Leveraging Simulation for Autonomous Vehicle Development

by

Mollie Marie Bianchi

Supervisor: Timothy D. Barfoot

April 2019

Abstract

Due to the complex nature of the challenges in Year 2 of the AutoDrive Competition and limited real world test facilities, a more advanced simulation environment is required. Development was undertaken in three separate simulation environments: CARLA, RightHook and MathWorks. CARLA was used primarily to collect photorealistic images of the competition signs and automatically generate corresponding labels. Since the RightHook simulation was created using real world HD map data of MCity, it proved useful in verifying the accuracy and completeness of the Zeus Map. After writing a ROS bridge between the RightHook simulation and the aUToronto system, tests were conducted to evaluate the robustness of aUToronto's planner. The MathWorks Simulation is a requirement for Year 2 of the competition. After a second ROS bridge was written and a map of the desired route in simulation was created, the existing aUToronto code was able to drive route in simulation. Point cloud information was used to determine the distances to static and dynamic obstacles. Each of these three simulation environments was able to add a unique resource to the overall autonomous vehicle development.

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Prof. Tim Barfoot for all his invaluable feedback and guidance. Prof. Barfoot, along with Prof. Angela Schoellig, also deserve thanks for being the faculty advisors to aUToronto.

Additionally, thanks are also owed to Jon Mullen of RightHook Inc. for not only sponsoring the MCity simulation environment but taking time to help with getting our system up and running. Jon has always responded quickly and with enthusiasm to accommodate any requests.

This research would not be possible without the many diligent members of aUToronto. Particular thanks go to Keenan Burnett for leading our team, to Joe Qian for assisting in planner integration, to Tracy Du for making modifications to the controller and debugging tf tree issues, and to Kaicheng Zhang for making many maps. I am also grateful to the team for making our meetings enjoyable.

Special thanks to SAE for creating, facilitating, and sponsoring the amazing experience that is the AutoDrive Challenge. This challenge would not be possible without the generous support of General Motors, MathWorks, Continental, Intel, OXTS, Velodyne Lidar, and all other sponsors.

Table of Contents

List of Figures

1 Introduction and Motivation

In response to the rising popularity of autonomous vehicles, the Society of Automotive Engineers (SAE) launched the AutoDrive Challenge in 2017. This is a 3 year challenge for 8 universities selected across North America to convert a standard Chevrolet Bolt into a vehicle capable of Level 4 autonomous driving. aUToronto is the University of Toronto's team competing in this competition and their vehicle has been named Zeus.

The first year of the competition, held in May 2018, was comprised of basic autonomous tasks such as straight and curved lane keeping, stopping at stop signs, and detecting and avoiding obstacles. aUToronto is currently preparing for the second year of the competition to be held in June 2019 at MCity. MCity is a test facility designed specifically for the testing of automated vehicles in urban and suburban environments. MCity contains a variety of 2, 3, and 4 lanes roads, various signs and traffic control devices, different road surfaces, trees, crosswalks, bike lanes, adjustable buildings facades, etc. See Figure 1.1 for a map of the testing ground.

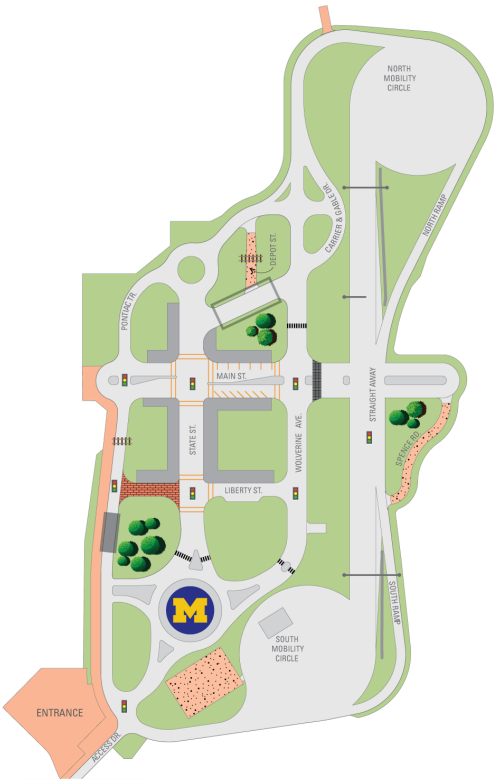

Figure 1.1: Map of MCity.

There are four dynamic challenges in Year 2, the Traffic Sign Challenge, Intersection Challenge, Pedestrian Challenge, and MCity Challenge. These challenges are designed to be more complex than those in Year 1. The Traffic Sign Challenge requires the vehicle to navigate based solely on traffic control signs, such as left turn only, right turn only, do not enter, and speed limit signs. For the remaining three challenges, teams are provided with GPS waypoints by which the vehicle is to navigate. The Intersection Challenge involves correctly handling a variety of traffic light signals at intersections. The Pedestrian Challenge will involve dynamic pedestrian "dummies" waiting to cross at designated crosswalks and intersections. The vehicle must stop if a pedestrian is present and wait for them to fully cross before proceeding. If after five seconds the pedestrian does not begin to cross, the vehicle must proceed. The MCity Challenge combines

the previous challenges and adds additional dynamic animals, GPS occluding tunnels, and other components. [1]

Due to the added complexity of these challenges, the team's previous simulation testing capabilities were not enough. A new simulation environment was needed this year for two main reasons. The first being limited access to real world testing facilities. aUToronto has access to a short stretch of private road at UTIAS (see Figure 1.2) and a single intersection at UTM, but not to a facility with multiple intersections like those present at the competition. Additionally, the ability to create certain scenarios with pedestrian models crossing at intersections and changing traffic signs is difficult and time consuming.

Figure 1.2: Aerial view of testing at UTIAS' private road.

For safety reasons, it is also desired to do an initial integration test in simulation before trying new code on aUToronto's vehicle, Zeus. The second reason for a new simulation environment is that the previously used simulation environment in Gazebo lacks high fidelity graphics, dynamic obstacles, and a MCity map. This research aims to leverage the capabilities of existing high fidelity simulation environments to aid in aUToronto's autonomous vehicle development.

2 Background on Existing Simulation Environments

2.1 Gazebo

In the first year of the competition, the simulation environment was developed entirely in Gazebo. Gazebo is a 3D dynamic simulator designed for testing robotic algorithms. It has a wide variety of sensors available for use, the ability to provide accurate physics models, and it uses Robot Operating System (ROS) to interface with the algorithms being evaluated.

The simulation environment was designed to mimic the test track in Yuma, Arizona where the first year competition was held. There were three test tracks representing each of the three competition challenges, straight line, lateral, and obstacle detection. These challenges took place on flat asphalt in the desert. Figure 2.1 shows these simulated challenges. The vehicle itself used the readily available physics of a Toyota Prius with the shell of a Chevrolet Bolt. There were few 3D assets in the simulations aside from the vehicle, primarily stop signs, stationary pedestrians, and other parked cars. The simulation was configured with the same sensors as on Zeus, i.e. a monocular camera, a stereo camera, and a 64 beam Lidar. Controls to the vehicle were supplied as normalized values for steering angle, throttle, and brake.

Figure 2.1: Images from the Gazebo simulation environment.

This environment was useful in testing the integration of the autonomy nodes; however, the feedback was primarily qualitative from watching the visual performance of the car. For example, an aerial view helped to tell when the vehicle was outside of the lanes in the lateral challenge. Data from running tests in Gazebo was never utilized to a major extent.

The environment was created by placing a Google satellite image onto the ground and adding lane lines on top. This did not create a very realistic environment. Additionally, the physics model the vehicle was based on was not accurate. It was relatively easy to flip the car in simulation.

Gazebo is intended to be used in a variety of robotic applications. As a result it has a lot of additional features that are not needed for autonomous vehicle development and a lot of work is required to customize the simulation to fit the specific needs for aUToronto. There are many other simulation environments which are specifically targeted to self driving cars as detailed below.

2.2 CARLA

Carla is an open-source, urban driving simulator rendered using Unreal Engine 4 to provide a photorealistic environment. It includes two separate neighbourhoods whose roads are laid out in a grid (see Figure 2.2). There are exclusively two lane streets with one lane of traffic in each direction. The intersections are simplistic with no turning lanes. The traffic lights are always three vertical lights on a pole on the right side of the road. 3D models of static objects such as buildings, vegetation, traffic signs, and infrastructure are distributed throughout the environment. Since it is implemented as an open-source layer over Unreal Engine 4, it is easy to add, remove, or modify existing 3D assists and self-created assets.

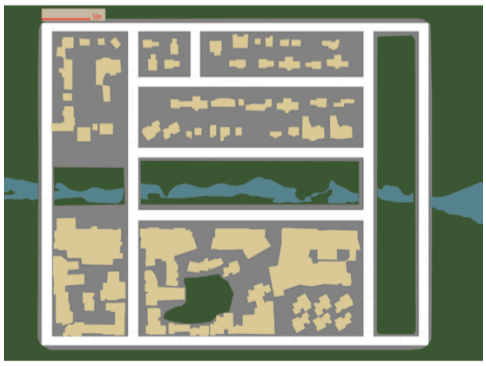

Figure 2.2: Map of one of the CARLA neighbourhoods.

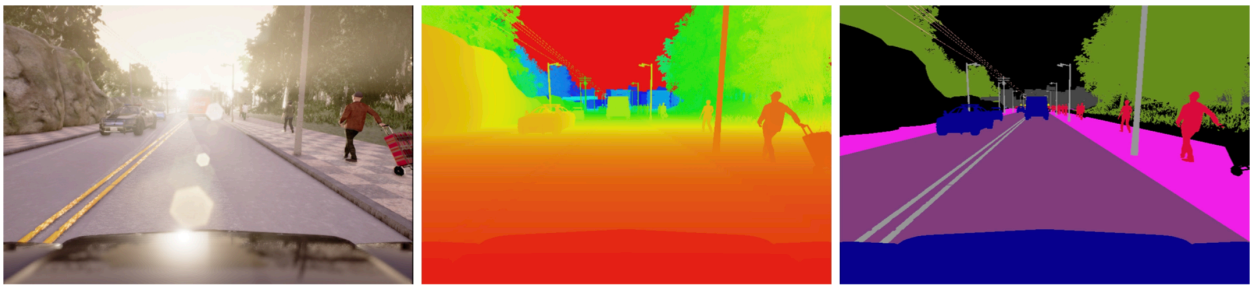

Figure 2.3: A scene in CARLA captured by a RGB camera, depth camera, and semantic segmentation camera.

Pedestrians and vehicles are present and act according to a predefined map of allowable states; however, it is difficult to configure their actions to specific scenarios. There are 18 possible illumination-weather conditions that differ in the position and colour of the sun, the intensity and colour of diffuse sky radiation, as well as ambient occlusion, atmospheric fog, cloudiness, and precipitation. The configurable sensor suite includes RGB cameras, depth cameras, semantic segmentation cameras, and a rotating lidar (see Figure 2.3). Compatibility with ROS was added in June 2018. To control the vehicle, CARLA expects normalized values for throttle, brake, and steering. [2]

2.3 RightHook

RightHook is a commercially developed, physics-based simulation environment created from real world HD map data. Significant care has been taken to ensure that the simulation is as close as possible to the real world environment. Digital artists created a variety of 3D assets with realistic colours, shapes, and reflectivity. RightHook provided aUToronto with a sponsored MCity simulation. The MCity simulation is an exact replica built from HD map data provided by Carmera. It has the correct lane markings, number of lanes, traffic lights, buildings, and infrastructure.

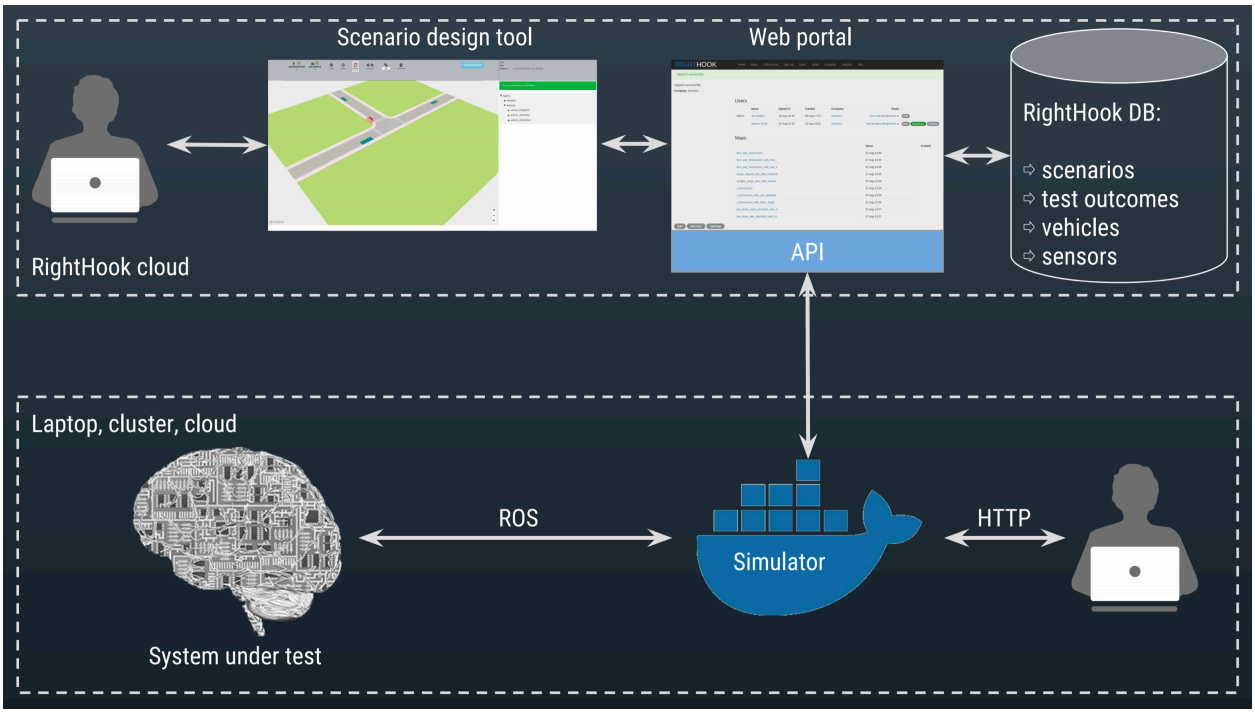

Figure 2.4: The RightHook Pipeline

The scenario design tool is accessible through RightHook's web portal. Through this tool one can select the map for a new scenario. There are a variety of mini-maps available in addition to the MCity map. A variety of vehicles, such as hatchback, sedan, pickup, etc., can be added and positioned. The hatchback is modelled after a Nissan Leaf which is a comparable size to the Chevrolet Bolt. Each vehicle will follow drive lines embedded in the map, unless it is specified as an ego vehicle. The possible sensor suite includes camera, infrared, lidar, radar, GPS, IMU, and ultrasonic. The GPS output matches the exact coordinates at MCity. These sensor can be configured for each vehicle through a JSON file uploaded to the design tool. There are adult and child sized pedestrians that can be added with a designated walking speed. The time of day is also specified which affects the lighting conditions

Figure 2.4 shows the RightHook pipeline. After scenarios have been created on the web portal, a HTTP request will launch the scenario on a docker container installed on your personal machine. Outputs from the simulation are published to ROS topics. Each vehicle designated as an ego vehicle has a `/vehicle_name/actuation/cmd` topic which takes a throttle and brake value from 0 to 1 and a wheel angle in degrees. Advancing the simulation by a time step requires a request sent to the SimControl service running inside the container's ROS system [3].

2.4 MathWorks Simulation Challenge

An additional component of the Year 2 competition is the static event: the Mathworks Simulation Challenge. This challenge is worth 100 points out of the total 1000. The objective is to have the ego vehicle drive clockwise around a simulated MCity block in the fastest time possible (see Figure 2.5). There will be static, parked vehicles along Main St., moving vehicles on State St. which need to be passed, and traffic in the opposing lane on Wolverine Ave. The speed limit for the ego vehicle is 8 m/s. The minimum lateral

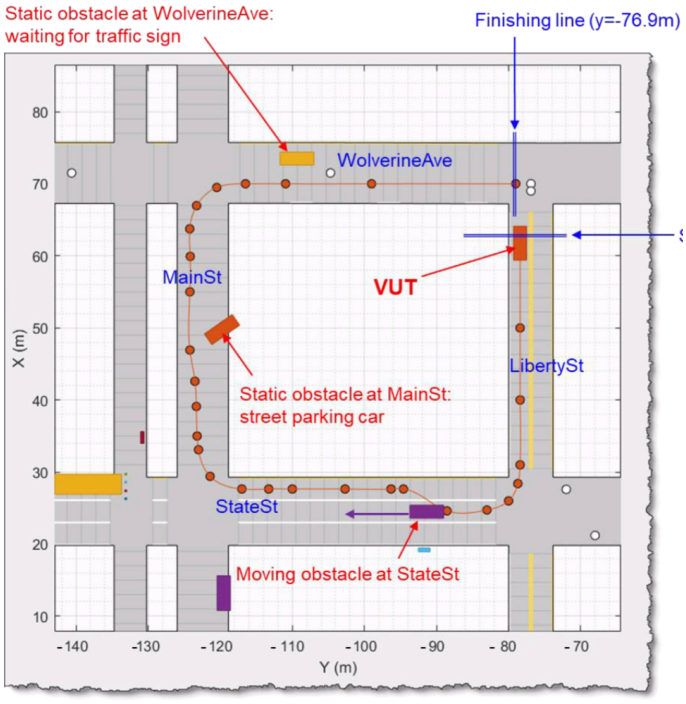

Figure 2.5: Map of intended route for the Simulation Challenge.

distance to obstacles is 0.5m and the minimum longitudinal distance to obstacles is 2.0m. The magnitude of the longitudinal acceleration must be less than 1.5 m/s^2. Points will be deducted for violating longitudinal acceleration limits, leaving the road boundary, or driving too close to obstacles. [4]

The simulation is launched through a provided Simulink model which must be run in Matlab 2018b on a Windows 10 computer. Sensors on the test vehicle can be configured by adding and modifying blocks in the Simulink model (see Figure 2.6). The sensors are limited to 2 cameras, 1 Lidar, and 4 radars. There is a custom message type to control the vehicle containing normalized values for throttle, brake, and steering. The dynamic obstacles are implemented as a Simulink block whose internals are hidden. [4]

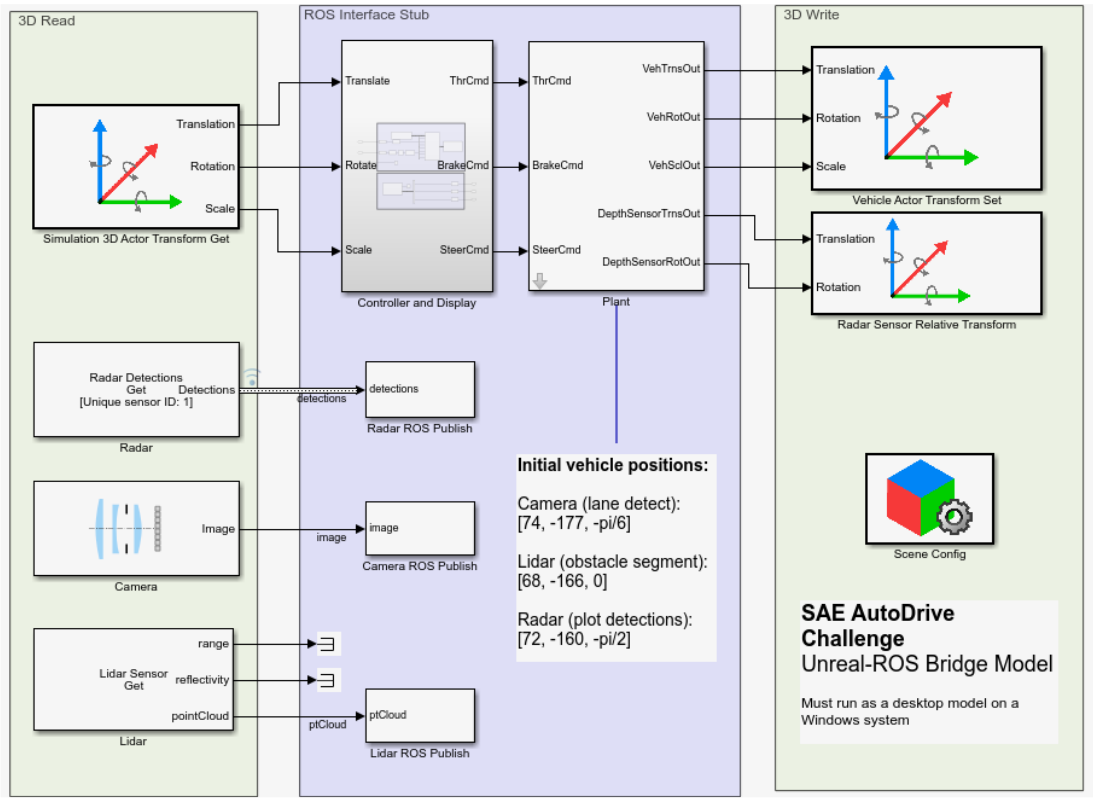

Figure 2.6: Simulink Model of the Simulation Challenge

3. Approach Taken

3.1 CARLA

The following sections detail the work conducted in the CARLA environment. The proposal was initially to test the competition challenges in CARLA, but due to the limited ability to customize scenarios CARLA was primarily used for image collection.

3.1.1 3D Modelling of Signs

There are nine different traffic signs that will be used in the competition: “Do Not Enter”, “Left Turn Only”, “Right Turn Only”, “Speed Limit” (5, 10, 15, 20, and 25 MPH), “Parking”, and “Disabled Parking”. The signs originally included in CARLA were very different from the competition signs. The CARLA environment is able to be modified through the Unreal Engine Editor. This software allows 3D models to be imported. So Maya, the 3D modelling software, was used to create models of all the competition signs. The models were created with the correct dimensions and an image of the specific sign was applied as a mesh. Once these models were created they were placed at random throughout the standard CARLA environment.

3.1.2 Semi-Automatic Sign Label Generator

CARLA has a built in autopilot function that drives the ego car around the neighbourhood. By running the simulation with this autopilot for 8 hours, approximately 20,000 RGB and semantic images were collected in a variety of weather conditions. Originally there were twelve tags available in the semantic image for object classes such as: building, fence, pedestrian, pole, road, road line, etc. The semantic images are created by tagging every object in the scene when it renders based on the folder in which each mesh is stored. So for each additional competition sign added, it needed to be placed in its own correctly labelled folder. The name of these folders was added to the CARLA source code and a new corresponding label was added to the `ECityObjectLabel` enum in the Tagger.h file and to the file path check in the `GetLabelByFolderName()` function in Tagger.cpp file [5].

An example of a collected semantic image can be see in Figure 3.1. Image A shows the RGB image of the scene which includes a 20 mph speed limit sign. 20 mph speed limit signs have been given the tag #19. Image B is the corresponding semantic image before processing.

Since the image tags are only encoded in the red channel and they have low values (i.e. < 21) it is difficult to see anything in this image. Image C is Image B after histogram equalization and is purely for visualization. Image D is the semantic image where any pixels whose red channel value equals 19 has been set to 255 and all other pixels have been set to zero. Iterating over all the contours found by OpenCV's `findContours()` function and running `minAreaRect()` around each contour, a minimum bounding box was found for each sign [6].

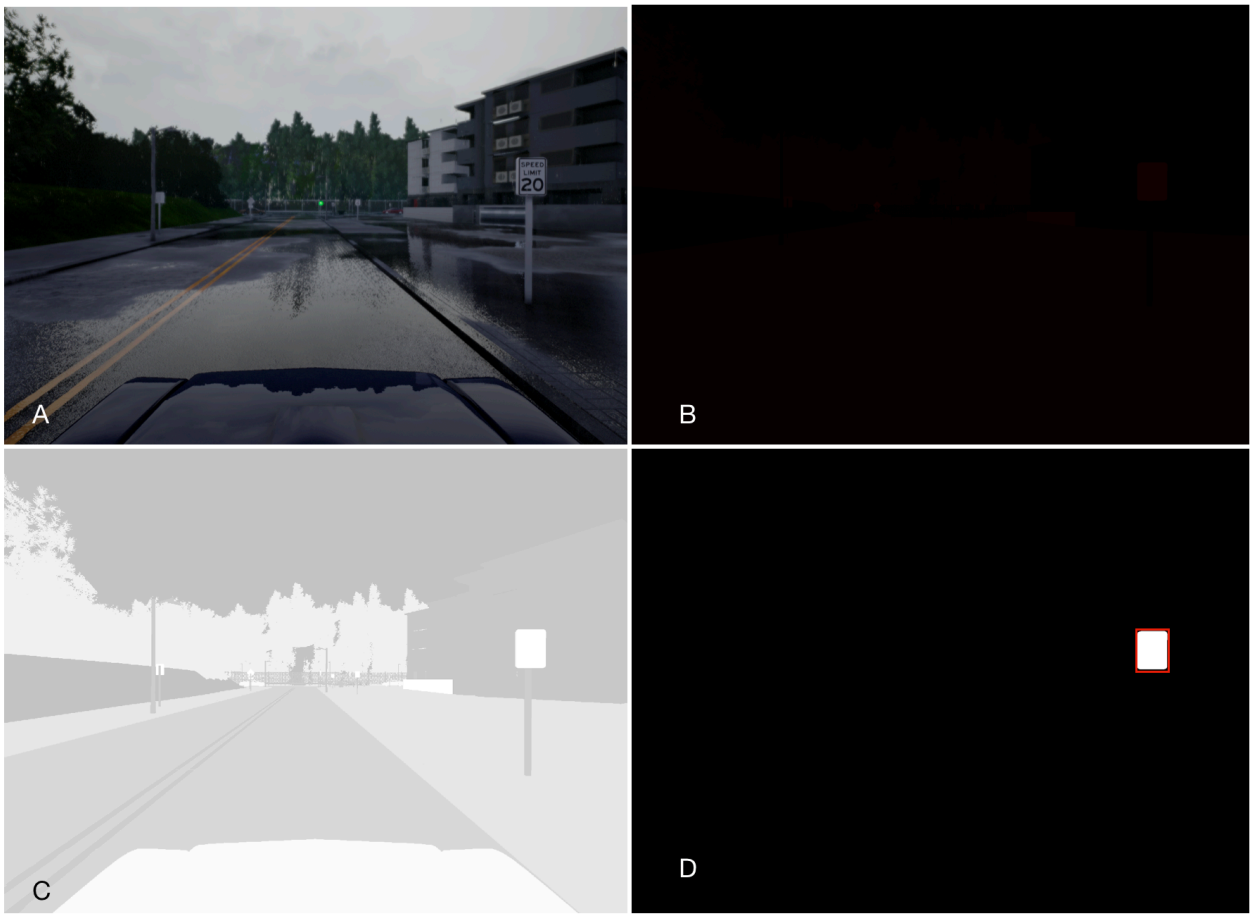

Figure 3.1: Using semantic images to isolate sign locations.

A weakness of the semantic image is that the direction the sign is facing is not taken into account. So if there is a sign on the other side of the street facing backwards to the camera it will be included in the thresholding step. To solve this, the left half of the image was ignored since all signs on the left side of the street were placed so that they will be facing backwards. Additionally, if there is an object bisecting the sign, such as a pole or a pedestrian, two bounding boxes would be created for a single sign. Images where two bounding boxes were generated

within a certain distance of each other were flagged and reviewed manually afterwards. A bounding box enclosing both sections of the sign would be annotated by hand in this case. In this way, the labelling process was semiautomatic.

After this labeled data set from simulation was completed, it was shared with the Lights and Signs sub team. They used the data to supplement the real world data set to train their classifiers. The inspiration for this method came from the paper "Playing for Data: Ground Truth from Computer Games" [7] where the authors were able to create semantic images from Grand Theft Auto images. The authors were motivated largely by the fact that labelling datasets by hand is very time consuming or expensive.

3.1.3 Map Customization

Some initial work was done to create a custom MCity map using the Unreal Editor. This involved placing a satellite image of MCity into the editor and then adding pavement and infrastructure on top. This approach was eventually abandoned due to its lack of accuracy and significant time consumption in favour of the RightHook simulation.

3.2 RightHook

Since the RightHook MCity simulation was created using real HD map data, the GPS output corresponds with the real GPS coordinates of MCity. This combined with the realistic physics modelling of the vehicles makes the RightHook simulation a great way to verify aUToronto's created MCity map and planner.

3.2.1 ROS Bridge

In order to use the RightHook simulator, an adaptor needed to be written to take output from the simulation and publish it the ROS topics expected by the aUToronto system; and take the controller command and convert it to the format of the actuation command expected by RightHook. A general outline of how this adapter works is shown in Figure 3.2. The green topics are outputs from the simulation. The `/Zeus/state/nav_sat_fix` topic is first republished with the name `/navsat/fix`. Then the latitude and longitude from this topic are converted to UTM coordinates and placed in the position field of an odometry message. The velocity of the vehicle in the x direction comes from the `/Zeus/state/state` topic and is also added to the odometry

message. The x velocity is also published directly to `/VehicleInterface/velocity`. The orientation of the vehicle comes from the `/Zeus/state/bb` topic. This orientation is also added to the odometry message before it is published to `/navsat/odom`.

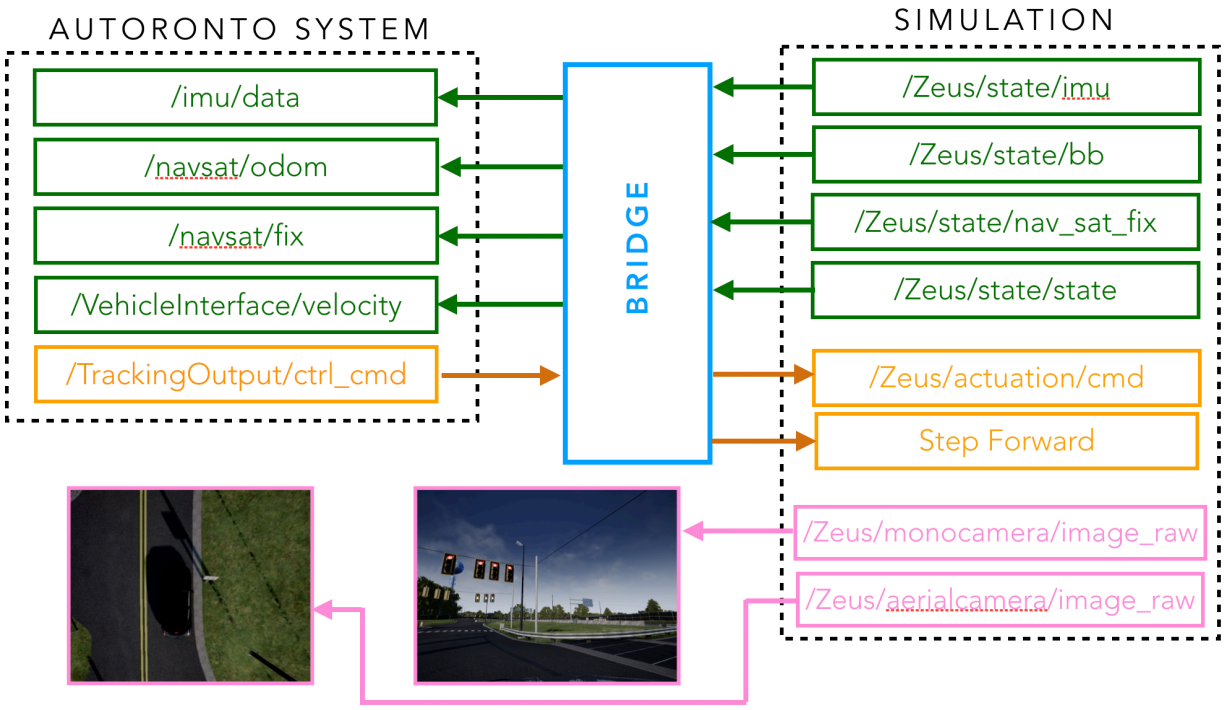

Figure 3.2: ROS Bridge between aUToronto System and RightHook Simulation

aUToronto has defined the tf tree for the various frames on the vehicle (e.g. base_link, front_link, chasis_link) from the imu_link frame as this is where the actual GPS unit on Zeus is installed. The RightHook GPS is positioned at the base_link of the vehicle, so the tf tree needed to be partially inverted so that all frames are defined from the base_link (see Figure 13). The base_link itself is broadcast immediately after the `/navsat/odom` topic is published.

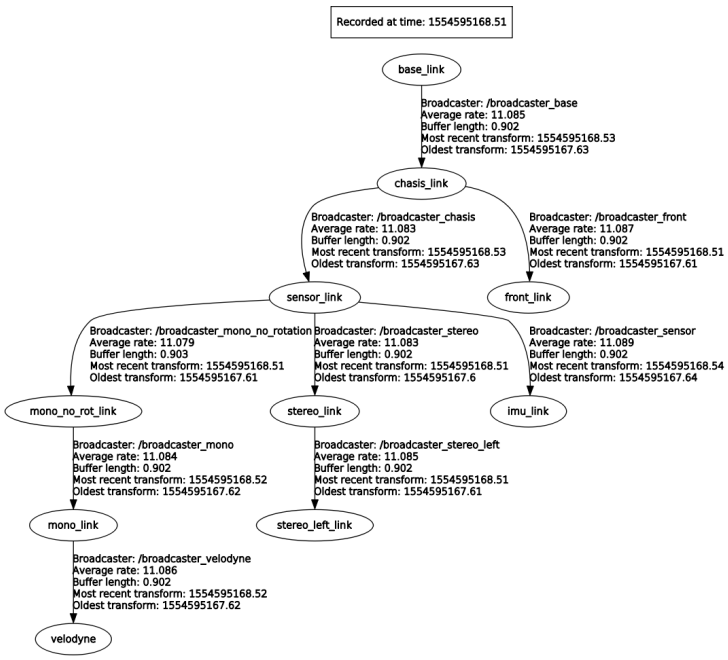

Figure 3.3: tf tree defined from base_link instead of imu_link

The RightHook simulation requires a

request to the SimControl service running inside the container's ROS system in order to advance by a time step. All vehicles specified as ego vehicles must receive an actuation command otherwise the simulation will not step forward. When a message is published to `TrackingOutput/ctrl_cmd`, the incoming torque value is normalized by a max torque currently set to 2000. If the torque value is negative, the normalized torque value is set to zero and the normalized brake value is set to 1. The incoming steering angle is converted from radians to degrees. The normalized torque and brake, and the new steering value are saved to a global variable to latch their values. There is a while loop that runs constantly sending requests to step the simulation while publishing the latched control values in a custom RightHook message to the topic `/Zeus/actuation/cmd`.

3.2.2 Preliminary Intersection Test

Creating the full map of MCity was a lengthy process and relied on the receipt of the HD map data. In order to conduct initial testing before the entire map was completed, a map of a single intersection was created. This intersection is shown in the first image in Figure 3.4. RightHook has the ability to allow vehicles to drive on the centre lines of the map automatically. By creating six scenarios with the vehicle in a different starting point each time, all six of the possible paths through the intersection were able to be driven using the autopilot feature. While

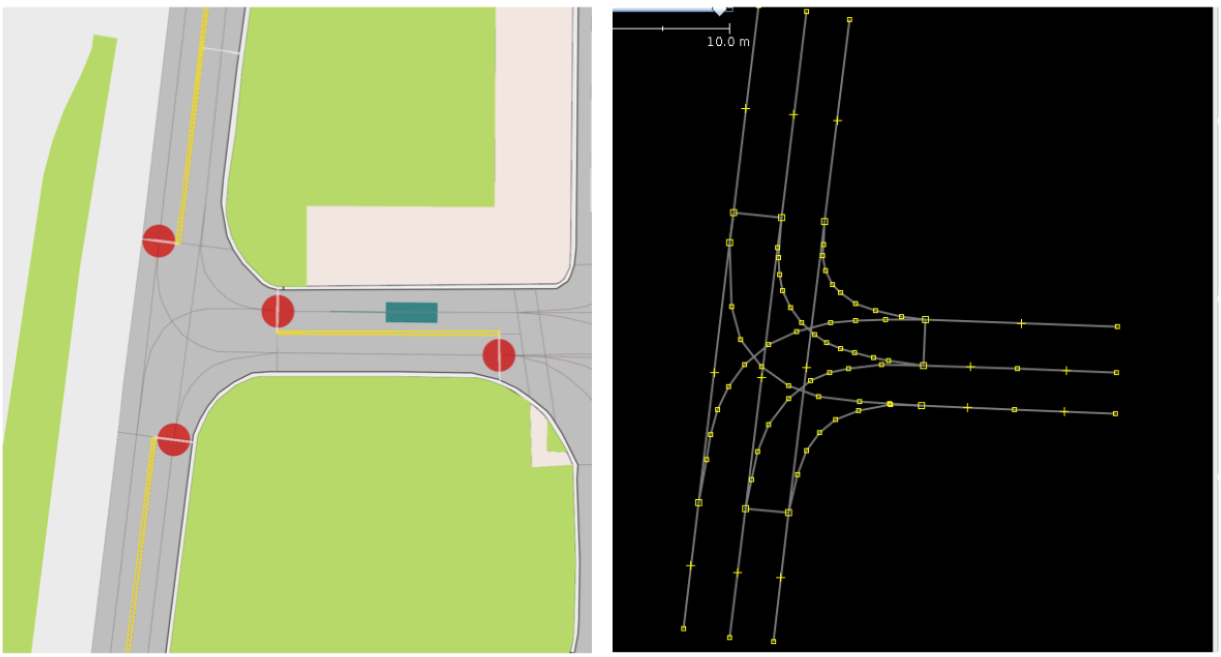

Figure 3.4: The simple intersection in RightHook and corresponding map

the paths were being driven, the GPS coordinates were collected. The mapping sub team was then able to create a map for this intersection, shown in the second image of Figure 3.4.

Since RightHook does not have the ability to customize or add traffic signs and it was desired to focus testing solely on the planner, the traffic sign perception node was spoofed. The planning sub team wrote a script that when it received keyboard input, i.e., a certain number, a message would be sent to the planner indicating that a Left Turn Only or Right Turn Only sign had been seen. By default, the planner chooses to go straight. In this way, all six different passages through the intersection were tested successfully. Slight issues in the planner, such as jagged paths, were observed and related to the planning sub team where they were rectified.

3.2.3 Zeus Map Verification

Once the map for competition, referred to here after as the Zeus Map, was completed, testing in RightHook was able to be carried out over the entire MCity environment. Figure 3.5 shows the Zeus Map visualized in rviz after it had been successfully loaded into the planner. The primary objective of this round of testing was to verify the completeness of the map, ensuring that there are no missing connections or mislabeled segments. This was accomplished similarly to the previously described method. The car was initialized in a given location and by default would drive straight. By publishing Left/Right Turn Only signs the vehicle was forced to drive throughout the map.

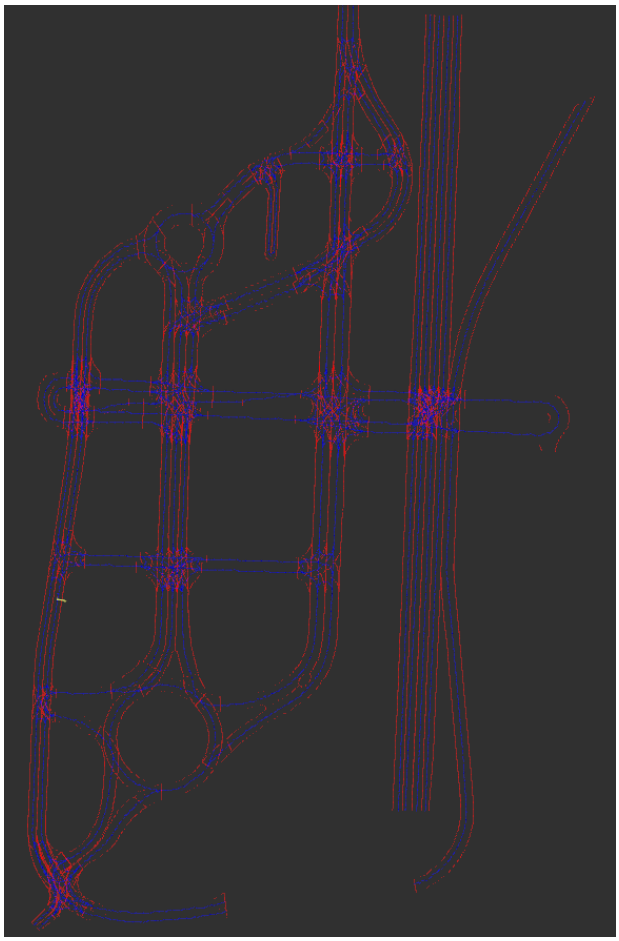

Figure 3.5: Zeus Map after being successfully loaded and visualized in rviz.

There are certain limitations to this strategy. The first being that many of the intersections are more complicated than the intersection in the preliminary test and have significantly more than six possible through routes. The other issue is that there are some lanes with multiple left turn possibilities.

That is, there are two different lanes into which the car can turn left from its current lane. Currently, the planner will choose at random which lane to turn into. The planning sub-team is currently working on modifying this to be more consistent.

Figure 3.6: Possible waypoints at MCity.

At the time of writing, a new testing strategy is being implemented. SAE has provided a set of all possible waypoints that will be used in the competition (see Figure 3.6). Many different paths and corresponding waypoint sequences will be created. These waypoint sequences can then be passed to the planner. This method is closer to what will occur in three of the four dynamic challenges when traffic signs are not the primary means of navigation. Ensuring that the route the planner decides on covers all of the waypoints in an optimal manner will be important before going to the competition.

3.2.4 Camera Output Usage

The RightHook simulation is extremely accurate, not only in GPS, but also in camera and Lidar output. Figure 3.7 shows a real world image of MCity on the left and the corresponding image from RightHook on the right. They are virtually identical. Of particular note is the identical placement and configuration of the traffic lights. aUToronto's traffic sign classifier is trained off of real world datasets and images collected of the team's own traffic lights. MCity has many unique traffic light configurations that are hard to find in datasets. The classifier will be tested on images collected from RightHook to verify that it can handle the exact configurations and placements of the lights at MCity.

Figure 3.7: A picture taken at the same location in the actual MCity and in RightHook's MCity.

3.3 MathWorks Simulation Challenge

The MathWorks Simulation Challenge is a requirement of the Year 2 Competition. The ego vehicle must be capable of driving around a simulated MCity block while correctly handling any static or dynamic obstacles it encounters. The following sections detail the work conducted in order to fulfill these objectives.

3.3.1 Network Setup

The MathWorks Simulation will only run on a Windows 10 computer. The code that runs the aUToronto system only runs on Ubuntu 16 with ROS Kinetic. A virtual machine running Ubuntu 16 was installed onto the native Windows computer. In order to run ROS across multiple machines the `ROS_MASTER_URI` in Matlab and in the virtual machine was set to the virtual machine's IP address. The `ROS_IP` variable in the virtual machine was set to the virtual machine's IP address while the `ROS_IP` variable in Matlab was set to the windows IP address.[8][9]

3.3.2 Map Generation

The first step in developing for the Mathworks Challenge was to create a map of the block. The challenge only requires the ego vehicle to travel around a single block in the simulated MCity. Which block is also known before hand. By manually publishing ROS

messages in the Matlab Command Window, the vehicle was able to be driven around this block in order to collect the position coordinates. These coordinates are in a local frame. MathWorks defines the global frame with the x axis pointing east, the y axis pointing south, and the z axis pointing down into the ground. The convention used by the mapping sub team has the y axis pointing north and the z axis pointing up. To correct for this, the y values of the collected position coordinates were multiplied by -1. This corresponds to 180º rotation about the x axis. The mapping sub team was then able to create the map shown in Figure 3.*8*. The original scope of the challenge did not require any lane changes so only a single lane was included in the map. Lane changes have recently been added to the challenge so further additions to the map are necessary. See Section 6.2: Future Work - Mathworks Simulation Challenge for further details.

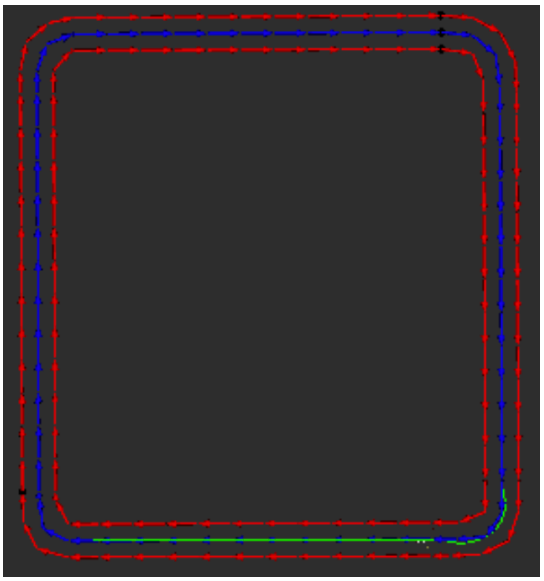

Figure 3.8: Map created for the relevant MCity block in the MathWorks Simulation Challenge.

3.3.3 ROS Bridge

Similar to RightHook, a ROS bridge to allow proper communication between the simulation and aUToronto's system needed to be written. Figure 3.9 shows the general form of the bridge. The simulation publishes to the vehicle pose to the topics `/pose` and the vehicle's linear velocity to the topic `/velocity`. After transforming the pose to account for the 180º rotation about the x axis to the map coordinates, the base_link frame is broadcast and an odometry message with this pose which is published to `/navsat/odom`. The velocity message is republished to `/VehicleInterface/velocity`. Since the simulation does not include acceleration data, the previous velocity value is saved and then subtracted from the new velocity value. This is then divided by the change in time between the two message to give an acceleration value which is published to `/imu/data`. The controller inputs are repackaged in a custom ROS message created by MathWorks before being published directly to `/vehicle_cmd`.

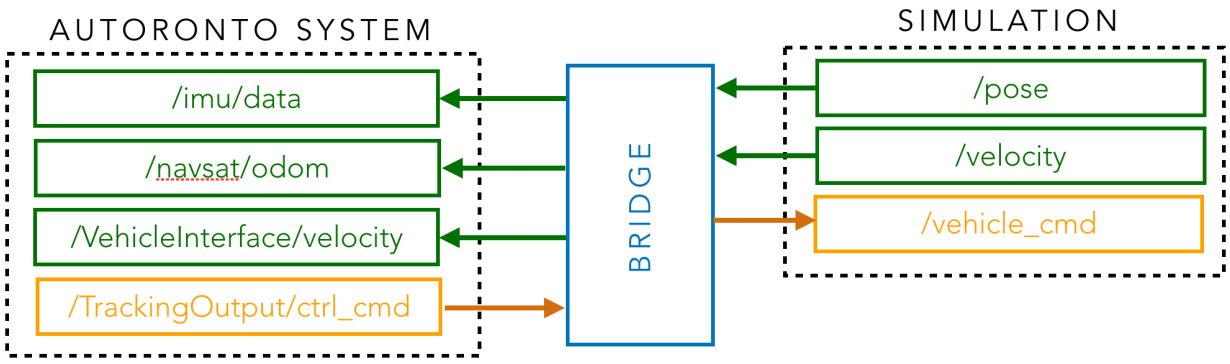

Figure 3.9: ROS Bridge between aUToronto System and Mathworks Simulation

3.3.4 Obstacle Detection

The only obstacles in the challenge will be vehicles directly in front of the ego car. So the obstacle detection method has been created specifically with this in mind. The simulation includes three types of sensors: camera, radar, and lidar. The initial idea was to use radar placed at the front bumper of the vehicle; however, it soon became clear that the simulated radar measurements were extremely unreliable. As a result, it was decided to use lidar. The lidar sensor returns a 360º field of view point cloud around the vehicle. The point cloud was filtered to remove any points greater than 10º from the centre line of the ego vehicle.

Figure 3.10: A scene from the MathWorks Simulation. An obstacle car is positioned in from of the red ego vehicle .

Figure 3.10 shows the ego car (red) placed in front of an obstacle car (black). Figure 3.11 shows the corresponding point cloud after it has been filtered to the narrower field of view. The next step was to remove any points whose z component was lower than -1.5m. The lidar sensor was mounted at 1.5m on the car and since the simulation has a perfectly flat ground plane, removing these

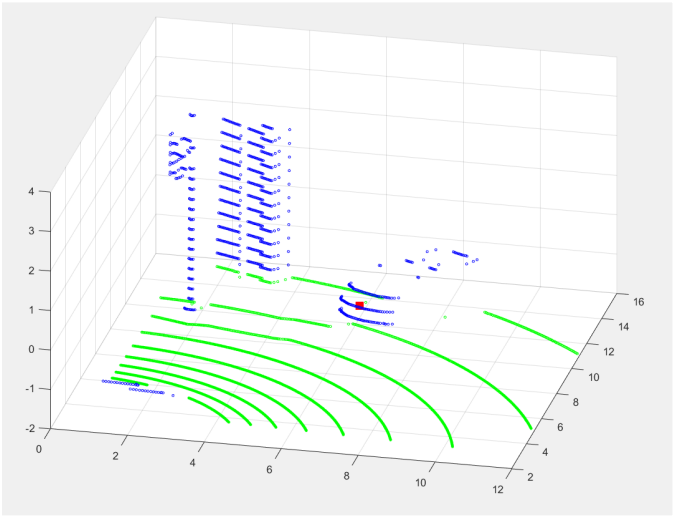

Figure 3.11: The point cloud corresponding to the above scene.

points removes the ground plane. This is seen in Figure 3.11 as all the green points. The remaining points were passed into the DBSCAN function in the scikit-learn library [10]. This function assigns the points to clusters. Clusters with fewer than 50 points were eliminated as any true obstacle should have greater than 50 points. As well clusters whose centroids were greater than 0.5m from the car's centre line were removed. The distance to the closest remaining cluster was then taken as the distance to the obstacle. This distance was then published for use by the planner.

4. Results

Three simulation environments have been successfully utilized for the aid in development of aUToronto's autonomous vehicle, Zeus. CARLA was ultimately deemed less suitable for testing aUToronto's code, and instead was used primarily for image collection. Interfaces have been written for both the RightHook and MathWorks environment that allow aUToronto's code to be run smoothly within the simulations. This research has provided a base from which further testing can be conducted throughout the remaining years in the challenge.

5. Discussion and Lessons Learned

5.1 CARLA

The major downside of CARLA was its limited ability to configure new map layouts. The intersection present only involved two lanes. There were no designated turning lanes. A large part of the planner testing is focused on ensuring it is able to handle the complicated intersection scenarios present at MCity. Since these complicated intersections could not be created easily in CARLA it was rejected in favour of RightHook.

The need for automatically labelled sign data sets from CARLA was also reduced once aUToronto decided to have data commercially labeled by Scale.

5.2 RightHook

A significant limitation of the RightHook simulation is the speed at which it runs. Due to the high quality graphics it takes a noticeable time to render each scene. A representative from

RightHook indicated that this rendering time is approximately linear with the number of sensors. Since RightHook is being used primarily for testing the planner, the other sensors, such as cameras, lidar, and radar, are not necessary. By removing all sensors, except an aerial camera which was used for visualization, the run time speed was able to be increased.

Ideally, the full stack could be tested in RightHook. It is unclear at this point whether this would be effective. Adding the other sensors may make the simulation run too slow. Additionally, there is some concern about how the classifiers would perform on simulated data as they were trained primarily on real world images. The testing of the traffic light classifier on RightHook images has not yet occurred. Although not expected, it should be noted that poor performance may not be due to problems with the classifier itself, and could be due primarily to its inability to generalize to simulated images.

5.3 MathWorks

The quality of the surroundings in the MathWorks environment is significantly lower compared to RightHook. There is less infrastructure and less care taken to match what is at MCity. The graphics quality is less photorealistic. As a result, once the simulation initializes it is able to run faster than the RightHook simulation. This is useful for testing with dynamic obstacles. The biggest limitation for testing in MathWorks comes from the position coordinates being published in a local frame. As a result of this, aUToronto is not able to test using the Zeus Map. If more extensive testing was desired outside of the single block for which the map has already been created, it would need to be mapped first. This is somewhat redundant when the Zeus Map already exists. A mapping from local coordinates to real world coordinates is another potential solution; however, based on observations it is likely that the MathWorks simulation is not an exact replica of the real MCity. So an exact mapping from local coordinates to global coordinates may not exist. Feedback to MathWorks would be to include a GPS sensor that outputs real latitude and longitude values.

6 Future Work

6.1 RightHook

Now that preliminary testing and the Zeus Map have been completed, the future work in RightHook will be running as many evaluations as possible before the competition. It is relatively quick to setup a new scenario in RightHook as compared to a test in the real world. Running more tests will help ensure the planner is able to handle all the different edge cases it may encounter at MCity. In addition to the standard navigation tests, trials will be conducted with spoofed traffic signals. This will evaluate the planner's ability to stop the vehicle correctly if there is a red light or obey the correct turn signals. Similarly, tests with spoofed pedestrians present at crosswalks will be conducted to verify correct behaviour. Ideally, the main components of all four challenges will be simulated in RightHook before the Year 2 competition in June.

6.2 MathWorks

At the time of creation of the MathWorks Challenge map, it was expected that the ego vehicle would only have to follow behind a delivery truck while staying its original lane. Recently, the guidelines were changed such that the ego vehicle should pass slow moving vehicles whenever possible. Looking at Figure 2.5 there are only passing lanes on State St. Coordinates have been collected while driving through the middle lane and have been delivered to the mapping sub team. Development is waiting for the mapping sub team to update the MathWorks map with this additional lane. By assigning a sibling relationship between the two lanes, the planner is capable of rerouting to the other lane if it is informed that the current lane is blocked. This will be accomplished by publishing a "Do Not Enter" sign if a vehicle is identified as being in the same lane as the ego vehicle on State St. Further work is needed to integrate and get these components.

Bibliography

[1] AutoDrive Challenge Year 2 Rules Document. SAE International, 2018.

[2] Dosovitskiy, Alexey. CARLA: An Open Urban Driving Simulator. In *1st Conference on Robot Learning (CoRL),* 2017.

[3] Simulator User Manual. RightHook Inc. October 1, 2018.

[4] MathWorks Year 2 Simulation Challenge Guidelines Rev02. MathWorks. 2019.

[5] "Cameras and sensors - CARLA Simulator", c*arla.readthedocs.io*, 2019. [Online]. Available: https://carla.readthedocs.io/en/latest/cameras_and_sensors/. [Accessed: 09- Apr- 2019].

[6] “OpenCV: Contour Features", *Docs.opencv.org*, 2019. [Online]. Available: https://docs.opencv.org/3.1.0/dd/d49/tutorial_py_contour_features.html. [Accessed: 09- Apr- 2019].

[7] S. R. Richter, V. Vineet, S. Roth, and V. Koltun. Playing for data: Ground truth from computer games. In *European Conference on Computer Vision (ECCV)*, 2016.

[8] "ROS/Tutorials/MultipleMachines - ROS Wiki", *Wiki.ros.org*, 2019. [Online]. Available: http://wiki.ros.org/ROS/Tutorials/MultipleMachines. [Accessed: 09- Apr- 2019].

[9] "ROS/EnvironmentVariables - ROS Wiki", *Wiki.ros.org*, 2019. [Online]. Available: http://wiki.ros.org/ROS/EnvironmentVariables. [Accessed: 09- Apr- 2019].

[10] "sklearn.cluster.DBSCAN — scikit-learn 0.20.3 documentation", *Scikit-learn.org*, 2019. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html. [Accessed: 09- Apr- 2019].