Software Testing Plan

Version 1.0

March 20, 2021

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 to problem solve and teaching them 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.

To accomplish that robotics platform, Dr. Leverington made the thirty-gallon robot, initially known as the robot-assisted tours 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. For students to use the thirty-gallon robot as an educational tool, it needs to be autonomous and support programmability. A student could create their robotics application or program and get hands-on experience with an actual robot; their program would use the robot's movement and navigation. The student would not have to worry about implementing these advanced modules.

2.0 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 will be using 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.

2.1 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.

2.2 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 will be tested.

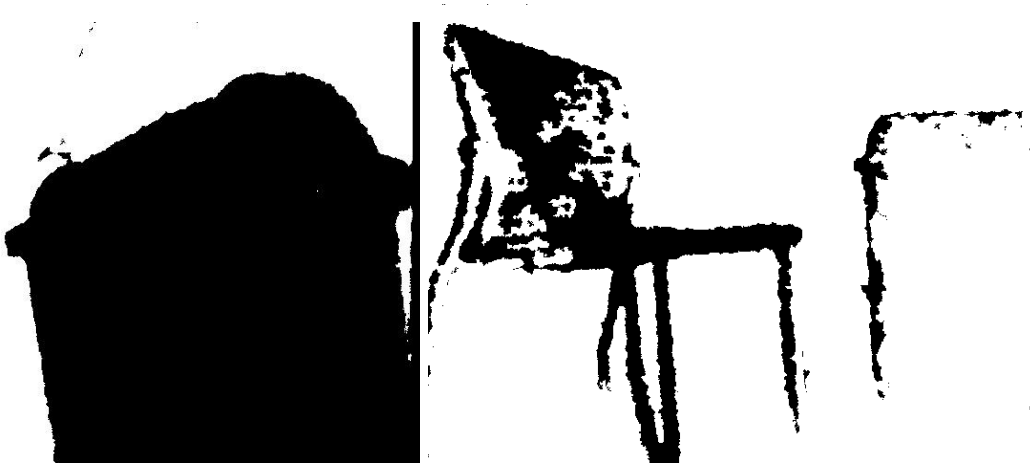

Figure 2.2.1 and Figure 2.2.2 Example Depth Images

- Obstacle Edge Detection - An important part of the obstacle avoidance algorithm is determining the side in 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 much quicker. 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.

3.0 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. Once the product has been tested, the team will be able to move to the final tests of the obstacle avoidance module. The tests of the Thirty Gallon Robot are conducted across the long hallway of the second floor of the engineering building.

3.1 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 the left hand side and vice versa.

3.2 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.

4.0 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.

4.1 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, 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. 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. 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 will analyze the data and feedback we get from the user and improve our front-end application accordingly. Having completed our alpha demonstration, we are looking to conduct extensive usability testing from April 4th through April 9th so we have enough time to adjust our software.

5.0 Conclusion

The tests for this project are broken up into 3 parts: unit testing, integration testing, and usability testing. Unit testing will be used to ensure the robot will be able to move functionally, and detect obstacles. This will be done by checking the values given to the GPIO board by the pi for movement, and checking the values of pixels given by the visual sensors on the front of the robot. Integration testing will be used to ensure all the functions of the robot will work together to achieve the overall goals of the robot. An example of this would be the robot detecting an obstacle then based on the given information, move the robot to avoid it. Usability testing will be done to ensure the command line functions to activate the robot will be easy to use by any capstone project that may follow the completion of this one.

In order to ensure the software that was developed is functional and holds to the required specifications, software testing is needed. Through this the team is able to properly keep track of what needs to be fixed or updated prior to the project being considered finished and given to the client.