DSC 180 – Autoware Final Report

Jie Wang, Andres Bernal, Amir Uqdah

March 7th, 2021

1 Introduction

We are developing a 3D simulation environment representative of rich real world data to reliably test and simulate robotic agents in dynamic environments. This work is important because it can help streamline the remote development process and help others visually debug and evaluate the components of the robot they are working on.

To accomplish this we are using the Unity 3D game engine to replicate the Thunderhill race track and create debugging tools for others to visually evaluate their algorithms in real world scenarios. We are using the Borregas Ave. Track in order to run an actual simulation. Additionally, we will use the LGSVL simulator to virtualize the functionality of common sensors like IMU, GPS, Odometry, and Lidar devices. Finally, we plan on exploring the feasibility of using the Autoware.AI framework to assist with localization, detection, prediction and planning computations of the robot. The reason for choosing Autoware.AI over Autoware.AUTO is because the other students of our class are all working with ROS1 and Autoware.AI supports ROS1 while Autoware.AUTO supports ROS2.

2 Goal

The main goal of this capstone project was to recreate the Thunderhill Race Track and import it into the LGSVL simulator so that other teams were able to test out their algorithms that they created for the sensors and cameras that will be on the robot. The idea behind this was that since it's an autonomous robot that should be navigating on its own we wanted to make sure that the algorithms that are responsible for avoiding collisions and keeping the robot within the track worked properly before racing in the Thunderhill Race track. Making sure that the robot did not crash while doing a lap in the track was one of our main priorities since that's what the simulator is for instead of testing it out in the real track. Our final goal is to ultimately do laps around the west side of the Thunderhill tracks which is about 2 miles long as safe and efficiently as possible with the help of the previous steps mentioned beforehand.

3 Abstract

We were able to replicate the ThunderHill race track using the Unity 3D game engine and integrated Unity with the track and robot into the LGSVL simulator. Once the integration was complete we were able to see our robot with the Thunderhill Track as our map in the simulator. We were then able to virtualize the functions of the IMU, odometry and lidar sensors and RGB-D cameras to better visualize what our robot perceives in the simulation. Finally we were able to fully visualize what our robot sees with the virtual sensors using Autoware Rviz which displays the location and point cloud map of the vehicle and its surroundings.

4 Methods

In terms of building the Thunderhill Race track we approached this problem in a variety of ways. One method that we first tried was to use google maps/earth API to import the thunderhill track by selecting the coordinates of tracks. While we were to successfully import the Thunderhill Track using google map API, the track was blurry even after trying to improve the quality so we opted to try another method so that we can make this virtual track as realistic as possible. We were then able to find some existing data that we used to recreate the track in Unity which was significantly better in quality than our first approach of using google maps. So we decided to stick with the data that we found since it also had elevation data which made it even more realistic and looked very similar to the actual track due to the elevation data. We were able to also further improve this thunderhill track in unity by using a HD render pipeline that was provided by Unity and made the pixels less blurry and more to their actual color. In terms of why we chose to use the LGSVL simulator is due to the fact that it builds on top of ROS2/AutoWare and mainly because it's compatible with the Unity 3D engine that we are using when doing the integration between these two software tools. The methods that we used to create a virtual environment for our vehicle includes: building a track from unity into the simulator, bridging a connection between the simulator and Autoware through a port. Finally we gave the car commands on Autoware using 2D navigation goal and let the car drive to its destination.

5 Data and.Conversion

In terms of grabbing actual data, we were lucky enough to be given the thunderhill data in pcap form by students from another class also studying this course in unity. Our data that we fed into unity needed to be in the format of a PCD or point cloud data format. What we had was the thunderhill data in a PCAP file format which was a binary compression of many PCD files. To get this extraction, we used an online converter we found that someone had coded. We fixed the code up a little bit and was able to successfully do the conversion. Using the

PCD data we were able to build thunderhill on unity. However eventually when exporting thunderhill to an asset bundle we ran into some major issues. This led us to eventually use Borregas Ave. Track from LGSVL's pre-made tracks instead. All necessary data was given to Autoware through a shared directory that the docker container for Autoware mounted to. This included all necessary launch files.

6 Autoware Setup/Requirements

We found that in order to be able to even run Autoware on a computer, there must be certain requirements that are met.

- CPU: An i5 is needed at the bare minimum but an i7 is recommended
- Memory: 16GB to 32GB
- Graphics Card: NVIDIA GTX GeForce GPU (980M or higher performance)
- SSD: At least 30 GB
- Alternative for GPU: Use an Nvidia Drive

We installed and ran Rviz which is the visualization for Autoware through build commands in Ubuntu. Using some sample data we created a visualization of the sample track shown here.

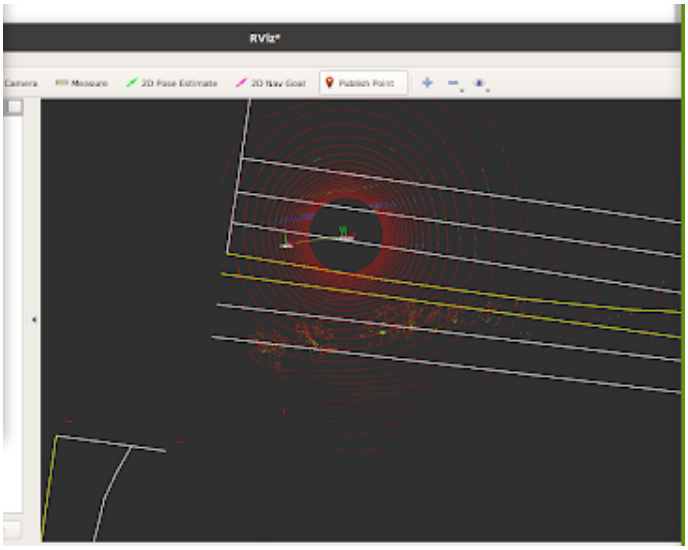

Figure 1: Rviz

7 ThunderHill Track

As we mentioned earlier one of our tasks for this project was to recreate the Thunderhill track in Unity-3d engine. In addition to using satellite imagery to generate track data we also were able to find a DEM(Digital Elevation Model) from Thunderhill track which shows the elevation data from the track which was used to help create the track

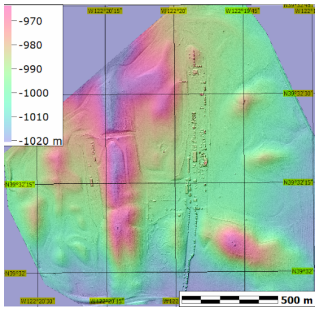

Figure 2: Digital Elevation Model

(a) Real Thunderhill track (b) Recreated Thunderhill Track

Figure 3

The portion of the track that we are more focused on is the west side of Thunderhill which is a 2 mile section of the track and can be seen in the picture below. We are just focusing on the west side of the track for the simulator because the racing competition only race in the west side and we want to make sure that the robot knows how to autonomously navigate the west side of the track as best as possible

Figure 4: West Thunderhill

8 Robot

The robot that we used were inspired from the ones they use in F1 car races but in smaller dimensions. The robot that we are using is a smaller version of the dimensions of the F1 cars that they use in car races. Some of the main hardware that is mounted on the robot consists of a LIDAR sensor, IMU, odometry, RGB-D cameras, power board, Nvidia Jetson Tx2 computer(CPU/GPU) along with other hardware that make up the robot.

(a) Robot

(b) Hardware on robot

Figure 5

9 Simulation

We selected to use the LGSVL Simulator to pair with autoware. Even though we eventually didn't get to use an asset bundle of thunderhill, we still had to build the simulator from source in order to prep for the thunderhill asset bundle. There were many package requirements for building the simulator. This include git LFS, node.js, unity version 2019.3.15. The procedure for building the simulator was to install unity hub, install the unity version 2019.3.15 with windows support dependency if you're on linux. Then clone the 2020.06 version of the simulator from the lgsvl github: LGSVL simulator 2020.06 version. This is extremely important as the wrong version will get you an asset bundle out of date error. Open the simulator as a project in unity and build the WEBUI. Then you are good to build the simulator. Open the simulator after its done building and in a web browser link the information for the vehicles and map, create a simulation and select the vehicle and map. Then run the simulation and it should pop up on the LGSVL simulator program.

10 Autoware and LGSVL

Autoware and LGSVL connects through a port. We provided a bridge connection for the 2 through LGSVL's bridge connection where we inputted the local ip of the machine at port 9090. Then we connected them through activating the launch file for sensing on Autoware. This allows for the simulator to be connected to Autoware. Now any actions in the simulator will be reflected on Autoware and vice versa. In order to verify the two are connected, you must see that the bridge status on the simulator shows as connected. With this we began 2d pose estimating as well as 2d navigation. After launching rviz, we used 2D pose estimate in order to calibrate the starting position of the car. This was done by drawing an arrow from the current spot of the vehicle through where it was facing. Then after turning on mission and motion planning in Autoware, we were able to set a 2d navigation goal for the vehicle to head to. An interesting observation we found was that the arrow needed to be in lane with the vehicle currently. Also due the the localization the car's appearance on autoware was very vague causing it to spin around and miss-localize.

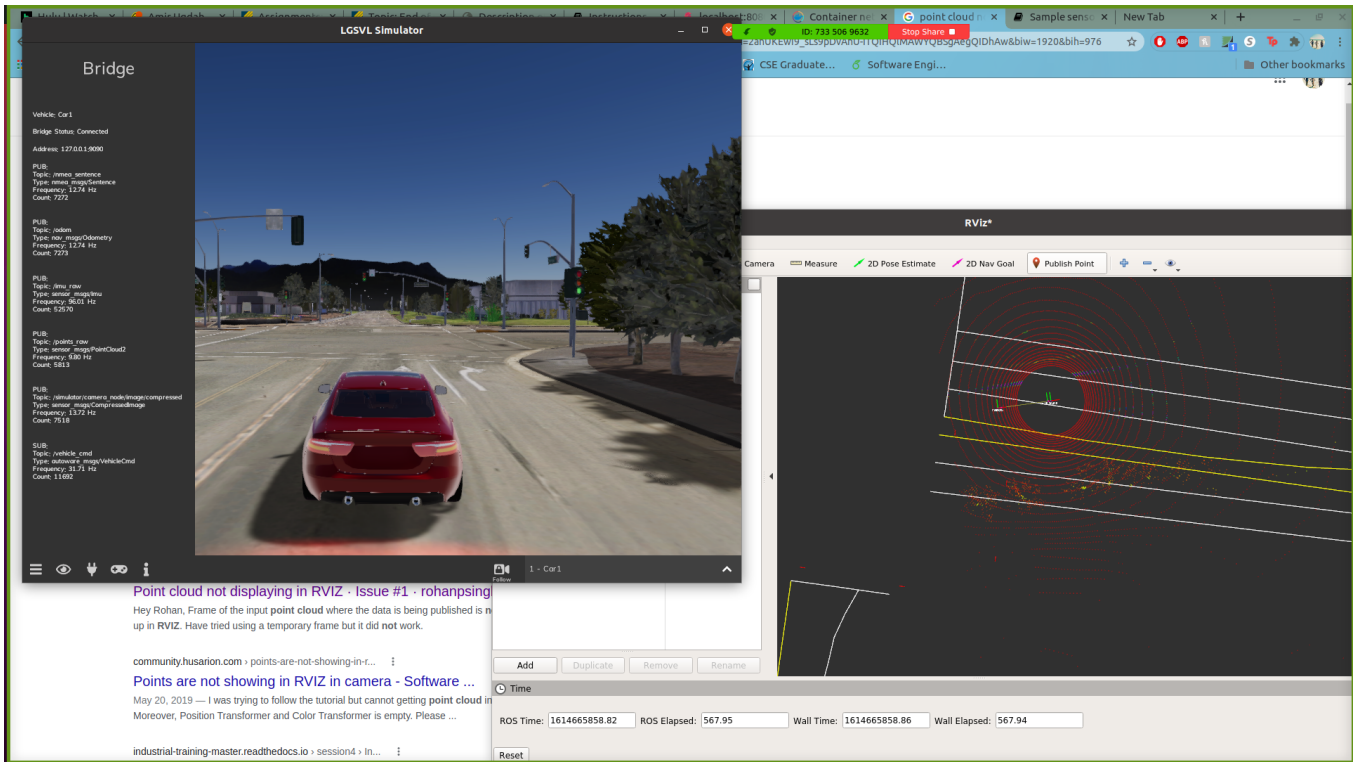

Figure 6: Connection of Autoware and LGSVL

11 ROS Stack Algorithms Used

Navigation Stack

11.1 Mapping

- already imported using Autoware however if we are making a fully autonomous vehicle we need to have a map; we can do this by utilizing g-mapping to create a 2-d occupancy grid map; from laser and pose data from the Lidar/RGB-D camera

11.2 Localization

- The navigation stack utilizes AMCL or Monte Carlo Localization - AMCL utilizes particle filters in order to find out where the robot is currently located This predicts position and orientation as it moves and senses the environment (localization)

11.3 Path Planning

- Handled by the move_base package in ROS - Utilizes Dijkstra's algorithm in order to determine the shortest path from current position to destination

11.4 Obstacle Avoidance

- Utilizes Lidar/camera data and 2D - occupancy grid map in order to determine where the obstacles are - If there is an Obstacle then we will create a new path using the move-base package

11.5 Autonomous Navigation

- Utilizes particle filtering, SLAM or Simultaneous localization and mapping Helps in using Odometry values and TF values to find a path for the robot to move

11.6 Localization utilizing GPS

- Utilize GPS API calls in order to get current location coordinates of the robot after obtaining these current location coordinates we can then publish this data to a Odometry node the navigation stack will subscribe to this Odometry node to obtain the current location of the robot it will then localize the robot on the map and then we can now plan a path (using move_base) for the robot to take

11.7 Autoware Navigation Stack

- Utilizing ROS2 (ROS2 Native Bridge), publishers and subscriber nodes, in order to publish data to nodes from surrounding and receive data (subscribe) to nodes for data. - Specifically, first the vehicle establishes localization by publishing the current Odometry, TF, etc data to the relevant nodes; and then navigation stack subscribes to these topics so it can utilize algorithms such as AMCL to establish localization - Because it establishes localization (the stack knows the current location) it can now create a path from its current position to its destination utilizing the 2-d occupancy map and Dijkstra's (shortest path algorithm). Afterwards, it would then publish this data so the robot can obtain the path/data - In order to do obstacle avoidance the navigation stack constantly subscribes to the topics necessary to detect obstacles in front of it so it can create a new path immediately in order to avoid these obstacles

12 Demo Video

The video linked here shows the vehicle traveling around a track autonomously:

Autonomous Navigation

13 Challenges/Debugging for Future Students

The challenges that we faced in the simulator and autoware setup included dealing with a lot of out of dated versions of packages and code from the github source code itself. This occured with both the simulator and autoware where we had to re-clone newer version of the data for autoware and the simulator builder code. This also occured with the autoware data launch files, ultimately the detection and localization launch files provided by autoware were outdated so we could not accurately localize and have it detect the movement around the track. This was an extremely big road block for us.
Other minor issues we faced had to deal with file conversions as well as the simulator using the wrong graphics card. In order to run the simulator properly, a Nvidia GPU must be used whether through a nvidia container or a local GPU. During the debugging process we found that we had to remove mesa drivers/the intel GPU so that the Nvidia one would be recognized.
A more major issue we ran into towards the end of the project was that the thunderhill track we created in unity was exporting into an asset bundle correctly. We have not resolve this issue and we have reached out to community forums about this issue. This is also the main reason we are using Borregas Ave. for our demo instead.
The out of date launch files was another unsolvable issue in our project. We attempted to change the computing settings of Autoware however this only make the localization glitch and did not improve it.

14 Conclusion

The remaining work that we have is to integrate Autoware using the rosbag file that we receive from converting the pcap file. Following that we are aiming to successfully get the simulator working so that we can visualize our vehicle move on the racetrack. Once we have all these previous steps completed then we will successfully have fully created a tool/pipeline for simulating our robot in a virtual environment for other teams to test out their code.We will have the Thunderhill race track generating data for us as the robot navigates through the virtual track.

15 Appendix

15.1 Project Proposal

Our team was previously working on the Lidar Sensor which is part of the car's obstacle avoidance/detection sensor. This sensor is in charge of 2D mapping its surroundings in a 25m radius of itself and reports back any obstacles that the sensor's laser scanner detects. We will no longer be working on the lidar sensor part of the robot since we already demonstrated how to integrate the LiDAR into ROS and will be shifting focus to Autoware virtual track simulator/testing.

A major part of this will be in the case that we can't test our robot on a similar track to the F110 and the ThunderHill track due to Covid limitations, we will have the Autoware simulator as a testing tool. A critical part of autonomous vehicles is to be able to visualize the robot in "action" and be able to test all the components of the robot such as the sensors, cameras, IMU, odometry , etc that will help navigate the car. Which is why for the second quarter of this project we will be focusing on creating our current robotics perception and actuators in the simulator based on real data.creating different virtual tracks using AutoWare , one of them being the ThunderHill track to simulate how well our robot would perform in the real track. Using the Autoware simulator will also enable us to test the other teams cameras and sensors that will give us a good idea of the effectiveness of the robots navigation and perception system. Due to Autoware being something new to all of us, our main goal for next quarter will be to learn the ins and out of Autoware AI simulator and become experts on it so that we can replicate the ThunderHill and F110 track environment and so that the other teams can test also test their algorithms and visualize their progress along with testing all of the parts of the robot.