F1/10 YellowTails

Requirements 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

12/13/19

Table of Contents

1. Introduction

F1/10 Yellowtails is a team that was formed with the goal to improve 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. Our clients are now 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 get high school students interested in STEM and recruit them 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 runs the Ubuntu Operating System which is then used to run the Robotic Operating System (ROS), in order to control the car. Everything must be run from the command-line and ROS adds additional hurdles like the need to source the workspace directory and use the Catkin compiler to compile all C code and packages which needs two other files to be correctly set up syntactically, with dependencies and the correct parameters for the package. Changing one setting might also require changing multiple files in different directories.

Dr. Truong Nghiem, Doan Nguyen, and F1/10 Yellowtails aim to make ROS 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.

2. Problem Statement

The barrier of entry to the field of autonomous driving is steep. The hardware needed to get started is expensive and most of the time requires the user to wire and solder all the components together. The F1/10 platform was made to be a cheap, scaled-down version of a Formula 1 car but still cost upwards of $3,000. For entry level autonomy there are many open source projects that work but require an extensive background in coding and Linux. Almost all of said solutions use ROS to control the robot and handle all sensors.

Main Problems:

- ROS's complexity
- Disconnected configurations
- No emergency stop

ROS is one of the main hurdles when learning robotics. It is the most used robotic framework in industries across the world. ROS was created to be a global collaboration to make the field of robotics more accessible. Its Modular design allows users to choose packages that will be useful to them and implement their own solutions. The main downside of ROS is its entry level requirements. To use ROS, everything is done through a command line interface. On top of knowing all of ROS's commands users need to be comfortable with Linux commands. ROS's compiler requires a certain file structure to work and there are multiple files per package that have to be in the correct subfolder with unique syntax. In Figure 1 below, the current ROS workflow is outlined.

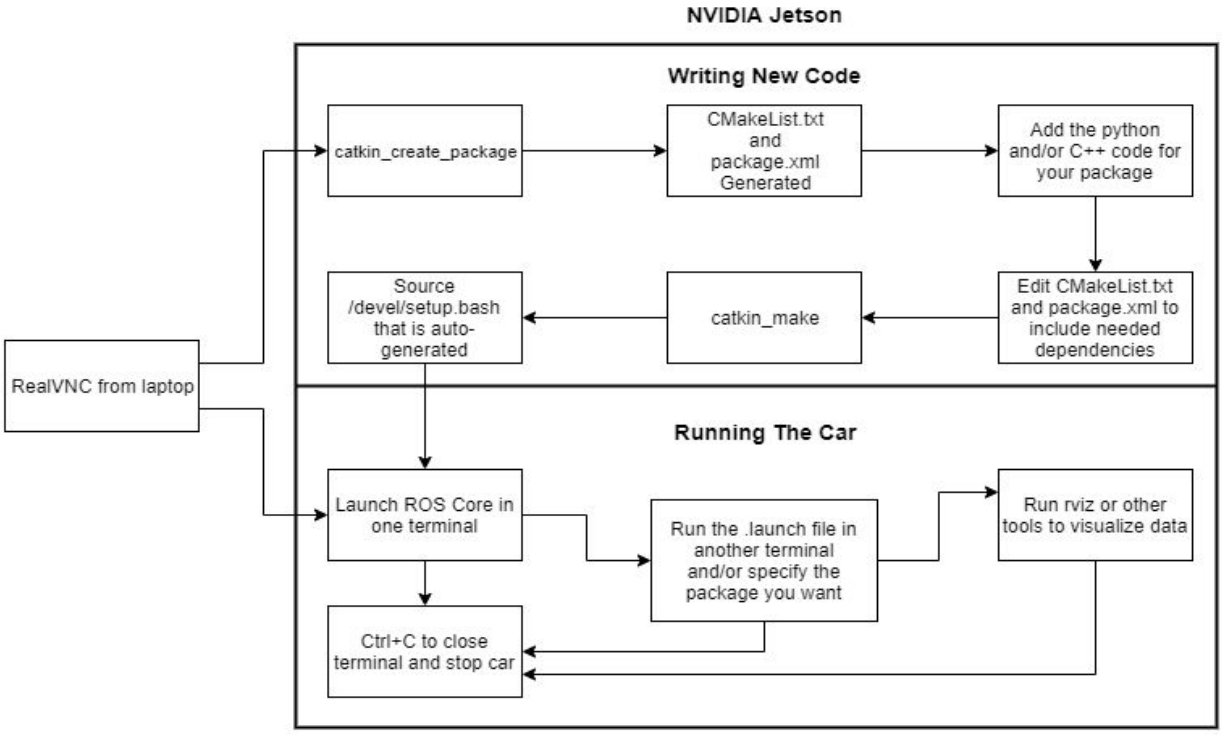

Figure 1: The current ROS workflow that our clients are using.

Additionally, to change the behavior of the car, or its configuration, multiple files in different locations need to be manually edited. For example, to test camera based algorithms, all files need to be edited to comment out or remove the lidar libraries and the lidar sensor. It is also possible to make configurations that will never work (i.e., trying to use a lidar based package with the camera). Our client has stressed that our product needs to allow students to choose configurations that are guaranteed to work.

Finally, the current workflow has only one way to stop the vehicle and is reliant on a constant connection to the car. If this connection is lost, then the client must manually stop the car which could be potentially dangerous as the car can reach speeds of 40-50 mph. This can be costly as certain parts of the car are susceptible to damage upon crashing into something at high speeds. We must solve this problem so that working with the F1/10 platform is safer and more reliable in terms of costs due to damage.

3. Solution Vision

We envision a solution that meets our clients needs by transitioning the current workflow to a highly interactable graphical user interface (GUI) that lowers the barrier of entry to use ROS. This GUI will be run on a Raspberry Pi that each group of students will have access to during the summer camp program. The GUI will remove any need for the students to use the command line and will only allow students to make valid configurations for the RC car. By abstracting away complexities involved in utilizing ROS to make a vehicle run, students will be able to instead focus on ascertaining an effective and enjoyable learning experience. Additionally, we will have a communication toolkit that enables a kill switch mechanism for the car. This will ensure the safety of every student involved and the safety of our client's very expensive product; the vehicle itself.

Key highlights of our software include:

- Communication Tool Kit
- Configuration System
- Implicit File Organization
- Profiler
- Console

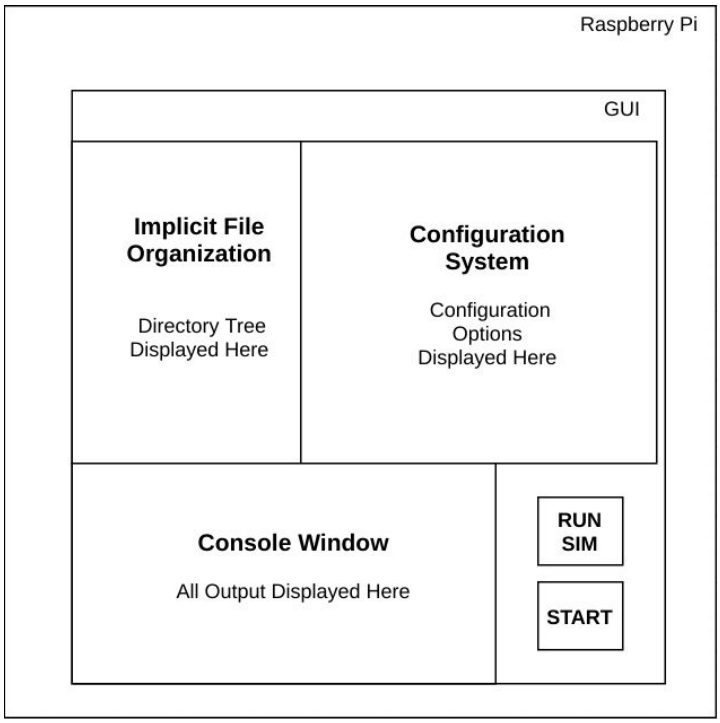

Figure 2: Envisioned layout including three of our five key features

As can be seen from Figure 2, our GUI will display all configuration options available to the students. To ensure that students cannot choose an incorrect setup, upon making a selection, all incompatible options will be automatically hidden. We will work closely with Doan in the ICONS lab to decide what options the students will be able to choose. Once the options are selected the students can choose to run a simulation of the car with their given configuration or to send the configuration to a car and run the car. The backend of our GUI will create a ROS launch file with the corresponding configuration.

The Profiler allows us to save the IP address of multiple RC cars and switch between them. This feature is not likely to be used by the students but will help the clients when setting up. The ROS launch file, which is a plain text file, is the only necessary information to be sent to the car. However, after the launch file is sent and executed we will have a button on our GUI that stops the car. If we lose connection, then an onboard listener script will stop the car. This makes losing control of the car less likely to damage itself or injure students..

Since users are no longer required to interact with the command line interface, key steps associated with launching and running ROS will happen in the background. Our GUI will contain a console window the console is where all necessary output reporting will occur. Messages to explain that the simulation is loading or that the configuration is being sent to the car will appear in this console. We will ensure that messages displayed to the console are simple so as to limit the confusion for users.

4. Project Requirements

After many weeks working with our clients, we have the necessary requirements for this project. The team is very confident that we are able to complete all of the requirements. To be able to complete the solution in the above section we need to create a detailed list of requirements. Clearing documenting our requirements enables us to begin constructing a prototype that demonstrates our requirements with the goal of creating our envisioned solution.

4.1 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.

4.1.1 Graphical User Interface

The graphical user interface defines our front-end component of the solution. This is the primary and only tool users will be provided with in order to operate and interact with the vehicles.

4.1.1.1 Communication Toolkit

The communication toolkit includes a script on board the vehicle and an interactable stop button on the GUI. The on board script will have the function of consistently checking for connection. If the connection is lost, then the script will shut down the vehicle. The stop button on the GUI will provide users with the ability to stop the vehicle with a single click. This button will have a red background with the words “Stop Car” in order to increase the likelihood of recognition by the user.

4.1.1.2 Console

Since the team is creating a GUI, there is no longer going to be any use of the command line by the users. The downside to this is the users do not get to see exactly what is contained in the output of the program. This is where the console will come into play. The information of the output will be displayed on the console. This will occur because the user needs to know that the program is still running, as opposed to being stuck on something. Additionally, the console handle all foreseen errors by displaying those errors to the user.

4.1.1.3 Profiler

The profiler will be a pop up window that has the user select which project they would like to run with our GUI. Once the user inputs the IP address of the project that they are trying to run, they will have to click a button to save it into our system. This ensures that when the project is ran the correct configuration is selected.

4.1.1.4 Run Simulation

The functionality of the run simulation is going to be a button that, when clicked, takes the user to the simulation of the car running. This is sourced from the University of Pennsylvania F1/10 simulation program, allowing users to test their program configuration before running it with the vehicle.

4.1.1.5 Run 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.

4.1.1.6 File Input[1]

The file input part of the GUI is to let the user choose which ROS nodes they would like on the car. This way we can have our project work with any type of car and not just the F1/10 Flagstaff program.

4.1.1.7 Edit Files[1]

Our GUI will include a feature that allows users to edit files for preference changes. There will be a tab for the user to switch between the configuration window and the edit window. Here, they will be able to see the file being changed so they can modify it to their preference.

[1] *This requirement will be a stretch goal that will provide extra functionality for the user.*

4.1.2 Configuration Package

This is the back-end of the solution created for our clients. This section will ensure that the user never gets an error while configuring the changes.

4.1.2.1 Configuration window

The configuration window is defined by four different options which determine various configurations for the user to choose from. As can be seen from Figure 3, these options will be radio buttons with labels: racing strategy, perception type, mapping, and planning.

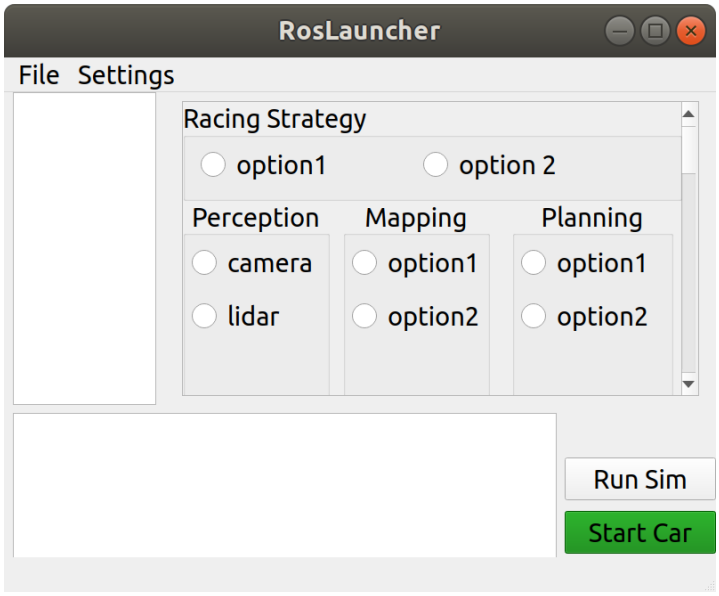

Figure 3: A depiction of our initial configuration window.

4.2 Performance Requirements

In this section, we will discuss how each of the above functional requirements will work. We will effectively describe how our system must behave and establish constraints of its functionality. For two of the functionalities this will include a speed metric for tracking execution time.

4.2.1 Communication Toolkit

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 that 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.

4.2.2 Graphical User Interface

This is the front end part of the solution the team looks to provide to the clients. This is what the user is going to use to interact with the project.

4.2.2.1 Console

Since the user is unable to see the command line output the team is going to work on creating code in the background that will pull information from the terminal that we are running ROS in and display information to the users. This is will be done and users should be able to see what is going on within 30 seconds of the command being run in the terminal to help guide the user to knowing what is going on. The messages should be simplistic enough that it is able to be read by anyone that is running our program. The time it takes for something to run should be displayed here so the user knows how long it takes.

4.2.2.2 Profiler

Once the user puts in all the required information about the profile they would like to use, the back end of the solution will input the given information into the necessary parts of the code to be able to run smooth lessly. This will be done as soon as the user is done selecting the correct project that they are wanting to run.

4.2.2.3 Run Simulation

When the user clicks on the run simulation button, this will call the script that the team wrote to start the simulation process. This script goes through and starts all of the required parts of the simulation and when it is completely running it will pop up the simulation in a new window. Depending on how much RAM and the amount of video memory will control how long it takes to open up Gazebo for the simulation.

4.2.2.4 Run Car

The goal for this button is that when the user is confident that it's going to work and click the button it will start the process of connecting to the car and running the program. The user should be testing the program first with the simulation to prevent errors or even worse damaging the cars. The time span it took from the click of the button to the time of the actual car running will be recorded and displayed on the console. As the script is going through its code they should be printing important information to the console part of the GUI so the user can understand what is going on and the time it took to get there.

4.2.2.5 File Input [2]

This functionality will be done by drag and drop, so it should only take a few seconds for the files to show up in the viewer. Once the file is inside the viewer the user is able to edit the file and that information will be discussed in the section below.

4.2.2.6 Edit Files [2]

When the user wants to edit a specific file it should appear in this section of our solution. It should take no more than 5 seconds to show up. That way the user is able to start making the edits that need to be made.

[2] *This notation is going to be stretch goals that would provide extra functionality for the user to customize easier.*

4.2.2.7 Usability

While creating this solution we are going to make sure that high schoolers are able to use our GUI. This is going to be needed because if the team needs to make it more user friendly we will try and will provide instructions as needed. This will help the team make sure we are providing the best easy to use GUI for high schoolers.

4.2.3 Configuration Package

Our solution will abstract away all configuration file handling from the users. The configuration package has a sole purpose of writing the user-chosen configurations to a file for exporting to the vehicle. Additionally, the configuration package will ensure that users are unable to choose invalid configurations.

4.2.3.1 Configuration window

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.

4.3 Environmental Requirements

The environmental requirements are components within the summer camp program that our product must interact with and use. These environmental components are either set forth by our

client, Dr. Nghiem, or have been deemed as the only viable options. There will be two types of environmental requirements reviewed below, namely software and hardware.

4.3.1 Software

This section will discuss all of the software environmental requirements that the team faces. There are four main topics that will be discussed in further detail in their own sections: ROS, Linux Ubuntu, Python/C++, and Gazebo.

4.3.1.1 ROS

ROS is one of the most important, if not the most important, environmental requirement because this is what is running on the car. Also, all the code that is written for the car is done inside ROS.

4.3.1.2 Linux Operating System

Our solution has to run in Linux, specifically Ubuntu. The ubuntu system that we are running is 18.04 and do not have to worry about cross platforms for this project.

4.3.1.3 Python/C++

All of the ROS nodes are written in Python and C++, so the team must use these languages for anything that is going to be written for this project. Anything that we want to create for this project has to be done in these languages.

4.3.1.4 Gazebo

In the GUI when the user wants to run the simulation. When the user clicks the run simulation button it will then go through the code and open up Gazebo which allows you to see what the car is going to do if you were to run the actual car.

4.3.2 Hardware

This section will discuss all of the hardware environmental requirements that the team faces. There are two main topics that will be discussed in further detail in their own sections. These four are the NVIDIA Jetson, Raspberry Pi, Lidar, and the vehicle.

4.3.2.1 NVIDIA Jetson

The NVIDIA Jetson board is what the car uses to run. The Jetson board holds a CPU and a GPU. The NVIDIA Jetson also has all of the connections that the car needs to be able to function.

4.3.2.2 Raspberry Pi

The Raspberry Pi is the host computer containing Linux Ubuntu that high school students will be using to interact with the GUI.

4.3.2.3 Lidar

When the user goes to run and view the lidar they are using the GPU to be able to see what the lidar is seeing. This is what is built in on the car currently for path detection. This is a main key since this is the device that will detect if there is any obstacles in the way of the car or not.

4.3.2.4 Car

Below is a list of predetermined components and features that define the vehicles to be used by our client.

- Lidar
- Wheels
- NVIDIA Jetson
- Lidar
- WIFI Connectivity
- Bluetooth

5. Potential Risks

Since the real world and the ideal experiment are different, we must consider not only the software risks, but also the risks we may encounter in actual operation, especially those of which have serious consequences. We outline below the severity and likelihood for each risk we may face. Both likelihood and severity can be considered as low/medium/high, and these risks can be considered as four parts:

Losing connection:

- How it happened: As we are using a wifi-SSH connection method for the car and software, SSH requires that both machines be connected to a network. In our case the car and Raspberry Pi will be on the same wifi network. Thus, all that is necessary is corresponding IP addresses. If there is a network interrupt, the wifi signal will be lost and the wifi-SSH connection will loss, car will lose control; similarly for the hardware. If the part of the hardware that is connected to the network is not well connected, then the car will lose control. For example, the connector of the WiFi-SSH is in poor contact with the vehicle, or if the part is detached or damaged in actual operation.

- Consequence: If the connection is lost, then the car can not be controlled by a user any longer. During racing, the car will run at a high speed, so it is easy to hit the wall or any other damage-inducing object. The worst case is the car hits the user at high speeds and the result is a damaged car and injured customer.
- Severity: High
- Likelihood: High
- Mitigation Strategy: Our mitigation strategy for losing connection will be the implementation of the communication toolkit as mentioned in Section 4. The communication toolkit includes an onboard script that consistently checks for connection between the NVIDIA Jetson and the Raspberry Pi. If connection is lost, then the onboard script will shut down the vehicle. Additionally, users will have the option of clicking a stop button provided by RosConnect's interface. The stop button gives users the ability to shutdown the vehicle.

Internet error:

- How it happened: The actual operation at the location of the summer camp program may cause a network outage. Moreover, our connection method, wifi-SSH, requires both car and Raspberry Pi to be connected to the same network. So, if there is an internet error and the two machines are connected to different networks or not connected to any network, the car will not work.
- Consequence: If internet connection is lost, then we our connection to vehicle is lost. This means that we lose control of the vehicle and risk damaging an expensive piece of equipment. Additionally, loss of internet connection resulting in uncontrolled vehicles means that legitimate racing for students is jeopardized. Furthermore, an internet error could imply a risk of danger to users in close proximity to uncontrolled vehicles.
- Severity: High
- Likelihood: High
- Mitigation Strategy: Internet conditions cannot be guaranteed. We cannot ensure network conditions, so in fact we cannot solve the problem of network errors. However, before the summer camp begins and racing commences, we will check whether the vehicle and Raspberry Pi are able to connect to the appropriate network. Additionally, we will run the simulator to ensure that any programs chosen by the users can work properly.

Operational error:

- How did it happen: Actual operation will produce error, and this error is inevitable.
- Consequence: As actual operation causes error and this error for the robot racing may cause the car to deviate from the original track, and this will make the racing game hard to complete, and as the car will be running at a high speed, it is easy to cause damage to

the car. If the operational error is a large number, cars are likely to collide with each other during the game.
- Severity: Medium
- Likelihood: High
- Mitigation Strategy: Operational error is difficult to eliminate entirely. However, to help prevent these errors, we will run all possible configurations provided by the summer camp program in the simulation. This will guide us in our construction of RosConnect to minimize the error before the actual racing starts.

Compatibility:

- How did it happen: The high level parts of our system including the GUI, Rasperry Pi, Ubuntu, and ROS, must all be compatible. All of these components are technologies that are constantly updating and changing. Therefore, it is feasible that two of these components become incompatible with one another.
- Consequence: If any two components are not compatible, then the project would have to be revamped.
- Severity: High
- Likelihood: Low
- Mitigation Strategy: Before completion of our RosConnect, we will fix a compatible version for each component and will thoroughly test compatibility if versions change. However, we can not guarantee compatibility if the version of any given component is updated or changed.

6. Project Plan

Our current project execution plan involves a multitude of milestones. As can be seen from Figure 5 in the Section 8 Appendix, we will be gaining further experience with our hardware constraints, documenting all requirements necessary for building our product, and constructing an initial technical prototype. A timeline of our these milestones is again documented in Figure 5. We will briefly outline and describe our milestones for the Fall and Spring semesters below:

Our plan for Fall semester (currently):

- Increase our familiarity with ROS and other environmental constraints
    - Spend two hours each week with our client to learn more about ROS and gain experience working with hardware constraints such as the NVIDIA Jetson and Raspberry Pi

- Complete our Requirements document
    - Analyze the requirements and feasibility and document them as necessary
    - Send final version of Requirements document to our client Dr. Nghiem for review and confirmation
- Build our Technical Prototype for demonstration
    - Construct a basic graphical user interface that displays and functions as specified in our predetermined grading rubric
    - Basic GUI features include, but are not limited to, a working ssh connection to the vehicle, a directory tree-view display, and a profile selection tool as a pop-up window

Our plan for Spring semester (1/19 - 5/20):
As can be seen from Figure 5 in the Section 8 Appendix, we will continue our research on ROS to gain a better understanding of its complex inner workings. We will enhance the features of our current technical prototype and construct any additional and necessary features which are outlined in this document. Finally, we will test all of said features thoroughly to ensure a robust design. Additional project milestones, as depicted in Figure 5, include documenting our software design, building our second technical prototype, establishing two additional design reviews, and lastly finishing and submitting our product to the clients.

7. 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 effectively outlined our product, RosConnect, by detailing our envisioned solution and every requirement necessary in constructing this solution. We discussed all environmental aspects that are required to build our product. We detailed hardware constraints such as an NVIDIA Jetson and a Raspberry PI that will be used as the brains of the vehicle and the operational tool hosting RosConnect, respectively. We detailed software constraints such as the Linux OS that will be used as the host operating system and ROS that will be used as the vehicle-controlling operating system.

Further discussion in this document involved our functional and performance requirements. For our functional requirements, we outlined a graphical user interface that will contain features such as a communication toolkit for checking connection, a profile/project selection tool, and a console to communicate active events to the user. Additionally, we detailed a simulation tool for enhanced user experience and a start car button to run the vehicle with user specified functionalities. For the performance requirements, we analyzed each of our functional requirements. That is, we provided working descriptions and in some cases, such as the communication toolkit and the console, we provided timing metrics for our functionalities to abide by.

We are currently on track in meeting every projected risk associated with building RosConnect. To establish assurances, we will apply the corresponding mitigation strategy for each risk as outlined in Section 5 of this document. 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 interactable application that will give high school students access to a truly intriguing and exhilarating experience with F1/10 autonomous racing!

8. Appendix

Figure 4. Fall Semester Schedule to support Project Plan description in Section 6.

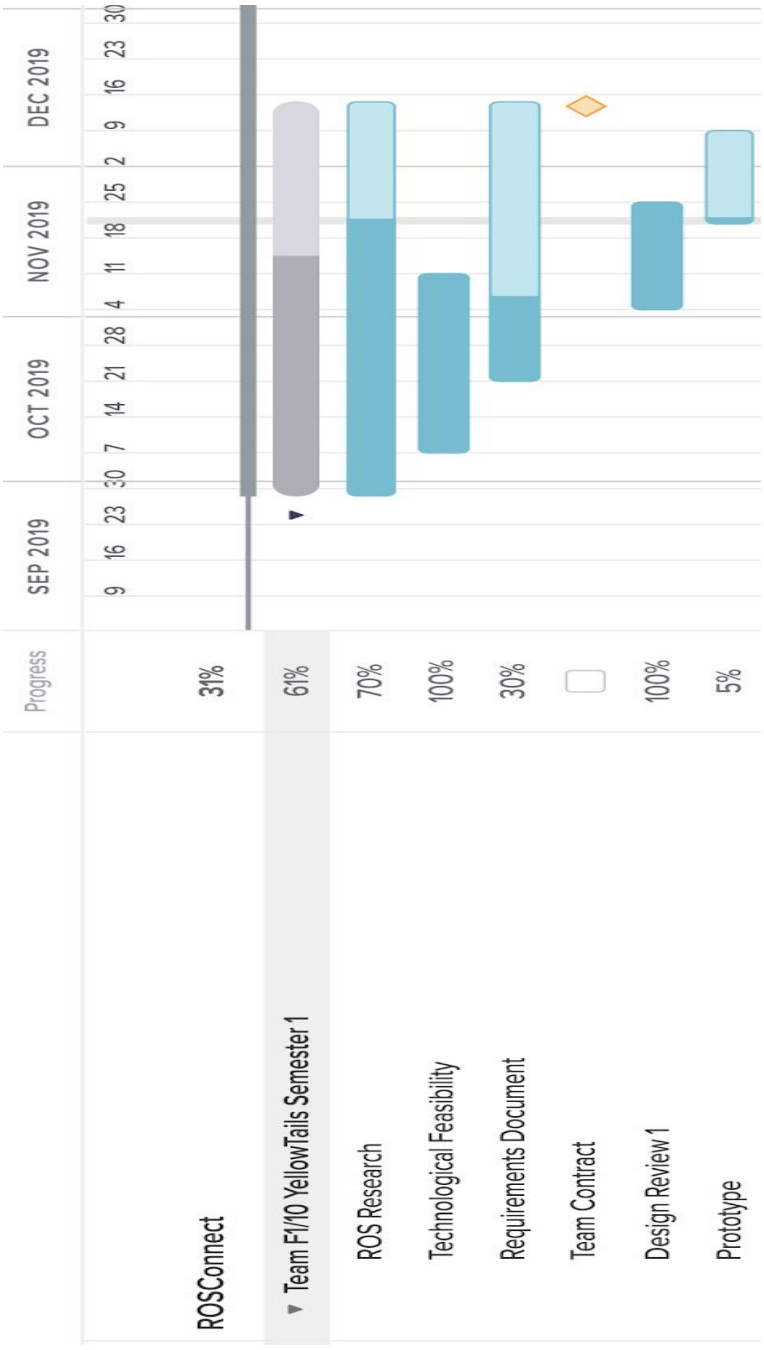

Figure 5. Spring Semester Schedule to support Project Plan description in Section 6.