Software Design: Version 3.0

2/16/2023

Floor Explorer Algorithms Team

Sponsored By: Michael Leverington

Mentored by: Rudhira Talla

Members:

Jacob Doyle, Armando Martinez, Luke Domby, Aiden Halili, Vincent Machado

Table Of Contents

1. Introduction **1**

2. Unit Testing **3**

3. Integration Testing **5**

4. Usability Testing **8**

5. Implementation Plan **12**

Figure 5.1 12

6. Conclusion **14**

1. Introduction

As the years have gone by, robotics has seen an increase in relevance in various fields of application. Unfortunately the classroom has not been one that has been able to keep up with this increase in demand. We have seen non-trivial simulators and limited functionality robots in the classroom setting, but what if we had a higher standard for robotics software? What if there was a way to allow for a standardized application of open source software to be used to educate the next generation of computer scientists? One that allows for its application on various robotics platforms?

Our sponsor, Michael E. Leverington has been attempting for years to create just that; a robot capable of being modular in its use and programmability; and a navigation software capable of keeping up with this varied platform. He recently came up with the idea of using the IRobot Create 3, to help create the baseline of the navigational software. The Create 3 is a relatively cheap (under $500) solution, and easily accessible to those with computer access. It comes standard with basic mobility actuators and a variety of sensors, all which can be programmed to complete user decided tasks. Our team, the F.E.A.T., is dedicated to creating a software platform that can be applied as needed to a variety of different yet similarly functioning robots. Aside from creating a modular platform, it will serve as a proof of concept and allow for the future education of students in robotics programming at Northern Arizona University.

We begin this challenge by looking at the previous versions of the project, all of which met with an early end. The project now varies in how it approaches the navigational component in its application to the robot, with object avoidance and localizations being its main features. Previous teams' failures have stemmed from multiple areas, with the biggest issue being their

focus on one specific aspect of their robot's sensor capabilities to control navigation. For example, one team chose to focus solely on self localization via router triangulation, which proved to be far too slow to be used in real time. Another team focused on virtual map creation but failed to come up with a solution for object avoidance.

Our solution is a more practical and modular approach to programming the robot that aims to combine the different attempts that previous projects attempted, in tandem with a newly created system. The use of the robust yet simple Create 3 to create a program capable of self navigation that could be moved to other robot platforms with minimal amounts of configuration. Our goal is to use real time sensor data to respond to possible obstacles in the robots path while using a combination of a digital map and wifi localization to ensure the robot is navigating to given coordinates. More specifically, it will use its various physical sensors and odometry mechanisms to keep track of its location and movements in real time while checking its supposed location using wifi to triangulate its position and confirm. This combination of approaches should allow the robot to successfully navigate around unforeseen obstacles, while maintaining a backup system, to ensure it is not moving off course. Our code will also maintain modularity by containing multiple constants that can be calibrated to fit sensors and actuators of those of which are not provided or compatible to the Create 3. We believe these changes from previous attempts will be enough to successfully resolve the programming problem, and in turn, help push their use in various classroom environments.

Now that we have a clear understanding of how we will accomplish our goal, the next step is to create a blueprint that will allow team F.E.A.T to be the first team to fully realize the goal of this project.

2. Implementation Overview

2.1 Solution Vision

Our plan to combine multiple aspects of previous teams solutions while creating an overarching navigation and avoidance system is both robust and straightforward. As stated before we will utilize the full extent of the sensor and odometer capabilities of the robot in combination with wifi triangulation. The sensors and odometry should be used to keep track of the robots location and movements for most practical applications while occasionally using wifi triangulation to self-localize using various nearby routers when necessary to ensure the robot is on track to its destination. For the sake of modularity, the code will have changeable constants that can be adapted to fit different types of sensors and actuators.

2.2 Tools and Techniques

- iRobot Create 3
    - This will be used for testing. It has its own proprietary functions, but they were only used to test the robot's capabilities and should not be implemented in the code. We intend to only use open source libraries through ROS and Python.
    - It has multiple onboard sensors (7 ir sensors, 4 forward facing cliff sensors, two motors for movement)
- Lidar Sensor
    - An external sensor that can scan in 360 degrees at 10 meters by emitting lasers and reading the reflections. This will help us build a map of the environment quickly. It can also be used to help detect objects in its path.
- Webots

    - A robotics simulation software that can be used to test code. Also allows for world creation.
- ROS2
    - Libraries that allow us to communicate with the robot as well as manipulate it.

3. Architectural Overview

3.1 Architecture Diagram

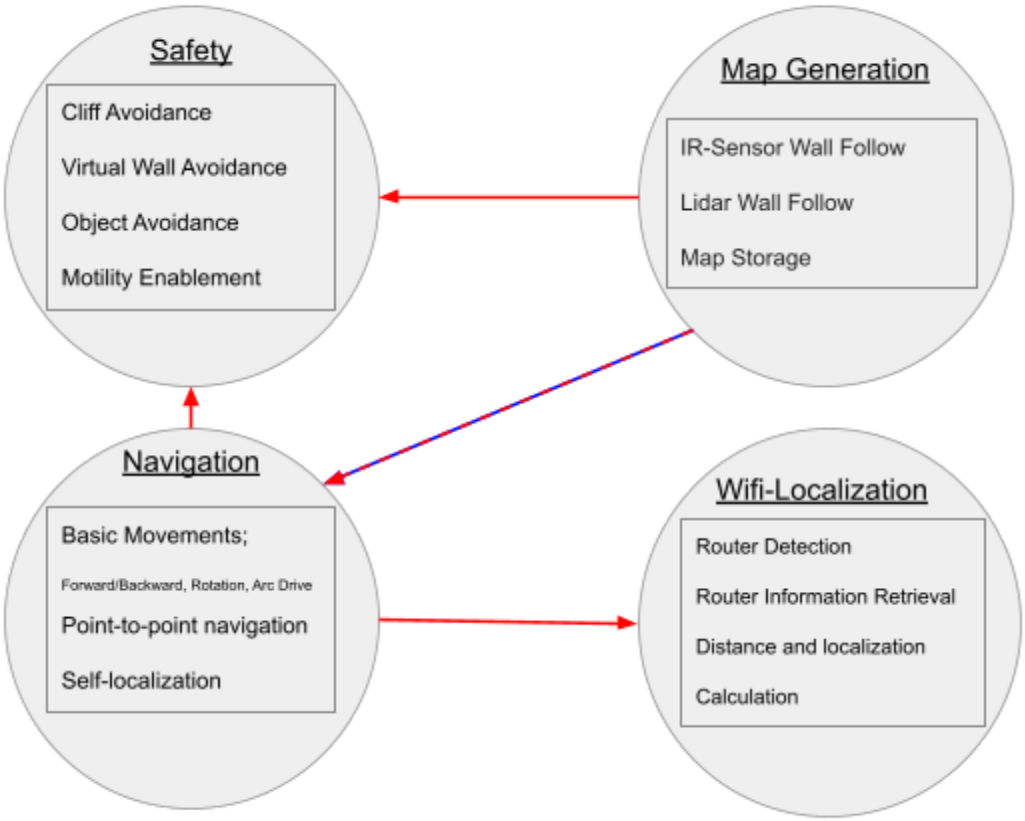

Figure 3.1

3.2 Architecture Discussion

The foundation for this system lies in its ability to create and store maps of a given building with capability to program in checkpoints along the way using a coordinate system. The quality of the map depends on the quality of the robots sensors, actuators, and odometry. The robot should be moved about its environment collecting sensor data on a guided exploration of its new environments.Once a map has been generated, the nearby routers have been categorized and the creation of various checkpoints on the newly created map with set router response times and signal strength, the robot will then be able to navigate the building provided that it has reliable wifi access. Due to the fact that the robot is supposed to be used indoors, it cannot use GPS since it generally requires access to the sky.

In order for the navigation aspect to work properly, we have decided to take a multi-faceted approach to object avoidance. It will read from the stored map and points to plot out where it needs to go and then execute that in real time. It will then rely on the sensors, both built in and external, to detect any obstacles it needs to move around. When it moves around an obstacle, the robot may have strayed a little ways from its intended path and may need to get back on track. Our solution to this problem is to have the robot triangulate with the wifi router corresponding to its most recent checkpoint so that the robot can self-localize and reorient itself without external help.

Since we are using the Lidar sensor in conjunction with IR sensors, we need to control the flow of input so that our software can respond swiftly to incoming data. Its front-facing sensors will always need to be reading input, but input from the lidar sensor should be slowed down to a manageable level, both to conserve battery life and preserve computing resources to ensure a speedy response to imminent obstacles. The wifi triangulation will be used sparingly due to the fact that it is a somewhat slow process which can eat up a large amount of navigation

time if solely relied upon. It also suffers from a long response time each time it's used, needing to wait for a response back from multiple routers and then comparing that to a database of expected responses.

4. Module and Interface Description

As our system is currently configured, we have 4 different modules that work together to provide a functioning robot. Let's start with our safety and object avoidance module.

4.1 Safety and Object Avoidance

It is important to note that this module is critical to safety and overall system functionality, and thus has been prioritized in the project progression plan. We have several functions here that require direct intervention from the current process, so the critical portions of this module will be used in every module that is developed after it. For example, the easiest application of this idea lies with the implementation of our cliff sensor reaction. During any movement process, the changing condition of the infrared cliff sensor will trigger the execution of a complete stop. This reaction ensures the robot’s safety and after moving away from the danger allows it to continue to navigate to its destination. The module can be broken down a bit further into multiple subsections.

- ➔ Class - Safety
  - ◆ Method - Cliff Avoidance
  - ◆ Method - Virtual Wall Avoidance
  - ◆ Method - Object Avoidance
  - ◆ Method - Motility Enablement

Each of these methods triggers a parameter change in the system and causes them to execute as they are detected.

- Cliff Avoidance triggers the parameter in ROS2 called “ uint8 CLIFF=2 ” is set. This will trigger the eventlistener nested a depending function and execute the protocol to backup and avoid the cliff.
- Virtual Wall Avoidance is similar in that a changing variable “ “ executes this process. The virtual wall is meant to prevent the robot from accessing areas it shouldn’t be in.
- Object avoidance executes the same avoidance technique while also taking movement speeds and proximities into account.
- Motility methods require less human interaction by triggering reactions to ensure mobility. For example, if the robot were to stall on one wheel, we can program the robot to attempt dislodging it. Similarly we can play a series of altering sounds to attempting kidnappers.

As you can see this entire class works together to maintain the robot’s well being and lets it adapt to a dynamic environment.

This module is also meant to utilize different sensor types to support modularity, so there will be a class for three different sensor types: lidar, infrared, and sonar. There will be a generic sensor class that the three sensor specific classes will inherit from. The difference is that each class will have its own constants for determining when an object is in close proximity.

4.2 Map Generation

Map Generation is more of a stand alone module. It does not depend on any other module, and only provides a database to be used for another module, rather than providing functionality for a different module. We constructed the system to simply hand the collected database to another module as an object. Let's break up this class into its individual methods.

- ➔ Class - Map Generation
  - ◆ IR-Sensor Wall Follow - Create3 implementation, documentation on changing nodes and middleware.
  - ◆ Lidar-Sensor Wall Follow - Manual control/Realtime Tracking

Manually control the robot, continuously collecting data points that are stored into the database in ROS2 coordinates. Note that this is all dependent on the origin of the robot pose, but adjustments can be made mathematically.

Realtime Tracking - As the robot follows a hallway, it should use AI algorithms to find its way around unexplored areas. This class functions independently in all cases except for this one, since the AI algorithms are likely to call our Safety class during its execution.

4.3 Navigation

Our Navigation class is fairly simple and most applicable to a user program. This class applies movements and navigation to desired coordinates, and depends on the Safety class in multiple instances. Let's take a look at it.

- ➔ Class - Navigation
  - ◆ Basic Movements - Forward/backward, rotation, arc drive
  - ◆ Point-to-point navigation - ROS2 provided
  - ◆ Intervention upon a safety condition being triggered
  - ◆ Self-localization information - Uses data from the robots sensors and odometry in order to self-localize. This function will be able to support different sensor types for the purposes of modularity.

4.4 Wifi Self-localization

While depending on the physical sensors on the robot to help with tracking movement during operation there will be moments where attempting to solely rely on them will not reliably ensure that the robot is where it needs to be. In order for this module to work it needs to be able to communicate with the various wifi routers inside the building. The map used for navigation will have a series of checkpoints built into it where the robot will use wifi triangulation to self-locate at the arrival of each one. This module should retain all relevant information about the routers it communicates with and only be active when called upon by other components.

- ➔ Self-locate at checkpoint
  - ◆ When the robot reaches a checkpoint, call both self-locate functions below and compare the results.

- ➔ Self-locate with odometry
  - ◆ Gets an estimate of the robot's current position using odometry data from the navigation module and should be continuously running during normal operation to help navigate.

- ➔ Self-locate with wifi
  - ◆ Scan for a set of nearby wifi routers and try to find a match with the stored data on all the routers inside the building and get the matching data which will then be used to triangulate the robot's position. May cause the robot

to relocate if it is found to be off course.

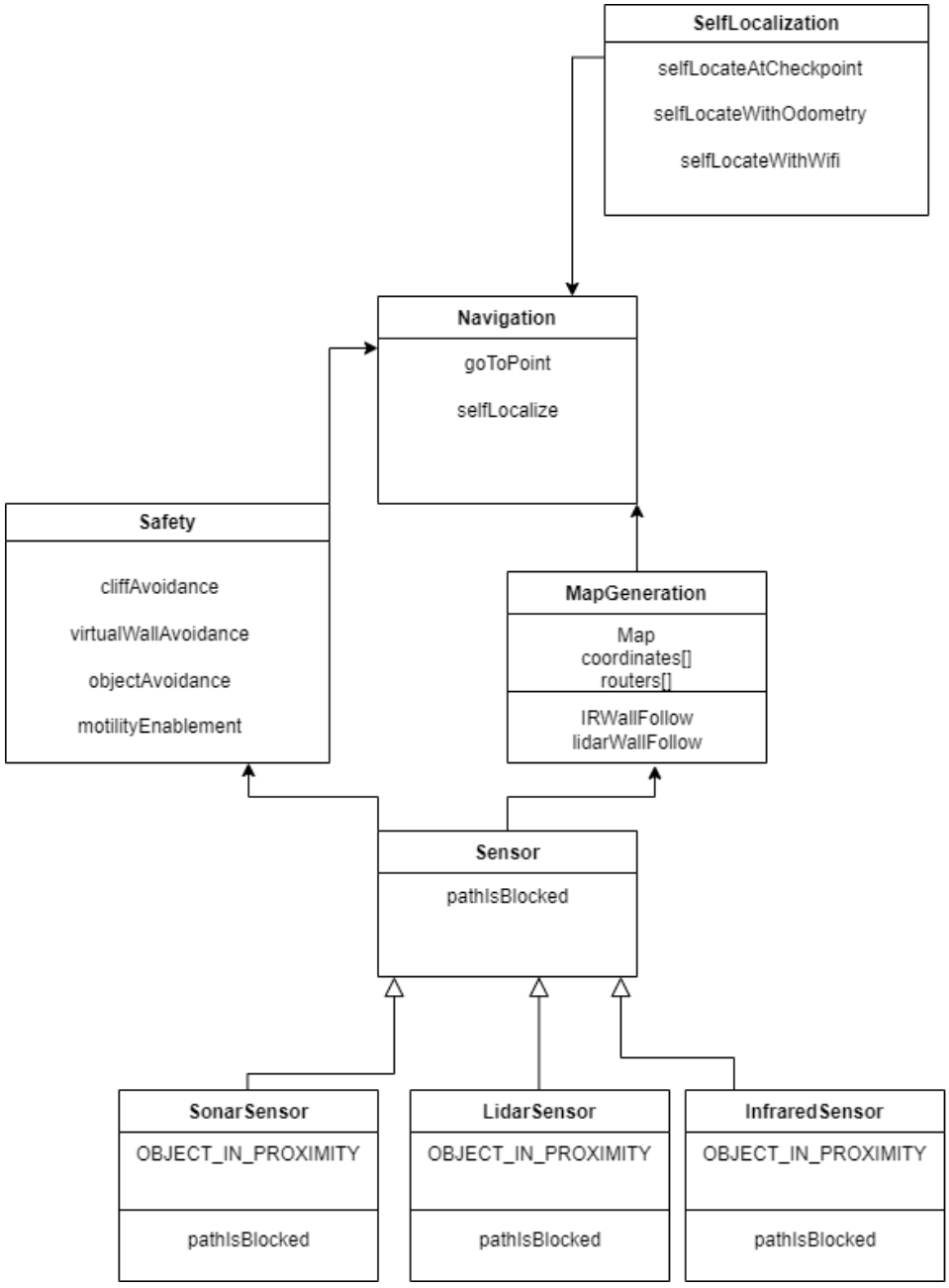

Figure 4.1

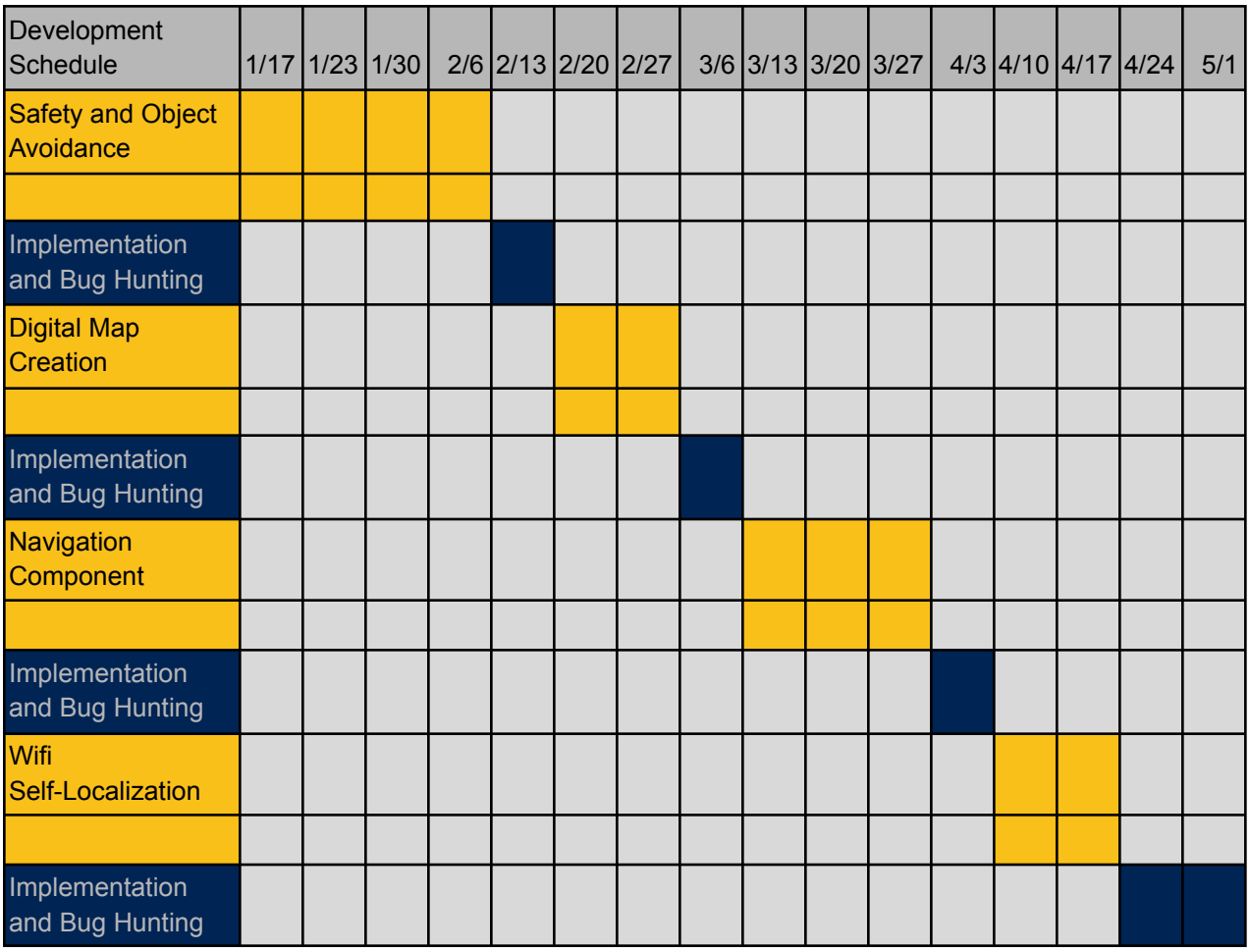

Figure 5.1

There may be four parts we need to complete for this project but not all sections are created equally. The first phase, safety and object avoidance, is a large part of our system and will be relied upon by almost every module after it. As such it is important to get this done first and be given enough time to ensure that it is reliable and fully complete. This will most likely be done with something similar to an event listener which will interrupt the movement so it can decide how to keep moving forward. As such an extra amount of time is allotted to this section

to ensure its functionality, both to make sure we don’t have to return to it too often and to ensure the safety of our robot as we begin with live tests.

Next is the creation of a digital map, the two following systems will pull data from it and it will also be our basis for navigation. However, as we have made significant progress in its research, we feel confident that it will not be necessary to alot a large amount of time too. We will use the sensors, specifically lidar, to gather data on the area being navigated. This will be done by using a wall function or manually piloting the robot around the desired area. If choosing the use of the wall follow function, the mapping will depend on the safety module to ensure the robots safety as it goes around corners to collect data.

After being able to create a map of an area the robot will need to actually move. We will implement functions in order for the robot to move to different locations on its own. This section will be the culmination of the work from the last two modules and as such requires a significant amount of time to ensure its functionality and robustness.

Our last step to finishing this project is to have the robot use the WiFi in order to place itself on its map. This is important because the robot has a margin of error with its coordinate system so it will get off track slowly and will help aid in the robot's initial boot, allowing it to function even if it starts away from the ideal starting position. With this function it won’t be an issue that we need to worry about.

At the moment we have everything planned to take at least 2 weeks with a week to test and implement the system, with exceptions to the two larger and more daunting sections. Currently we are planning to work as a team on each part during the specified times, but also dedicating a team member to preemptive research and planning for each section to help teach the other members of the team once that section's work begins.

6. Conclusion

With the use of ROS and Python libraries to create a robust and modular programming platform, we will be the first team to be able to complete this project. Our goal to help bring about a more accessible platform for robot programming and education is within our grasp and with our roadmap being laid out in front of us we feel strongly about our chances of success. We believe in the cause of this project and that its full completion is vital to the improvement of education for future generations of computer science students and opening opportunities for learning about robotics. We hope to only be the beginning of what will be a long line of computer science students learning about and improving upon the field of robotics at NAU.