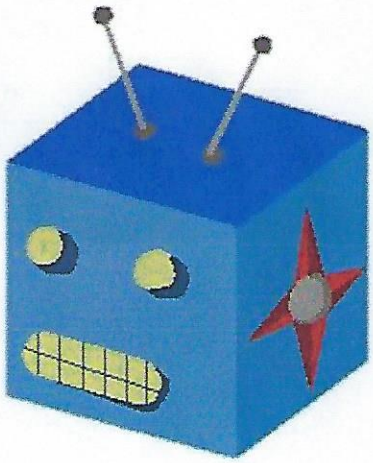


NaviBot Systems

Michael Leverington - Sponsor

Scooter Nowak - Mentor

Diva Ferrell, Logan Behnke,

Peter Aaron Giroux, George Cadel-Munoz,

Benjamin Peterson

Requirements Specifications

Version 1.0

12/12/2019

Overview

This document outlines the requirements for the "Thirty-Gallon Robot, Part Deux" project. These requirements will be categorized into functional, performance and environmental requirements, and will be discussed in detail. After all of the requirements are outlined, we will provide detailed solutions to address them, and lay out a timeline in which these solutions will be implemented.

Accepted as baseline requirements for the project:

_____ _____

(Project Sponsor) *(Team Lead)*

Table of Contents

1. Introduction 2

2. Problem Statement 4

3. Envisioned Solution 6

4. Project Requirements 8

    4.1 Functional Requirements 8

    4.2 Performance requirements 13

    4.3 Environmental Requirements 14

5. Potential Risks 18

6. Project Plan 20

7. Conclusion 22

1. Introduction

Robotics is a branch of engineering and science that includes mechanical engineering, electrical engineering, information engineering, computer science, and other fields. The technology itself serves the purpose of replicating human actions. Robotics have become much more common since the 20th century, doing time-saving tasks that people would rather not do. Not only can robots take on mundane tasks, but also tasks that would be too hazardous for humans, such as rescue missions and exploration expeditions. Needless to say, it is a steadily-growing industry with numerous applications; according to *Million Insights*, the market size for robotics was estimated at $25.68 billion in 2013, and is expected to reach $40.00 billion as early as next year, 2020 (Source: https://www.prnewswire.com/news-releases/industrial-robotics-market-size-is-to-reach-4000-billion-by-2020--million-insights-676656983.html).

One particular area of robotics and automation that has advanced very quickly in recent years is navigation. Many of today's largest companies are establishing self-navigating robots into the lives of everyday people. Companies such as Tesla, Uber, and Google have self-driving cars travelling on the roads alongside humans, some of which are even available to consumers. Even here at NAU, there are robots that deliver food nearly anywhere on campus. The Smithsonian museum has a guided robot, Pepper, with the capability of giving guided tours of its exhibits.

Needless to say, robotics and navigation is a rapidly growing field and will soon become a very key part in all of our lives. Although we interact with robots more and more as time goes on, the barrier for entry still remains high. Very few people possess the knowledge and capability to understand, maintain, or create robots. This lack of knowledge surrounding the topic of robotics can largely be attributed to the costs and lack of opportunities to learn about it.

Our sponsor Dr. Michael E. Leverington is a professor at Northern Arizona University. He has taught multiple computer science courses including Operating Systems and Data Structures, and

also has a masters in educational psychology. As a result of his interest in the latter, he is fascinated by the way students learn, as well as the best way for them to do so. He has always held a passion for robotics and has been following the industry for many years. He believes that the Thirty-Gallon Robot would be the perfect tool to gain interest in computer science and engineering, in addition to being a cost-effective teaching tool.

2. Problem Statement

The Thirty-Gallon Robot is a three year long project to develop a robot with the ability to give guided tours of the Engineering Building. Currently, NaviBot Systems is on the second year of this project.

Last year's team created the robot with basic movement capabilities for Part One, and they named it Robot-Assisted Tours, or R.A.T. for short. R.A.T. is a robot, and the term "R.A.T." and "robot" may be used interchangeably throughout this document. As the project title implies, R.A.T. is contained in a thirty-gallon barrel and can only move via direct human input. This input is received using a hardwired Xbox controller, which means that R.A.T.'s controller must trail five to six feet behind it in order for it to move at all. Although this is currently the only method for R.A.T. to move around, it is equipped with certain technologies such as a Kinect sensor, wheel counters and a Raspberry Pi that will assist in the transition to self-navigation much easier. While last year's team did a great job in actually building the robot, there is still much to be desired.

Dr. Leverington is looking for a robot that can give tours of the Engineering Building, and NaviBot Systems must produce a working navigation module for it. The following is a list of problems which we must overcome in order to implement self-navigation for R.A.T.:

- ***Obstacles.*** R.A.T. must be able to avoid any obstacles throughout the Engineering Building such as walls, people, doorways and stairs.
- ***Self-navigation.*** R.A.T. must be able to navigate to a desired location with minimal human input.
- ***Mapping.*** We must be able to create a map of the Engineering building which is readable by R.A.T.
- ***Localization.*** R.A.T. must be able to determine its location anywhere throughout the Engineering Building.
- ***Taking input.*** R.A.T. must be capable of receiving a location as input.

- ***Graphical User interface.*** We need a GUI that can track R.A.T.'s movement and status in case problems occur.

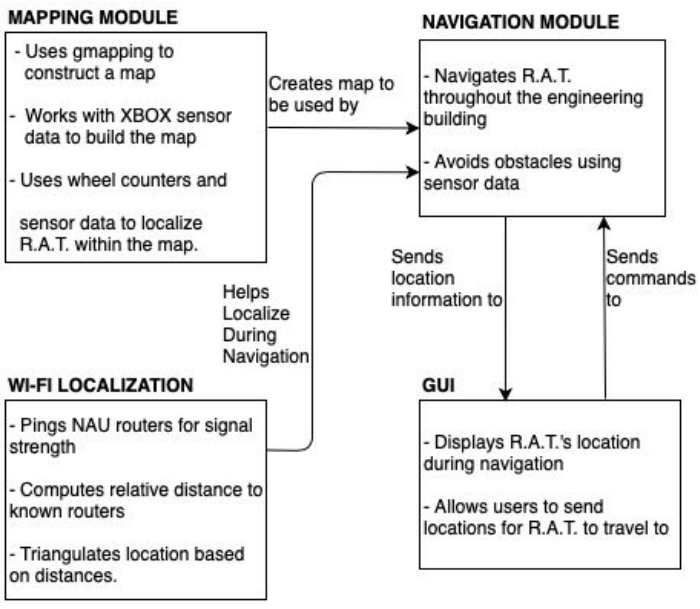
The above points represent the highest level problems in the existing systems. Through research and feasibility testing, we have identified solutions to each of the above problems that we will be looking into in the next section.

3. Solution Vision

In order to fulfill Dr. Leverington's idea of a tour-guiding robot, NaviBot Systems will be building upon last year's work to bring a navigation module to the Thirty-Gallon Robot. This module will require us to provide a way for R.A.T. to obtain a map of the Engineering Building, some kind of localization method to help R.A.T. determine its location, a GUI that tracks R.A.T.'s movements and status, and we need a module for reading and interpreting sensor data. Here are the ways we plan to fulfill these needs:

- ***Mapping with Gmapping Library.*** The gmapping library will allow R.A.T. to roam around Engineering Building and construct its own map using sensor data.
- ***Sensor Data via XBox Kinect.*** R.A.T. will use an Xbox Kinect to acquire its sensor data. The data from the Kinect can be converted to 2D sensor data to use during the 2D map construction.
- ***WiFi Localization.*** R.A.T. will use NAU's WiFi network to ping nearby routers and determine its relative distance using signal strength.
- ***Navigation with Map, Localization and Sensor Data.*** R.A.T. will use all of the above mentioned techniques along with wheel counters to measure distance in order to traverse through the building.
- ***Graphical User Interface.*** We will be developing a GUI using Qt Designer that tracks R.A.T.'s movements and status as it traverses the Engineering Building.

The mapping process is done using the gmapping library which is a SLAM (Simultaneous Localization And Mapping) algorithm. This allows for R.A.T. to roam the building make a map using the sensor data collected as it moves. During this process, R.A.T. will continue roaming and localizing methodically until all areas of the



building have been mapped. This map will be in the form of a PNG image which will be used heavily by the navigation module as seen in *Figure 3.1*.

Figure 3.1 shows the interactions between the different components of our proposed solution.

The sensor data necessary for the mapping and navigation modules will come from an Xbox Kinect. The Kinect takes in RGB-D data which can be converted into a 2D scan in order to construct the 2D map. An added bonus of using the Kinect over something like a LiDAR sensor is that it initially collects 3D data which will better equip us to detect stairs.

The navigation module will be fairly straightforward as long as the components mentioned above are all in place. R.A.T. will use the map along with data gathered from its wheel counters in order to determine it's rough location throughout the building. This may accumulate error as wheels slip and the counter data becomes less accurate, but we can solve this problem using our WiFi localization technique in order to re-localize R.A.T. and make sure it's perceived position matches reality. Lastly, the Kinect sensor will be used to prevent R.A.T. from running into obstacles such as stairs, people, backpacks, etc. Refer to *Figure 3.1* for a more in-depth visualization of how all of these components work together to facilitate navigation.

The GUI will allow a user to view what is happening with R.A.T. in real time. This GUI will be implemented using Qt Designer and it will display data about R.A.T.'s position and status. Along with displaying R.A.T.'s location, the GUI will provide a place for users to give R.A.T. commands for locations to navigate to.

These features will transform R.A.T. from a simple robot that can only move with human input, into a self-navigating robot capable of receiving locations. These changes will also do an excellent job of setting up R.A.T. with all of the tools necessary to give guided tours. Now that our solutions have been covered, let's take a look at all of the requirements that we've identified for the Thirty-Gallon Robot.

4. Project Requirements

For this iteration of R.A.T. functionality developments, we will need to create the software necessary to allow R.A.T. to perform the following actions:

1. Navigate throughout the Engineering Building
2. Take location input and travel to the desired coordinates
3. Interact with the user via a GUI that will display R.A.T.'s status
4. Localize itself within the Engineering Building

First, we look at functional requirements, which will cover the functionality of the software that will be provided at the completion of this project. Next, we cover performance requirements, which discuss how our functions are expected to perform. Finally, we discuss environmental requirements, which are constraints and limitations that we must acknowledge during software development.

4.1 Functional Requirements

We will develop four specific modules that are key in R.A.T.'s operations. These are functional requirements that are essential to developing a self-navigating tour-guiding robot. The navigation module will allow R.A.T. to operate its own hardware in order to traverse throughout the Engineering Building. The mapping module will provide R.A.T. the capabilities to create its own maps by utilizing sensors and collecting visible information obtained while navigating through the Engineering Building. The WiFi localization module will give R.A.T. the ability to self-locate itself within the Engineering Building, regardless of initial startup. Finally, the GUI module will allow communication of commands between R.A.T. and an end user.

4.1.1 R.A.T. Navigation

In order to implement a successful navigation module, the following requirements need to be fulfilled. The requirements are referenced as they appear in Figure 4.1.1.

R.A.T. needs to be able to navigate throughout the engineering building on its own (1.F.1). This is the highest functional requirement for navigation and represents the overall goal for this project.

R.A.T. needs a navigation module capable of detecting obstacles (1.F.1.1). While a map of the engineering building will give R.A.T. a general idea of its surroundings, it won't be able to account for obstacles such as chairs, people, desks or backpacks. While the map should also contain markers for stairs, the navigation module should have additional features in place to further ensure that stairs are noticed by R.A.T. before a collision occurs.

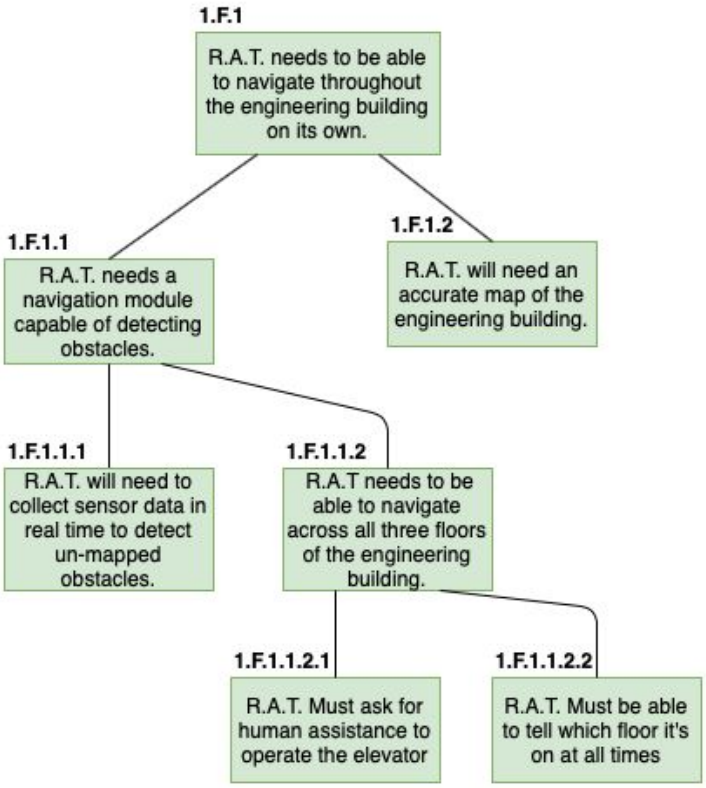


Figure 4.1.1 illustrates the top-down structure of the navigation functional requirements

R.A.T. will need to collect sensor data in real time to detect un-mapped obstacles (1.F.1.1.1). The navigation module should use sensor data to path around such obstacles as well as assist in avoiding more dangerous things such as stairs. This will ensure R.A.T.'s ability to traverse the building during busier times when the hallways aren't clear as the map would imply.

R.A.T. needs to be able to navigate across all three floors of the engineering building (1.F.1.1.2). Since the end goal of the project is for R.A.T. to give tours of the engineering building, it is important that our navigation module allows R.A.T. to access all three floors.

R.A.T. must ask for human assistance to operate the elevator (1.F.1.1.2.1). In order for R.A.T. to fulfill requirement 1.F.1.1.2, it will need to stop at an elevator door and ask for surrounding humans to take him to the desired floor.

R.A.T. must be able to tell which floor he is on at all times (1.F.1.1.2.2). With R.A.T. relying on bystanders to help it traverse amongst the different floors of the building, there is a possibility that they may take him to a floor that he didn't ask for. To cover this case, R.A.T. should be able to use localization techniques to ensure that he always knows which floor he is on. The requirements for the localization module will be discussed in detail in section 4.1.3.

R.A.T. will need an accurate map of the engineering building (1.F.1.2). A map is a crucial component in the navigation process. This map will help guide R.A.T. throughout the engineering building and is absolutely necessary to have for the robot to be able to travel to a given location. The requirements for the mapping module will be discussed in detail in section 4.1.2.

4.1.2 R.A.T. Mapping

In order to implement a successful mapping module, the following requirements need to be fulfilled. The requirements are referenced as they appear in Figure 4.1.2.

R.A.T. needs a map of the Engineering building (2.F.1). This is the highest level functional requirement for the mapping module. An accurate and map is absolutely necessary for the navigation module. The following requirements will outline the important aspects of a quality mapping module.

The map should label key points and dangerous obstacles (2.F.1.1). The map should have labels for key points such as rooms and elevators. This will help to better facilitate the goal of having R.A.T. give guided tours of the engineering building. The map should also have labels for

dangerous obstacles such as stairs. Having the stairs marked on the map will allow R.A.T. to avoid the stairs as a possible path all together.

The map should be easily storable and readable (2.F.1.2). The maps of the different floors should be easy to store retrieve so that they can be reused each time R.A.T. is booted up to navigate. The maps should also, of course, be readable by R.A.T. so that it can make full use of the data stored within the map.

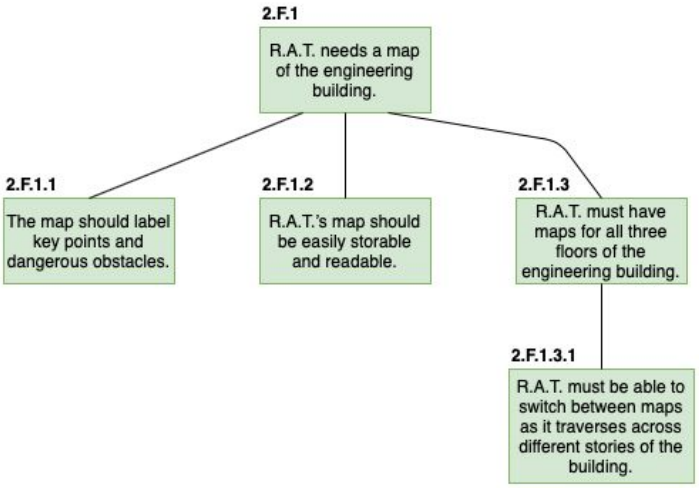


Figure 4.1.2 illustrates the top-down structure of the mapping functional requirements

R.A.T. must have maps for all three floors of the engineering building (2.F.1.3). Since we want R.A.T. to be able to navigate across all three floors, it is important that R.A.T. has a map for each of them.

R.A.T. must be able to switch between maps as is traverses across different stories of the building. (2.F.1.3.1). Since each floor will have its own separate map, we will need some way of switching the map that R.A.T. is reading data from to the relevant floor as R.A.T. moves on to a different level of the building.

4.1.3 R.A.T. WiFi Localization

In order to implement a successful WiFi localization module, the following requirements must be filled. The following functions will be referenced as they appear in Figure 4.1.3.

R.A.T. needs to be able to self localize itself within the engineering building (3.F.1). This is the most important aspect of the WiFi localization module, as it will allow R.A.T. to find its location regardless of where the robot boots in the engineering building.

R.A.T. will need to scan for nearby broadcasting WiFi signals (3.F.1.1). This is necessary for R.A.T. to perform so that it may collect locations of nearby routers. Obtaining the router information is crucial for later distance calculations.

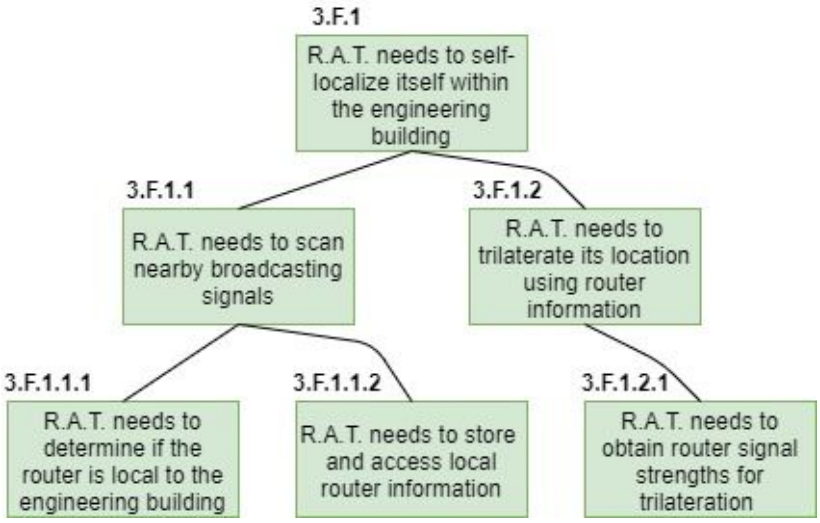


Figure 4.1.3 illustrates the top-down structure of the WiFi localization functional requirements

R.A.T. will need to determine if the broadcasting WiFi signals are local to the engineering building (3.F.1.1.1). Since many devices such as cellphones and printers can provide wireless connectivity options (i.e. hotspots, shared connections), R.A.T. should only perform calculations using local routers. This is for the sake of consistent data, as routers are permanently established within the building.

R.A.T. will need to store router information and access the information as necessary (3.F.1.1.2). This will ensure that the routers scanned in the area allow R.A.T. to compare what routers have been stored in memory and access this information for later localization.

R.A.T. will need to trilaterate its location using router information (3.F.1.2). Trilateration will allow R.A.T. to calculate its approximate location in the engineering building by using signal strengths and determining the distance from each router.

R.A.T. needs to obtain router signal strengths for trilateration (3.F.1.2.1). The signal strengths are important for R.A.T., aiding in calculating distance from each router, as each router is physically anchored in specific areas throughout the engineering building.

4.1.4 R.A.T. GUI

The GUI is meant to be the middleman between operator and R.A.T. In other words, it is how a user is able to see R.A.T.'s movement in real time, as well as give it commands. Said commands should be about where in the Engineering Building R.A.T. needs to go to. The interface should be able to handle sending R.A.T. to specific rooms and floors, while also displaying the location R.A.T. is currently at using a map and a pin that shows its position, and perhaps even the location of the routers R.A.T. would be using to reposition itself.

One of our stretch goals includes allowing R.A.T. to send specific status updates to the GUI. This would ensure that an operator knows when R.A.T. is in critical condition, such as suffering from low power, a hardware malfunction, or if it happened to fall down the stairs. In the case that it's on an elevator, the GUI would then display that its enroute or if it needs help pressing the buttons to get to its designated location. Additionally, a second stretch goal is implementing an abort button to the GUI. In the event that there is a disconnect between R.A.T. and the user interface, and the operator is unsure of R.A.T.'s location, it would be resourceful to have a command that shuts down all of R.A.T.'s movement. Or, if it was observed that R.A.T. was heading towards the stairs with no intentions of avoiding a certain fall, effectively stopping the robot in its tracks would be necessary.

4.2 Performance Requirements

Next, we will address the performance requirements. In this section, we will lay out metrics for which we expect our system to perform. The following performance requirements are enumerated as follows:

- 1.P.1 = R.A.T. needs to determine which floor it is on within 60 seconds
- 1.P.2 = R.A.T. needs to stop within a 50 cm^2 area in front of a doorway
- 2.P.3 = R.A.T. must build a map that is accurate to the real world with errors no larger than 10 cm
- 2.P.4 = R.A.T. needs to generate a map of the Engineering Building in under one hour
- 3.P.5 = R.A.T. GUI should update the operator approximately every twenty seconds regarding the data it is retrieving
- 3.P.6 = GUI should start moving within twenty seconds of receiving a command about where to go
- 3.P.7 = R.A.T. should report to the GUI every twenty seconds its current condition (stretch goal)
- 4.P.8 = R.A.T. needs to obtain router information within 60 seconds
- 4.P.9 = R.A.T. should be able to match its estimated location on a map within 20 seconds
- 4.P.10 = R.A.T. should orient itself accordingly within 10 seconds, meaning R.A.T. will know how it currently sits during the boot phase

We will address these by referring to the top level requirements.

4.2.1 R.A.T. Navigation Performance

When R.A.T. switches floors during navigation, it should be able to determine which floor it resides on within 60 seconds (1.P.1). This pertains directly to the functional requirement, "R.A.T. must be able to tell which floor he is on at all times, " (1.F.1.1.2.2). The longer the delay, the longer a user will have to wait for R.A.T. to function accordingly (the stretch goal is to develop a robot that gives tours, and keeping those accompanying the robot from waiting an excessive amount of time).

R.A.T. should also accurately lead to a specific location, and with careful considerations of the actual size of R.A.T., we believe that R.A.T. being able to stop within a 50 cm^2 area in front of a doorway is a reasonable performance requirement (1.P.2).

4.2.2 R.A.T. Mapping Performance

While the mapping process should ideally only need to be done once, it is still important that the process is time efficient. This will be most beneficial to the person/people who are watching R.A.T. during the mapping process in order to ensure things go smoothly. Therefore, we can establish the performance requirement that R.A.T. should be able to generate a map of the Engineering Building in under one hour (2.P.4). This performance requirement pertains to the functional requirement, "R.A.T. needs a map of the Engineering building," (2.F.1). This will allow R.A.T. to take enough sensor data to build an accurate map without wasting the time of those in charge of facilitating the process.

Another important requirement for a good map is to ensure that it is accurate to it's real world counterpart. From this, we can establish the performance requirement that R.A.T. must build a map that is accurate to the real world with errors no larger than 10 cm (2.P.3). This performance requirement follows the parent function requirement of, "R.A.T. will need an accurate map of the engineering building," (1.F.1.2). This will ensure that R.A.T. can navigate throughout the building using the map without risk running into an obstacle that wasn't represented accurately on the map.

4.2.3 R.A.T. GUI Performance

For the GUI performance, it should update the operator approximately every twenty seconds regarding the data it is retrieving. Specifically, it should show up on the map every twenty seconds minimum where it is currently moving to and stationed in the building (3.P.5). It should also start moving within twenty seconds of receiving a command from the interface about where to go (3.P.6). We do not want the operator to be confused as to R.A.T.'s status, which would result from a disconnect between its actual location and the reported location. For the stretch goal of showing status updates, it should also report to the GUI every twenty seconds its current condition, so that the operator can catch issues in a timely manner, whether they are big or small (3.P.7).

4.2.4 R.A.T. WiFi Localization Performance

During daily operations with R.A.T., localization should not need to occur very often. Prime times for self-localization should only occur while R.A.T. boots from a cold start. A cold start is when R.A.T. is set in some position within the Engineering Building and is switched on, initiating the self-localization phase. Since there a number of operations are required to run only once, we would like to keep self-localization operations under 60 seconds after initial boot (4.P.8). During this time, R.A.T. will be pinging local routers, obtaining router information and estimating its location. This time frame is subject to fluctuate in time, as router operation is affected by network congestion. Any other time self-localization should occur is when connection to the network is lost (i.e. system reboots, wireless connectivity issues).

For calculating its position with obtained router information, R.A.T. should be able to match its estimated location on a map within 20 seconds (4.P.9), as no network responses are required for this portion of the boot operation. R.A.T. is running an algorithm to trilaterate its position, and a simple operation would not need much time to compute a number and pinpoint the coordinates on the generated map. Finally, the robot should orient itself accordingly within 10 seconds, meaning R.A.T. will know how it currently sits during the boot phase (4.P.10). This will ensure that R.A.T. navigates appropriately once it localizes to the building location and its map.

4.3 Environmental Requirements

Since R.A.T. is a continuation of a previous capstone group's project, we are met with very specific and unavoidable constraints. The first is that we must use ROS for device control and package management. It currently receives input from the Xbox controller, and the Raspberry Pi (general-purpose computer, usually with a Linux operating system), which is another constraint. The Raspberry Pi sends commands to the Arduino (a simple computer capable of running one program over and over again), another built-in hardware constraint. The Arduino also controls the motor drives, which is a necessary feature of R.A.T.

Because we are using ROS, that puts a constraint on the programming language we are able to use, as ROS is written in the programming languages C++, Python, and Lisp. Also, since the prior code that has already been written for R.A.T. was in C++, it was fairly reasonable to assume we would have to use C++ as well. The Xbox Kinect sensor was also somewhat of a constraint since it came with the project already built-in; however, we had the option to use a different sensor at the risk of it not working as well for navigation and also not being able to integrate with the hardware. Thus, we decided to keep it as a tool, especially since we discovered its ability to do stair detection.

As for our navigation packages, a constraint is that we do have to write a wi-fi localization package ourselves, since there weren't any current packages that would have what we need. However, there is an open source multi-map navigation package that we can modify in order to easily switch between maps of the building, and has an elevator script that can be modified to allow us to control the elevator. In general, the decision to attempt wi-fi localization of using routers within the building came from a constraint that the client placed: we are not allowed to place any sort of objects/tape/etc. around the Engineering Building that would help R.A.T. determine where it is.

For mapping, we were constrained by two options: creating a map of R.A.T. ourselves, or letting R.A.T. create its own map by roaming around the building. Once we decided on the latter choice, we were constrained by gmapping and the SLAM algorithm since it is one of the few algorithms that will work well with the Kinect sensor provided to us. SLAM is also supported and very well documented in ROS via the gmapping library.

5. Potential Risks

There are multiple issues that may come with programming a navigation system for our Thirty-Gallon Robot, Part Deux project. R.A.T. runs the risk of becoming a hazard to both itself as well as to the people in its environment if it cannot avoid stairs or obstacles. Its severity would be very high for the former, and moderately high for the latter, with low likelihood for both. The challenge for both would be accurately getting data from the Kinect sensor, which should be feasible with gmapping, our navigation package, and the chosen sensor.

The next risk refers to possibly losing R.A.T., and/or R.A.T. happens to malfunction. We determined that this also has a low likelihood of occurring, with the challenge of ensuring minimal disconnects between the GUI and R.A.T., and a potential abort command. This would be feasible with adjustments to acceleration and our navigation package. The last risk we have in mind is R.A.T.'s potential to ping unofficial routers rather than the ones it needs to self-locate. The likelihood of it actually coming to pass with these changes made is low to medium. The challenge is wifi localization using routers and differentiating router IDs, which should be feasible with the wifi localization packages we're writing and using. *Table 5* has a summary of the above.

Risks	Severity	Challenges	Feasibility	Likelihood
R.A.T. doesn't detect stairs	Very high	Accurately getting data from the Kinect	Feasible with gmapping, our navigation package, and the chosen sensor	Low
R.A.T. doesn't avoid obstacles/people	Moderately high	Accurately getting data from the Kinect sensor	Feasible with gmapping and our navigation package	Low
Losing R.A.T. and/or R.A.T.	Moderate to very high	Ensuring minimal	Feasible with adjustments to	Low

malfunctions		disconnects between the GUI and R.A.T., and a potential abort command	acceleration and our navigation package	
R.A.T. pings unofficial routers	Very high	Wifi localization using routers	Feasible with wifi localization package	Low/Medium

Table 5. Displaying risks, severity, challenges, and feasibility.

6. Project Plan

In order to prepare for potential challenges and risks during development, our team has drafted a plan to keep our development well timed and on track. It is split into three parts: early development and technological study, mid development (building off of working prototypes and alpha testing), and late development (finalizing and beta-testing). Early development will take place before the end of the semester on December 15th. Mid development will begin after the semester ends on December 15th and will continue until mid-February. Between mid-January and mid-February there will be some overlap with late development as alpha-versions are finished and beta-versions are begun. Mid-February will mark the official start of late development where beta-versions will be created and tested before the end of April. This will leave the beginning of May to the end of the semester to finish testing and any final adjustments on the project, as well as any possible extra development time that may be needed.

6.1 Early Development

Our early development consists of testing and prototyping our four functional requirements: navigation, mapping, localization, and GUI. In the below figure we have these requirements in their testing and prototyping phases. This includes a short phase to ensure that RAT functions as it was supposed to at the end of last semester, which we have shown to be fully functional to last semester's standards.

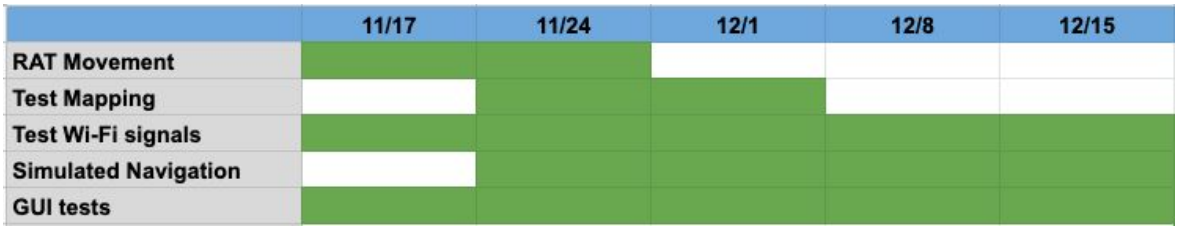


Figure 6.1 Early development Gantt chart

6.2 Mid Development

Mid development will focus on moving from learning technologies, and testing prototypes to building functional alpha versions of our four main functional requirements. It will start with

mapping, because the mapping package is central to the navigation functioning properly. The mapping package, GUI (with only design, not functionality), and localization will all begin December 22. Once the mapping package is finished in its alpha form we will begin development of the single floor navigation package based on data from the mapping package. Since the navigation package is essential to preventing many risks, it will be a full team effort for the beginning of the planning and development. Once a strong baseline for navigation has been established, elevator functionality and GUI functionality development will begin.

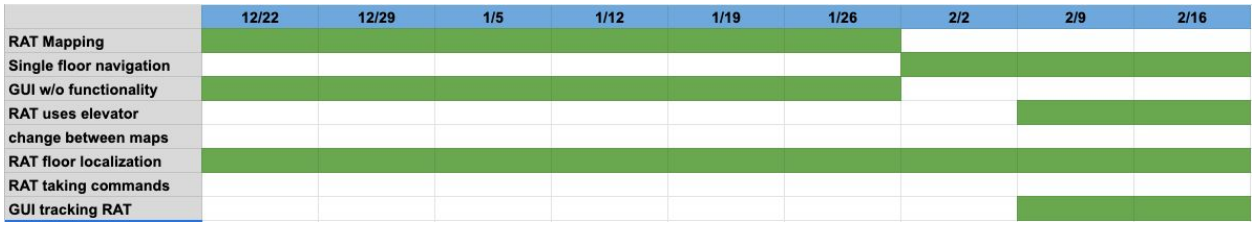


Figure 6.2 Mid development Gantt chart

6.3 Late Development

In the late development, we will expand on alpha versions of each package with beta functionality. This will include taking commands remotely from the GUI, tracking from the GUI, multi-floor navigation, and localization on boot. The below chart leaves an extra week at the end of April in the event that extra time is needed in any development step. Once all packages are finished we will do final testing and final adjustments leading up to the final project display in May.

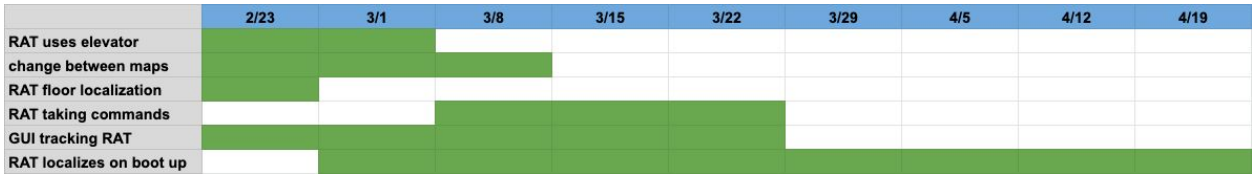


Table 6.3 Late development Gantt chart

7. Conclusion

Robotics is a growing field with high teaching potential, especially in higher learning. However, the field of robotics has a high bar of entry and can be expensive. Dr. Michael Leverington has commissioned our team, Navibot Systems, to build a fully functioning and cost-effective robot for teaching and recruitment. Robot Assisted Tours, or R.A.T., will take guests on tours of the engineering building using systems that our team will develop. Navibot Systems aims to create four main functional packages to bring R.A.T. to the next level of functionality. These packages will be a mapping package, navigation package, localization package, and GUI package. Our team has identified potential risks and challenges that may impede our ability to reach our goals, and to combat these Navibot Systems has drafted a project plan to keep our team on track with developing around these challenges and risks. We are team Navibot Systems, and we are confident that our team will overcome all challenges to create a fully functional robot that fills our client's needs.