F1/10 YellowTails

Software Design Document

Version 2

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

2/18/2020

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 University. 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. Implementation Overview

The solution our team is building for our clients is a GUI that will serve as the component that allows more accessibility to high school students in operating autonomous vehicles. Additionally, this section will overview technological dependencies in order to deliver the solution to our clients.

The GUI will run on a Raspberry Pi 4 that students in our client's summer camp program will be using as a desktop. RosConnect will have the ability to run simulations based on user constrained configurations. These same configurations can then be sent to the Nvidia Jetson on the RC car via SSH so that the car can begin self-driving.

At our GUI's core is the PyQt5 python package. PyQt5 is a python binding of QT's cross-platform C++ libraries that implements high level Application Performance Interfaces (APIs) for accessing many aspects of modern desktop and mobile systems. PyQt5 has a designer tool in which we have created UI files. We then compile the UI files into Python files and import them into our main Python executable. This allows us to tweak the UI and recompile it without having to change our code.

RosConnect will have the ability to use SSH to send a kill command to the car as an emergency stop feature. As the car may lose the connection, we will have a script that checks the SSH connection. This script will consist of a pair of subscriber-publisher ROS nodes, one on the Raspberry Pi and one on the Nvidia Jetson. Simply put, these nodes will ensure that if connection is lost, then the vehicle stops. A more detailed description of how these nodes work can be found in Section 4.

The main functional aspect of RosConnect is the configuration window. This window provides users with the ability to choose among various options in each of the autonomous categories of perception, planning, mapping, and racing strategy. A key feature of this configuration window is to save and reload previous sessions on closing out of and reopening the application. That is to say, if a user opens RosConnect and makes some set of choices by clicking options inside the configuration window and closes the application, then upon reopening the application, the configuration window settings from the prior session will be saved. In order to save a configuration window's settings, we will save the settings to disk and upon reload will store and access the settings in cache memory. A python module named klepto will be used to implement this configuration saving feature.

Although RosConnect efficiently abstracts away the complexities of the Robotic Operating System, there still exists active processing running in the background for operating a simulation or launching the vehicle. Therefore, we must gracefully handle the exiting of these processes. To do so, we use the os and signal modules in Python to communicate with and kill active processes once RosConnect is closed by the user. Further, all such exiting is performed without requiring input from the user.

The subprocess package allows us to run shell scripts straight from our python code. This allows us to call ROS commands in the background of our application without the user being overwhelmed. This package was also useful in testing the SSH connection to the car.

Lastly we use partial from the functools package. PyQt does not allow functions connected to buttons to take in parameters as it subs in a signal call that simply calls the function if the button is clicked. Partial gives us the ability to pass parameters to these functions.

3. Architectural Overview

Now that the big picture has been set, and you understand how the system is going to work, let's take a deeper look into how the system is actually built. In this section, we will provide diagrams of the software and hardware architecture and give details of each component in these diagrams.

3.1 Architecture Diagrams

Figure 1 is the hardware version of the architectural design and figure 2 is the software architectural design. The software has six modules and those are: configuration file, logging file, option module, simulation module, communication module, and finally, the car module.

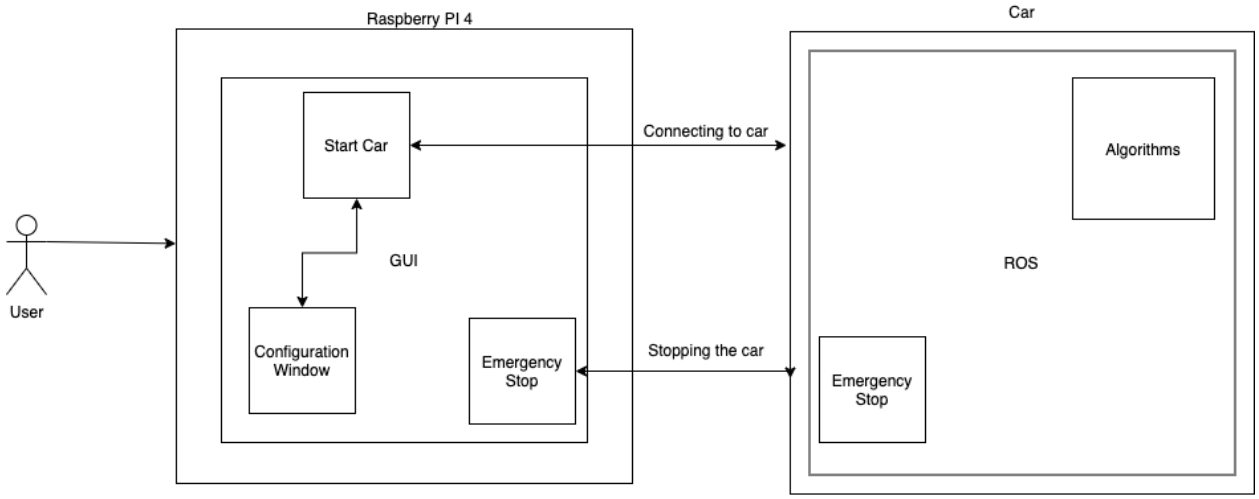


Figure 1: Hardware Architectural Diagram

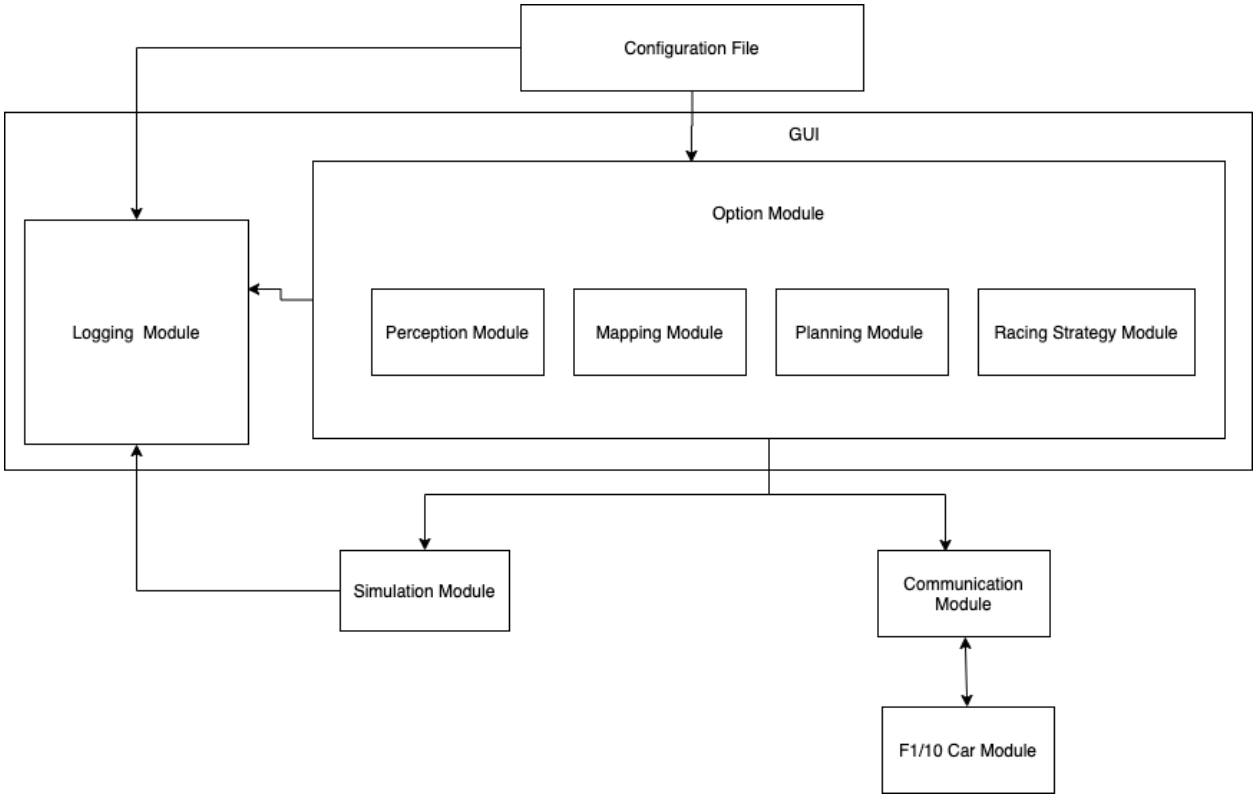


Figure 2: Software Architectural Diagram

3.2 Component Overview

3.2.1 Configuration File

The configuration file is where the owner of the robots can put types of options they want their modules to be able to run. These options will be talked about in the next section. When the user starts selecting the options this will be saved and will run when the run car or run simulation buttons are clicked. If the user clicks the run car button the launch file that is created from the options the user selected will be transferred to the car via SSH.

3.2.2 Option Module

The option module is where the user has the choice of what types of algorithms they would like to see the car's perform. There are four categories that the user has to select in order for the car to work. The categories they have to choose from are perception module, mapping module, planning module, and finally the racing strategy module. The amount of options for each of the categories will be provided by the owner of the car. The user doesn't have to worry about creating these. Once the user selects one option in a category the rest of the options will look to see if they still can apply with what is selected. If it can't work with what is already selected you will not be able to select it. This will make sure that the user is always creating a configuration that will work.

3.2.3 Logging Module

The logging module is provided so that the users can see some output that is happening while they are using the GUI. This logs any time a button is clicked and will show the user that it is in fact running. The logging module will also receive information from the other modules if they have to pass any information so the user can see it.

3.2.4 Simulation Module

The simulation module is where the user is able to test all of their configuration options in a simulation to see how the car is going to work with the configuration they have set up. This could help prevent the user from going straight to the car and damaging or even destroying the car. When you click the start simulation there will be a message that gets sent to the logger saying the simulation has started along with any ROS information that the user needs to see.

3.2.5 Communication Module

The communication module broken into two parts. The first part is the emergency stop part and the second part is a script on the car that is listening to make sure that we are still connected via ssh. If we were to lose connection we would be able to stop the car so that the car doesn't crash. The emergency switch that we can click will send a script over to the car telling it to shut off.

3.2.6 F1/10 Car Module

The F1/10 car module is the actual module where all of the code is held. The users send their configuration file to the car and which will run it with the behavior that they are wanting. The communication module for this is keeping a connection with the GUI by sending messages to each other. This is allowing the car to make sure that it still can receive input from the user if

needed. There will also be a communication onboard the car that is checking for a script or command to be called telling it to stop.

4. Module and Interface Descriptions

In this section, we take a closer look at the finer details of how our system works. The proceeding subsections will provide a discussion of the functionality of each part of the system, along with a class UML diagram to show what methods are going to be used to ensure proper functionality.

4.1 Configuration Module

The first module of our project is the configuration module. This mainly consists of a .yaml file that contains the available options for our 4 option modules. The configuration file allows our clients to determine the options available to the students and change those options in the future to fit their summer camp program. This is the first module in our application and handles a majority of the startup process.

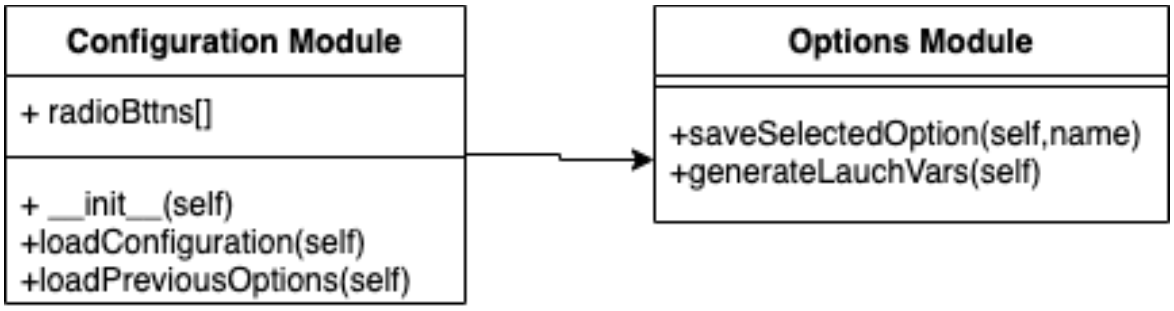


Figure 3: UML Diagram of the Configuration Module

__init__(self) - This function is automatically run at startup and loads our UI files. This is where we connect our functions to the buttons and menu options of the UI. This function calls the loadConfiguration() and the loadPreviousOptions() methods.

loadConfiguration(self) - Here we load the .yaml file and populate the options in each of the option modules.

loadPreviousOptions(self) - Looks at an archive file and loops through the radio buttons toggling the ones that were marked on last exit.

4.2 Options Module

The options module consists of 4 sub modules; perception, mapping, planning and racing strategy. At the start of our application, each of the sub modules is populated by the configuration module with a radio button for every option. Depending on what the students select in each sub module the options module generates variables that will be passed to ROS that then launches the simulator or runs the car. The options module also saves the currently selected options on close and reselects those options on application start up.

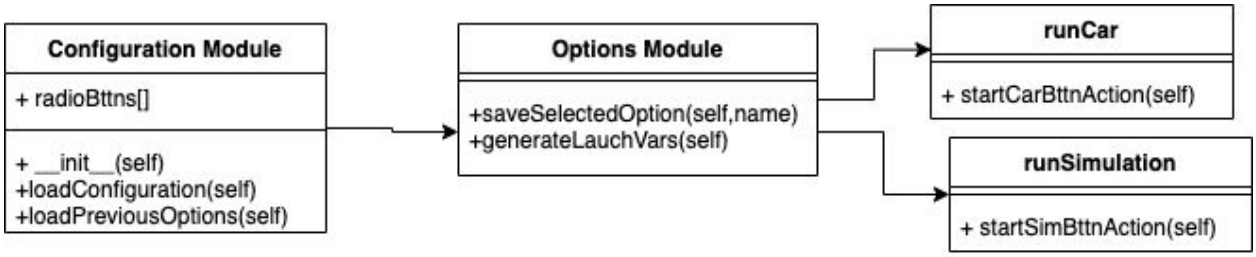


Figure 4: UML Diagram of the Options Module

saveSelectedOption(self, name) - This is called whenever a radio button is clicked and changes the values in the archive file to correspond. The archive file is accessed like a dictionary and when an option is clicked its value is assigned a ‘y’ and whatever it replaces is set to ‘n’.

generateLaunchVars(self) - This function gathers the selected options and generates the string that will be passed to the simulator or the car.

4.3 Simulation Module

The simulation module takes the output from the options module and passes runs a Gazebo simulation with those options affecting the behavior of the car.

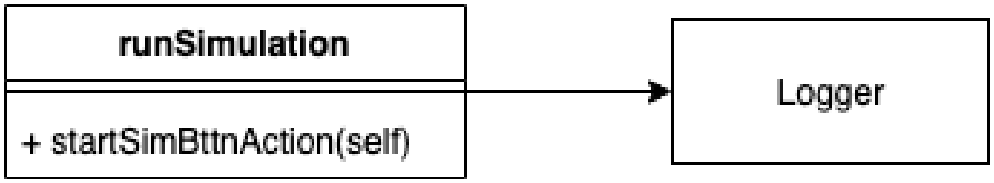


Figure 5: UML Diagram of the Simulation Module

startSimBttnAction(self) - Lanches the simulator via a hidden command line with the current launch variables

4.4 Car Module

The car module takes the output from the options module and passes the launch file to the car in order for the car to run.

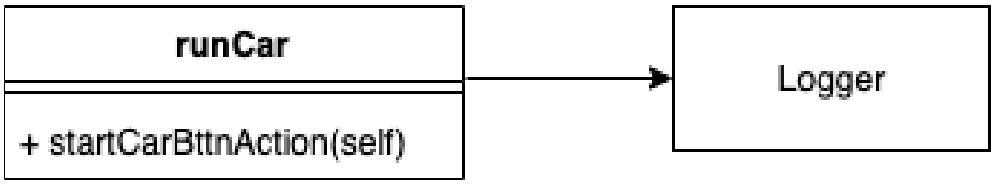


Figure 6: UML Diagram of the Car Module

startCarBttnAction(self) - Lanches the car via a hidden command line with the current launch variables

4.5 Communication Module

The communication module will have the ability to use SSH to send a kill command to the car as an emergency stop feature. As the car may lose the connection, we will have a script that checks the SSH connection. This script will consist of a pair of subscriber-publisher ROS nodes, one on the Raspberry Pi and one on the Nvidia Jetson. The node on the Jetson will subscribe to an active topic created from the node on the Pi. This active topic on the Pi will publish a message every epsilon interval, where an epsilon interval refers to some time interval less than 1 second. If the node on the Jetson fails to read this message at any given iteration, then the script will execute a shutdown sequence of commands. That is, If the connection is lost, then the script sends the kill command to the system that stops the car.

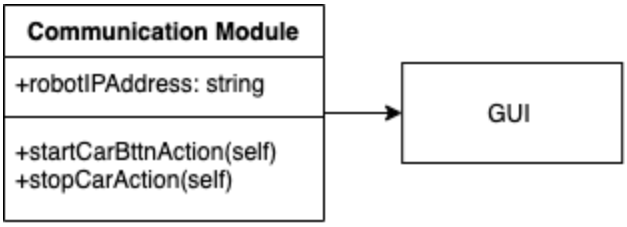


Figure 7: UML Diagram of the Communication Module

startCarBttnAction(self) - Uses the current launch variables and set IP address for the robot to start a ssh connection to the car. With the passed variables the car will start its own ROS launch file and our check-connection node.

stopCarAction(self) - Use the open ssh connection to send the kill command to all ROS nodes stopping the car. If we lose our connection the check-connection node on the care will automatically kill all nodes on the spot.

4.6 F1/10 Module

The f1/10 car module will mostly be set up by our clients who will have all the packages that correspond with their given configuration file. This enables us to send just the string of variables and get the corresponding behavior we want from the car for example the emergency stop car button that corresponds to the stopCarAction in the previous module.

4.7 Logging Module

All of the modules also have the ability to interact with our logging module and push messages to the console on our GUI. The logging module handles the output of our GUI in a way that the students will be able to understand.

5. Implementation Plan

We have talked about all the details of all the modules in the previous section, and we now require a plan in order complete implementation of our project. The development of our project is broken into four sections, connecting to the car, the GUI, emergency stop, and the launch file manipulation. This is represented in Appendix A, in the form of a gantt chart. Under each of these four sections there are three sections involved, planning, implementation and finally testing. Each of these are denoted by a separate color. Our planning parts of the project are the blue color, while orange is the implementation and finally red is the testing part of those sections. The green line is where we currently are at in the progress of getting the implementation done.

In order to have a successful implementation of each section we are planning for a couple of weeks before we start the actual code. We are taking a couple of weeks to plan out the information to make it easier for us when we actually code it. The team is handling all of the planning at the same time. By doing all the planning at once the team will be able to have a solid idea on what is going to be needed to be done in all of the modules.

As can be seen in the gantt chart we have a lot of time for implementation of the GUI. The team has planned to take a couple of weeks in order to make sure that all of the modules get programmed the way we want them to work. Taking this much time is critical for the success of the GUI since we need it to be completely working like we want it to and how the clients want it implemented.

For the testing part of each of the sections we are going to be writing tests to make sure it is going to work how we would like it to. One thing that we are going to also work on in the testing phase of the project is make sure that the system that we have created is as break proof as possible. This is important since our product will be used for a high school summer camp. We have to make sure that we are not letting any of the students break the project.

We are working on all of our programming tasks to have everything programmed for our Alpha demo that will occur on week 9. This is our next major programming task. After all the major sections of our code are implemented we will work on testing and look into adding some stretch goals.
We will start having different types of users to test our GUI after all implementations of the code is done. This will be after our alpha prototype scheduled for week 9. That way we can have the full design done for them to look at and give us feedback on what they find confusing and needs changing.

Now that we have a well developed implementation plan, the team will go on to determine what tasks each person is working on for the week and follow up on what tasks have been completed to stay on track. With the implementation plan this will help us remind ourselves what tasks we have left to work on to get all of the programming and testing done for this project. As a team we feel really confident in what we have come up with in our plan to have a successful rest of the semester and completing this project.

6. 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.

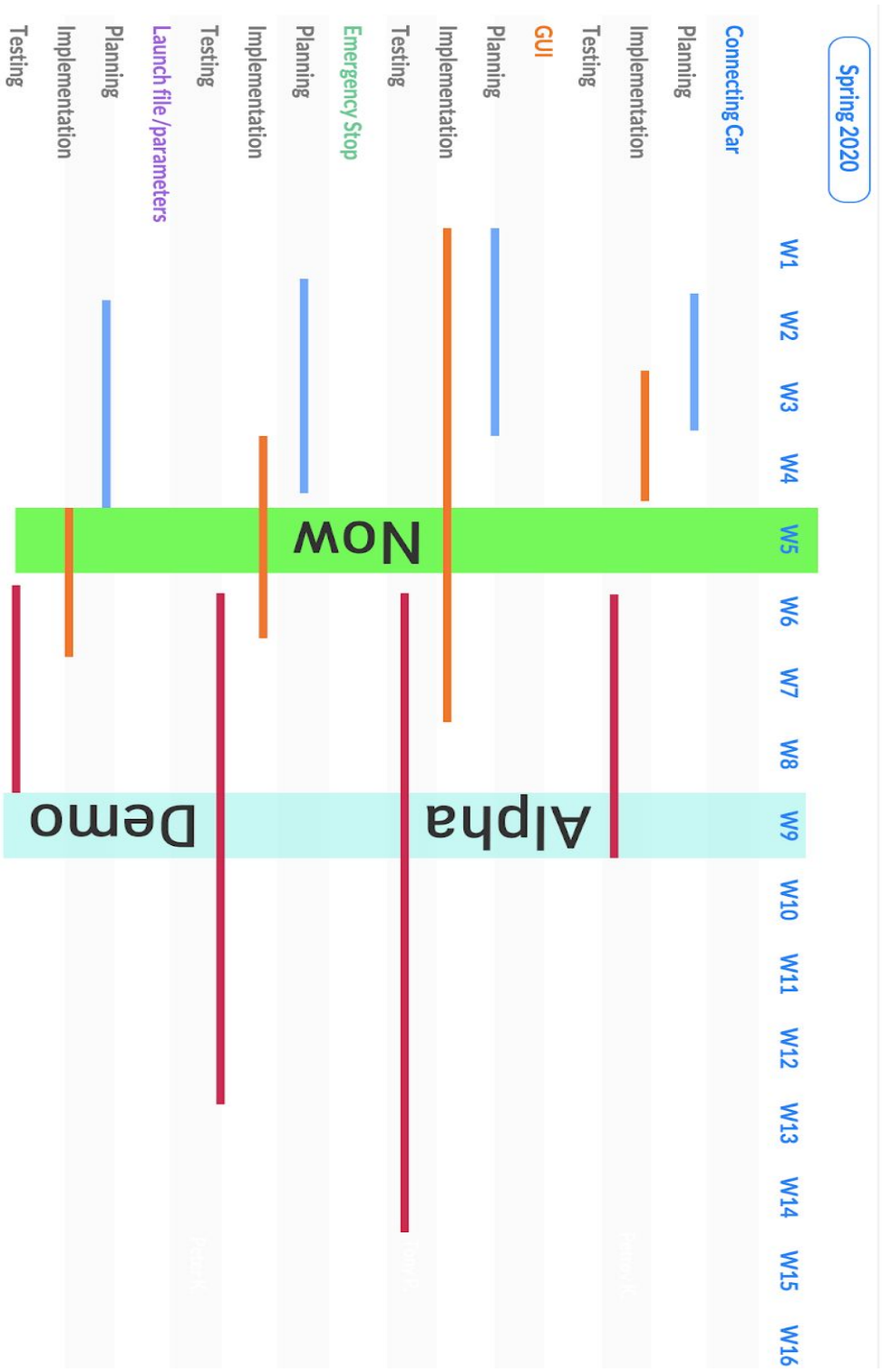
The problem is that the F1/10 autonomous platform, in its current state, is complicated to operate. This means that many students who lack the necessary coding experience are unable to participate in our clients summer camp program. So, Dr. Nghiem, Doan Nguyen, and the F1/10 Yellowtails aim to make this 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 design of our product, RosConnect, by detailing our envisioned solution and every aspect necessary to implement this solution. In Figure 1, we detailed the hardware architecture which included high level components such as the Raspberry Pi and the vehicle. Additionally, in Figure 1, we presented lower level hardware components involved in the architecture such as the start car button, configuration window, and emergency stop mechanism.

Further discussion in this document involved a presentation of the software architecture from Figure 2. This figure outlined major components such as the configuration file, logging module, option module, and communication module. In section 4 we presented the UML diagram of the module and interface descriptions to introduce each of the modules we used for our project. We concluded with an implementation plan that outlines four sections of development: connecting to the car, building the GUI, emergency stop, and launch file manipulation.

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!

Appendix A


Spring 2020
W1
W2
W3
W4
W5
W6
W7
W8
W9
W10
W11
W12
W13
W14
W15
W16
Connecting Car
Planning
Implementation
Testing
GUI
Planning
Implementation
Testing
Emergency Stop
Planning
Implementation
Testing
Launch file /parameters
Planning
Implementation
Testing
Now
Alpha
Demo