F1/10 YellowTails

Software Testing Plan

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

04/03/2020

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. Unit Testing

As our software is meant to simplify complicated systems it is important that our software works as intended and does not add to the stress of working with the RC cars and autonomous robotic concepts. To accomplish this goal we split our software into individual units that we can test to determine if they do what they are supposed to do. No matter what input these units are given they should handle it gracefully as to not crash the application or hang. For example, if given a bad input the code should display an error message instead of crashing.

We will be using the PyTest library to streamline our testing and get more details. Due to the fact that our software uses bash scripts to accomplish some task we have developed some test scripts as we progressed to test these features.
We will be testing the following Units:

- Loading the Configuration
- Dependencies in the Configuration
- Saving and Loading Profiles
- GUI Output
- Simulation Startup
- Car Initialization
- Kill Switches

2.1 Loading the Configuration:

We worked with our clients closely to come up with a configuration system that populates the GUI. This was important so that in the future our clients could change their curriculum without having to recode anything. If the configuration is not a Yaml file or if the file is not formatted properly our application should show a message stating such.

2.2 Dependencies in the Configuration:

The options that are selected are bound by dependencies that are established in the config file. This means that our software does not allow now-working configurations as long as the config file is correctly setup. When selecting options in the GUI options that can no longer be selected due to conflicts of the depency tree are not selectable.

2.3 Saving and Loading Profiles:

The collection of options that are selected are called profiles. Our client wanted a way for the students to quickly have one setup and switch to another one. It's important that these profiles are connected to a corresponding configuration file as the options in the profile need to exist in the loaded config. This is handled by a version number that both files contain and our software checks for equivalency. If the number of the profile you are trying to load does not match the version number of the loaded config the GUI prints a message to the console that the profile can not be loaded due to non-matching version numbers. Saving a file is easy and its content is auto generated by our software.

2.4 GUI Output:

When the car or simulator is started the GUI takes the current selected option and converts into a specific string format that the Simulators and GUI need to run the corresponding configuration. We test this with a script that gets the output string and allows us to test it against what it should be.

2.5 Simulation Startup:

The output from the GUI determines what simulator is run. There are checks in place so a user can not open more than one simulator as each instance is resource intensive. On close our software makes sure to properly close all the background processes that the simulator opens.

2.6 Car Initialization:

When the car starts, it takes the input from the GUI and opens an SSH session that passes the GUI string to a launch file. This launch file starts the Robotic Operating System including a process that will stop the car if the SSH session is lost. The launch file on the car corresponds to the config that is loaded into our software. The client opted for this method so that they can change and adjust all packages they might use in their curriculum.

2.7 Kill Switches:

There are two kill switches to protect the user and expensive equipment that is on the car. The first kill switch is located on the GUI and sends a kill command over the SSH connection. The second kill switch is an emergency backup on the car that stops it if the SSH connection is lost.

3. Integration Testing

Integration testing is a mechanism that it used for checking that modules of a system interact as expected. The goal of integration testing is to expose the existence of communication errors between a system's modules. For the purposes of integration testing on our product, RosConnect, we focus on the communication between three of our major modules: Configuration, Options, and Communication. We will be structuring our integration tests with a *top-down* approach. That is, we will validate the functionality of module integration starting from the user-facing elements. To perform integration testing we will use the *pytest* Python library as was determined in our Technology Feasibility document.

3.1 Configuration File to Configuration Window

The configuration file is a YAML file that is constructed by our clients. The agreed upon format of this file has the following structure,

Category 1
- choice 1
    - Title
    - Dependencies
- choice 2
    - Title
    - Dependencies
    …
- choice n
    - Title
    - Dependencies
    …

Category N
- choice 1

    - Title
    - Dependencies
- choice 2
    - Title
    - Dependencies
    …
- choice n
    - Title
    - Dependencies

Note that this structure is assumed for some reasonable value of *n*. The dependencies for each choice within a category must be consistent. This means that we must check that the dependencies graph for any given configuration file is in fact a tree, i.e. a non-cyclic, connected graph. So, when a user loads RosConnect, a configuration file must be parsed and the graphical user interface is then populated from this parsed information. After parsing this information from the configuration file, we run a checking function that ensures the structure of the file is as shown above and that the dependencies graph is consistent.

3.2 Options Chosen to Options Saved

After a configuration window has been populated in our graphical user interface, a user has the ability to choose among the options (or choices) for each category as shown above. Given that the user has chosen some set of options, the Options module must correctly read this user input and save the input so that if the GUI is closed, then these options are reloaded upon starting RosConnect again. We must ensure that these saved options correspond to available configuration window options upon such a reload.

To do this we invoke a function call that checks the correspondence between these options, and if a failure occurs, then we load the configuration window without the saved options showing. Additionally, we will display a warning message to the user in the logging console.

3.3 Options Saved to Vehicle Communication

Once a user has selected a set of options from the configuration window, then the user might either choose to start the simulation or run the vehicle. If the user chooses to run the vehicle, then the Communication module must take input from the options saved and create a set of launch file parameters to be sent to the vehicle. Each element from this set of parameters must correspond to

an option from the configuration window and the parameters must be correct in terms of the dependency tree. Our system will take these options saved and create the parameter set by validating the correctness in terms of the configuration window and dependency tree.

4. **Usability Testing**

The last form of testing of our product is usability. We are going to handle this by letting our clients use the system to see if they can easily follow it. We are going to also have people that don't know what is going on with our project. The goal of usability testing is to gather information from the user to help make the product better and ensure that we have met all of the requirements that the clients need.

Testing to make sure that anyone is able to use our product is a major part of our projects since we are making it easier for users instead of working with the command line. This product will be used by multiple high school students so we have to design the user interface so high-school students can have no problems using it. We need our product to be easy to use so the clients are not having to spend time explaining how to use it in their summer camp and can focus on teaching the students about autonomous racing.

When testing the users they will have to show that they are able to do the following tasks:

- Selecting Values
- Saving a Profile
- Loading a Profile
- Clicking Car
- Stopping the Car
- Running the Simulation
- Exiting the GUI after the user made selection but didn't save it.

This is all of the major functionality all the kids in the camp are going to have to be able to follow so we want to make sure it is clear on how to do everything so there is no confusion. While having the students complete the tasks above we will not provide them with a whole lot of direction to see how hard it may be to find certain aspects of the GUI. This will help us by letting us know about how detailed we need to be in all of our documentations. If we need to be more specific that is good to know so we can give step by step directions.

Some of these questions we will ask are:

- What did you find difficult to find and why.

- Was there any time that it seemed like we should have inputted more information to the console so you know what is going on?
    - If so when?
- What gets printed to the console makes sense and is easy to read?
    - If not, what didn't make sense and was hard to read.

While conducting these tests we will have a developer watching and taking notes on what they see to help us gain what we should change. This will give us more information if the user testing it doesn't feel comfortable telling us that they struggled with a part. At the end of the testing the development team will have a short survey for the user to take to give honest feedback on what they would like to see improved and what they would like to see get taken out if anything. We will take these changes into consideration. While taking it into consideration we will be working with what time we have left. If we do have time to implement it we will get that into the next version of the product.

4.1 Testing

Since we are about five weeks from the semester being over we are going to have to act on the testing fast to ensure that we have plenty of time to create new versions for students to try if possible.

- Week of March 30th First round of testing:
    - In this testing we will have our clients to go through and try working all of our functional requirements to see if we have everything that we have agreed to.
- Week of April 6th:
    - In this phase of testing we will have other students and even possibly high school students test the system to see how easy they are able to interact with this product.
- Week of April 13th:
    - This is when we will show off the product with any changes we were suggested to make by either the client or the testing of other students and possibly the high school kids.
- Week of April 20th:
    - This is going to be reserved for final testing to ensure that we are going to get exactly what students and our clients want to see in the product.

5. **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!