Requirements Specification: Version 1.0

12/08/2022

Floor Explorer Algorithms Team

Sponsored by: Michael Leverington

Mentored by: Rudhira Talla

Members:

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

X_____ X_____

Table of Contents

1.0 Introduction

As the years have gone by, we have seen an increase in the use of robotics in many different fields, but unfortunately, classrooms have not been able to keep up with this rising 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 integrate more of this at an “affordable” cost? When we say affordable, it’s important to denote that this means affordable in terms of the average household, and not just that of an academic organization.

Our sponsor, Michael E. Leverington, has been attempting for years to create both; a robot capable of being modular in its use and programmability, and modular software implementing a basic navigational component. He recently came up with the idea of using the IRobot Create 3, to help test the navigational software components. The Create 3 is a small circular robot with similar sensors and capabilities to that of a Roomba vacuum, a common robot found in society today. The Create 3 is a relatively cheap (under $500) solution, and easily accessible to those with access to computers. It comes standard with basic mobility actuators and a variety of sensors, all which can be programmed to complete user decided tasks. Team F.E.A.T. is dedicated to creating a robotics platform that can be programmed as needed to a variety of similarly functioning robots. Aside from creating a modular platform, it will show proof of concept and function as a tour guide for the 2023-2024 academic year at Northern Arizona University.

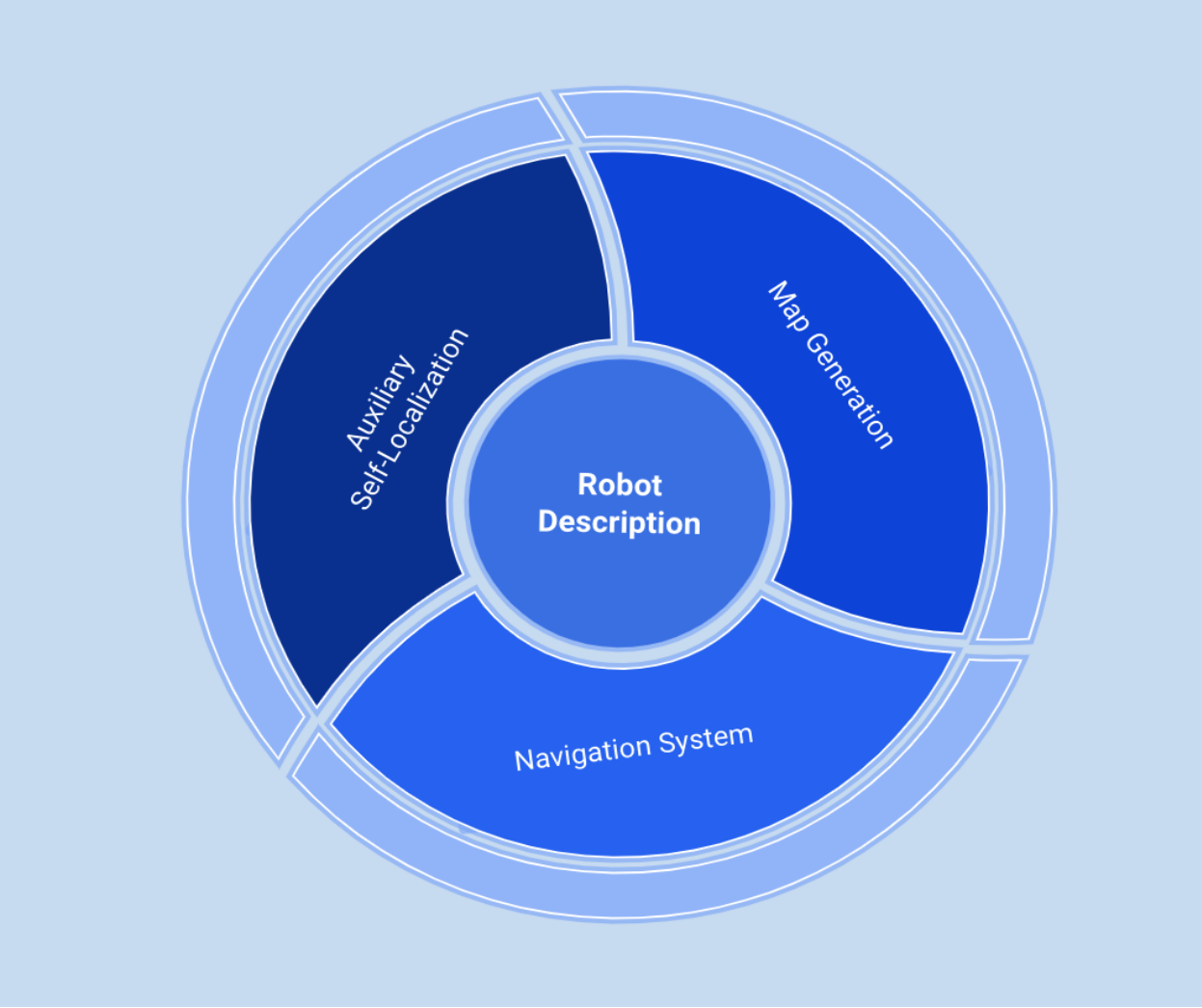
2.0 Problem Statement

As we stated above, getting your hands on robotics equipment is becoming easier and more affordable, but how does this benefit the academic community? Our Create 3 robot comes standard with mobility tools and data collection, but all of this happens under the hood of the proprietary product. For example, we can program the robot to move in any direction, rotate, and even sing in any number of different orders and lengths. This might be beneficial to beginners, but we aim to make a navigational software that intermediate and more advanced students can learn from. This is exactly why our product will be an open source, easily implemented, and most importantly, a robust piece of software. Academia will greatly benefit from this product as it can manipulate the code however the organization seems fit, which will ultimately give students the tools they need to better the robotics industry.

More specifically, our problem is within the idea of self localization. That is, that anywhere in a designated area, the robot should be able to detect its location. Dr. Leverington has attempted to create this software for years now, and now that we actually have hardware that is up for the task, we believe that he will finally see this project succeed. Previous halts in production included this lack of hardware, lack of testing, and most importantly, a lack of modularity. A previous attempt did very well for this idea of self localization, however, the software was not implemented on any robotics system, and in turn, not very applicable to the end goal of this project. We've seen many teams conceptually understand the idea, but either the system was imperfect or not robust enough, have been the main reasons this product hasn't seen the public eye yet.

Ultimately, we have a clear set of objectives with this project:

- Self localization via;
    - Wi-Fi triangulation
    - Coordinate system orientation
- Efficient mobility and obstacle avoidance
- Guaranteed safety mechanism
- Sensor compatibility



3.0 Solution Vision

In order to accomplish these goals, the software cannot be single use for this specific robot. This is the main difference in how FEAT will conduct the project, and why it has not had success in the past. Previously, the team was assigned a robot and the task of implementing self localization. With the previous version robot, there were various hardware problems that caused the project to fail. Another version of this project, in fact, didn't have a working robot. We can see that although having a modular software component, we need the ideal robot to test and implement the software on. From there, the modularity will have already been virtually completed due to how we design the navigation system. The fundamental idea that our project is based upon while having an easily accessible and programmable robot at our disposal could very well be the difference that this project desperately needed.

We plan on using an industry known robotics language called to ensure that the modularity of the software is maintained. It is easily recognized by multiple embedded computer/robotics systems, and is the perfect tool to execute commands without having to dive too deep into the logistics of robot actuators.

4.0 Project Requirements

Our project is based on four domain-level requirements: **Robot Description, Map Generation, Navigation System and Auxiliary Self-Localization**.

- **Robot Description** lists the hardware and software recommended for this project as well as the hardware/software provided by the client. It also describes the data required by other requirements .
- **Map Generation** focuses on the use of sensors and odometry to create a map, a pre-requisite for Map Navigation and Auxiliary Self-Localization.
- **Navigation System** targets the navigation through the previously generated map, following a given route given by the user, and reacting in real-time to obstacles along the way. In addition to that, the Navigation System is designed to keep track of the robot's location, storing the odometry data provided by it in real-time.
- Given the problems involving self-localization from previous iterations of this project, we have decided to add an additional requirement, **Auxiliary Self-Localization**, which uses Wifi Routers Triangulation to give an additional layer of self-localization to the robot.

4.1: Robot Description

In order for our project to maintain modularity, it is necessary to set a few standards for anyone planning on implementing this software. For example, we must ensure that the users have compatible sensors and actuators in the robot they plan to use. We plan on constructing our code so it can support a number of different sensors, but the user must ensure that theirs is going to be compatible with our code.

4.1.1 Robot Description: Functional Requirements

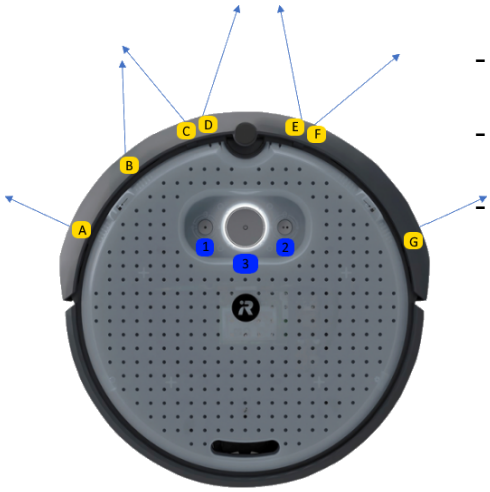
- For the user to implement this project, it is required to have:
    - A robot with the following features:
        - Capable of running ROS2 commands.
        - Access to a mountable LIDAR scanner (or equivalent) in real-time, or similar IR functioning sensors.
        - A consistent and accurate source of real-time odometry data.
        - An onboard raspberry pi unit, or any other embedded computer unit.
        - A battery life with at least 1,800 mAh and maximum Voltage of 16.8 V.
        - A storage capacity of at least 128GB

4.1.2 Robot Description: Performance Requirements

- The features described in 4.1.1 will be used in the following way:
    - Running ROS2 commands in fundamental to run this project's code
    - LIDAR real-time data (or equivalent) will be used for the Map Generation and Navigation System requirements.
    - Odometry data will be used in all other requirements.

4.1.3 Robot Description: Environmental Requirements

- Robot Hardware/firmware provided by the client:
    - The robot provided by our client is an iRobot Create 3, that has the following specifications:



        - Firmware version: WIP
        - 7 pairs of IR proximity sensors (Letters A-G in Image 1)
        - Three physical buttons
            - Power Button (Button 3 in Image 1)
                - Features a ring of six RGB LEDs for indication.
            - Buttons • and ••
                - Buttons that can be repurposed by the user

        - Four cliff sensors (Letters H - K in Image 2)
        - Optical Odometry Sensor (Number 4 in Image 2)

4.2: Map Generation

As stated above, a known coordinate system of the robot's environment will be our primary focus in this version of the project. Diving deeper into this concept, it is important to understand that this is not a "plug and play" component. An initial coastal navigation function must be performed before this piece of the software can be useful to its users.

4.2.1 Map Generation: Functional Requirements

- The robot should be able to map an entire floor, which it will accomplish based on the following functions:
    - A coastal navigation function used to draw a map of the floor
        - Alternatively, LIDAR or similar IR sensors allow for the robot to travel down the middle of halls without needing to move alongside a wall.
    - A function that will read sensor input, which will be called by the coastal navigation function. The sensors should constantly be accepting input which it will store as points with coordinates to build the most accurate map possible.
    - An obstacle avoidance function that will work slightly differently than the one we plan on using for normal navigation since the robot has to reorient itself to be parallel to the wall or object it encountered as opposed simply going around it. The most common scenario where

this function will be implemented is when the robot runs into a corner, or any type of object resting against the wall

- A function to handle turns for cases when the robot either goes into a corner, or if the robot needs to round a corner.
- In the event that the robot stops due to the cliff sensors, it will turn at a specified angle so that it can follow ledges such as the top of a staircase.

4.2.2 Map Generation: Performance Requirements

- The functions will work as described below:
    - The coastal navigation function will implement the wall-follower algorithm, which dictates that following one side of a wall in a maze where all walls are connected (i.e. a simple maze, which is how most buildings' floor plans could be categorized) will lead either to an exit or back to the starting point. It should be able to follow the same side regardless of sensor type. It can be set to follow a wall to its left or right.
    - The LIDAR sensor has a greater effective range than the IR sensors. While the default IR sensors can detect objects from only one foot away, the LIDAR sensor can detect objects up to 12 meters away. This means that the LIDAR sensor will capture more data than the IR sensor will, thus allowing for an accurate map to be made in less time since it doesn't need to follow a wall.

- The time it takes for the coastal navigation function to finish depends on the size of the floor plan, but the most important aspect is that it shouldn't require any human assistance beyond opening doors or calling elevators.
- If the robot is following the left wall, it will always turn left to round a corner and right when running into a corner. If it is following the right wall, then it will always turn right to round a corner and left when running into a corner.

4.2.3 Map Generation: Environmental Requirements

- The accuracy of the map being generated hinges on the hardware provided by the client.
    - Since the left side of the iRobot Create 3 is more sensor-heavy than the right side, it would be more practical to perform coastal navigation based on its left side.
    - The software platform as a whole must be modular so that it can be used on other robots. So while it may be tempting to use an existing Create 3 wall follow function, it is proprietary and will not work on the thirty gallon robot as it isn't a Create 3 robot.
    - There are certain areas in the engineering building that the robot should stay out of. For instance, it shouldn't be going into bathroom stalls and it should never leave the building if there happens to be an open exit.

4.3: System Navigation

Once the robot has finished performing a coastal navigation of the building the outcome should be a useful map stored in an easily accessible database. At this point, it will then be ready to plan and execute a path to take on the floor it's on.

4.3.1 System Navigation: Functional Requirements

- After the coastal navigation has been completed, the robot is now ready to perform the following functions:
    - Add coordinates to the map, which will serve as checkpoints the robot should visit.
    - Each checkpoint should be sorted from closest to farthest from the starting point to prevent unnecessary backtracking.
    - An obstacle avoidance function which will be called when sensors coming from in front of the robot reach a certain value. It should execute in the following steps:
        - While the front sensors read an obstacle, it will continue to rotate in increments of 45 degrees until there is space in front.
        - Move forward until the obstacle is no longer detected by peripheral sensors.
        - Rotate back to its original position and continue moving to the next checkpoint

4.3.2 System Navigation: Performance Requirements

- Generating the route in which the robot will take is dependent on user input of coordinates. This will require the user to understand a given robot's coordinate system. The Create 3, for example, defines positive x-dimension as being forward and positive y-dimension being to the left of the robot.
- When trying to move around an obstacle, the robot should be able to alternate between clockwise and counterclockwise depending on which side has the least input to the sensors. This is useful in case a person runs into the robot and accidentally moves to the side it was trying to rotate.
- When it finishes reaching a checkpoint, it should be accurate within a certain margin of error depending on the odometry. The Create 3 is capable of moving to a location with a 5mm margin of error, but other robots should be accounted for.
- The plan is to take advantage of wifi triangulation to ensure as accurate of movement and positioning as possible.

4.3.3 System Navigation: Environmental Requirements

- The robot should not hit anything while navigating, so bumper sensors should not be utilized when avoiding obstacles.
- Avoidance functions will have to work with real time movements in the environment as well, such as people crossing the robot's path. This will include varying types of avoidance functions that can adapt accordingly. For example, a still chair in the middle of the hallway should invoke a different avoidance mechanism than that of a student walking directly in front of the robot for a short period of time. Depending on the type of sensor that the robot is using, these functions will have to vary slightly.

4.4: Auxiliary Self-Localization

Auxiliary Self-Localization is meant to be a complementary requirement to the System Navigation component. While System Navigation relies on the map previously generated to keep track of odometry and sensors, Auxiliary Self-Localization relies on a map of Wifi Routers, provided by the user, to determine its position.

4.4.1 Auxiliary Self-Localization: Functional Requirements

- Auxiliary Self-Localization is intended to work in the following order:
    - Given the odometry and localization data provided by System Navigation, determine an estimate of the robot's current position.
    - Then, the robot starts scanning for nearby wifi routers.
    - Match the wifi routers detected by the robot with the router map we already have. Extract the relevant data from the map.
    - After that, we use the data obtained to triangulate an estimate of the robot's position
    - We compare both estimates (odometry based vs triangulation based) and confirm that both of these estimates confirm each other within a margin of error.

4.4.2 Auxiliary Self-Localization: Performance Requirements

- The auxiliary self-localization will be able to accurately estimate the distance of nearby routers, and using that information, it will be able to find its own location.
- This will all happen in roughly 15 to 20 seconds which is far too slow to function as a primary localization method. This is why we've chosen to use it as an auxiliary technique. From the information gathered among previous project attempts, we've concluded that this mechanism is highly accurate yet quite inefficient, which should function perfectly as we intend to use it.

4.4.3 Auxiliary Self-Localization: Environmental Requirements

- For the purposes of this project, we are going to limit our scope to the list of routers available at NAU. More specifically, we must narrow this down to the routers in the engineering building and those of the buildings surrounding it.

5.0 Potential Risks

The immediate potential risk that comes with our current plan is the compatibility between different sensors. For example, the Create 3 comes standard with 7 infrared sensors. We are currently implementing compatibility with a 360 degree lidar sensor. The amount of data points from this sensor greatly outnumber that of our standard sensors, so we must ensure that we can handle all of this data. This problem also occurs with sonar sensors that Dr. Leverington plans to use for guiding tours throughout the Northern Arizona University engineering building, and various other sensors that someone might wish to use.

Secondly, the difference in mobility between robots might be a factor when dealing with robustness. More specifically, the odometry between robots might be different in that its accuracy might vary. In many ways however, this is out of our project specifications. This is a hardware requirement that is out of our control.

Something important to consider is the difficulty in connecting the Create 3 to the NAU wifi due to an inability for the robot to input a username sheds light on the fact that another school's protection mechanism for their wifi can be a major roadblock for the auxiliary self-localization feature, which is essential for making corrections should the robot go off-course due to obstacles.

Another potential risk, although minor, is that the cliff sensors on a robot may prevent it from entering an elevator, and subsequently prevent it from exploring other floors without additional human intervention.

6.0 Project Plan

Our first milestone in this project is to ensure robot safety. Although our current robot has built in protection with downward facing IR sensors, not every robot will. We can however assume that every robot will have a mechanism for detecting stairs or drops, but we must initially write a function that can act upon red flags that represent danger to people or the robot itself.

Second, we must write a function that will allow a robot to learn the environment. To begin this task, we must first ensure that a robot can perform simple tasks such as; move along a wall while simultaneously collecting data points, store collected information in an accessible database, and apply an accurate coordinate system to these collected data points.

Third, we must implement object avoidance. To do this we will use a lot of the same functions that our “coastal navigation” system uses. At this point, the robot should be able to recalculate a trajectory because of its known database of coordinates.

Fourth is implementing self localization via wifi triangulation. For this we plan to either use an onboard router, or the robots MAC address to determine what routers it can detect nearby. From this information, we will use a plan that a previous team used, although the mechanism is not efficient in a timely sense.

Finally, we will show proof of concept by programming a simple tour throughout the second floor of the engineering building. This should be extremely nontrivial at this point as it will rely on a known coordinate system to maneuver from point to point and conduct simple tasks in between.

7.0 Conclusion

With robots becoming commonplace it is imperative that schools and institutions are able to keep up with the ever changing world around us. Educating future students on the complex topic of robotics will require multiple learning tools to give the students the resources necessary to make programming robots approachable. Our goal is to help further that standard of education, to create a baseline that will allow for increased student engagement. It is important that we create a system that will let future students be able to learn and experiment with the software of varying robot platforms with ease.

Our solution is to create a modular software system based off of the Create 3 robot that allows for students to be able to have a starting point to use for any robot system that uses ROS2. To create a functional navigation system that works by using the available sensors to map a building and then using a coordinate system to help navigate the now digital map. This should be compatible with any robot with only minor alterations to specific variables to account for differences in sensor feedback and sensor strength. Our program will allow for the robot to avoid objects while moving to its desired location and will confirm its movement along the way with occasional self localization. Our research into the technology that will allow us to complete our project has led us to various different discoveries regarding the sensor capabilities that the robot possesses. We found that while they do have a fast response time they do not possess the range we were hoping for and as such have started looking for alternative sensors such as mountable LIDAR. We also discovered that various systems in ROS2 function on quaternions and use a different x and y plane than expected.

These important discoveries will allow us to create an autonomous system for navigating the real world using digital maps and allow for students to have a platform to further their education with. It is important that we help open new opportunities for students as the world gets more advanced and complex. This project will have both a practical use of being a tour

guide and an educational use as being an important stepping stone towards furthering student's education in the field of robotics.