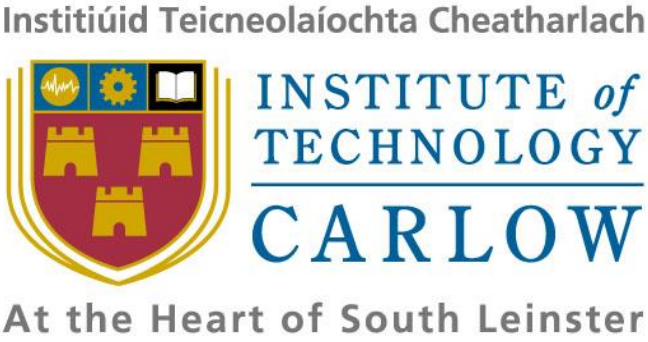

An exploration of Simultaneous Localisation and Mapping using Lidar

Paul Mahoney – C00227807

Final Document

April 2021

Contents

1. Introduction

I chose this project because through our four years of college we have had very little opportunity to work on the software applications in hardware. Before making my project proposal I had known my project must have some sort of hardware component. I also wanted to work in Linux and use Python as I gained an interest in them from our third-year web application module with Paul Barry. The initial intention of this project was a robot capable of obstacle avoidance. While it did so comprehensive maps were to be created with simultaneous localization and mapping (SLAM). Additional features such as the ability to store and reuse maps, take manual control, and get a live feed from a mounted camera were planned.

I will do so under a series of different topics. The first of these is what I achieved.

2. Achievements

In this section, I will be discussing what was achieved within the scope of this project. Considering the description in the introduction, we can break down this project into a few subheadings. I will discuss the progress of each of these headings.

- **2.1 Obstacle Avoidance:** This feature is currently implemented in an altered state. As the Picar had to be removed from my project I had to rethink and adapt how this project would work. How this was meant to work was SLAM would communicate with the obstacle avoidance script built within ROS's navigation stack. This navigation stack would be reliant on the car for its trajectory and other data. This idea fell apart as the car was meant to provide odometry (a method of providing locational data) for navigation. My solution was to independently develop this with a Skoltech RPlidar driver I found. I built this new system with the wall follower algorithm which outputs text display to the user. An example of this may be seen below:

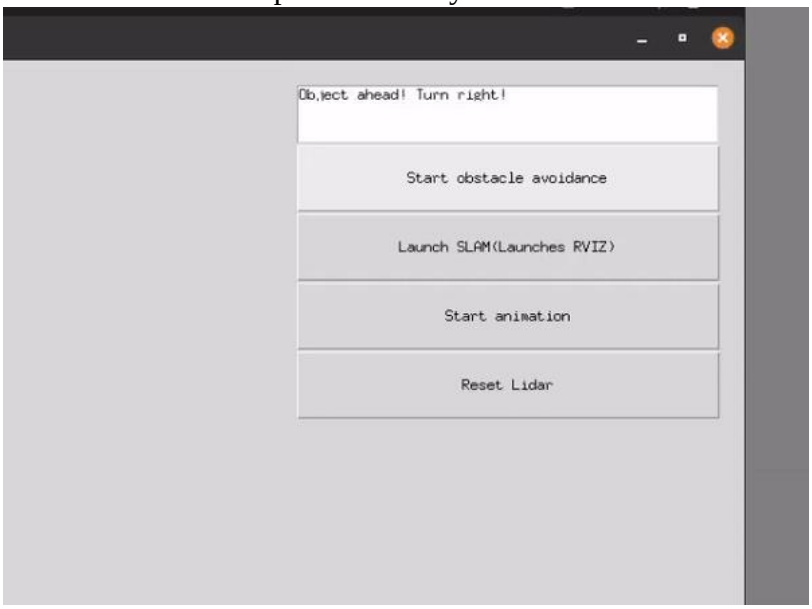

Figure 1: An example of obstacle avoidance in effect

- **2.2 Navigation:** The navigation portion of the project had to adapt to the situation with the car. This is now done manually with the user walking around a room with the lidar in their hands. As obstacle avoidance and mapping cannot work together (see Issues) building maps rely on the user for obstacle avoidance.

- **2.3 Mapping:** The mapping portion of the project works as intended. I adapted my launch package that connects Google cartographer with RPlidar. This was quite a simple process changing templates I found online. Comprehensive 2D maps can be created as seen below.

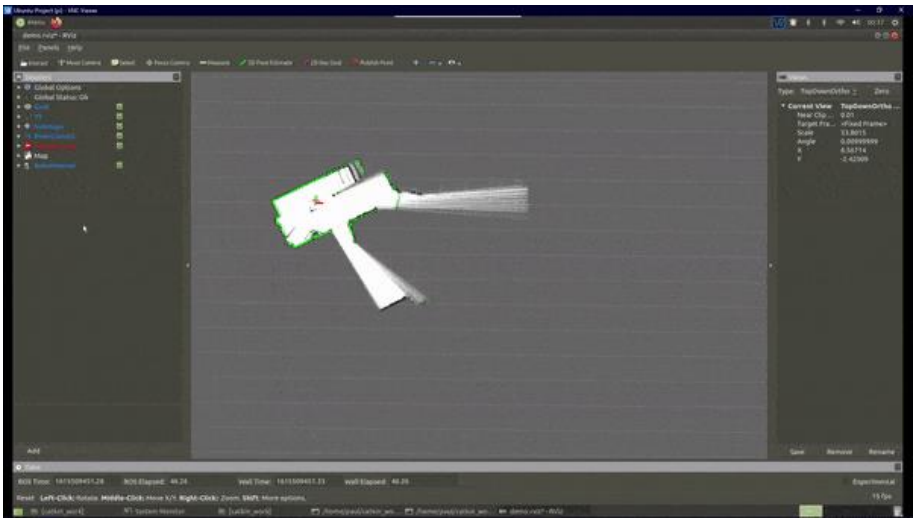

Figure 2: An example of SLAM in effect

- **2.4 Take Manual Control:** This feature has since been removed. How this initially was intended to work was to allow the user to select a button in the application. This would connect to the webserver on the Raspberry Pi and take control of the mounted camera. The user would control the car via W, A, S, D, or the arrow keys. The live camera feed would populate the same space as the visualization for obstacle avoidance.
- **2.5 Application:** An application was made to house all this functionality within. This was made using Python and Tkinter**.** I attempted my best to follow the GUI layout I had initially intended within the design document. I found Tkinter to be both easy and annoying to use. It is quite easy to get something implemented but fine-tuning it within Tkinter proved buggy and unintuitive. An example of this was attempting to get the placement of my buttons. The grid manager was quite difficult to use, and I ended up having to define the size of each column in the grid to get the effect I wanted. My frustration with this tool is correlated with my inexperience in using it. With enough practice and research, I imagine it can be quite useful for building applications within Python.

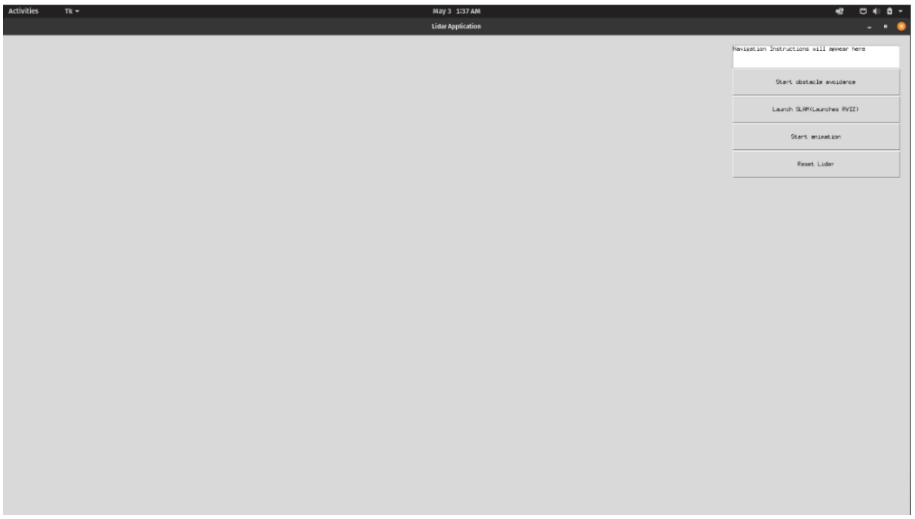

Figure 3: Application on startup

3. Issues

In this section, I will be outlining issues that arose within my project. Unfortunately, at the early stages of this project, these issues were nonstop. This is due to the combination of my inexperience in working within Linux and an array of hardware issues. Furthermore, I will break this section down into software, hardware, and ‘other issues.

3.1 Software Issues

One of the biggest issues that I came across is with the installation of the robotic operating system (ROS) on the raspberry pi. A large amount of time was lost here. During its installation, it would crash for various reasons such as memory usage or missing dependencies, etc. PYQT5 was the biggest issue regarding this. PYQT5 is an alternative package to Tkinter that adds a set of GUI widgets and other tools for building applications. It is a dependency for both ROS and the PiCar source kit. I attempted a wide range of installation methods such as attempting to use Pip, Sudo, updated GitHub Repos, manual installation and with little success. In addition, the installation of ROS took well over an hour so troubleshooting these issues became quite time-consuming. I would try one fix, wait an hour to have it fail, and repeat the process for quite a few weeks. During this time, I tried reinstalling Raspberry Pi OS multiple times to no avail. I never resolved this issue, but I speculate that the issue was with Raspberry Pi OS itself. After swapping to Ubuntu-mate (see hardware issues) this issue seemed to resolve itself. Other factors that caused ROS to crash were the default allocation of swap space being too small, CPU usage hitting 100%, and crashing the Raspberry Pi itself. These issues were resolved by increasing the swap space and limiting the CPU usage with the -J1 flag which allocates 1 core to the installation.

Interestingly I encountered none of these issues when I swapped to developing on my Laptop. I concluded that my project was too computationally heavy for my model of Raspberry Pi and have since solely been developing on my Laptop. The model of raspberry pi I was provided (Raspberry Pi 4b) possessed two gigabytes of ram and a quad-core 1.5 gigahertz CPU. In contrast, my laptop possesses 16GB of ram and a quad-core 3.5 gigahertz CPU. The strain of simultaneously running an operating system with a GUI and hosting my application seemed too much for the raspberry pi which is why I decided to develop it on my laptop solely.

In addition to this, the source kit for the Picar did not work. For context, this involved setting up a local server on the raspberry pi and connecting to it either from another machine or web browser. On the web server end, it kept crashing due to the dependency for PYQT5. On the laptop end, I found the software quite buggy, and I had to reinstall the software several times. I could not isolate the source of this issue and on one of my attempts of installation, it just worked. The furthest I got with this software was being able to send commands from my laptop through the local server and visually seeing the commands being received on the webserver end. This is where I began to identify hardware issues with the Picar.

3.2 Hardware Issues

Most of my hardware issues were all concerning the Picar-V which has subsequently been removed from the project. Troubleshooting issues with the car proved quite difficult as it is outside my area of expertise. The only conclusion I could come to was the servos were faulty. I properly followed the instructions included in the assembly of the car. During the

construction, it prompted me to test the servos while the car was half assembled to ensure they were installed in the right orientation. I followed this step which no success. The servos would not spin. After a bit of research, I hoped the issue was with the software and I continued and completed the car. This process totaled over 1-2 hours.

Once I got the car completed, I continued with the software installation. Here I ran into another set of issues. The first being the source kit would crash on startup. This circles back to the PYQT5 compatibility I mentioned earlier. I did some research and attempted a plethora of fixes suggested online, but none worked. Finally, after attempting another round of fixes I got the source kit to compile.

Once I got the source kit to compile, I encountered another issue. I would enter the IP for the raspberry pi to connect and I would get a timeout message. As the Pi was on the same network, I attempted both Raspberrypi.local and its local IP Address 192.168.0.47 but to no avail.

From here I was out of ideas and opted to restart again. I completely disassembled the Picar and flashed the SD Card. I opted for Ubuntu Mate over Raspberry Pi OS this time. I chose Ubuntu Mate as although Raspbian is the officially supported OS Ubuntu-mate comes with less bloatware and better hardware acceleration. Adding these factors to the hardware constraints I was already facing I decided to make the move. I then installed ROS and the Source kit. I couldn't get the servos to spin, and I had a new round of errors with ROS but I got both installed with fewer issues this time. I was now at a point where I could compile ROS and read Lidar Scan data. Finally, I could connect to the Picar via the source kit but again I ran into a new error. Despite what I did I could not get the car to move. I would launch the server on the Raspberry Pi, connect remotely from my Laptop and I could see through the mounted camera in the provided software. I could see commands being sent to the car on the server hosted on the PI, but the car did not move. From here I tested an additional few online methods of fixing the car. The most notable was I partially disassembled the car and swapped out any cables for the spares in the kit. Again, I could see through the camera, and see the commands being sent through the server. This is where I concluded that the Servos in the car were faulty and consulted my supervisor and we mutually agreed that I should move on from this aspect of the project. I had spent too much time on this aspect of my project and made little to no progress on manipulating the Lidar scanner.

3.3 Other Issues

The first issue I stumbled across interestingly was I could not gain access to the Lidars serial port via my desktop PC. I never solved this, but I assume this issue is either a compatibility issue with my motherboard or some setting within my BIOS. Regardless this was not the case with my laptop, so I moved on.

Another inconvenience was all the documentation surrounding my Lidar scanner came from poorly-translated Chinese so learning about the inner workings of the lidar was difficult. This is something I could not resolve so I had to rely on the documentation of similar products/projects to learn about the lidar.

Additionally, another issue encountered was Mapping and obstacle avoidance could not be done at the same time. This is because only my obstacle avoidance script in python or SLAM may have access to the hardware at once. I have attempted launching SLAM while obstacle avoidance was in effect and vice versa. My initial intention was for both these subsystems to

be a part of ROS but due to the car being faulty I had to abandon this route and incorporate my obstacle avoidance script into Python.

Finally, the last issue I encountered was with my chosen SLAM package. For a major portion of the project, I had narrowed my search down to Hector SLAM and Gmapping. I had initially intended to use Gmapping as it utilized both laser scan and odometry to build maps while Hector slam relies solely on laser scan data. After considerable testing, I concluded that Gmapping would better serve my purpose as hector slam had an issue where it kept losing its position in a room. After some research, I discovered Hector SLAM is best used in conjunction with other sensors such as a proximity sensor. After I concluded that the Picar must be scrapped from this project Gmapping also had to be scrapped due to its reliance on odometry provided from the Picar. This was a pivotal stage in the project where I needed to find a replacement. This is where I discovered Google's Cartographer and began to incorporate it into my project. I chose Google's Cartographer as it has its built-in method to replace odometry. The algorithm is best compatible with Lidar scanners and is made to not rely on odometry for positional data. Where the cartographer differs is that it can use existing data to correlate where it is within a room. It estimates its position from pre-existing scanned objects known as 'pose'. This works quite like visual SLAM discussed in my research document.

4. What I would change

The biggest formative change is that I would not use the Picar. Having been through this project I am more interested in the applications of this technology outside of autonomous vehicle navigation. For instance, I would pursue this project to create a helmet/hat that helps blind people get around. I would also stay away from ROS. Although it is a powerful tool, I found it quite difficult to work with. It has a steep learning curve and at my current skill level, I could not utilize it to its full potential. Instead, I would use the time afforded by dropping the navigation to develop my type of SLAM algorithm in python.

Another change I would make is to allow myself more time to research a package/technology before using it. This was most prevalent with ROS and Tkinter. More research and practice with these technologies at an earlier stage of the project would have been much more beneficial and subsequently ended in a better application. This is the kind of mistake that had to be made for me to learn otherwise.

I think my biggest pitfall in this project was the reliance on premade packages. Between ROS, cartographer, and the navigation stack I would have had very little coding input. Although I learned a lot about these technologies, and they were quite difficult to implement I am not sure this route would have aligned with the purpose and learning outcomes of the project module. For this, I am glad that the car was faulty as it tested my skills to adapt, and I got to utilize Python which is the reason why I chose this project in the first place. Through our four years of college, we have had every opportunity to explore the hardware applications of software. I am grateful for this project and what I have learned throughout it.

5. Acknowledgements

I would like to thank my supervisor Oisin Cawley. Without his guidance and expertise, this project may not have been possible. He set my mind at ease when the project had to deviate from its intended path and ensured I was getting something out of it.

6. Plagiarism Document

- I declare that all material in this submission, e.g. thesis/essay/project/assignment, is entirely my own work except where duly acknowledged.

- I have cited the sources of all quotations, paraphrases, summaries of information, tables, diagrams, or other material; including software and other electronic media in which intellectual property rights may reside.

- I have provided a complete bibliography of all works and sources used in the preparation of this submission.

- I understand that failure to comply with the Institute's regulations governing plagiarism constitutes a serious offense.

Student Name : Paul Mahoney

Student Number : C00227807

Student Signature : Paul Mahoney