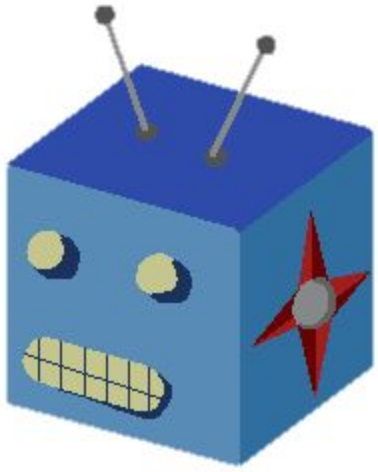

NaviBot Systems

Dr. Michael Leverington - Sponsor
Scooter Nowak - Mentor
Diva Ferrell, Logan Behnke,
Peter Aaron Giroux, George Cadel-Munoz,
Benjamin Peterson

Technological Feasibility Analysis

11/04/2019

Overview

This document outlines the technological feasibility of the “Thirty Gallon Robot, Part Deux” project. This will include major technical challenges that may arise during the production process, as well as some possible tools that will be utilized to overcome such challenges.

1. Introduction

We are NaviBot Systems, the capstone team responsible for the "Thirty Gallon Robot, Part Deux" project, also known as Robot Assisted Tours (R.A.T). Our team is comprised of five members:

- Diva Ferrell, *Team Lead*
- Benjamin Peterson, *Customer Communicator*
- Aaron Giroux, *Team Recorder*
- Logan Behnke, *Release Manager*
- George Cadel-Munoz, *Team Architect*

Together, under the mentorship of Scooter, we will be working on further developments of the "Thirty Gallon Robot," sponsored by Dr. Michael Leverington. Dr. Leverington has proposed an affordable alternative to incorporating robotics into a learning environment. Specifically, Dr. Leverington would like to create a robot that, once complete, will be capable of providing self-guided tours throughout the Engineering Building located at Northern Arizona University.

This is the second year of work on the Thirty Gallon Robot project. While last year's team worked hard on getting the robot built and moving, there are still several shortcomings that we would like to improve, such as:

- R.A.T. can only move with direct human input.
- R.A.T. requires a connected xbox controller in order to receive input.
- R.A.T. is not capable of reading or building maps.
- R.A.T. is not capable of navigating a building on its own.
- R.A.T. is difficult for an average person to start up and get it running.

Our client's current issue is the robot's lack of self-navigational components. These components include software that will allow R.A.T. to operate without a human driver, the ability to read map inputs, and the ability to self-locate regardless of boot location within the Engineering Building.

As a solution, NaviBot Systems would like to provide a navigational module, along with mapping and location modules, that have the ability to interact with the robot through a web-based GUI. In this document, NaviBot Systems will cover technological challenges we may encounter during production, an analysis for each design challenge, a documented approach to overcome listed challenge, and integrations of solutions into pre-existing systems.

2. Technological Challenges

For the future success of R.A.T., NaviBot Systems will have to address challenges that currently exist in the current version of the "Thirty Gallon Robot." There are three important tasks that must be analyzed to create a solution for the proposed minimum viable product at the completion of Capstone 2019-2020:

- Navigation: R.A.T. will require a navigational module. This module will provide R.A.T. with the ability to use its currently installed components (i.e. such as sensors, wheels, and motors) to drive while also allowing the robot the recognize obstacles, such as walls, staircases, objects, and people.
- Mapping: R.A.T. will need to know how to read a given map. This module will allow R.A.T. to understand physical constraints within a given physical area, and the ability to understand where special landmarks, such as class rooms and offices, are located in the building.
- GUI: R.A.T. will need the ability to communicate with end users. By providing the robot with a GUI interface, end users will have the ability to view the status of R.A.T., and get live updates of important events that occur during the robot's tours.

In addition to the main requirements listed above, NaviBot systems will need to address higher level challenges that may arise during development. The main challenges which we will be focusing on include:

- We will need to create a meaningful map that the robot may utilize.
- We will need to research hardware and software capabilities to accommodate the use of Wi-Fi localization.

- We will need adequate sensors that provide data for the robot's navigation module and map-making process.
- We will need a navigational library and/or algorithm with capabilities to support multi-floor navigation.
- We will need a framework in which to build a graphical user interface that can receive data from the robot and display the information to the user in real time.

With these technological challenges in mind, an analysis of each possible design challenge must be made to create an appropriate design for the final iteration of the "Thirty Gallon Robot."

3. Technological Analysis

An analysis on our technological challenges will provide NaviBot Systems with a deeper understanding of current issues that we will have to face during the development process. After analysis, we will consider a possible solution to each issue, as well as alternatives with their pros and cons. Next, we will conduct research on each likely solution. Finally, we will provide summations of each decision, along with technology demonstrations that will be developed in the future.

3.1 Robot Mapping

In order for R.A.T. to navigate around the engineering building, we will somehow need to acquire a map which is readable by the robot. There are two main ways in which we could address this. The first is that the robot could roll around the engineering building (either on its own or with human instruction) and take in sensor data in order to build a map. The other option is for us to construct the map manually. This would require us to take careful measurements of the engineering building and would likely require us to develop a simple map making tool. While the map making tool would probably not be very complex, it would still take up quite a lot of time and resources that might be better spent in other areas. In the interest of time and future ease of use, we will focus on exploring the options involving R.A.T. creating its own map. In order to

determine our best solution to the challenge of robot mapping, we will be ranking our options in the following categories:

- Robot Controlled: Does the mapping method rely on the robot to control itself as opposed to requiring human input?
- ROS Libraries: Does the mapping method have ROS libraries which are maintained, well documented and easy to use?
- Hardware Limitations: Does the mapping method work with the current hardware limitations of R.A.T.?
- Stair Avoidance: Does the mapping method grant the ability for RAT to avoid a catastrophic stair collision?

Now that our criteria have all been laid out, let us begin by looking into what options we are considering to solve the robot mapping problem.

3.1.1 Options for Robot Mapping

One particular method to tackle the robot mapping problem is using an algorithm called SLAM (Simultaneous Localization and Mapping). SLAM allows a robot to map out a room on its own using incoming sensor data to detect obstacles. While the robot is mapping, it is simultaneously searching for key points on the map in order to determine its location relative to its surroundings. The robot will explore all paths available to it in the map and eventually will output a file containing the map. One library that we are considering using for this is the gmapping library for ROS.

Gmapping implements the SLAM algorithm by having the robot move around the map and take in sensor data to build the map as new areas are found. During this process, the robot also self-localizes on the map. Once the robot is finished mapping, gmapping can save the map in a pgm (portable gray map) file to be used for navigation in the future.

The gmapping library already aligns with our current hardware limitations. Currently, all of the code for R.A.T. uses ROS and is mostly done in C++, R.A.T. also has wheel counters to measure

distance and a kinect to take in some kind of sensor data. While the kinect is not the most ideal for taking in sensor data there are plenty of documented cases of building a map using gmapping with a kinect. This will only be made more viable as we look into more high-tech sensors to add on to our robot. Since there are already several documented projects using gmapping with the same or similar hardware as R.A.T., the gmapping library will receive a score of ***five out of five*** for the "hardware limitations" section (Table 3.1).

Gmapping would prove very useful as a way to build reliable maps with little human input needed. Having the robot drive around on its own during the mapping phase would be a great way to ease us into the robot navigation side of the project. This may ultimately prove to be more challenging than a human controlled alternative, but it will serve as a great learning experience which should help us further on in the project. While gmapping allows for robot controlled mapping, it does require some human control in the beginning of the mapping process in order to get things going. Therefore, gmapping will receive a score of 4 out of 5 in the "robot controlled" section (Table 3.1).

Another major benefit of this approach is that the SLAM algorithm, and gmapping in particular are very well documented. SLAM seems to be the most common way for robot map building, and gmapping seems to be the most popular implementation of this in ROS. There are several documented projects using gmapping with hardware which is very similar to what R.A.T. is using. Additionally, there are pages of in depth tutorials of gmapping on the ROS website to help get things up and running. Therefore, gmapping will receive a score of ***five out of five*** for the "ROS Libraries" section (Table 3.1).

The layout of the multi-floor layout of the engineering building will provide us with some additional challenges during the mapping portion of the project. One of the things that we desperately need to avoid when the robot is moving around the building in any fashion is the stairs. We are not sure if gmapping, or a SLAM algorithm in general would have the capabilities of detecting stairs. Seeing as sensors typically are looking to detect and map walls and there are

no walls in the beginning of the stairs, this would likely pose a potentially big issue as a failure to detect stairs could lead to a catastrophic failure. For this reason, gmapping will receive a score of ***one out of five*** for the "stair avoidance" section (Table 3.1).

Another method in which we could acquire a map for our robot is to drive him around the engineering building manually while it collects sensor data. It seems that this method has far less support in the ROS libraries. It has proven extremely difficult to find any instance of this method being implemented within a ROS project, which is a definite cause for concern. Since there does not seem to be much support for this method within ROS, we would likely have to write a fair bit of code on our own in order to implement this method. This could prove to be a risky time sink which could lead to a big setback in the project. Therefore, the human controlled mapping will receive a score of ***zero out of five*** for the "ROS Libraries" section (Table 3.1).

While the lack of support and documentation in ROS could prove to be a problem, having the mapping process controlled by humans could be a viable solution to the stairs problem presented earlier. If a human is controlling the robot, they will have the capability to detect the stairs and thus avoid potential catastrophe. In either case, this would not stop the robot from falling down the stairs when using the map to navigate, since that would just be an unmapped area. We would likely have to draw our own line surrounding the stairs to represent a wall and thus making the stairs unreachable whether we use SLAM or we control the robot on our own. For the purpose of creating a map, human controlled mapping will receive a score of ***five out of five*** for the "Stair Avoidance" section (Table 3.1).

R.A.T.'s hardware limitations should prove no issue for a human controlled mapping approach. As long as R.A.T. is able to take in sensor data (covered by the kinect) and has the ability to move via human input (covered by the motor and xbox controller), a human controlled approach to mapping has all of the tools needed to be a viable option. Therefore, the human controlled mapping will receive a score of ***five out of five*** for the "Hardware Limitations" section (Table 3.1).

Lastly, it should be fairly obvious by now that the human controlled approach to mapping does not rely on the robot to navigate on its own. If we were to implement this approach, we would need to follow the robot around throughout the whole engineering building in order to control it via xbox controller while the map is being built. Therefore, the human controlled mapping will receive a score of ***zero out of five*** for the "Robot Controlled" section (Table 3.1).

3.1.2 Proposed Solution for Mapping

	GMapping	Human Controlled
Robot Controlled	4	0
ROS Libraries	5	0
Hardware Limitations	5	5
Stair Avoidance	1	5
Overall Score	15	10

Table 3.1 ranks SLAM vs. a human controlled mapping process in 4 different categories on a scale from 0 (worst) to 5 (best).

Based on the metrics shown in *Table 3.1*, our team will be using the gmapping implementation of the SLAM algorithm to build maps for our robot. The SLAM algorithm will work well with the kinect sensor that is provided to us and will work even better with the addition of a more technically proficient sensor of our choosing. It is also supported and very well documented in ROS via the gmapping library. Having the robot build its own map would serve as an excellent introduction for us as a team to robot navigation and localization techniques. While it lacks the ability to detect stairs during its mapping process, we feel that the ROS support and documentation will provide us with a much easier time implementing a solution to the mapping problem.

3.1.3 Future Feasibility Testing of GMapping

We have several methods of testing gmapping's feasibility going into the future. To begin our testing, ROS has built-in simulating software which we can use to test the library's map making capabilities. While the simulation will help further prove the gmapping library's ability to build maps, it may not hold entirely accurate to the problems faced by a real robot in a physical setting. To accomplish this, we have several smaller robots provided by our sponsor for testing purposes.

These robots can act as crash-test dummies to test this mapping software without the risk of catastrophe at unexpected obstacles such as stairs.

3.2 Wi-Fi Localization

Most localization systems, e.g. phones or robots, use both GPS and Wi-Fi signal to help determine the exact location of the device. However, unlike most systems our robot will not need GPS capability to localize itself properly. There are three reasons for this. First, our robot will not be changing buildings and will only need to know its position relative to the Engineering building. Second, since our robot will be inside a building that has poor phone signal, the GPS signal would likely be poor. Third, our budget does not allow for the purchase of GPS technology. For these reasons, a robust and precise Wi-Fi localization system is necessary for R.A.T. to define its position in the Engineering building correctly. In order to do this we will need accurate physical locations of Wi-Fi routers identified on RAT's internal map, and a way for R.A.T. to interpret Wi-Fi signal into location data.

There are two ways of adding the physical locations of the Wi-Fi routers onto RAT's map, add the routers manually with human input or to have R.A.T. add them based on signal strength. As discussed in section 3.1, having R.A.T. do the mapping via the SLAM algorithm is better than mapping via human input. Similarly, having R.A.T. add Wi-Fi routers to its maps will make R.A.T. much more modular and adaptable than hard coding them.

3.2.1 Options for Wi-Fi Localization

There are a number of proposed solutions to indoor localization using Wi-Fi. Some of them discuss using AMCL (Adaptive Monte Carlo Localization) in conjunction with Wi-Fi signal strength data (Sources:
http://wiki.ros.org/indoor_localization, https://github.com/cra-ros-pkg/robot_localization,
https://github.com/RMiyagusuku/ros-wifi-localization,
https://github.com/subpos/subpos_receiver).

Plenty of these proposed solutions are more than two years old and were developed or proposed for older versions of ROS. One or two frameworks (not listed) required proprietary IR modules to be set up in the building. This is far less than ideal. Thus, the indoor-localization ROS framework and package will be our best bet for accurate and proper localization. While this package is made specifically for UWB sensor localization, it can be configured with relative ease compared to the other possible frameworks to work with Wi-Fi localization.

3.2.2 Wi-Fi Localization Framework Scores

The age of a package (being determined by the last date package code was edited) is important, because more recent packages are more likely to have continuing support from their developers. For this reason, the Indoor Localization and Robot Localization packages have both received a 5 out of 5 for their current implementations and up-to-date support (Table 3.2). The subPOS system was last edited 4 years ago, and thus is probably not outdated but may not be actively supported by developers should our team have questions. Therefore, the subPOS system has received a score of 3 (Table 3.2). The ROS_wifi localization package was last changed 1 year ago, and while not currently being worked on will probably still receive support from its developers thus giving it a score of 4 (Table 3.2).

	Indoor Localization	ROS_Wifi Localization	Robot Localization	subPos
Package Age	5	4	5	3
OS Developed For	Kinetic(5)	Indigo(3)	Unknown(0)	Arduinos(0)
Documentation	5	3	4	3
Modularity	5	3	2	5
Ease of Setup	0	2	3	3
Overall Score	20	15	14	17

Table 3.2 Scores the wi-fi localization packages in five categories from 0 (worst) to 5 (best)

Another important aspect of determining the package our team will use is the Operating System that a specific package was developed for. Packages developed for the Kinetic version of ROS receive a 5, while packages developed for older versions of ROS will receive a score based on how different from Kinetic that operating system is. The subPOS package is built for arduinos, and in order to use it our team would need to program arduinos with it and them implement those

into our system which could be complex and time consuming. For this reason the subPOS package received a 2 out of 5 (Table 3.2). The Robot Localization package does not have any documentation on which operating system it was developed for and thus receives a 0 out of 5 for being a possible wild-card during development (Table 3.2).

The other scoring categories are Documentation, Modularity, and Ease of Setup. Packages with more complete documentation receive higher scores in the Documentation category. This includes the Indoor Localization package's tutorials on usage (a documentation feature that no other package in consideration has). Packages that can easily be set up to work in a variety of environments with a variety of technologies receive higher scores in the Modularity category. For this section, the focus was on whether a package had a lot of hard-coded values or not. Since the nature of our team's work is fairly cutting edge, our team needs packages that will allow changes without the need for editing package code. For this reason both the Indoor Localization and subPOS packages received 5 out of 5 for this category (Table 3.2). The final scoring category is Ease of Setup. This category is based on how simple or complex a package is, and how straight-forward its installation and implementation process is. For example, the Indoor Localization package received a 0 out of 5 due to its complexity and the difficulty that may occur in setting the package up to work with RAT's systems (Table 3.2).

3.2.3 Chosen Localization Package

Based on the scores in the above figure, the Indoor Localization package will be our best option for Wi-Fi based localization despite its current usage of UWB technology. The systems in place for storage of UWB data will be fairly easy to modulate to work with our Wi-Fi systems instead as the current data structures in place for storage will carry all necessary Wi-Fi information. In addition to this, writing a system to base Wi-Fi router location for localization and mapping use will be necessary regardless of the package used due to the nature of our specific project, and can even be adapted from the UWB framework with relative ease.

To demonstrate the ability of Wi-Fi-based position a number of free programs exist for mobile applications, Windows OS, and Mac OS. These programs can be used to show how distance from a router can be measured based on signal strength and ping time.

3.3 Sensors

For navigation and mapping, we need sensors to be able to read in data from the surrounding environment to make decisions. There are many types of sensors that can be used for navigation and mapping. The two main types of sensors are lidar (Light Detection and Ranging) and rgb-d(red green blue depth). The criteria we used to compare these two types of sensors were ability to detect stairs, ROS supported, SLAM supported, and cost. The three sensors we are looking into are the slamtec RPLIDAR A1, the Microsoft Kinect, and the Intel RealSense.

3.3.1 RPLIDAR

The slamtec RPLIDAR A1 has ROS support packages and supports SLAM navigation.we give this sensor a 5 out of 5 for ROS support and SLAM support because it is well documented and there are tutorials on how to get started with this sensor. Since it is a lidar sensor it would not be able to detect stairs because of the limitations of lidar. The cost of this module is about $160 for that we gave it a score of 3 out of 5 for cost because it is not the most expensive of our sensors we are looking into.

3.3.2 Microsoft Kinect

The Microsoft Kinect is a cheap rgb-d camera. It takes a picture and a depth image to determine if there are obstacles in the area. There are ROS packages that will allow us to take the 3D data from the depth image and convert it to a 2D scan using the closest points in each column. Using those converted data we can use it in our mapping and SLAM algorithms. The Kinect can be used to detect stairs using cliff detection. Since the Kinect can detect stairs we gave it a 5 out of 5 for stair detection. Since we already have this sensor the cost is nothing for this we gave it a 5 out of 5.

3.3.3 Intel RealSense

The last sensor that we looked into was the Intel RealSense rgb-d camera. This is produced by Intel and has ROS support built into it. It is used to just like the Kinect but it was developed specifically for use with SLAM navigation. We can use it to detect stairs from the 3D data. The cost of this sensor is from $150-300 to get both tracking modules.

3.3.4 Chosen Sensor

Out of these options we chose to keep using the Kinect sensor as our main sensor. We chose it because the cost is the lowest and we are still able to move around the building with the data it can provide. It has the ability to detect stairs very well and it has well documented ROS packages with tutorials that will help us get started.

	Detect Stairs	ROS Support	SLAM Support	Cost
Salmtec RIPLIDAR A1	1	5	5	3
Microsoft Kinect	5	5	5	5
Intel RealSense	5	5	5	1

3.4 Multi-floor Navigation

Multi-floor navigation is required to be able to give a full tour of the engineering building. In order for this to be achieved the robot will need to be able to switch between maps of the different floors, navigate to transition points between the maps, control elevator commands, and avoid obstacles that may be in the robots way. There are three main ways we can approach this problem. The first way is to use the Navigation package from ROS. the second way is to add on to and update the Multi_map_navigation package. The final approach is to create a ROS package from scratch that incorporates all of these features that we require to guide the robot.

3.4.1 ROS Navigation

This first approach using the Navigation package from ROS will allow us to move the R.A.T from the start with very little development on our part. We will be able to set up goals for R.A.T to navigate and avoid obstacles. This package does not allow us to switch between maps while the robot is navigating around the building.

3.4.2 Multi Map Navigation

The next approach is to update the open source multi_map_navigation pack that is on Github. Updating this pack will take time and effort to do but will allow us to easily switch between maps of the building and has an elevator script that can be modified to allow us to control the elevator. The multi map navigation will be able to navigate to goals on the currently selected map to other maps stored on the robot.

3.4.3 Build Our Own

The final approach is to create our own navigation package from scratch. This will take the most time and effort on our end. It would be able to achieve all of our goals that we have set up. It will also require very in depth knowledge of how ROS works and creating a navigation algorithm.

3.4.4 Chosen Navigation Approach

Our team choice is to modify the multi map navigation package. This option will allow us the best results because it has a proven navigation algorithm. Updating the package will take some time but the person who updated it last said that they would answer any question that we have about it.

	Navigation	Multi-map Navigation	Create Our Own
Switch Maps Between Floors	0	4	3
Transition Points	0	4	3
Elevator Commands	5	4	3
Avoid Obstacles	5	5	3
Ease of Setup	5	3	0
Overall Score	15	20	12

Figure 3.4 ranks navigation, multi-map navigation and a self made navigate tool on five categories from 0 (worst) to 5 (best)

3.5 Graphical User Interface

Not only is it important for the robot to be able to know where in the Engineering building it is, but it is also necessary for an operator to be able to track RAT's movements and see its location. In addition, a rather helpful stretch goal of ours is to also be able to see its status, such as whether or not there's a problem in its functions or, in the worst case scenario, it happened to fall down the stairs and was incapacitated. It could send an alert for the operator to see. And so, in order to do any of this, it is crucial to have some form of a graphical user interface.

Dr. Leverington made it clear that he would like something web-based on a computer platform rather than a mobile application (not that he couldn't have use for one in the future) for the GUI. Other than this, we are not limited by many constraints, since our main goal is navigation. However, we will still be looking for a GUI framework that would be the most organized when displaying RAT's movements on a map, the easiest to implement, and the simplest way to receive alerts from RAT. For the future, another thing to consider is the ease with which it is possible to integrate system admin or user accounts. These are the requirements we will use to compare our options.

First, to investigate two Python GUI frameworks. Python in general was chosen as a programming language because it's an interactive programming language and is known to be fairly simple to use and gaining traction. It also has a wide range of GUI frameworks, but the two picked from over a dozen options are Kivy and TkInter. Both were listed on a development blog titled "The 6 Best Python GUI Frameworks for Developers," Kivy for supporting multiple platforms including Raspberry Pi which we are using; and TkInter for being commonly bundled with Python, simple, and having a good GUI itself (Source: https://blog.resellerclub.com/the-6-best-python-gui-frameworks-for-developers/). CEF Python and Qt Designer were the environments used for testing based on the list from the Python Wiki (Source: https://wiki.python.org/moin/GuiProgramming).

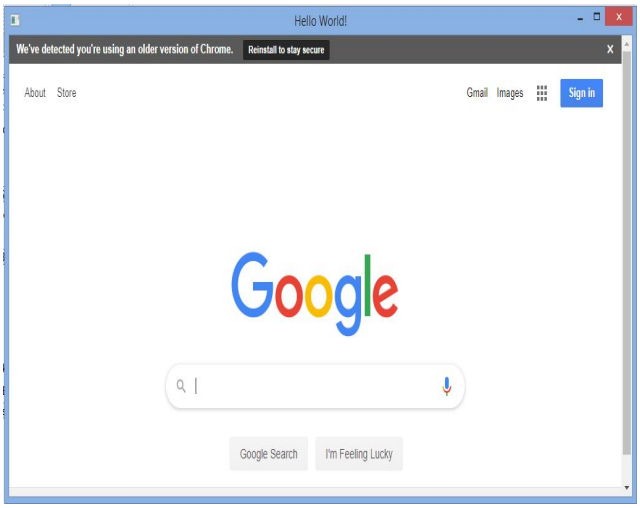

Figure 3.4a CEF Python GUI

CEF Python wasn't too difficult to figure out; a simple "Hello World" program was used and created a new GUI window (Figure 3.4a). The code is reminiscent of Python and JavaScript, which members of the group are familiar with.

Qt Designer is a bit less intuitive when testing as it is a form design tool with plenty of options that made it complicated to see what it was doing. However, because it gives much more control to the user as to the formatting, we can easily see it being a good tool to use in order to make a simple GUI for RAT's operator, and thus it is what we went with.

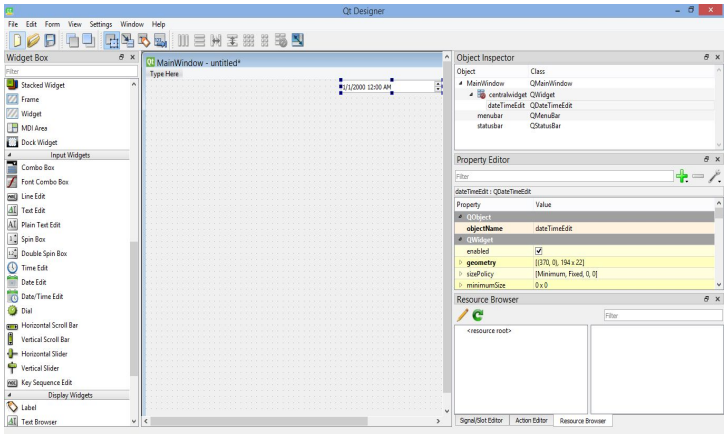

Figure 3.4b Qt Designer GUI IDE

4. Technology Integration

While software is the primary concern for NaviBot Systems, we must also account for integration with pre-existing hardware. We need to understand the physical capabilities of our current tools, and what limitations may or may not exist regarding hardware. Not only will NaviBot Systems need to develop three core features for the functionality of R.A.T., but we will also need to integrate the functionalities of the main features. Each module will be capable of collecting necessary information, as well as communicate the information to other modules in a meaningful format.

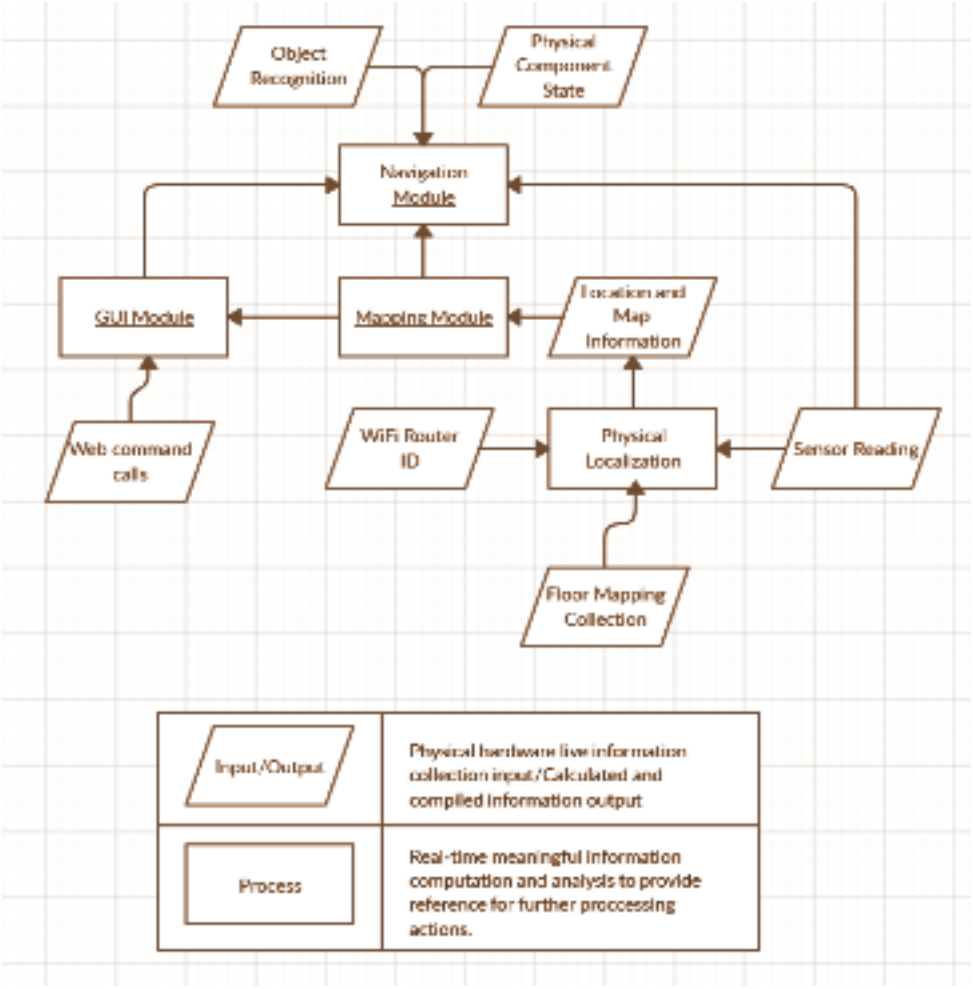

Figure 4: A system diagram for a current envisioned approach to developing a self-navigation robot.

In *Figure 4*, a system diagram depicts the envisioned interaction of individual modules planned for R.A.T. implementation. The three main components, GUI, Navigation, and Mapping, are all connected. Each module gathers information, calculating their given inputs, and converts the inputs into readable outputs for other modules in the series.

4.1 Navigation and Mapping Cohesion

Once R.A.T. gains the ability to read map inputs and operate equipped hardware, the navigation and mapping modules will need to communicate with one another to establish location. This means that the robot will need to record sensor information, such as hallway distance, reference points, and other surrounding inputs. After assessing the current area, the robot will transfer the information to the mapping module, and attempt to match similar layouts to maps stored in the robots memory. This will allow the robot to determine its location by simply referencing given data and comparing analyzed data.

Along with localization, R.A.T. will be capable of utilizing the given map to search for a specified location. This means that the robot can path find to a destination, since the robot will know its current location after performing a localization assessment. The mapping module can be seen transferring routes to the navigation modules in *Figure 4*.

4.2 Navigation and GUI Cohesion

While direct interaction with R.A.T. is required, this iteration of the GUI module will only need to allow the robot to communicate status updates to the end user. In cohesion with the navigation module, the GUI will provide the user real-time events. These events include:

- A current action: What movement task is currently being performed by the robot.
- A status: A message containing details of the robot's current state, such as moving and idling.
- An error message: Should some occurrence that forces the robot outside of normal operating conditions, an error must reach the end user.

The integration of inputs both modules utilize can be viewed in *Figure 4*.

4.3 GUI and Mapping Cohesion

Finally, NaviBot Systems will need to provide a method that will allow the GUI module to communicate with the mapping module. Without allowing the GUI to receive information from the mapping module, the GUI will not be able to display finer details to the end user. The mapping module will need to pass R.A.T.'s local information. Once retrieved, the GUI shows the user the robot's mapping operations, which includes the current location of the robot and the set destination for the robot.

5. Conclusion

The “Thirty Gallon Robot,” commonly referred to as R.A.T., for Robot-Assisted Tours, is an ambitious project. This robot is designed to be self-guided, with the ability to provide tours for future students and educational departments, particularly in electrical engineering and computer science. NaviBot Systems’ primary goal is to provide necessary software for the current iteration of R.A.T.; a GUI to allow end user communication; a mapping module to allow localization and path finding; and a navigational module to enable self-driving capabilities. This document discusses the technological feasibility of the “Thirty Gallon Robot,” and addresses the multiple technical challenges that we will face during production.

We are pleased to present our research, which we believe will lead to successful software integration and implementation of the overall objective of the project:

Technical Challenge and Solution Table		
Technical Challenge	**Proposed Solution**	**Confidence Level**
Implementing a navigational module that will allow R.A.T. to operate hardware for movement and environment analysis.	SLAM Algorithm (Gmapping): Excellent cohesion between software and hardware capabilities.	15 / 20 : SLAM Algorithm utilizes tools that pre-existing hardware provides.
Implementing mapping and localization that will allow R.A.T. to determine its current location and develop its own maps.	Indoor Localization Package: Despite variation in hardware, the package can be modulated to accommodate current hardware with included wifi localization mapping methods.	20 / 25: While it will take extra time to reconfigure the software to be compatible with R.A.T.’s hardware, the Indoor Localization Package is promising with documentation and tools provided.
Acquiring sensor data to facilitate mapping and navigation.	Kinect: The kinect provides an inexpensive way to gather sensor data.	5 / 5: The Microsoft Kinect sensor provides useful data scanning inputs, and related library support and documentation is widely available.
(Continued on next page)		

Technical Challenge and Solution Table (Continued)		
Implementing a navigational module which can handle multi-floor navigation.	Multi-Map Navigation Package: Built in escript will allow R.A.T. to control the elevator.	20/25: While the package still does not provide us with everything needed for multi-floor navigation, we feel confident in our ability to modify it to our needs.
Implementing a GUI that will provide end users with status updates and process information given by R.A.T.	Qt Designer: Although it is slightly less intuitive to work with, Qt Designer provides great formatting options which will allow us to create the best GUI possible.	8/10: QtDesigner will allow us to fulfill all requirements of the GUI as specified by Michael Leverington.

Figure 5: A technical challenge and proposed solution table, giving descriptions of each challenge planned to be solved using a researched solution, along with a confidence level

The table shown in *Figure 5* showcases a list of technical challenges documented. These challenges are obstacles that NaviBot systems will hope to overcome. *Figure 5* also displays proposed solutions; tools that will help aid in the future development of R.A.T.. These solutions are accompanied by confidence level ratings, meaning that the research spent in studying the listed tools have been chosen to help in solution development.

NaviBot Systems is excited to be given the opportunity to work on “Thirty Gallon Robot, Part Deux,” and look forward to developing and incorporating our proposed solutions elegantly into new software for this project. While it may take time for us to implement our features, we are sure any new challenges that arise during development will undergo a similar analysis process to ensure consistent and reliable solutions.