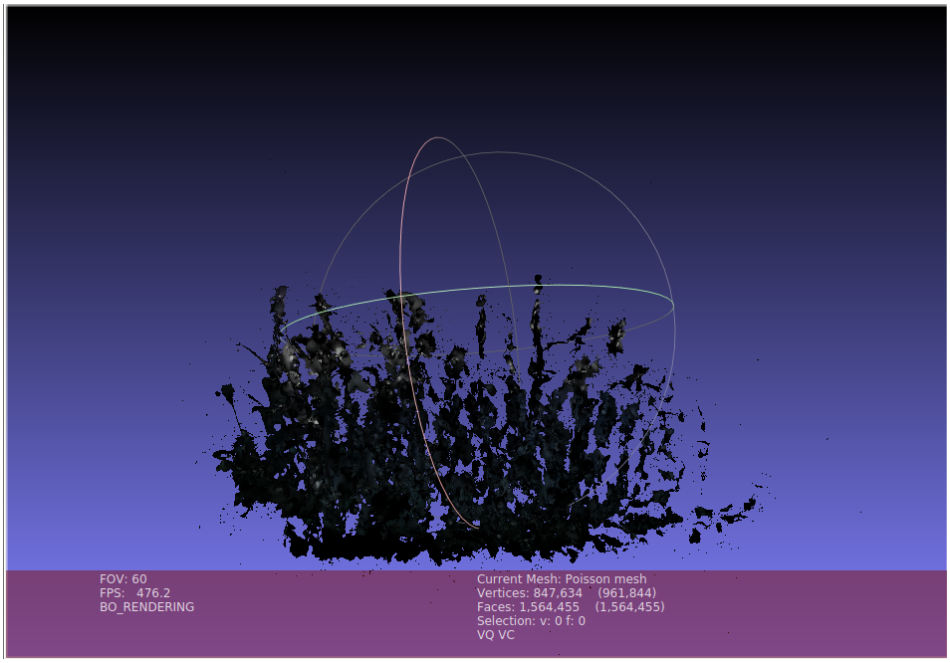


(C) The red raspberry in Gazebo

FIGURE 3.17: The red raspberry turnout.

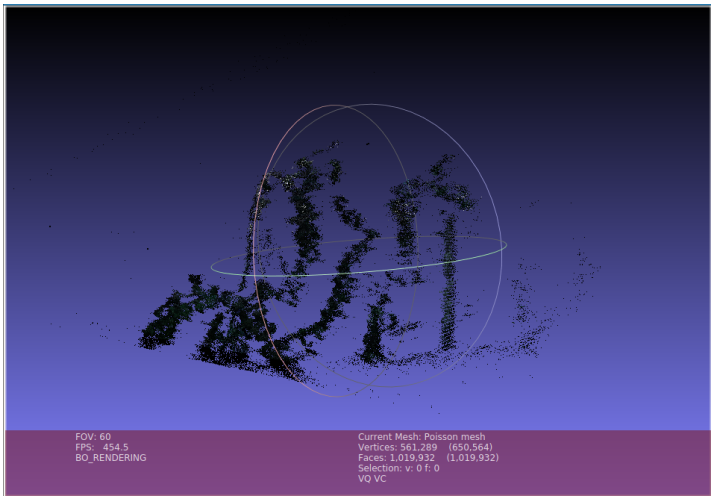


(A) The initial point cloud of the black raspberry crop.

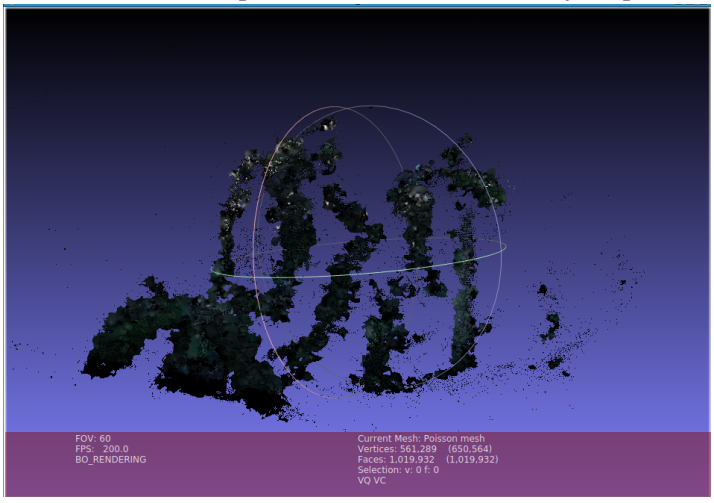


(B) The meshed model of the black raspberry.

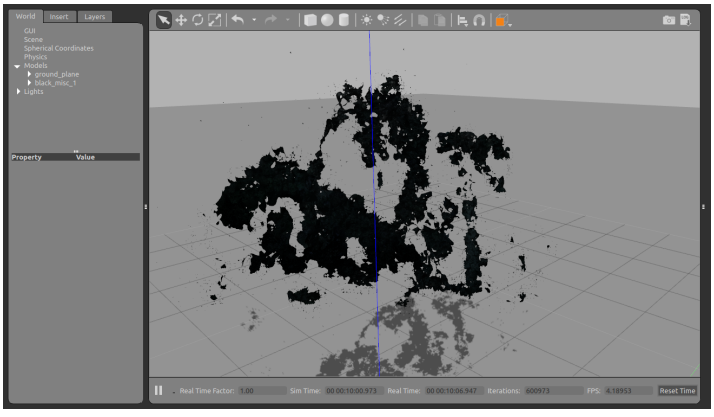
FIGURE 3.18: The black raspberry turnout.



(A) The initial point cloud of the blackberry crop.

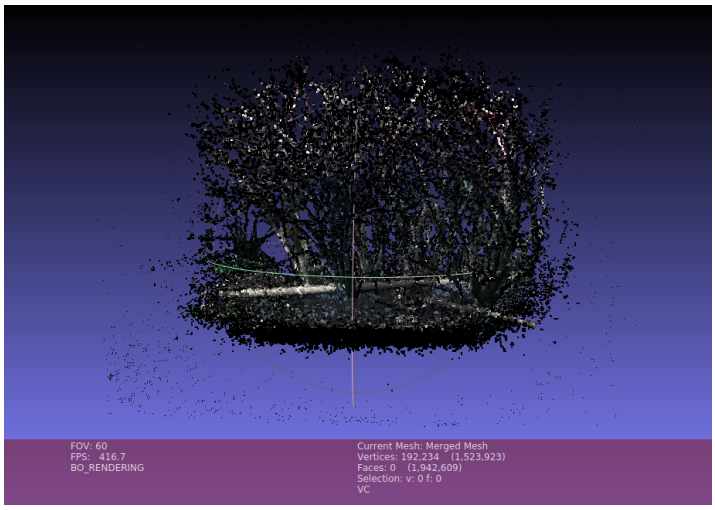


(B) The meshed model of the blackberry.

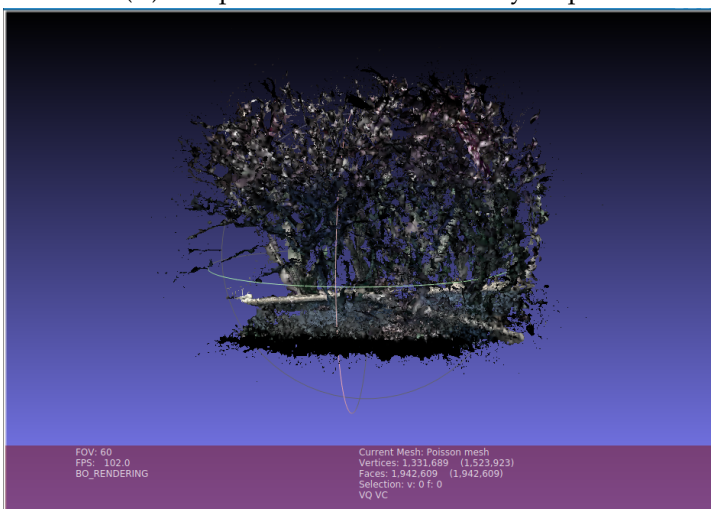


(C) Gazebo turnout of the blackberry crop.

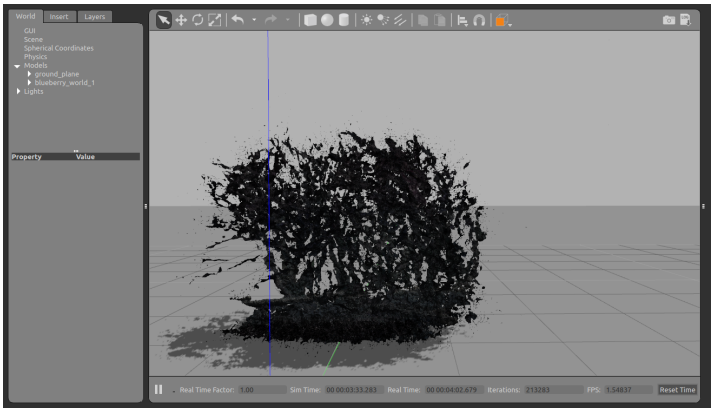
FIGURE 3.19: The other blackberry crop turnout.



(A) The point cloud of a blueberry crop.



(B) Meshed model of a blueberry crop.



(C) Gazebo turnout of the blueberry crop.

FIGURE 3.20: The "Elliot" blueberry turnout in the first data collection.

Chapter 4

Data Collection 2

4.1 Data Collection: NWREC New Method

With the previous data processed and takeaways recorded, it was decided that re-scanning the same crops with a new method would create more accurate models and allow a comparison between the effectiveness of the techniques. Half of the scans done in the second round were the same as in the first, including the scanning of the Columbia Star crop and "Elliot" blueberry crop. Three new crops were scanned, including a blackberry crop called "Triple Crown", a red raspberry crop, and a black raspberry crop. The red and black raspberry crops were part of the same Primocane Fruiting Black and Red Raspberry crops planted in 2015.

4.1.1 New Technique Description

Data was collected with the same equipment as the previous data collection. The difference between the techniques was a focus on gathering as many points as possible with each scan by scanning the crop from between 2 and 3 feet away, instead of between 6 and 8. There are multiple reasons for the increased point return of this method. The first is that since light is shown from the camera in a conical shape, the farther the crop is from the camera, the more pixels will not return as they will be proceed around/above the crop unhindered. The second is that since each pixel represents a certain amount of data from the crop, the farther away the camera is from the object, the more data each pixel represents. This is because the number of pixels remain constant, but the size of the image captured increases. As the camera moves farther away, there is an increasing chance that pixels cannot be properly returned, as they represent an increasingly complex surface geometry. This contributes to the presence of occlusions and causes the culling process to have an increasingly drastic effect. A pixel representation of a farther and closer scan is displayed in Figure 4.1.

Crops were divided into between 2 and 4 subsections to be scanned, depending on the crop's size and shape. For the blackberries with large, looping sections, 3 sectional scans allowed a focus on each of the 2 loops and the center stock. For the black and red raspberries, 4 sections were created to capture the tallest and lowest stems for the left and right sides of one crop. For the blueberries, only 2 scans were used to capture the upper and lower features of the crop.

4.2 Results

The results of this data collection can be found in Figure 4.2 through Figure 4.9. Each series of images contains a point cloud depiction, a meshed depiction, a mesh with measurements,

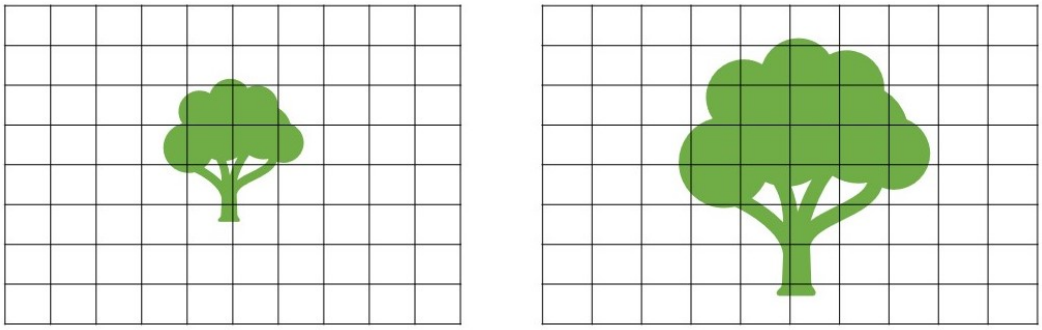


FIGURE 4.1: A depiction of a far vs. close pixel representation of an image.

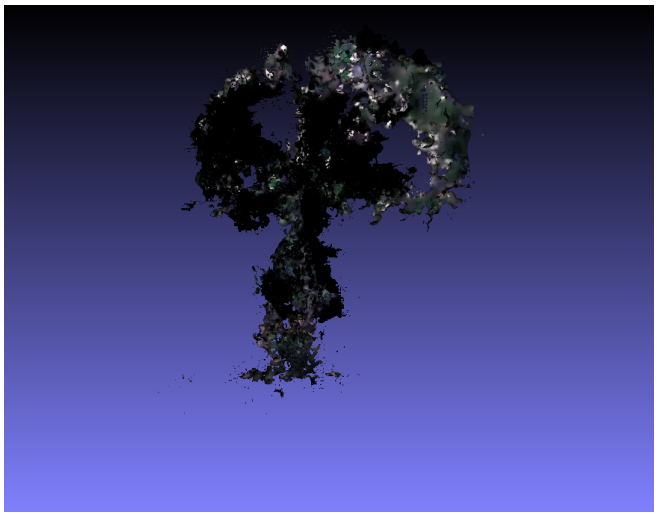
and, for some models, the final output in Gazebo. There were some models in this data collection that could not be manipulated in Gazebo due to available processing ability. These included the "Elliot" blueberry crop, the black raspberry crop, the red raspberry crop, and the Columbia Star crop. In some of the results, additional views were added to aid in understanding of failures or successes in the rendering process.

4.3 Second Data Collection Initial Takeaways

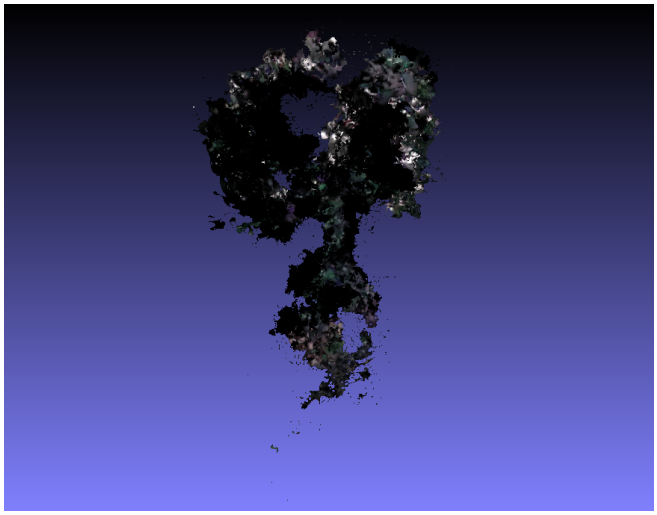
There were a few takeaways from the second round of data collection, namely that the new process was phenomenally more successful in the specified criteria. The point cloud density for each model was between 3.1 - 14.7x higher, resulting in surfaces with up to 50x more faces (translating to up to 50x higher resolution). Surfaces were rendered significantly more accurately and completely, especially for the "Elliot" blueberry crops (Figure 4.5) and the "Triple Crown" crop (Figure 4.7).

Another takeaway was that there was a strong effect of glare in the color detection of the Kinect. This can be seen especially in the data collected of the red raspberry crop (Figure 4.3), in the Columbia Star blackberry crop (Figure 4.2), and on the left side of the "Triple Crown" crop (Figure 4.7). Color returns are either partially or entirely black in these areas. A deeper discussion of these two data collection takeaways will occur in the final review of the results (Section 5.1).

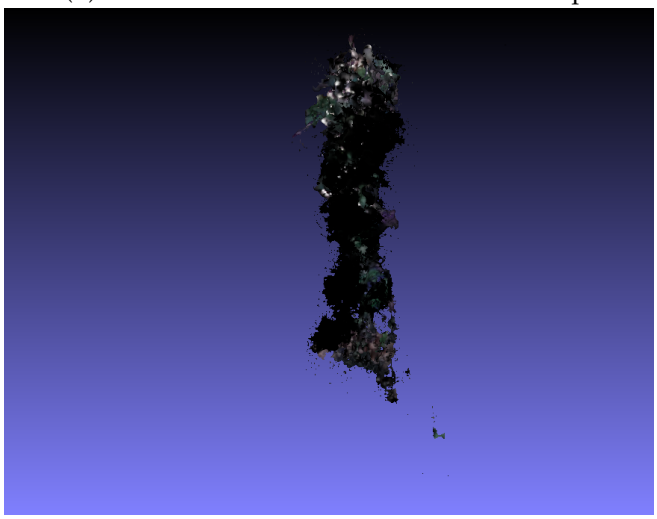
These results validated the second data collection method over the first. Sectional scans returned the either the full, or nearly full pixel range (217,088 pixels), and proved to be significantly more usable and efficient than the previous method. Occlusions became significantly less harmful to the model integrity as culling did not distort the model as it had done previously. It is recommended that any future use of this scanning technique with any low-cost RGB-D sensor use the second method of data collection.



(A) The meshed model of the Columbia Star crop.

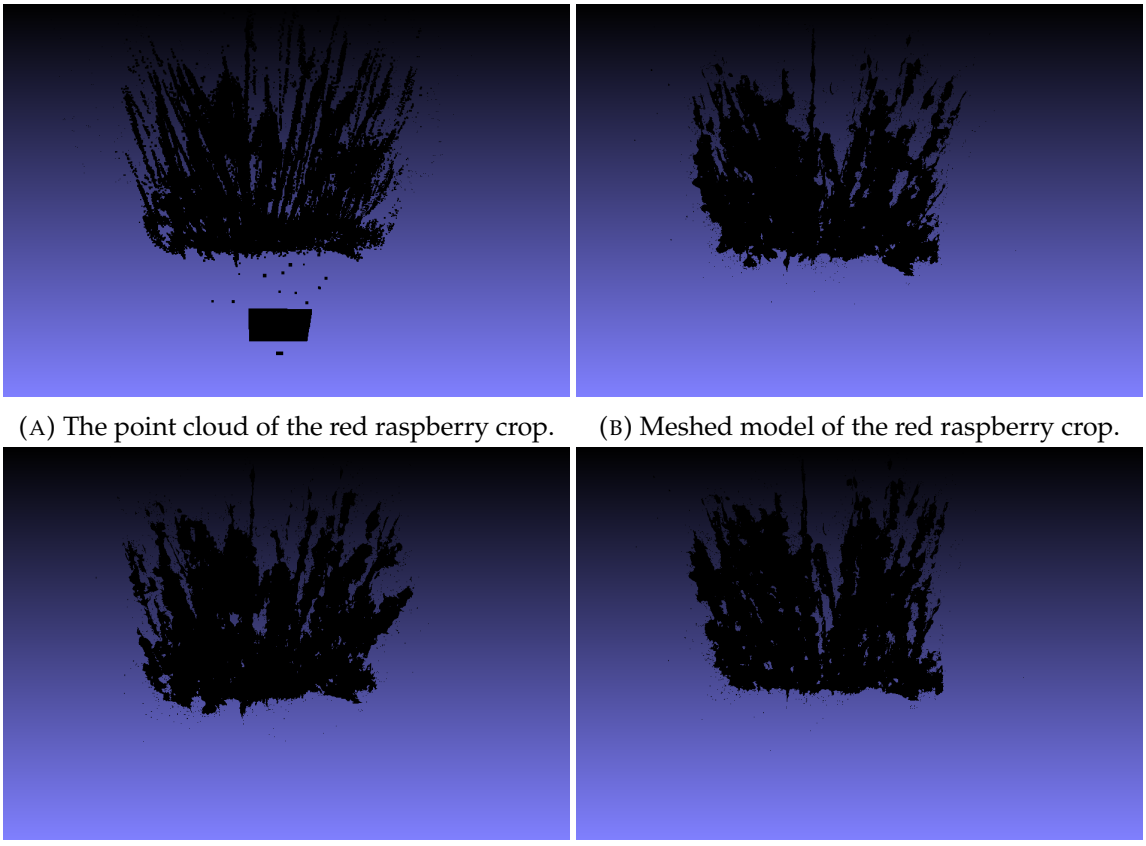


(B) Alternate view of the Columbia Star crop.



(C) Alternate view of the Columbia Star crop.

FIGURE 4.2: Columbia Star turnout in the second data collection.



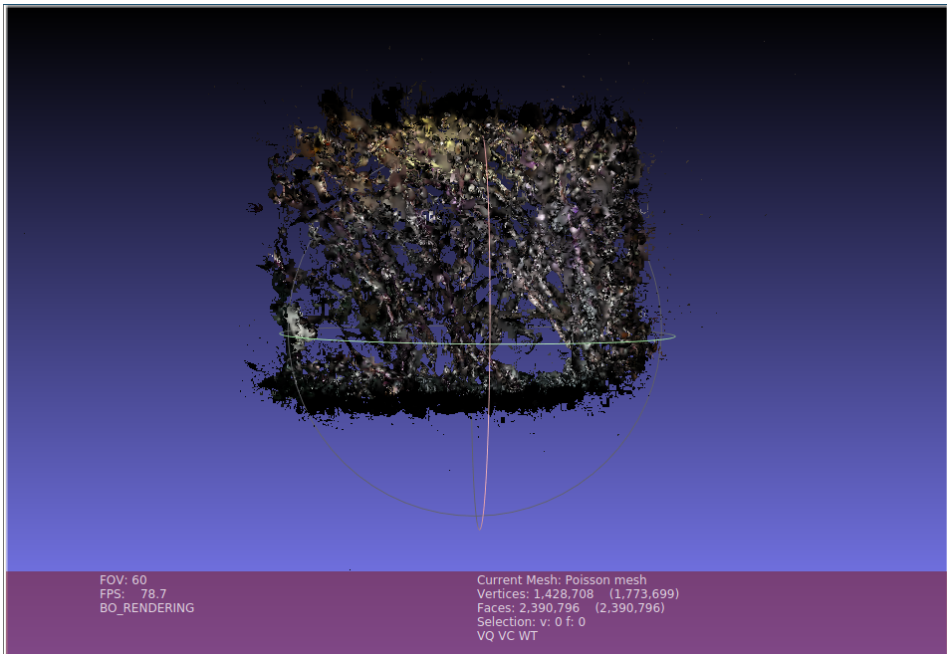
(A) The point cloud of the red raspberry crop.

(B) Meshed model of the red raspberry crop.

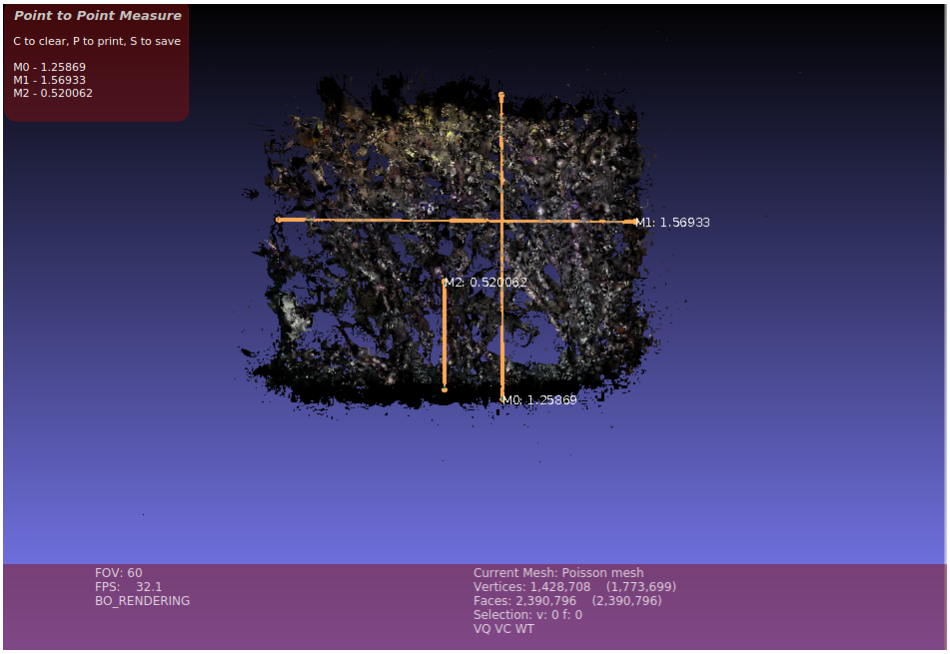
(C) Alternate view of the red raspberry meshed model.

(D) Alternate view of the red raspberry meshed model.

FIGURE 4.3: The red raspberry turnout in the second data collection.

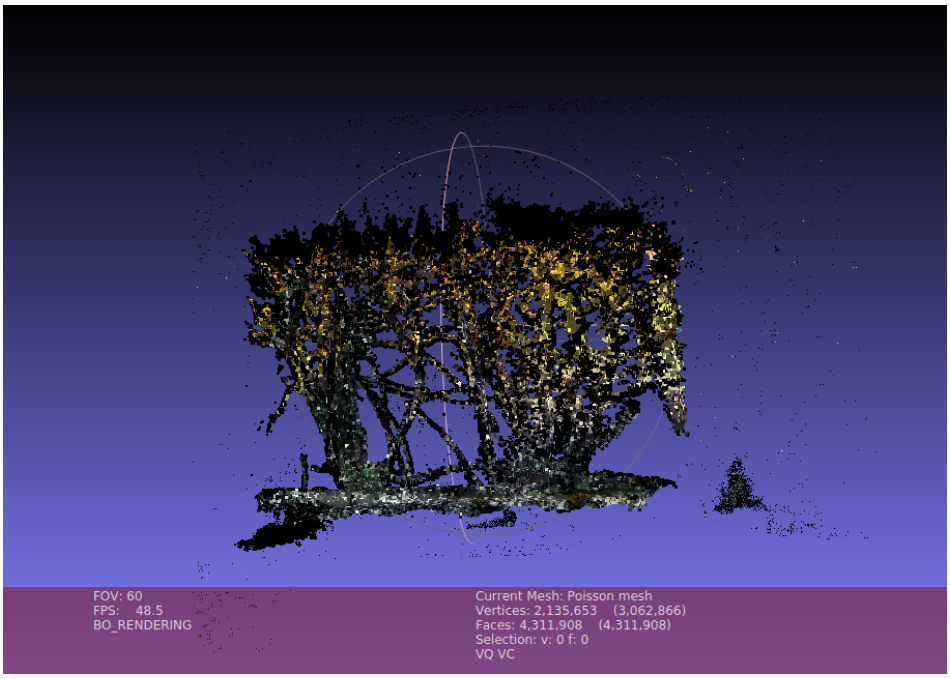


(A) The point cloud of the black raspberry crop.

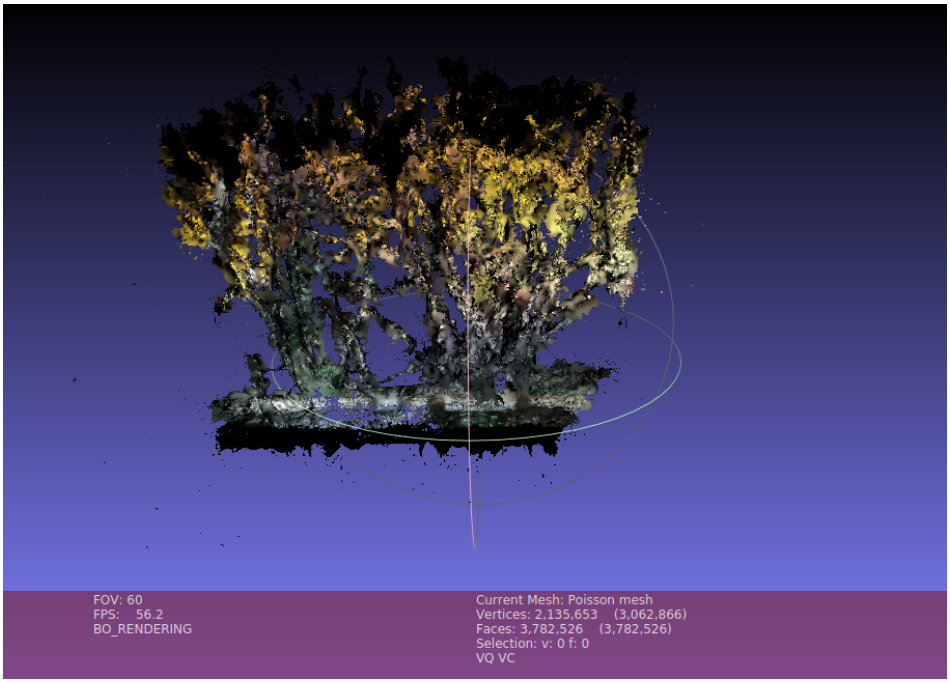


(B) Meshed model of the black raspberry crop.

FIGURE 4.4: The black raspberry turnout in the second data collection.



(A) The point cloud of the "Elliot" blueberry crop.



(B) Meshed model of the "Elliot" blueberry crop.

FIGURE 4.5: "Elliot" blueberry turnout in the second data collection.

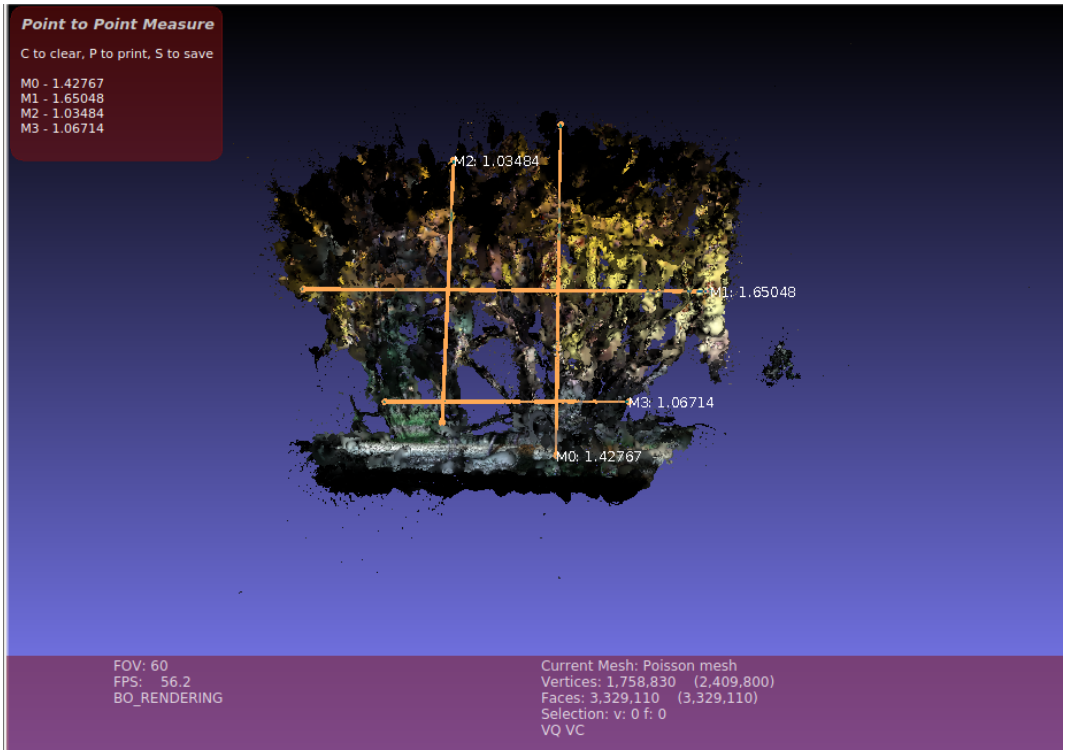
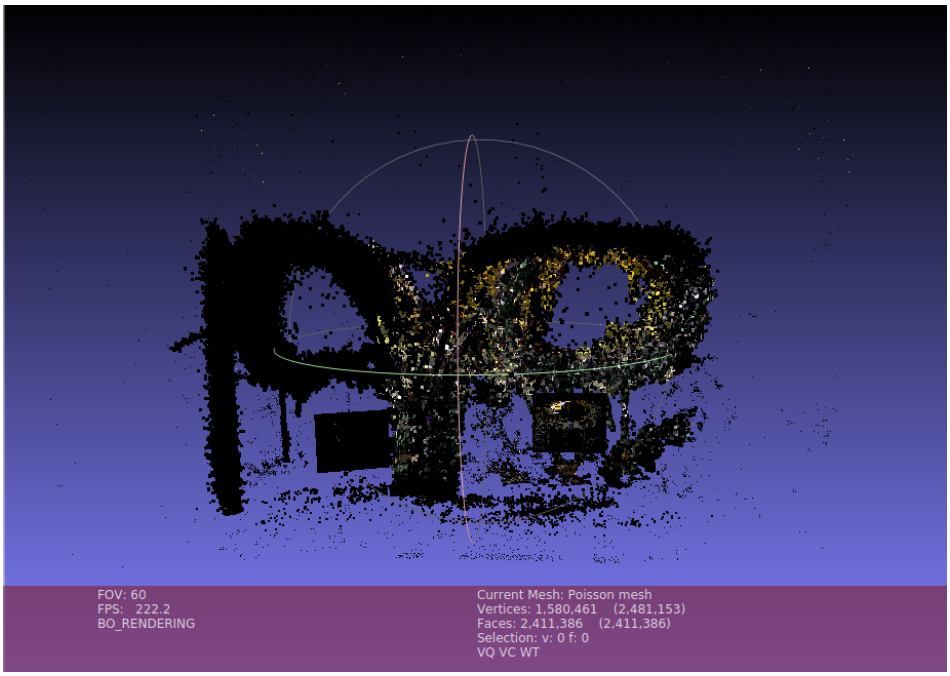
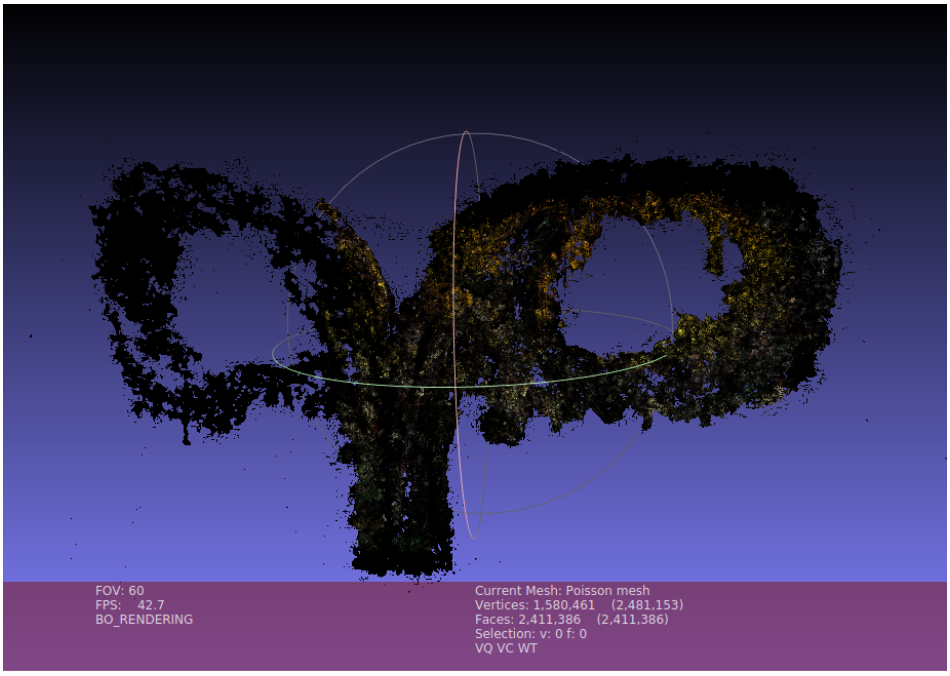


FIGURE 4.6: The measurement of the "Elliot" blueberry crop.



(A) The point cloud of the "Triple Crown" crop.



(B) The meshed model of the "Triple Crown" crop.

FIGURE 4.7: The "Triple Crown" turnout in the second data collection.

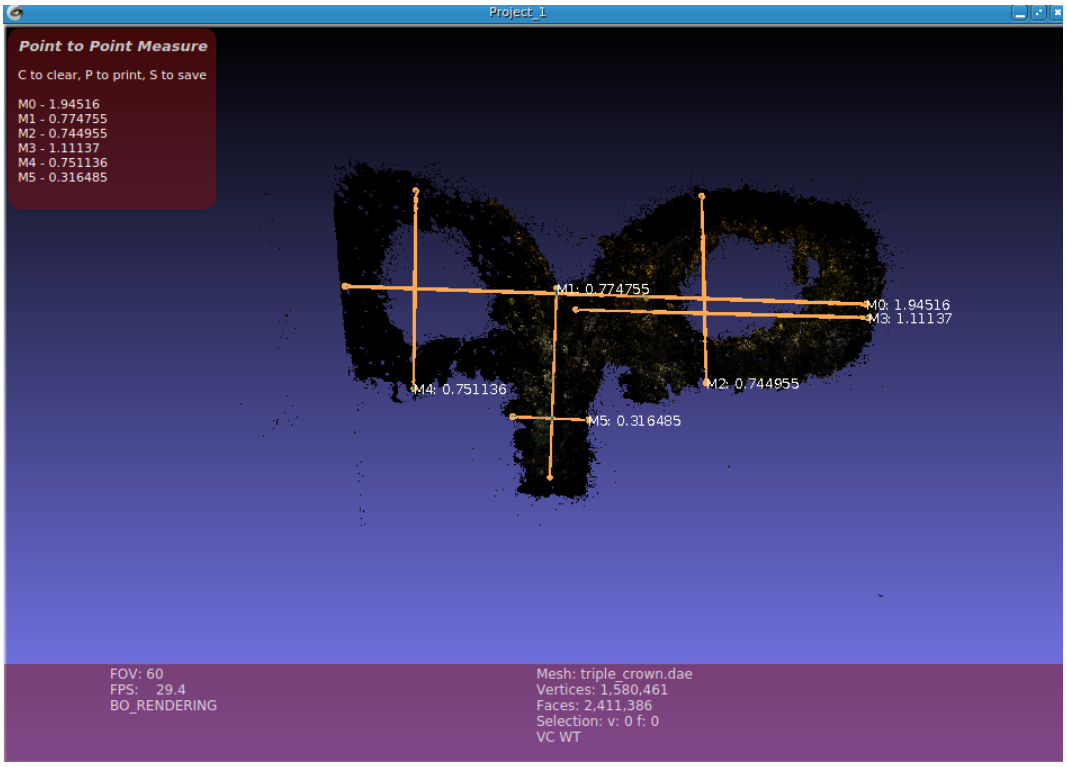


FIGURE 4.8: Measurement results of the "Triple Crown" model creation (m).

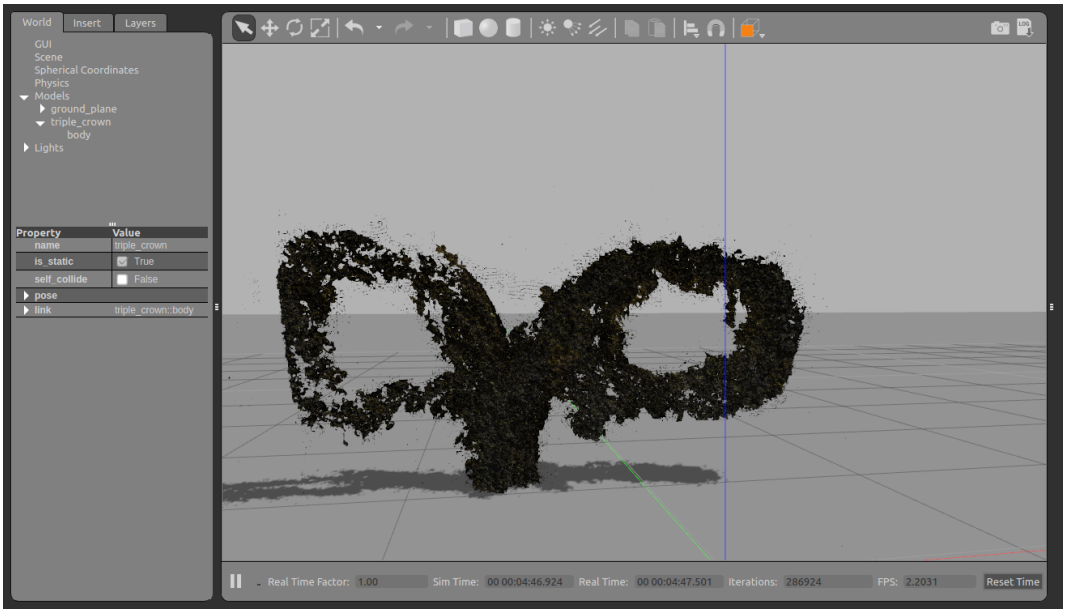


FIGURE 4.9: Gazebo results of the "Triple Crown" model.

Chapter 5

Discussion

5.1 Review of Results and Usefulness of Models

As discussed in Section 3.5, results from the first data collection yielded very poor renderings of the target crop. Every success criteria developed was clearly failed. Even the best models, like the rendering of the "Elliot" blueberry crop (Figure 3.20), were very inaccurate, even by the subjective standards. The point cloud density per model (ranging from 55,000 to 157,000 total after between 3 and 7 scans) was very low, considering that per scan the Kinect can return up to 217,088 pixels. It is easily seen in Figures 3.15, 3.16 (D), and 3.17 (C) that final models generally have significant gaps in throughout the mesh. The model of the "Elliot" blueberry crop is the best model rendered in this data collection in quantity of points gathered, however it is clear upon closer examination (by zooming close to the mesh in Meshlab) that extreme error is still found. It was noticed that the crops with more prominent features, like the thicker branches of the blueberry crops or the larger leaves of the blackberry crops, returned more points overall than the raspberry crops which had much thinner branches.

As mentioned in Section 4.3, the second data collection yielded much, much higher resolution results. This success was quantified by the increase in the number of points in each point cloud (between 3.1 and 14.7x) and resulting increase in meshed surfaces (up to 50x), as well as the lack of change within the model after the culling feature was implemented. From a resolution and geometry standpoint, the "Elliot" blueberries (Figure 4.5) and "Triple Crown" blackberries (Figure 4.7) were especially well-rendered. The point cloud in the "Elliot" blueberry model contained 700,000 points, while the "Triple Crown" blackberry crop point cloud contained 900,000. Upon very close examination of these crops, branches and leaves can be distinguished, especially in the "Triple Crown" crop. The exact size and shape of these features, however, cannot be confirmed accurate through the current process.

The overall size of each model in the second data collection was measured through the measurement tool in the Meshlab software, as discussed in Section 4.3. These measurements, however, were ultimately deemed unusable, as measurements done in the field for this project could not be accurately matched and compared to the measurements done in Meshlab. Features were not mapped in the recording of the measurements, and led to great difficulty in trying to accurately recreate the measurements virtually. Markers with known locations should have been placed in the scene during data collection to allow accurate matching and verification to that of the rendered model (much like the calibration checkers were of known size and orientation). For this reason, they are not included in this project. This is an extremely critical area of improvement for future model rendering accuracy.

Additionally, the change in process produced models with more accurate color and geometric rendering, aside from those affected by sunlight (as discussed in Section 4.3). The

largest improvement in color return can be seen in the comparison between the black raspberry crops in the first and second data collection. The model in the second data collection (Figure 4.4) displays the signature white-purplish hue of the branches, whereas the first data collection displayed no such color (Figure 4.3). There is a yellow hue displayed in the "Elliot" blueberries (Figure 4.5), which is guessed to have been caused by a reflection of sunlight from the branches. The color of the branches should not have changed in color from the base of the crop. For further testing, cloudy, bright conditions are recommended for significant ambient light, but without direct sunlight.

Since this method of model creation included a semi-subjective success criteria, it is difficult to recommend the created models to be used in final simulation testing. Clever machine learning could perhaps overcome the uncertainty in the best of the models generated, as features are distinguishable to the human eye. It is likely that simulations over these models would be difficult, as the simulation sensor would approximate over a model that is itself an approximation, compounding the uncertainty. It is recommended that a better camera be utilized in future testing to accomplish the goal of a creating a sufficiently accurate model (for specific fruit picking or pruning). More points are needed over the model to capture enough of the complexities in the model to capture . If the crop treated solely as a rigid body, the models created from this project could serve adequately for general dexterity testing of a robotic arm. They serve as a good surface approximations of the real crop, and a general idea of how a robot could interact with various points in the environment could prove to be helpful. There is not enough accuracy for further use.

5.2 Future Improvement

The future improvement discussion will be divided into two sections: improvement of this process, and improvement of model creation endeavors of specialty crops.

5.2.1 Improvement of this Process

Based on the outcomes of this project, there are several ways to improve future endeavors into 3D crop rendering with a low-cost RGB-D sensor, including improvements to equipment, control of the environment, and changes to the technique. The first and easiest recommendation is to use the latest low-cost sensor technology, which would be to attempt this process with the Intel Realsense D435. Similarly priced, the Intel sensor offers a much larger pixel matrix (up to 1280x720, or approximately 4.25x the point cloud density per scan). Additionally, it allows a greater depth variation (0.1 - 10m vs. 0.5 - 4.5m), and is smaller and easier to power and maneuver [43]. This camera could provide an immediate improvement to this process.

To verify the size and shape of the crop rendered to the crop in the field, markers should be placed throughout the model in known, varying x, y, and z locations. Enough markers should be placed to properly encapsulate the majority of the important features, and the marker position vectors should be measured and recorded in each trial. This addition will create points to which the measurement tool in Meshlab can connect, allowing accurate, verifiable comparisons to the real locations of the markers and the rendered ones. Scans of each crop should be done with and without the markers present, allowing one model to be created specifically for model accuracy, and one to be used for simulation study.

In addition to large-scale measurements of the overall crop, measurement of specific features in a crop should be measured. Though small features could be found, they could not be verified to be accurate

Before the undertaking of this project, the effect of the outdoor environment and the control required to offset it was not entirely known. A distinct difference can be seen in both data collections when sunlight was directly shining on a crop and when it was not (direct sunlight is seen as a black color return). For best results, this study should be done again on a day without any direct sunlight, but enough ambient light to illuminate the scene well. As this is hard to control (and many roboticists have avoided allowing ambient sunlight altogether [14]), it is recommended to either wait for the ideal day (which is uncontrollable), move the crop indoors, or to shield the crop from the sunlight while outdoors and provide artificial light. If the third option is chosen, it would be important to shine the light neither directly toward the camera or toward the crop, as doing so produced errors in calibration and would likely reproduce similar errors in the field. This aspect of the data collection technique presents the most difficult challenge for optimization in future developments.

After data collection, the only variable aspect of the model rendering process was the meshing process in Meshlab. Within Meshlab, each step in the model rendering process has multiple adjustable parameters. As was discussed, each step had its own partial optimization technique, but the fact was unaddressed that the adjustable parameters provide hundreds of potential rendering pathways, the whole gamut of which was not investigated. It is likely that for each individual scene, there is a different, optimum process to create the best rendering of that model, and that the local optimum of each step would not necessarily provide the overall optimum model. An algorithm could be developed to find this true optimum pathway, and if done, has the potential to greatly improve the model resolution and turnout.

5.2.2 Improvements for Low-Cost Sensor Endeavors of Specialty Crops

While improvements to this process could generate more robust scans and provide better models, there are two inherent shortcomings of the project that should be addressed to greatly increase the model rendering capability of a low-cost sensor. These are: the need to define all success criteria objectively, and to minimize human input from this experiment.

Though the process produced models that are useful in their own right, accuracy can never truly be minimized because of the human input into the process (the alignment of the point clouds was partially subjective, development of the parameters was subjective, and evaluation of success was largely subjective). The first inherent shortcoming is that this process did not provide completely objective success or failure criteria. Much of this was due to a lack of expertise in the algorithms needed to provide such information and the lack of necessary computing power. While Meshlab assists in some of this (such as the accuracy of the alignment of the point clouds), an algorithm to detect continuity of the model, distinguishability of features, color return accuracy, and other such details should be developed to better evaluate success of this process. Markers of known distances (both from each other and from the ground) should be placed into the scene to allow objective verification of x, y, and z coordinate accuracy. All of these analyses would be contained within an objective function which would combine the separate success criteria and conclude an objective success value.

There is ample opportunity to automate the process developed through this project. The alignment of features could be done by adding a known, unique feature in each scan that

can be matched exactly from different orientations to allow exact alignment algorithms to be developed. The caveat is that the object would have to be distinct in all 360 degrees, as alignment to say a circular or cylindrical feature could easily result in a skewed "successful" match. It would also have to be sufficiently discrete so as to not detract from the point cloud generation of the crop. Additionally, a camera could be alternated with the depth sensor to capture a .JPG image (with a same pixel matrix size to the depth sensor) to compare to the RGB values in each one. This would allow and objective RGB return success to take place. Finally, the algorithm developed would have to take into account the changing projections as the mounting height of the camera and the distance from the target object are changed. Though this was minimized in this project by defining a scan radius and camera height and orientation that was unchanged throughout a trial, to maximize the effectiveness of the automated process, the software would have to account for inevitable errors in trial consistencies. The automated rendering improvement described in the previous section would be combined with this automated point cloud alignment to fully automate the process. It could then run the objective function and build a model with a value of success.

Automation of this process has interesting implications when applied to the data collection method, as it could fully minimize the need for human interaction. A robot could be developed to collect data, although the development of its process to localize the center of a crop correctly, identify the proper distance away and height to scan, and the rest of the discussed complexities would be far more complicated than the development of the model rendering process. This robot could be similar to the developed pruning and harvesting robots where a chassis is built to contain a robot arm with cameras and sensors to allow it to scan any desired angle toward the crop. Additionally, a programmable, scanning drone could be created much more simply, and for a much lower cost. The cost of a programmable quadcopter drone capable of powering an Intel Realsense camera, the camera itself, and a majority of the necessary power and other such equipment could cost less than $1000. If the proper objective algorithms are developed (and prove to be worthwhile), these drones could be a low-cost alternative to the much more expensive (generally between $2000 and $50,000) mapping drones.

The main foreseeable difficulties are that 1) the robot would need to accurately know many of the aspects of the crop before it scans them, essentially needing to know the information it is tasked with finding out, and 2) it would need to successfully create sufficient ambient light and block direct sunlight in any season. If those challenges are overcome, the development of a robot of this kind would be a novel and helpful asset for roboticists in the field of agriculture and experimental farmers alike. It could allow a 3D rendering of how a crop was changing, allow a large model library to be created for the automated harvesting and pruning community to draw upon, and could be usable in almost any agricultural setting (and many other 3D model rendering applications).

5.3 Closing Remarks

This initial endeavor into a way for roboticists in agriculture to have a universal method to render large specialty crop models is to serve as a baseline for increasingly useful and accurate model creation. The hope is that future improvement can utilize this method to create an impactful, worthwhile difference in the agriculture community, and from it larger society.

Appendix A

Developed Python Code

This code was developed specifically for this project.

```
#!/usr/bin/env python

import roslib
roslib.load_manifest('point_cloud_saver')
import sys
import numpy as np
import message_filters
import cv2
import time
import rospy
from sensor_msgs.msg import Image
from std_msgs.msg import String
from cv_bridge import CvBridge, CvBridgeError
import numpy as np
from PIL import Image as Image2
import ros_numpy
import time

class image_converter:

		#Subscribe
	def __init__(self):

		self.bridge = CvBridge()

		image_subscriber = message_filters.Subscriber("/kinect2/sd/image_color_rect",Image)
		print "subscribed to /kinect2/sd/image_color_rect"
		depth_subscriber = message_filters.Subscriber("/kinect2/sd/image_depth_rect",Image)
		print "subscribed to /kinect2/sd/image_depth_rect"

		self.ts = message_filters.ApproximateTimeSynchronizer([image_subscriber,depth_subscriber], 1, 1)
		self.ts.registerCallback(self.callback)

		self.n = 0

	def callback(self,rgb_data,depth_data):

		try:
			#bridge = CvBridge()
			color_image = self.bridge.imgmsg_to_cv2(rgb_data,"bgr8")
			depth_image = self.bridge.imgmsg_to_cv2(depth_data,"passthrough")
			depth_array = np.array(depth_image, dtype=np.float32)
			color_array = np.array(color_image, dtype=np.uint8)
			# depth_array[np.isnan(depth_array)] = 0
			#cv2.normalize(depth_array, depth_array, 0,1,cv2.NORM_MINMAX) - Don't want to normalize
			print "depth_array is saving"
			# print np.max(depth_array)
			# print np.min(depth_array)
			# print np.nanmax(depth_array)
			# print np.nanmin(depth_array)
			# print len(depth_array)
			# print len(depth_array[1])
```

```
                        #np.savetxt('test.xyz', (x,y,z))
                        cv2.imshow('Depth Image', depth_array)
                        cv2.imshow('Color Image', color_array)
                        cv2.waitKey(2)

                        # rospy.loginfo(image.shape)
                        cv2.imwrite('/home/nickalvey/camera_rgb.jpg', color_array)
                        cv2.imwrite('/home/nickalvey/camera_depth.pgm', depth_array)

                        depth_array = depth_array/float(1000) # Converts the value to a depth in meters - *8m is the maximum sensing depth - unreliable at this distance

                        # fx = 367.286994337726/1000
                        # fy = 367.286855347968/1000
                        # cx = 255.165695200749/1000
                        # cy = 211.824600345805/1000
                        # k1 = 0.0914203770220268
                        # k2 = -0.269349746097515
                        # k3 = 0.0925671408453617
                        # p1 = 0
                        # p2 = 0

                        fx = 4.0236858073093703e+02
                        fy = 4.0146649517883532e+02
                        cx = 2.6912688564328454e+02
                        cy = 1.9766080937182471e+02
                        k1 = 2.7651809577446890e-01
                        k2 = -4.8929587572587707e-01
                        k3 = -3.6666892712909709e+00
                        p1 = -1.2600271497938275e-03
                        p2 = 1.1124924395238512e-02


                        f = open('/home/nickalvey/black_raspberry_continuation_' + str(self.n) + '.txt', 'w')
                        for y in range(len(depth_array)):
                                for x in range(len(depth_array[y])):

                                        x_value = (x - cx)*(depth_array[y][x]+.0768469)/float(fx)
                                        y_value = (y - cy)*(depth_array[y][x]+.0768469)/float(fy)

                                        # r = np.sqrt((x_value)**2+(y_value)**2)

                                        # x_correction = x_value*(1+k1*r**2 + k2*r**4 + k3*r**6)
                                        # y_correction = y_value*(1+k1*r**2 + k2*r**4 + k3*r**6)

                                        # x_corrected = x_correction + (2*p1*x_correction*y_correction + p2*(r**2 + 2*x_correction**2))
                                        # y_corrected = y_correction + (p1*(r**2 + 2*y_correction**2) + 2*p2*x_correction*y_correction)

                                        f.write(str(x_value))
                                        f.write(",")
                                        f.write(str(y_value))
```

```
                                        f.write(",")
                                        f.write("%.5f" % (depth_array[y][x]+.0768469))
                                        f.write(",")
                                        f.write(str(color_array[y][x][0]))
                                        f.write(",")
                                        f.write(str(color_array[y][x][1]))
                                        f.write(",")
                                        f.write(str(color_array[y][x][2]))


                                        # f.write(str((x - cx)*(depth_array[y][x]+.0768469)/float(fx)))
                                        # f.write(",")
                                        # f.write(str((y - cy)*(depth_array[y][x]+.0768469)/float(fy)))
                                        # f.write(",")
                                        # f.write("%.5f" % (depth_array[y][x]+.0768469))
                                        # f.write(",")
                                        # f.write(str(color_array[y][x][0]))
                                        # f.write(",")
                                        # f.write(str(color_array[y][x][1]))
                                        # f.write(",")
                                        # f.write(str(color_array[y][x][2]))

                                        # f.write(", ")
                                        # f.write(str(color_array[y,x]))
                                        f.write('\n')

                        f.close()
                        print "File has been written"
                        print "A scan has been completed. Reinitiating in 15 seconds. Pleace hit Ctr + C to cancel another
scan."
                        time.sleep(15)

                        self.n = self.n+1

                except CvBridgeError as e:
                        print "The Try Statement Failed"
                        print e


def main(args):

        print "Initiating point_cloud_saver"
        ic = image_converter()
        rospy.init_node('image_converter', anonymous=True)
        try:
                rospy.spin()
        except KeyboardInterrupt:
                print "Shutting Down"
        cv2.destroyAllWindows()

if __name__ == '__main__':

        main(sys.argv)
```

Appendix B

Data Collection Form

Attached here is the data collection form developed for this project.

File Name: _____

Plant Type: _____

Plant Age: _____

Location: _____

Important/Notable Information: _____

_____

Planar/Non-Planar (180/360): _____

Number of Scans to be taken: _____

Height (Upper or Lower Section): _____

Depth from Plant (Radius of Scan Circle): _____

Orientation of Kinect 2: _____

Other Information: _____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Appendix C

Calibration Raw Data Results

Attached here is the raw calibration output from the Kinect iai_kinect2 package.

Calibrate Color:

[INFO] [CameraCalibration::calibrateIntrinsics] re-projection error: 0.167574

[INFO] [CameraCalibration::calibrateIntrinsics] Camera Matrix:
[1109.787855242609, 0, 950.3998716860669;
 0, 1108.486765988225, 469.6867701210928;
 0, 0, 1]
[INFO] [CameraCalibration::calibrateIntrinsics] Distortion Coeeficients:
[0.08658483597637615, 0.7932864505773665, -0.01657176348026653,
0.004834081659840905, -6.482391757095931]

(100 Images)

Calibrate IR:

[INFO] [CameraCalibration::calibrateIntrinsics] re-projection error: 0.0890217

[INFO] [CameraCalibration::calibrateIntrinsics] Camera Matrix:
[402.368580730937, 0, 269.1268856432845;
 0, 401.4664951788353, 197.6608093718247;
 0, 0, 1]
[INFO] [CameraCalibration::calibrateIntrinsics] Distortion Coeeficients:
[0.2765180957744689, -0.4892958757258771, -0.001260027149793827,
0.01112492439523851, -3.666689271290971]

(100 Images)

Calibrate Sync:

[INFO] [CameraCalibration::calibrateExtrinsics] Camera Matrix Color:
[1109.787855242609, 0, 950.3998716860669;
 0, 1108.486765988225, 469.6867701210928;
 0, 0, 1]
[INFO] [CameraCalibration::calibrateExtrinsics] Distortion Coeeficients Color:
[0.08658483597637615, 0.7932864505773665, -0.01657176348026653,
0.004834081659840905, -6.482391757095931]

[INFO] [CameraCalibration::calibrateExtrinsics] Camera Matrix Ir:
[402.368580730937, 0, 269.1268856432845;
 0, 401.4664951788353, 197.6608093718247;
 0, 0, 1]
[INFO] [CameraCalibration::calibrateExtrinsics] Distortion Coeeficients Ir:

[0.2765180957744689, -0.4892958757258771, -0.001260027149793827,
0.01112492439523851, -3.666689271290971]

[INFO] [CameraCalibration::calibrateExtrinsics] calibrating Color and Ir extrinsics...
[INFO] [CameraCalibration::calibrateExtrinsics] re-projection error: 0.151311

[INFO] [CameraCalibration::calibrateExtrinsics] Rotation:
[0.9997806907742494, -0.006610479745502609, 0.0198713842622737;
 0.006172177593599414, 0.9997380824201222, 0.02203793961988995;
 -0.02001186095088881, -0.02191045678371735, 0.9995596316902814]
[INFO] [CameraCalibration::calibrateExtrinsics] Translation:
[-0.05935726750341991;
 0.008390397126743332;
 -0.05985254442442067]
[INFO] [CameraCalibration::calibrateExtrinsics] Essential:
[0.0002015130729931982, 0.05965303055719175, 0.009705729021864925;
 -0.06102726759294689, -0.0009048908117988042, 0.05814177553452744;
 -0.008754920631747621, -0.05928625624130419, -0.001474840682660618]
[INFO] [CameraCalibration::calibrateExtrinsics] Fundamental:
[5.368623967952382e-08, 1.592821197266832e-05, -0.002122403796581583;
 -1.627770377756514e-05, -2.419023839661132e-07, 0.01066853890947268;
 0.005005876235221542, -0.03259280252050514, 1]

(100 Images)

Calibrate Depth:

[INFO] [DepthCalibration::compareDists] stats on difference:
     avg: 0.0768469
     var: 6.03179e-06
  stddev: 0.00245597
     rms: 0.0768862
  median: 0.077381

Bibliography

[1] *Ag and Food Sectors and the Economy*. https://www.ers.usda.gov/data-products/ag-and-food-statistics-charting-the-essentials/ag-and-food-sectors-and-the-economy/. Accessed: 2019-01-10.

[2] Linda Calvin and Philip Martins. "The U.S. Produce Industry and Labor: Facing the Future in a Global Economy". In: *Economic Research Service* (Nov. 2010), pp. 1–57.

[3] A. Gonzalez-Barrera. "More Mexicans Leaving Than Coming to the U.S." In: 19 (Nov. 2015), Online.

[4] Manoj Karkee and Qin Zhang. "Mechanization and Automation Technologies in Specialty Crop Production - September 2012". In: *Engineering and Technology for Sustainable World - September 2012* (Sept. 2012), pp. 16–17.

[5] C. J. Hawthorn, K. P. Weber, and R. E. Scholten. "Noncitrus Fruits and Nuts 2016 Summary 06/27/2017". In: *National Agricultural Statistics Service* 27 (June 2017), pp. 1–121.

[6] Long He and James Schupp. "Sensing and Automation in Pruning of Apple Trees: A Review". In: *Agronomy Journal* 30 (Sept. 2018).

[7] Noboru Noguchi et al. "Field Automation Using Robot Tractor". In: *Automation Technology for Off-Road Equipment* 26 (July 2002), pp. 239–245.

[8] M. Kise, Q. Zhang, and F. Rovira Mas. "A Stereovision-based Crop Row Detection Method for Tractor-automated Guidance". In: *Biosystems Engineering* 90.23 (Feb. 2005), pp. 357–367.

[9] Dario Floreano and Robert J. Wood. "Science, technology and the future of small autonomous drones". In: *Nature* 521 (May 2015), pp. 460–466.

[10] Steven A. Fennimore and Douglas J. Doohan. "The Challenges of Specialty Crop Weed Control, Future Directions". In: *Weed Technology* 2 (Apr. 2008), pp. 364–372.

[11] *What is a Specialty Crop?* https://www.ams.usda.gov/services/grants/scbgp/specialty-crop. Accessed: 2019-01-17.

[12] J.R. Feucht and H. Larsen. *Training and Pruning Fruit Trees - Fact Sheet No. 7.003*. Sept. 2009.

[13] E. J. Van Henten, C. W. J. Hol D. A. Van 't Slot, and L. G. Van Willigenburg. "Optimal Design of a Cucumber Harvesting Robot". In: *Proceedings of AgEng 2006* 3–7 (Sept. 2006).

[14] Tom Botterill et al. "A Robot System for Pruning Grape Vines". In: *Journal of Field Robotics* 34.4 (Aug. 2016), pp. 1100–1122.

[15] Christopher Lehnert et al. "Autonomous Sweet Pepper Harvesting for Protected Cropping Systems". In: *IEEE Robotics and Automation Letters* 2.2 (Apr. 2017), pp. 2377–3766.

[16] Christopher Lehnert, Tristan Perez, and Christopher McCool. "Optimisation-based Design of a Manipulator for Harvesting Capiscum". In: *ICRA 2015: IEEE International Conference on Robotics and Automation* 26 (May 2015).

[17] J. Davidson et al. "Hand-Picking Dynamic Analysis for Undersensed Robotic Apple Harvesting". In: *American Society of Agricultural and Biological Engineers* 59 (May 2016), pp. 745–758.

[18] Karina Gallardo, Mykel Taylor, and Herb Hinman. *2009 Cost Estimates of Establishing and Producing Gala Apples in Washington*. Feb. 2012.

[19] Lenka Pitonakova et al. "Feature and performance comparison of the V-REP, Gazebo and ARGoS robot simulators". In: *Proceedings of the 19th Towards Autonomous Robotic Systems Conference* (Feb. 2018).

[20] Lucas Nogueira. "Comparative Analysis Between Gazebo and V-REP Robotic Simulators". In: *Universidade de Campinas* (2014).

[21] Jamie Carter et al. *Lidar 101: An Introduction to LIDAR Technology, Data, and Applications*. National Oceanic and Atmospheric Administration, 2012.

[22] Adar Vit and Guy Shani. "Comparing RGB-D Sensors for Close Range Outdoor Agricultural Phenotyping". In: *Sensors (Basel, Switzerland)* 18.13 (Dec. 2018).

[23] Hamed Sarbolandi, Damien Lefloch, and Andreas Kolb. "Kinect range sensing: Structured-light versus Time-of-Flight Kinect". In: *Computer Vision and Image Understanding* 139 (Oct. 2015), pp. 1–20.

[24] *About ROS*. http://www.ros.org/about-ros/. Accessed: 2018-11-20.

[25] Morgan Quickly, Brian Gerkey, and William D. Smart. *Programming Robots With ROS: A Practical Introduction to the Robot Operating System*. Vol. 1. O'Reilly Media, Inc., Dec. 2015.

[26] *Meshlab*. http://www.meshlab.net/. Accessed: 2018-11-20.

[27] *Advanced Techniques on Manipulating the Models*. http://www.cs.cmu.edu/~reconstruction/advanced.html. Accessed: 2018-11-20.

[28] *Meshing Point Clouds*. http://meshlabstuff.blogspot.com/2009/09/meshing-point-clouds.html. Accessed: 2018-11-20.

[29] Marija Popovic. *Working With Meshes in Gazebo*. https://github.com/ethz-asl/rotors_simulator/wiki/Working-With-Meshes-in-Gazebo. Accessed: 2018-11-20.

[30] Sharon Nissimov, Jacob Goldberger, and Victor Alchanatis. "Obstacle detection in a greenhouse environment using the Kinect sensor". In: *Computers and Electronics in Agriculture* 113 (Apr. 2015), pp. 104–115.

[31] Joan R. Rosell-Polo et al. "Kinect v2 Sensor-Based Mobile Terrestrial Laser Scanner for Agricultural Outdoor Applications". In: *IEEE/ASME Transactions on Mechatronics* 22 (Dec. 2017), pp. 2420–2427.

[32] *Converting between ROS images and OpenCV images (Python)*. http://wiki.ros.org/cv_bridge/Tutorials/Con Accessed: 2018-10-1.

[33] *OpenCV Library*. https://opencv.org/.

[34] Lingzhu Xiang et al. *LibFreenect2*. ://github.com/OpenKinect/libfreenect2n.

[35] *iai_kinect2*. https://github.com/code-iai/iai_kinect2.

[36] *Kinect Coordinate Mapping: Summary and Pitfalls*. http://blog.elonliu.com/2017/03/18/kinect-coordinate-mapping-summary-and-pitfalls/.

[37] *Kinect2 Calibration*. https://github.com/code-iai/iai_kinect2/tree/master/kinect2_calibration. Accessed: 2018-12-20.

[38] *Camera Calibration*. https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_calib3d/py_calibration/py_calibration.html#calibration. Accessed: 2018-12-20.

[39] *Downsampling a PointCloud using a VoxelGrid filter*. http://pointclouds.org/documentation/tutorials/voxel_grid.php. Accessed: 2019-1-20.

[40] *North Willamette Research and Extension Center*. https://extension.oregonstate.edu/nwrec. Accessed: 2019-01-05.

[41] *Berry Crops Production Systems Program*. https://agsci.oregonstate.edu/berries-and-small-fruits. Accessed: 2019-01-05.

[42] Amanda Vance. personal communication. Jan. 15, 2019.

[43] *Intel® RealSense™ Depth Camera D400-Series*. https://software.intel.com/en-us/realsense/d400. Accessed: 2019-2-1.