FINAL REPORT

Automated 3D Model Building Using a Robot and an RGB-Depth Camera

Michigan State University
ECE 480 - Senior Design
Team 2
29 April 2015

Sponsor: Dr. Daniel Morris
Facilitator: Dr. Hayder Radha

Manager: David Zoltowski
Webmaster: Nick Saxton
Lab Coordinator: Nick Zuzow
Document Prep: Kexiang Qian
Presentation Prep: Jacob Kneibel
Executive Summary

With the widening availability of RGB-Depth cameras to industry and to consumers, new applications for these sensors are in high demand. 3D modeling is one such application, and while done before in various research projects, the process of using RGB-Depth cameras to model a space has yet to become an automated process. The goal of this project was to combine a robotic moving platform and a commercially available RGB-Depth camera to autonomously build a 3D model of an unknown environment.

We used an iRobot Create 2 Programmable Robot, a Microsoft Kinect for Windows v2 camera, a laptop, and requisite power and structural supports to create a robotic device. The device can be placed into a room and remotely controlled to start. Once started, the device executes movements to navigate a room.

Movement is powered by the iRobot, and the iRobot is controlled using a Python script. At each decision step, the script incorporates depth sensing data from the Kinect camera to make a decision. Throughout the process, a 3D model building software is also acquiring information from the Kinect to build a 3D model that can be saved and viewed upon completion.

The device does a mostly successful job in avoiding objects, but it does have a few problems, such as when no clear path is present or the device does not turn enough to clear an object at its sides. However, despite these issues, we were able to successfully demonstrate the use of the robot in multiple situations.
Acknowledgement

ECE 480 Design Team Two would like to thank the following people: first, our sponsor, Dr. Daniel Morris, for his continued availability and guidance throughout the development of our project. Dr. Morris is an expert in robotic devices and gave us many useful suggestions in our discussions. Also, he understood when we encountered problems and was always flexible to look for the best solution. Next, we would like to thank our facilitator, Dr. Hayder Radha, for his advice and critique with the technical documents. We would also like to thank Matthias Niessner and Michael Zollhoefer for their Voxel Hashing code and for Matthias Niessner’s correspondence while we worked with their code. Finally, we want to recognize our team member, Kexiang Qian, for his generosity in allowing us to use his laptop for the project. The laptop given to us by the ECE department did not have sufficient processing power to utilize the desired 3D modeling software, but Kexiang’s laptop did.
Table of Contents

1 Introduction and Background 5
   1.1 Introduction 5
   1.2 Challenges 5
   1.3 Background 6
   1.4 Summary 7

2 Exploring the Solution Space and Selecting a Specific Approach 8
   2.1 FAST Diagram 8
   2.2 Design Specifications 9
   2.3 Conceptual Design Descriptions 9
   2.4 Feasibility and Decision Matrices 13
   2.5 Chosen Solution 14
   2.6 Solution Summary 18
   2.7 Initial Project Management Plan 18
   2.8 Initial Estimate of Budget 18
   2.9 Initial and Final Gantt Charts 19

3 Technical Description of Work Performed 20
   3.1 Hardware 20
   3.2 Software 26

4 Testing 32
   4.1 Test Run 1 32
   4.2 Test Run 2 33
   4.3 Test Run 3 34
   4.4 Capabilities 35
   4.5 Shortcomings 35

5 Final Cost, Schedule, Summary, and Conclusions 37
   5.1 Final Budget 37
   5.2 Final Budget (projected cost per unit for multiple units) 38
   5.3 Final Schedule 38
   5.4 Future Work 39
   5.5 Summary / Conclusion 39

Appendix 1: Technical Roles, Responsibilities, and Work Accomplished 40
   Jacob Kneibel 40
   Kexiang Qian 41
   Nick Saxton 42
   David Zoltowski 43
   Nick Zuzow 44

Appendix 2: Literature and Website References 45

Appendix 3: Detailed Technical Attachments 46
1 Introduction and Background

In this section, we provide an introduction to our project and discuss relevant topics associated with our project. First, we state our project goal, introduce our project objectives, and comment on potentially challenging aspects. Then, we provide background on current work in this area. Finally, we recapitulate how our project builds on this work, describe a successful project, and comment on the contributions a successful project would make.

1.1 Introduction

The goal of this project is to produce a combined piece of hardware (robot, camera, and onboard computer) capable of entering an unknown environment and autonomously building a 3D model of that environment. It is required that the modeling process be completed in a reasonable amount of time, which is defined as approximately less than thirty minutes. An additional requirement is that the robot must navigate the room without getting stuck so that it is able to complete the model. A basic requirement for the 3D model is that it includes objects up to and including the height of a table. Two other major requirements are reliability and reproducibility. First, the robot should be able to build a complete 3D model a large majority of the time. Next, all of the hardware and software should be documented and delivered to the sponsor so that the project can be replicated and further developed. Provided the previously mentioned requirements are met, a secondary, performance-related goal is adding the ability to produce a model that incorporates objects above table height while also factoring in texture and color.

1.2 Challenges

Several challenging aspects are involved in automating 3D modeling. First, there are several hardware related issues that have to be addressed to make it a truly automated process. The RGB-Depth camera, computer, robot, and associated power supplies will have to be combined into a single structure that allows the robot to navigate the room. The structure will have to be light enough so that the robot can still move and stable enough so that the camera can take good images. Additionally, power supplies for the RGB-Depth camera will have to be developed so that it can be untethered from a power outlet.

An additional challenge is developing software to integrate all of the hardware components. The computer will be connected to the RGB-Depth camera and the robot. The data from the RGB-Depth camera must be used for two tasks: the data must be used to develop a 3D model and the data must be incorporated into a script to control the robot’s movement. Luckily, algorithms and software have been developed to perform 3D model reconstruction from RGB-Depth cameras, and we will investigate using a previously developed piece of software for that task. If the software does not work as desired or cannot be run on our computer, we will investigate developing a simple algorithm to perform the 3D model reconstruction.

While the 3D modeling process is automated, it does require human interaction to initiate the robot movement and camera data collection. Since the computer controlling the robot and the camera is onboard, a non-cumbersome method of starting the scripts is required. Additionally, it is helpful for debugging and demonstration purposes to be able to see in real-time the commands
the robot is executing and the image the camera is currently collecting from an offboard location. A remote desktop client provides all of this functionality and there are many free, quality options available.

1.3 Background

1.3.1 Overview

With current technology, the traditional 2D floor plan does not satisfy people’s desires. In commercial activities, 2D panoramic photos and rotated continuous photos have been used to display 3D environments. The market for 3D models is potentially large, and 3D modeling has important uses including exploring an unknown environment and detailing objects for 3D printing.

Before the advent of RGB-Depth cameras, 3D models of objects had to be constructed one of three ways: by hand in a 3D modeling software application, by using a 3D scanner, or by capturing several images of an object from various points of view with a traditional camera before stitching them together. The software method, while still used today to make movies and video games, can be very time consuming and can often contain slight inaccuracies. Research into the use of 3D scanners began in the 1980s and scanners capable of producing high-fidelity 3D models exist today. However, these scanners are typically quite slow and there is a trade-off between the size of an object capable of being modeled and the cost of the scanner [1]. Several images captured with a traditional camera can be stitched together to create a quasi-3D model, but this method fails to create a true model of the object and in order to get a final result that is close to resembling the original object, time-consuming software must be used for the stitching process [2].

Now that RGB-Depth cameras are widely available to consumers and commercial enterprises alike, these previous methods of 3D modeling can be replaced by a faster, more accurate approach. While being able to model individual objects in an easier manner with these cameras is certainly appreciated, they possess a more compelling feature, the ability to create 3D models of entire environments relatively quickly and relatively inexpensively. A fairly large amount of research has been done utilizing this capability, however most of it has been performed in a manual, hand-held way [3]. Automating this process would be very beneficial from several standpoints, from potential use in military operations to improved accessibility for physically handicapped people.

1.3.2 SLAM

In the field of robotics, the real-time mapping of an environment and knowledge of location is referred to as the Simultaneous Localization and Mapping (SLAM) problem. When the RGB-Depth camera captures an image, it receives location measurements with associated RGB and depth values. However, the camera does not know from what point in the environment it captured the image. To overcome this, the data from several different images of the environment needed to be combined to determine not only the robot’s path through the space but also the pose (angle relative to the path) it captured the image from. This can be a computationally intensive
process and most consumer notebook computers lack the necessary computing power. Therefore, a computer with a quality processor and dedicated graphics processing unit is required.

There has been a large interest in this area of research and several methods have been created that can solve this problem. Different sensors need different algorithms to properly produce a map of the environment. These different algorithms can vary in complexity based off of the sensors and the degree of accuracy required. One algorithm, FastSLAM [4], uses knowledge of the robot’s movement and path to create a relative location and then maps the area around the robot identifying landmarks individually. This method uses multiple sensors to partition the SLAM problem.

SLAM can be thought of as a precursor to 3D modeling. FastSLAM can be used to identify locations and relative positions of objects, but it does not provide a full 3D model. In the next section, we will discuss algorithms and software that do provide 3D models.

1.3.3 3D Model Reconstruction

Algorithms and software have been developed to perform 3D modeling. At its core, 3D model reconstruction is a very complicated and computationally expensive task. First, in a paper titled “Real-time 3D Reconstruction at Scale using Voxel Hashing,” Matthias Nießner and his team at Stanford University developed an efficient algorithm for 3D model reconstruction [5]. The algorithm utilizes a simple spatial hashing scheme that compresses space to allow for real-time access and updates of implicit surface data. Next, Microsoft and university researchers have developed Kinect Fusion [6,7]. Kinect Fusion is a piece of software used for 3D object scanning and model creation. The software allows a user to capture a scene with the Kinect camera while simultaneously viewing and interacting with a 3D model of the scene. In this project, we investigate using these algorithms and software to implement our 3D model reconstruction.

1.4 Summary

In our project, we will develop a robotic device that will perform automated 3D model reconstruction of an environment. Our project builds on previous work in 3D modeling by automating the process with a robot and by focusing on mapping room environments. Several challenges involve selecting hardware components, integrating all of the hardware components using software, developing algorithms to process data and control robot movement, data transmission, and 3D model building. A successful robot will complete the objectives of being able to navigate without getting stuck and completing a model of an environment, at least a majority of a time. We believe that we have the necessary skills and guidance to achieve this task, and will utilize all facets of the design process to achieve success. A successful device could be used to model dangerous environments, among other things, and could be extremely useful for military and industrial purposes.
2 Exploring the Solution Space and Selecting a Specific Approach

This section takes the critical customer requirements (CCRs) and examines possible design solutions while utilizing multiple engineering design tools including a FAST diagram, the House of Quality technique, and decision and feasibility matrices. Additionally, the final design decision and proposed budget will be discussed in detail. Finally, the initial schedule and plan will be introduced and compared to the final schedule.

2.1 FAST Diagram

A FAST (Function Analysis System Technique) diagram provides a visual representation of a product’s functionality based on a set of customer requirements. It eschews any consideration of possible design solutions and instead looks specifically at the problem at hand. Also, it makes very clear the separation between primary and secondary functions of the end product. The FAST diagram for our project is shown below in Figure 1.

It can be seen that the primary function of our project is to output a 3D model. In order to do this, two secondary functions are required: the ability to collect images and process images. Each of these functions in turn requires additional tertiary and quaternary functionality.

![Figure 1. FAST Diagram](image-url)
2.2 Design Specifications

The design criteria were separated into two different types: criteria that must have been satisfied and performance criteria. The criteria along with their type and weight are displayed in Table 1.

<table>
<thead>
<tr>
<th>Number</th>
<th>Criteria</th>
<th>Weight</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Full, autonomous navigation of room</td>
<td>5</td>
<td>Must</td>
</tr>
<tr>
<td>2</td>
<td>Completion of low-level 3D model</td>
<td>5</td>
<td>Must</td>
</tr>
<tr>
<td>3</td>
<td>Integration of color</td>
<td>3</td>
<td>Performance</td>
</tr>
<tr>
<td>4</td>
<td>Integration of texture</td>
<td>3</td>
<td>Performance</td>
</tr>
<tr>
<td>5</td>
<td>Completion of high-level 3D model</td>
<td>3</td>
<td>Performance</td>
</tr>
<tr>
<td>6</td>
<td>Reproducibility</td>
<td>4</td>
<td>Must</td>
</tr>
<tr>
<td>7</td>
<td>Reliability</td>
<td>4</td>
<td>Must</td>
</tr>
<tr>
<td>8</td>
<td>Speed</td>
<td>2</td>
<td>Performance</td>
</tr>
<tr>
<td>9</td>
<td>Model Quality</td>
<td>4</td>
<td>Performance</td>
</tr>
</tbody>
</table>

Table 1. Design Criteria

Criterion 1, full, autonomous navigation of room, required that the robot was able to navigate the room it was placed in without getting stuck. Criteria 2 and 5 detailed the height of the model. Completion of a low-level 3D model meant that the model was built to approximately a table height, while completion of a high-level 3D model meant that the model was built to a higher level. Criteria 3 and 4 were performance goals, which would be met by incorporating color and texture into the model, respectively. The entire product must have been documented and delivered to our sponsor (Criterion 6), so that the product can be improved in future work. Regarding reliability, criterion 7, the device must have been able to complete the 3D model a majority of the time and should have had a stable design. The final two criteria were performance goals. The faster the model was built the better, but we were to be satisfied if the device was able to produce the 3D model within 30 minutes. Finally, model quality referred to the fidelity of the model to the room.

2.3 Conceptual Design Descriptions

This subsection provides brief descriptions of several conceptual designs. Each proposed solution considers different hardware configurations with the necessary software components remaining constant.
Solution 1

In the first proposed solution, shown in Figure 2 below, an iRobot Create 2 Programmable Robot would be combined with a Kinect v2 camera and a laptop. The camera would be placed at about 1 meter above the iRobot using a steel structure. The Kinect v2 would be powered by an external power supply.

![Figure 2. Solution 1](image)

Solution 2

The second proposed design is similar to Solution 1 and can be seen in Figure 3 below. However, in this case, the camera would be lowered to a height of about 1.5 feet. Thus, while this limits the area that we can model above our robot, it would lower the robot’s center of gravity and allow the robot to move into more places, including under most tables.

![Figure 3. Solution 2](image)

Solution 3

A third possible design solution involves the use of a Lego Mindstorm robot for navigating the room, acting as a platform for a Kinect v2 connected to a laptop for data collection and transmission. The use of a Mindstorm robot opens up the possibility of adding additional sensors to help with room navigation. However, the smaller size of the robot could have problems supporting the camera and computer.
In the fourth design, depicted below in Figure 5, a 1.5-foot high steel rod structure will be built over the iRobot. The laptop will be placed over the top surface of the iRobot. There will be a bar to fix the Kinect camera at the end of the holder.

The fifth possible design solution would utilize a controllable arm as opposed to using a structure with a fixed height. This could allow the camera to be moved up and down, not only allowing more of the environment to be modeled but also providing an improved ability to avoid certain obstacles such as tabletops. The controllable arm would be affixed to the front portion of the robot with the computer placed behind it. However, the arm and the motor to position it would add a fairly significant amount of weight to the design while also adding another level of complexity in terms of programming the arm control. A visual representation of this design can be seen below in Figure 6.
Solution 6

In the sixth possible solution, the iRobot Create 2 and the Kinect v2 would still be utilized but a smaller, lighter, single-board computer in the form of a Raspberry Pi would be used instead of a laptop as seen in Figure 7. The benefits of this design would be reduced size and weight. On the other hand, using a Raspberry Pi would require an offboard computer for the model building and it features limited support for Windows, which could cause problems with the Kinect.

Solution 7

The seventh proposed design, seen below in Figure 8, would also use the iRobot and the Kinect v2 but the height and angle of the camera would be adjusted. In this solution, both the Kinect and the laptop would be placed directly on the iRobot. The Kinect would be mounted at a set angle, approximately 30 degrees, using a small wedge shaped object. This would allow for an adequate viewing height while minimizing the extra height of the physical equipment above the iRobot, allowing for easier navigation around certain obstacles that could be present in the room.
Solution 8

The final possible solution that we considered would use the iRobot Create 2 Programmable Robot, the Kinect v2 camera, and laptop in a manner similar to that of many of the other possible solutions. However, this solution would have the camera and robot connected to the laptop via a cord such that the robot could move about the environment while the laptop remained stationary. This design would remove a significant amount of weight from the robot in the form of the computer but would also introduce the potential for the cord to get caught on an object in the room. This design is depicted in Figure 9.

![Figure 9. Solution 8](image)

2.4 Feasibility and Decision Matrices

The most important objective was for the device to build a full 3D map of the room; therefore, we determined the most important design criteria were to build a device that can fully navigate a room without stopping and to build a low-level 3D map. Those were the first two feasibility criteria. The last two feasibility criteria were reliability and reproducibility. The feasibility of each of our designs is marked in Table 2. We were careful to only consider practical designs, and therefore all of our designs were feasible and were further considered in the solution decision matrix.

<table>
<thead>
<tr>
<th>Feasibility Criteria</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
<th>S6</th>
<th>S7</th>
<th>S8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full, autonomous navigation of room</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Completion of low-level 3D model</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Reproducibility</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Reliability</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 2. Feasibility Criteria
All of our designs were feasible and were considered in Table 3, the solution decision matrix. In this table, we rated each design on each of the design criteria on a scale 1-3-9 scale, where 9 indicated the highest rating and 1 the lowest. For each design, we summed the products of the criteria weights and ratings to determine the most desirable design. In this case, solution two and solution eight had the highest totals. The two designs were similar; the only difference was the angle of the camera. In our proposed solution, we proceeded with solution two but kept in mind solution 8.

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Weight</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
<th>S6</th>
<th>S7</th>
<th>S8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full, autonomous navigation of room</td>
<td>5</td>
<td>9</td>
<td>9</td>
<td>3</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>Completion of low-level 3D model</td>
<td>5</td>
<td>9</td>
<td>9</td>
<td>3</td>
<td>9</td>
<td>9</td>
<td>3</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>Integration of texture</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Integration of color</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Completion of high-level 3D model</td>
<td>3</td>
<td>9</td>
<td>9</td>
<td>1</td>
<td>3</td>
<td>9</td>
<td>3</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>Reproducibility</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Reliability</td>
<td>4</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Speed</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Model Quality</td>
<td>4</td>
<td>9</td>
<td>9</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>-</td>
<td>193</td>
<td>201</td>
<td>85</td>
<td>155</td>
<td>169</td>
<td>129</td>
<td>201</td>
<td>95</td>
</tr>
</tbody>
</table>

*Table 3. Solution Decision Matrix*

### 2.5 Chosen Solution

#### 2.5.1 Overview

Our proposed design solution is solution 2, briefly described above. The RGB-Depth camera that we will use is the Microsoft Kinect v2. This camera has a higher image quality and better depth and object recognition than the Kinect v1 [8]. Our navigation robot is the iRobot Create2 Programmable Robot. This robot will give us a platform for the camera that has multiple sensors that will be beneficial to the room mapping process. Additionally, the Create2 design can be modified to give us a stable platform with which we can attach mounting hardware to hold the laptop and Kinect. A detailed description of the proposed solution is given in this section.
2.5.2 Power and Structure

The Kinect camera has an individual power supply that connects to a wall outlet. However, our system needs to be wireless, so we need to develop a device to wirelessly power the camera. Our proposed solution will be to use a combination of 8 to 10 AA batteries to supply the 12 volts input needed for the Kinect 2.

A two-level, 1.5-foot holder will be built on top of the iRobot, as seen in Figure 9. The holder will cover the laptop and the Kinect v2 camera will be placed on top of the holder. This arrangement places the camera at a desirable height to collect images.

![Figure 9. Proposed structural design overview](image)

2.5.3 Room Navigation

The choice to use the iRobot Create 2 offers many sensors that can be utilized to effectively map the room, and maneuver around the room even in a cluttered room. The Create 2 can tell when it encounters an object, wall, or drop-off. It also has the ability to keep track of its relative position to where it started, using counters on the two main drive wheels. Input from the Kinect camera will be used to avoid objects and will be input into the room navigation algorithm. Ultimately, our algorithm will utilize the iRobot Create 2 sensors and movement abilities in conjunction with depth data from the Kinect to navigate.

2.5.4 Building a 3D Model

We propose using Kinect Fusion to build our 3D model. Kinect Fusion provides 3D model creation and object scanning, can be used to see a detailed 3D model of a scene, and has been previously effective in creating 3D models [6,7]. If Kinect Fusion is not adequate, we will investigate using the software developed in “Real-time 3D reconstruction at scale using voxel hashing” [5]. Microsoft provides a test program that determines if a computer is capable of utilizing the Kinect v2, and the laptop provided to us by the ECE department passes this test program. Therefore, we believe that the programs can be run on the laptop. However, if we are unable to use the programs on this laptop, we will investigate acquiring a different processing
device, transmitting the data to a different processing device, or developing a 3D model on our own.

2.5.5 Data Transmission

If data transmission to a different processing device were required, the most effective solution would have been to wirelessly transmit the data from the onboard computer connected to the Kinect camera over a network to an offsite machine. As for the transmission protocol, TCP would most likely have sufficed, as it offers a level of reliability that is not present with UDP, while the relatively slower speed would not have been an issue with the amount of data being transmitted. However, the laptop provided by our team member Kexiang proved computationally powerful enough to handle the modeling software.

2.5.6 High-Level Models

A high-level system architecture that details the interactions between the main three components, the robot, camera, and computer, is shown in Figure 10. Next, a high level model of our algorithm is presented in Figure 11.

**Figure 10.** High-level system architecture: The camera, robot and computer will interact to build the 3D model. The computer will process information from the camera, transmit information, and instruct the robot where to move. The robot will be the base for the computer, and will navigate the room for the camera to acquire images. The camera will transmit images to the computer and will be used to identify objects/areas to avoid.
Figure 11. High-level model of 3D model building algorithm: First, we will power the robot, camera, processing units, and any other necessary devices. The robot will be placed into a room and the mapping procedure will be initiated. The device will collect data from the camera and transmit the data to a processing unit. The processing unit will use the depth image data to estimate the location of the robot and will integrate the data into the 3D model. After updating the 3D model, the processor will determine any unviable locations for the robot to move. While this is happening, the robot will continue to move and send depth image data during its mapping process, and it will avoid any unviable locations. We will determine a confidence measure that will indicate how certain we are of our model and that the model is complete, and the method will be completed until we reach a certain confidence threshold in our model.
2.6 Solution Summary

Our chosen design solution combines two very powerful components, an iRobot Create2 Programmable Robot and a Microsoft for Windows Kinect v2 camera, with a laptop into a device fully capable of acquiring and processing images from an environment, navigating an environment, and producing a 3D model. On top of that, the navigation and acquisition process will be automated.

2.7 Initial Project Management Plan

The team working on this project consists of five members, each of whom were initially assigned various technical tasks within the project along with non-technical roles. Jacob Kneibel was responsible for the mapping and localization of the robot in the room, which also included a movement algorithm that the robot would follow throughout the room. Jacob was also responsible for managing presentations for the team. Kexiang Qian was responsible for the power supply of the Kinect camera and the device structure. Nick Saxton was responsible for researching and implementing the transmission of the sensor data from the onboard computer to the off board computer for computation, if necessary. Additionally, Nick designed and managed the team’s webpage. Nick Zuzow and David Zoltowski were both responsible for the 3D model building software. More specifically, Nick was responsible for interfacing between the software and the other devices, while David was responsible for using the 3D model to detect objects and avoid collisions.

2.8 Initial Estimate of Budget

The ECE Department provides us with a budget of $500, and we initially estimated a breakdown of our proposed expenses in Table 4. The table is complete with a description, cost, and justification for each item. Our project is inherently expensive; it requires a programmable robotic device to navigate a room and a depth camera to acquire images. The majority of our budget was proposed to be spent on those items and the rest of the expenses were proposed to go towards power and structural supports.

<table>
<thead>
<tr>
<th>Number</th>
<th>Item</th>
<th>Cost ($)</th>
<th>Justification</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>iRobot Create 2 Programmable Robot</td>
<td>199.99</td>
<td>The robot will be our navigation device.</td>
</tr>
<tr>
<td>2</td>
<td>Microsoft Kinect Camera for Xbox v2</td>
<td>149.99</td>
<td>The Kinect camera will acquire rgb-d images.</td>
</tr>
<tr>
<td></td>
<td>Description</td>
<td>Cost</td>
<td>Details</td>
</tr>
<tr>
<td>---</td>
<td>--------------------------------------</td>
<td>-------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>3</td>
<td>Kinect to Windows adapter</td>
<td>49.99</td>
<td>The adapter will connect the Kinect camera to our laptop for processing.</td>
</tr>
<tr>
<td>4</td>
<td>Power Supplies</td>
<td>79.99</td>
<td>The remaining budget will be used to power our devices.</td>
</tr>
</tbody>
</table>

Table 4. Initial Budget

2.9 Initial and Final Gantt Charts

The initial Gantt chart detailing our proposed schedule for project work and deadlines is shown in Figure 1 and Figure 2 of Appendix 3, and the final Gantt chart detailing the final schedule is shown in Figure 3 and Figure 4 of Appendix 3.

Our final schedule differed significantly from the initial, proposed schedule. Mainly, development of the Kinect-related software was delayed, which affected the rest of the software development and testing. At first, we were using the laptop given to us by the ECE department to run the software. The laptop was able to run the Kinect along with Kinect Fusion and other Kinect applications. However, after testing Kinect Fusion, we determined that it was inadequate for our model building purposes. We moved to use the Voxel Hashing software for our 3D model building. However, while this software could simply run on the laptop, it almost always crashed. We were in conversations with our sponsor to change our project goals when Kexiang Qian, one of our team members, lent us one of his laptops. His laptop has more processing power and was able to run the Voxel Hashing software. Once we acquired his laptop, we moved forward with development of the navigation script, 3D model reconstruction, and structure.
3 Technical Description of Work Performed

In this section, we will describe all of the work done to create our final prototype, which can be seen in Figure 5. First, we will discuss the hardware components, including the electronic devices, power supplies, and structure. Then, we will discuss the software we wrote and used to control and integrate the devices. As illustrated in Figure 12, the iRobot serves as the base of the device, and a structure was built on top of the iRobot to hold both the laptop and the Kinect camera.

![Figure 12. Final prototype.](image)

3.1 Hardware

3.1.1 Kinect and Kinect Power

The Kinect for Windows v2 sensor is a device that can sense depth, capture color images, emit infrared, and input audio [8]. More specifically, the sensor is capable of sensing the depth at a 30 Hz frequency and 512 x 424 resolution and for points further than 0.5 meters from the sensor. The color camera captures 1080p video, also at 30 Hz, and the infrared emitter has similar capabilities. The Kinect for Windows v2 sensor improves on the first version of the device, providing the designated technical spec but also improved body tracking, an expanded field of view, and higher depth fidelity [8]. The sensor used for the Xbox One, but can also be connected to and run on a Windows computer. All of these facts combine to form the notion that the Kinect
for Windows v2 sensor (hereafter simply called the Kinect) is a very powerful sensor for use in computer vision and human computer interaction technologies.

One problem with the Kinect sensor is that it cannot detect objects (make depth readings) of objects within 0.5 meters of it. This strongly limits the devices ability to see objects very close to it, and sometimes causes the device to run into objects. However, we feel that we dealt with this limitation well.

Kinect v2 sensor was purchased to act as our RGB-Depth camera and an adapter to connect it to Windows computers. A device connected to a wall outlet powers the Kinect, and we had to design a replacement mobile power supply.

Several options were researched to supply mobile power to the Kinect. Ultimately, a nickel-metal hydride (NiMH) Onyx 7.2V 5000mAH battery and an Adjustable Boost DC Regulated Power Supply Voltage Converter Module were chosen and ordered. The NiMH battery is a type of rechargeable battery that is high used in mobile phones, digital cameras, laptops and a large number of other electronic products. The advantages of this battery are that it has higher energy capacity per kilogram, is safer, and has a more stable output when compared to a nickel-cadmium battery [9]. The NiMH battery can efficiently support the Kinect camera working at least half an hour with a relatively lightweight and cheap price. Due to the fact that the battery is not able output exactly 12V all the time, a boost DC power is necessary to apply at the connection between the battery and the Kinect camera [10]. The supply has an adjustable resistor, which can help to modulate any output voltage at least slightly higher than input. This converter fits the requirement 12V inputs of Kinect and the ideal 7.2V output of the battery.

After the order was received, assembly of the power supply began. Due to Microsoft’s special standard, the DC power male plug is not sold on the market. The critical step was to cut the plug from the original DC power cable in order to build the mobile power. Another big issue was whether the 12V output from the boost converter module was stable. After testing the power output using a digital multimeter, the supply proved stable enough to be safely used with the Kinect camera. Further testing was conducted using the battery power supply while the Kinect was running and so far it has performed well. The Kinect can be normally continuously used for around 2 hours before the battery drains completely. The final power supply is shown in Figure 13.
3.1.2 iRobot Create 2 Programmable Robot

The iRobot Create 2 is a STEM resource that allows for educators, students and developers to conduct robotics research at an affordable price [11]. It is made from remanufactured Roomba platforms, and includes the full functionality of the Roomba without the vacuum components. The Create 2 was used as the robotic base for this project. It provided the mobile functionality, which was a key feature of a proper design.

One of the key benefits of the Create 2 over other similar robots is its open interface. The Create 2 has a serial port, and comes with a USB-to-serial communication cable which allows for commands to be sent to the robot from a connected computer. This was pivotal in the team’s ability to work on this project. With the availability of this feature the team was able to write a program that could control the robot’s movements based on input from the Kinect data. This program is described in more detail in section 3.2.

Another large benefit of the Create 2 is the ability to alter the exterior. iRobot defines the Create 2 as a “hackable” robot, which they endorse by including safe places to drill into and modify the exterior. This allowed the team to easily build and attach the structure, needed for the camera and laptop, to the robot. More detail about the design and creation of the structure can be found in Section 3.1.4.

3.1.3 Computer

Originally, our plan was to use the laptop provided to us by the ECE department. The ECE department laptop was a Windows laptop with 256 GB of storage, an Intel Core i5 2.70 GHz processors, 8 GB of memory, and an AMD Radeon HD 8600M Series graphics card. However,
laptop was unable to properly run the “voxel hashing” software, and our graphics card did not support the best version of the software.

Instead, Kexiang Qian loaned us one of his laptops, which we used for the final prototype. The laptop is a MacBook Pro with a 15-inch retina screen. It has a 2.6 GHz Intel Core i7 processor, 16 GB of memory, and a 500GB SSD hard drive. Most importantly, the laptop has an NVIDIA GeForce GT 750M graphics card. The graphics card was sufficient to run all of the requisite software.

3.1.4 Structure

The structure designed and built to support both the onboard computer and the Kinect camera was crucial to achieving our goals in this project. Our initial proposal called for the iRobot Create 2 itself to serve as a platform for the computer, with a metal structure added to the robot to support and elevate the Kinect camera. However, we continually refined the device structure throughout the life of the project.

We came across several issues once we received the iRobot and began to attempt to build our initially proposed structure on top of it. One issue was that when sitting directly on the robot, the laptop would conceal the robot’s buttons, which would need to be accessed. Additionally, the data cable made for an uneven platform when plugged in to connect the robot’s serial port to the computer. This forced us to consider adding a middle platform specifically for holding the computer to overcome these problems.

We were also limited in the areas that we could modify on the top of the iRobot, without necessitating a serious rework of the robot. Two small square sections on the back of the robot were all that was available to be drilled and cut in any fashion. Smaller drill points were also available but required a very specific drill size and depth to be used without damaging anything. This limited where we could put structural supports to hold platforms for both the Kinect camera and the computer.

Our next iteration of the structure used a commercially available plastic crate. This design positioned the crate on its side and utilized the crate’s ridges to fit on the robot squarely and evenly. The crate was secured to the Create 2 using a set of zip ties that went through the holes of the crate. The laptop was positioned on the inside of the crate, which provided an even surface unlike the first iteration. The Kinect was then positioned on top of the crate. This design, while achieving all of our main functional goals, lacked elegance, and the positioning of the Kinect was lacking.

We decided to then build a custom structure as our final iteration. We also decided another stipulation we wanted to add, was to have height of the Kinect camera variable, for testing purposes and to allow the final product to be adjusted based off of the needs of the user. The materials we decided to use were aluminum flat bars, and zinc sheets. These two main materials were chosen for the structural strength and low cost. The flat bars were used as the main skeleton and attachment points to the iRobot. The main size of the bars we used were ¾ in. by ⅛ in, and a few other parts used larger 1-⅛ in, by ⅛ in. bars. The zinc sheets were both 26 gauge and 16 in.
by 12 in. One was used for the laptop base and was riveted to the aluminum bars that would set it 3 inches above the iRobot. The base was attached to the iRobot with two contact points in the back each with 2 ¼ hex bolts that went through the cutting area. These bolts were ideal because they were flush with some ridges of the iRobot so regular function was uninterrupted and their size provided the strength to support the structure. Since there were no areas of the iRobot that could be drilled into near the front, two additional contact points were put in that would just put pressure on to the robot instead of being held in place. In Figure 14, an image of the structure, it is of the base and where it attaches to the Create 2 robot. In Figure 15, an image of the area that the laptop is placed.

![Figure 14](image.png)

**Figure 14.** The part of the structure that attaches to the Create 2. There are four points where hex bolts attach to the Create 2. And two contact points in front that support the weight of the structure but do not attach to the robot.
To support the Kinect camera another zinc sheet was used which had two strips of the main aluminum flat bar along the sides for structural support. This sheet was held up by 4 ⅜ inch diameter, 12 inch length threaded rods, one on each corner. Each rod had 4 ⅜ inch nuts on it; the two pairs would hold the Kinect sheet and the laptop base. Adjusting the nuts can raise or lower the Kinect. To increase the stability of the Kinect platform, two 16-inch sections of the larger aluminum flat bar were placed on top of the laptop base where the threaded rods connected, this restricted the movement of the threaded rods. During testing of this structure, we decided the Kinect camera should sit further back than our original placement of at the front, however positioning in back gave an issue of the Kinect platform being part of the depth image captured by the Kinect. To solve this we took a piece of the larger aluminum flat bar and bent it to provide an extra 2 ½ inches of height to the kinect from the platform. The Kinect was secured to the bar by a ¼ inch bolt that matched the threaded hole that the Kinect has in its base. This is the final design of the structure that is currently used as seen in Figure 12 above. In Figure 16, the top platform is shown where the Kinect 2 camera is affixed.
3.2 Software

3.2.1 Kinect Software

The software associated with the Kinect camera served two purposes. First, we needed to have software to acquire depth data from the camera and save it to a file. The file would then be used in the script controlling the iRobot navigation. For this purpose, we developed an application in Microsoft Windows Studio. The second purpose of the software associated with the Kinect was to build a 3D model. In this area, we tested two different software packages and ultimately used a code developed by the authors of “Real-time 3D Reconstruction at Scale using Voxel Hashing,” where this software will further be denoted as “voxel hashing” [5].

First, we will describe the work performed for the depth frame acquisition application. The application was developed in C# in Microsoft Visual Studio 2013, and it requires Windows 8 or 8.1 (x64) and the Kinect for Windows SDK 2.0 [12]. The application can be summarized as performing two main functions. First, the application reads the depth frame data from the connect and saves it to a file. Second, the application processes the depth frame data into an image and displays that image. The application has multiple functions that accomplish these tasks.

First, a function called OnLoaded connects with the Kinect sensor and starts depth frame reading functionality. A screenshot of this function is seemed in Figure 17. Next, each time a depth frame arrives, the depth frame is read. We chose not to save or display every depth frame.
because approximately thirty depth frames arrive every second, and that would burden the computer and was unnecessary for navigation purposes. Instead, we implemented a counter, which effectively told the application to process the depth frame once every half of a second. Each time we processed the depth frame, we both saved it to a file and transformed the data into a displayable format using a function found in an online tutorial [13]. The full source code for the application is in Appendix 3, and an example output screen is shown in Figure 18.

Figure 17. OnLoaded function.

Figure 18. Example application output.

While the depth data capture application was running, we concurrently ran 3D model building software. We did not develop the software, but we tested two different packages and chose the higher performing software.
First, we investigated using Kinect Fusion for 3D modeling, which is a part of the Kinect for Windows SDK 2.0. However, the Kinect Fusion software was better suited for building a model of one object, and did not perform very well when the Kinect camera was moved. Due to that problem, we tested the voxel hashing software. The software is available online at [14].

The technical work performed regarding this structure involved installing it onto our machine and running the software each time we want to build a 3D model. The software has two different versions: the first is titled “DepthSensing” and the second is titled “DepthSensingCuda.” “DepthSensingCuda” is more advanced, but it requires an NVIDIA graphics card, which we had on our new laptop. The other requirements for the software are the DirectX SDK June 2010, Kinect SDK previous to 2.0, NVIDIA CUDA 6.5, and for our purposes (using a Kinect v2 camera), the Kinect SDK 2.0.

Once the software is installed, it is simple to start the 3D model creation process. First, connect the Kinect camera. Then, run the program “DefaultBinaryDump.bat.” Once this program is running, it will collect all of the data from the Kinect, and the Kinect should be moved to cover the desired area. When we were creating our 3D models, we started this program before starting our robot navigation script. Once you are done collecting data, simply press “9” and a 3D model scan is saved to a file for viewing, which is what we did after our robot navigated for a sufficient amount of time. Example models created by this software with data that we collected will be shown in the test data section.

### 3.2.2 Robot Navigation Software

We utilized a script written in the Python programming language to interface between the onboard computer and the iRobot Create 2’s serial port for the purpose of providing the robot with the commands it required to autonomously navigate unknown environments. This script underwent several iterations and many rounds of testing before we arrived at the final version.

Throughout the project, we made use of the iRobot Create 2 Open Interface specification document available on the iRobot website. This document contained all of the necessary command codes that we would need to control the robot. It also provided the basic serial port connection information we needed to successfully connect to the robot. The Open Interface specification informed us that the Create 2 communicates at 115200 baud (the rate of symbols received/sent per second). Additionally, it provided descriptions of the various modes that the robot is capable of operating in. We needed to place the robot in Full mode for all of the driving commands while ensuring that we placed it in Passive mode when not in use to conserve battery life.

Initially, we planned on writing the robot navigation software in the same language that was used to create the Kinect depth application, namely C#. This would allow us to easily take the depth information acquired by the Kinect and use it to avoid running into obstacles with the robot without the need to transfer the depth data between two separate programs. However, implementing serial communication with the robot in C# proved to be overly complex and time consuming so we decided to go with Python at the expense of finding a way to still access the depth data.
Our first iteration of the robot control code served mainly as a method to ensure that we could successfully send commands to the robot through the serial port. The initial Python script used the freely available pySerial library, which provided all of the serial communication code we needed. This allowed us to focus on writing functions to get the robot to perform certain operations such as driving forward or rotating in place. The first program we wrote allowed us to manually control the robot by using the computer’s up and down arrow keys to drive forward and backward respectively. Additionally, the left and right arrow keys could be used to rotate the robot, allowing for turns in the robot’s path. However, this code only served as a starting point for automating this process.

The second iteration of the robot navigation code implemented rudimentary autonomous control. The Kinect depth code was modified to write current depth data to a file every few seconds. The Python script then read this depth data file before sending a command to the robot. We first experimented with checking the depth value of a single point in the middle of the camera’s view when positioned centered at the front of the robot. Additionally, we attempted to keep track of where the robot currently was based on its previous movements. However, this method caused the robot to collide with many objects that were not wide and solid such as chairs and tables, while also proving to be very difficult to debug based on the movement logic.

The programs commands for movement of the robot are focused around the various sensors on the Create 2 robot. We use two main types of movements in our autonomous functions, a drive straight, and a rotate function. The drive straight function uses the two wheel encoders, which the Create 2 stores as a two-byte count value. The drive straight function examines the encoder counts and sets the drive motors to run until the encoder count has incremented the desired amount. Due to weight and differences in the wheels of the robot, when driving straight the robot would tend to curve to one side. To fix this, when we check the encoder values, we also compare how fast the left and right encoders are reaching their goal value and will speed up the side that is further from its goal value and slow down the side that is closer to its goal value. This way the robot will correct for its tendency to one side. The rotate function utilizes the angle value the robot stores from its sensors. The rotate function sets one wheel to move forward and one to move backwards until the angle reading matches its desired value. These functions are used in all subsequent navigation algorithms.

Following the failed second attempt at automated robot navigation, we decided to reduce the complexity of our driving algorithm to that of a relatively simple wall following algorithm. We reoriented the Kinect so that it was facing slightly to the robot’s right side in order to detect obstacles to both the side and front of the robot. Additionally, instead of checking a single point in the center of the camera’s view, we began checking entire columns of depth values at both the right and left sides of the current depth image, providing us an idea of how close the robot was to obstacles in front and to the right of it. This method still ran into problems with chairs and tables but performed better than the previous iteration. We knew that we would still need to improve our navigation technique, however, if we were to achieve our goals.

We then decided to process the depth data further, to use all the data available at an instant in time to make a decision for the robot’s movement. We iterate over every pixel in the depth image
and use its horizontal position and its depth value to index into another 2D array and place a ‘1’ in the index indicating there is some object at that location. The pixel’s depth value is an integer that is the distance from an object in millimeters. This gives us a map of the objects that are in the Kinect’s view, which we refer to as an occupancy grid. An occupancy grid with its raw depth data is shown below in Figure 19. We then take the occupancy data out to a certain distance and take five portions of the data. In each portion the number of points are counted. From these five counts, we are able to locate an area in front of the robot that is unoccupied.

Figure 19. The raw depth data from the kinect is shown on the left. The occupancy grid made from the raw depth data is shown on the right. The grid shows the objects in the Kinect view as groups of points that are in the depth data as pixel values.

If we find an unoccupied region in front of the robot, we send a command to move forward 750 counts of the wheel encoders. If we find an unoccupied region that is not in the center we rotate a certain number of degrees based off of how far away the unoccupied space is from the center. From this point we repeat checking the depth data and moving for a pre-determined number of iterations.

3.2.3 Remote Control

Software was required to remotely connect an offboard computer to the onboard computer responsible for controlling the robot and interfacing with the Kinect camera. This allowed for the required software applications to be initiated, namely the Python script for controlling the robot navigation, the depth sensing Kinect program, and the Voxel Hashing application, without having to deal with the rather bothersome process of physically interacting with the onboard computer. Additionally, this provided team members and others the ability to view the commands the robot was executing and the images the Kinect was capturing in real-time.
The remote viewing software utilized throughout this project was TeamViewer 10, which is freely available for private use through the TeamViewer webpage. The TeamViewer application provided many benefits and no real drawbacks. As previously mentioned, TeamViewer is free which was beneficial for our budget constraints. Additionally, the application was very easy to install, setup, and was compatible with Windows, Mac OS X, and even mobile platforms. Finally, TeamViewer not only allowed for programs on the onboard computer to be initiated from a different computer, it also provided the team with the ability to make small edits to the Python driving script and immediately test them without having to remove and replace the onboard computer.

In order to begin the remote connection, TeamViewer must be installed on both the computer to be controlled and the device that will be used to remotely control it. Once installed, the application must be opened on both machines. The TeamViewer window on the onboard computer provided both an ID number and a password for allowing remote control. Connection between the machines was initiated by entering the onboard computer’s ID number into the remote computer’s TeamViewer window and hitting the Connect button. The remote computer was then prompted for the partner computer’s password before fully connecting.
4 Testing

Due to the nature of our project, the testing of the product is best done by placing the robot in an unknown location before starting our automated process. This simulates the situation the device might be in if it had just entered a room through a door. After a certain time, we can choose to stop the robot and build the 3D model. To demonstrate the capabilities of our robot, we will provide the results of three different test runs.

4.1 Test Run 1

Our first test was done in a hallway in the Engineering Building, as shown in Figure 20. We placed the robot next to a wall, and placed a few objects in front of the robot that it should capably avoid. When we started the robot, it successfully avoided the first box on the right and the trashcan on the left. After it maneuvered between these objects, we stopped the robot and built the 3D model, which is shown in Figure 21. This test demonstrated that the robot is capable of avoiding objects in navigating near a wall. The run lasted approximately one minute.

![Figure 20. Test Run 1 setup](image)
4.2 Test Run 2

Next, we placed the robot at a location in our design lab, shown in Figure 22. The robot navigated mostly around the table that can be seen on the right of Figure 22. The robot successfully navigated for approximately seven minutes around the table before encountering one of our problems. Infrequently, the robot reverts to its “vacuum” navigation mode, in which it starts to move in a circle. When this started, we stopped the robot and built the 3D model, which can be seen in Figure 23 and Figure 24.

This test again demonstrated that the robot could successfully avoid many objects, as it navigated for over seven minutes. However, the test does demonstrate that there is future work to be done.
In a third test run, we placed the robot in a location such that it would turn into a nook, seen in Figure 25. We wanted to test the ability of the robot to enter into an area with no exit, besides where the robot came from. The robot navigated successfully for about three minutes. However, once it reached a corner area, it turned repeatedly. After a while, we decided to stop the run and build the 3D model. In this case, the 3D model clearly shows the white board of the room, as shown in Figure 26. In this test, we again demonstrated the positive capabilities of the robot, while also finding an opportunity for future work.
4.4 Capabilities

Our device has numerous capabilities with which we think deem it a success. First, it can navigate unknown environments relatively well depending on the complexity of the room. Next, it is capable of operating for an extended period of time (more than 30 minutes on a full charge). The device produces fairly strong 3D models that integrate both texture and color, and the models go above the height of a table. Finally, the device is aesthetically pleasing, and the software is free and reproducible.

4.5 Shortcomings

Given the positive aspects of the project, we also have some problem areas that could be fixed for future work. Given enough time, the robot will most likely run into an object, as the collision
avoidance functionality is not yet perfect. Also, the robot sometimes gets stuck in front of a wall, or starts to spin in a circle. While we worked to address these problems, we could not fix them and but do believe they could be addressed with future work, which we will discuss in the next section.
5 Final Cost, Schedule, Summary, and Conclusions

5.1 Final Budget

A breakdown of our final expenses for this project can be found below in Table 5. The table is complete with a description, cost, and justification for each item. Unfortunately, the final budget was $529.03, which is approximately $29 greater than the $500 provided to us from our sponsor. The increase in budget is due to an increase of $30 spent on our structural supplies, and an increase of $17 spent on our power supplies.

We would like to note two things regarding the budget. First, our project is inherently expensive; we spent over $400 on two irreplaceable components (the iRobot and the Kinect camera), leaving us with less than $100 for the power supply and structure. With the remaining money, we were able to complete the power supply. At first, we were using a small crate to serve as a structure, and this cost us no money. However, we wanted to make our prototype more aesthetically pleasing, and we purchased with our own money supplies to build the final structure. Therefore, we did not spend all $500 given to us by the ECE department, and instead used our own money to simply improve the aesthetics, but not the function, of the design.

<table>
<thead>
<tr>
<th>Number</th>
<th>Item</th>
<th>Cost ($)</th>
<th>Justification</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>iRobot Create 2 Programmable Robot</td>
<td>211.99</td>
<td>The robot will be our navigation device.</td>
</tr>
<tr>
<td>2</td>
<td>Microsoft Kinect Camera for Xbox v2</td>
<td>149.99</td>
<td>The Kinect camera will acquire rgb-d images.</td>
</tr>
<tr>
<td>3</td>
<td>Kinect to Windows Adapter</td>
<td>49.99</td>
<td>The adapter will connect the Kinect camera to our laptop for processing.</td>
</tr>
<tr>
<td>4</td>
<td>Power Supplies</td>
<td>67.06</td>
<td>We will need to have a custom power supply for the kinect</td>
</tr>
<tr>
<td>5</td>
<td>Structural</td>
<td>50.00</td>
<td>To hold the laptop and kinect to the create 2</td>
</tr>
<tr>
<td>Total</td>
<td>-</td>
<td>529.03</td>
<td>The increase in budget was due to costs associated with aesthetic upgrades to the structure.</td>
</tr>
</tbody>
</table>

Table 5. Final Budget
5.2 Final Budget (projected cost per unit for multiple units)

Discounts are usually offered from companies when products are bought in bulk. If this project was scaled to accommodate multiple units, the price of each individual unit would be reduced. Although we do not have exact price discounts offered by these various companies, we can still make an educated guess. If an average discount of 10% is assumed for each individual component, then the final total for one complete unit would be $476.13. A breakdown showing the final cost of the individual sections can be found in table 6 below.

<table>
<thead>
<tr>
<th>Number</th>
<th>Item</th>
<th>Cost ($)</th>
<th>Justification</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>iRobot Create 2 Programmable Robot</td>
<td>190.79</td>
<td>The robot will be our navigation device.</td>
</tr>
<tr>
<td>2</td>
<td>Microsoft Kinect Camera for Xbox v2</td>
<td>134.99</td>
<td>The Kinect camera will acquire rgb-d images.</td>
</tr>
<tr>
<td>3</td>
<td>Kinect to Windows Adapter</td>
<td>44.99</td>
<td>The adapter will connect the Kinect camera to our laptop for processing.</td>
</tr>
<tr>
<td>4</td>
<td>Power Supplies</td>
<td>60.35</td>
<td>We will need to have a custom power supply for the Kinect camera.</td>
</tr>
<tr>
<td>5</td>
<td>Structural</td>
<td>45.00</td>
<td>To hold the laptop and Kinect to the Create 2</td>
</tr>
<tr>
<td>Total</td>
<td>-</td>
<td>476.13</td>
<td>The increase in budget was due to costs associated with aesthetic upgrades to the structure.</td>
</tr>
</tbody>
</table>

Table 6. Final budget per unit assuming a 10% discount

5.3 Final Schedule

The final Gantt chart is displayed in Figure 3 and Figure 4 of Appendix 3. We summarize our schedule as follows. Before spring break (3/6/15 - 3/10/15), we acquired the main hardware components and downloaded the required software. In the next few weeks, we created the power supply for the Kinect camera, tested the 3D modeling software, developed the depth data capture
application, and started working on the room navigation algorithm. However, during this time we were still using the ECE department laptop. On 4/3/15, we tested the new laptop, and achieved success with the 3D modeling software. Once this was completed, we continued to work on the navigation script and built the structure. We demonstrated our final prototype on 4/22/15 and made a couple of modifications to the structure in the days afterwards.

5.4 Future Work

Given more time, we have identified multiple areas that could be improved in our device, which we think would serve as great future ECE 480 projects. First, the robot indeed collides with objects frequently, after a large number of decisions. An improved method to avoid collisions would be greatly beneficial. One potential solution would be to incorporate multiple cameras. This would give us more and better data to use in our navigation algorithm.

Next, we would like to create a better solution for detecting if the room has been completely covered. Currently, we only run the navigation algorithm for a certain number of decision steps. However, we would like to develop functionality such that the robot continues to navigate the room until it has completely covered the room and/or returned to its original spot.

Finally, it would be interesting to investigate controlling the robot remotely. The iRobot can be controlled by keyboard inputs using a python script, and it would be interesting to combine “automatic” and “manual” drive modes.

5.5 Summary / Conclusion

While we do acknowledge that our device is not perfect, and have identified areas in which it could be improved upon if we had more time or if someone else took up this project, we ultimately believe that we have been successful in achieving our goals. In this project, we have made great strides towards accomplishing the initial goal: to create a fully autonomous system capable of creating a 3D model of an unknown environment. We successfully combined an iRobot Create 2 Programmable Robot, a Kinect for Windows v2 camera, and a laptop into one functional machine that is able to detect most objects in its path and build 3D models of unknown environments. Through discussions with our sponsor, we are sure that we have lived up to his expectations, while laying a solid foundation for future improvement.
Appendix 1: Technical Roles, Responsibilities, and Work Accomplished

Jacob Kneibel

In the Proposal, Jacob’s technical role was to focus on the mapping and localization of the robot, which included a movement algorithm that the robot would follow throughout the room. However, he also worked on the 3D model building software, and the structure of the robot.

Jacob first worked on installing and building the two different 3D model building software packages for the Kinect 2, the Microsoft Kinect Fusion, and the Voxel Hashing. He tested the limitations of the Kinect Fusion code and then worked on setting up the environment to run the Voxel Hashing code once the team decided to use that over Kinect Fusion.

Jacob tried to get the Voxel hashing code to run the original laptop we were using. This included downloading and installing the correct libraries that the Voxel Hashing code called for. After that the code was built and upon testing the program would not run properly. He engaged in discussions with Matthias Niessner to find a solution. We discovered the problem was with the hardware of the laptop, and once Kexiang offered his laptop to be used, Jacob helped install Voxel Hashing there.

He next worked on depth data processing and with Nick Saxton’s movement code he helped create a new navigation script. The data processing that he did, converted the raw depth data into an occupancy grid of the space in front of the robot and found areas of the grid that the robot could move to. He and the team worked to send the commands to the robot to move to those locations.

After taking ideas and suggestions for the structure, he found the parts and assembled the structure and modified the Create 2 to allow the structure to be attached.

He also helped in the testing phase of the robot and helped make changes after the testing found issues with the current implementation.
Kexiang Qian

In the proposal, Kexiang’s role was to build a mobile power module for the Kinect sensor, a component that was central to the project. He verified that the component was functioning correctly and working properly through the conduction of various tests and remedial measures.

Kexiang did research on a variety of batteries, including the aspects of the energy capacity per kilogram, stability and safety consideration like burning. In the end, he chose a NiMH battery and connected the battery to a DC Regulated Power Supply Voltage Converter booster.

The mobile power was able to support the team using the Kinect camera without attaching it to a wall outlet. During all the testing, the performance of the battery was always stable and never overheated. The fully charged battery was enough support for the Kinect to continue running until the iRobot ran out of power.

After building the power module, the team was encountered with a problem on running the Voxel Hashing code on the department’s standard laptop for each team, due to the poorer graphics card. Kexiang checked his personal MacBook Pro, and found that it reached the requirements for the code. After installing the Windows 8.1 Pro required by the Kinect V2, Kexiang contributed his laptop to the group to run the all the programming for the group. The laptop’s hardware fully supported all the testing on the robot and was able to build the 3D model smoothly without any crashes caused by hardware.

In addition, Kexiang attended almost all of the robot testing based on his flexible semester schedule. During every test, he gave suggestions on how to fix the variety of problems that were encountered during testing.
Nick Saxton

In the initial stages of the project and as presented in the proposal, Nick’s technical role was to write software for data transmission between the onboard computer and an offboard machine due to the suspected computational intensity of the required software. However, upon discovering that a team member had a computer capable of handling the computation onboard, he began work on communicating with the robot through its serial port.

Nick researched serial communication, specifically with the iRobot Create 2, and attempted to implement basic robot control, initially with a C++ application before settling on a Python script that better accomplished the task. He first developed a tethered, manual control application by utilizing code provided on the iRobot Create 2 webpage. This code was used during the first prototype demonstration. Following this, he began work on automated control code.

Nick developed the initial version of the automated robot control Python script. He utilized the pySerial library and depth data from the Kinect to rudimentarily avoid running into objects. Testing proved this version of the code to be far too unreliable for real-world use however. At this point in the project, Nick began working with Jacob to develop a more reliable robot navigation script, which would eventually be used in the final version of the product.

Throughout the length of the project, Nick was actively involved in testing the robot’s performance. He worked alongside other team members to develop specific test situations and helped ensure that they were executed properly. He not only suggested and implemented refinements to the robot control code but also offered suggestions to the group members working with the Kinect software. Additionally, through testing, Nick was able to provide input on the design of the structural support for the Kinect camera and computer along with ideas for the best placement and orientation for the camera.
David Zoltowski

In the proposal, David’s formal role was to implement methods for the device to avoid collisions with objects. However, he was also involved in developing a prototype structure and in testing.

David wrote an application that acquires depth image data from the Kinect, displays the data, and saves the data to a file multiple times per second. The application was developed in Windows Visual Studio in C#. He wrote the code to capture the depth data and save it, and utilized a previously written function to transform it into a displayable, informational format.

With this application, the team was able to develop a Python script that controlled the robot. The script read the depth file at each decision step and incorporated the file into its algorithm to determine where to move. The frequency in which the depth image is saved to the file can be controlled in the application to accommodate the needs of the Python script.

On top of this, David was involved in testing the device throughout the design process. When the robot, camera, and laptop were ready for testing, David contributed a small crate to act as a temporary structure. The crate was placed on top of the iRobot, the laptop was put into the crate, and the Kinect camera was placed on top of the crate. While crude, the temporary structure allowed the team to further develop the navigation script until the final structure was ready.

Additionally, David was involved in the testing process by developing situations to test the prototype and by contributing to discussions. He helped identify situations in which the prototype was failing, and contributed suggestions to address those problems. Finally, David was involved in preparing and analyzing the 3D modeling software, where he helped optimize the software parameters for the team’s use.
Nick Zuzow

In the proposal, Nick’s technical role was defined as working on the 3D model reconstruction section of this project. Along with this, Nick also contributed to the robot navigation, and the testing of the system.

Initially Nick spent time researching the different types of open source 3D modeling solutions that were currently available. This research mainly focused on Kinect Fusion and 3D Reconstruction at Scale Using Voxel Hashing since the groups sponsor, Dr. Daniel Morris, suggested them. However, other possible solutions were investigated as well.

After discussion between team members, the Voxel Hashing code was determined to be the best solution for 3D reconstruction. Problems occurred after the initial installation and setup of the Voxel Hashing code. Nick was involved in troubleshooting these problems, and after it was determined that the issues were due to hardware, Nick helped with the setup of the software on the second computer.

Nick also helped with the implementation of the robot navigation. When problems were encountered with the reliability of the initial navigation script, Nick started investigating the possible use of the robots sensors for navigation. This included altering the python script to utilize the wall sensor of the Create 2. Unfortunately, after basic testing it was determined that the IR sensor of the robot detected a wall at a maximum distance of about 5 inches. This was not suitable based on the needs of the project. Although Nick did not write the code for the navigation, he was involved in many discussions with other team members about how the robot should handle certain situations.

Finally, Nick was involved with testing the system throughout the semester. He worked with other members of the team to test the systems’ response and performance in a multitude of situations. Throughout the testing of the navigation code, Nick kept his original role in mind by making suggestions on the speed, driving distance, and rotation angle to help produce a better overall model.
Appendix 2: Literature and Website References


Appendix 3: Detailed Technical Attachments

Initial Gantt Chart

<table>
<thead>
<tr>
<th>Task Name</th>
<th>Start Date</th>
<th>Finish Date</th>
<th>Duration</th>
<th>Resource</th>
<th>% Complete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial Gantt Chart</td>
<td>1/1/2015</td>
<td>12/31/2015</td>
<td>12 months</td>
<td>All</td>
<td>100%</td>
</tr>
</tbody>
</table>

Figure 1. Initial Gantt Chart, Page 1
Figure 2. Initial Gantt Chart, Page 2
Final Gantt Chart

<table>
<thead>
<tr>
<th>ID</th>
<th>Task Name</th>
<th>Start/End Date</th>
<th>Percent Complete</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Research Design</td>
<td>3/2/12 - 3/12/12</td>
<td>100%</td>
</tr>
<tr>
<td>2</td>
<td>Review of paper 1</td>
<td>3/2/12 - 3/9/12</td>
<td>100%</td>
</tr>
<tr>
<td>3</td>
<td>Review of paper 2</td>
<td>3/3/12 - 3/9/12</td>
<td>100%</td>
</tr>
<tr>
<td>4</td>
<td>Review of data 3</td>
<td>3/6/12 - 3/9/12</td>
<td>100%</td>
</tr>
<tr>
<td>5</td>
<td>Summary Report 3</td>
<td>3/9/12 - 3/12/12</td>
<td>100%</td>
</tr>
<tr>
<td>6</td>
<td>1st Prototype Dev</td>
<td>3/9/12 - 3/15/12</td>
<td>100%</td>
</tr>
<tr>
<td>7</td>
<td>Implement 1st Dev</td>
<td>3/10/12 - 3/16/12</td>
<td>100%</td>
</tr>
<tr>
<td>8</td>
<td>Order Parts</td>
<td>3/10/12 - 3/16/12</td>
<td>100%</td>
</tr>
<tr>
<td>9</td>
<td>Kit final</td>
<td>3/13/12 - 3/16/12</td>
<td>100%</td>
</tr>
<tr>
<td>10</td>
<td>Prototype Test 1</td>
<td>3/15/12 - 3/15/12</td>
<td>100%</td>
</tr>
<tr>
<td>11</td>
<td>Final Proposal 2</td>
<td>3/18/12 - 3/25/12</td>
<td>100%</td>
</tr>
<tr>
<td>12</td>
<td>1st Prototype Dev</td>
<td>3/18/12 - 3/25/12</td>
<td>100%</td>
</tr>
<tr>
<td>13</td>
<td>Final Prototype Development</td>
<td>3/28/12 - 4/4/12</td>
<td>100%</td>
</tr>
<tr>
<td>14</td>
<td>Build prototype</td>
<td>4/4/12 - 4/7/12</td>
<td>100%</td>
</tr>
</tbody>
</table>

Figure 3. Final Gantt Chart, Page 1
**Figure 4.** Final Gantt Chart, Page 2
Depth Acquisition Application Source Code

MainWindow.xaml

<Window x:Class="KinectTutorial.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="MainWindow" Height="350" Width="525">
    <Grid>
        <Image Name="camera" />
    </Grid>
</Window>

MainWindow.xaml.cs

using Microsoft.Kinect;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Shapes;

namespace KinectTutorial
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            this.Loaded += OnLoaded;
        }

        // initialize Kinect sensor and depth frame reader
        KinectSensor sensor;
        DepthFrameReader reader;
        int counter = 0;
    }
}
// act on depth frame when a depth frame arrives
void OnLoaded(object sender, RoutedEventArgs e)
{
    this.sensor = KinectSensor.GetDefault();
    this.sensor.Open();

    this.reader = this.sensor.DepthFrameSource.OpenReader();
    this.reader.FrameArrived += OnFrameArrived;
}

// if the frame is not empty, process the frame once every half a second (15/30)
void OnFrameArrived(object sender, DepthFrameArrivedEventArgs e)
{
    using (var frame = e.FrameReference.AcquireFrame())
    {
        if (frame != null)
        {
            if (counter % 15 == 0)
                camera.Source = ProcessFrame(frame);
            counter++;
        }
    }
}

// convert depth data into image data
private ImageSource ProcessFrame(DepthFrame frame)
{
    //string depthData = "";

    int width = frame.FrameDescription.Width;
    int height = frame.FrameDescription.Height;
    PixelFormat format = PixelFormats.Bgr32;

    ushort minDepth = frame.DepthMinReliableDistance;
    ushort maxDepth = frame.DepthMaxReliableDistance;

    ushort[] pixelData = new ushort[width * height];
    byte[] pixels = new byte[width * height * (format.BitsPerPixel + 7) / 8];

    frame.CopyFrameDataToArray(pixelData);

    System.IO.File.WriteAllText(@"C:\Users\72920\Desktop\Project\Robot\TEST.txt", String.Join(" ", pixelData.Select(p => p.ToString()).ToArray()));
int colorIndex = 0;
for (int depthIndex = 0; depthIndex < pixelData.Length; ++depthIndex) {
    ushort depth = pixelData[depthIndex];

    byte intensity = (byte)(depth >= minDepth && depth <= maxDepth ? depth : 0);

    pixels[colorIndex++] = intensity; // Blue
    pixels[colorIndex++] = intensity; // Green
    pixels[colorIndex++] = intensity; // Red

    ++colorIndex;
}

int stride = width * format.BitsPerPixel / 8;

return BitmapSource.Create(width, height, 96, 96, format, null, pixels, stride);
}