GUI Design for Automated Mission Planning and Controller Synthesis

Edin Guso, 23435

July, 1 2019 – August, 29 2019

University of Texas at Austin

Prof. Ufuk Topcu

September 12, 2019

Sabanci University

Faculty of Engineering and Natural Sciences

Mechatronics Engineering - Computer Science & Engineering

Abstract

Mission planning and controller synthesis for autonomous robots is a complex task and requires knowledge about several topics such as control, robotics and linear temporal logic. Wrapping these tasks with a graphical user interface and making these tools usable by people without the technical knowledge is important. Because that would result in an increasing number of people getting involved and thus make the field grow.

The main objective of my internship in Autonomous Systems Group at the University of Texas at Austin was to create this graphical user interface. The constraints for this software were that the software had to be easy to use, include all the necessary features, handle all the mission data in an efficient way and also be able to output this data in a meaningful way so that the controller synthesis software could take it as the input.

In addition to developing the mission planning software, a path planning software has been developed to process the data from a simple mission to confirm that everything works as intended. Finally, a simulation of those paths has been created to have a demonstration of the whole process, from the user inputs to the simulation.

The project required a knowledge of C++ and autonomous robotics topics such as path planning. During the project, I learned to use Qt Creator and used it to develop the software. This project offered me an opportunity to improve my C++ skills, learn software development and develop my research skills.

Contents

1. Introduction

I have conducted research and developed software as my summer internship course in the University of Texas at Austin. The main objective of this project was to create a software that allows people with no control or path planning knowledge to be able to configure their missions easily and reliably. My part of the project was to create a prototype software that will be the foundation for the further development of the user interface part. Other than developing the user interface, I created a path planning software which can take a simple mission from the user interface and plan all the required paths. These paths were later used in the Gazebo simulation environment to demonstrate the whole procedure, starting from getting the user inputs and ending with a simulation of the mission.

The project does not end with my internship. The progress I made during my internship is the foundation for a project that will continue at least two more years. The user interface will be improved, and new features will be added. A controller synthesis software taking Linear Temporal Logic (LTL) specifications as input will be developed which will allow all kinds of missions to be processed.

And finally, the front-end and back-end will be connected to make the process automated, all the way from the user inputs up to the simulations.

Currently, only people with knowledge about robotics and controls can design and plan missions for autonomous robots. This project is very important for the future of autonomous robots since it allows people with no knowledge about these topics to get involved in mission planning for these robots. As more and more people are able to get involved, the field will expand much quicker leading to more funding, more research and more advancements in the field.

2. Company/Institution Background

2.a University of Texas at Austin

University of Texas at Austin is a well-known university in the United States with more than 51000 students and 3000 teaching faculty [1]. The university was founded in 1883 [2]. It is one of the highest ranked universities.

> The university is one of the top 20 public universities according to U.S. News & World Report, with the No. 1 accounting, Latin American history and petroleum engineering graduate programs in the country — plus more than 15 undergraduate programs and more than 40 graduate programs ranked in the top 10 nationally. [1]

The core purpose of the university is "to transform lives for the benefit of society." [3]. The mission of the university is described on their website as:

> The university contributes to the advancement of society through research, creative activity, scholarly inquiry and the development and dissemination of new knowledge, including the commercialization of University discoveries. The university preserves and promotes the arts, benefits the state's economy, serves the citizens through public programs and provides other public service. [3]

2.b Oden Institute for Computational Engineering and Sciences

I conducted my research in the Oden Institute for Computational Engineering and Sciences. The institution was established in 2003 and has more than 330 people [4]. The institution is described on the website as:

> The Oden Institute for Computational Engineering and Sciences is an organized research unit created to foster the development of interdisciplinary programs in computational sciences and engineering (CSE), mathematical modeling, applied mathematics, software engineering, and computational visualization. [5]

2.c Autonomous Systems Group

I was a part of Autonomous Systems Group. The group consists of 18 PhD students and several post-docs and is lead by Prof. Ufuk Topcu. The group mainly does work on machine learning, autonomous robots and formal methots. The group's focus is described as:

> The Autonomous Systems Groups has a focus on the theoretical and computational aspects of autonomous systems with an emphasis on systematic verification and design of such systems. Its intellectual core will be in the intersection of controls, theoretical computer science, learning theory, and computational engineering. [6]

3. Internship Project: Description and Analysis

3.a Purpose

The main purpose of this project was to allow people with little or no background in robotics and control to get involved in the mission planning for autonomous robots. We want people to be able to place different types of robots with several missions each with different specifications on a map of their choice. We also want this process to be as intuitive and easy to use as possible. A couple of help texts should be enough for a person to start using the software. It should not take a long time for the user to get familiar with all the features.

It should be possible to extract the user inputs in a meaningful way and plan the high-level control based on those inputs in an automated way. Resulting high-level control must be close to optimal. For example, a quad-rotor navigating through the streets must not take unnecessarily long roads. However, this does not imply that the quad-rotor will always take the shortest route possible. The scope of this project does not include the low-level control for the robots since it is handled by ROS (Robot Operating System) with the use of Px4 flight control software.

The final part of the project is displaying the mission results. That is done by using the results from the high-level control in the Gazebo simulation environment. After automating all these steps, the input will be the user input and the output will be the simulation results. Having this simple input-output structure and keeping the complex computation in the back-end of the software makes it very easy to pick up and use, which is the main goal of the project.

3.b My Part

This project is very large-scale and it cannot be finished by a single person in two months. Therefore, I was given a part of the project which would be a foundation for the future steps. My objective was to create a graphical user interface prototype which would test most of the required features. This would allow the future developers to see which features are good, which should be changed, and which should be left out. Also, the prototype can be tested by users to get feedback which is very important for the development of a graphical user interface.

Other than creating the graphical user interface prototype, I wrote a path planning program which can get a simple mission as an input and output the necessary paths to achieve the mission (high-level control). Finally, the paths were simulated in Gazebo.

To summarize, my part in this project was to build up the foundation and create a small-scale test to see whether everything is working as intended.

3.c Methods

3.c.1 Software Used

The main software used throughout the project was Qt Creator. Qt is an open source widget toolkit and I have used it to create all the graphical user interface. It is a well developed tool for developing graphical user interfaces. There are many existing libraries for widgets that can be used. Also, it is possible to inherit from the existing classes and implementing custom widgets. The code was written in C++. The version of Qt used was 5.13.0 with the compiler MinGW 7.3.0 32-bit for C++.

For the simulation part of the project, several other software has been used. ROS (Robot Operating System) and Px4 flight control software has been used for handling all the aspects of the robots such as the low-level control of the robots. Gazebo simulator, an open source 3D robotics simulator, has been used to simulate the physics and the environment in which the robots will move.

3.c.2 Project Steps

In the first week of my internship, we had decided that I should first implement all the features that are planned to be used individually before implementing them as one software. This has allowed us to easily iterate on the individual features until we are satisfied with them. This process of trying out each feature lasted

for 2 weeks. Then, before moving onto implementing the graphical user interface with all the required features, I created a coding and documentation standard which would allow me to make the whole software consistent. I have taken inspiration from [7] and [8] while creating the document for the coding and documentation standard which can be seen in Appendix A. After preparing that document, I moved onto implementing the graphical user interface and it took me 4 weeks to complete it. After finishing the graphical user interface, the last 3 weeks were left for doing research on path planning, implementing a path planning software for a simple mission and simulating the calculated paths in Gazebo simulator.

3.d Design

3.d.1 Program Flow

In this part, only the mission planning software's program flow will be discussed since the path planning software is not planned to be used by the end user. The path planning software has been developed to demonstrate the whole process which will be automatic when this project is finished.

The program execution starts with displaying a simple window with two functionalities. The user can either create a new project or open an existing project. If the user chooses to open an existing project, they are asked for a folder containing the project and the project is loaded and displayed.

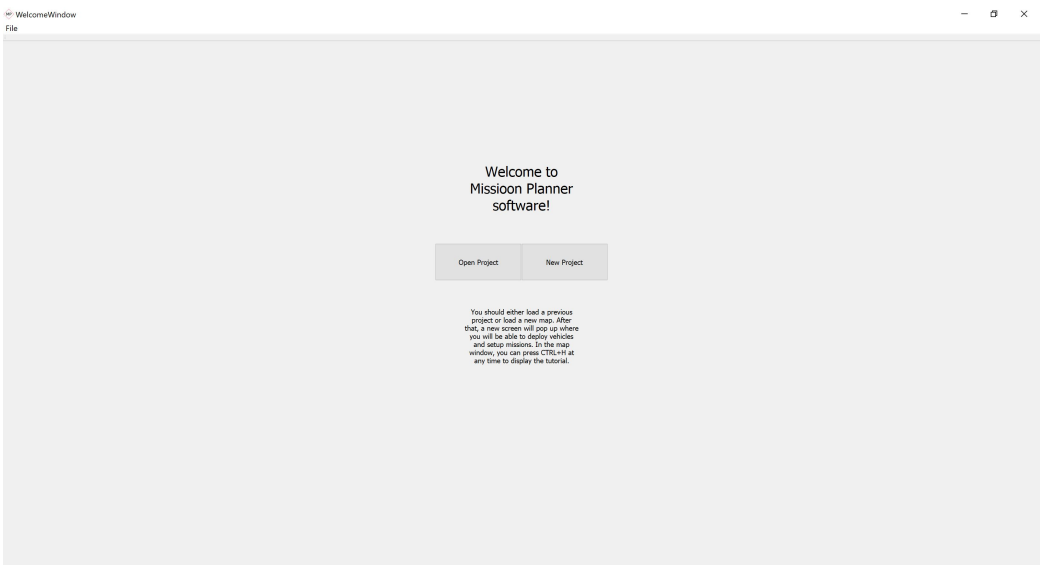

Figure 3.1: Welcome Window

However, if the user chooses to create a new project, they are asked for an image file containing the map they will be configuring their missions on. After a file is selected, the image from that file is displayed.

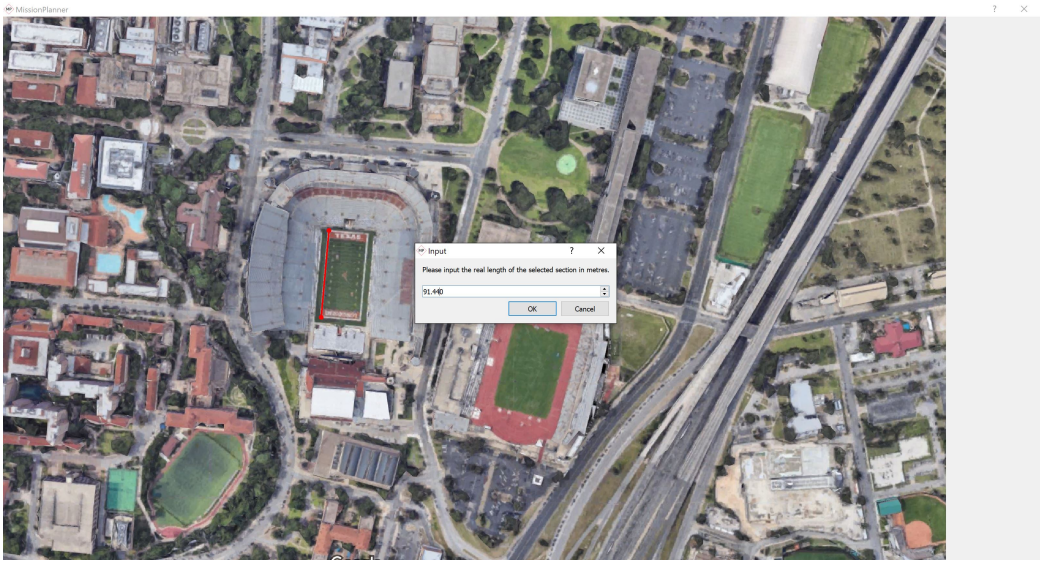

Figure 3.2: Scale Inputs

Then the user is asked to select a distance on the map and input its real-world distance, so that these values can be used as a scale for converting pixel values to real world values. Also, the user is asked to select the origin on the map. After these steps are done, the execution is at the same step as in the open project path.

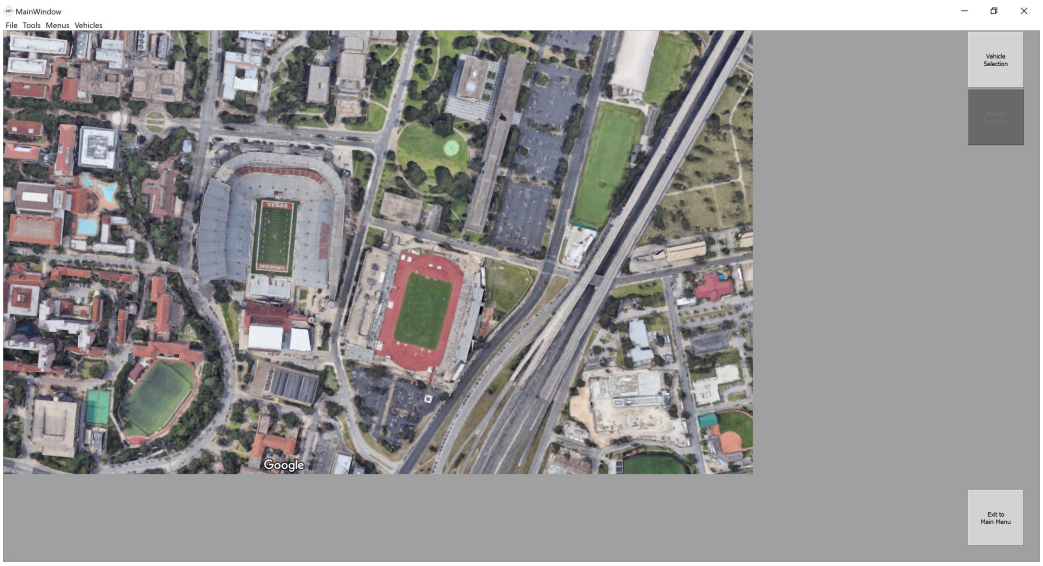

Figure 3.3: Mission Planning Environment

After the setup steps, the user has many options and can select whichever they want. They can deploy vehicles, setup missions, save their project, go back to the main menu, delete or move any vehicle/mission, undo the last action, zoom in/out on the map and many more. All these features have been tested in different orders and combinations until all the bugs have been fixed so that the user can have a smooth experience using the software.

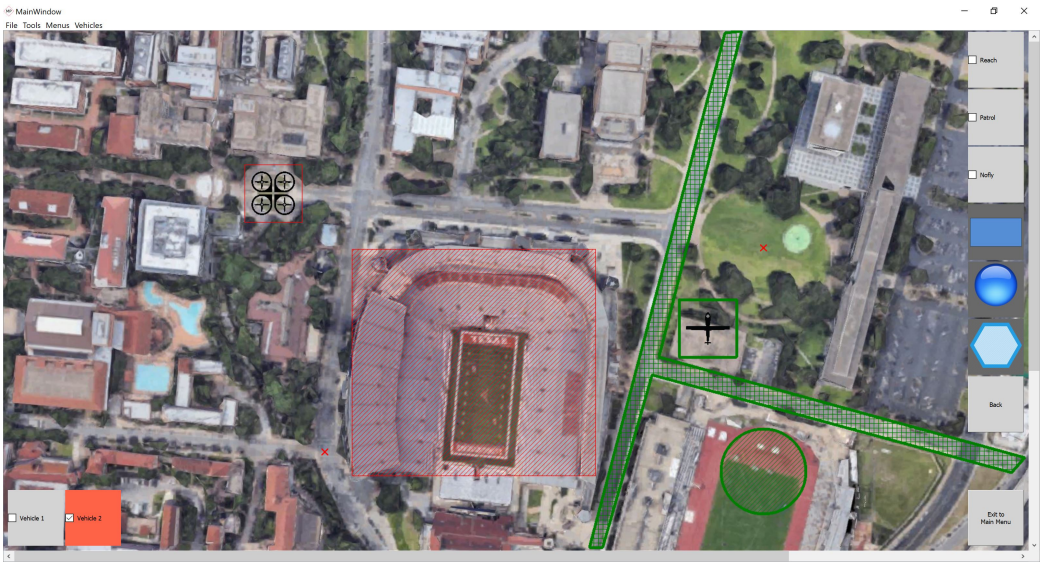

Figure 3.4: Two Vehicles Placed

In the current state of the software, there are 3 types of vehicles. Users can deploy rovers, quad-rotors and fixed wing UAVs. There are also 3 types of mission that users can setup. These are reach missions (with options to make the reach points in order or not in order and go always or go once), surveillance missions (which parts of the map should be surveyed by the vehicle) and no-fly missions (which parts of the map should be avoided by the vehicle). The surveillance and no-fly zones can be marked using one of the 3 shape tools. The available shape tools are rectangle, circle and polygon.

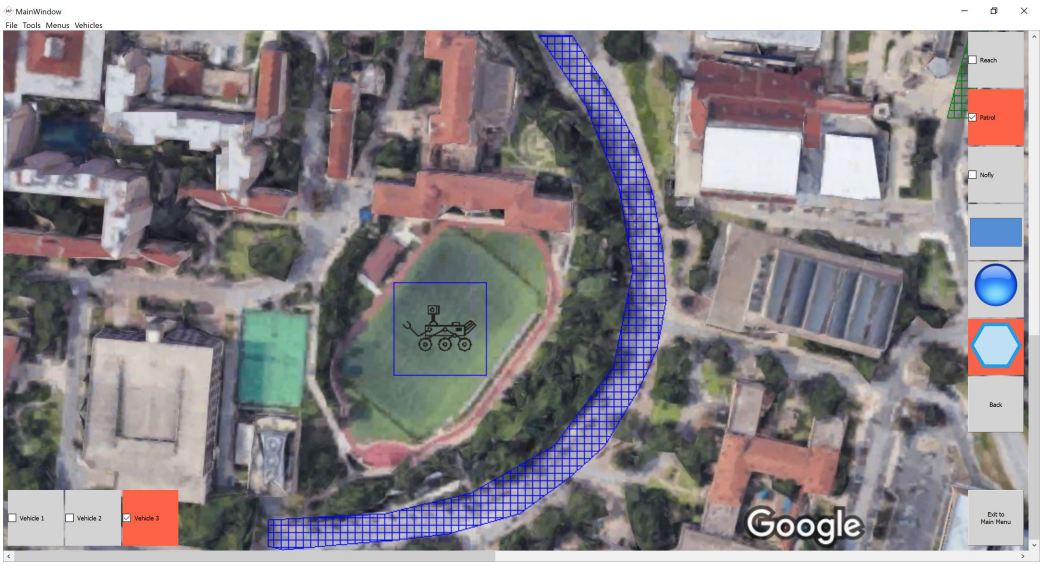

Figure 3.5: Another Vehicle Placed

The project can be saved at any step. A saved project can be opened by the user later. Also, this saved project includes all the necessary data for high-level planning that will be handled in the back-end. Save folder includes a .qmp file which is a text file including all the necessary information, and a .vmp file which saves the map in png format.

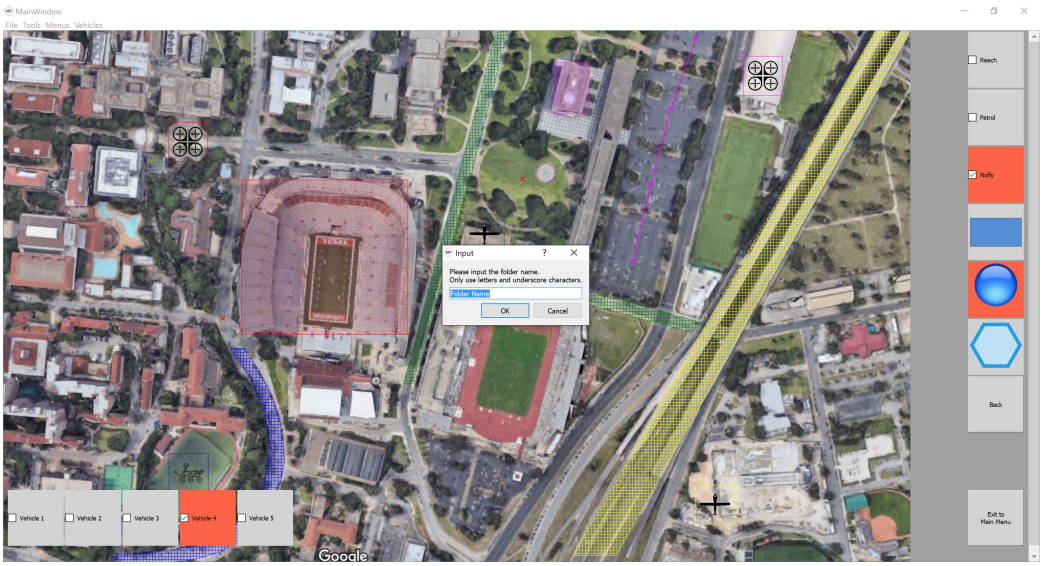

Figure 3.6: Mission Configuration Finalized

3.d.2 Object Design

Since this project had to handle a lot of data, I had to come up with an efficient and systematic way of storing the data. The most important issue I had to solve was to be able to hold all the different vehicles in a single container. Also, each vehicle had to hold all its missions (more importantly the shapes of the missions) in a single container. To do that I came up with the following class hierarchy.

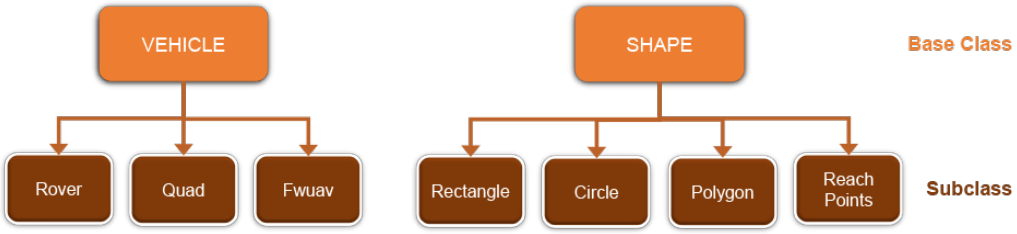

Figure 3.7: Class Hierarchy

I needed individual classes for Rover, Quad and Fwuav just to be able to hold different parameters for each type. However, the need for individual shapes was much more important. The first reason is that, for example, while a rectangle can be described using two points (each point being two integers), a circle would need a center point (two integers) and a radius (double). Another reason is that all these shapes have different ways of being painted onto the map. Also, when the user clicks on the map to select a shape (to either delete it or move it), all these shapes have different formulas for computing whether the click was inside the shape or not.

Since it was a necessity to have these different classes and I had to store them in a single container, I decided to design two base classes, Vehicle and Shape. Then I implemented all the necessary sub-classes by inheriting from these two classes. This allowed me to create different types of objects (eg. Rectangle or Circle) and store them in a single container (eg. vector<Shape>).

There were of course many other classes implemented in the process of cre-

ating the software. There are several classes for handling the buttons, text boxes and pop-up windows being displayed throughout the program. However, I am only mentioning the above class hierarchy in detail because it is a crucial part of this software.

3.d.3 Challenges, Solutions and Algorithms

3.d.3.a Deleting Vehicles and Missions

We wanted to have a way to delete the placed vehicles or configured missions since the user can make mistakes and there should be a way to reverse that mistake. Also, we wanted to have an intuitive way of selecting vehicles and missions. The most intuitive way of selecting them is clicking on the map to select them. However, detecting which one was clicked is not very easy since we only know the location of the click.

First, I had to come up with functions for detecting if the click is in a given vehicle or mission. All the vehicles have a rectangular box for click detection. However, the missions can be rectangles, circles or polygons. Therefore, there are 3 different functions for detection.

- For Rectangle: Consider the top left and bottom right corners being points P_1 and P_2 with coordinates (x_1, y_1) and (x_2, y_2), respectively. Set $x_{min} = min(x_1, x_2)$, $x_{max} = max(x_1, x_2)$, $y_{min} = min(y_1, y_2)$ and $y_{max} = max(y_1, y_2)$.

Given these values, it is possible to check whether any point P with coordinates x and y is inside of the rectangle or not using the following formulas:

$$x_{min} < x < x_{max}$$

$$y_{min} < y < y_{max}$$

- For circle: Consider the center point O with coordinates (x_o, y_o) and the radius r. Given these values, it is possible to check whether any point P with coordinates x and y is inside of the circle or not using the following formula:

$$\sqrt{(x - x_o)^2 + (y - y_o)^2} < r$$

- For polygon: Detecting if a click is inside a polygon or not is more difficult compared to the case with a rectangle or a circle. In order to solve this problem I used the C++ implementation [9] of an algorithm [10] which draws an infinitely long (very long) horizontal line from the point which needs to be checked to its right and counts the number of intersections between that line and the sides of the polygon. If the number of intersections is an odd number, then the point lies inside of the polygon.

Now, using these shape detection algorithms, we can figure out which vehicle or mission was clicked. The following algorithm is executed for each mouse click.

Result: Returns the clicked vehicle or mission
Inputs: cursorPos; vehicles;
for *each vehicle in vehicles* **do**
    if *vehicle.isInside(cursorPos)* **then**
        return *vehicle*;
    else
        for *each mission in vehicle.getMissions()* **do**
            if *mission.isInside(cursorPos)* **then** **return** *mission* ;
        end
    end
end
return *NULL*;

Algorithm 1: Vehicle and Mission Detection on Mouse Click

3.d.3.b Zoom in on the mouse location

One of the features that were required for the software was being able to zoom in on the point where the cursor is pointing. This is a very important feature for user comfort; however, it is difficult to implement. In order to resize and move the image with respect to the cursor position, I implemented the following formulas into the program.

The first formula computes the distance of the cursor to the center of the image $d = (x_d, y_d)$ using the given values of horizontal scroll bar position x_{sb_h} and width w_{sb_h}, vertical scroll bar position y_{sb_v} and height h_{sb_v} and cursor position $P_c(x_c, y_c)$.

$$d = (x_c, y_c) - (x_{sb_h} + \frac{w_{sb_h}}{2}, y_{sb_v} + \frac{h_{sb_v}}{2})$$

Using the found distance $d = (x_d, y_d)$, given horizontal scroll bar position x_{sb_h}, page step s_{sb_h}, and the current zoom factor Z, I have came up with a formula which calculates the new horizontal scroll bar position ${x_{sb_h}}'$

$$ {x_{sb_h}}' = x_{sb_h} + (Z - 1) \times \frac{s_{sb_h}}{2} + x_d $$

A similar formula can be applied for the new vertical scroll bar position ${y_{sb_v}}'$

$$ {y_{sb_v}}' = y_{sb_v} + (Z - 1) \times \frac{s_{sb_v}}{2} + y_d $$

Using these formulas to calculate the new scroll bar position it is possible to get a very natural feeling when scrolling in and out.

3.d.3.c Translating pixel values into real values

Since all the inputs received from the user are in pixel values, all the computations are done with the pixel values throughout the software to make everything consistent. However, the pixel values do not mean anything when it comes to the real world and thus, we have to convert the pixel values into real values. That is why at the beginning of the program execution, the user is asked to input a scale and an origin. Using those values obtained from the user, it is possible to convert the pixel values into real values. Given that origin is $P_o(x_o, y_o)$, pixel length of

the selected line is l_p and real length of the selected line is l_r, the following three formulas are written.

- Given a rectangle with pixel values for top left corner $P_1(x_1, y_1)$ and bottom right corner $P_2(x_2, y_2)$, the real values $P_1{}'(x_1{}', y_1{}')$ and $P_2{}'(x_2{}', y_2{}')$ can be computed using the following formulas:

$$(x_1{}', y_1{}') = (\frac{x_1 - x_0}{l_p} \times l_r, \frac{y_1 - y_0}{l_p} \times l_r)$$

$$(x_2{}', y_2{}') = (\frac{x_2 - x_0}{l_p} \times l_r, \frac{y_2 - y_0}{l_p} \times l_r)$$

- Given a circle with pixel values for center $P_c(x_c, y_c)$ and radius r, the real values $P_c{}'(x_c{}', y_c{}')$ and r' can be computed using the following formulas:

$$(x_c{}', y_c{}') = (\frac{x_c - x_0}{l_p} \times l_r, \frac{y_c - y_0}{l_p} \times l_r)$$

$$r' = \frac{r}{l_p} \times l_r$$

- Given a polygon (or reach mission) with pixel values for corners $P_1(x_1, y_1)$, $P_2(x_2, y_2)$, ... , $P_n(x_n, y_n)$ the real values $P_1{}'(x_1{}', y_1{}')$, $P_2{}'(x_2{}', y_2{}')$, ... , $P_n{}'(x_n{}', y_n{}')$ can be computed using the following formulas:

$$(x_1{}', y_1{}') = (\frac{x_1 - x_0}{l_p} \times l_r, \frac{y_1 - y_0}{l_p} \times l_r)$$

$$(x_2{}', y_2{}') = (\frac{x_2 - x_0}{l_p} \times l_r, \frac{y_2 - y_0}{l_p} \times l_r)$$

$$\vdots$$

$$(x_n{}', y_n{}') = (\frac{x_n - x_0}{l_p} \times l_r, \frac{y_n - y_0}{l_p} \times l_r)$$

3.d.4 Path Planning

3.d.4.a Translating the data into a gridworld

First step of the path planning for the given mission is to discretize the space and create a gridworld. Discretizing the space is an arbitrary process, however translating the vehicle and mission data into a gridworld is more complicated. For vehicles, it is not as difficult since the vehicle must be placed in one of the cells even if it is occupying more than one. A vehicle cannot be in two cells at once, so choosing the cell closest to the middle is good enough, especially if the resolution of the gridworld is high. For reach missions it is also the same case, since a reach mission is made up of points that need to be reached.

However, missions such as no-fly zones are more difficult to compute. We need to set all the cells that have even a single pixel of no-fly zone in it to be a no-fly cell in order to avoid collisions. This also helps in the long run since this technique naturally creates a larger no-fly zone than the initial one and decreases the chances of collisions in case of errors. Of course, the amount it enlarges the no-

fly zone is inversely proportional with the resolution of the gridworld. Following is the algorithm that does all the computation.

```
Result: Creates and fills the gridworld with the given vehicles and missions
Inputs: vehicles; missions, gridworld;
for each cell in gridworld do
    gridworld[row][col] ← 1;
end
for each vehicle in vehicles do
    for each cell in gridworld do
        if cell.isInside(vehicle.pos()) then gridworld[row][col] ← -1;
    end
end
for each mission in missions do
    if mission is reachmission then
        for each cell in gridworld do
            if cell.isInside(mission.pos()) then gridworld[row][col] ← 2;
        end
    else
        for each cell in gridworld do
            for each pixel in cell do
                if mission.isInside(pixel) then gridworld[row][col] ← 0;
            end
        end
    end
end
```

Algorithm 2: Vehicle and Mission Translation to Gridworld

Discretizing the space and translating the data can be computationally very expensive depending on the resolution of the gridworld. Therefore, it is important to analyze the complexity of the algorithm. Before going into the analysis, we

have to define some variables that will be used in the analysis. The number of cells in the gridworld is C, the number of pixels in the image is P, the number of vehicles is V and the number of missions is M.

There are three consecutive for-loops and they have to be analyzed individually. The first for-loop only assigns a value to every cell and its complexity is $O(C)$. The second for-loop iterates over every vehicle and for each vehicle over every cell. Since this is a nested loop, the complexity is $O(VC)$. The third loop has an if else condition depending on the mission type. The computation for the reach mission is less complex compared to the others therefore, the worst case will be considered that none of the missions are reach missions. In that case, it looks like there is a nested loop. However, the algorithm iterates over every pixel on the image only once. Thus, the complexity is $O(MP)$.

Since these were all consecutive operations, the overall complexity of the algorithm is:

$$O(C + VC + MP)$$

This can be easily simplified to:

$$O(VC + MP)$$

In most of the cases there are much more missions than vehicles. Also, the

number of pixels is always more than the number of cells. Therefore, if we assume that $M > V$, which is the case most of the time, this simplifies further:

$$O(MP)$$

The complexity of the algorithm depends on the number of missions and the size of the image.

3.d.4.b Search Algorithm

In order to keep it simpler for this small example, I decided to have only one vehicle. I also only included reach missions of type "go always, no order". This meant that the algorithm has to compute a path from the initial point to every reach point and all the combinations between the reach points. But since if we know the path from point A to point B, then we can just reverse the order of the path and have a path from point B to point A. I used this feature to decrease the number of paths that need to be computed. At the end, if there are N reach points, the number of paths that needs to be computed is $\frac{N\times(N+1)}{2}$.

Then the question becomes how to compute the individual paths. I first wanted to use Dijkstra's Algorithm [11]. However, it is not necessary to compute a path to all the cells from a given cell. Most of the time the total number of cells will be much larger than the the number of cells that need to be reached. Therefore, A*

Search Algorithm [12] seemed like the best solution. I used a C++ implementation [13] of the algorithm. I implemented the general search algorithm that computes all the necessary paths as follows:

```
Result: Computes all the necessary paths
Inputs: gridworld, source, reachpoints;
for each destination in reachpoints do
    aStarSearch(gridworld, source, destination);
end
for each start in reachpoints do
    for each end in reachpoints do
        aStarSearch(gridworld, start, end);
    end
end
```

Algorithm 3: Path Computation

Computing many paths on a large gridworld can be computationally heavy and it is important to analyze the complexity of the algorithm. The complexity of the A* Search Algorithm is $O(E)$ [13] where E is the number of edges of the graph. In this case, every cell will have at most 8 edges. Each edge will have 2 cells attached to it and therefore will be counted twice. Therefore, the number of edges of our example is $O(4C) = O(C)$. This will be the time complexity for each path.

As previously discussed, the number of paths is $\frac{N \times (N+1)}{2}$ given that N is the number of reach points. Therefore, we can say that the number of paths is $O(N^2)$. Since we know the number of paths to be computed and the complexity of each

path computation, the complexity of the algorithm will be:

$$O(CN^2)$$

The complexity of the algorithm depends on the size of the gridworld and the square of the number of reach points.

3.d.4.c Complexity Analysis

The complexity analysis has been done for both algorithms in the previous part. Since they are executed one after the other, the complexity of the whole path planning software is:

$$O(MP + CN^2)$$

Simplifying this further would require many assumptions. Therefore, it is better to not simplify further. The complexity of the software depends on the number of missions, image size, gridworld size and the square of reach points.

3.d.5 Testing

Testing the software is a major part of its development since most of the bugs are found in the testing phase of the software. Also, if people other than the developer can test the software, it is possible to get valuable feedback from the users. Therefore, after finishing implementing the software, I decided to deploy

it as an application so that the other people in the group can easily test it on their computers and give feedback.

In order to deploy the application, I used windeployqt.exe which is a deployment tool designed to automate the process of creating a deployable folder. The folder contains the Qt-related dependencies (libraries, QML imports, plugins, and translations) required to run the application from that folder [14]. This tool was very handy because it made it possible to make small changes to the software and redeploy it quickly.

3.e Results

In order to show the results of the software, an example project has been created using the mission planning software. The example project can be seen below.

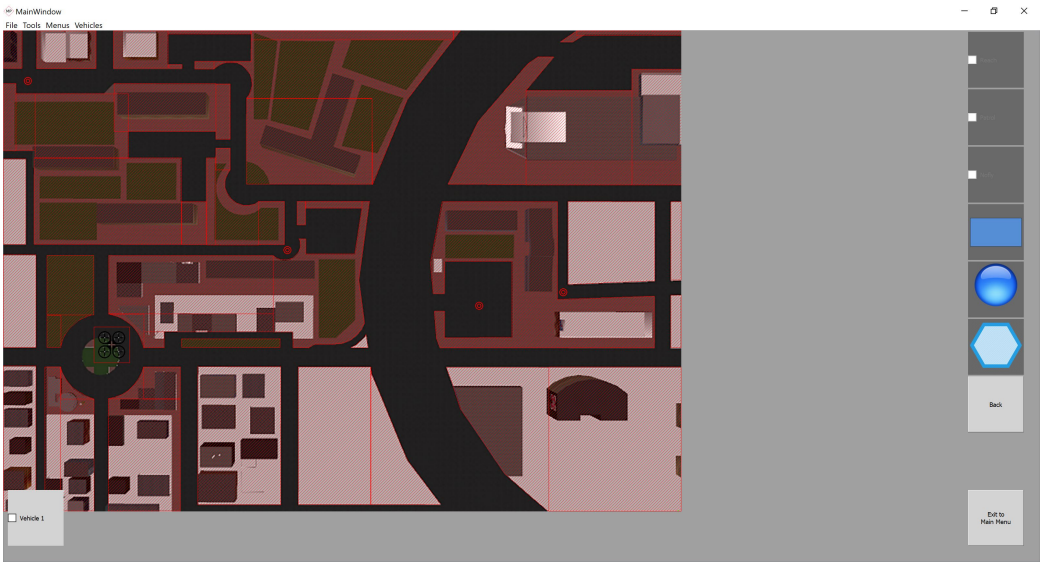

Figure 3.8: Example Project

Then, this project was loaded into the path planning software. As discussed before, this would be an automated process. However, for the demonstration purposes, the path planning software is used and setup manually. After the project was loaded, the user is asked to input the grid resolution.

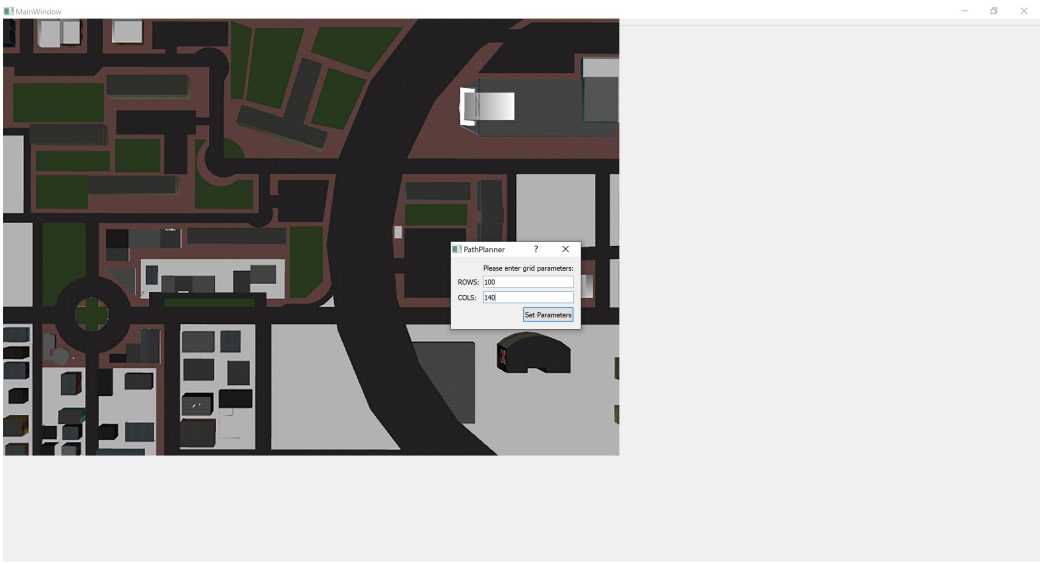

Figure 3.9: Gridworld Resolution Prompt

The created gridworld is then displayed. From here, it is possible to initiate the path planning computation.

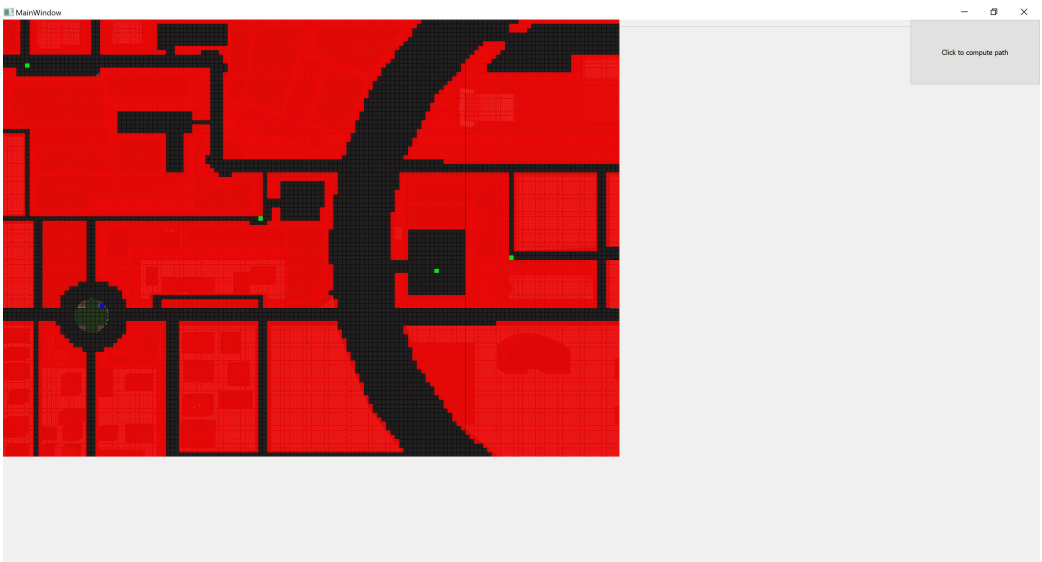

Figure 3.10: Gridworld Created

Some of the computed paths look like the following images. It is possible to display any path at any point or save the paths to a txt file that will be used to create the simulations.

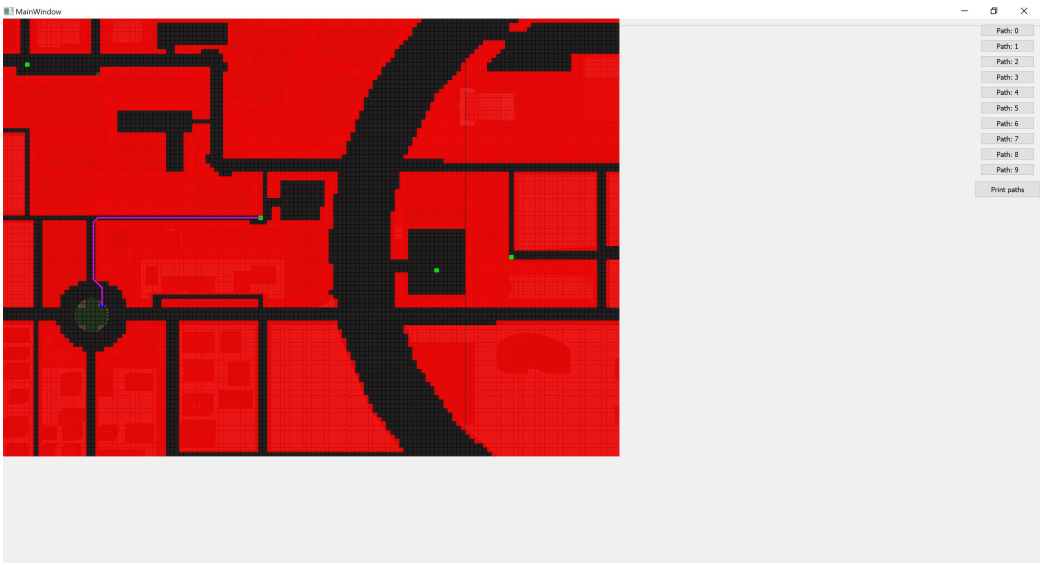

Figure 3.11: Example Path 1

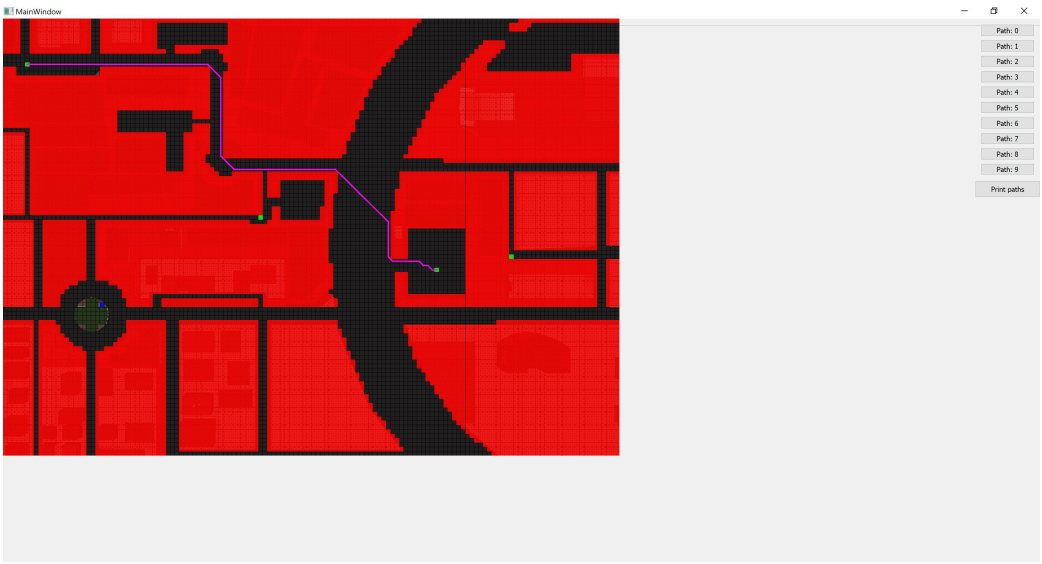

Figure 3.12: Example Path 2

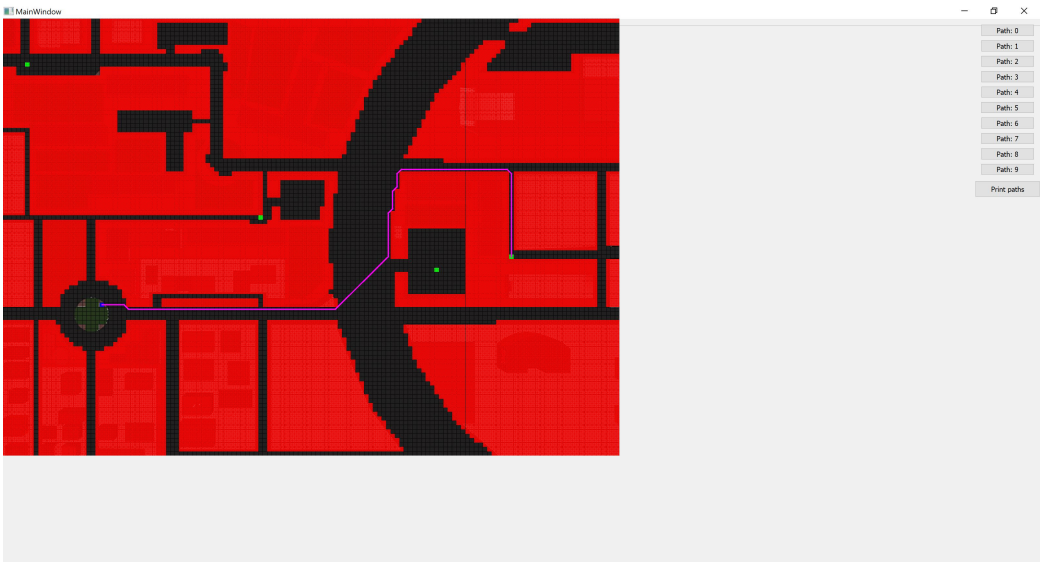

Figure 3.13: Example Path 3

The Gazebo simulation of the Example Path 1 (Figure 3.11) can be seen by clicking **here**.

4. Internship Experience: Observations and Analysis

This internship has proven me how much the classes I took during my first three years at Sabanci University were important. They created a very solid foundation for me to keep improving from. Since a large part of this internship has been software development, the knowledge I obtained from the CS courses I took have been crucial for me. Data Structures course has been especially important for me to handle all the data in such a large project in an efficient way.

Other than the CS courses, the Autonomous Mobile Robotics course has been very helpful with the methods I used in the path planning part of the project. But this project was not the only aspect where I benefited from the courses I had taken. Having the background in several different topics obtained from being a mechatronics student has allowed me to have discussions with the PhD students in the group. Being in such a large group and being able to discuss their research projects has allowed me to gain knowledge in diverse fields.

Even though I benefited from the courses I took, the experience and the responsibility in the courses and this project were very different. The main difference

was the amount of guidance given. In all the courses I have taken, I was guided through the topics step by step, with the help of lectures, homeworks and recitations. However, in this project I was given an objective and was told to reach that objective. I wasn't told how I should reach the objective or which steps to take. This was something new and challenging for me. But it has thought me a lot of things. I have learned how to solve problems on my own and do efficient research for solving the problems. Also, this internship has taught me valuable skills such as software development, path planning and most importantly working independently.

This internship has managed to meet and exceed all the expectations. The objective was to create a simple prototype for a future software. The result came out much better. It is a very polished and efficient software with the addition of the path planning software, which I decided to add since I had time left. All the PhD students and the professor were very excited with the results they saw during my presentation, and they are looking forward to incorporating this software to their projects.

I am sure that this internship will be very beneficial for me in reaching both my education goals and my career goals. Having met so many people and having established good relationships with all of them will be very important for my future graduate school applications.

5. Conclusion

I have done my internship in the Autonomous Systems Group at the University of Texas at Austin. The objective of the project was to develop a prototype for a graphical user interface that would allow anyone to create missions for robots and automatically synthesize controllers. Six weeks of the internship duration were spent on developing the graphical user interface. I have also developed a path planning software to demonstrate the future steps of the project. The path planning software outputs the necessary paths for a given mission. A simulation has been created using Gazebo Simulation to show what the output of the whole software would look like.

This internship has been very beneficial for me as it made me learn software development and path planning skills. It also taught me to solve problems individually and guide myself through the steps of a project. I managed to meet and develop good relationships with many people in the group I was working in which will be very helpful in graduate school applications. Overall, this was a very important experience for my career.

6. Recommendations

The most important recommendation I would give to future interns is to start searching as early as possible. I started searching for an internship in March and it was very difficult for me to find an internship. I would suggest the students to start applying by December or January latest. Starting early gives the student both more opportunities and more time to think.

Even though I started applying late, I was lucky enough to be accepted for this great internship. Anyone who is planning to do their internship in Austin, I would strongly encourage it. Austin is a very nice college town. University of Texas at Austin is a very good opportunity to work on interesting projects and meeting many people that would be crucial in graduate school application.

Autonomous Systems Group is a large group mainly working on machine learning, formal methods and autonomous robots. All the people in the group are very welcoming and are willing to help with any questions. They also like discussing about their projects and that helped me broaden my horizons about the topics I would like to work on in the future.

7. References

[1] University of Texas at Austin. Overview. https://www.utexas.edu/about/overview. [Retrieved on 28 Aug 2019].

[2] University of Texas at Austin. Facts & figures. https://www.utexas.edu/about/facts-and-figures. [Retrieved on 28 Aug 2019].

[3] University of Texas at Austin. Mission & values. https://www.utexas.edu/about/mission-and-values. [Retrieved on 28 Aug 2019].

[4] Oden Institute for Computational Engineering and Sciences. Welcome to the oden institute. https://www.oden.utexas.edu/about/welcome/. [Retrieved on 28 Aug 2019].

[5] Oden Institute for Computational Engineering and Sciences. About oden institute. https://www.oden.utexas.edu/about/. [Retrieved on 28 Aug 2019].

[6] Oden Institute for Computational Engineering and Sciences. Autonomous systems group. https://www.oden.utexas.edu/research/centers-groups/asg/. [Retrieved on 28 Aug 2019].

[7] Ed Parrish. How to document and organize your c++ code. http://www.edparrish.net/common/cppdoc.html, (n.d.). [Retrieved on 16 Jul 2019].

[8] Bjarne Stroustrup and Herb Sutter. C++ core guidelines. https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#Ri-array, June 2019. [Retrieved on 16 Jul 2019].

[9] Iman Sahebi. How to check if a given point lies inside or outside a polygon? https://www.geeksforgeeks.org/how-to-check-if-a-given-point-lies-inside-a-polygon/, (n.d.). [Retrieved on 11 Jul 2019].

[10] Moshe Shimrat. Algorithm 112: position of point relative to polygon. *Communications of the ACM*, 5(8):434, 1962.

[11] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[12] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[13] Rachit Belwariar. A* search algorithm. https://www.geeksforgeeks.org/a-search-algorithm/, (n.d.). [Retrieved on 7 Aug 2019].

[14] Qt Documentation. Qt for windows - deployment. https://doc.qt.io/qt-5/windows-deployment.html, (n.d.). [Retrieved on 5 Aug 2019].

Appendices

A. Coding and Documentation Standards

- Variable Names: Use only letters. Capital letters are used as separators. All other letters should be lowercase letters.
    - o int localVariable;
    - o double radius;
- Constant Variable Names: Use only capital letters. Underscores are used as separators.
    - o const int CONSTANT_VALUE = 20;
    - o const double PI = 3.14;
- Global Variables: Avoid using global variables.
- Function Names: Use only letters. Capital letters are used as separators. All other letters should be lowercase letters.
    - o double circleArea(double radius);
    - o int min(int x, int y);
- Class Names: Use only letters. Capital letters are used as separators and as the first letter. All other letters should be lowercase letters.
    - o class ImageLabel;
    - o class Circle;
- Curly Braces: Starting curly braces should always be in the same line of the keyword rather than below it.
    - o if (condition) {
    - o //code
    - o }

- Spacing Around Operators: Leave a blank space around every operator to make the code more readable.
    - o int x = 5 + 3 * 4;
    - o cout << "Hello world!";
- Single Line Comments: If there is a need to explain a concept that is not clear from the function comment block or the code itself, there can be a single line explanation.
    - o /// Following if statement checks whether the circle pointer is nullptr
    - o if (circleptr == nullptr) {
- Line Length: Lines shouldn't be longer than 100 characters to make the code more readable.
- File Comment Block: Every file must have this comment block at the beginning. Comment block should start with /** and end with */ and these symbols should be on their separate lines. All the comments written inside must be indented one level with respect to the start and end symbols. First line after the start symbol is project name, second one is file name, third one (it can be more than one line if the description is long) starts with Purpose: and continues with a very short summary of the file. After that, there should be a blank line followed with three lines. First one will start with @author and be followed with the author's name. Second one will start with @version and will be followed with the version of the file. Third one will start with the @date and will be followed by the date the file was last edited.
    - o /**
    - o User Interface Project
    - o circle.cpp
    - o Purpose: Handles area calculations for circle shapes
    - o @author Edin Guso
    - o @version 1.3
    - o @date 15/07/2019
    - o */
- Function Comment Block: Every function declaration must have this block before it. Comment block should start with /** and end with */ and these symbols should be on their separate lines. All the comments written inside

must be indented one level with respect to the start and end symbols. First line (it can be more than one line if the description is long) after the start symbol is the description on how to use the function. It is followed by a blank line. After the blank line there must be a line for every parameter which start with @param<paramName> followed by a short description of what the parameter is. After all the parameters are listed and if the return type of the function is not void, there should be a line starting with @return followed by a short description of what the returned value is.

```
o /**
o     This function is used by passing
o     the radius of the circle and
o     it returns the area of the circle
o
o     @param<radius> Radius of the circle
o     @return Area of the circle
o */
o double circleArea(double radius);
```

- Class Comment Block: Every class/struct must have this block before it. Comment block should start with /** and end with */ and these symbols should be on their separate lines. All the comments written inside must be indented one level with respect to the start and end symbols. First line (it can be more than one line if the description is long) after the start symbol is the description on what the class/struct is for. It is followed by a blank line. After the blank line there must be a line for every variable which start with @variable<variableName> followed by a short description of what the variable is.

```
o /**
o     This class stores information
o     about the circle object
o
o     @variable<radius> Radius of the circle
o     @variable<center> Center point of the circle
o */
o class Circle;
```