CS CAPSTONE PROJECT ARCHIVE

MAY 30, 2020

SIMULATION AND TOOLS FOR AN AUTONOMOUS RACE CAR

PREPARED BY

GROUP 30

SIMULATION AND TOOLS

RYAN CRYAR _____ Signature _____ Date

ARIN REINSCH _____ Signature _____ Date

ENRICO SUNDJOJO _____ Signature _____ Date

CONTENTS

1 Forward

This project is divided up into three parts that are going to need continuing: simulation, devops, and testing.
Simulator needs to be worked on continually throughout the years that GFR remains an active team for FSAE. Where the simulator goes depends on the direction of the team, but in the near future what needs to be done is to create a way for the simulator to have an automated testing environment with the teams devops and testing environments. Every part needs to be communicating with each other.

2 Introduction

This project was requested by the Autonomous system team within Global Formula Racing, in order to develop the simulator and develop tools to assist in the building of the autonomous race car. This is an incredibly important part of the autonoumous system software, because without the simulator, the vehicle software goes untested and is unpredictable. This way we can test the software that gets written for the vehicle, and we can continually develop and test to get the best outcome when competitions start.
Our client is Global Formula Racing, a team of students from different disciplines coming together to build two race cars: one with a driver, and one driverless. These cars then compete in Formula Student, a gathering of all teams from all over the world to compete with each other.
Our members are as follows. Ryan Cryar develops the simulator, and was in charge of picking the new high fidelity simulator in place of the previous simulator that was in use.
We were directly supervised by two technical leads, Kyle O'brien and Conner Kurtz. Each subteam has a technical lead in charge of the subteam in order to maintain structure and support the students developing each subsystem. Our two technical leads assisted whenever they could, making much needed decisions and assisting with questions.
In spring term, we had to switch operations to be completely online. While this effected a lot of aspect of the team as a whole, our part of the project did not have much change. While the construction of the car has come to a halt, our portion was already completely online and only writing sfotware. So while competitions are on hold, we were still able to continually write software for the cars.
For the next team, use our documentation to assist in the continual development of the simulator and other tools. The main thing that needs to be done is to hook up every portion of the project into one environment for continual development and testing of the vehicle software.

3 Requirements Document

3.1 Purpose

The purpose of the simulation and tools team in Global Formula Racing Team is to provide an accurate representation of what the car will perform in a certain set of parameters. Parameters must be fully tested to be fully accurate with the result that then going to be useful for the Autonomous System Team. These parameters are anywhere between the design of the car to the differences in track contours. The tools that will be provided can be anything requested; such as, data management, method to optimize the simulation software, or increasing user accessibility.

3.2 Scope

The scope of this project is only within the driverless sub-team of Global Formula Racing that focuses on their Simulations and Tools system. All other components outside the driverless sub-team are not within our scope of work.

3.3 Product Overview

3.3.1 Product perspective

The product in this case is the simulations and the tools that are being provided for the driverless vehicle. There is no specific end delivery which is going to be rolled out for simulations. They will be run on an as needed basis and will be produced and optimized as best as possible. Tools are also rolled out on an as needed basis.

3.3.2 Product functions

The function of the simulations is to provide accurate, real time simulations of the car and the different parameters/-tracks required to gain an understanding of what the car is going to perform.

3.3.3 User Characteristics

The characteristics on the users part, is to create a functioning UI in order to easily change some of the parameters should they be needed to change. It is also needed to have the simulation run quickly and efficiently in order to minimize problems of understanding and slow simulations.

3.3.4 Limitations

Some of the limitations are that we have to not deviate within our specified architecture. Which is most likely not going to happen anyway, but could pose a potential problem should an idea arise that we want to implement but cannot due to architecture limitations.

4 REFERENCES

Kyle and Connor, are going to be our main resources on how to go about certain simulations. There are also past examples on what simulations were ran, and how they were ran. Since there is a set architecture, this makes it easy to document and read about how things work and where they work. The main reference document is also a great reference for understanding and learning. A lot of work is going to be done through the Robotics Operating System (ROS), so the ROS documentation is also a great reference for continuing work.

5 SPECIFIC REQUIREMENTS

5.1 External interfaces

In order to interface with the simulations, they need to be ran within Robotics Operating System, and through our specific architecture. The simulations themselves will require a UI for interfacing with different parameter intakes, and also different tracks. These parameters are data that has been fed through machine learning algorithms to run different tests.

5.2 Usability Requirements

Requirements for Usability are that the simulation can be modified in such a way that it can change parameters and tracks on a given request. Upon receiving these requests the simulation should successfully recognize the new parameters and should change the outcome of the simulation. These outcomes could be good or bad depending on the parameters. The UI must be usable enough so that other members of the team can run simulations based on the requests. For example, if a driver wants to see their average times over a simulation, that will be input as the constraints and the expected output should equate to the average run time.

5.3 Performance requirements

The simulations must run quickly and efficiently. Optimization of the simulations should be done within the CUDA framework or the OpenACC framework, as the system runs on a Nvidia GPU. Both of these frameworks provide a great way to run calculations in parallel and run them extremely quickly. Not all calculations should run in parallel, as that could make it so the system is run inefficiently. The requirement here is that intensive calculations such as contouring calculations, should be run in parallel to quickly calculate and continue on. Efficiency of the simulation is heavily dependent on the run time of code running calculations for other systems on the car. Simultaneous Location and Mapping (SLAM) for example, handles the compilation of global positing and sensor data to determine where the car is on the track. This and similar systems will need optimization before we are able to run simulations effectively in real time.

5.4 Design Constraints

We are limited in our design by a number of factors. Firstly we have semi-limited hardware resources available with only one compute server for the entire GFR team. Meaning that we cannot make this tool too cumbersome. Secondly, we will be working with fairly large data sets meaning we will likely need to handle the data through a pipeline and compute each estimation in real time. We must also consider the wide variety of tracks that the team will encounter in global competitions. Fortunately per eSAE standards all tracks are flat, meaning our physics engine will not need to account for elevation changes that are common in race tracks.

5.5 Software System Attributes

The system is going to be running Ubuntu 18.04, and also running on top is going to be ROS Melodic that runs the driverless simulation. We will be building the simulator using an already established physics engine such as Unity, Unreal, or Physx. We will be adjusting our decision based on the ease of integration into our run time environment and overall modularity of the engine. Furthermore, this system will need to be modular, with the ability to easily incorporate and display data from a variety of sensors and systems on the car.

6 APPENDICES

Some of these requirements are subject to change due to unforeseeable circumstances that might arise.

6.1 Assumptions and Dependencies

We are assuming that we do not have to take action in other parts of the project. We are a purely software team and will be focusing purely on the driverless simulations. We are going to be depending on the tech leads to be dictating whether something is finished to standard. Deadlines are also going to be decided by the tech leads in order to conform with the proper deadlines for the team as a whole. We can assume that each track is relatively uniform so we will not need to account for much road noise on the track that could potentially affect the performance of the car.

7 Design document

7.1 Changes Made

Change Made:	Made By:	Date:
Removed section about server migration as that was not a step that was needed.	Arin Reinsch	4/22/20
Changed CI tool from Bamboo to Jenkins. Changed SCM tool from Bitbucket to GitHub.	Arin Reinsch	4/22/20
Updated CI flowchart to represent the new structure.	Arin Reinsch	4/22/20
Kept in all of simulator stuff as that was relevant to the previous term for Carla, but the new simulator is on hold so there are no updates for the new simulator.	Ryan Cryar	4/22/20

7.2 Introduction

7.2.1 Purpose

There are three components that have been chosen for implementation, and now are being looked at for design of the implementation. The first component that is being designed is the Simulator. The three choices before were a self implementation, gazebo, and CarMaker. CarMaker was chosen for this implementation because of the valuable capabilities it brought, as opposed to the other two implementations which did not have the benefits that CarMaker had. For our continuous integration and continuous deployment workflow Bamboo and Docker will be used. Bamboo allows for builds integrated with Jira, Bitbucket and Docker. Bamboo will allow the team to validate their code and ensure a clean master. Using Docker we can both validate builds and run numerous instances of the simulator. The last component is working with the interface of the simulator. This project prioritize testing intertface to be implemented before other components to be working since each output might get affected by the change. Moreover, our team is required to communicate with other sub-teams to integrate their data output of the interface.

7.2.2 Timeline

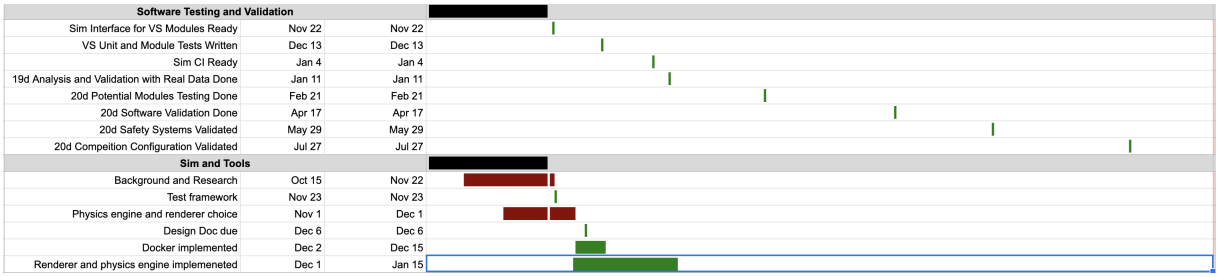

Fig. 1. Timeline

7.3 Design Components

7.3.1 Viewpoint: Simulator

7.3.2 Choice

The choice that was made for the Simulator is carMaker. CarMaker had a lot of really nice functionalities that are going to be very useful for implementing our simulator.

7.3.3 Server design

The first component of the design is that since there is only one license, we must use it on a server that can spawn multiple processes of it so it can be used by many members of the team at once. This will be achieved by running docker on our server, in order to spawn multiple environments of carMaker. This process will be discussed in later sections of the documents. This will allow carMaker to be spawned into its own environment with each user running their own environment on the server.

7.3.4 Vehicle Model

The second component of the design is to implement the vehicle model within carMaker. This will be done in the carMaker GUI, as it allows you to define each of the components as a standalone component and hook together via the GUI of the program. This will be done by the vehicle dynamics team, so they can define each component standalone, and test each component standalone as well. For example, tires and chassis are two separately defined parts of the vehicle, but come together within the GUI and allow for a full model to be assembled.

7.3.5 Vehicle Dynamics

Once the second component is done, the third component is defining more of the vehicles dynamics within the IPGKinematics software package. This package interfaces with what was defined within the carMaker GUI, and allows for a more substantial physics model to be defined. What IPGKinematics does that carMaker does not, is that it defines specifics such as load weights on the car model, spring dynamics, and chassis load. These are more detailed dynamics that are extremely useful to have given that we might need to change parameters such as load on different sections of the car. Should these changes be necessary we can see how it effects our times of the laps.

7.3.6 Track design

After that component, the last component that needs to be done is the track, and the autonomous definitions. The track can be done via the same carMaker GUI editor as the vehicle model. There can also be cones that can be placed into each side of the track, which is exactly what we need for the track definition. There can be multiple tracks so we will take advantage of the track builder to make these tracks.

7.3.7 Autonomous definitions

The last part that needs to be implemented is the autonomous definitions, which is the most complicated part. carMaker comes with an integrated lidar system, which can be attached to the car model. This basically performs the basic need that the system requires, which is sensing the cones. What needs to be done after is implementing a camera that sees the cones to determine what is a cone and what is not, which is not currently implemented into the carMaker platform. This is going to need a separate implementation that can interface with carMaker. The same is needed for the trajectory calculation, as it is calculated via dulaney triangulation, and that is also not a carMaker defined operation. These will need to be implemented as a separate part, via ROS to integrate within carMaker.

All of these components will make up the physics engine, and allow for the output to the renderer which is built into carMaker.

7.3.8 Virtualization and Product Management

7.3.9 Workflow

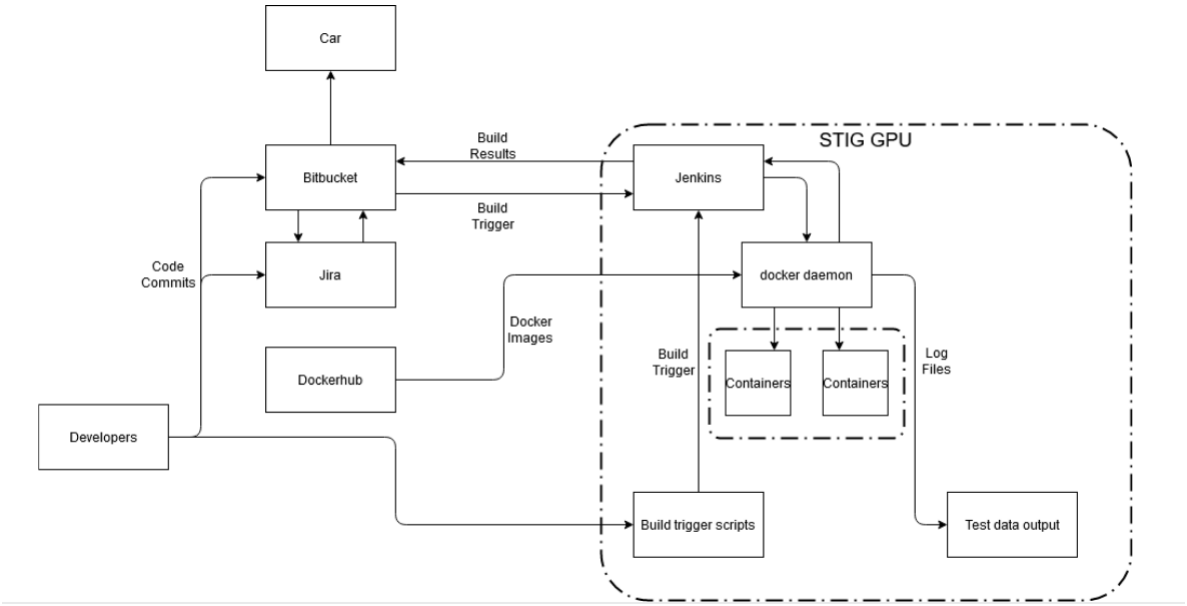

Fig. 2. Continuous Integration Workflow

7.3.10 Docker

To effectively host the many simulator instances that will be needed to generate sufficient data to test our car we will be using Docker. This tool offers the best performance of the virtualization tools that openly available. Since Docker can be run natively on Ubuntu it can be setup with minimal overhead. Moreover, we can omit the hypervisor and virtualization interfaces that are required in a Windows environment.

In order to create the dynamic containers that are needed by the GFR team there will need to be a docker image created. This image will contain all of the dependencies that the simulator requires to run. Creating this image requires that we have the simulator fully operational. To ensure that the environment is ready for the new version of the simulator we will configure the necessary containers using our current version of the sim.

Docker images can be created by first installing Docker on the Ubuntu GPU server. Once Docker has been installed a Dockerfile will need to be created. This file will be used to construct containers, containing information from the host environment such as: working directories, necessary packages and commands that are needed to run the simulator. Once a base image has been established it will need to be optimized with a Docker exclude file. In which files that not necessary to run the simulator will be omitted. This will reduce the overall launch time and size of each container, allowing for more instances to be run if needed. Finally there will be resource allocation configuration that will be needed to fully optimize each container. Docker allows for CPU time, RAM amounts and GPU usage to be manually configured. This will allow us to further control how each container is run.

Container management will need to be done by an admin on the GPU server, however, sufficient resources should be available at any time to accommodate the needs of the global team.

7.3.11 Continuous Integration

Validation of code through testing is crucial to maintaining a reliable code-base. Jenkins is an open source continuous integration (CI) tool which offers build and unit tests along with integration with GitHub. Firstly, Jenkins must be installed on the Ubuntu server. Where a directory will need to be created to hold server configurations. A Jenkins server instance can then be run and accessed through a local-host connection. Eventually this will need to be accessible globally, further configuration will be needed to make the web interface accessible. After accessing the web interface and completing initial setup, the Jenkins server can be pointed to the GitHub repository, gfr20_ci_ws. A trigger can be set to run tests whenever a pull request is made. Jenkins will need a Docker container to run the build test. A container will always need to be available to run tests. Once tests are completed the results will be automatically posted to both the Jenkins web interface and the pull request on GitHub.

7.3.12 Viewpoint: Framework Testing

7.3.13 Reason

The main reason behind framework testing is to improve the data flow and ease of access for an output. GFR wanted a system that could retrieve data from all ROS system into a database and inject previous outputs into the simulation when needed. The first important step of having a working framework is to have a Universal Data Format from all the system. In this case, GFR team sticks with using the Unified Robot Description Format (URDF) as the ROS; framework's UDF. URDF will use an XML file as an output for all ROS system; including the simulation system. It is important to have a UDF, so that all the data can be zipped into a package without any concern of running into compatibility. For example, Simultaneous Localization and Mapping (SLAM) does have a vector output that could be parsed individually as a system into the simulator as a 3D model of the location of the car. Other systems have different parameters; such as, image output, kinematics output, and distance vector input (LIDAR), which also has different file format that I need to implement to each of them to be used in the simulation.

Another important part of creating an effective framework testing method is to have data classification of each output from the system. The classification of data must be implemented in the back-end of the system and built into the GFR server (ROS server). It is important to have a classification system in every system, so that each output would be persistent and no errors will occur in the future. In this case, we will use Machine Learning Classifier, which is a built-in system of ROS. A machine learning classifier is useful since GFR is planning to implement a machine learning into their server. Moreover, it is easier to have a machine learning system into the server to avoid code smells when implementing data classification for each data recording system. The classifier will be written in python and it will classify outputs; such as, Kinematics Data (Speed, Wheel Speed, Torque), Tilt and Yaw, Camera output or LIDAR output from the simulation.

7.3.14 Interface Definition

Once all the data are classified by the machine learning system and have its Universal Data Format, we will implement rostest as a medium of the framework testing. The GFR simulation system, which runs with gazebo is currently using roslaunch to launch the simulation, and rostest is compatible with roslaunch. Rostest will enable us to test each packages in the server and. However, to implement data retrieval, storage, and injection, it would require modifying and writing into the CMakeLists.txt file.

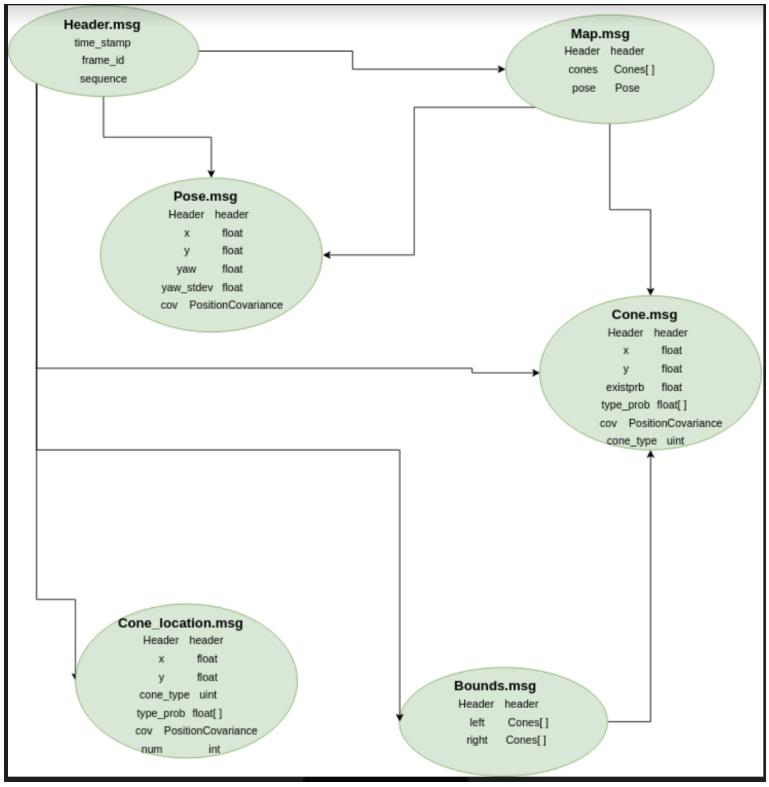

Figure 2.3 [4]

Each of the simulation system will have a message file that will receive data and formatted into URDF or XML file. Figure 2.3 explained required data to be reformatted and documented properly: Car position in the simulator, Mapping environment, cone location, boundary estimation (with noise), trajectory, and dynamics (acceleration and velocity). Each of the data output must have a file receiver (.msg) to format with dedicated frequency. It is important to have a universal data format and organized data sets that other group members could interpret easily.

8 Tech Review

9 Introduction

Simulation and Tools is made up of several different technical functions that improve Global Formula Racings ability to accurately develop a formula style racecar for competitions. There are three functions that are going to be focused on in this technology review: a physics engine, a database, and a machine learning for data implementation.

10 Physics engine approaches

10.1 Background

The way the physics engine plays into the simulator is by providing calculation output to the renderer. Our current setup is using gazebo for our physics engine, and rviz for our visualization package. This setup is a modified version of team AMZ from formula student, who has allowed us to copy their simulator to make modifications to fit our needs. This simulator however is limited to the functionality it comes with. This year we have decided to rewrite a major portion and go with a few different options for our simulator. This tech review will be focused solely on the physics engine part of the larger simulator.

10.2 Criteria

The criteria for the physics engine is that it must be able to output the vehicle dynamics calculations for the renderer to output, and it also must be able to model the vehicle such that the dynamics can be adjusted should they need to be. This criteria is rather loose as physics engines can be very basic to very complicated, but what we need for it to do is just to run calculations on our current vehicle dynamics equations.

10.3 Approach overview

There are three possible approaches that are being looked at for the physics engine implementation.

One option Gazebo, which is the engine we currently use. Another option is CarMaker. And finally a self written implementation through Robot Operating System (ROS).

All three implementations will have the same car dynamics defined within the physics engine itself in order to properly calculate parameters such as: drag, yaw, roll, and slip. This is not a complete list, as the car dynamic equation is still being modeled.

10.4 Approaches

10.4.1 Approach One: Gazebo

Gazebo is our current physics engine that is being used in conjunction with rViz to display our current simulator. Gazebo already has an implementation with our current dynamics model. This is one of the benefits that gazebo has when comparing it to other softwares.

Gazebo has a great way of definiting vehicle dynamics. It has a builder where the user can use a nice GUI and implement the design through CAD and also through the use of formulas. The design has already been implemented, and also currently runs in conjunction with ROS and rVis.

This approach basically does not need much done to the engine itself, besides some parameter editing. But Gazebo also does not provide some of the functionality that could be wanted in the future.

10.4.2 Approach Two: CarMaker

CarMaker is a industry software for testing and visualizing vehicles. One of their main benefits is because they are an industry level software, their package includes a lot of different functionalities that are useful to autonomous vehicle testing [**?**].

The issue currently with acquiring this software, is licensing. Because they are a larger industry wide company, their licenses are extremely expensive. They do have the option for license leasing for students, but that aspect still needs to be explored. Should this be a product we use, we will need to come up with an option that allows us to use one license and have a shared space to run multiple instances of it.

CarMaker is also a full fledged software that we can use for both the physics engine, as well as the visualizer. It comes with a lot of the things right out of the box that would be useful to take advantage of. For example, it comes with it's own lidar system that can be modified in several ways such as: using a 360 view, turning off certain parameters, and using different tests to view how well it performs. This is one feature that is very useful for performing validation and to also have to not reinvent the wheel.

The software can be easily integrated into Robot Operating System, and provides support to use the software to its full capabilities within the ROS environment, which is extremely important to keeping everything within one system and integrated correctly.

It also comes with a very comprehensive vehicle dynamics engine. The vehicle dynamics are defined by the user and can be edited as well as redefined at a later point. This is a useful thing to have as the vehicle model is still in development, so we can still test on last years model and see where we can improve things. The setup is similar to gazebo in the sense of drawing and inputting the parameters of the dynamics.

A disadvantage that comes with this software is because it is so full fledged, it could be very bulky to run as there are so many features that aren't used. More research needs to be done with this as hands on in order to verify.

10.4.3 Approach Three: Self implementation

The final approach is a self implementation written in C++. This comes with several disadvantages, as it will never be as fully featured as something like CarMaker. It will not be as optimized as CarMaker, and it could be more difficult on our end to completely rewrite something.

This in mind, a self implementation is considered, because it could be used to implement exactly what we need. This is very useful as something like Gazebo might not include a certain functionality that we want to track or to put into our simulator. With our own implementation, if something is not included then it can be written and quickly implemented.

Another reason to consider a self implementation, is because of integration. There is no need to worry about weather something is compatible with ROS, or works with our current visualizer. With the self implementation, it would need to be written to actively be compatible with the current visualizer. There would really be no other option.

One disadvantage that is very large that could pose a very difficult problem is that this implementation will require a large amount of time. With one person working on it, there is a complete possibility that the promised capabilities might not be deliverable due to programmer knowledge.

10.5 Discussion

While a choice is still unable to be made due to not being able to go fully through CarMaker, due to licensing, should a choice have to be made at this time CarMaker would be it. CarMaker provides everything that we need as well as more functionality should we want to take advantage of it.

Compared to the other approaches, it has fewer disadvantages and also much more advantages to it. The only disadvantage that could pose a problem the size of the other approaches problems, is the licensing issue with it only being on one machine. But there is already a workaround being worked out for that where we have a server to run it on and do work via SSH on it. The other only disadvantage would be that it could run slower than the other approaches due to how large the software is. While it might have accurate results, it needs to be considered that a slower simulator might not be one that we want to have.

The main decision that is being taken into account is how well will it work with our current stack. This will have to be decided once a full license is acquired and it is able to be tested. Should CarMaker work well, which is anticipated, it will be decided to go with it.

Another aspect that effects the decision is how featured is the approach. Each will bring its own unique features but there are a few that must be the same across all options. Each approach must have: the correct physics being calculated for the renderer to output, must be able to be quick, and must be able to run with ROS and the visualizer. With these

key components in mind, other features are definitely a plus. One feature that is very useful to have is a GUI that helps define the car dynamics. CarMaker and Gazebo both have this, where you can draw the design and also have the option of inputting specifics. While this is useful, it is not necessary. As the self implementation might not have a GUI, but it will have a specific spot for vehicle definitions to make it easier to implement.

One other aspect that effects the decision is timing. Ideally, the approach that is chosen should be one that can be completed in a reasonable amount of time and would yield results quickly. The completed approach should also be correct to provide accuracy to our simulations. Thus a self implementation is not the most ideal approach due to the amount of time and possible error that could happen. This is another reason why CarMaker might be the ideal choice, as it will have next to no incorrect calculations, as well as faster to complete.

All of these reasons are why CarMaker is possibly the best choice for the physics engine's approach, should we chose to implement this, then we can yield more accurate, and more fine tuned results due to this.

11 DATABASE MANAGEMENT

11.1 Background

There are two main parts of the project that generate a tremendous amount of data: the vehicle running and the simulations. This data needs to be put into a unified data format, and also put into a database so we can easily access it should we want to run data analysis. The format that this is going to be in, is going to be a data format that is going to be generated by all of GFR applications/vehicle stacks. Anywhere that generates data will have this format, which will then be inserted into the database.

11.2 Criteria

The main deciding factor for this, is how much storage we are able to use and how fast are we going to be able to run these queries. This is because the data that will be inserted into the database could be in the hundreds of gigabytes. So a database management system will first need to be able to handle the amount of storage, and second be able to run these queries quickly.

11.3 Approach overview

There are three applications that we are taking into consideration. The first is Microsofts SQL server, this is a huge enterprise database that many companies use. The second is MongoDB which is a general database that is very popular. The final is MYSQL, which is another very popular database that is also open source.

11.3.1 Approach One: Microsoft SQL Server

The main advantage that Microsofts SQL Server has, is that it is managed by a huge company. This way it is trustworthy and used by many people. In fact, 98 of the 100 fortune 100 companies rely on SQL server [**?**]. Another main benefit is that it is incredibly fast and reliable, which is very important for our uses. A disadvantage is that it requires a license to use, which for our purposes would run about $1000 to $13000 depending on the option we went with [**?**]. However, should Microsoft be willing to give us either a discount or let us use one as a sponsor, it would be very easy to implement with our current server stack. It also would open up the use of Azure should we decide to go that route.

11.3.2 Approach Two: MongoDB

MongoDB is a general purpose database with some very interesting features. It allows you to use JSON files within the database to allow for quick storing and querying, which could be extremely useful given the need for us to be able to insert a file with a unified data format [**?**]. The queries are also implemented in a JSON format which will allow easy integration with our server stack and to interface with our applications that will be using it. Another benefit of this is that the queries are much faster because they are JSON. The downside is that they are not as big as Microsoft, and licensing could be a problem as it is unknown how they provide their enterprise application to big companies.

11.3.3 Approach Three: MYSQL

MYSQL is an open source database that is extremely widely used due to its open sourceness. It also, like MongoDB, stores and queries data through JSON, which is a huge benefit for it being very fast and easier to integrate into applications and our server stack [**?**]. A disadvantage is that it doesn't have a free option to try out, it has only licenses which run from $2000 to $1000. This is a huge disadvantage because it is ideal if we do not have to pay for this in order to save the limited money that we have.

11.4 Discussion

All three options have their strengths and weaknesses, but MongoDB and Microsoft SQL Server seem to be more fully featured and more useful for our specifications. Microsoft is more likely to give us a free copy as they have expressed interest in sponsoring us in the past. We have also been looking at using Azure for some of our machine learning needs. With that in mind, Microsoft SQL Server seems like it would be better suited for our needs.

12 MACHINE LEARNING FOR SIMULATION

12.1 Background

In order to make use of the simulator, we want to make use of the data that it outputs and how it is inputted. From all of the data that we have, we want to run machine learning on all this data to put into the simulator and see what trend the data is following overall. The implementation of this is going to be self made and put in direct communication with the simulator to listen to data that we provide and output the visualization of that machine learned data. This will help us understand more about what is going on with our laps overall in more than just timing, but also what goes wrong too.

12.2 Criteria

The criteria for the framework that we are going to be using is also very loosely defined. Mainly it comes down to ease of use, and how well it is optimized to run on large datasets. Optimization is key because of how quickly we want to run these frameworks. While it would be nice to have extremely accurate data predictions, it needs to be run within a shorter timeframe than is truly optimal. Like over a day rather than over a week.

12.3 Approach overview

There are three frameworks that are being looked at. The first one is Tensorflow, which is a very widely used framework that has many functionalities. The second is pytorch, which is a end to end machine learning framework that is also very widely used. And finally MLflow which has many functionalities that could be very useful to us.

12.3.1 Approach One: Tensorflow

Tensorflow is a widely used framework that is easy to use and learn. One of the features is how easy it is to build models which is very useful for training and getting up and started very quickly [**?**]. It also is well optimized so that it can run very fast and be able to be used at enterprise scale. Another benefit is that you can deploy it through many languages, so it is much easier to integrate into our system.

12.3.2 Approach Two: Pytorch

Pytorch is another widely used framework, primarily using python. One of the main functionalities is all of the tools that it can provide. Because so many people work on it, there are many useful tools that could prove useful[**?**]. While the main use is within python, it also has a c++ implementation which would make it very easy to integrate within our system, which is already written in c++. The downside is that it is a bit harder to learn due to all of the possible tools that it has to offer. The syntax is a bit more complex and it could take more time to directly implement.

12.3.3 Approach Three: MLflow

MLflow is a full open source machine learning lifecycle platform. It has very robust user interface which makes implementation really easy. You can easily use multiple languages with it and also track the implementation in the environment of your choice. It also provides a way to organize your projects by using a standard format for each project. This would make it much easier to track which piece of code using the implementation is doing. It also provides a very easy way to create models through the MLflow models packages [**?**]. The downside is that because it is open source it could be much longer to run as optimization might not be a key feature of it like other implementations.

12.4 Discussion

Looking at these three approaches, the framework that makes the most sense to go with is Tensorflow. The reason is because it is very optimized and is very easy to learn. Another huge benefit is the amount of support that it has because of how widely used it is. There are many quickstart guides, and many people on campus that could provide support with it. The quick model building is also a huge gain as learning and building these models is one of the hardest parts about learning new frameworks. It is very useful to have a framework that can be quickly learnt and start being used due to our timeline.

13 Framework Testing

13.1 Background

The main reason behind framework testing is to collect all the output from the simulation system and could be re-injected into the simulation if the output is most ideal.

13.2 Criteria

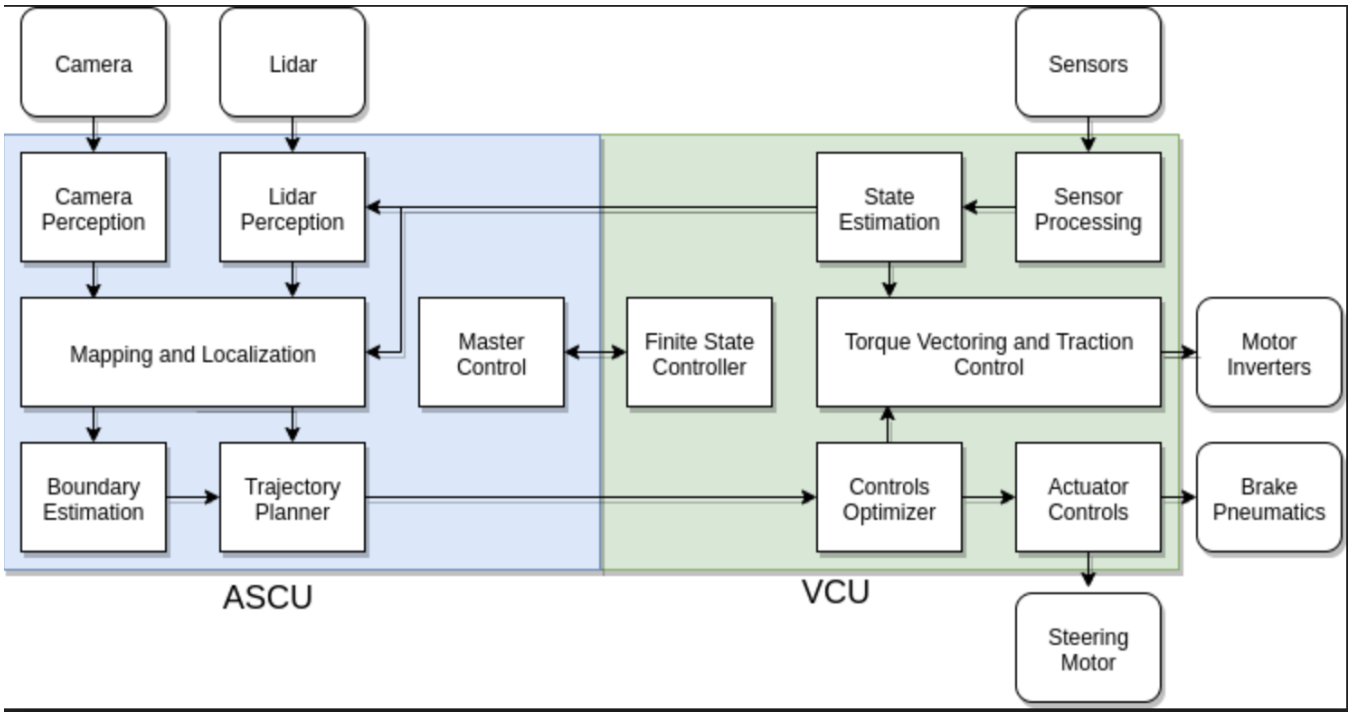

Figure 13.1

In Figure 13.1, I'm responsible learning each data that flows inside the ASCU column, which includes: Camera input, LIDAR input, and Simultaneous Location and Mapping System (SLAM). Fortunately, our client had given us more input on the given standard URDF script by ROS from the previous team that worked on it last year, but it needed more functions added to it. For example, the Engineering team is trying to add tilt and yaw to the data collection, and it could be useful for a new application that could retrieve and send data to and from the simulation system.

13.3 Approaches

13.3.1 Create a Script from ground-up

There are several methods in framework testing. Our client wants a method that could make the group able to test past and future data with ease. Creating a new script will cause a freedom on how and where the output of the system going to be retrieved. Furthermore, the script must be able to inject past outputs to the simulation system if the output is an improvement.

13.3.2 Using rostest from ROS Utilities

The advantage of using rostest is that it is built into the ROS system. As a result, it is easier to implement and more efficient to use the built in framework testing. Rostest would create an XML output, which is an URDF format of ROS and it is convenient for the workflow since it's a familiar format for the GFR team

13.4 Discussion

Our client asked to view the benefit and disadvantage of creating a new script or revamping the rostest. The GFR team have been using rostest, and our client wants to know other options other than rostest itself. Creating a script to be a middle part in the workflow, and it is easier to customise how the testing of the framework going to suit to our client.

14 CLASSIFICATION OF ROS DATA

14.1 Background

Our group must be able to understand every output from every ROS system. As a result, it is important that every system need to have a universal format to make other team members and our client to understand the meaning of the output.

14.2 Criteria

The classification of data must be implemented in the back-end of the system and built into the GFR server (ROS server), which would be beneficial for other group members to not have any problem in unreadable data in the GFR database.

14.3 Approaches

14.3.1 Creating a Script from scratch

Each ROS system sent out different types of data. For example, the simulation system have different types of output; such as:

- Kinematic Data (Car speed, Wheel Speed, Torque)
- Tilt and Yaw (In the process of being implemented into the Simulation)
- LIDAR/ Camera input

14.3.2 Use ml classifier

ML CLASSIFIER is a classification system built into ROS. It is short with Machine Learning Classifier. Our client have an idea of implementing machine learning into the server. It will beneficial for our client if machine learning is implemented earlier in the schedule, so that there would be another priorities in the project.

14.3.3 Autoware

Autoware re-use existing code to classify if the output is the one the database desired. Furthermore, it is convenient to use since it can be run with or without GPU [6]. However, it needed a license to run autoware for GFR, and there is a possibility that getting the license approved would cause the schedule to shift behind.

14.4 Discussion

Our client have said that framework testing and classification were the priority for the first quarter of this project. As a result, it is wise to find a method that is time efficient and easy to understand for other and future group members. Implementing autoware into ROS system would be time consuming since it requires sufficient documentation reading, and license agreement. The most time efficient method is using the ml classifier since it is built into ROS system, and our client wanted Machine Learning to be in the system in the future.

15 Recommended Technical Resources for Learning More

The ROS website was super helpful for learning more on how to build up data transfer pipelines in Robot operating system. ROS also has a stackexchange that was super helpful for learning about what other people have done.
For Carmaker that specifically is a little bit harder. We have all of the documentation that IPG sent us on the google drive folder, but there isn't much as far as learning more online.

I found that the most helpful resources were the vendors official documentation. Both Docker and Jenkins provide extensive documentation for their products, with installation and usage guides for most configurations. These documents were good for specific implementation details, very similar to a man page. For higher level overviews I referenced a number of articles.
For setting up the docker environment:
https://stackify.com/docker-build-a-beginners-guide-to-building-docker-images/
For jenkins configuration and usage:
https://www.jenkins.io/pipeline/getting-started-pipelines/

16 Conclusion

Through this document there were three functionalities of Simulation and Tools that were looked at, physics engine, database management, and machine learning for simulation. Each option had three different approaches that were looked at and discussed to make a choice on what approach we would be going with for each. Each functionality now has an approach that is going to be implemented with our system and be of great use to us.

17 Expo poster

18 Project Documentation

18.1 CarMaker simulator

Lots of the CarMaker documentation is run by IPG. There is a folder in the GFR workspace in Simulation-¿CarMaker that has all of the documentation for getting all set up.

The overall simulator can be run by doing roslaunch gfr_sim.launch gfr_common, this will spin up rviz and also every data transfer node so that the data can be correctly transfered between each part of the simulator. RQT can be used to monitor this data transfer by running the command rqt right after the simulatior has been spun up.

Here is the data transfer diagram for the TF.

This diagram represents how data is transferred within the simulator for just the TF node. Below is the full data diagram flowchart that classifies each node and where it transfers data to.

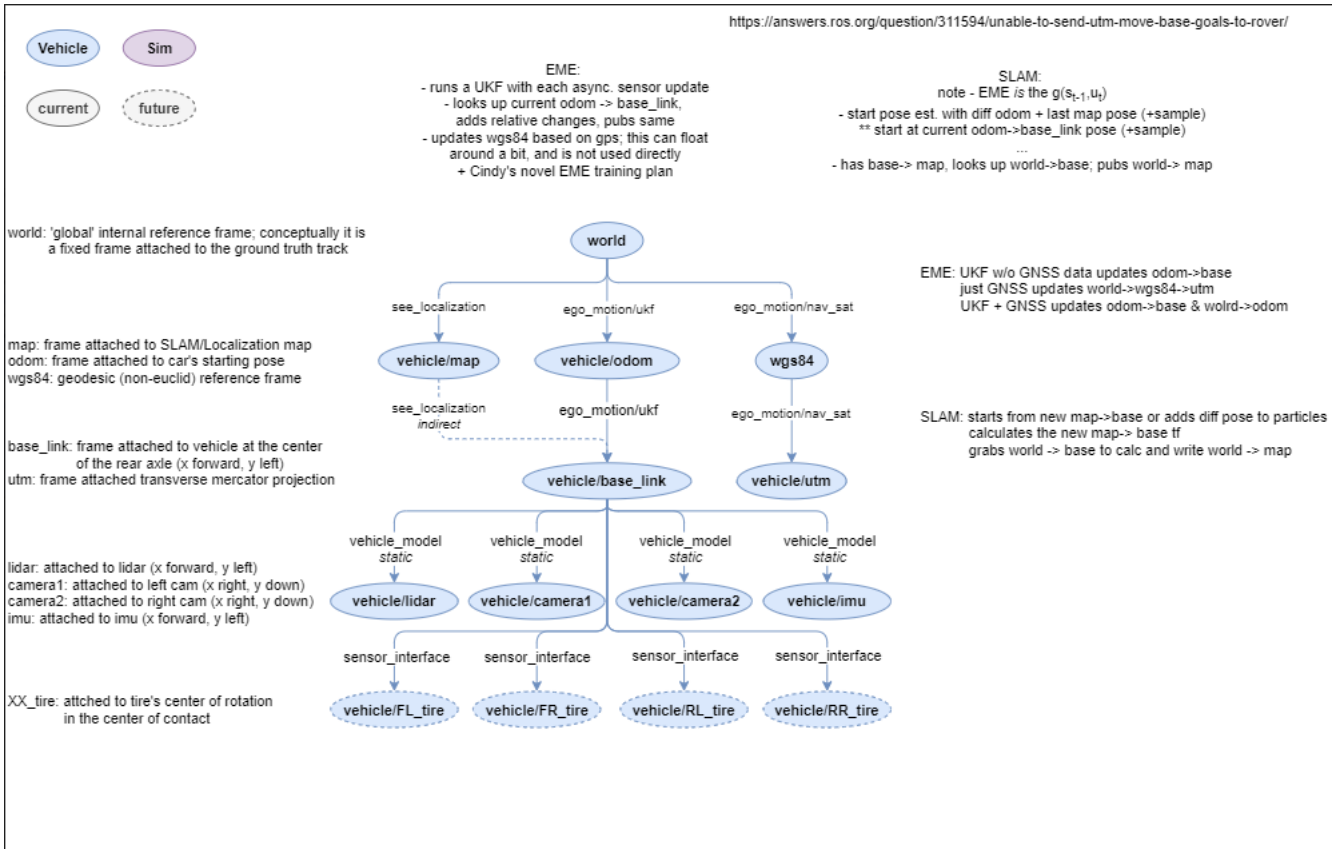

Fig. 3. Data transfer diagram for TF

These diagrams represent how the overall simulator is structured. Each part is transferred through the topics and messages.

18.2 Continuous Integration Configuration

18.2.1 Install Docker

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
sudo bash -c 'echo "deb [arch=amd64] https://download.docker.com/linux/ubuntu
disco stable" >> /etc/apt/sources.list.d/docker-ce.list'
sudo apt-get update
sudo apt-get install -y docker-ce
sudo systemctl status docker
```

18.2.2 Add docker to root user group

```
sudo usermod -aG docker ${USER}
su ${USER}
```

18.2.3 Pull docker image

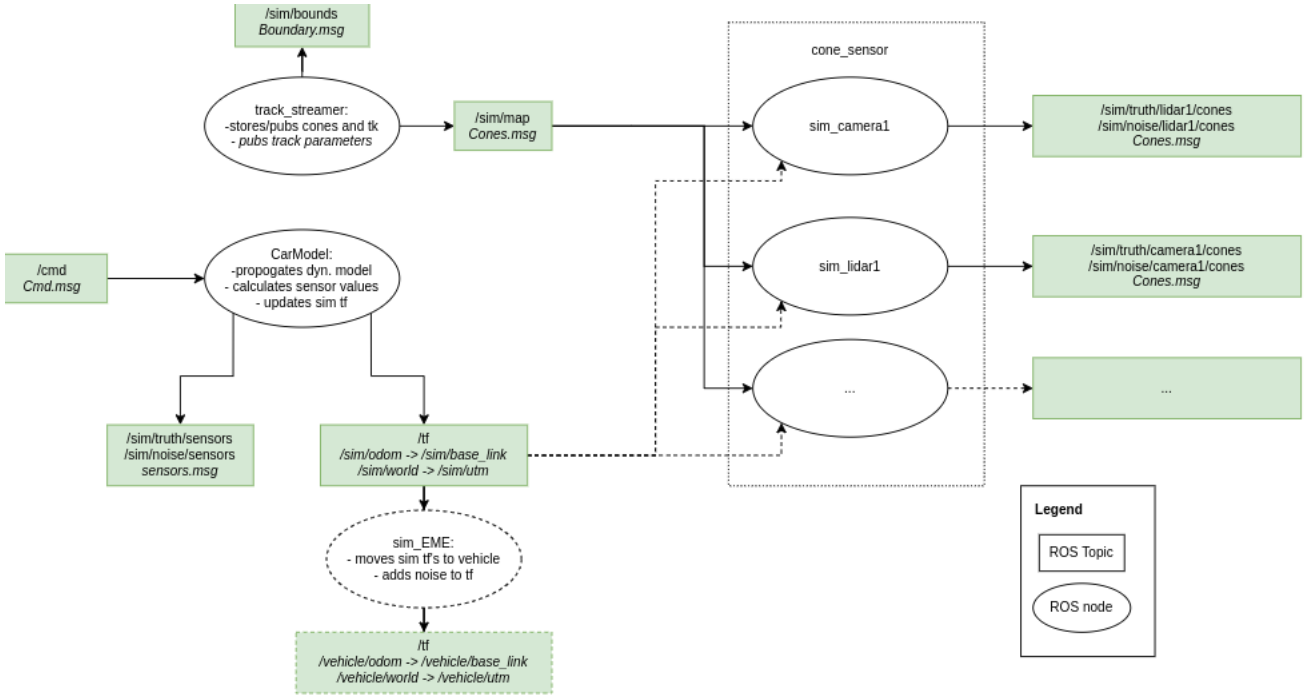

Fig. 4. Data diagram for robot operating system nodes

```
docker pull /arinreinsch/gfr20_ros_image:latest
```

18.3 Install Jenkins

```
wget -q -O - https://pkg.jenkins.io/debian/jenkins-ci.org.key | sudo apt-key add -
sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/
> /etc/apt/sources.list.d/jenkins.list'
sudo apt-get update
sudo apt-get install jenkins
```

18.3.1 Initial Setup

- Copy the key provided by the installation script
- Open a browser and navigate to http://localhost:8080/
- Paste the key into the field and begin setup
- Ensure the plugins listed below are selected. These can be changed after installation
- Finish the setup steps

18.3.2 Add SSH Keys

- ssh-keygen -t rsa -b 4096 -C “gfr_email”
- Hit enter for no password and default location

18.3.3 Add Public Key to GitHub

- Run: cat /.ssh/id_rsa.pub
- In GitHub, open the dropdown next to the user profile
- Select ‘settings’
- Select ‘SSH and GPG keys’
- Select ‘new SSH key’
- Paste in the key from above and click ‘add SSH Key’

18.3.4 Add Private Key to Jenkins

- Run: cat /.ssh/id_rsa
- In Jenkins, select ‘credentials’
- Under ‘Stores Scoped to Jenkins’ select the drop-down next to ‘global’ and select add credentials
- From the ‘Kind’ menu select SSH Useranme with private key
- Select ‘enter directly’ and paste the private key from above and select ‘ok’

18.3.5 Create a New Build

- To access the jenkins web interface, open a browser to http://localhost:8080/
- Select ‘New Item’ and create a freestyle project
- Under ‘Build Triggers’ select GitHub hook trigger for GITScm polling and Montiry Docker Hub/Registry for Image Changes.
- Under ‘Pipeline’ -¿ ‘definition’ select Pipeline Script from SCM
- Ensure ‘SCM’ is set to Git
- Under ‘Repositories’ set the Repository URL to the target git repo URL
- From the ‘Credentials’ menu select the appropriate credential that was previously configured
- Set the ‘Script Path’ to the Jenkinsfile path in the git repo
- Click ‘Save’
- Select ‘Build Now’

18.3.6 Plugins

Other plugins will be installed by default, the following are required in this configuration.

- SSH Agent
- GitHub Branch Source
- Docker-Build-Step
- Docker
- Build Token Trigger
- CloudBees Docker Hub/Registry

19 Weekly blog Posts

19.1 Ryan

19.1.1 Fall Week 3

Progress

This week I have been working on getting everything organized for my team and I. This includes getting one of our team members caught up, starting all the documents for the requirements and the problem statements. I set up our team meeting for both the TA and for the sim and tools meeting with our Team lead. We are making progress on the requirements document and will try to finalize it before friday to make sure that we can chat during the meeting to see if anything else is needed.

Problems

No problems have been occurring, everything is running smoothly.

Plans

By friday we will have our requirements document done. During our full team meeting we will understand more of the architecture of the driverless system for our car. Next week will be focused on getting used to ROS and getting used to the architecture of the system.

19.1.2 Fall Week 4

Progress

This week we finally divided up work for the simulator. I am going to be working on developing the physics engine that is needed for the simulator to properly run.

Problems

No problems as of yet. I have been running into some issues with the simulator running on an old version of gazebo which is our physics engine that we are running right now. But I have a meeting with someone on sunday that should help that go along.

Plans

I have a meeting on sunday with someone on the german side of things that is going to help me with the simulator, and going to start working on the replacement to gazebo.

19.1.3 Fall Week 5

Progress

This week was focused entirely on research on possible approaches to developing the physics engine for the simulator. I narrowed it down to a few approaches. Carmaker, Gazebo, and our own implementation. Our own implementation would require some more setup and going through ROS to do some fancy footwork. So far, I have some basic setup code running a small simulation environment with my own implementation.

Problems

I can't look into carmaker fully as they require a license to go through their software, and getting a license is extremely expensive. But will be looking into getting the licenses for discount or for free as we are a student team.

Plans

Going to be contacting the people at carmaker to try and see if they are willing to give us some licenses at a discount or for free. I am also going to try to finish the barebones stuff for my own implementation of a simulator.

19.1.4 Fall Week 6

Progress

This week we made more progress on deciding what software we should use for the physics engine. It is looking like we are going to be going with carMaker over a self implementation or over our current simulation stack. We are still deciding but going through the documentation and using the example that they sent us it was clear they had a lot of things that we want in a software.

Problems

There hasn't been any problems this week, mostly just looking into how to get access to a carMaker license.

Plans

Finishing up the tech review and starting the design document. Going to be getting the carMaker license soon, and going to be trying to integrate it into our software stack to see if it will work with everything. Also going to continue to work on getting my simulation code to run and compile with all of our interfaces, currently there are no errors but it isn't outputting anything. Going to address that with Kyle in meeting tomorrow.

19.1.5 Fall Week 7

Progress

This week it has been decided that we are going to be focusing mostly on the testing framework for the simulator. So far that has not gotten much work on there. CarMaker was installed on our servers so now I have been working on getting that to work with our current vehicle model.

Problems

No problems as of yet. It seems like there isn't a specific package for processing lidar within CarMaker but that is going to be investigated more.

Plans

Docker is also currently being put on the server, once that has been fully integrated we can start spawning multiple environments with it and using CarMaker to be a full simulator. We also need to work on whether we can use CarMaker as a full simulator by trying to implement the seeing cones and lidar processing into it. Currently there is no progres or updates on that because we just got access to it. But it should be easy to figure out whether it is feasible or not.

19.1.6 Fall Week 8

Progress

This week has been focused on the test framework entirely. I wrote one publish and subscribe test for the boundary planner so that the current vehicle stack can update their tests on it. I have also been tinkering around with carmaker and trying to see how it will fit our needs. Unsure as of right now whether we will keep carmaker on but that is still to come. We also have finished the design document, and are going to be attempting to finish up before the turn in date.

Problems

Mostly trying to figure out how the code interacts with each other. It was hard to write that one test because I was very confused really how to go about it. I didn't know how ROS transfered data. But once I got the hang of it it was much easier. Other than that there were no other problems that arose.

Plans

Since we wrote the boundary test, we are gonna finish the cone, map, and pose messages since they should be much easier to implement now that bounds is implemented. That needs to be done by next week so that we can begin testing more over winter break so we can validate as much of our system as possible.

19.1.7 Fall Week 9

Progress

This week was pretty light as far as work goes since it is thanksgiving. But I finished the test framework and am now validating all of the tests to make sure they work.

Problems

When I go to run the simulator it seems to die immediately. So that is going to be next on the table.

Plans

I am going to finish up validating tests and look into more bugs when needed

19.1.8 Winter Week 1

Progress: Over the break, I was able to complete a lot of work. I got to be able to finalize the picking of a new physics engine. It also comes with a built in renderer so we were able to skip a large chunk of time having to get a separate renderer setup. I decided to go with Carla because of it ability to be integrated really easily with robot operating system. It also has a lot of functionality that can be used that the previous simulator cannot have. This simulator is also open source so it will be a lot easier to find documentation and others who have had similar problems.

Problems: The current problem is getting the simulator to be running on the server. Because the simulator is optimized for vulkan, it requires a different driver than the one that is installed on the server. But because ubuntu does not have a good package for switching drivers it has become a larger issue.

Plans: The first thing is to finish up getting carla on the server, which needs to happen as soon as possible. I need to talk to the current tech that is in charge of the server because I am not allowed admin privileges so that needs to happen. Once the server has vulkan and is running the simulator, it needs to be bridged with ROS and then vehicle models ported over. once that is finished we can finally start the actual simulated testing that needs to be done.

19.1.9 Winter Week 2

Progress: Currently, there has not been much progress as there has been issues with our server so there has been some work on the server.

Problems: There has been lots of issues with our server and the simulator related to drivers. There is a driver that the simulator needs to run with, but the issue is that the driver is incredibly difficult to install on the server. I cannot move forward until that gets resolved, and this is a huge blocker to try to get by so it might need to be installed by a professional IT team.

Plans: I am going to try to talk to the CoE IT department to get the driver installed on the system. Once that is done, we can finally get to finalizing the simulator and make it so it can be used by everyone on the server.

19.1.10 Winter Week 3

Progress: This week I finally got our simulator up and running and am going to be making the changes needed. The server still doesn't have the correct drivers so i am going to try to talk to IT to see if they can add the drivers for us.

Problems: The drivers still dont work at all.

Plans: I am going to be making a new track to load into the simulation project, our other team member is going to be putting the vehicle model into it. I am also going to be configuring the sensors so they do what we need and also allow for us to do dulauney triangulation on the cones.

19.1.11 Winter Week 5

Progress: This week I have been trying to finalize stuff for alpha release. So I have been getting the lidar to display properly and also have it be outputted to a ros bag so that we can use it for visualization in later benchmarks. I also have been doing map modeling and car modeling, which has not been working well but one of my team members is going to be helping.

Problems: The map and the car modeling has been an absolute nightmare. So one of my team members is going to pick that up as they have more experience with modeling.

Plans: I am going to finalize alpha functionality, and also start doing more testing of the simulation environment. We are also going to get jenkins setup to spin up the simulation in a docker container, so we can start doing testing with our vehicle team.

19.1.12 Winter Week 6

Progress: Progress has been huge this week. I outputted a ton of data from running the simulator and we are validating whether this is fitting out needs. Everything is looking good for getting the simulator all put together. We have also found a way to get unreal working, as we are going to run the engine on my home server then compile it and keep it on our team server.

Problems: None so far

Plans: Once the maps have been generated, I can write the script that will generate the car and will also generate the track parameters. I also need to configure the camera sensors to make sure they are properly detecting the cones.

19.1.13 Winter Week 7

Progress: This week was full of good progress. We got map making process streamlined so we can finally get the maps we need for testing purposes. I also got my home server set up so that my team can use that to compile everything together.

Problems: The connection is really weak so that could pose an issue.

Plans: We just need to get the map created then i can finish up the rest of the simulator. After that we are basically ready to start validation.

19.1.14 Winter Week 8

Progress: This week hasn't been much progress as we had to complete design reviews and there have been a ton of midterms, but all of that is over now so we can finish up the maps.

Problems: I cannot compile our new maps that Arin made because my computer does not have enough ram to run Unreal.

Plans: Arin has some extra ram that I can borrow so I will be able to compile everything on my computer. Once that is complete i can finally start coding up the simulator and finally get it all integrated.

19.1.15 Winter Week 9

Progress: This week was mostly figuring out how to get carla to work on the server. I got some extra access to ram through Arin but the same issues were coming up with everything.

Problems: The server still was unable to compile carla through the Ram that Arin got me, still crashing at the same breakpoints so it is assumed that this is a unreal error not a server error.

Plans: I am going to talk with my tech lead to see what the next steps for the simulator is and whether or not we are going to be going through with using carla or going back to the old simulator.

19.1.16 Winter Week 10

Progress: not much progress has been made this week, however we were able to figure out what my next steps are for working on things after carla.

problems: none so far just have had a lot of schoolwork to do

plans: this week I should be able to finish up more of converting messages to the new messages we are using for the sim. Everything is still going well.

19.2 Arin

19.2.1

Fall Week 3 Progress: I have begun setting up our development environment. The tool we are working on requires a native install of Ubuntu 18.0.4 and does not work with remote access. I have set this up using a boot manager and two drives on my laptop. I have also been reading through the organizational requirements and configured git and bit bucket for use on the new OS install.

Problems: Our sub team meeting for this week with our client was cancelled so we are missing some directions in our project, making the requirements doc a little difficult to write.

Plan: We have a work session on Saturday to get our development environments fully setup as well as hopefully meet with our client to discuss the requirements and timeline for the project.

19.2.2 Fall Week 4

Progress: This week I finished setting up the development environment in Ubuntu 18.04 and got bitbucket and ROS configured. We were also able to meet with our client again during our Saturday work session. Where we discussed in more detail the systems that allow the autonomous car to navigate the track.

Problems: We ran into some issues setting up the simulator software. A file was removed from the package as it was no longer needed. However, there were hundreds of references to this file causing a number of install errors. This was done by our team and should be resolved shortly.

Plans: Continue to research different physics engines and read more on driving simulations. We will also be working closer with our client to further narrow our project scope and assign team members specific rolls.

19.2.3 Fall Week 5

Progress: This week I have continued reading up on various rendering techniques. I have also been working with my client on narrowing my search scope for specific tools, primarily looking into proscan and unity for rendering.

Problems - None to report this week, all is going well.

Plans - Going to continue to work on deciding on a rendering tool and working with my client to get any needed licenses. I will also be working with them to determine all of the assets that

19.2.4 Fall Week 6

Progress: This week I worked with my client to determine that we will likely be using Unity for the front-end rendering of our simulator. I also learned that I am going to be assisting with server-side deployment and virtualization of our simulation tools. I began researching the best methods, tools and practices. Finding that Docker will likely be the best tool for this, I setup a test environment and have been working with the tools some.

Problems: No major problems to report. I have run into some technical issues with my Docker testing machine. It is seeming that a Windows update earlier this week changed some of my virtualization settings which is now not allowing Docker hub to run. I am still working on narrowing down the cause.

Plan: I am going to continue working on the Docker issue described above, I may need to do a clean install but that is a last resort. I will also work with my client further to set a timeline and plan out how we are going to be configuring Docker on our GPU servers.

19.2.5 Fall Week 7

Progress: This week I have become much more familiar with Docker. I have established a workflow for creating dockerfiles, base images and containers. I am currently testing and researching the best methods for configuring a dockerfile for our specific simulator.

Problems: No major problems.

Plans: For this week I am going to be prioritizing assisting my teammate in the implantation of the test interface. I will be getting more information about what this entails in our weekly work session tomorrow morning. I can also ideally get a working docker file made, graphical output is not required at this time which will make the initial configuration much easier.

19.2.6 Fall Week 8

Progress: Since last week I have continued to work on implementing a continuous integration platform. I had been working on creating an instance of Bamboo. However, our team may be moving from Atlassian products due to licensing concerns and various implementation difficulties. Since this development I have been exploring other CI tools, primarily Jenkins. I was able to easily setup a Jenkins server on my test VM and run a few tests with a test repository.

Problems: I was having difficulty getting a Bamboo server setup on any of my test machines. I ran into a number of communications issues with the Java virtual machine even on fresh installs of Ubuntu. After talking with my client, we determined that may be best to move to another, open source platform.

Plans: I am going to work on setting up a Jenkins server to interface with our working repository. I am also going to do some more research into how to run Jenkins builds in a Docker container so that we can run multiple builds at once if needed.

19.2.7 Winter Week 1

Progress: Over winter break I was able to complete the setup of our ROS environment hosted in a Docker container, and published the file to dockerhub for easy transfer to GFR's remote servers. I also implemented a Jenkins pipeline script to test new builds that have been pushed to bitbucket.

Problems: Due to some changes made to our main Ubuntu server I am no longer able to access the Jenkins server that I deployed on 1/4/20.

Plans: Address the server access issue (some additional research required). Finish deployment of CI pipeline by publishing build results to the appropriate channels. I will need to consult with my sponsor on how they would like this feature implemented.

19.2.8 Winter Week 2

Progress: Worked with the team to create a plan for the rest of the term and assigned new tasks where needed.

Problems: We have been having issues configuring our GPU server to use the correct drivers needed for the physics engine to run. We need Nvidia driver 440.44 as it supports Vulcan, however we have not been able to get that driver install properly as it always reverts to 440.33 which does not support Vulcan. This is also causing issues with Unreal 4 studio as it requires Vulcan to run as well.

Plans: Continue to work on getting drivers installed on the GPU server. Once this is resolved I will start working on linking the physics engine into UE4. I am also going to begin adding tests more tests to the CI pipeline. After a meeting with my client we determined that it would be advantageous to add unit tests and some forms logging.

19.2.9 Winter Week 3

Progress: Jenkins build status notifications are now working in my test environment. Ryan has been able to setup CARLA and Unreal in their test environment.

Problems: Nvidia drivers still not installing on GPU server. Deployed Jenkins server has been having issues pulling the pipeline script from our bitbucket instance. These issues may have been related to some know issues with jira/bitbucket but I have not had time to test further.

Plans: Deploy pipeline script changes to primary Jenkins server. Continue to troubleshoot driver issues and hopefully begin work on rendering in UE4. Begin to research 3D modeling methods to create a model for use in the future simulator.

19.2.10 Winter Week 5

Progress: Driver issue has been resolved on GPU server, CARLA and UE4 are now able to run. Jenkins is now able to pull from bitbucket, found that it was pulling an older version which did not contain the Jenkins build script.

Problems: None this week!

Plans: Work on getting a map, cone and car model built in UE4 to export into CARLA. Continue to work on creating a Jenkins pipeline to run simulations in containers.

19.2.11 Winter Week 6

Progress: Wrote build script in bash using curl to remotely trigger Jenkins builds. Pulled down containerized Jenkins server and moved to a locally running instance. Create Jenkins service account on GPU server to hold data and separate SSH keys. Got access to teammates desktop to work with CARLA and the UE4 editor.

Problems: Unable to launch the UE4 editor from server the engine is still having driver issues. However, I can develop on teammates machine.

Plans: Finish configuring new Jenkins instance. Edit CARLA map with UE and get cone and car textures imported to CARLA.

19.2.12 Winter Week 7

Progress: Prepared for design review. Got UE4 editor working on Ryan's machine. Created road textures, map and converted cone models to .obj for use with UE4.

Problems: Using TeamViewer to access Ryan's machine cause issues with mouse input to the unreal editor making it impossible to navigate the view panel. I had to move the project over to my desktop and edit the maps in UE and then return them to Ryan's machine. This was a very slow process as the base project is around 10GB zipped.

Plans: Create additional tracks to run tests on, finalize Jenkins configurations. Add additional tests and CARLA build to Jenkins scripts.

19.2.13 Winter Week 8

Progress: Had design review. Built first map in unreal. Resolved a number of issues in the Jenkins build process, required modification of the docker image as the build is dependent on catkin which was not previously configured.

Problems: We are not able to compile our simulator on Ryan's server due to RAM limitations.

Plans: Trying to find more RAM for Ryan's machine. Build more maps. Integrate scripts with Jenkins.

19.2.14 Winter Week 9

Progress: Resolved some additional Jenkins issues. Wrote documentation on Jenkins/docker configurations in case we need to redeploy the system in the future.

Problems: Ran into an issue with entry points in docker. And I may need to setup a docker registry to effectively use my images pulled from dockerhub

Plans: Work on entry point issues. Finish documentation. More research on map building in UE4.

19.2.15 Winter Week 10

Progress: Preparing final deliverables.

Problems: Been very busy with final projects, haven't had much time to work on project.

Plans: Integrate tests written by teammates into the Jenkins build. Resolve container entry issues.

19.3 Enrico

19.3.1 Fall Week 3

I was unavailable for the second week of the term for personal reasons. However, I caught up with my teammates to discuss how and what we should do to get going. Thankfully, the first client meeting was this Friday the 18th, so I could be more involved in group discussion and decision. Ryan explained the required systems and programs to do this project (Global Formula Racing Simulations and Tools), which is installing Ubuntu 18.0.4, and ROS melodic to get started.

One of the issues I'm encountering is that I'm using a Macbook Pro for school, and I left my old Windows laptop back in Indonesia for my siblings to use last term. Installing Ubuntu in macOS would be difficult since it will take at least a day to configure and divide harddrives into to operating system. As a result, I need to have a second system to do all my assignments that I missed for a week. Another issue is that we are having a client meeting late in the week, and the project requirement paper draft is due later this Friday the 18th. We are going to have a brief understanding on the requirement and task we need to do for this project.

The plan is to either lend a laptop or desktop from a friend for this project or buying a desktop PC and running ssh on my Macbook Pro for this project. Fortunately, I contacted my teammates and our client about this problem. Thankfully, this project is still in the process of familiarizing with Ubuntu and ROS systems, which we're going to be presented in this Friday meeting. Furthermore, we're going to have a work session every Saturday from 11-5pm that we're going to catch up and be on the same page. Other than those issues, we are full steam ahead in terms of progress.

19.3.2 Fall Week 4

Week 4 have been extremely productive. Kyle and Connor have given us more information on what we are supposed to work on' furthermore, they created a timeline on when specific important deadlines are required to be accomplished. We met every Mondays, Fridays, and Saturdays. Each of the day have different agendas. Mondays usually correspond with having team meetings with the Autonomous System meeting. Fridays correspond with the individual meetings

with our sub-team in Autonomous System, which is Simulations and Tools. Saturdays are the most productive among the other meetings since it is a workshop meeting with our client, which they helped getting started and familiarize with the Robot Operating System (ROS).

Thankfully, I found the solution with my Ubuntu compatibility problem for my MacBook Pro. I decided that it is best to buy an external SSD to store Ubuntu OS. I ran into a little problem on keyboard and trackpad compatibility for Ubuntu, and accidentally installed the drivers over on the onboard NvMe SSD that contained MacOS. However, I sorted that out before this Friday the 25th meeting to be on the same page as others. We also researched on several physics engine to work on since Kyle wanted us to decide whether it is sufficient to work with the previous system or revamp a new physics engine. Unity and unReal were the most compatible and suited for this project. However, we

In addition, Kyle and Connor explained more thoroughly about our divided work in today's meeting. Ryan volunteered to work on the Physics Engine, Arin volunteered to work on the Rendering of the Simulation system, and I chose to design the framework of the system. Therefore, on Saturday meeting, we would have a clear idea on what we are supposed to focus on.

19.3.3 Fall Week 5

This week was more going towards individual work since there are fewer meetings that we need to attend. Previous Saturday cleared up most of the ambiguity of the project; however, since the simulation still hasn't work properly, we don't have a lot of measurements to look into. Especially, I was in charge of testing and data formatting for the simulation output. I only got to work with previous data from the previous year that GFR's team research. However, we talked to Kyle and Connor about the problem, and they come into a conclusion where most of the work needed to be done from other members; including Ryan that worked with the Physics Engine.

I'm focusing on data format management, and I looked into several skeleton script for data management to be stored into a file that can be used for other GFR members to look at. However, I'm looking into previous data to work with since the simulation is still in development and improvement; therefore, I worked with previous data, so I could be ahead when I'm needed to implement my script to the simulation output.

In a bigger picture, we're still clueless about the workload and schedule of what we needed to do in this project. We gained more information of what we're supposed to do in every meeting, but we were introduced to more problems and tasks to accomplish, which create more ambiguity to the situation. Fortunately, Kyle and Connor assign a lot of meetings so we could work as a bigger team and solve problems easier with them. Our plan is to work with other engineering capstone, and consult with Kyle and Connor with our proposed design.

19.3.4 Fall Week 6

We were able to get the simulation working; as a result, we can implement most of our assignments for the group. In the past, we used past system's output. We are able to use the output we created, and tinker with each of the parameters. Furthermore, I'm able to start on the framework testing and use all the research I've done in the previous week.

Plans and Problems: Our client assigned us to another side project as a group on creating a design statement for the team. Moreover, our client wants framework testing to be worked as a group since we were a little bit behind schedule. However, it is not the main priority for the project. We are still need to familiarize with ROS systems so that we could implement our assignment.

19.3.5 Fall Week 7

Progress: Ryan and I were assigned to focus on framework testing, which we were given information from Kyle last week. We implemented the first MSG file (Boundary) and will clarify with Kyle if he would want us to implement other MSG file in the same format.

Plans: Ryan and I are planning to separate our work on framework testing. We have 6 MSG files including Boundary that we've done to work, and we will separate those files accordingly. Moreover, we are not going to be seeing each other for next week (Thanksgiving Holiday); as a result, we planned ahead on assigning files to work.

Problems: None at the moment. There would be a problem arising if Kyle wanted a different format of MSG file that we already done.

19.3.6 Fall Week 8

Plans: We plan to finish the testing interface this week, so we could continue on testing the physics engine and docker integration. Ryan and I are working on the testing interface and Arin is still working on docker integration.

Problems and Progress: We had our testing interface due after Thanksgiving, which would be hard for Ryan and I to communicate and work together. However we still have 2 more days to work on the testing interface and hope that we're able to meet on the weekend after thanksgiving

19.3.7 Winter Week 1

Progress: Over the winter break, all of us were assigned to different task towards the implementation phase. Since there are a lot of holidays during the winter break, our team met Connor (our client) in different times. As in my progress, I was assigned to start on completing the noise nodes that produce ground truths for Cones. I had a meeting with Connor via Hangouts that he briefed what I needed to do to write the cpp code. We also had a meeting at the start of the week on Wednesday (1/7/2020) to discuss our progress and plans for the following week.

Problems: The only problem is that we have no idea what other members are doing and their progress. Furthermore, our schedule changed for this Winter term and we need a new meeting times for our client and TA.

Plans: We are transitioning towards the implementation phase; as a result, we are finishing up our winter break assignment. Furthermore, we are planning to meet up on Saturday tomorrow to discuss the implementation phase, and I would finish up the noise nodes assignment and make sure it's satisfiable.

19.3.8 Winter Week 2

Progress: Our tech leads have given us more in depth assignments towards this term. Connor (our client) have given me an extra assignment to do on the side from my main task. It is to create an optimized script for other groups, so they could integrate into the new interface without having to change some of the old features to the new features of the simulator. Currently, I'm still working on the Noise nodes, and it is in the finalization phase with Connor.

Plans: For this Winter term, we decided to work more closely with our client. Every Saturday, there will be a work session for us to work and ask questions with our client. Moreover, GFR have focused on 1-on-1 meeting for each group rather than a general meeting with all the sub-groups presenting their works since we are focusing on implementation this term. Furthermore, Ryan and Arin are implementing a new server, so testing could be done remotely for other groups to use.

Problems: Meeting times are hard to agree on because all members are having different class times than the previous term and it includes our client. Therefore, our meeting times will differ for each week throughout the term.

19.3.9 Winter Week 3

Progress: Met with one of our tech leads, and finalized one of my assignment. Furthermore, I volunteered myself to work on web development and help Ryan and Arin on integrating the new physics engine and CI server.

Problems: Currently no problems in my side. We're up to schedule and we're having progress from the previous week.

Plans: I'm trying to finish up creating a separate file to take the noise output of cones being read by the simulator, and start on integrating testing interface on the Continuous Integrating Server from Arin part.

19.3.10 Winter Week 5

Progress: Finally I finished up with all noise nodes. As a result, it is going to be used as a middle function to add noise to any car position, velocity, and cone positions.

Plans: Furthermore, Alpha is in few days, and our tech leads are going to schedule more meetings to discuss the alpha release.

Problems: A lot of projects are going to be due next week. As a result, it is going to be hard to schedule a meeting with our tech leads.

19.3.11 Winter Week 6

Progress: We finished the week by completing the Poster Draft and went to two meetings with our tech leads to discuss our different Alpha release timeline.

Problems: No problems at the moment. However, we expect we going to have some problems with merging our codes together in the Jenkins Continuous Integration Server.

Plans: We decided to invite our tech leads to our design review presentation on the 25th of February.

19.3.12 Winter Week 7

Progress: We've been integrating our code together, and started to compile what we did in this term. Moreover, we have been meeting with our tech leads more than usual to discuss our design review

Plans: We decided to work on our design review on the work session this Saturday, and also follow up with testing our codes.

Problems: None

19.3.13 Winter Week 8

Progress: I spent a lot of time preparing the design review presentation with our tech leads and team mates. Furthermore, our tech leads decided to apply more strict on uploading our codes to git. As a result, we needed to upload our code to git before the general meeting

Plans: Improving SLAM and boundary estimation would be the next step after our tech leads validate my noise codes.

Problems: Delayed on the progress of developing code since our design review was this week

19.3.14 Winter Week 9

Problems: None.

Progress: Currently working in pace with GFR's schedule on Alpha release. Going to start working on a new project next week

Plans: Our tech leads going to assign a new project and validate all testing interfaces ready for continuous integration server

19.3.15 Winter Week 10

Progress: We finally have our last meeting with our tech leads before the finals week, and we discussed our next term projects. Furthermore, we are planning to do our demo video separately and combined them into one big video because of the virus outbreak.

Problems: It's hard to meet in person since the virus outbreak limits our whereabouts; however, we plan to finish our individual demo video this weekend.

Plans: We plan to finish our demo video and submit our final codes into bitbucket for our tech leads to look at.

20 Conclusions and Reflections

20.1 Ryan

What technical information did you learn?
I learned a lot about how to write software for robotics. There has been a lot of useful real world applications that I have discovered when writing software for the car, and for the simulator. I learned great ways to transfer data around using ROS, and how to setup a simulation environment for a vehicle. What non-technical information did you learn?
I learned how to work in a team more effecively, it was really difficult to manage every section of the project and also communicate with the other teams within the project. I also learned how to take more direction, as my technical lead was mostly in charge of the things that I did, I didn't have free reign over what I wanted to do.
What have you learned about project work?
I learned that it is really difficult to be in a project with a large amount of people. There are so many moving parts it is hard to keep track of. But I also really learned how to effectivly communicate in such a large team. Emails, and scheduled meeting times really are key when working within a project.
What have you learned about project management?
Project managment requires a lot of moving parts, and it requires incredible organization skills. I felt very lost through it and had to work really hard to maintain a calendar and keep track of everything i had to do.
What have you learned about working in teams?
I learned that teamwork can be difficult. You have to really divide up the work so everyone clearly knows what they are doing. Constant checkups are also a great thing because then tracking progress can be so much easier If you could do it all over, what would you do differently?
I would start earlier on certain things. Because i felt like I was really running out of time by the end of it and I was starting to get really close to deadlines without having much done. I also think I would have stopped working on the part that really didn't work and took up about 3 months of time, that would have solved a lot of problems.

20.2 Arin

What technical information did you learn?

I have learned to work with a number of CI and virtualization tools. Most of my project was focused in docker and Jenkins. I am now very comfortable with docker and its supporting tools such as dockerhub and docker-compose. I also gained experience with Jenkins pipelines and exposure to the tool's potential. Having configured both the docker image and Jenkins server by hand I have learned the importance of making software easy to deploy and configure. What non-technical information did you learn?

I learned how to quickly adapt to an organizations policies. Specifically, with meeting attendance, email etiquette, and document management practices. What have you learned about project work?

I learned that with long term projects that it is very important to maintain a consistent work schedule. I found that scheduling time specifically for project work helped me to continually make progress. I also have a newfound appreciation for project update meetings. Having to frequently discuss the technical aspect of the project helped me find areas that I needed to learn more about and made me more comfortable with technical verbal communication. What have you learned about project management?

Using a project management tool such as Jira or GitHub makes tracking a large team much easier. As long as tasks and milestones are up to date there is a constant log of the project's progression. We did not get as much direct experience in this since our client provided us with rigid roles and tasks. What have you learned about working in teams?

I have learned the importance of over communication. Since our team had many meetings per week, we all knew what was happening within the team. Keeping everyone in the loop helped steer the project in the right direction and made sure we were meeting deadlines. If you could do it all over, what would you do differently?

I would make better use of past resources and personnel. Since GFR has been building software for their cars for years they have thorough documentation and experienced alumna. I feel that I could have better leveraged these resources early on to get a better idea of what features would be beneficial to the team. I would also be more involved with the rest of the GFR team as a whole. They hosted events weekly throughout the year and I was able to make it to very few of them. This includes the weekly work sessions. I was able to attend most of these but most of my work could be done from home so I was not present as much as I would have liked.

20.3 Enrico

What technical information did you learn?

I have learned using ROS and my C++ skill to be in tandem. Learning how to create nodes to certain ROS subscriber that capture different data from the simulator that could be altered with different functions. For example, add noise to data or alter different message types to any data.

What non-technical information did you learn?

This is my first time working with mostly technical engineers in a project; as a result, I felt more confident getting into meetings with tech leads or clients that might be useful for future use in a work place. It also taught me to respect and accept what I've been tasked to do.

What have you learned about project work?

Working in a huge project made me realized that communication is important to manage time and tasks. Especially, when there are multiple sub-groups in a project that need to communicate to keep progressing on the project.

What have you learned about project management?

I learned time management is important since I'm still a full-time student while working on GFR project and needed 12 minimum credits to be one. As a result, I need to set aside a certain day and time to work on the project.

What have you learned about working in teams?

Furthermore, communication to other team members is also important since this project is geared towards individual task.

If you could do it all over, what would you do differently?

I would have manage my classes more carefully if I knew that Capstone project would take a lot more time than I thought. I would've taken hard classes before taking Capstone or set aside some Bacc core classes to compensate the Capstone difficulty.

21 APPENDIX 1

21.1 Dockerfile

This file is used to specify the build environment for the simulator. An already built image is available for use on dockerhub at arinreinsch/gfr20_ros_image

```
FROM ros:melodic
LABEL MAINTAINER "Arin Reinsch <Arin.Reinsch@global-formula-racing.de>"

ENV ROS_DISTRO melodic
#Update container
RUN apt-get update -qq && apt-get -qq install --no-install-recommends -y apt-utils gnupg wget ca-certificates lsb-release
RUN echo 'debconf debconf/frontend select Noninteractive' | debconf-set-selections
#Install supporting programs
RUN apt-get -y install gnupg1; \
    apt-get install -y git; \
    apt-get install -y python-catkin-tools
#Configure ROS environment
RUN apt-get install -y ros-melodic-desktop
#Install catkin
RUN apt-get update
RUN apt-get install python-catkin-tools
#Set permissions
RUN useradd -m docker && echo "docker:docker" | chpasswd && adduser docker sudo
USER docker

CMD ["/bin/bash"]
```

Fig. 5. Dockerfile

21.2 Jenkinsfile

This file used to specify the script run by Jenkins in the docker container that was defined above.

```
pipeline {
  agent any
  stages {
    stage('Agent') {
        agent {
            docker {
                image 'arinreinsch/gfr20_ros_image:latest'
                args '-u 0'
            }
        }
        steps {
          script {
            sh '''#!/bin/bash
            ./install.sh
            '''
                }
        }
    }
  }
}
```

Fig. 6. Jenkinsfile

21.3 Simulator

There is not much code from CarMaker to put down here as the new simulator has not been developed enough to have any code besides the initial ros setup.

Listed below is the only segment of code from the new Simulator that allows for data transfer:

```
/* Name might be different after remapping! */
LOG("Initialize ROS Node '%s'", hellocm::node_name.c_str());

/* ROS initialization. Name of Node might be different after remapping! */
ros::init(Argc, Argv, hellocm::node_name);


/* Node specific */
RosIF.Cfg.Node = ros::NodeHandlePtr(new ros::NodeHandle); /* ToDo: */
ros::NodeHandlePtr node = RosIF.Cfg.Node;


/* Publish specific */
LOG("  -> Publish '%s'", hellocm::tpc_out_name.c_str());
RosIF.Topics.Pub.Ext2CM.Pub = node->advertise<hellocm_msgs::Ext2CM>(hellocm::tpc_out_name, 10);


/* Subscribe specific */
LOG("  -> Subscribe '%s'", hellocm::tpc_in_name.c_str());
RosIF.Topics.Sub.CM2Ext.Sub = node->subscribe(hellocm::tpc_in_name, 10, ros_HelloCM_CB_TpcIn);


/* Services */
LOG("  -> Create Service '%s'", hellocm::srv_init_name.c_str());
RosIF.Services.Init.Srv = node->advertiseService(
        hellocm::srv_init_name, ros_HelloCM_CB_SrvInit);


/* Parameter specific */
/* ToDo:
 * - Parameter must be set in all external nodes?
 *   -> currently setting in CM seems to be sufficient! */
strcpy(sbuf, "/use_sim_time");

/* ToDo:
   - getParam seems to return 0 even the value was received
 */
if ((rv = node->hasParam(sbuf)) == true && node->getParam(sbuf, rvb) == true) {
    node->getParam(sbuf, rvb);
    LOG("  -> Has param '%s' with value '%d'", sbuf, rvb);
} else {
    LOG("  -> No param '%s' is set", sbuf);
    rvb = false;
}

if (rvb == true)
   LOG("  -> Use time provided by Clock Server!");
else
    LOG("  -> Use system time!");

/*
 * Use cached parameters?
 * - http://wiki.ros.org/roscpp/Overview/Parameter%20Server
 * - should be used rarely to avoid overloading the master
 */
strcpy(sbuf, hellocm::prm_cycletime_name.c_str());
```

Fig. 7. Publishers and subscribers

22 APPENDIX 2

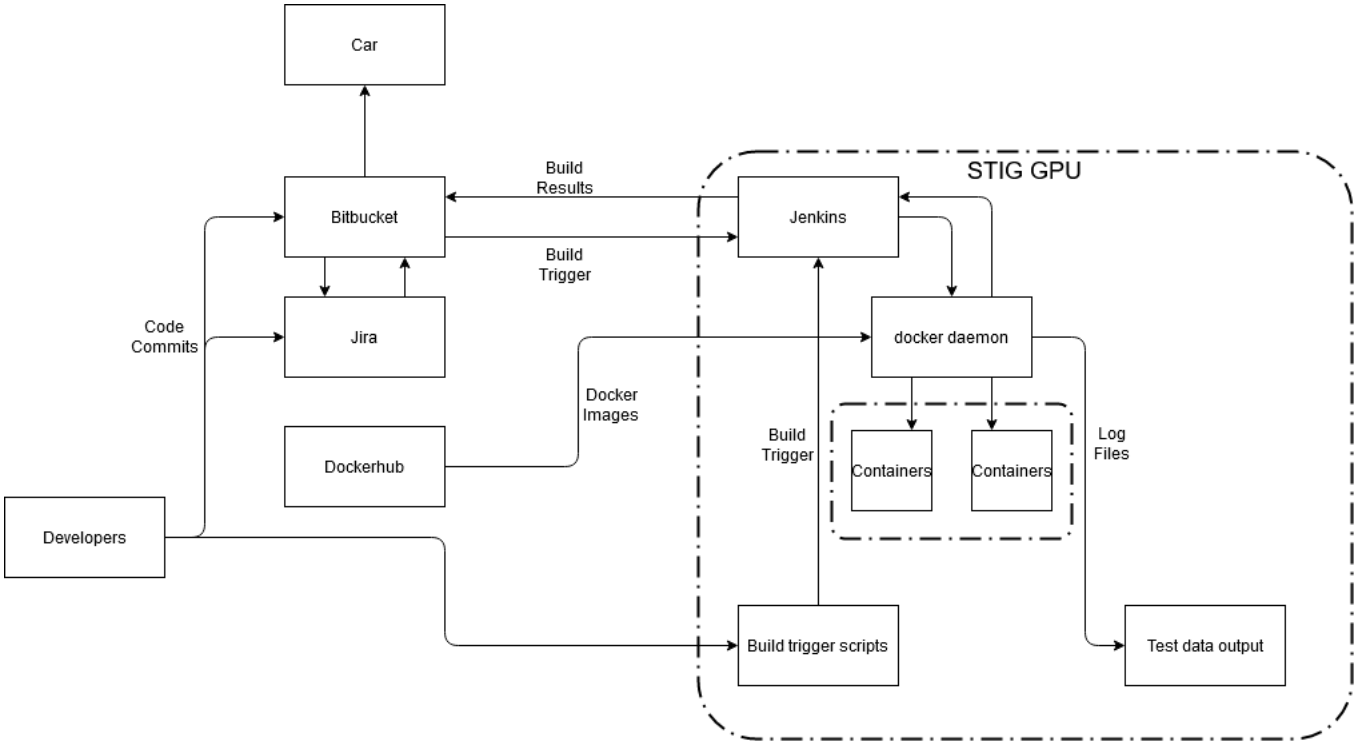

Fig. 8. Devops Architecture

23 APPENDIX 3

Code review feedback and responses:

Category	Description	Reviewers Comment	Action
Build	Could you clone from Git and build using the README file?	Yes, build is successful and README is clear.	No action taken
Legibility	Was the flow sane and were variable names and methods easy to follow? Does the code adhere to general guidelines and code style?	The design and implementation of various methods are very reasonable. The variable names are also following the same pattern.	No action taken

Implementation	is it shorter/easier/faster/cleaner/safer to write functionally equivalent code? Do you see useful abstractions?	In vehicle.cpp some function may implement as inline function but if it is not been called frequently, it is not necessary.	Inline functions here is not needed because they are called frequently elsewhere so they are not needed.

Maintainability	Are there unit tests? Should there be? Are the test covering interesting cases? Are they readable?	There are not unit tests. However I know there will be module tests for different ROS modules.	There are more tests being added as simulation gets developed in future Unit tests will soon be included in the Jenkins build. I have the build setup for the easy addition of tests. Once the testing framework is complete the two can be joined.
Requirements	Does the code fulfill the requirements?	Yes, the code full fill the requirements.	No action taken.

Other	Are there other things that stand out that can be improved?	The carnoise.h may have better place to placing?	Car noise needs to stay in that spot due to it having multiple dependencies.

Group 30

Category	Description	Reviewers Comment	Action
Build	Could you clone from Git and build using the README file?	Yes, I can clone from Git and it built easily. I already have the system installed and updated on my laptop so once I found the correct branches I had no issues.	No action taken
Legibility	Was the flow sane and were variable names and methods easy to follow? Does the code adhere to general guidelines and code style?	Yes, the files all seemed organized and pretty well separated out. Made it easy to follow, I didn't get lost as much as I felt like when I first looked at our code. Seemed like the naming was logical. Could use more comments in cone_sensor.cpp. That would make it easier to tell what the transforms are doing. Or where the data is being published to. How the noise is added… etc.	Cone sensor is being commented Data flowchart has been written

Implementation	is it shorter/easier/faster/cleaner/safer to write functionally equivalent code? Do you see useful abstractions?	No, I didn't see any abstractions other than the potential inline function that Yuchen pointed out. The code seemed to be broken up well, I didn't see any code that was repeated in other functions.	No action taken

Maintainability	Are there unit tests? Should there be? Are the test covering interesting cases? Are they readable?	No unit tests. I don't think you need them. However, if you were going to add them, maybe making sure what you are publishing after adding noise is correct?	Testing after noise is completed during new simulation setup. Unit testing is in progress, we are looking to add testing as part of the CI pipeline. However, this is dependent on the completion of the testing framework Testing are being done every new noise implementation is created, and we wanted to improve on how we implement noise into the msg type to make it more streamlined.
Requirements	Does the code fulfill the requirements?	Yes the code fulfilled the requirements. The team made progress researching a new simulator, updated the current simulator and starting CI for the team.	No action taken

Other	Are there other things that stand out that can be improved?	Nope, it all looked good to me!	No action taken

Category	Description	Reviewers Comment	Action
Build	Could you clone from Git and build using the README file?	Yes. The instructions for building the simulator and the CI elements were clear and easy to follow. The included scripts for installing dependencies and starting a build make everything pretty easy to do in just a few steps which is nice.	No action taken
Legibility	Was the flow sane and were variable names and methods easy to follow? Does the code adhere to general guidelines and code style?	Some of the code alignment in track.cpp was a bit off which made it a little hard to follow. Overall all the code was fairly clean and consisted of well named variables and a logical flow. More comments could be helpful explaining the general use of functions or blocks of code.	More comments added, alignment fixed.

Implementation	is it shorter/easier/faster/cleaner/safer to write functionally equivalent code? Do you see useful abstractions?	The implementation seemed to be well thought out and efficient. There didn't appear to be any places where there could be much improvement made to the code in a safety/speed respect. I particularly liked some of the error handling in the CI jenkins file, and the conciseness of the methods in the carnoise.cpp and cone_sensor.cpp files. Some helper functions for decomposing the yamlToCones method in track.cpp could be useful to clean up the flow a little. In a few of the files there are functions that are used repeatedly and it could be possible that they could be inlined to provide a boost to performance.	No action taken, as the yaml to cones is being worked on by someone else in the future.

Maintainability	Are there unit tests? Should there be? Are the test covering interesting cases? Are they readable?	There are no unit tests for the simulator since it sort of serves as the testing method for the other aspects of the code stack. Some simple sanity tests for the CI elements could be useful although I do not know if this is practical or not. Other potential tests that could provide some sanity checks would be interface	For tests run by CI we are working on finiding a way to make the build logs easily available. So that the results of a build or test can be manually verified if needed. By default Jenkins logs all of this data, but it is not easily accessable.

		tests making sure the correct message types are coming out of the correct nodes.	
Requirements	Does the code fulfill the requirements?	The code does fulfill the requirements laid out for the project. All of the parts seem to work well and accomplish their purpose well. I think the CI will be particularly useful to the team for automating builds and maintaining a functioning master branch in addition to a paper trail of builds and potential errors.	No action taken
Other	Are there other things that stand out that can be improved?	Nothing stood out as needing a lot of improvement but I think that more comments in the code would be useful. Ensuring that the formatting of the code is consistent would be helpful for increasing readability as well. As mentioned previously some of the repeatedly used methods like getting noise could potentially be inlined to save on stack frames and provide a performance boost although this might not be necessary.	Code is being cleaned up, and aligned properly Added additional comments to the docker and jenkins files to make it clear what each step in the script will do. Noise are added into the same cone_sensor file instead of having another node to fill it out; however, we are implementing another noise node for cone_sensor to take in cone id number