Lev Tours

Dr. Michael Leverington - Sponsor
David Failing - Mentor

Members
Erik Clark
Ariana Clark-Futrell
David Robb
Kyle Savery
Alexis Smith

Technological Feasibility Analysis

10/23/2020

Overview

This document is an outline to go over the feasibility of the technology used in the “Thirty Gallon Robot, Part III” and will go over all of the challenges that may come up in the process of creating this software. It will also cover our proposed solutions to those problems.

1. Introduction

In the field of computer science, mobile robotics is capable of increasingly complex tasks, and the necessary hardware for these tasks has become far less demanding in recent years. As a result, the costs associated with these materials have also decreased, allowing for a greater number of individuals and organizations to take part in the research and development of mobile robotic platforms. Most significantly, educational organizations can give students the opportunity to interact with robotics either from a mechanical or programmable standpoint. To take full advantage of mobile robotics, one of the main goals of our project, “Thirty Gallon Robot Part III, The Smiling Tour Guide”, designed by our sponsor Dr. Michael Leverington, is to demonstrate that an institution's budget for robotics can be further reduced, and used more effectively, by utilizing inexpensive materials. For Dr. Leverington’s robot, one of the most inexpensive components is the thirty gallon barrel that all of the other components are built around. When this project is complete, the robot will be an example of how instructors can get students involved in the world of robotics, which could lead to an expansion of the field, and even more breakthroughs to improve our society.

This project is also known as Robot Assisted Tours (R.A.T), which in addition to being low cost also aims to act as a tour guide for people exploring a given building. Those buildings will be located on Northern Arizona University’s (NAU) campus, starting with the engineering building and then at least one other building for testing. Our team, Lev Tours, will be responsible for designing a piece of software that is capable of localizing, or determining the position, of the device it is executing on via the Wi-Fi signal strengths of nearby routers. Based on this positioning, our software will be able to direct the user around the current building to specified destination points. Initially, the user will be an individual on our team running the software on a laptop or smartphone to later be integrated with the robot. Team Lev Tours is made up of:

- Erik Clark, Team Lead
- Ariana Clark-Futrell, Team Communications/Recorder
- David Robb, Team Release Manager
- Alexis Smith, Team Coder
- Kyle Savery, Team Architect

In addition to our sponsor, our team also has a mentor that will help guide us through this extensive project.

- Dr. Michael Leverington, Team Sponsor
- David Failing, Team Mentor

The Thirty Gallon Robot project is in its third year of development. The first team created the foundation for R.A.T. by programming the robot to respond to commands from an Xbox controller. The second team built upon R.A.T by creating a way for the robot to generate its own maps by navigating around the building. A stretch goal for the second team was to develop a framework for Wi-Fi localization, but due to the 2020 COVID-19 pandemic they were unable to test their design within the engineering building and so that development was not completed at

the time. Therefore, this task now falls upon our team. The need for Wi-Fi based localization came from the fact that the robot must be able to move around the building with minimal outside help by using either a map it creates or a map it is given by the user. This will allow for the robot to navigate on its own throughout a building without the requirement of the Xbox controller. Requiring user control defeats the robot's purpose of being an automated tour guide. Our team will develop the location software independent of the robot in order to support modularity, and allow for efficient testing that does not rely on the physical presence of the robot.

2. Technological Challenges

Creating a software product capable of Wi-Fi localization and subsequently guided touring will be accompanied with several technological challenges. For this project there are two main challenges we have to address. One challenge is that our software must use its map data to traverse a given building with a 95% success rate. That success rate is derived from testing how well our software is able to position the device within the building. More concretely, if the device is directed to a specific room, then if it is able to position itself within 2 meters of the doorway it will be regarded as a success. The other main challenge is that we must use Wi-Fi router signal strengths in order for the device to determine where the robot is on its map. With these two challenges met, we will have a minimum viable product (M.V.P).

Specific challenges for the M.V.P:

- Determining which programming language will be best for our software.
- Finding relevant libraries for our chosen language that enable our software to use Wi-Fi signal strength.
- Processing an image of a map to not only be displayed to the user, but used by the software for various calculations (i.e. shortest path to requested destination).
- Building a Graphical User Interface (GUI) that the user can interact with and make destination requests.

After the M.V.P is met we will start completing additional testing goals. For example, once the software has achieved its 95% success rate in the engineering building we must then move the robot to another building and start the process of mapping and teaching the software a new location in order to meet another 95% success rate in the new building.

3. Technological Analysis

These challenges require thorough research and extensive comparison between viable solutions that address functionality we will need for this project. The following section goes into more depth on these comparisons and details in an objective way why each solution was chosen.

3.1 Programing Language for Wi-Fi Based Location

In order for R.A.T. to know its location within a building we must create a program that can receive and analyze incoming data from all nearby routers. This will include the router ID and signal strength, which will then be used to determine position. The biggest question our team had to answer on this topic was which programming language would best suit our needs. The

next challenge will be to use the chosen language and its relevant libraries to correctly output accurate signal strengths for the routers.

The chosen programming language must be able to:

- Communicate with the robot's operating system, known as Robot Operating System (ROS).
- Run on low cost hardware, as this project is designed to be affordable.
- Access router signal strengths and ID's.

For a language that satisfies the above requirements, the following metrics must also be met:

- Accurate retrieval of router strength and ID's.
- Sufficient and updated modules/libraries to capture router data.
- Minimal resource footprint.

3.1.1 Programming Language Options

Language	ROS Compatible	Runs with minimal hardware	Able to access router information easily
Python	Yes	Yes	Yes
C/C++	Yes	Yes	No
Java	No	No	No

Table 3.1.1 A look at our options and criteria

For initial consideration, the languages we compared were Python, C/C++, and Java (Table 3.1.1). However, Java was removed from contention when it was discovered it could not communicate with ROS, while Python and C/C++ were compatible with ROS [1]. Delving further into Python and C/C++, Python can run on low cost hardware such as Raspberry Pis and most Linux based systems [2]. It is also well known that C/C++ can run on low cost hardware.

An application we will be able to utilize for accessing wireless information is called Wireshark. Even though Wireshark is a C/C++ based program, it is also compatible with Python. There are also several modules available for Python that would be able to interpret the data that we collect from Wireshark. Furthermore, there are open source modules for Python that include tools for GUIs, which will assist in other aspects of our team's solution. In C/C++, there are very few readily available libraries that are capable of interacting with wireless information. The libraries that we were able to locate were all deprecated and no longer functional. On the final metric mentioned, C/C++ would be more efficient resource-wise than Python, but because C/C++ is unable to access router information easily this fact is ignored. Consequently, C/C++ fell behind Python as a viable option for our programming language of choice.

Based on the above information, we selected Python as our programming language, as we are certain that this language will be as efficient as possible for the metrics listed in 3.1.

3.2 Library to Assist in Wi-Fi Localization

Our software must retrieve router signal strengths in order to localize the R.A.T. on a map, so we consider the Python modules available to achieve this.

When deciding on a module we had 3 main questions to take into account:

- When was the library last updated?
- What operating system (OS) was it originally developed for?
- Would it be easily set up to operate with Wireshark and our mapping system?

3.2.1 Library Options

Module	Last Updated	OS Developed for	Easily Setup with Wireshark and our mapping system	Total
RSSI	4	Linux (2)	3	9
Indoor Localization	5	Kinetic ROS (5)	4	14
subPos	0	Arduino (1)	0	1
Robot Localization	3	Linux (2)	3	6

Table 3.2.1 A scoring system of 0-5 was used to help us decide on the best module

In Table 3.2.1, we score our candidate modules on a scale of 0 - 5, with 5 being the best score, relative to the four criteria above.

- RSSI - (Received Signal Strength Indicator) [3].
- Indoor Localization [4].
- subPos [5].
- Robot Localization [6].

It is important that the module we choose be current, as older modules may not be compatible with the version of Python that we selected. For this reason, RSSI received a score of 4 because it is 2 years old, but has fair documentation and seems to be well supported. Indoor Localization received a 5 because it was updated earlier this year and is supported with comprehensive documentation. The subPos system received a 0 as it is 4 years old and incomplete. Robot Localization received a score of 3 because it is up to date however it loses 2 points due to a lack of any useful documentation.

The next point we considered was what OS the module was developed for. Within this category RSSI received a 2 because it was developed for linux with specific linux commands that we will not be using. Indoor localization received a 5 because it was developed for Kinetic ROS which will integrate well with our system. SubPos received 0 points because it is set up for Arduino microcontrollers which is not relevant for our project. Robot Localization was also developed for Linux, and as such received the same score as RSSI.

Finally, we needed to check if the modules would be compatible with our mapping systems and Wireshark. For this, RSSI received a 3 since it would need just a few modifications for us to be able to use Wireshark to gather the router data needed for the system. However, it lost an additional point because it will not be easily integrated into the mapping system. Indoor localization received a 4 losing only 1 point because similar to RSSI some modifications would be required for it to operate with Wireshark, but it should integrate well with our mapping system. SubPos received a 0 because after further investigation, key pieces of this module are not finished. Robot localization received a 3 for the same reasoning that RSSI received a 3.

3.2.2 Proposed Solution For Wi-Fi Based Localization Module

Indoor Localization received 14 out of 15 total points and meets more of our requirements than any other module considered. To reiterate, it has been recently updated which keeps the module stable and it will work well with Wireshark and our mapping system.

3.3 Image Processing

Our software must be able to take in an appropriate, to-scale image in several types of formats. The image processing module should also be able to determine the height and width of the image in pixels. Other functionalities may prove useful in the future, such as the ability to overlay one image over another.

This module must:

- Be usable by the Python programming language.
- Utilize multiple image formats.
- Be able to determine the dimensions of a given image in pixels.

3.3.1 Image Processing Options

The options we researched included the Pillow and the openCV library. The initial criteria that both of these libraries needed to meet was being able to accept an image, which both of these libraries are able to do. We then moved to comparing the features that these libraries offered to determine which would be the best for our mapping system.

The first library examined will be Pillow. One challenge was finding what type of file formats the library was able to work with. Pillow is able to reliably use standard image formats such as JPEG, PNG, and PPM along with a large number of file formats that are infrequently used. Listed below are some of the features that Pillow comes with:

- Image rotation

- Image cropping
- Image resizing
- Image transposition
- Overlaying an image over another image
- Creating a histogram from an image.

Moving onto openCV, the file types openCV was able to work with include PAM and PNG. As far as features are concerned, openCV is limited to basic mathematical operations, and so to achieve similar features that Pillow is able to offer we would need to include additional libraries.

3.3.2 Proposed Image Processing Solution

Considering our requirements, Pillow is the only suitable alternative. The wide variety of image formats it can take in as well as the multitude of features it comes with made it easily stand above openCV. Pillow will help us not only take in an image of a map, but also help resize that map to fit on any screen. By using the size of the map we can also accurately use our GUI to overlay a grid onto the image in order to place bounds on the actual map. All of these features and functionality were exactly what we were looking for in a solution to image processing.

3.4 Graphical User Interface

Once our image processing system is operational, we must design a GUI that can display navigation data and is accessible by visitors to a given building, our team for testing purposes, and eventually the robot to display information to the user.

A few issues to consider when picking a GUI:

- It must have grid functionality.
- It must be able to operate on and display images.
- It must be compatible with Python.
- It must accept input to correctly define how the grid is overlaid on an image of a map.

3.4.1 Options for Python GUI's

	Tkinter	Command line GUI	OpenCV
Grid functionality	Y	N	N
Operate on images	Y	N	Y
Compatible with Python	Y	Y	Y
Accepts inputs for grid definition	Y	Y	N

Table 3.4.1 Options for Python GUI's

There are several options for GUI's that our team can potentially use for this project. Those options include a simple command line based GUI, Tkinter, and openCV that are all included in Table 3.4.1.

Tkinter has the following features:

- Mouse tracking
- Grid functionality
- User inputs via mouse or keyboard
- Familiar GUI System much like JavaScript
- Built on Python

The command line GUI would have the following features:

- Simple to use
- Only accepts keyboard input

OpenCV has the following features:

- Mouse tracker
- Mouse drawing ability
- Video playing

3.4.2 Proposed GUI Solution

Our team decided upon using Tkinter as our GUI since it has all the key features we are looking for including the use of a grid and an ability to edit, then project an image. Tkinter has also been shown to be able to work nicely with our image processing tool, Pillow. In addition, Tkinter has a multitude of features that could become valuable in the future. It works much like GUI programs our team members have used in the past which have 'containers' as a way to create scenes on a canvas, which will allow us to quickly get up to speed with how to effectively use the library.

4. Technology Integration

Now that our team has decided upon a programming language, image processing tool, a process for linking our map with Wi-Fi, a method for navigation, and a GUI, we can discuss their integration. As we will not be working directly with the robot, the software designed to physically move the robot is irrelevant. As such, our software will be extremely modular and capable of being ported to many different devices. Our main goals are to make an interactive map, display the map to the user and take in the input from the user on where they want to go.

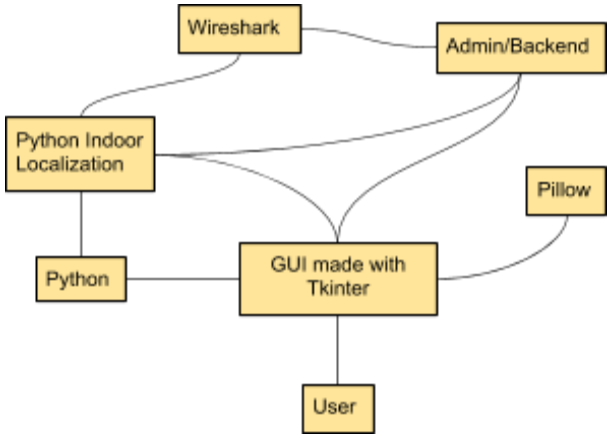

Figure 4.1 Systems Diagram

Figure 4.1 shows a Systems Diagram of the connections between our processes. The central hub is the GUI, since it must be able to work with both the backend and frontend users, as well as being able to communicate with the Pillow library, Python code, and Python Indoor localization module. The back end user will use the GUI as a means to train the software to prepare it for the front end user. While the front end user will use the GUI as a means of being able to operate the device without needing a deeper understanding of the software in order for it to function. The Pillow library and GUI will be used in tandem in order to take in an image of a map, scale it correctly, and then overlay a grid onto the image for navigational uses.

The Python code must be able to work with the GUI so that it knows what process should be running at what time, such as knowing when it needs to start calculating the shortest route to a point or even where that point is within the building. The localization module must communicate with the GUI in order to display important information about Wi-Fi signal strengths to the backend user for testing purposes. Wireshark, which can scan for routers and test their strengths needs to be accessed by both the Python Indoor localization module and the backend user. The backend user needs to access the Wireshark program in order to properly input the necessary information for the Wi-Fi scan and observe the raw output data. The Python code and the Indoor localization module must work together in order to take in the coordinates that the localization module calculates and display them on the GUI.

This integration is our proposed solution to the problem of designing a software product capable of localizing itself within a building based on nearby network routers.

5. Conclusion

The capabilities of mobile robotics continue to expand and the societal problems that can be solved by them are growing. As a result, mobile robotics is an increasingly valuable field and the more people who can get involved early on expands the subject area both further and faster. Our project, “The Thirty Gallon Robot”, seeks to demonstrate affordable ways that educational institutions can get students involved with robotics. By contributing our software to a robot that is

as inexpensive as possible, our team (and previous teams) believe that we can show other instructors that getting their students involved with robotics is not only possible, but deeply rewarding.

Our team addressed several questions in its analysis of software components:

- Which programming language should we use?
- How should we retrieve router data?
- What image processing tool would best suit our needs?
- How could we handle the task of navigation?
- What type of Graphical User Interface (GUI) would not conflict with previous solutions?

Our solutions to the above problems and their accompanying rationale are:

- *Python*
    - Compatible with the Robot Operating System (ROS)
    - Can operate on minimal hardware, which is necessary for keeping this robot inexpensive
    - Has several viable modules that would allow us to interact with wireless information

- *Indoor Localization*
    - Regularly updated with comprehensive documentation
    - Compatible with Wireshark and our mapping system

- *Pillow*
    - Takes in many different image formats
    - Has a variety of useful features, such as:
        - Image manipulation (rotation, reverse, transposition, etc.)
        - Overlaying images onto one another

- *Tkinter*
    - Built on Python
    - Grid functionality
    - The format of this GUI is familiar to our team which will let us easily begin working with it

After extensive research into each one of these topics, our team is certain that once all of these components are integrated we will have created a functional prototype that meets the expectations of our team sponsor, Dr. Michael Leverington. Our next steps in creating an alpha prototype are to solidify our project's requirements and then begin working up small and modular testing prototypes to ensure that all major design decisions are meant throughout our development process.

6. References

[1] Saito, I. (Ed.). (2016). APIs. Retrieved October 09, 2020, from http://wiki.ros.org/APIs

[2] Mazzel, D. (2016). Embedded Python. Retrieved October 09, 2020, from https://wiki.python.org/moin/EmbeddedPython

[3] Villagomez, J. A. (2018). RSSI Python module. Retrieved October 09, 2020, from https://pypi.org/project/rssi/

[4] Erdogan, E. (Ed.). (n.d.). Indoor_localization Package Summary. Retrieved October 9, 2020, from http://wiki.ros.org/indoor_localization

[5] Wyatt, B. (2016). Subpos/subpos_receiver. Retrieved October 09, 2020, from https://github.com/subpos/subpos_receiver

[6] River, C. (2019). Cra-ros-pkg/robot_localization. Retrieved October 09, 2020, from https://github.com/cra-ros-pkg/robot_localization