Software Design Document

Version 2.1

February 14, 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 become an ever-expanding field, with endless possibilities. In fact, many industries nowadays use robotics to build cars, phones, and other products. Examples such as welding robots, that use robotic tungsten inert gas and metal inert gas welders are able to position different types of welding applications.Yet, autonomous robots are far from perfect, and even though many are impressive, there is a long road towards improvement, such as Tesla motors which seeks to improve autonomous driving.But Tesla's auto-pilot is still being developed. In fact much of Tesla's auto-pilot mode still requires human input, to use and is not fully autonomous, and is at level 2 autonomy.

With robotics becoming cheaper and more accessible, many organizations have started to put more effort into researching robotics. Due to this, educational organizations such as universities can open the doors to students to study robotics, from an electrical or software perspective. The Thirty-Gallon Robot, formally known as Robot-Assisted Tours (RAT) was created in an attempt to capture this trend of affordable robots. The RAT project sought to make a robot capable of giving tours of the school, autonomously moving through the engineering room.

The client, Dr. Michael Leverington, a lecturer of computer science at Northern Arizona University (NAU) is our team sponsor and his goal for the Thirty-Gallon Robot is to make robotics more accessible with an affordable platform. Dr. Leverington is interested in robotics and has seen this decrease in cost, as an opportunity to bring robotics to the forefront. The Robot Assisted Tours was started, which later transitioned into the Thirty-Gallon Robot to help students with a passion for robotics to have access to a platform. Also, many institutions that may not have the funding for robotics may be helped, opening their accessibility to the field of robotics.

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. For students to use the thirty-gallon robot as an educational tool, it needs to be autonomous and support programmability. Autonomy will allow the robot to give tours of the unversity, and allow anyone working on it to test their programs on the robot. 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.

The problem with the robot's current implementation is that it only has the hardware implementation completed, and there is no software to control the robot currently. These are the two features that the capstone project will focus on:

- Autonomous Movement: The robot will need to move independently of human input or interference. In this case, it will drive down the long hallway of the second floor of the engineering building and back.

- Obstacle avoidance: The robot must be able to detect obstacles through a sensor. Once an obstacle is detected, the robot must reroute itself to avoid it or stay on the path.

These two modules make up the fundamental part of the project. While the robot will not be capable of full navigation, it can follow a preprogrammed or dynamic path. To ensure that it gets there safely, it will use its obstacle avoidance obstacle to prevent crashing and keep on the correct path. More advanced modules for the robot, such as navigation, can not be implemented without the fundamental movement of the robot. Therefore, our solution is integral to getting the thirty-gallon robot to its end goal of fully autonomous movement, navigation, and programmability. This project will then be given to future capstone, which will be able to fully implement the thirty-gallon robot.

There are three domain-level requirements, autonomous movement, obstacle avoidance, and finding the end of the path. The main components that will achieve this goal, are the:

- A Raspberry Pi 4B model which will serve as the main control unit of the robot, the robot will serve as the base of the programs. It will have many parts of the autonomous module, such as obstacle avoidance and visual feedback.
- Two Xbox Kinect sensors that will allow the robot to detect obstacles. By giving the Raspberry Pi visual feedback, the Kinects will play an integral part in obstacle avoidance and finding the end of the hall. The end of the hall will require the integration of a simple use of computer vision, which will involve color gradient recognition. Such as reaching the end of the hallway will be recognized only when the robot reaches the carpet near the stairs of the second floor of the engineering building.

By amalgamating these requirements, the robot will be able to achieve a simple autonomous movement, and will traverse the long hallway of the second floor of NAU's engineering building and achieve the client requirements.

2.0 Implementation Overview

As mentioned in the introduction, the autonomous movement and obstacle avoidance modules are going to be implemented for the thirty-gallon robot. For the current module, there are two key features that this solution will require:

- The robot will be able to move autonomously down a straight hall and back
- The robot will be able to avoid obstacles while moving

To complete these features, there are multiple hardware and software components. Each component provides an integral part of the system.

Hardware

- Raspberry Pi 4 - The raspberry pi serves as the main computer for the system. It is responsible for gathering the data from the camera component for obstacle detection, and for sending signals to the motor drivers to control the robot's movement.
- Xbox 360 Kinect - This is the camera component for the obstacle detection system. The Kinect sensor has a built-in infrared sensor (IR) that can detect depths. Using this data, the distance to objects can be calculated with relative ease.

Software

- Python - It is the programming language that will be used for the obstacle avoidance and autonomous movement modules. Raspberry Pi can support any programming language that is supported for Debian systems. However, Python is installed by default and has access to the RPi.GPIO library which is integral for interacting with the onboard pins of the Pi and sending signals to the motor drivers.
- OpenCV - OpenCV is an open source library for Python and C++. It provides tools for computer vision, image processing, and machine learning. Once the data from the camera and depth images are received by the Raspberry Pi, OpenCV will be used to manipulate these images to create the obstacle avoidance algorithm and end of path detection
- Libfreenect - Libfreenect is a driver or library that allows for accessing Xbox Kinect sensors. This library is necessary for gathering the camera and depth image data provided by the Kinect sensor

3.0 Architecture Overview

The implementation of the system will require all the discussed hardware and software components to interact with each other. In this section, the flow of data from all components in the system will be explained.

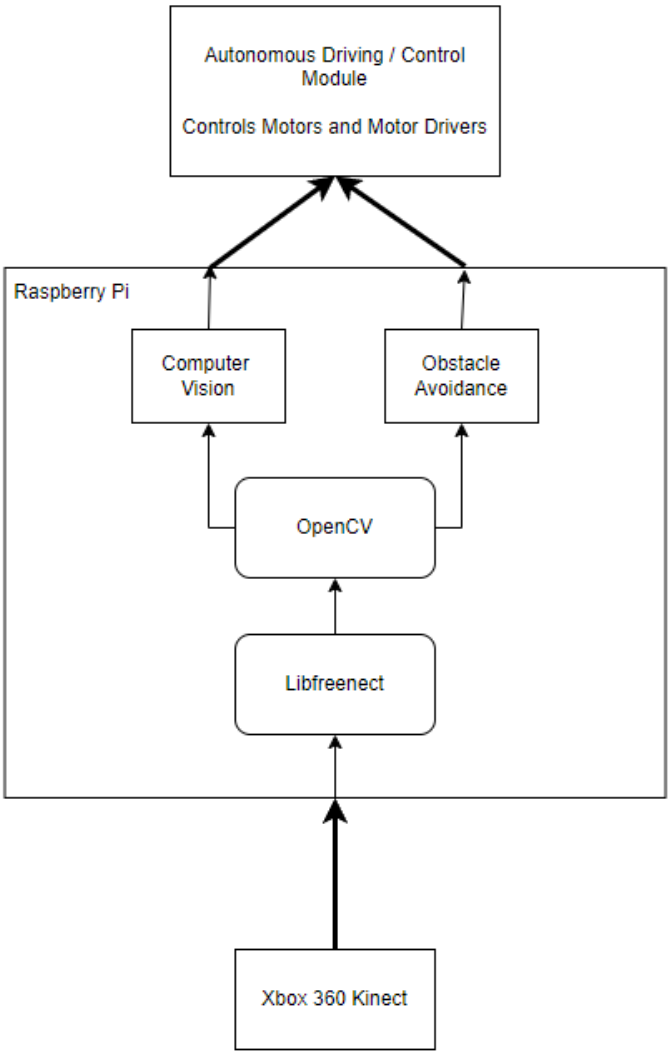

Figure 3.1: Diagram of the flow of data.

Figure 3.1 illustrates the flow of data in the hardware and software of the system. The Xbox Kinect collects two separate pieces of data, that being the RGB color image, and the IR depth image. The Raspberry Pi is where the code is held and computed to complete the system. Using Python and the library Libfreenect, the two images can be

obtained from the Kinect sensor. Using the OpenCV library allows for these images to be gathered, manipulated, and used to create the obstacle avoidance and computer vision modules. Each frame or camera image from the Kinect sensor is processed by both modules' algorithms to determine outcomes. There are two specific occurrences that each module is looking for, an obstacle in the way, and the end of the path respectively. The obstacle avoidance system must determine whether there is an obstacle in the way to avoid it. And the computer vision system must determine if the end of the path has been found and must stop or turn around. After the outcome has been determined from the algorithm, movement commands are sent to the control module or autonomous movement module. This module has access to the motor drivers and motors to create movement. In order to send the movement information to the motor drivers, it requires a GPIO pin connection to the Raspberry Pi. Using Python and wired connections from the Pi to the driver board, voltage signals are sent to set the strength and direction of the motor. Finally, the motor drivers allow power to flow through from the batteries on the system to the motors. The drivers use the voltages sent from the pin connections to determine the amount of power to flow resulting in the speed of the wheels.

This illustrates the full flow of data between every component of the system. The Raspberry Pi, Kinect sensor, Libfreenect library, and the OpenCV library make up the components that are integral to the system. The highest level of the system uses these components and includes three modules: Control module or autonomous movement, the computer vision module, and the obstacle avoidance module. These modules will be further discussed in section 4.0.

The control module refers to the interconnected systems that will allow the robot to move. Specifically the motor drivers, the RPi.GPIO library, and the software that will control the robot. It is basically the lower level control, which will be used by the obstacle avoidance and the computer vision system.

The computer vision module, is a very important part of determining the end of the hallway. However, the version used will be much more simple, and will used to analyze different color gradients. It will use a patter recognition algorithm to determine a darker color, as the hallways is made from lighter colors, the end of it will be the carpet along the halway. Given its permanence, this will be better as it will not be tampered with.

The obstacle avoidance module will be very crucial, as it will go hand in hand with assuring that the robot will be able to avoid obstacles. It will use the Xbox Kinect Sensors in order to determine whether any obstacles are present, if there are it will reroute its path. Otherwise, keep going.

4.0 Module and Interface Descriptions

This section describes the high-level modules that will be used to implement the solution. The components that were mentioned in the implementation and architecture overviews are used within these modules. As mentioned, there are three modules: the control module, the computer module, and the obstacle avoidance module.

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.

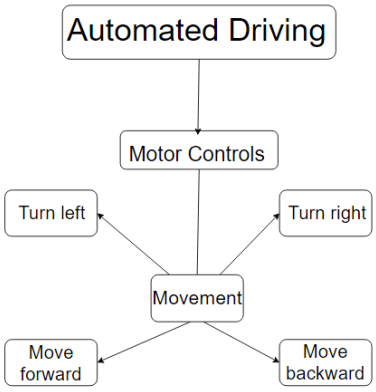

Figure 4.1.1 Control System Flow

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 [2]. 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)

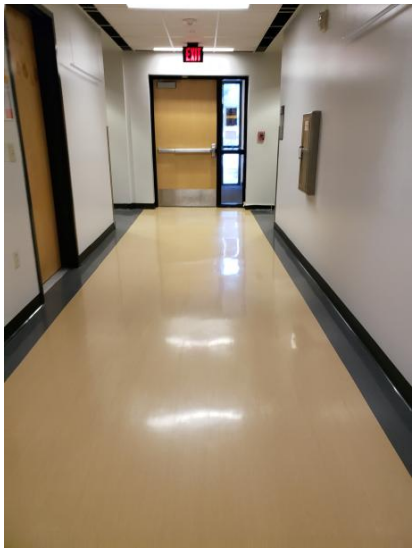

Figure 4.2.2 South End of Hallway in NAU Engineering building

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.

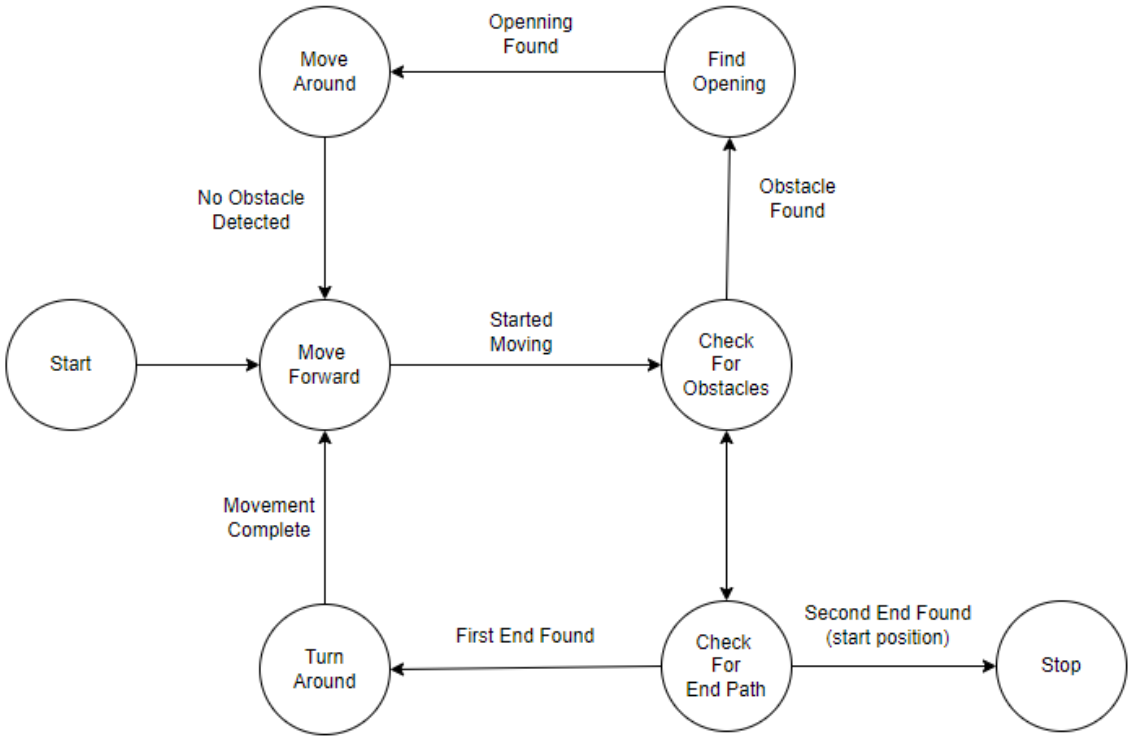

Figure 4.3.1 Simplified State Diagram of Obstacle Avoidance Algorithm

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.

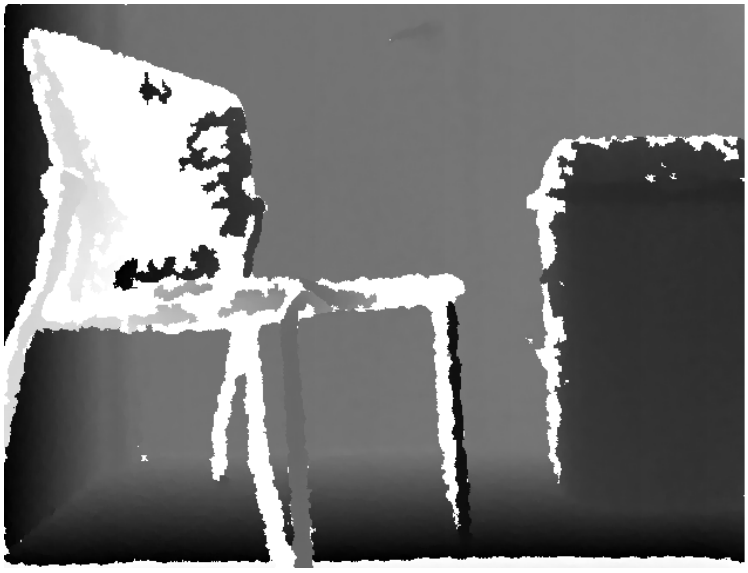

Figure 4.3.2 Depth Image

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

5.0 Implementation Plan

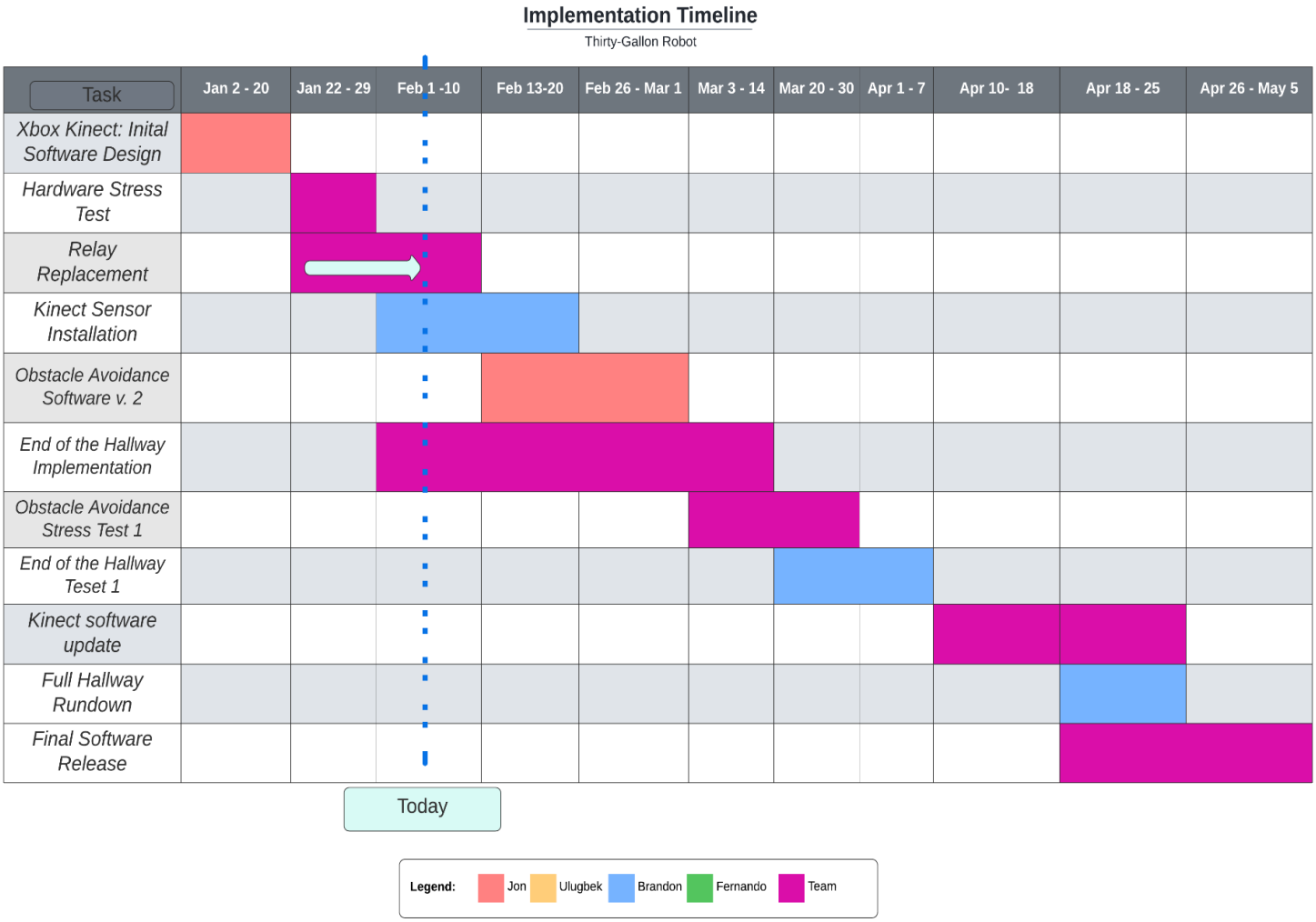

Figure 5.1: Gantt chart of Implementation Timeline.

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 entire team is currently working on replacing the relays with appropriate models so

further software implementations can be tested. Brandon, the release manager is currently working on installing the Kinect sensors on the robot in an angled way so the 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 will be done as soon as the Thirty Gallon Robot is returned to the team. On the other hand, Jon will be focusing 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 the software is expected to be released on March 1. Since detection at the end of the hallway is a non-trivial implementation, the team is allocating a few weeks to ensure that the design is flawless and bug-free. The new obstacle avoidance software will be integrated into the end of the hallway detection, and the implementation will be stress tested. Both tests are separated into two timeline segments and may take a few weeks to complete since the functionality of the software is crucial to proceed 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.

A full hallway rundown test will be held at least one week before the final release due in April 22, in order to solidify the robot's functionality in terms of whether the requirements are met and how the performance of the robot can be improved. The final software will be released at the end of the given timeframe along with the user manual on how the software works.

6.0 Conclusion

In conclusion, although robotics components have become affordable and more powerful, there are few classroom settings that use a working robot to prototype. The client, 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 students would be able to develop their own robotics applications and programs; this enables more practical and physical learning. 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 will 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.

In this project we aim to implement the thirty-gallon robot with the ability to move autonomously and avoid obstacles. In this document we have outlined the various hardware and software components that are required, and the high-level modules. 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 Raspberry Pi will be used for not only computation but also to send movement signals to the left and right driver boards.
- With two Xbox Kinect sensors 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.
- Using Libfreenect we can retrieve the camera images from the Kinect sensors, and using the OpenCV, these images can be manipulated and used within the main modules.

And the three main modules are detailed as follows:

- Control Module - Responsible for managing the movement commands to the motors and motor drivers. Other modules and systems can use this module to send their movement commands to the robot.

- Computer Vision - Responsible for detecting and determining when the end of the path has been found. Uses different algorithms to detect the end of each path in the NAU Engineering building.
- Obstacle Avoidance - Responsible for finding obstacles and subsequently avoiding them by moving the robot. Analyzes depth image information from the Kinect to determine whether there is an obstacle.

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.

Works Cited

[1] “Computer Vision vs. Machine Vision - What's the Difference?” *Appen*, 29 July 2020, https://appen.com/content-types/blog/computer-vision-vs-machine-vision/.

[2] “What Is Computer Vision?” *IBM*, https://www.ibm.com/topics/computer-vision.