Final Report

Version 1.0

April 30, 2022

Team Poseidon Way-Finding

Sponsor: Michael Leverington

Faculty Mentor: Han Peng

Team:

Fernando Diaz

Ulugbek Abdullayev

Brandon Jester

Jonathan Gomez

1.0 Introduction

Robotics has been an area of interest for almost as long as computer science has been. The concept that machines could be given a task and complete it more efficiently than a human could, has been even more prevalent in the past few decades than ever. This trend will only continue as technology and automation become more pervasive in our everyday lives. Because of this, robotics has been a continuously growing sector of computer science and will remain integral for decades to come. In the past, the main inhibiting factor to robotics was its power, cost, and complexity. Electronic components needed for robots have become significantly cheaper while simultaneously becoming more powerful. Because of the costs, in the past, there has been a severe lack of learning opportunities for students to use a physical robot until now. Robotics in classrooms has been too expensive to create and use. However, it has become feasible to create fully autonomous robots that remain inexpensive.

The client, Dr. Michael Leverington, is a lecturer of computer science at Northern Arizona University (NAU), and his goal has been to forge the minds of future computer scientists. His business has involved teaching students how to solve otherwise complex problems. His motto relies on his ability to forge young minds to wield the powers of technology, mainly computer programming. Dr. Leverington is interested in robotics and has seen this decrease in cost and lack of educational opportunity and came up with a solution to it. His answer is to develop a flexible, cost-effective robotics platform in college-level programs for educational purposes. The overall goal of the project is to eventually have a robot that can give tours of NAU's engineering building to potentially bring in more students to engineering programs.

To accomplish that robotics platform, Dr. Leverington made the thirty-gallon robot, initially known as the robot-assisted tours project or RAT. The thirty-gallon refers to the tank which encases the robot's components. The thirty-gallon barrel uses a wooden dolly as the base and has access to components such as two motors and a Raspberry Pi. The components in total cost approximately $1000. In order to give tours of the building, it needs to be autonomous and support programmability. A prospecting student that is touring NAU might be impressed by the autonomous tour they are being given that was developed by students. This would hopefully convince them to pursue a career in computer science or engineering at NAU.

2.0 Process Overview

For the development of this project, our group split up into roles that would allow us to work concurrently on different parts of the project. Fernando Diaz was the team leader and Customer Communicator, Ulugbek Abdullayev was the team recorder, Jonathan Gomez was the main coder for the project, and Brandon Jester was the release manager.

During the development process, the team had weekly meetings with mentor Han Peng to ensure the development was on track. Meetings began with a progress report on what was developed or finished during the time since the last meeting. After that, the team looked at what was upcoming and needed to be worked on or completed for the week. Next, the team discussed questions or concerns about the project with the mentor. Finally, the struggles that the team was encountering were discussed so that a resolution may be found/ During team meetings all decisions were made with a ⅔ majority vote with the team lead being the final say in the case of a deadlock. During the meetings all discussions were kept to the topic of the project so that the team could stay on-track and finish the project in a timely manner.

During the development of the project, the team used AzureDevOps to keep track of versions and updates made to the code. Whenever a change was to be made, the team would review a pull request before the code was allowed to be merged into the main codebase. Issues with the project were also tracked through DevOps so that they could be monitored and resolved in the same place updates were handled.

For deliverables and papers that were required the team used the Google Drive suit of products for writing and editing. This was chosen as all NAU students are given a google account which shares a drive.

3.0 Requirements

Now with the development process discussed, we had to gather our requirements and specify what the client wanted from the thirty gallon robot. To gather these requirements, we met weekly with the client over zoom meeting. We asked questions about the kind of features and specific attributes he wanted for the robot. From the discussions with the client, three domain-level requirements were outlined:

- **DR1. Autonomous Movement:** The client specified that the robot will require an autonomous movement module for the first high domain-level requirement. This means that the robot will be able to move through some program without any human input. This movement could be a specific path or a dynamically generated path.

- **DR2. Obstacle Avoidance:** The second domain-level requirement is for the robot to have an obstacle avoidance system in place. If the robot encounters a wall or obstacle, it will reroute itself to move around or away from the obstacle. It will then continue down its current path. The client required that the robot use a sensor to detect obstacles and avoid them autonomously.

- **DR3. Returning To Start:** For the final domain-level requirement, the robot will be able to recognize the end of its path, turn around and return to its starting position. While not a specific requirement, the client expressed that in order to achieve this, the solution should be contained to the robot itself. This means that an object placed by the team to detect the end of the path would not be used.

To fully understand the functional requirements, each requirement is broken down into smaller low level requirements Each of the following high-level requirements will be broken down into these low level requirements to demonstrate a complete understanding. These requirements also have specific performance requirements to which these functional requirements' performance will be measured and quantified. There are also some environmental requirements regarding what hardware will be used that the client requires.

3.1 Functional Requirements

Each Domain Level Requirement (DR#) will be followed by ones of its lower level functional requirement (FR#) and the number of the functional requirement. The requirement will be explained, and if it has any sub requirements they will be noted by bullet point.

DR1-FR1. The computer for the robot must be able to send movement commands to the motor drivers housed on the robot.

- Program held on the raspberry pi will send commands using python to the motor drivers through its GPIO pins.
- These commands must also be able to be sent independently of each motor. The robot must be able to move one motor faster and in opposite directions than the other in order to achieve turns and rotations.

DR1-FR2. The robot must be able to follow a pre-programmed or pre-planned path.

- Programs could be written and executed on the robot, and the robot will perform the given path.

DR1-FR3. The robot's movement must not be controlled by any human input.

- A program could be executed on the robot, and no human can directly impact the robot's movement. Does not include indirect impact such as obstacle avoidance.

DR2-FR1. The obstacle avoidance system will be able to detect obstacles

- The robot will use a sensor to gather information and determine if there is an obstacle in front of it.

DR2-FR2. The robot must be able to reroute around obstacles or away from them.

- Upon detecting an obstacle in the path movement commands will be sent to move the robot around until obstacle is no longer in view
- Upon detecting a wall, movement commands will be sent to keep the robot in the center of the path
- Robot must continue on path after rerouting.

DR2-FR3. The robot will wait for wait for obstacles to pass if they are moving

- Upon detecting obstacle and slowing down, the robot will wait briefly to see if the object has moved, and if so will wait for it to pass

DR2-FR4. Robot will not hit or bump into obstacles

- Robot will stop before it hits the obstacle once detected

DR3-FR1 Robot can recognize and identify the end of the path

- Sensor on robot will take in information, and will interpret through the robot's computer whether the end of the path has been reached

DR3-FR2 Robot will return back to its original starting position

- Robot will perform a 180 degree turn at the end of the path
- Robot will move down its path again going the opposite direction

DR3-FR3 Robot can recognize the original starting position

- Sensor will detect the end of path or original start position
- Robot will stop upon detection.

3.2 Performance Requirements

DR1-PR1. The robot must maintain a speed of ~0.5 meters per second

- Will slow down for obstacle avoidance

DR2-PR1. Obstacles must be detected from a distance between one to three meters

DR2-PR2. Robot must be able to detect obstacles that are ½ meters tall and above

DR2-PR3. Robot will stop around one meter to one foot away from the obstacle it detects

- Needs enough space in order to clear the obstacle or move around it.

DR2-PR4. Robot will be able to move around obstacles within 20 seconds after initial detection

DR3-PR1. Robot will detect the end of the path or its original starting point within five seconds of reaching the designated end point.

3.3 Environmental Requirements

Along with requirements on how the robot needs to function, the Client provided a list of intrinsic requirements that must be followed during the development of the project. The client has set a set of hardware requirements that must be followed. There are three major requirements identified for hardware:

- Thirty Gallon Robot Housing
- Raspberry Pi
- Budget Limit

Thirty Gallon Robot Housing

The Thirty Gallon Robot was designed and built by previous Electrical Engineering capstone students and is intended to be the robot used in the final version of the thirty gallon robot project. The robot that was provided includes the thirty-gallon barrel as the housing, wooden dolly for the base, and significant hardware components such as the raspberry pi, two motors, and two motor drivers. Any additions in terms of modules or hardware components must be contained inside the provided thirty-gallon robot. Any electrical components will also need to be powered off of the batteries onboard the robot.

Raspberry Pi

A Raspberry Pi was requested to be used by the client Dr. Leverington, specifically the Raspberry Pi model 4B, with 8 gigabytes of ram. The Raspberry Pi is affordable and is a device that will not break the budget. Raspberry Pi's start at 35 dollars before tax, kits range at 170 dollars; it is not an extremely expensive machine. In addition, it allows for multiple devices to be connected to it, from keyboards to monitors. Also, the 40-pin GPIO header enables the Raspberry Pi to be connected to the robot's motors and control them. This Raspberry Pi will serve as the computer of the thirty gallon robot. It is where the codebase will be held and where most of the computation for movement and obstacle avoidance will be computed. Functionality can be extended and made modular

with other microcontrollers and microcomputers, but the Raspberry Pi will remain the main computer for the robot.

Budget Limit

The total cost of the robot will be cheap enough that other organizations such as colleges would be able to purchase and build this same robot. Therefore, a budget limit has been set for the new parts that can be purchased for the movement and obstacle avoidance modules. The budget limit has been set at $300.

4.0 Architecture and Implementation.

With requirements established, this section will discuss our architecture and implementation. It is what we created in order to meet these explicit requirements. The implementation of the system will require all the discussed hardware and software components to interact with each other. In this section, the major modules and their detailed implementation will be discussed.

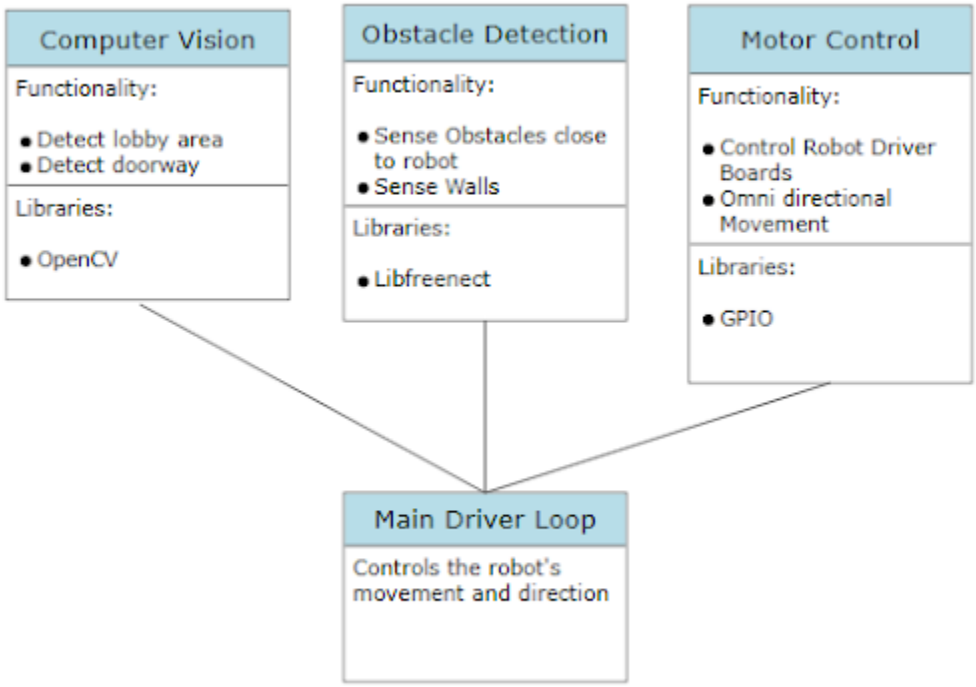

Figure 3.1: Diagram of major components and relation

Figure 3.1 shows our three main modules and their interaction with the main software driver loop. Our first module is the computer vision module. This uses the OpenCV library for python for image processing. It is responsible for processing the depth image from the kinect, and for detecting each end of the hallway using the Raspberry Pi Camera. The next module is our obstacle avoidance module. This module is responsible for getting depth camera images from the kinect sensor, and getting distance information from the sonar sensors. It uses this information to determine if there is an obstacle in the way. The last module is the motor control module. This is responsible for both the hardware and software connection between the Raspberry Pi and the motor driver boards. Using Python, we can send signals via the hardware

connection to the motor drivers to control the speed and direction of each motor independently.
This section details the implementation of each module and the main driver loop of the program.

4.1 Control Module

The main goal of this project is to create a robot that is fully autonomous. As such the basis of the system as a whole must be a component that allows the robot to move without input from an outside source (i.e. a human). This component will be the main driver of the system and provide a basis for all other systems. More specifically, this component will allow the robot to move in a straight line without any user interface, input devices, or other peripherals. In short, when the robot is turned on it should be able to move on its own in a single direction. To achieve this goal, the software must be able to control the onboard hardware for turning on and off the motors. As it turns out, the drivers that the computer has access to are able to control the motors in a finite manner, meaning that the motors are able to be powered with varying amounts of electricity, allowing for finite control over the speed and direction of the robot. With these drivers and a python library called RPi.GPIO which allows the pi to control the pins on the drivers, a module will be developed that can accomplish the required task. With the aforementioned pins on the driver boards, the program is able to control how much power goes to each motor, which will be used to create movement sub-functions. Below is a diagram of the sub-functions needed by the automated driving system.

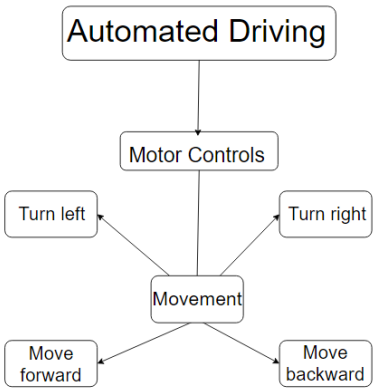

Figure 4.1.1 Control System Flow

As basic as this module may be, it is the foundation for the operation of the robot. The Automated Driving component is made up of four sub-functions that control the direction the robot is able to move in. By powering one motor more than the other, the program is able to turn the robot left or right 90 degrees at a time, and by powering both motors

equally, the program will be able to move the robot forwards and backward. The reason we do not include any other sub-functions in addition to the four seen here is that they would either be redundant with other modules, or extraneous for this module.

```
def move_forward(power):
    GPIO.output(motor_a_dir, a_direction_fwd)
    GPIO.output(motor_b_dir, b_direction_fwd)
    pwm_motor_a.ChangeDutyCycle(power)
    pwm_motor_b.ChangeDutyCycle(power)
```

Figure 4.1.2 Implementation Of Basic Movement Command

Automated Driving will be the basis for all other functions of this system. This module is intended to give access to the direction functions which will allow other systems to control what direction the robot will move in. The general implementation of one of the movement commands is shown in Figure 4.1.2. From this figure, it is very simple to call the *move_forward()* function with a power amount between 0-100. Then It sets the direction pins on the motor drivers to the forward direction, and each motor's power level to the passed in power amount to control speed. A similar implementation is held for all other directions as well. A functional example of the system using these movement control commands would be when an object is detected in front of the robot, the obstacle avoidance system will decide which side of the obstacle is clear, turn the robot in the determined direction with either the turn right or turn left function, then use the move forward function to move a safe distance past the object, and finally repeat this process to return to the forward position. Another example would be the computer vision module detecting the end of the robot's path, which would then call upon this module to turn the robot around in a similar fashion to the obstacle avoidance module and begin another straight path. Without this module, all other systems of the robot would not be able to function.

4.2 Computer Vision

The computer vision module is necessary to complete the end of path detection. In this project, it is required that the robot is able to navigate down the second floor of the long hallway of the NAU Engineering building. The robot must be able to start and go from the south end down to the north end, turn around, come back to the south end, and end the program. In order to find each end of the path, computer vision will be used.

Computer vision is a field of artificial intelligence that allows computers to infer the meaning of certain items, or what they are, such as chairs and windows. In fact, computer vision attempts to replicate human vision, in the sense that it tries to recognize objects and patterns in order to categorize them. Computer vision requires a lot of data to analyze, in order to compare the similarities of certain objects, such as different types of tires. By comparing a vast amount of references, it can form a more accurate perspective on the differences and similarities between certain objects. Also, algorithms must be used in order to allow the machine to learn recognition and discern different objects from one another.

Figure 4.2.1 North End of Hallway in NAU Engineering building (Lobby Area)

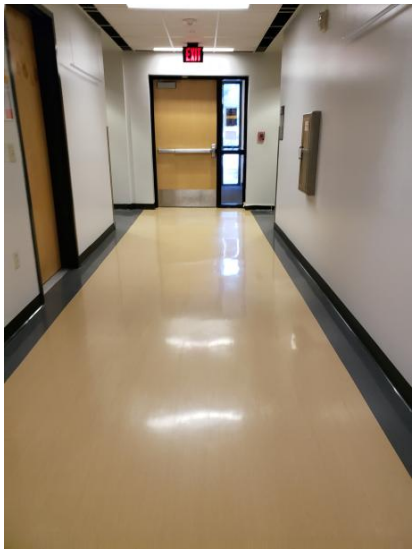

Figure 4.2.2 South End of Hallway in NAU Engineering building

For the end of path detection, computer vision will be used to find the patterns and recognize the end of each hallway. From Figure 4.2.1, it shows the north end of the hallway, which is a lobby area, and Figure 4.2.2 shows the south end of the hallway. Both need to be recognized in order to turn around or stop. In Figure 4.2.2, it can be seen that the light-colored tiles line the entire hallway up until the lobby area from Figure 4.2.1. The approach taken is that for the lobby area detection, computer vision will be utilized to recognize the color difference between the darker carpet area of the lobby and the lighter tile area of the hallway. Using OpenCV, the differences in color can be discerned quickly by comparing the pixels and area of the ground that the Kinect sensor will see. If we reach a confidence level or an area of the ground that has become dark enough, then the robot will stop and turn around. For the full termination of the path on the south side in Figure 4.2.2, is even more simple. As will be detailed more in 4.3, the depth image from the Kinect can be acquired. As the robot approaches the south side, the depth camera will begin to pick up that the full wall including the door is in range. The algorithm will determine whether it sees a full wall within the distance that it is too close, and if so, will stop and end the program.

4.3 Obstacle Avoidance

The most complex component is the obstacle avoidance system. The responsibility of this component is to ensure that the robot can get to the end of its path safely by avoiding any obstacles or objects in the way. It does this by using the camera data received from the Kinect sensor, processing the data through an algorithm to determine if there are obstacles in the way, and if so, how to move around them. These determinations by the algorithm will result in the autonomous movement commands sent to the motor drivers.

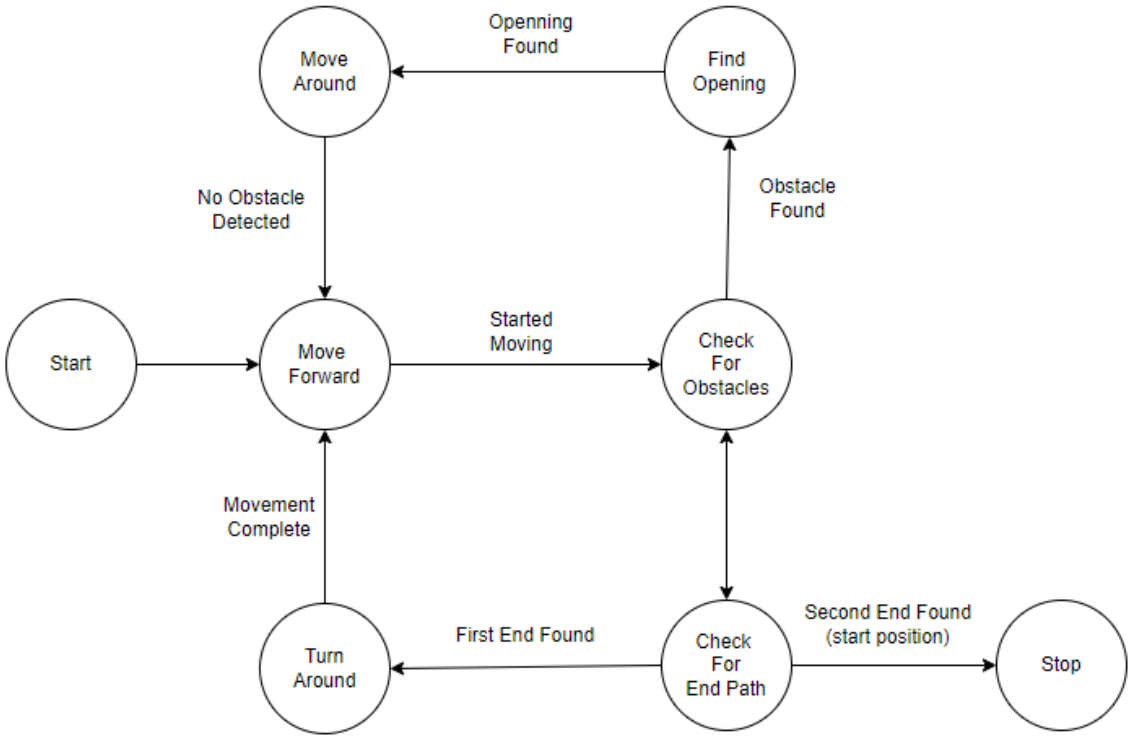

Figure 4.3.1 Simplified State Diagram of Obstacle Avoidance Algorithm

Figure 4.3.1 is a simplified version of the process or flow of the obstacle avoidance algorithm. It is represented as a state diagram where each circle in the diagram represents the calculation or computation being performed, and the arrows represent how the computations flow to each other with a text explanation of that event that occurs. The robot will start by setting the motor drivers to move forward with a given speed. After this it will then calculate whether there is an obstacle in front of the robot using the depth image received from the kinect sensor.

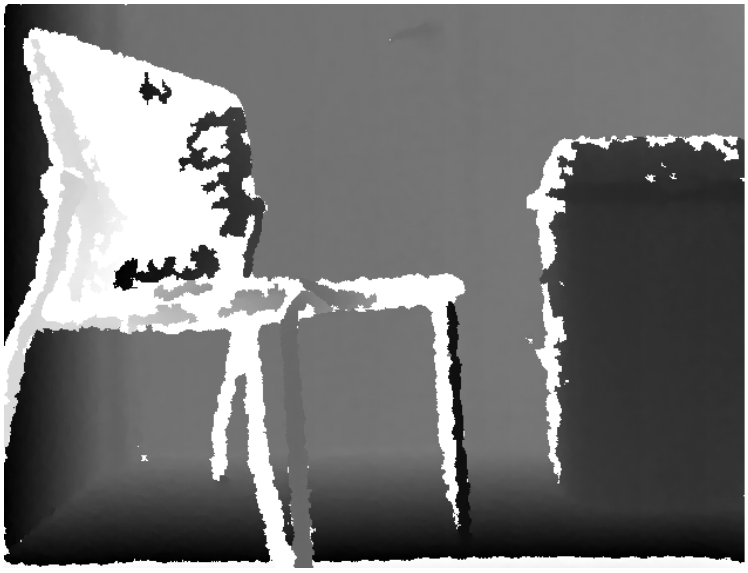

Figure 4.3.2 Depth Image

Figure 4.3.3 Depth Image With Threshold Applied

From Figure 4.3.2, the depth sensor gives us a grayscale image where the darker the color, the further away from the sensor it is. From Figure 4.3.3, this image is converted into a pure black and white image using a depth range threshold value. The black pixels indicate that there is an obstacle detected within a given depth range, and any white pixels are obstacles outside this range. From here, the grouping of black pixels indicates that there is a full object in view. To determine if the object is in the way, the area of the grouped black pixels is calculated using contours, and if large enough will trigger that an obstacle was found and stop movement.

```
def find_contours(frame, depth):
    depth_gray = depth.copy()
    contour_img = frame.copy()
    contours, hierarchy = cv2.findContours(depth_gray,
                                           cv2.RETR_EXTERNAL,
                                           cv2.CHAIN_APPROX_NONE)
    cv2.drawContours(contour_img, contours, -1, (255, 0, 0), 3)
    cv2.imshow('Contours', contour_img)
    return contours
```

Figure 4.3.4 Code Implementation of Finding Contours From Depth Image

```
contours = find_contours(frame, depth)
for cnt in contours:
    area = cv2.contourArea(cnt)
    if area > OBSTACLE_AREA:
        print("Obstacle Detected")
        stop()
        state = 2
```

Figure 4.3.5 Code Implementation of Using Area of Contours

Figure 4.3.4 shows the implementation of how the contours and area of the depth image are acquired. The function *find_contours()* is called with the color and depth image. Using the OpenCV function *findContours()* returns the contours which hold an area value. This value can be easily accessed as shown in Figure 4.3.5 and compared to see if the value is large enough to determine if there is an obstacle.

The algorithm then calculates to find an opening by moving the robot and the camera and using the same black and white pixel technique. Upon finding an opening, the robot will move through it and go back to detecting obstacles once the path is clear.

After checking for obstacles, if none are found, the robot will then begin to calculate whether the end of the path has been reached. As mentioned in the computer vision section, it is required that we determine the end of the path in order to turn around and stop. This is where the computer vision strategies that were outlined are used. When the algorithm finds a positive for the first end of the path, its robot will then be given commands to stop, turn around, move forwards, and begin the process over again.

Now, the operations determine whether the original starting position has been reached. If so, the robot will stop and the program will end.

```
while (True):
    frame = get_camera_image()
    depth = get_camera_depth()

    if state == 0:
        move_forward(60)
        state = 1
    elif state == 1:
        check_for_obstacle(frame, depth)
        check_for_end_path(frame, depth)
    elif state == 2:
        rotate_find_openning(frame, depth)
```

Figure 4.3.6 Code implementation of State Diagram

Figure 4.3.6 outlines the general flow in the code implementation of the state diagram. The program happens within a while loop in which the frame and depth images are received from the respective functions. Then, a series of if else statements determine the action that needs to be taken based on the current state of the program. Some of the other states are contained within other functions like turning the robot around when finding the path occurs within the function *check_for_end_path().* The program will jump between these different states by setting the state variable when certain events occur as outlined in Figure 4.3.1

From our implementation overview in our software design document, our final version differentiates from that implementation in one way. The obstacle avoidance module was not sufficient enough using only one kinect sensor. We had to include a sonar sensor facing the front of the robot because of false positives from the kinect when facing down either end of the hall. This sonar in the front gives a basic distance value to the object. We can account for this false positive by using the sonar sensor to confirm that an obstacle is in the way. We then had to add two more sonar sensors, one on each side of the robot. This is to account for the walls. The kinect sensor is not always wide enough to detect objects on the side. These sensors allow the robot to sense when it is too close to the wall. When it is, it will rotate in the opposite direction and continue.

5.0 Testing

5.1 Unit Testing

The purpose of unit testing is to ensure that even the smallest functions, parts, or units of a software project work as intended. The code is tested in small pieces called units. These units are given tests with certain input, usually decided by the developer, and then is checked against the expected output after the unit has been run. If the real output matches the expected output, then the test passes, and otherwise will fail. Often these units will be tested before and after the code and program changes to make sure that the changes made did not break any part of the program.

For this project, images are the primary data being used within the program. It relies on the processing and manipulation of this data. These images are gathered from the Kinect sensor during program operation. For testing, images are too complex to determine the expected output manually. To illustrate this, the simplest example of a unit test would be a sum function that takes two numbers, adds them together, and gives the result. This function may be tested with numbers, two and three against the expected result, five. This test would pass, but it was much easier to determine what the end result should be. For an example of image data, one function we use is to flip the black and white colors of an image. The images we process are 640 by 480 pixels leading to a total amount of just over 300,000 pixels. Even with a single image, determining every expected pixel value would be too time consuming. Therefore, our unit tests will be concerned with all parts of the program that do not involve returning images as result of computation.

To test the software, the team used the unittest module for Python. This module gives the tools necessary to create multiple tests for our code. These tests will be run to find any errors or bugs that the software does not handle. In the next sections, we outline the module the units belong under and discuss which units will be tested.

5.1.1 Control Module

The control module is solely concerned with and responsible for sending the output signals from the Raspberry Pi's GPIO pins to the motor driver boards. The robot is too unbalanced to handle large movement speed changes in a quick amount of time. Doing so could result in the robot jostling and possible tipping over. In order to account for this the function for sending signals to the motor drivers must ramp up and ramp down the speed smoothly.

- Motor Speed - The unit tests for this module are simple in any function such as *move_forward()*, can be tested against what the expected signal is from the GPIO pin. If the value of the signal is HIGH or ON then the test will pass. LIkewise, testing the *stop()* function with the signal value of LOW or OFF will pass.

- Motor Direction - One pin controls the direction that the motor is set to. Depending on the configuration of the motor, a HIGH or LOW signal will result in clockwise or counter clockwise movement. These pins can be read identically to the pins that control speed. For forward movement, we test that the robot's left motor direction pin is set to HIGH and the right pin is set to LOW since the motors are flipped but wired the same. For a left rotation, we test that both direction pins are set to HIGH to ensure that they are moving in opposite directions to create rotation.

5.1.2 Computer Vision and Obstacle Avoidance

These two modules are responsible for gathering the images from the Kinect sensor and raspberry pi camera and manipulating them to create the obstacle avoidance algorithm and to determine each end of the hall. As mentioned earlier, operations that return images as a result are too complicated to test accurately, so all other operations were tested.

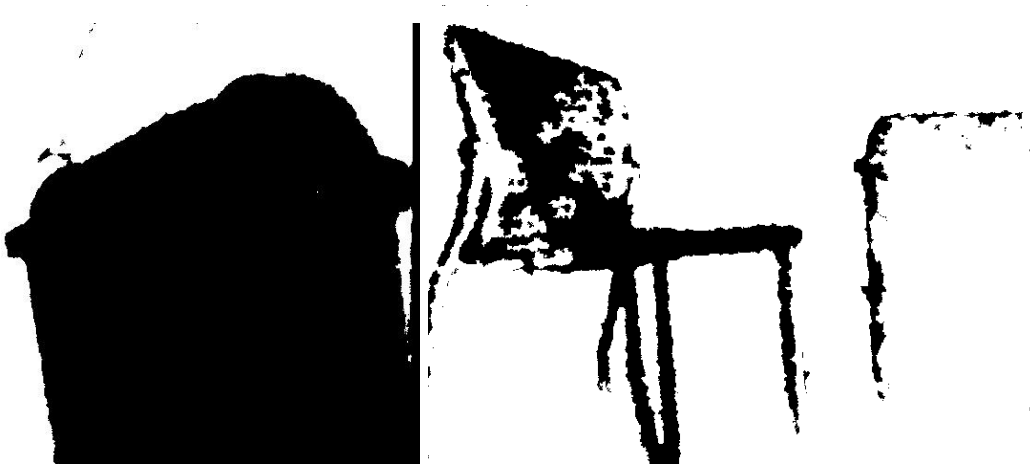

Figure 2.2.1 and Figure 2.2.2 Example Depth Images

- Obstacle Edge Detection - An important part of the obstacle avoidance algorithm is determining the side which an obstacle is primarily on. Knowing which side the obstacle is on also determines which side it is not. This allows for rotating to the

opposite side where an opening would be found. For this unit, it takes the depth image, finds all black pixels which equate to pixels of an object that is too close, and counts the total amount for each side of the image. Whichever side has more pixels found, that side is where the obstacle is primarily. Figure 2.2.1 shows an example depth image. In this example image there would be more black pixels detected on the right side of the image and detect the right side. And Figure 2.2.2 shows a depth image where more pixels would be detected to the left side. To test this unit, we use input images and an observation of which side an obstacle would be on.

- Obstacle Detection - The detection of an obstacle relies on finding the total area of these black detected pixels. If the total amount of pixels found is greater than a chosen value, the function returns true. A true result equates to an obstacle being detected. To test this we assert that with a given input depth image that an obstacle would be detected or not.

- Opening Detection - The principle used for detecting obstacles is nearly the same for detecting an opening in the path. The major difference is that the algorithm is counting the area of white pixels, or pixels belonging to objects out of range. The camera must also detect the majority of the pixels within the middle of the screen so that the robot can move forward through it. Testing this involves giving input depth images and the expected result whether an opening is detected or not.

- Lobby and Door Detection - Both ends of the hallway are detected using a machine learning model. Videos are given to the model which help to train it in deciphering what features belong to which end of the hallway or if they belong to the hallway itself. During runtime, the Raspberry Pi camera's images are captured and given to a function that uses the model to predict where the robot is in the building. For testing, we give the function images of both ends and the middle of the hallway. We assert that the given result should be what the image shows.

5.2 Integration Testing

The goal of integration testing is to bring together the components of the overall system in the Thirty Gallon Robot, in order to ensure that the product as a whole operates as expected. Integration testing will mitigate any possible bugs and errors in the communication between the modules that have been developed to run the autonomous module and obstacle avoidance. The tests of the Thirty Gallon Robot were conducted across the long hallway of the second floor of the engineering building.

5.2.1 Obstacle Avoidance and Autonomous Movement

The obstacle avoidance module works closely with the autonomous movement module, data given to the obstacle avoidance module takes 200 hundred milliseconds to process through the Raspberry Pi. As such the Raspberry Pi uses part of its processing power in order to gain information, such as obstacles and the end of the hallway. Computer vision uses the images received through the Raspberry Pi camera installed into the robot, and in order to find the end of the hallway, it attempts to recognize the differences in color and type of the material in the hallway. Because the lobby uses carpet, the final strip of the hallway is darker, the module attempts to detect it. However, this system is not perfect and might give false positives, as such the module must get three consecutive positives in order to recognize the end of the hallway. The other end of the hallway has a door instead of open carpet, and computer vision is also used here for detection accuracy. This approach also uses three consecutive positives in order to ensure the door is detected as the end of the hallway. The autonomous movement interacts with the obstacle avoidance module in order to turn around when the end of the hallway has been found. However, the obstacle avoidance module must also recognize any possible obstacles, as such a more generalized approach is used in order to create a better degree of practicality. This approach uses OpenCV in order to detect obstacles. This solution is desirable, as it allows the Thirty-Gallon robot to save on computation resources and ensures a good degree of speed in other processes such as detecting the end of the hallway. The autonomous movement is used in order to navigate around an obstacle, depending on the manner in which the obstacle is placed, if the obstacle is on the right hand side then it circumvents the obstacle by moving to the left hand side and vice versa.

5.2.2 Hardware and the Raspberry Pi

The hardware of the robot integrates with Raspberry Pi through the GPIO pins that connect to the motor drivers. The Pi connects to the sensors through USB, with the Kinect and through a ribbon cable for the Raspberry camera. Integration between all the components allows for a smooth functioning of the Thirty Gallon Robot, as it must operate the motor drivers for movement, and use the external sensors to determine where the robot should move. As such it is crucial that it has a complete connection to the Raspberry Pi. The Kinect is used in order to give live data about the environment that is being traversed and any possible obstacles present. The secondary camera is used in order to detect the end of the hallway, and constantly interacts with the computer vision function of the obstacle avoidance module. By using the hardware of the robot in tandem with Raspberry Pi, the obstacle avoidance, and autonomous movement module, the hardware in the robot is able to be fully used in order to move across the long hallway of the second floor of the engineering building.

5.3 Usability Testing

Usability testing is one of the most important aspects of a software system. End user interaction is ultimately what the software design revolves around and testing the agility of the application is required before the deployment of the software into a platform. There are many usability testing methods and picking the right one is crucial for the success of the project.

5.3.1 Qualitative Usability Testing

The thirty-gallon robot is built around automation and machine learning therefore minimal end-user interaction is needed for the operation of the robot. However, since minimal user interaction is needed for the front end of the application, there will be a command line user interface to communicate between the thirty-gallon robot and the end user. To ensure that appropriate measures are in place to satisfy the user experience, we conducted qualitative usability testing. As a facilitator, we asked participants to perform end-user operations using the command prompt connected to the raspberry pi remotely. As required from our client, we need to ensure that there is minimal interaction between the end user and the robot. To that essence, the end user will need to initiate the program by executing the python command “python3 thirty-gallon-robot.py” from one end of the 2nd floor hallway of the engineering building. Once the command is initiated, the thirty-gallon robot will start moving at a human speed so the user can follow along with the robot. Since there are multiple cameras connected to the robot, the end user can see the images in real-time using the “-v” command implemented into the software. The robot will also constantly feed the end-user with the location it is proceeding towards and the current location it is at as a

message, such as “hallway” or “end-of hallway” or etcetera. The end user will also be able to see how long the robot is taking to analyze obstacles and to which direction it will proceed next. Throughout qualitative testing, we are assuming that the end user has intermediate knowledge of python commands so the software can be initiated. Since the robot is equipped with machine learning, initiation of the software is all that is needed for the robot to proceed.

While there is no particular graphic user interface implementation on the front end of the software, qualitative usability testing is data will be collected. We are planning to get a minimum of five participants to test our software extensively and gather data on how we can improve user interaction between the software and the robot.

Currently, the robot has a command line based user interface and has constant display messages to the user on the status of the robot. From the qualitative usability testing, we analyzed the data and feedback we get from the user and improve our front-end application accordingly.

6.0 Project Timeline

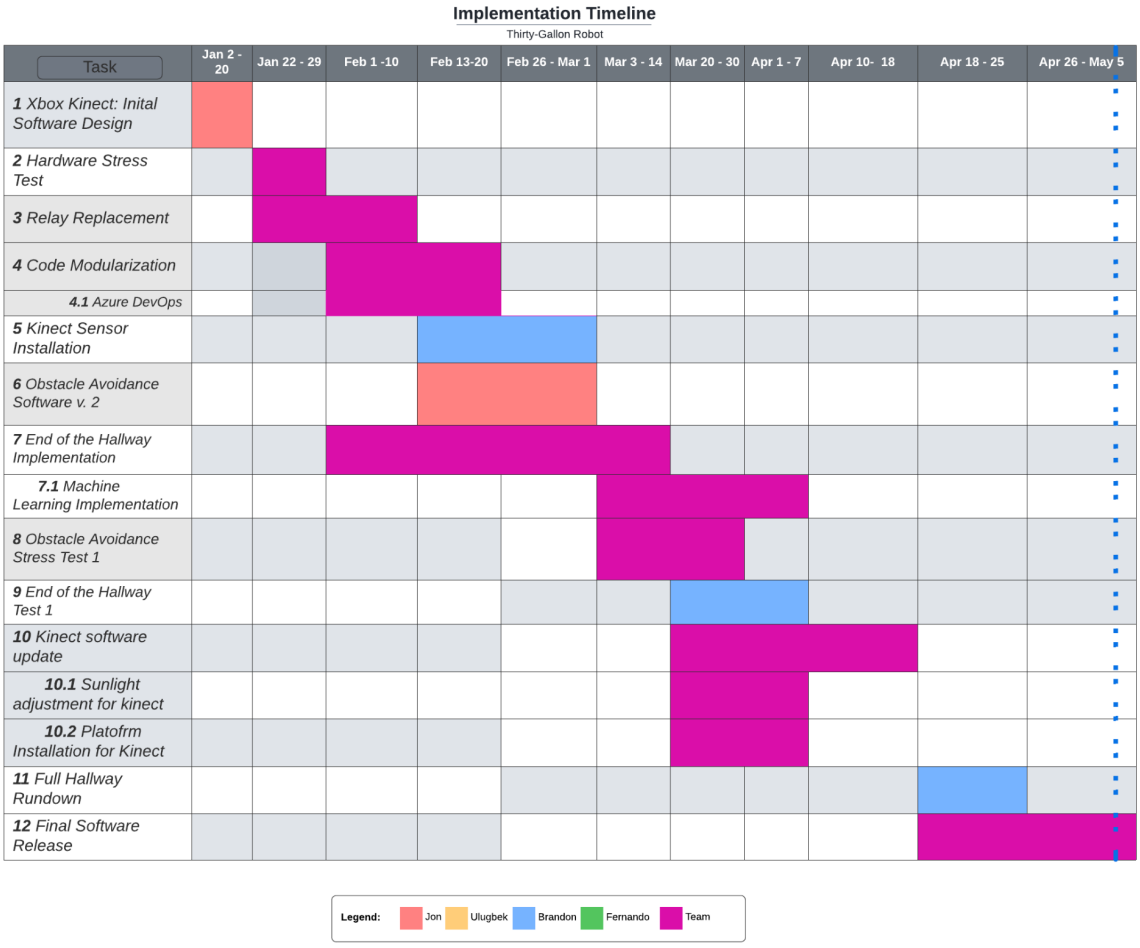

Figure 6.1: Gantt chart of Implementation Timeline.

As the architectural design of the software implementation has been established and different modules have been created for the robot's functionality, team Poseidon Wayfinding was able to create a well-structured timeline to ensure that the team is on track. During the winter break, Jon, the architect, was able to create an initial software design for the Xbox Kinect sensor for obstacle detection. While in theory, the program works as needed, the stress test proved that the robot has a right motor relay that is no longer functional. Two relays (left motor and right motor) were installed by previous teams and are one of the important electrical components of the robot. Therefore, the electrical architecture of the robot has been restructured. Brandon, the release manager is currently working on installing the Kinect sensors on the robot in an angled way so that optimal image detection can be achieved, in figure 5.1 it was expected to be done

during the week of January 31, 2022. However, it was delayed until the team was able to have the robot repaired. While the robot was being repaired, Jon focused mainly on the next software design for obstacle avoidance while working in conjunction with the team to implement end of hallway detection, the second version of the software was released on March 1. Since detection at the end of the hallway is a non-trivial implementation, the team allocated a few weeks to ensure that the design is flawless and bug-free. The new obstacle avoidance software was integrated into the end of the hallway detection, and the implementation was stress tested. Both tests are separated into two timeline segments and took a few weeks to complete since the functionality of the software is crucial to proceeding to the next phase. Although Kinect sensor software has been designed at the start, future updates may be required depending on the previous stress tests.

The final software will be released at the end of the given timeframe along with the user manual on how the software works.

7.0 Future Work

The future of this project is bright. Poseidon Wayfinding's implementation of autonomous movement and obstacle avoidance has laid the foundation for a robot that will one day be capable of giving fully automated tours of NAU's engineering building. The next steps that are expected would be to add a localization module to the robot, so that it is able to know where in the building it is at any given time. In addition to this it is expected that a team will be able to take this information and direct the robot around the building using the software. Finally, it is expected that a UI will be implemented so that users can select a tour to be taken on or a location in the building to be taken to.

On the hardware side, we expect multiple improvements to be made to the wiring and overall build quality of the robot. Specifically, we expect the wiring to be done in a way that would allow modularity with sensors and other components such as an arm that would be able to push buttons in an elevator. The overall build quality of the robot is also expected to improve as currently the robot jolts around jostling the components inside. An improvement that could be made would be the addition of a second rear wheel, providing the robot with increased stability.

8.0 Conclusion

Robotics is an ever-expanding field with limitless possibilities and the power of fast computations. Robotics components have gotten cheap and more powerful, yet there are few classroom uses of robots. Dr. Leverington noticed these two trends and came up with the thirty-gallon robot as the solution. The thirty-gallon robot is going to be an inexpensive autonomous robotic platform for use within college-level programs for educational purposes. This robot needs to be autonomous and programmable so that it will be able to give tours of the engineering building; this hopefully brings in more students to the program and inspire them. Dr. Leverington also hopes to make this solution extendable to other colleges and organizations in the future. Because the robot will be inexpensive it should be reasonably priced for these groups. Using the thirty-gallon robot as a recipe, these organizations would be able to create their own autonomous robotic platform for use in education. The full thirty-gallon robot project requires full autonomous navigation, and as a proof-of-concept that it supports programmability, will be given the task of giving tours of the engineering building. For our solution, we are implementing the first steps to this, autonomous movement and obstacle avoidance. In order to complete this, the onboard motors and motor drivers can be used to move the robot. Along with this, a Kinect sensor can be used to detect obstacles in the way. By implementing these two modules, it brings the thirty-gallon robot project closer to its end goal of full autonomy.

The giaTeam Poseidon Wayfinding is to ensure that the first steps towards a complete autonomous tour-giving robot are taken. The team is looking to implement an initial movement algorithm integrated within ROS to make the robot move autonomously in the 2nd floor long hallway of the engineering building while avoiding obstacles within the capabilities of harming the robot's movement. The software and hardware architecture of the robot is as follows:

- The Raspberry Pi 4B model will be the base computer of the robot. The team will install ROS on this computer to send movement signals to the left and right driver boards. The autonomous movement module will control the power and speed of the motors through this program, in which the rate of the robot is pre-determined to human walking speed. Consequently, the python algorithm will send initial directions and power to the motors through the driver boards.

- One Kinect sensors are positioned left and right-angled on the robot to ensure that any significant objects (with potential threat for the robot) within the range of 1-3 meters are being recognized.

The team's goal is to ensure that the architecture above is accomplished promptly. Therefore, the next big step towards achieving this goal is to create a mock-up version that will showcase the movement and obstacle avoidance features. Overall, team Poseidon Wayfinding is very optimistic about the prospects of this development. A well-structured timeline will help the team keep on track.

9.0 Glossary

Azure DevOps - A product created by Microsoft that allows software teams to keep track of a wide variety of things such as version control while working on a large codebase.

GPIO - General Purpose Input Output. Allows modules to interact with each other by connecting wires to pins.

Raspberry Pi - A small handled computer module. These are typically used to work on smaller hardware projects that do not require a large amount of computing power.

Xbox 360 Kinect - A sensor that was originally used to play motion-controlled games on the Xbox 360 gaming system. It was used in this project for its depth sensor which is capable of tracking objects.

10.0 Appendix A: Development Environment and Toolchain

10.1 Hardware

The Raspberry Pi uses the Ubuntu MATE operating system. For development, we advise using a linux environment to test python modules and other functions that do not explicitly require the robot. For computer processing power, even low-end machines are able to write code and transfer the files to the Raspberry Pi. These files can then be executed on the Pi without the need for a high-end computer.

10.2 Toolchain

VSCode - This tool is a text editor that we used for writing our Python code. It has many features that make writing code easier such as code highlighting and autocompletion. There are many plugins that can be used for Python, but in our experience the base settings for VSCode are sufficient.

FileZilla - This tool is used for transferring files between the computer used for development and the Raspberry Pi. It provides a graphical user interface to allow for easy visual transfer of files between computers. Anytime we had code to test on the robot, we used FileZilla to transfer the files from our computer to the Pi for execution.

Windows Subsystem for Linux (WSL) - This tool is extremely useful for those programming in a Windows environment that do not want to use a virtual machine or dual boot system. This subsystem provides all the tools needed to interact with a Linux filesystem and includes access to the vast majority of linux commands. For our project, it allows for python modules to be executed in a Linux environment and allows for remote ssh login into the Pi easily.

10.3 Setup

VSCode can be easily downloaded and installed from https://code.visualstudio.com/download. Similarly, FileZilla can be downloaded simply from https://filezilla-project.org/download.php?type=client. WSL takes a bit more work in order to get it properly installed. First you need to look in the microsoft app store and search for Ubuntu. You can choose a specific version, but the version that is just tilted “Ubuntu” will be fine. From here it is advised you restart your computer before proceeding. Now you can launch the application named Ubuntu. This will launch a terminal window allowing you to work in a linux environment. It will make you create a username and password first however.

10.4 Production Cycle

The main development cycle our team had adopted involved having a repository in which all our code was held. When we wanted to test with the robot we met up and edited the python files. Once we were ready to test it we transferred over the file to the Raspberry Pi using FileZilla. From here we used WSL and a bash terminal to ssh into the Pi and execute the python script remotely. From here, we test to see what works and what doesn't. We repeat the process, continually editing files, transferring them, and executing them. Once we finished the work for the day or the session, we pushed our code to the repository for the next time we were developing.