F1/10 YellowTails

Final Report

Version 1

Bowen Boyd
Hanyue Wang
Kyle Watson
Jordan Wright

Faculty Mentor:
Isaac Shaffer

Project Sponsor:
Truong X. Nghiem, Assistant Professor, SICCS, NAU
Trong-Doan Nguyen, PhD Student, SICCS, NAU

05/03/2020

Table of Contents

1. Introduction

F1/10 Yellowtails is a team that was formed with the goal of improving access to the F1/10 autonomous racing platform. The members of F1/10 Yellowtails are Bowen Boyd, Hanyue Wang, Kyle Watson, and Jordan Wright. We are creating a new autonomous racing interface system called RosConnect. The project is sponsored by Dr. Truong Nghiem and his graduate research assistant Doan Nguyen. Dr. Nghiem is the Director of the Intelligent Control System Lab, or ICONS lab, at Northern Arizona UniversityIntroduction. At the ICONS lab our clients are creating new theories and algorithms for intelligent and high performance control systems. This includes developing solutions for the transportation industry. Currently, our clients are focused on improving algorithms for self driving cars.

Self-driving cars have the potential to be the future of the automobile industry. The ever-growing need for greater road safety, reduced road congestion, and environmental stability means that vehicle autonomy advancements are particularly vital for society. This new industry needs more engineers to help extend outreach to the general population and solve the problems that are holding it back. However, there are currently no high-school programs that provide autonomous technology education to students, especially those with limited coding experience. This lack of access to the emerging field of autonomous technology results in potential engineers who choose other fields and under-educated leaders of tomorrow.

Dr. Nghiem and Doan Nguyen are creating a summer camp for high school students that focuses on autonomous vehicles that lowers the barrier of entry to this technology. Through this new learning opportunity, they hope to attract high school students interested in STEM and ultimately recruit these high school students to study at Northern Arizona University. The F1/10 autonomous racing platform is an RC car that is one-tenth the size of a formula one race car. It is powered by an Nvidia Jetson computer onboard the RC car and has sensors that are found in real self driving cars. Our clients want high school students to receive hands-on experience with the F1/10 RC cars to understand how autonomous cars work.

The problem and premise for the creation of this project, is that the F1/10 autonomous platform, in its current state, is complicated to operate. The Nivida Jetson located on the vehicles runs the Ubuntu Operating System. The Ubuntu OS is then used to run the Robotic Operating System (ROS) which controls the car. Further complications include the command-line, ROS's need to source the workspace directory, ROS's need to use the Catkin compiler to compile all C code, and ROS packages. Additionally, ROS packages require two files to be correctly set with dependencies and parameters. So, changing one setting may require changing multiple files in different directories. Even further, if a user happens to overcome the previous barriers of

complications, there still exists the problem of shutting down the vehicle while it is running in order to prevent accidents and damage.

To clearly convey the problems at hand, we present the following list of our three problems:

1. The system controlling an autonomous vehicle is overly complicated to operate.
2. Configuration prior to operation requires changing multiple files in different directories, i.e., there exists a "disconnected configuration".
3. There does not exist an emergency stopping mechanism to halt the operation of the vehicle.

Dr. Truong Nghiem, Doan Nguyen, and F1/10 Yellowtails aim to alleviate the problems listed above by constructing and implementing RosConnect, a Graphical User Interface (GUI) for driving autonomous F1/10 vehicles. RosConnect gives autonomous technology experience to students by providing an interactable configuration window. In this window, students will choose from four options: perception, mapping, planning and racing strategy. These options determine what variables are passed to a simulator for the car and eventually the car itself. Thus, the options selected determine the car's behavior and capabilities. Furthermore, this "smart" configuration window prevents all incorrects sets of chosen configuration options, and any options chosen before closing RosConnect will be saved and loaded on the next startup. Lastly, RosConnect provides a logging window that communicates major processing steps which are handled in the background, including but not limited to roscore startup, simulation startup, and program termination.

To clearly outline how RosConnect solves each of our previously listed problems, we present the following list of solutions:

1. Operating an autonomous vehicle is made simple through RosConnect's intuitive design that only requires the user to choose configuration settings by clicking buttons on the GUI, and then clicking a "run car" button.
2. Once a user has chosen a set of configuration options, RosConnect automatically sources and links the necessary packages to launch a vehicle operation, eliminating the need to change files in different directories.
3. RosConnect contains an emergency "stop car" button that can be clicked by the user to halt further vehicle operations.

With this intuitive interface design, RosConnect succinctly addresses each of our client's problems and ultimately gives access to autonomous racing to every high-school student, irrespective of her or his coding competency.

2. Process Overview

From the beginning, when we got assigned this project to work on, planning was a big part of how we were going to be able to get the best result possible.

The first half of our development process was made up of planning and requirement aciquision. We began by talking with our clients to see what exactly they were looking for in a solution. After we had many discussions with them we got all of the requirements that they needed to be in their solution that we created.

In order to complete all of our requirements we had to take time testing different technologies that we are going to have to use to complete them. After a couple of weeks of trying different technologies that would allow us to be able to create a solution fairly easy and allow us the support to use other dependencies. We had to come up with a way to create our GUI, how to create our documentation, test all of the functions, and finally how we are going to communicate. We decided to use PyQT5 for the creation of GUI, Sphinx for the auto documentation, PyTest as our testing library that we are using and finally using WIFI and ssh as our communication method. At this point we had finished all of our research and it was time to write our Technology Feasibility. In this document we are proving that all of the technologies we listed above will work together and are the best fit for our solution.

Now that we had this document created it was time to start creating the solution to our problems that we were facing. We started by creating our first prototype that was going to have all of our main features. That way we could just build on top of what we had for this first prototype to enhance the features and make it better. When we showed the demo we were very happy that everything that our clients were wanting was basically all built in already. After we got done with the demo we started fixing some issues that came up during our demo before the start of the Spring semester. We had also been researching new ways to improve our Graphical User Interface.

At the beginning of the Spring Semester we sat down with our clients again to talk about what we have and what they would like to see now that they have seen what we have built to this point. With this brought new changes that required us to remove some features as well as adding in new ones. So we had to come up with a plan again to make sure that we are hitting all of the requirements that our clients would like us to have.

Once we had this new plan we started implementing those changes that our clients wanted us to remove and add. As we went through and did this we started adding in functionalities that we

were looking into at the end of the semester. At this point it was time to create our Alpha prototype design so we could show it off.

With everything going on with this pandemic we had to completely switch to an online setting which threw a big curve ball at us since we were no longer meeting with our clients in person. This caused some miscommunications, and us not being able to work with our car to ensure that everything we were doing actually still works with the car. Since all of this started after our Alpha demo we had to work on refinements all online and couldn't test it without the data. This also led into testing. This was difficult since we were no longer able to test it with our car along with it was so much harder to get users to test our product.

This was our development process at a glance, now let's take a more detailed look at the product itself, starting with the Requirements for the product.

3. Requirements

Our requirements acquisition from our weekly 2 hour long meetings in the first semester with our clients. As the meetings went along we got enough requirements to create our Requirements Document that discussed in much detail all of our requirements that we have to follow both in Function, Non-Functional and finally environmental. The major requirements will be discussed below.

Functional Requirements

In this section, we will discuss the functionalities that define our system and its components. We will outline our two high level functional requirements as Graphical User Interface and Configuration Package. Furthermore, each of these high level sections will contain more detailed lower-level requirements.

Our first functional requirement is our front end part of the application, the Graphical User Interface. This is what the user is going to interact with. Eight major subsections of this requirement include a communication toolkit, a console, a run simulation, a run car, a profile, a load profile, a save profile and finally a clear profile.
The first subsection is the communication toolkit, which is what allows us to communicate between the car and GUI. This communication toolkit consists of two different functionalities that we are going to use. The first one is a script on the car that is listening for connection and will stop the car if a connection is lost, it shuts down the vehicle. The second part of the communication toolkit is a stop button on the actual GUI that allows us to press this and it will

stop the car from running. In our GUI the stop car button is right under the Start Sim button. As soon as the user presses this button in our console there will be a message displayed letting the user know that the button has been pushed and will say successfully stopped the car.

The second subsection is the console, which allows us to input any major information that we think is necessary for the user to see. This gets information from all parts of our GUI, mainly the star car, stop car and start sim buttons. Another major part is when the user tries to load/save a profile. This just gives the user another way to look at it without having to use the command line.

The third subsection is the run simulation, which gives us the ability to run any of the selected configuration options (to be talked about after all of the GUI functional requirements) in a simulation. This simulation allows users to test their configuration choices to see how they are going to run so they do not just go straight to the car. This will help the user to understand what is going to happen. This is sourced from the University of Pennsylvania F1/10 simulation program.

The fourth subsection is the run car, which allows us to connect to the car and run the script to start the car. The run car button will function as a tool to transfer the configuration setup, predefined by the user, to the vehicle so that the vehicle may begin running. The color of the button will be green to ensure a higher likelihood that the user recognizes this button for starting the vehicle.

The fifth subsection is the profiler, which is the configuration setup created by the user. This will hold information on all of our parts of the configuration. This is what gets passed to both the run simulations and the run car. These are the parameters that will determine how the car/simulation will act.

The six subsection is the load profile, which as it sounds loades a saved profile back into our GUI. This is done by us checking the configuration to make sure that all of those options are still viable options, if it is they get loaded into the GUI and as they get passed in they get selected. This allows the user to go through and load in a previous profile that they already created.

The seventh subsection is the save profile, this is what allows the user to save their selected options to any type of file to be used later on. This is important because the user is now allowed to create a profile and later on if they want to use that profile again they are able to. This will be helpful when the user has a lot to choose from and they already had a working simulation or car how they would like.

The final subsection for the GUI is the clear profile, which goes through and clears all of the options from the GUI and unselects them all. This is essential that way the user doesn't have to go through and deselect all of the options. This will allow for users to quickly clear the selections if they wouldn't want people seeing this.

Our second functional requirement is the Configuration Package. This has been implemented so the user is able to select what they would like to see. There are only two subsections, which are a configuration window and the configuration file.

The first subsection for the Configuration package is the configuration window in our GUI. This is the part of the GUI that has all of the available sections that the user can choose to run the car/simulation from. There are four sections to this window and they are: Racing Strategy, Perception, Mapping and planning.

The final subsection is the configuration file itself. This is the file that our GUI will read from and how it creates the window of our GUI. This file is in the form of a YAML file. We worked with the clients to come up with the easiest way for us to integrate with our GUI.

Non-Functional Requirements

Non-functional requirements are stated as a requirement that can be used to judge the system as a whole instead of specific behaviors. We have three main non-functional requirements, communication toolkit, Graphical User Interface which has many subsections, and finally configuration package.

Our first non-functional requirement is the communication toolkit, which we will describe all of our qualities. For this functionality to work, we are going to have the script running on the car. While this is running its checking to see if there is still a connection with the car. This should be done by checking constantly that way if it stops it will know right away and act quickly . If at any point it returns the connection between the Raspberry Pi and the car it should stop the car. This will be done immediately. If the user sees that something isn't correct with the actual car and wants to stop it with the stop car button on the GUI it will work like the script does stopping the car immediately.

Our second non-functional requirement is the Graphical User Interface, which is what the user will be interacting with. We will summarize all of the non-functionals as there are five subsections. For our stop car we are going to want the car to stop immediately when the user notices something wrong and pushes the stop car button or when the script stops the car. We

need this to end immediately so the car doesn't break or get damaged because the car ran into some object. As building the solution we are going to remember to make it as usable as possible since high school students will be using it.

The third and final non-functional requirement is the configuration package. Users will only see the front end designs of the configuration information. The team is going to handle all of the configuration settings in the backend. This is because we do not want the user to have to write any of the code for this section. As the user is done editing the configuration we are checking to make sure there are no errors. If they select one option and it is not compatible with the other options you are not going to be able to select it. When the user is done editing the configuration and click save it should automatically handle this and as soon as they press save. When we go to transfer it to the car it will take 5 seconds to transfer it. This is going to be saved into either a YAML or XML file that way if the user wants to reconfigure and only change a few things they do not have to restart everything.

Now that we have explored the requirements of our application, we will move forward and explain the architecture and implementation of our final product.

4. **Architecture and Implementation**

Within this section we will cover in-depth the architecture of the product in order to paint a general picture of how the software works regardless of the changes it may experience in the years to come.

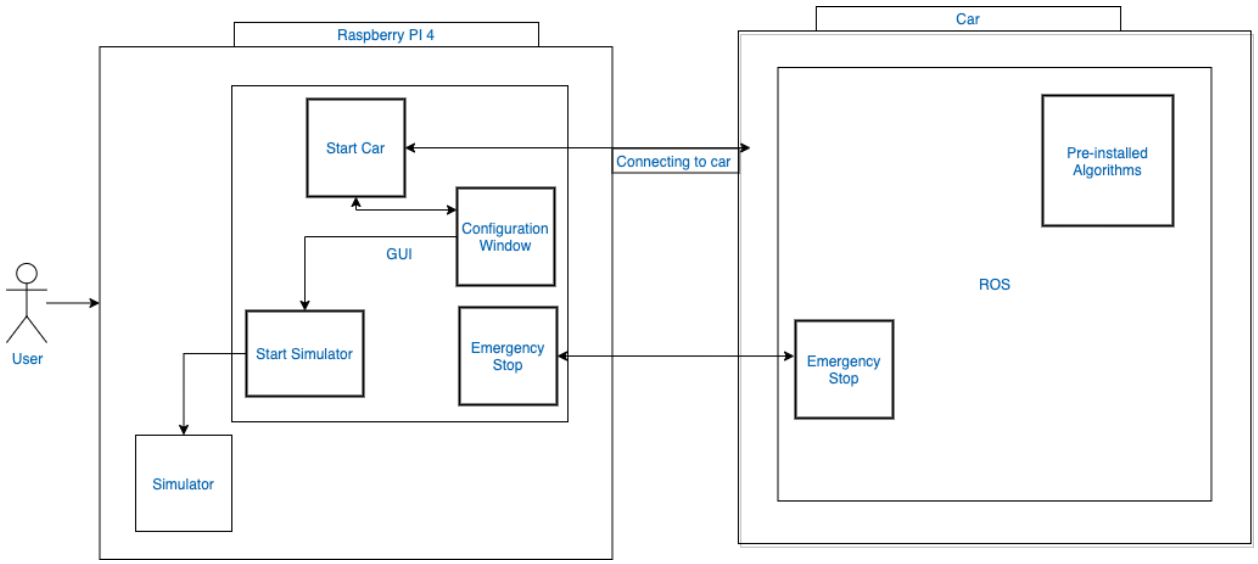

Figure 1: Implementation Overview

The user will be interacting with the Raspberry Pi 4, which will hold all of our GUI information. Linux ubuntu is going to be installed on the Raspberry Pi. The user will mainly interact with the configuration window. Once the user selects what they would like in the configuration window they are allowed to then either select the start car which will communicate over to the car and send a script or the start simulator which starts the simulation to see how the system works. The recommended way the user goes about is starting the simulation first before sending it to the car. Once the user clicks start car, it sends the files needed to the car. On the car is ROS which the car runs on, an emergency stop and all of the algorithms that the users are going to be able to select from. Once the car successfully gets the files it will run the car for the user to look at. As noted there is an emergency stop on the car as well as the actual GUI. This is there to prevent anything bad from happening to our car. With this is functionality on both the car and the GUI we are able to stop it on both ends. Now that we looked at the high-level overview introduced, we will take a closer look at each of the components in order to understand the finer details of the software.

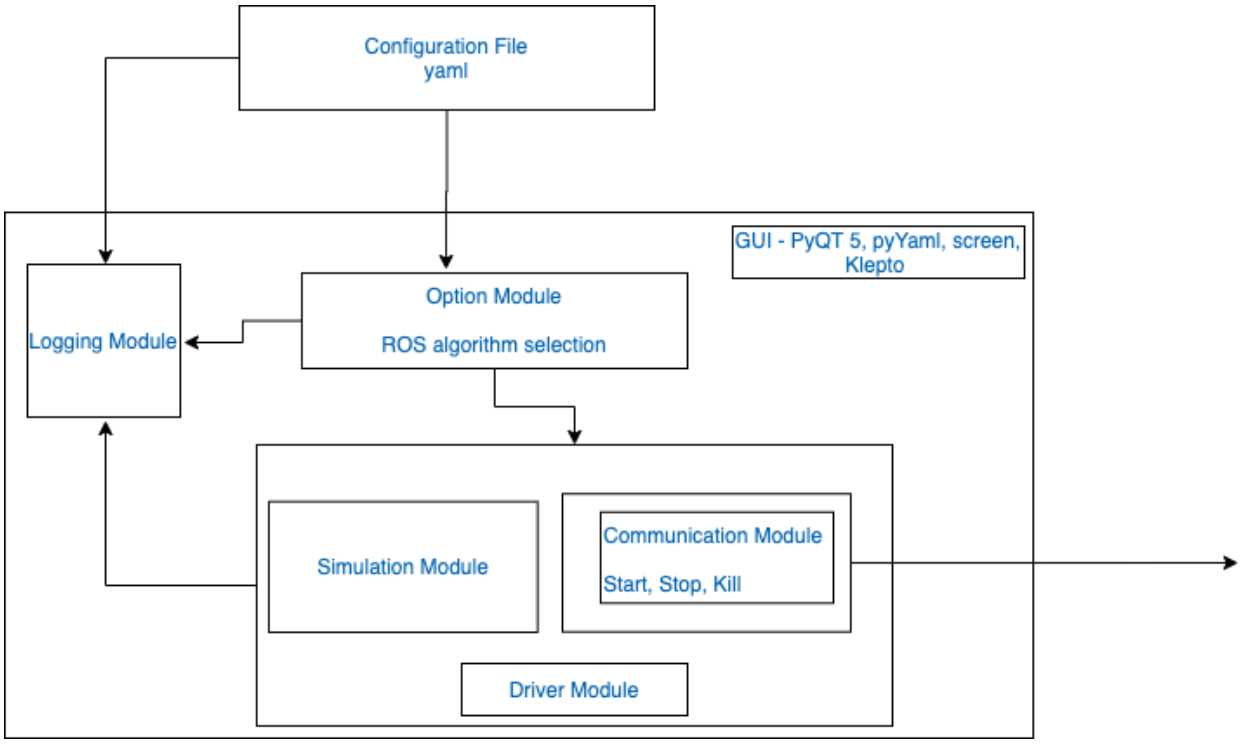

Figure 2: Architectural Software Overview

As you can see in Figure 2, there are two different major components. It all starts with the Configuration File that goes into our GUI. Inside our GUI there are going to be three major components. We will discuss all of these features following:

- **Configuration File:** This is the file that the clients will provide to the user. So the user doesn't have to write any code to make sure that it will correctly be uploaded to the GUI. This is made using a YAML file.

```
# config.yaml - for creating a configuration window


# used to keep track of different configs and relate to saved profiles
Version: 2020021

Racing Strategy: #module name
  variable: strat #varname that gets passed in string
  choices:  #list of buttons and their values
    Racing_1: #button value passed to the string
      title: Vroom Vroom #title of the button in the GUI
      dependencies: Perception_1, Mapping_1, Planning_1
    Racing_2:
      title: Lightning McQueen
      dependencies: Perception_2, Mapping_2, Planning_2
    Racing_3:
      title: Manual Override
      dependencies:
```

Figure 3: Configuration File Outline

  - The very first thing that you see is there is a version number. This is important because when the clients make changes to this file they update the number so they are not able to load in only the current information.
  - The configuration file also holds a set structure that the file must follow in order to be able to work with our GUI. In Figure 3, you can see the structure.
    - The first thing is the Module name, as you can see this is the first thing that is important for the section. Everything that falls under this is guaranteed to be in the correct module.
    - After the module name is the variable that is going to be passed to the launch file. Whatever option is selected is going to be passed to this variable.
    - Choices, this is the list of options that are available to the user to select.
      - Button value that is passed to the string.
      - The next thing that you see in Figure 3, is the title that gets put into the GUI.

- The following section is the dependencies. In Figure 3, you can see that all of the options listed here when you click on the name of the option all the dependencies listed are the only ones able to be selected.
- After dependencies is the description, what this does is the clients are able to put a description of what the option does. When the user hovers over the option and the description will appear.
- The next part is only for Racing Strategy. This is where the sim are put so they are dynamic.

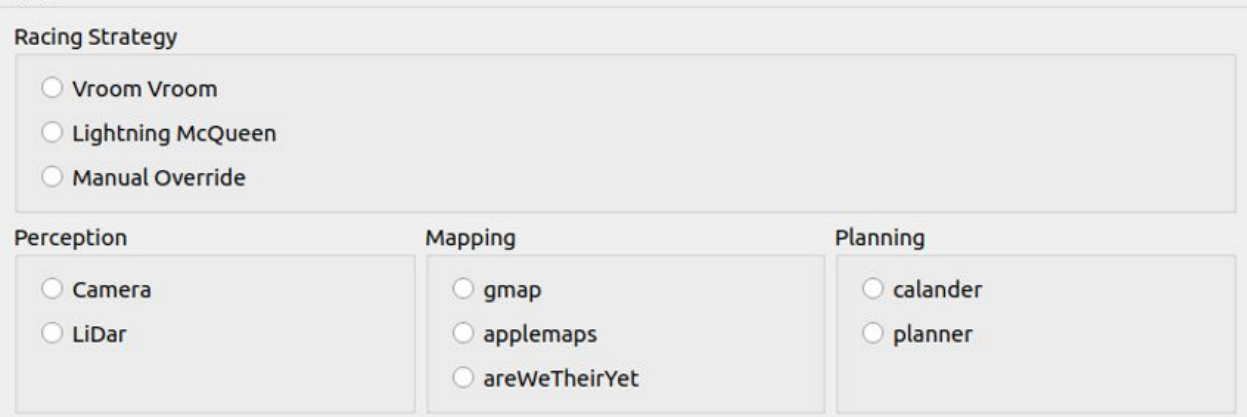

Figure 4: Configuration Window

    - The GUI will parse the configuration file and create the window as shown in Figure 3. This goes through and adds all of the headers to begin and then from there goes through adding all of the selectable options. This is dynamic so the clients are able to put any number of options in and the GUI will handle it. In this case there are three options for all except Perception which will only have two options in general.
- **Graphical User Interface (GUI):** This is the main part of our solution. As the user will interact with all of the main functionality here. The GUI is made up by using PyQT5, pyYAML, screen and finally klepto.
    - The first aspect of the GUI we are going to look at is the options module.
        - This is where the information from the Configuration file gets parsed into. This is the part of the program that allows the user to select which ROS packages it would like to use on the

car/simulator. When the user selects all of the options they want it gets sent to the driver module.

- The second aspect of the GUI is the driver's module. This is where all of the actions happen for the car.
  - Inside the driver module there are two separate modules. The first one is the simulation module. The second module is the communication module.
    - Simulation module:
      - This is the module that holds all of the simulation information. In this is the start simulation that will load the specified simulation option that the user is wanting to see. This can be a 3D sim, 2D wall follower, or a 2D keyboard simulation.
      - What the user will see in this simulation is what the car should run like.
    - Communication module:
      - This module has three parts to it. Those are the Start, Stop and kill switch.
      - When the user clicks star car this sends a script to the car that holds all of the parameters for it to know what exactly to run. This also can print them out if you would like.
      - When the user presses the stop car button it reports that and then will ssh into the car and stop the car.
      - The kill switch is on the actual car. What this is looking for is when the car loses connection to the GUI. If this happens we are able to stop the car and disconnect. That way everything is safe.
    - Logging Module:
      - This module is what takes in any important information from the other two modules in this section. They also pull any information about loading/saving of the profile. This gives the user information about what is going on. This includes starting the car, stopping the car.

Now that we have taken a look at how the system exactly works let's go into what the auto documentation gave us. For this we have cleaned it up a little bit to match our website. The images that are going to be provided below in Figures 5-7 are taken from our team website.

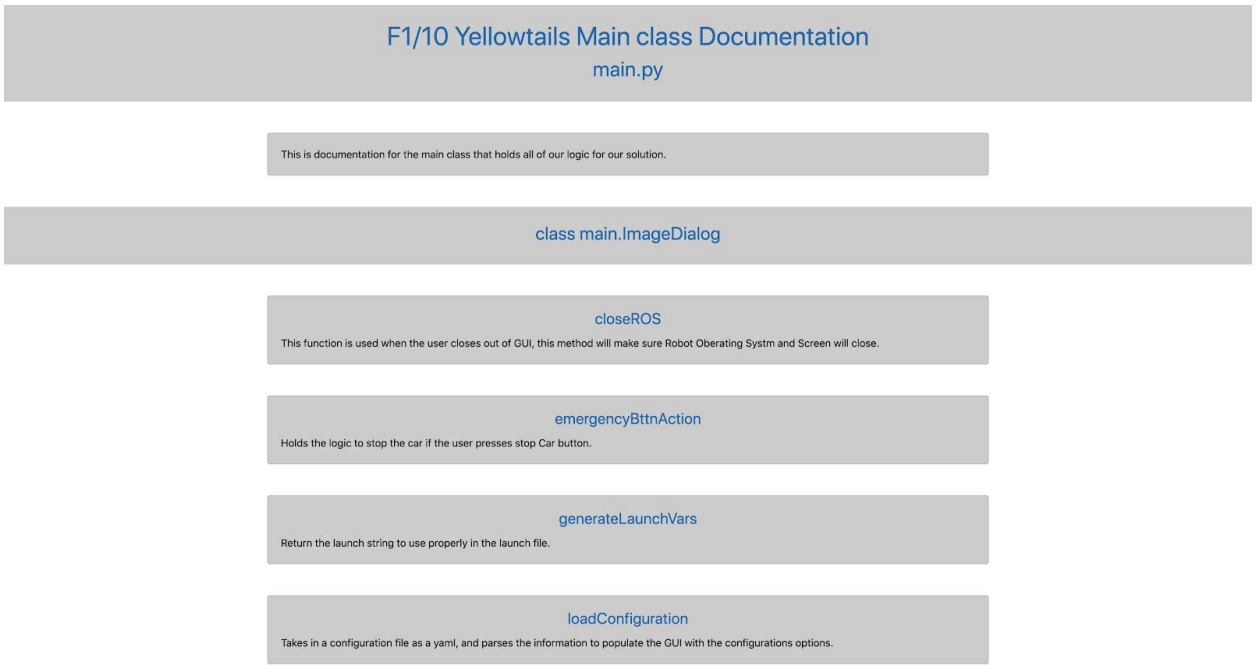

Figure 5: Documentation from Closing ROS to loadConfiguration

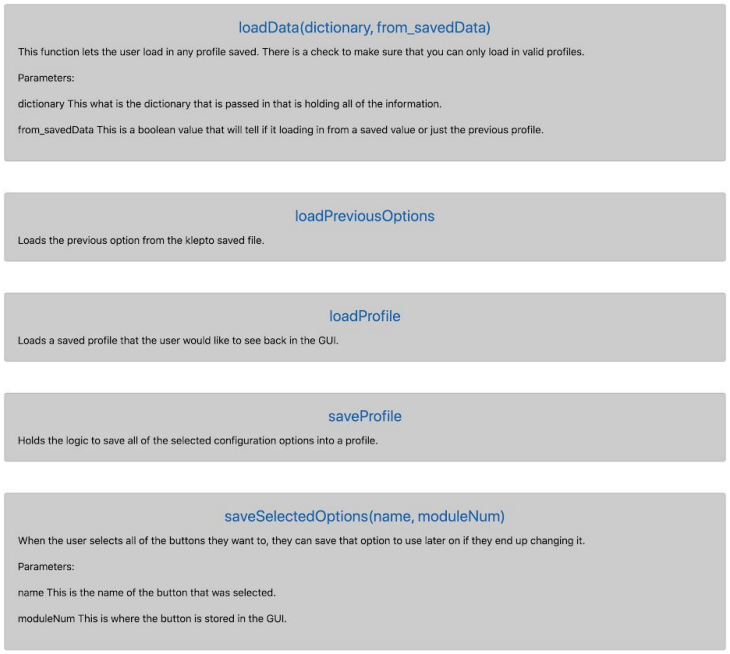

Figure 6: Documentation from loadData to saveSelectedOption

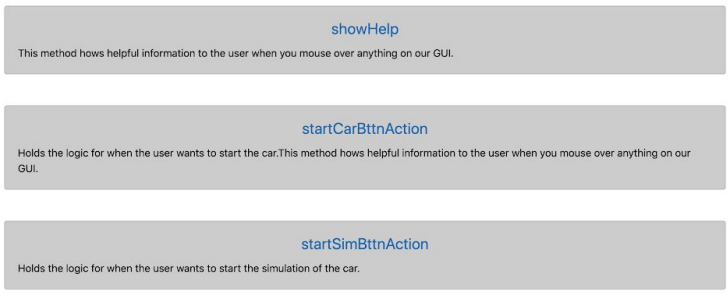

Figure 7: Documentation from showHelp to startSimBtnAction

This is also given in the code under the doc/build/html folder. We just made it match that way we could easily put it on our team website. In the following section we will go over all of the testing that we have completed.

5. **Testing**

Due to the fact we had weekly meetings with our clients we took full advantage of the Agile development cycle. We would show our clients what we had done during the past week and use their feedback to change the software to their liking. This led to a lot of the testing being done as we were developing and our clients who would be one of the main users got to give us their input throughout the development process. We had wanted to use one of our clients EE labs to get more user tested feedback but due to Covid-19 this did not happen. Most of our written tests were scripts that allowed us to see what was being sent to the RC Car or simulator from our GUI. There are little to no written unit tests, and this was one of the main takeaways we learned. That is, that we should have spent time developing these unit tests from the beginning of development. Thankfully, due to our weekly meetings with our clients, we can verify that our GUI in the end did behave and function as desired. In the next section we will discuss our project timeline which covers the past year (Fall 2019 - Spring 2020) of capstone.

6. **Project Timeline**

Below in Figure 8 is our schedule for the Fall 2019 semester when we were first introduced to our project. In that time we gathered information from our clients and came up with requirements guidelines that our client signed off on. We then did some research and trials for the tools and systems we wanted to use to meet those requirements. We ended the Fall 2019 semester with a Prototype demo that met about 75% of our requirements.

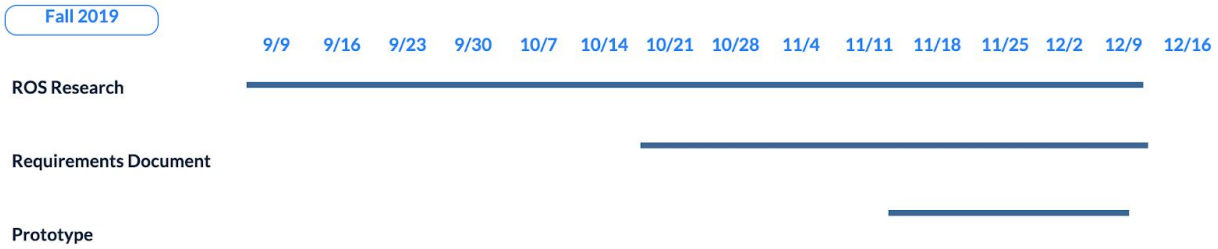

Figure 8: Fall 2019 Capstone Development Timeline

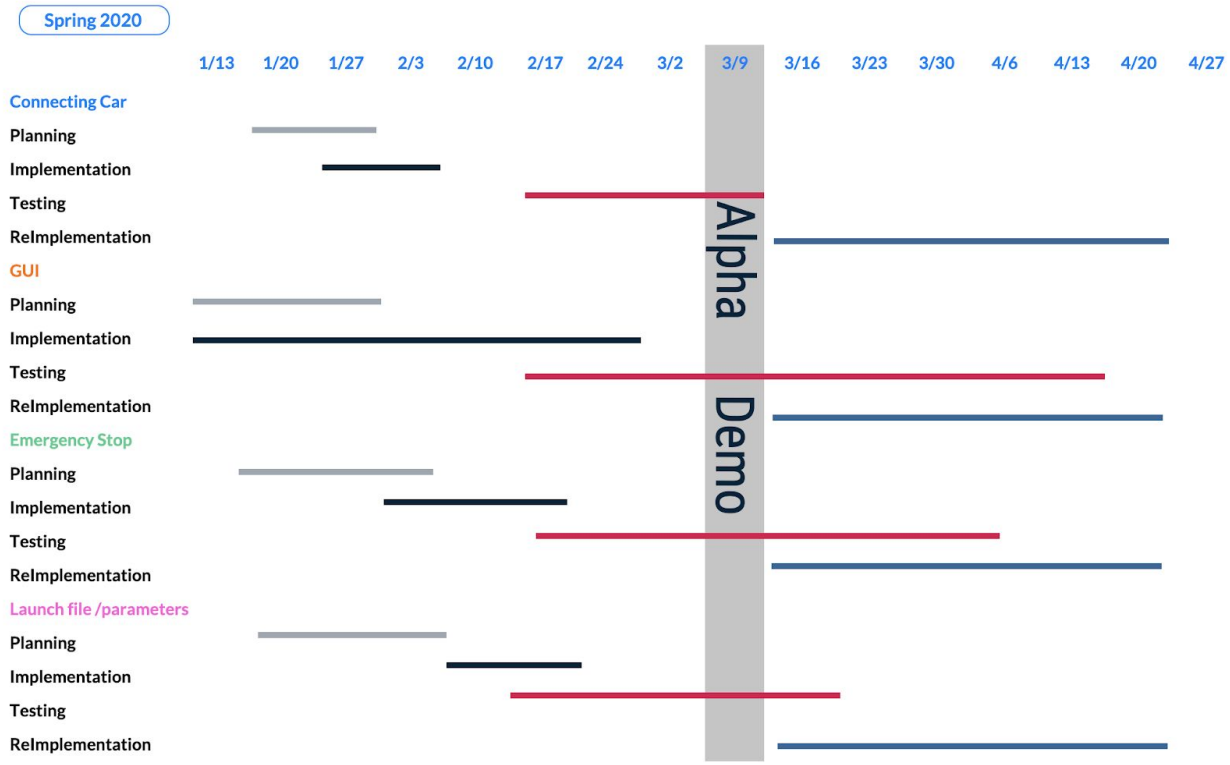

Figure 9: Spring 2020 Capstone Development Timeline

After coming back from break we agreed to keep meeting weekly with our clients as you can see from our Spring 2020 schedule in Figure 9. The first half of the semester we continued development and before spring break had an alpha demo that met all the requirements. During the rest of the semester we added additional features and polished the look and user experience of our software based on our clients' feedback.

Now that we have gone over our project's schedule the following section is going to take a closer look at what could be done in the future to make our product better.

7. **Future Work**

Even though our team has created a working solution we would like to see a few changes made to make our solution even better. The first thing that we would like to see in the future is for the next team to implement the packages that our clients would like to use. We were unable to do this so this would be the next step for our solution. The second part of what we would like to see is them adding tests to fully ensure that everything works correctly even though we worked with the clients to make sure everything worked correctly. The final part of the project that we would like to see is them adding some error checking to make sure that if someone misspells a part of the configuration or if someone deletes some of the parts it doesn't throw an error. This will help prevent any errors on the command line that the user's shouldn't have to worry about.

8. **Conclusion**

Autonomous vehicles are quickly emerging as the future of the automobile industry. However, safety concerns involving this technology require further innovations from future generations. Moreover, there are currently no high-school programs that provide autonomous technology education to students, especially those with limited coding experience. This lack of access to autonomous technology results in under-educated future autonomous innovators. That is why Dr. Nghiem and Doan Nguyen are creating a summer camp for high school students that focuses on racing autonomous vehicles.

The problem is that the F1/10 autonomous platform, in its current state, is complicated to operate. This means that many students who lack the necessary coding experience are unable to participate in our clients summer camp program. So, Dr. Nghiem, Doan Nguyen, and the F1/10 Yellowtails aim to make this platform more accessible and easier to use with RosConnect, a Graphical User Interface (GUI) for driving autonomous F1/10 vehicles. With it's intuitive design, this new interface system gives access to autonomous racing to every high-school student, irrespective of her or his coding competency.

This document has outlined the various tests for our project of our product. In unit testing, we have explained in detail what method we will use for unit testing and we separated the whole software project into seven units as we use bash scripts. For each unit we give a very clear description for how to test it and the information it will display after the test.

Further discussion in this document involved integration testing and usability testing. For integration testing, it is an activity based on unit testing to test whether each part of the software unit meets or realizes the corresponding technical indicators and requirements in the process of assembling all the software units into modules, subsystems or systems according to the requirements of the general design specification. We focus on the communication between three of our major modules: Configuration, Options, and Communication. And we will use the pytest Python library to perform integration testing.

For the usability testing we are trying to let our clients use the system to test whether they are easy for use. Because in this part we are not only testing its functionality, but we are also gathering information from our users to help make the product better and to make sure that we are meeting all the requirements that our customers need. And make the product more easy to use as this product will be used by multiple high school students. We separated the usability into seven tasks for users to test whether they are able to do or not. At the same time, we will also ask them about their use experience to better improve our products.

Our current implementation progress includes a fully developed configuration module with all necessary functionalities, a functional simulation module, and an established mechanism for transmitting information from the Raspberry Pi to the Jetson Nvidia board on the vehicle. This is to say, we are currently on track in meeting every projected design detail in building RosConnect. Moreover, to establish assurances, we will thoroughly test all components for proper functionality so that each component meets the criteria of its design. The idea that autonomous vehicle technology will be available to high school students and that our team plays a critical role in providing this unique opportunity is both gratifying and fascinating. We are excited and eager to provide this highly interactive application that will give high school students access to a truly intriguing and exhilarating experience with F1/10 autonomous racing!

9. **Glossary**

- Graphical User Interface (GUI):
    - A form of user interface that allows the user to interact with the system using graphics
- Klepto:
    - Module for creating a cached KeyMap in Python
- Lidar:

    - is a remote sensing method that uses light in the form of a pulsed laser to measure ranges
- Nvidia Jetson:
    - Embedded computer that has both a CPU and GPU.
- PyQt5:
    - Is a plug-in for Python that allows users to easily create GUIs.
- Python:
    - High-Level programming language, is used to create the GUI and all of the
- PyYAML:
    - Python packages that allow Python to work with YAML files.
- Raspberry Pi 4:
    - Single board computer.
- Robot Operating System (ROS):
    - Is like an operating system that robots run on. It is not a true operating system though.
- RosConnect:
    - Name of our solution as a GUI.
- Screen:
    - Terminal multiplexer, allows for a connection to be stable when the user is no longer connected to it
- Ubuntu 18:
    - Open-source, long term service operating system from the Linux distribution.
- YAML:
    - Is a human-readable data-serialization language

Appendix A: Development Environment and Toolchain

A. Hardware

Our entire team developed on linux though 75% of the linux environments we used were through virtual machines. Our software needs to run on Ubuntu 16 due to the client wanting to use the Kinetic Kame distribution of the Robotic Operating System. There are two versions of the simulator we used throughout the development process both obtained through our client. The 3D version of the simulator requires more resources to run the 2D version. The client also wanted the software to run on the new 2GB/4GB Raspberry Pi which is why we mainly focused on the 2D version of the simulator in the end.

B. **Toolchain**

- Vim:
    - Highly configurable text editor used for editing files in this project
    - Since this project was originally run strictly via the command line, Vim was a good solution to work efficiently.
- Pip3:
    - Package installer for Python
    - This installer helped ease the process of installing packages such as PyYAML and Klepto
- PyYAML
    - Python packages that allow Python to work with YAML files.
    - Necessary since we use a YAML file to populate the configuration window of RosConnect
- Klepto:
    - Module for creating a cached KeyMap in Python
    - This is used to ensure the saved session functionality of RosConnect
- PyQt5:
    - Is a plug-in for Python that allows users to easily create GUIs.
    - The interface is simple and clear, which makes it convenient for customers to use.

C. **Setup**

After installing your choice of VMWare Player or Virtual Box and getting a copy of the Ubuntu 16 installation media from https://ubuntu.com/, follow the process for installing the new Operating System. You can also install Ubuntu 16 on a harddrive if you want it to be the main operating system on that machine. Then clone the RosConnect repository and follow the instructions on the github page or in the user manual. The setup script will install all dependencies required for our software.

D. **Production Cycle**

If you would like to make a change to the layout of the GUI, then you must use the PyQt5 designer. First you will navigate to the PyQt5 bin by changing the working directory on the command line with the following command: cd /usr/lib/x86_64-linux-gnu/qt5/bin/. Then you can

run the designer by executing the following: ./designer. The PyQt5 designer is a graphical user interface for creating the exact look of the GUI you desire. Once you have inserted your widgets and any other containers or buttons you feel necessary, you will save this file in the RosConnect UI directory. After you have saved your UI file with any changes to the layout of the GUI, you can run the following command to generate a Python file with all the objects generated: pyuic5 -x UI_FILE_NAME.ui -o NAME_OF_EXEC.py. Finally, you will need to update the driver Python file of RosConnect called main.py. In this file, you must now import your newly created UI Python file in place of the preexisting one (originally called mainWindow).

To update or append information to the configuration window of RosConnect you must change the configurable YAML file located at the top level directory with the naming scheme config.YAML. Convention in the YAML file follows that of any other YAML file with the following additions. Every module must be a top level node and must have the following lower level nodes: variable, description, and choices. Choices refers to buttons. These buttons can vary in amount and name but must have the following lower level nodes: title, dependencies, description, and sim. The naming convention is critical to functionality. Thus, please ensure that all lower level nodes are spelled as seen in this document. Once a new configuration has been constructed within the YAML file, you simply save this file and run RosConnect by either executing the main.py file or re-establishing the setup, in order to run the RosConnect desktop application, by following the directions in Part C of this Appendix.