Predicting Desired Temporal Waypoints from Camera and Route Planner Images using End-To-Mid Imitation Learning

Thesis

Presented in Partial Fulfillment of the Requirements for the Degree Master of Science in the Graduate School of The Ohio State University

By

Aravind Chandradoss, Arul Doss

Graduate Program in Electrical and Computer Engineering

The Ohio State University

2020

Thesis Committee

Prof. Levent Guvenc, Advisor

Prof. Bilin-Aksun Guvenc, Committee Member

Abstract

This study is focused on exploring the possibilities of using camera and route planner images for autonomous driving in an end-to-mid learning fashion. The overall idea is to clone the humans' driving behavior, in particular, their use of vision for 'driving' and map for 'navigating'. The notion is that we humans use our vision to 'drive' and sometimes, we also use a map such as Google/Apple maps to find direction in order to 'navigate'. Therefore, in this study, we replicated this notion by using end-to-mid imitation learning. Besides, this work also places emphasis on using minimal and cheaper sensors such as camera and basic map for autonomous driving rather than expensive sensors such Lidar or HD Maps as we humans do not use such sophisticated sensors for driving. Therefore, in this work, we imitated human driving behavior by using camera and route planner images for predicting the desired waypoints and by using a dedicated control to follow those predicted waypoints.

The other reason behind this approach is that numerous research [1] have already been conducted in the modular and end-to-end pipeline. Both the techniques were found to be promising, however, even after decades of research, their results were found to be un-generalizable for all road scenarios indicating the need for a better approach. In particular, Waymo researchers [1] have empirically found out that the end-to-end learning approach

is not generalizable even with millions of data points. With all that said, this work tries to divide the end-to-end approach and to explore the autonomous driving problem in an end-to-mid fashion by finding a reasonable spot for ‘mid’ in the end-to-end pipeline.

This thesis also includes the work carried out for developing a 3D photo-realistic environment with Lidar and Google Earth images using SLAM, meshing, and StructureFromMotion techniques. The main application of this work is for AV simulations and pre-deployment testing in a simulator environment and the primary focus is to explore the possibilities to generate a simulation environment where one can validate autonomous driving algorithms before deployment and to conduct an experiment which would be unfeasible in real-world settings.

Dedication

To my parents, Chandra & Arul Doss.

Acknowledgments

First of all, I would like to thank my advisor, Prof. Levent Guvenc for his guidance and constant support for my thesis research. Prof. Guvenc always gave me the freedom to work on various topics, and also encouraged me to pursue the research topic that I desired. All the meetings with him were meaningful and constructive.

I also would like to thank Prof. Bilin-Aksun Guvenc for being my committee member and for all her valuable insights, suggestions, and guidance throughout my thesis research. I cannot express my gratitude enough to these two people. Thanks for everything.

Special thanks to Automated Driving Lab and its team for their thorough support during my time at ADL. Thanks for all those get-togethers and food. I would like to especially thank Xinchen for sharing our lab equipment, that too, at the very end of my thesis work.

I am deeply indebted to my family for their unconditional love and affection, and also for being with me at all times. Finally, I would like to thank my friends especially my roommates for their support. Thanks!

Vita

April 2014 .....Sri Sankara Vidyalaya MHSS, India.

2014 – 2018.....B.E. Electronics and Instrumentation Engineering, Anna University, India.

2018 - 2020 .....M.S. Electrical and Computer Engineering, The Ohio State University, USA.

Publications

Li, X., **Arul Doss, A.C.,** Aksun Guvenc, B., and Guvenc, L., "Pre-Deployment Testing of Low Speed, Urban Road Autonomous Driving in a Simulated Environment," SAE Technical Paper 2020-01-0706, 2020, doi:10.4271/2020-01-0706

Fields of Study

Major Field: Electrical and Computer Engineering

Table of Contents

List of Figures

Chapter 1. Introduction

1.1 Motivation and Intuition

The first section of the study is focused on autonomous driving with camera and route planner data. This work is inspired by and tried to imitate human driving, because, humans are able to drive seamlessly using just their vision regardless of the situation. In addition to vision, humans use basic maps such as google/apple maps to find routes in order to navigate. In other words, humans do not use any sophisticated sensor such as Lidar, Radar or HD maps for driving, they rely only on their vision and map to identify traffic signals, signs, other vehicles, pedestrians and route, and use their motor control (hand/leg) to drive the vehicle. In this work, we replicated this notion by dividing the driving task into two units, namely the perception and control unit. The perception unit acts as a vision block and is used to predict the desired waypoints for the vehicle, while the control unit is used to calculate the needed throttle and steering command to follow the predicted waypoints.

Why the camera and basic map alone? We hope it is intuitively reasonable why these two go together and why they would fail when used stand-alone! For example, consider you are using only the camera, in that case, there will be no information for the

network to take the decision in the road intersections. To overcome this, one can train branched networks for each possible decision and picking the one on the fly, which would be computationally expensive. On the other hand, if we use only the map, we would not have information about traffic signals, signs, location of other vehicles, and pedestrians to take appropriate action. In other words, the map (navigation) would only give the spatial waypoints, however in practice, we need temporal waypoints in order to navigate. Besides, in some situations, we might need to intentionally deviate from the spatial waypoints (from navigation). For example, imagine some car is parked partially on the side of the road, in that case, we would normally nudge to the other side while crossing that parked car (i.e. intentionally deviate from the spatial waypoints from navigation). For these reasons, in our study, we use camera and map data together for our perception unit to predict the desired temporal waypoints. Once the desired waypoints are predicted we use a dedicated control unit to follow those waypoints. This is the overall motivation and intuition behind the first section of this thesis.

The second section of the thesis is focused on developing a simulation environment from Lidar and GoogleEarth Image [19, 20] data. It mainly focuses on designing a reasoning reasonable workflow for generating environments by connecting the dot from various domains including photogrammetry [26, 27, 28] and robotics [21, 22]. The notion behind this work is to build a simulation environment that can serve as a test bench for pre-deployment testing and simulation. Also, the issue we found is that, although there are numerous researchers available to work on autonomy, their development is hindered

because of the lack of hardware or the resources needed to develop a simulator. Therefore, we worked on this issue and connected the start-of-arts works from various domains such as SLAM. Meshing, and SfM techniques for generating a simulation environment. In particular, we focused on combining the accuracy of lidar and photo-realisticness of google earth images.

1.2 Literature review

In the last decade, voluminous research has been carried out on autonomous driving and those research can be primarily put under categories such as a modular and end-to-end pipeline. In modular pipeline [33, 34, 35], the overall objective is subdivided into subtasks and each task is addressed separately, while on the other hand, end-to-end [3, 5, 6, 9, 15, 16, 17, 18] approach treat the objective as a single block and tries to solve it generally by using neural networks. Both approaches have their advantages and disadvantages due to the very nature of the design. To briefly highlight them, in the modular pipeline approach, the assumptions used to build the system are generally very constrained and would not encapsulate the real-world scenarios. On the other hand [14, 15, 16, 17, 18], although the End-to-End data-driven approach could theoretically solve the problem, it required an enormous amount of data to converge to an optimal solution which is practically infeasible – at least for now. Therefore, this led to the creation of new approaches, such as End-to-Mid [4, 7, 8, 9], Mid-to-Mid [1], Mid-to-End, where the part understudy is solved by using neural networks while others are solved using the classical math-based approach. Even after decades of research, the reasonable position for ‘mid’ in the End-to-End approach, as

well as, the trade-off between data-driven and math-based approach is not fully understood. In this study, we focus on the end-to-mid learning approach and tried to identify the reasonable location for ‘mid’ in the end-to-end pipeline.

To briefly mention, the research to achieve autonomous driving started decades ago, in 1989 Pomerleau showed ALVINN [11] can drive with a 3-layer network using camera and laser range finder data. Although this idea was promising, further research on using neural networks for autonomous driving was hindered by hardware required to optimize the network at the time. Therefore, the researcher [18, 33, 34, 35], tried to build a system based on the classical mathematical equation. This approach comes at the cost of the assumption used to build those mathematical equations. In practice, those assumptions were more constrained and would fail when tested in the wild. In recent times, due to the recent technological boom in hardware development, the use of neural network regained interest and voluminous research has been conducted.

In this part, we have discussed the work that are more related to our study. In Waymo’s ChaufferNet [1], the researcher tried to solve self-driving in an end-to-end fashion using 30 million data points, still were not able to generalize it for all driving scenarios. Therefore, they tried a mid-to-mid approach to solve the autonomous driving. The main focus of ChaufferNet [1] is on trajectory planning, therefore, they assumed that they have reasonable perception unit to needed data such as map, route, traffic light, speed

limit, locations of other vehicles at every instance. Similarly, Simon Hecker et al [3, 16, 17] also addressed the issue by using surround-view cameras and route planner data. Simon et al tried LSTM based network in an end-to-end fashion. Similar to our notion, Simon et al used map data as a key input, along with 360 degrees surround-view camera and GPS data. Recently, Dian et al [4] demonstrated a two-stage learning approach with DAgger [8, 9, 10] and outperformed existing results in Carla's benchmark. Dian et al, primarily used two agents for the training, where one agent would have access to privileged data such as vision, map, traffic signs, location of pedestrians and other vehicles, while the other agent would have access to only the vision data. In this work, Dian et al used a branched network for training for each possible decision in namely left, right, straight, and lane-keep (similar to [2, 5, 6, 7]). In [5,6], Codevilla et al tried conditional imitation learning in an end-to-end manner to map input image directly to steering and throttle commands. Axel et al [7], extended Codevilla et al work, by predicting the affordances for driving instead of direct control. Similarly, numerous research such as [14, 15, 18, 12, 13] can be in this domain. On analyzing, we found that most of the research was focused on end-to-end and end-to-mid (at of various levels) and many also relayed on sophisticated sensor data such as Lidar or HD maps. Therefore, in this work, we wanted to explore and restate the autonomy problem, and address it using the subtle intuition that we identify from human-driving (imitation/behavioral learning). We believe that such a subtle notion can help in restating the autonomy problem in a more solvable way. In short, in this study, we analyzed the feasibility of autonomous driving with only vision and basic map using end-to-mid learning, in particular, to predict desired waypoints using camera and map images; to use

dedicated control to execute those estimated waypoints. The reason behind this approach is discussed in the Motivation and Intuition section.

1.3 Contribution

The first contribution of this study is that we explored autonomous driving in end-to-mid fashion with more emphasis on using only vision and basic map as the inputs. We also incorporated intuition from human-driving to identify a sweet-spot for mid in the end-to-end pipeline. Finally, we provided the empirical results and showed the feasibility of our approach in an urban environment using the Carla Simulator (Unreal Engine).

The second contribution is that we proposed an effective workflow to generate a photo-realistic 3D environment using Lidar and Google earth images for AV simulations and per-deployment testing.

1.4 Scope and Organization

The thesis is organized as follows, in the first chapter, we have discussed the intuition and motivation behind our approach and also to review the recent work in the domain. In the second chapter, we have explained our approach to deal the autonomous driving problem, in particular, we have discussed the data collection, network architecture, formulation of controls, network training, and evaluation techniques. In the third chapter,

we have shown the simulation results and inferences for our work. In the fourth chapter, we have explained the workflow for environment generation and also showcased the final render of the environment. In the fifth chapter, we reviewed the work and discussed the overall results and inferences, and finally concluded with a discussion on future work.

Chapter 2. Waypoint Prediction Using Camera and Route Planner Images

2.1 Introduction

In this chapter, the overall objective, setup of our simulation environment, system architecture, data collection, data preprocessing, networks training, and evaluation methods are discussed.

The overall objective of our study is to reframe the autonomous driving problem in an end-to-mid fashion and to build suitable network architecture for the perception unit and to complete the pipeline with a control unit for executing the predicted waypoints.

2.2 Overall system architecture

As mentioned earlier, the overall diving task is divided into the perception and control unit. The perception unit is designed to take a front-facing camera image and a basic top-view map image as the inputs and predicts the desired temporal waypoints for driving as the prediction. We used separate convolutional neural networks for camera and map images before fusing them and feeding them to a dense layer for predicting the waypoints. We used early fusion technique much similar to [39] for fusing the camera and

map features. The developed perception architecture is fully differentiable as well. For the control unit, we developed a math-based algorithm to generalize a trajectory based on the prediction and to calculate the needed lateral and longitudinal control based on our vehicle kinematics. More details on network architecture and control techniques will be discussed in the later sections. The overall architecture of our system is shown in figure 1.

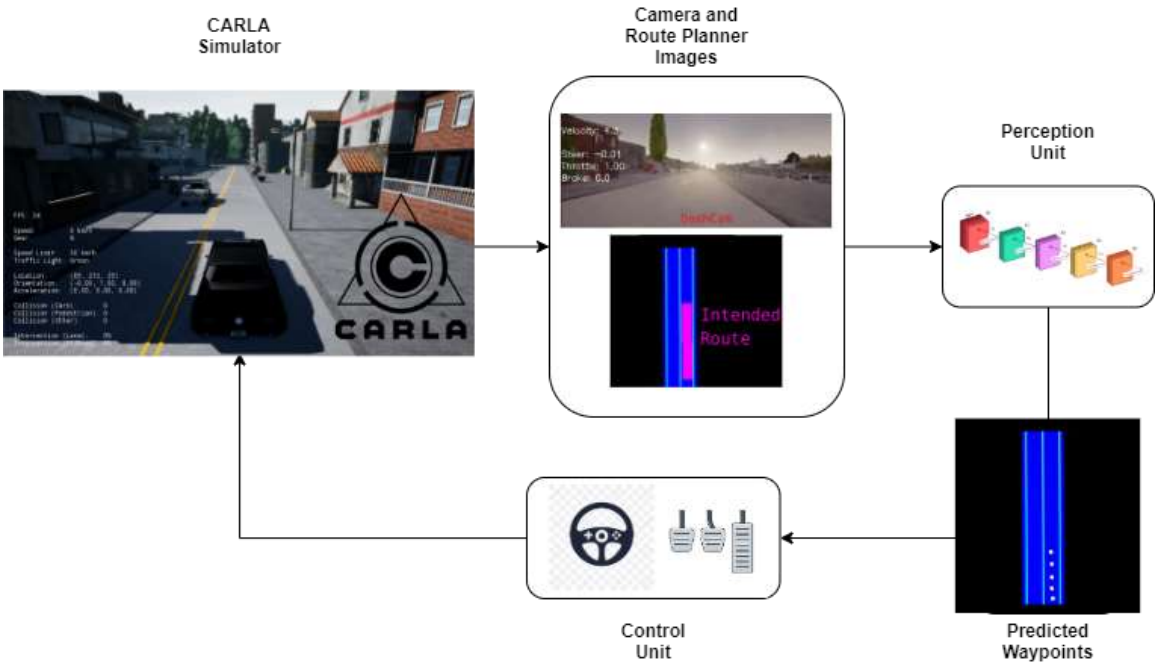


Figure 1: Over-all System Architecture

2.3 Carla Simulator

Carla is used as the core simulation environment for this study. Carla is a gaming simulation developed by A. Dosovitskiy et al [2]. and is widely used for various autonomous driving research. The Carla simulator provides various features such as sensors, traffic managers, maps, and weather conditions along with multiple towns and vehicles. In particular, the simulator provides seven towns with different landscapes ranging from rural to urban environments with various vehicles including bikes, cars, and mini trucks. The simulator also provides sensors such as RGB, depth, semantic cameras, radar, collision, GNSS, and lidar sensors. Some of the sensors are visualized in figure 2. The maps in the simulator also include open-drive information. One can get road lane information including the location of ground truth waypoint data from this open-drive. The simulator also provides various weather conditions ranging from sunny to rainy weather. For our study, we used two towns, randomly picked vehicles from all categories, RGB camera, maps information, and four kinds of weather conditions. More details on these input data will be discussed in the next section.

Figure 2: Carla Simulator [38]

2.4 Data Collection

In this study, we used two towns in Carla, namely Town1 and Town2, where the former town is used for training, and the latter town is used for testing (as a previously unseen town). Carla also provides options to change weather and traffic conditions as we needed. Therefore, for data collection, we preferred to use different weather conditions for training and testing. Similarly, we also collected the data at various traffic levels. To highlight, each town in the simulator is provided with open-drive information from which we can get data regarding the road lane information such as the location of lane center, next nearest waypoint, road junction, intended route (from path planning algorithm - global route planner), vehicle state, and other similar data. We utilized the open-drive information to generate a basic map and used it to emulate the real world navigation systems such as Google/Apple maps.

The following data are collected,

- Camera image (front-facing)
- Route planner (similar to google/apple navigation)
- Location (GNSS)
- Ego state (orientation, position, steer, throttle, velocity...)

Figure 3: Front Facing Camera - (a) real world [36]

Figure 4: Front Facing Camera - (b) Simulator

Since we were using a simulator, we made sure that the synthetic data from the simulator is much closer to the real-world data. We have compared the real-world and synthetic data in Figures 3 and 4. In practice, those real-world data for the camera and basic map images can be taken from dash-cam and screencast of google/apple navigation. Although we had the entire open-drive information from the simulator, we only considered the road, lane, and intended route information for generating the basic map image - in order to have a reasonable comparison to real-world scenarios. Therefore, no GPS/GNSS data were fed explicitly to the network. The intended route of travel is derived based on a straight forward A* based global route planner with a given start and end locations. The intended route is collected at each instance and later embedded on the basic map to generate the final input map. The ego locations are also collected along with the timestamp and later used as the target location (ground truth data - as desired future target) while training i.e. the actual locations traced by ego vehicle in the future (time-shifted locations) are used as the target locations for the network prediction during training. Besides, every state of our

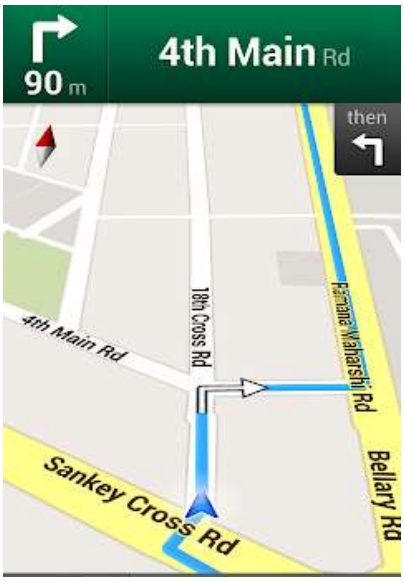
ego vehicle including velocity, steer, throttle, position, orientation, and other such measurements are also collected.

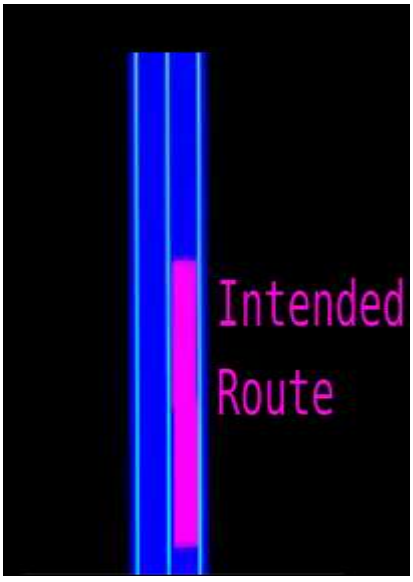
Carla's default autopilot agent is modified and used as the oracle agent. This modified oracle agent uses PID based control to reduce the cross-tract-error between the current location and nearest waypoint from the local planner. We also added intentional noises by overriding steer and throttle command at random instances while collecting the data and also by changing waypoint information in route planner. Besides, additional noises were added while generating the map i.e. noises were added to the position of ego and orientation of the map in order to emulate real-world GPS inaccuracies and latencies. The camera images are also augmented with noises before feeding them to the network.

2.5 Network Architecture

The proposed network will take two inputs images and predicts desired temporal waypoints. The two input images, namely, the front-facing camera and the top-view route planner (map) images, are 3-channeled. The front-facing camera image is a straight forward raw RBG image received from the simulator, while the image for the input map is generated from the collected data, with the intended routed embedded in it. The network would output five waypoints - represented by a vector of size 10 (5 waypoints – each waypoint is a pair of x and y coordinates in ego vehicle's frame of reference).

For our perception unit, we used early fusion technique [39], in which, the inputs images would initially be processed separately and then fused to predict the waypoints i.e. the input (camera, map) images would be processed separately to extract the high-level feature, and then the camera and map features are concatenated and fed to a densely connected layer to predict the desired waypoints. Unlike [4,5,6,7] methods, we did not use any branched network in our architecture rather we preferred a single network to handle all the possible turns at the road intersections. In other words, most of the research based on Carla simulator would inherit a branched architecture to resolve the directional ambiguity for the turn at the road intersections. Usually, they would use four branches to account for left, right, straight, and land keep, and would pick the needed branch on the fly based on local route planner. However, in our approach, we didn't follow this approach, rather preferred to have a single branch to handle all the direction at the intersection. The notion of our approach is to solve the ambiguity at the road intersection using the features from the map image. For both input images, we used res-net based architecture with a different configuration to extract the feature, in particular, we used resnet-18 and resnet-34 for map and image respectively as our base. While implemented, we removed a few layers from each block of resnet architecture. Once the features are extracted, the map and image features are linearized and concatenated to form the fused feature. We also added velocity while fusing the feature [4] for better prediction. Then, the fused feature to feed to a dense network to predict waypoints (vector of size 10, i.e. 5 pairs of (x,y)). The overall network architecture is shown in figure 5.





a. Real world [37] b. Simulator

Figure 5: Route Planner / Basic Map, (a) Real world, (b) Simulator

During the training, only the front-facing camera image was fed to the networks for every forward pass, whereas the map image was fed with a probability function. We found that this technique helps in better generalization, by removing the unintentional correlation between the map (route) and output waypoints. One can reason why this is the case, because, in most of the training data, the desired waypoints can be inferred from just the embedded route in the input map image. However, such a correlation would result in an undesirable effect in situations where the ego vehicle would have to stop. For example, imagine a J-walking pedestrian, and indeed the vehicle has to stop for the pedestrian. However, at such time, we found some peculiar results. We found that the network, trained without probability function (for map input), learned to find the correlation between input

map and output prediction and didn't stop at such situations. At the same time, the network trained with probability function (for map input), was able to stop at situations using the extracted feature from the front-facing camera. We also found out this technique made our networks to be more generalized. The intuition behind this technique will be discussed in the conclusion section.

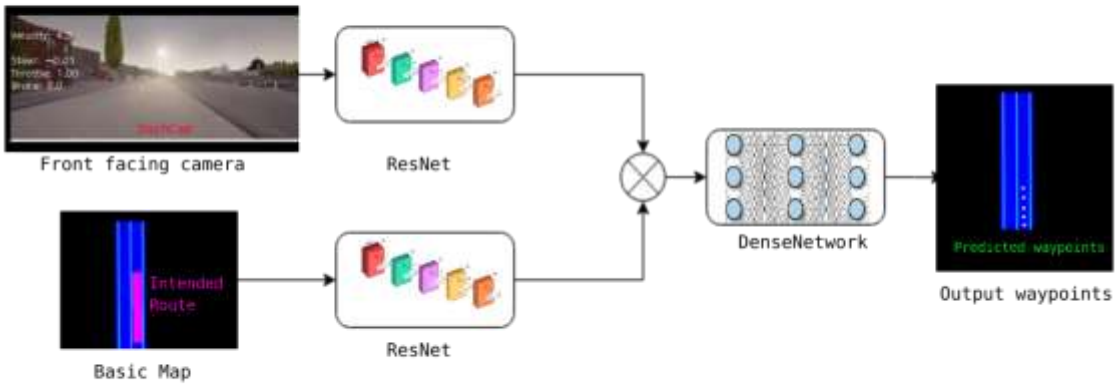


Figure 6: Network architecture of Perception Unit

2.6 Training

Our proposed network is fully differentiable and was implemented with PyTorch and trained using Nvidia GTX 2080i. We trained our network in a sequential manner with

slight variations at each step. For initial training, both camera and map images were fed as input. For the next part, we introduced decaying probability for map images, i.e. chances for map image to be fed is less in the initial part and increases with every training epoch. Besides, we found that the networks that were trained with square loss initially and with L1 loss in later part, to perform better in the evaluation stages than the networks trained separately with only L1 and L2 losses. For all the networks, we used Adam optimizer with a learning rate of 1e-4. At times, based on the losses we changed the optimizer, learning rate, and loss functions. For training, we used the future ego location as our target waypoint location.

2.7 Controls:

Once the desired waypoints are predicted by the perception unit, those waypoints are fed to the control unit to calculate the needed lateral and longitudinal control. We used the kinematics based model for vehicle control (steering and throttle). The overall workflow for this unit is similar to [7, 4] and uses the predicted temporal waypoints for driving.

To highlight, the temporal waypoints (5 waypoints) from the perception unit are in the ego vehicle's frame of reference. Therefore, we can directly use the waypoints to calculate the needed vehicle control. For lateral control, we interpolated the waypoint and calculated the needed steering based on the offset (cross-track) from the vehicle frame of

reference. Based on the offset, we used PID based control to reduce the offset. For the longitudinal control, as the predicted waypoints are in temporal sequence, we used the distance between consequent waypoints to calculate the needed throttle. Based on the distance, the needed target speed and throttle is calculated using PID control. Both lateral and longitudinal controllers are tuned heuristically.

2.8 Evaluation

In this part, the methods that we used to evaluate our model are discussed. For our study, we focused on evaluating our proposed network initially with unseen test data and later in the Carla simulator with the control unit. Due to our system architecture, we could not directly use Carla's default benchmark test. Therefore, we created modified benchmark tasks with insights from Carla's benchmark. Each task in the evaluation was designed to address the major real-world tasks such as lane-keeping, handling road junction based on the map, obeying traffic lights, and avoiding collision with other vehicles and pedestrians.

To briefly highlight, for the evaluation, we used the two towns in two settings. In the first setting, we use the same town where we collected the data but evaluate the performance at different weather conditions. In the second setting, we use a completely new town and evaluate our model for both previously seen and unseen weather conditions. Our evaluation approach is more oriented to test the perception unit, therefore, it is assumed that the control unit is reasonably certain and would calculate the needed steering and

throttle command based on the predicted waypoints. For empirical results, we followed the same metric as in [4,5], where each task would be considered a success if the ego vehicles would complete the task without any collision. We repeated each task multiple times with multiple settings as mention before. For calculation purposes, we considered the runs where the ego vehicle had traveled at least half a distance to the goal. The simulation results of our evaluation are discussed in the next chapter.

Chapter 3. Simulation Results:

In this chapter, we have showcased the results from this study and also discussed the inferences that can be made based on those results. This chapter is organized as follows; first, we briefly discussed the test scenario and visualized the inputs (front-facing camera and map images) and outputs (waypoint prediction) of our perception unit at those test scenarios. Then, the results for each task are shown and inferences are discussed.

The major tasks that we considered to handle are,

- Lane Keeping
- Road junction
- Traffic Light
- Pedestrian

3.1 Input and Outputs

There are two inputs namely front-facing camera image and basic map to our perception unit. Both inputs are 3-channeled and fed through our perception stack to estimate five temporal waypoints. Since the waypoints are predicted in our ego-vehicles

frame of reference, we can directly visualize them, here, we have visualized the output by overlaying the predicted waypoints over the map.

Figure 7: Visualizing the inputs (a) normal weather

Figure 8: Visualizing the inputs (b) Rainy weather

The basic map visualized (on the right side) will consist of road and lane information, along with the intended route overlaid on them (as a pink line). The Pink line

is extracted from the simulator's global route planner data, and as the ego vehicle travels, the pink line will also be extended such that it would always be representing a certain portion of the intended route (in the real world, one can get this data from google/apple map). The basic map is shown in the Appendix section.

Figure 9: Visualizing the input (c) J-walking pedestrian

In the above visualization (figure 8), the pedestrian is made to cross the road when the ego vehicle reaches a certain distance from the targeted location. In Figures 6 and 7, we have changed the weather conditions and visualized the inputs.

3.2 Lane Keeping

For this task, our objective is to predict waypoints such that we would drive in the center of the lane. It also includes the cases where the ego vehicle has to turn at road corners. For evaluation, we picked the roads with turns from both the towns and tested them at different weather conditions. The results are as follows,

3.3 Lane Keeping – Driving Straight

In this section, we have shown the predictions of our agent at the various instance of time and analyzed it based on the test scenario.

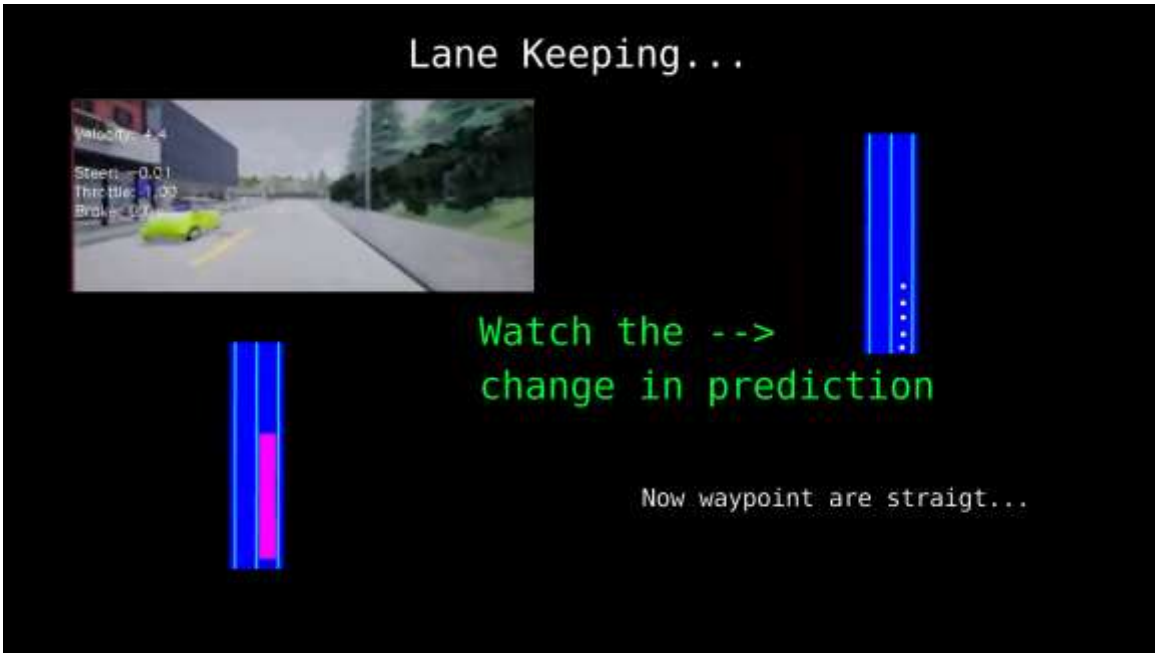


Figure 10: Lane-keeping – Driving Straight – Normal weather (a)

For visualization purposes, we have overlaid the waypoint predictions on the input map (right side in figure 9), i.e. only the locations for the white dots are predicted by our perception unit.

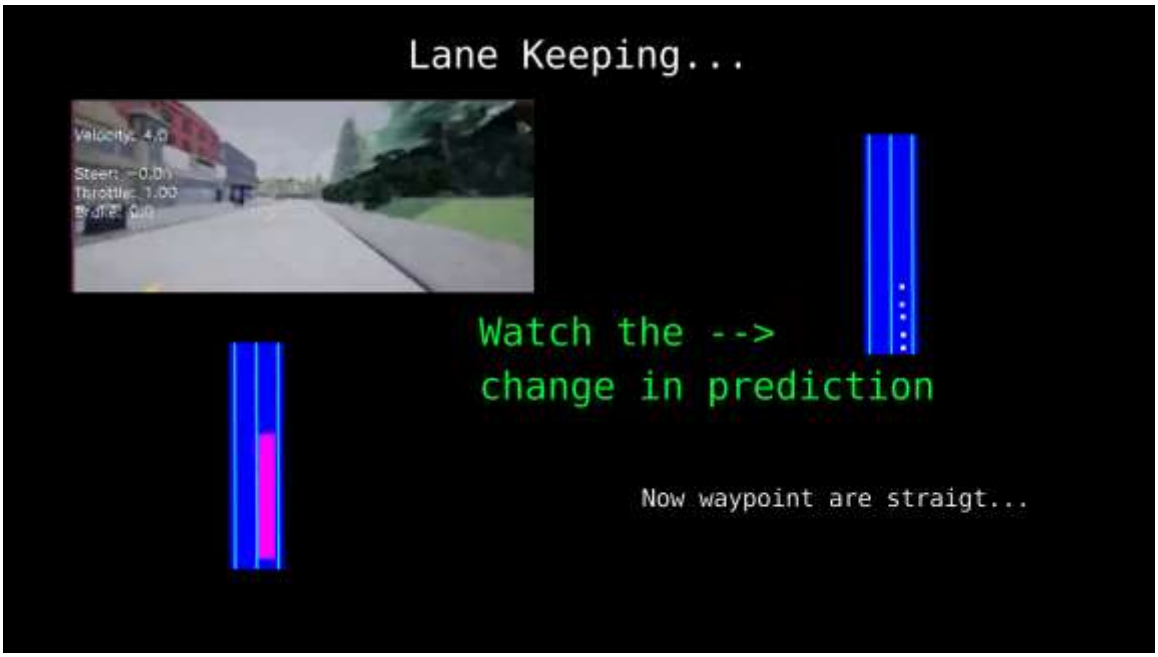


Figure 11: Lane-keeping – Driving Straight – Normal weather (b)

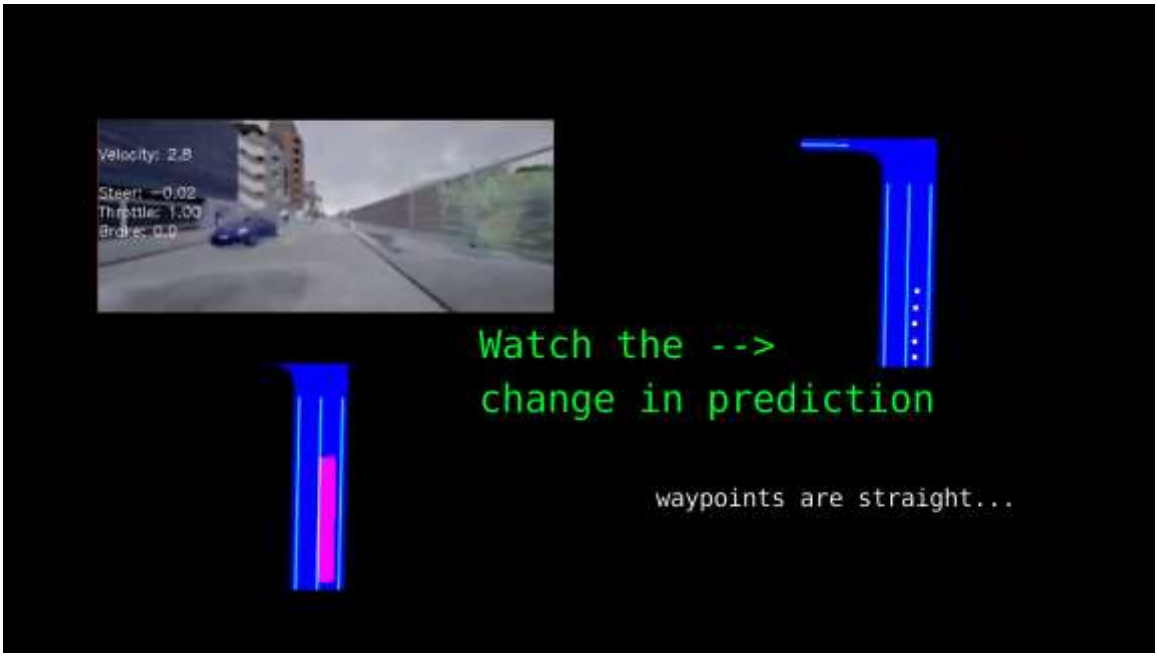


Figure 12: Lane-keeping – Driving Straight – Rainy weather

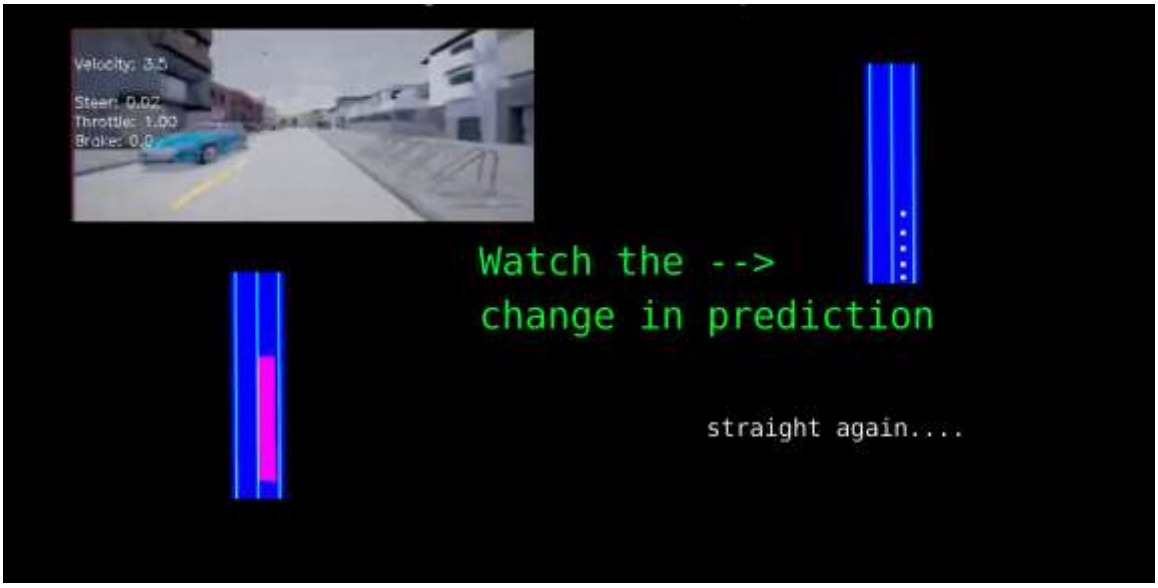


Figure 13: Lane-keeping – Driving Straight – Different Town

From the above figures, we can infer that the output predictions are in our desirable region and are oriented towards the center of the lane. These predictions are further used to plan trajectory and to calculate the needed throttle and steering.

3.4 Lane Keeping – Turning at corner

In this part, we extended the previous scenario where our ego vehicle has to take a turn (road corner). The output predictions are visualized at different instances of time for better understanding.

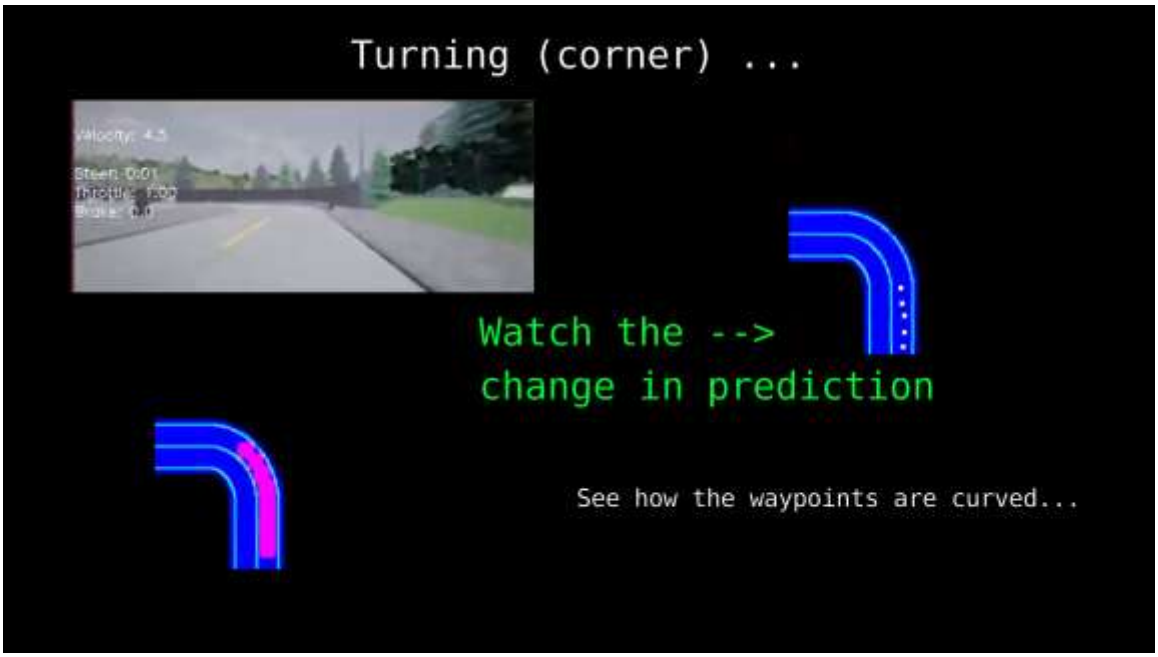


Figure 14: Lane-keeping – Cornering (a) - waypoints before taking the turn

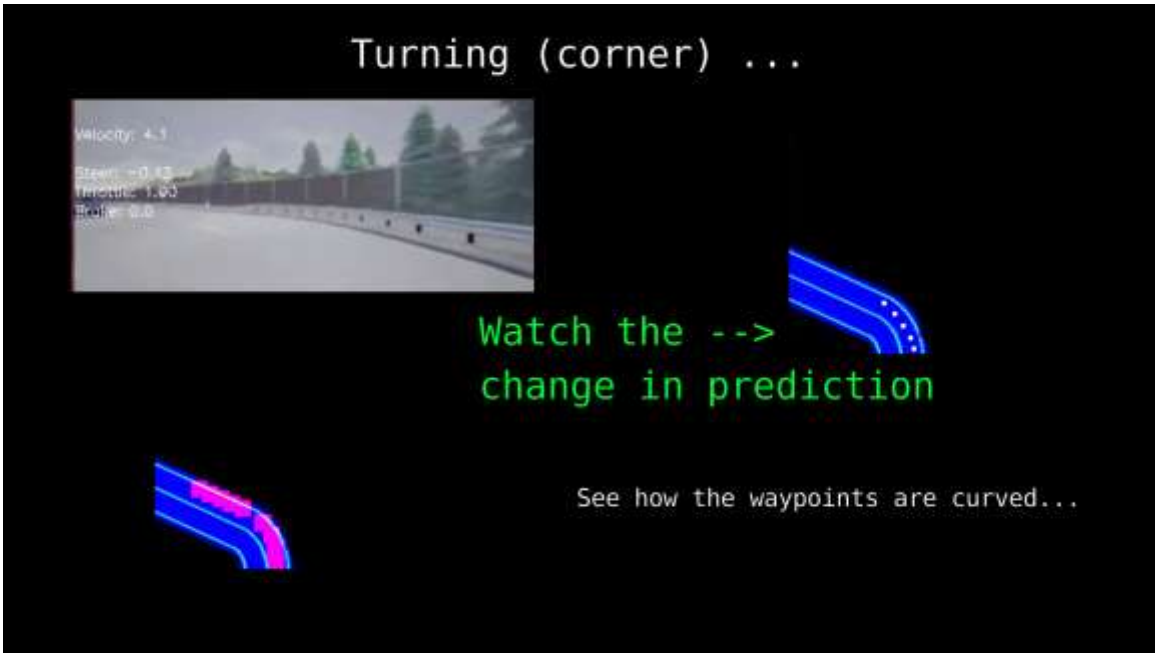


Figure 15: Lane-keeping – Cornering (a) – waypoints while taking the turn

From the above results, we can infer that the output waypoint predictions are in accordance with the situation i.e. curved when taking the corner and straight when driving in the straight road. If we ponder, we can also see that the waypoints are well-spaced when driving straight and slightly closer when taking the turning, indicating that the ego-vehicle has to slow down to take the turn.

In the next case, we changed the weather condition to mild rain and evaluated our agent's performance. We found that our agent predictions were similar to our previous case, i.e. the waypoints were curved when taking the turn and straight after taking the turn.

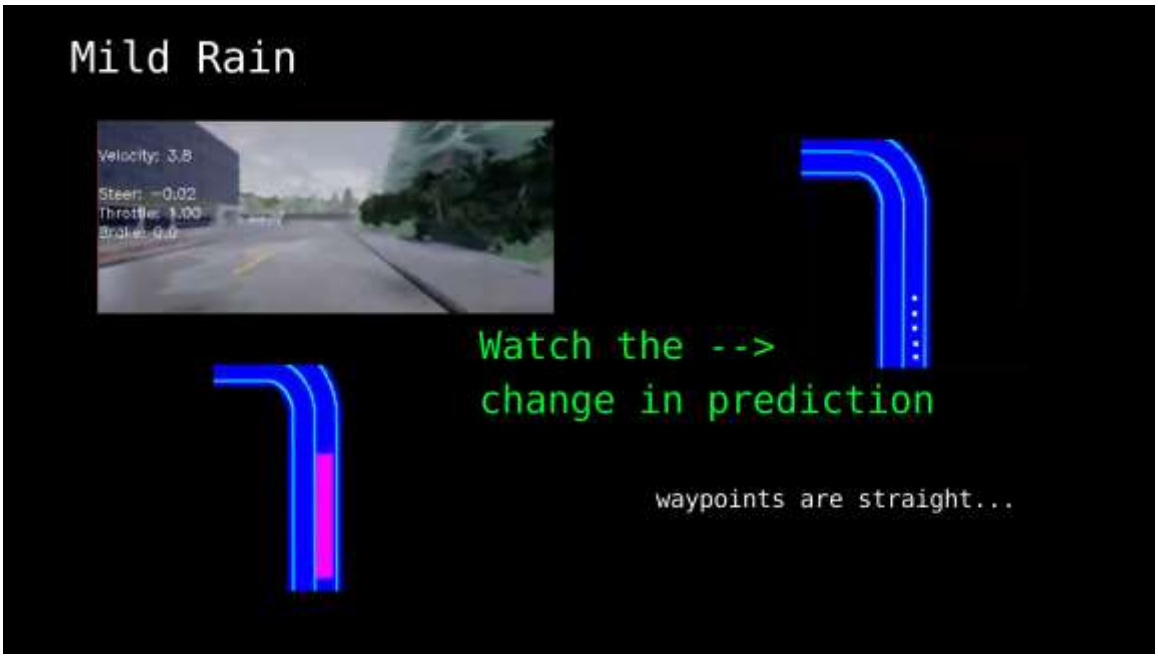


Figure 16: Lane-keeping – Cornering (b) – waypoints before taking the turn at mild rain

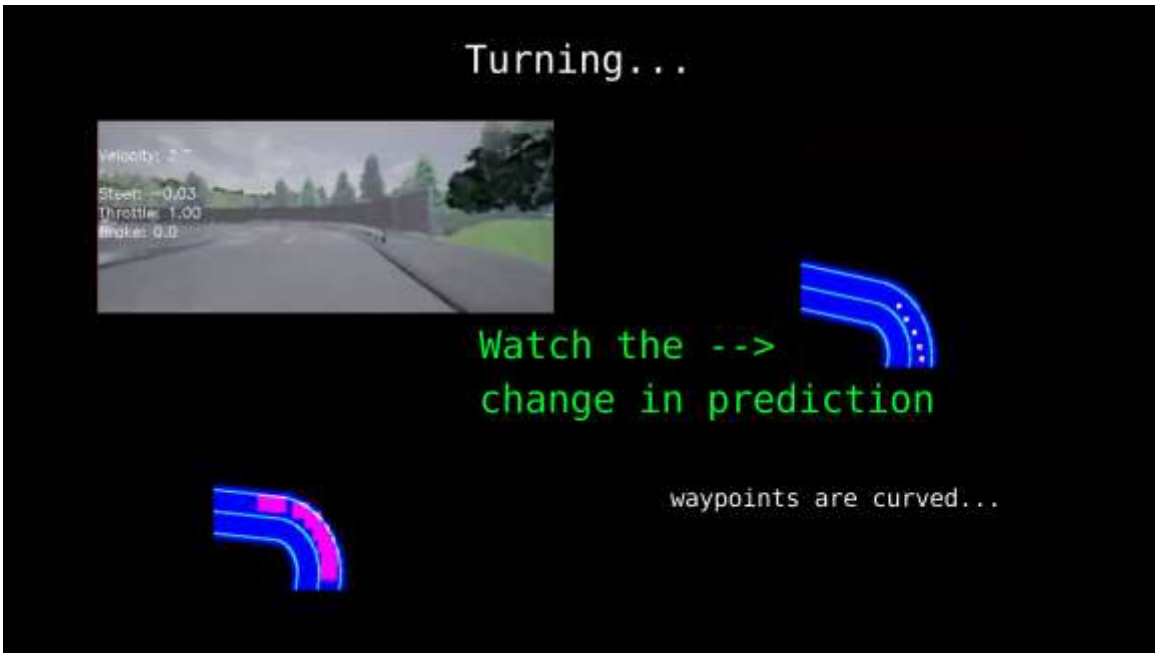


Figure 17: Lane-keeping – Cornering (b) – waypoints while taking the turn at mild rain

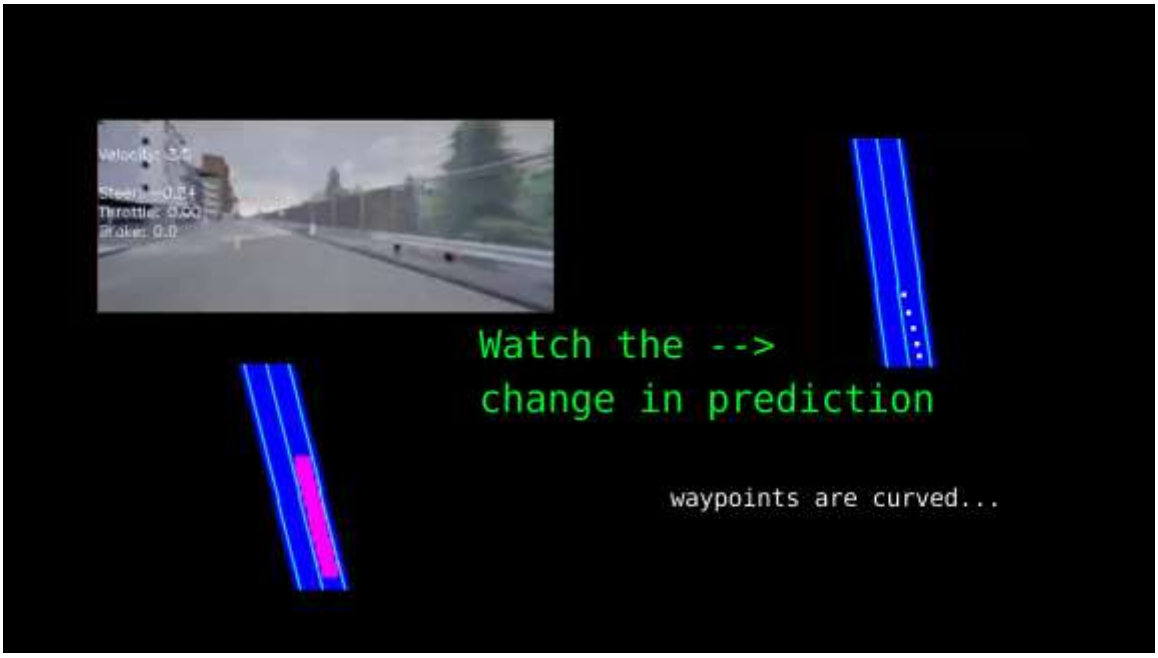


Figure 18: Lane-keeping – Cornering (b) – waypoints after taking the turn at mild rain

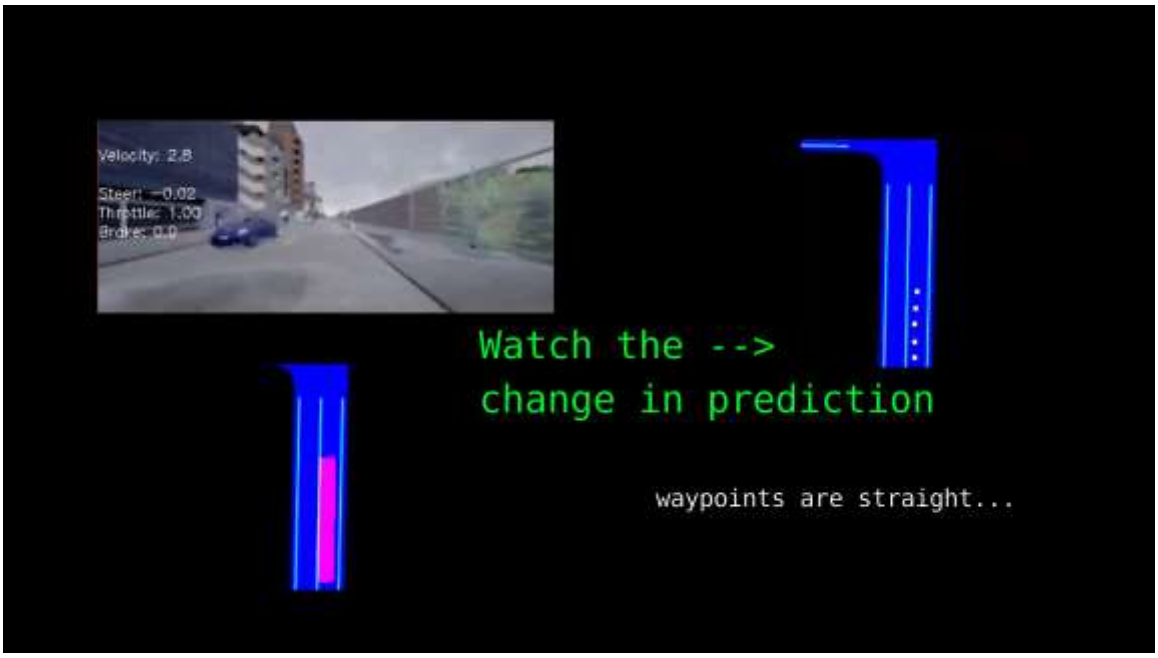


Figure 19: Lane-keeping – Cornering (b) – waypoints after taking the turn at mild rain

From the above results, we can infer that predicted waypoints are according to the situation. And if we see in fig 16 and 17, our agent is able to recover from the situation where the input map is tiled. In the real-world, this can be due to noises or latencies due to GPS in our navigation system.

In the next case, we tested our agent in a new town and with a new weather condition. The results for lane-keeping and cornering are below,

Figure 20: Lane-keeping – Driving straight – a new town with new weather

Figure 21: Lane-keeping – cornering (c) – waypoints before turning

Figure 22: Lane-keeping – cornering (c) – waypoints while turning (a)

Figure 23: Lane-keeping – cornering (c) – waypoints while turning (b)

Figure 24: Lane-keeping – cornering (c) – waypoints after turning

In the above examples, we can infer that predicted waypoints are also changing based on the situation. In figures19, 20, and 23, the waypoints are straight and well-spaced, while in figures 21 and 22. the waypoints are curved based on the turn. We can also see that the waypoints are comparatively shorter during the turn indicating that the ego vehicle has to slow down to make the turn.

3.5 Road Junction

In the previous case, the scenario was simple, the ego was not forced to make a decision, and it had to only predict waypoints to drive in the lane center. However, in this task, we extend the scenario to include road intersection/junction, where the model has to resolve the ambiguity in the direction of its turn. Here, the notion is that our model should infer the direction of its turn form the features extracted from the input map image.

Similar to the previous task, we have shown the results at various instances of time for better understanding.

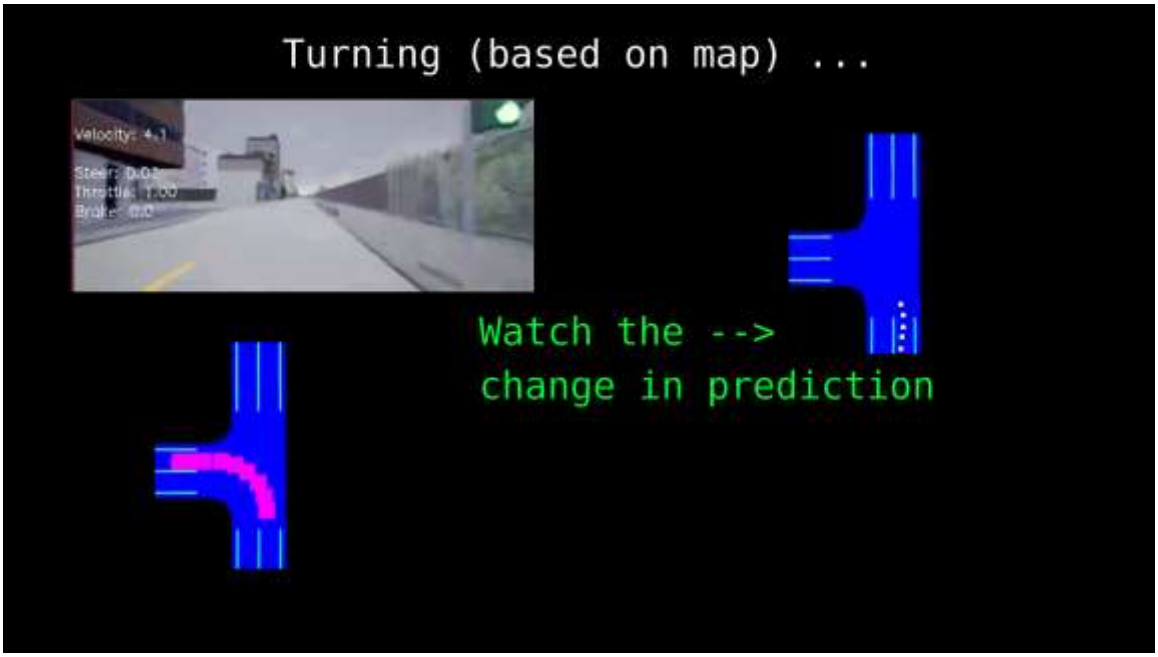


Figure 25: Turning at Junction – based on the map – waypoints before turning

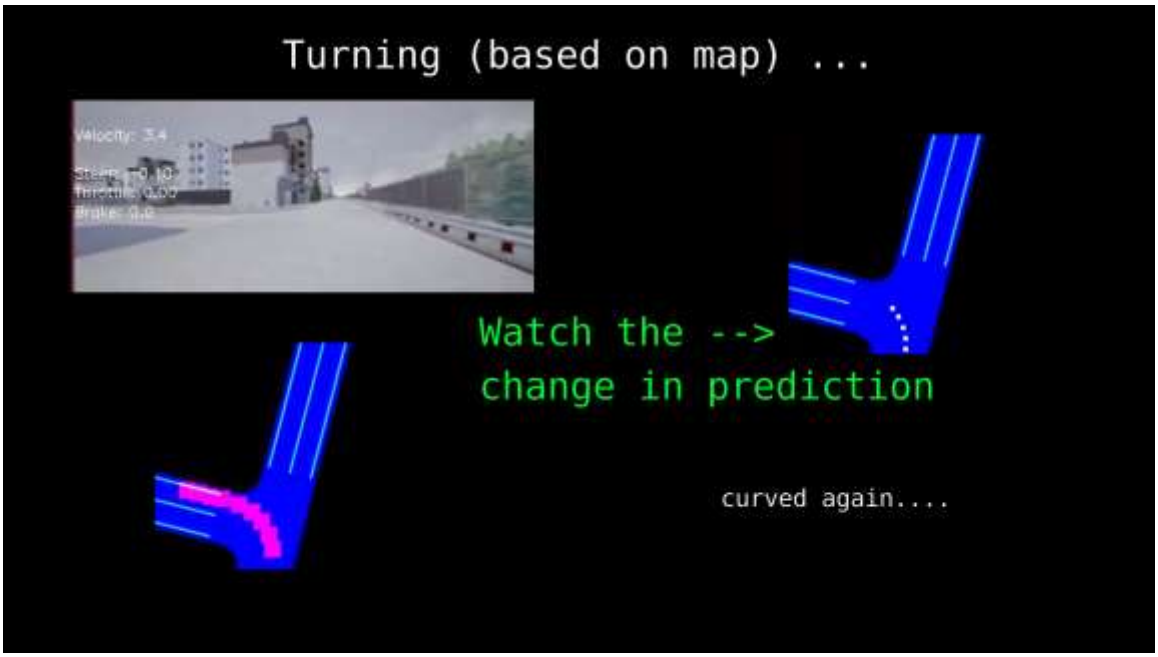


Figure 26: Turning at Junction – based on the map – waypoints while turning (a)

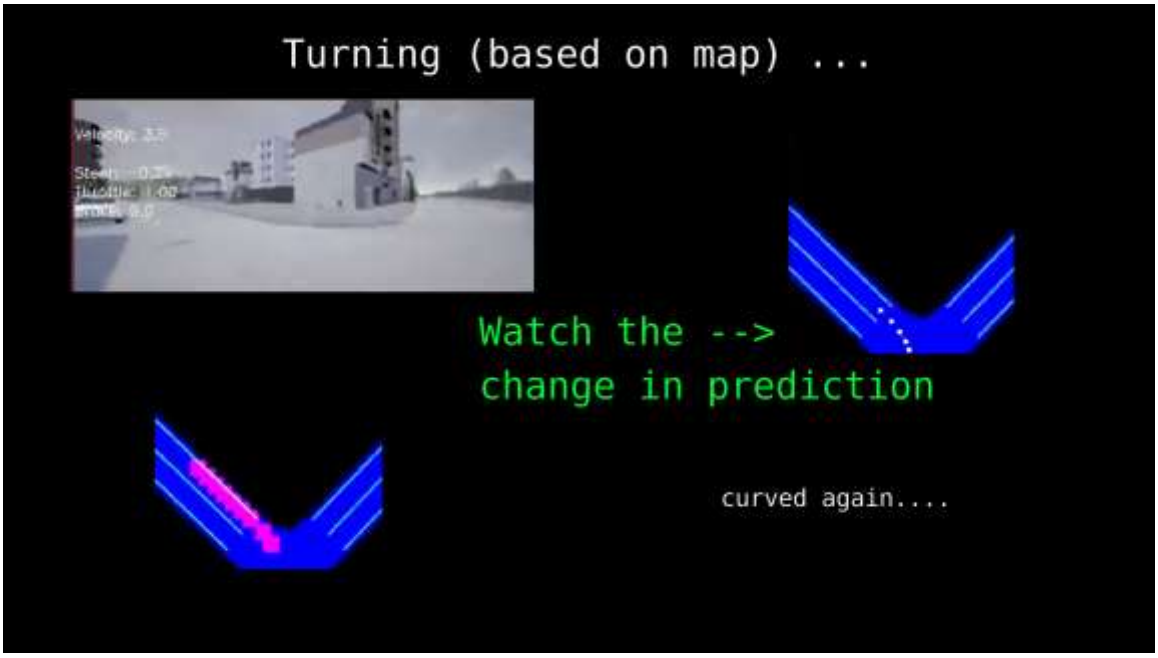


Figure 27: Turning at Junction – based on the map – waypoints while turning (b)

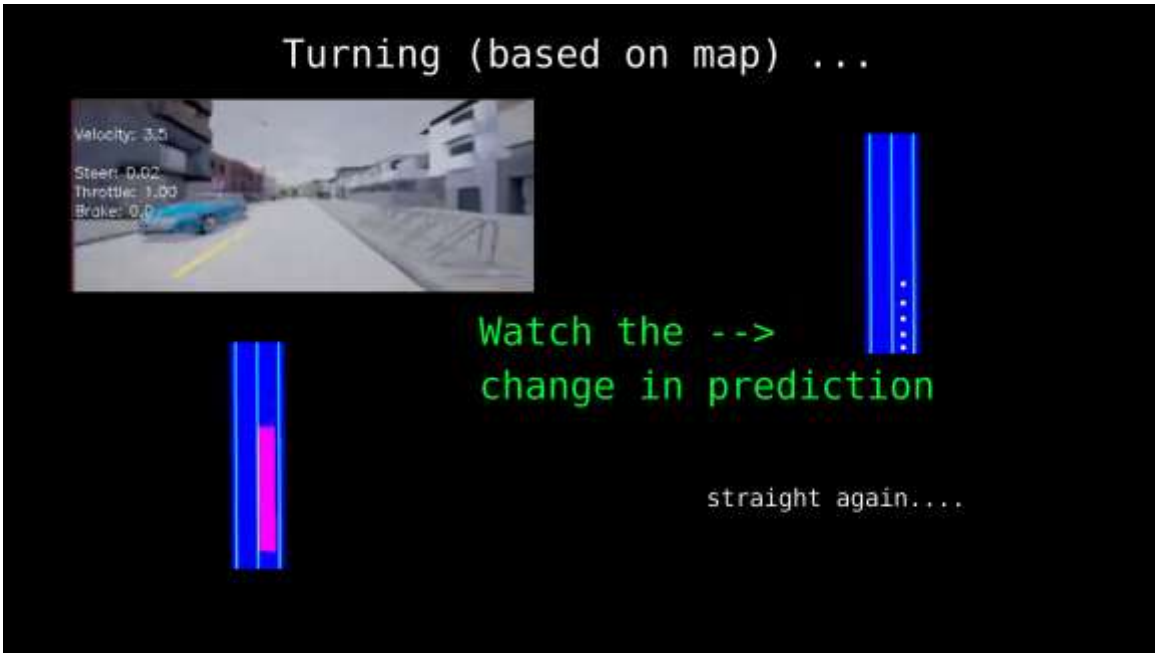


Figure 28: Turning at Junction – based on the map – waypoints after turning

From the above examples, we can infer that the waypoints are based on the embedded route in the map image, indicating the model is able to resolve the ambiguity while turning at the junctions. This would be a good place to justify the need for the input map for our model. Imagine the same scenario, however, we have not included the input map, in such cases, the agent would not be able to handle the intersection. We hope it is intuitively understandable. To overcome this, [4, 5, 6, 7] have used a branched network for each direction and pick the appropriate network on-the-fly using supervisory control. In our work, we did not include any such branched networks or supervisory control, but rather we used the basic map and single linear network to handle all the directions. From figures

24, 25, and 26, we can see that predicted waypoints at the junction are based on the embedded route in the input map.

3.6 Traffic Light

In this task, we focus on evaluating our agent on handling the traffic lights. For this, we pick routes with a traffic light and turned the traffic light to red while the ego reaches near it. Our objective is to make the ego vehicle stop for the red light and to resume again for the green light.

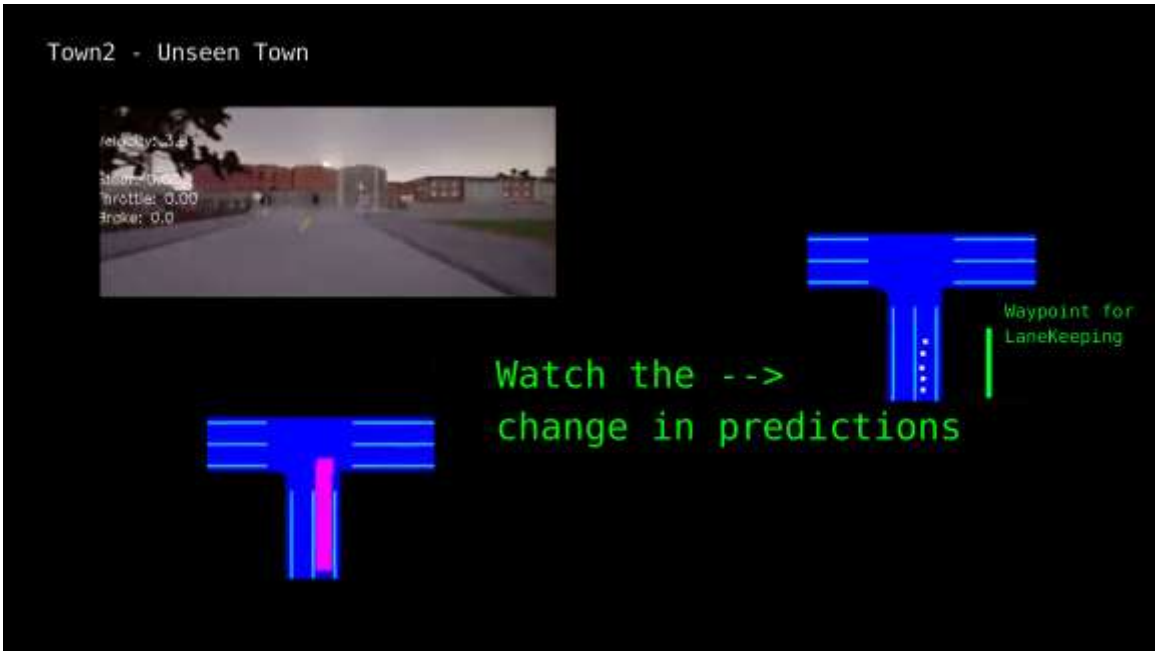


Figure 29: Stopping for a Traffic light – before seeing the red light

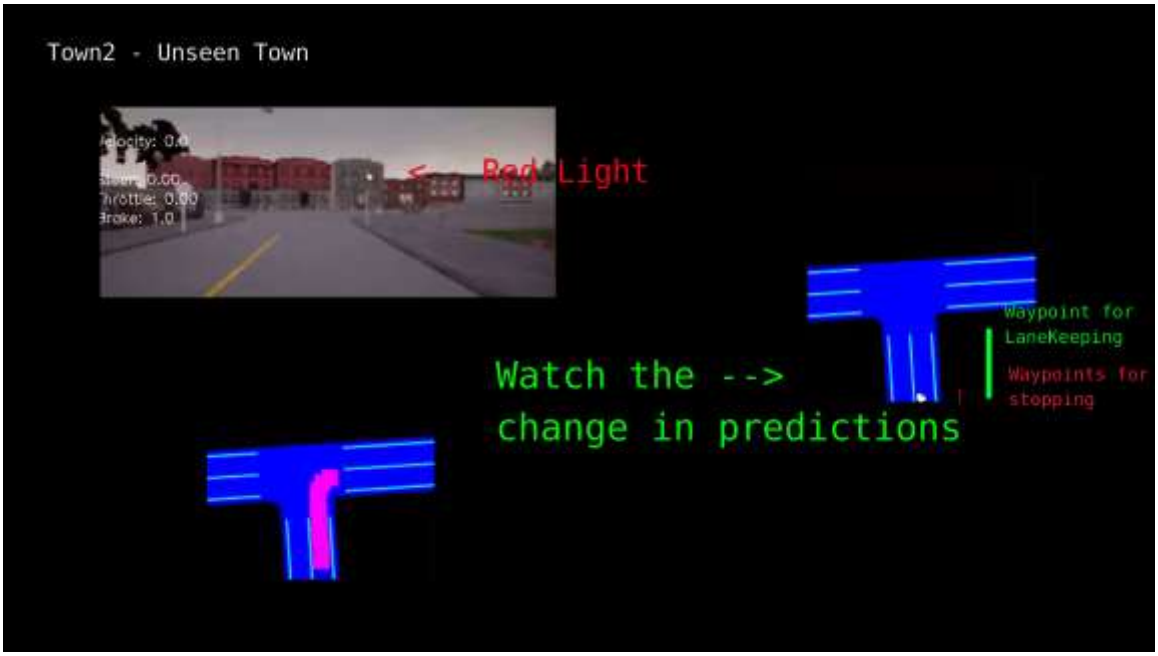


Figure 30: Stopping for a Traffic light – after seeing the red light (a)

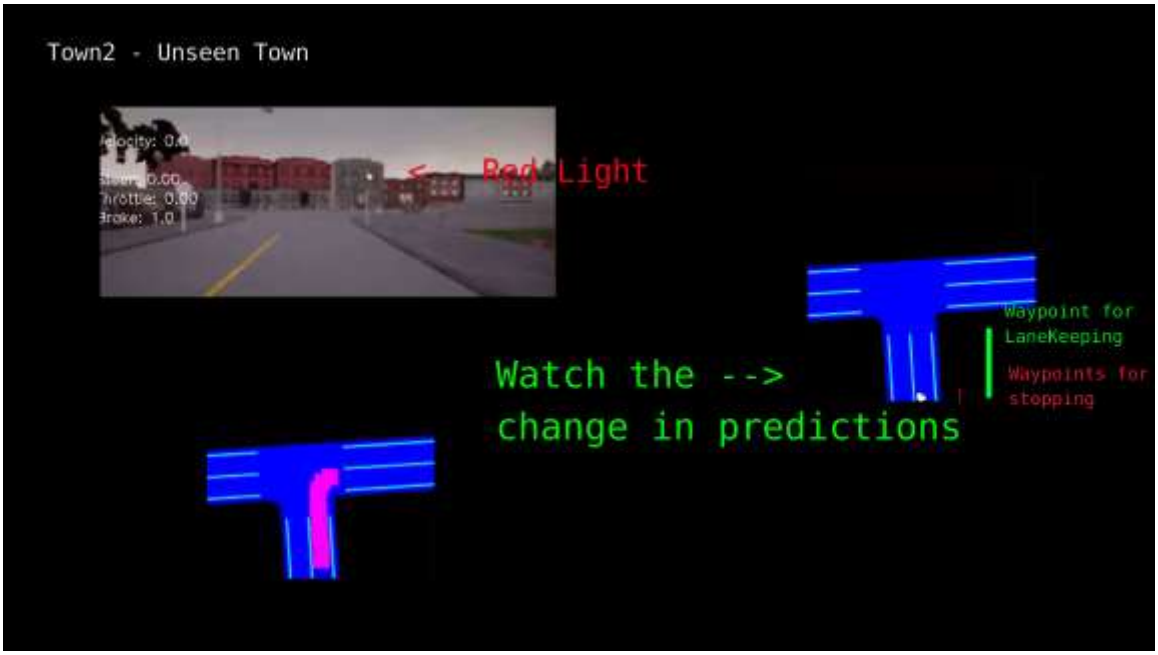


Figure 31: Stopping for a Traffic light – after seeing the red light (b)

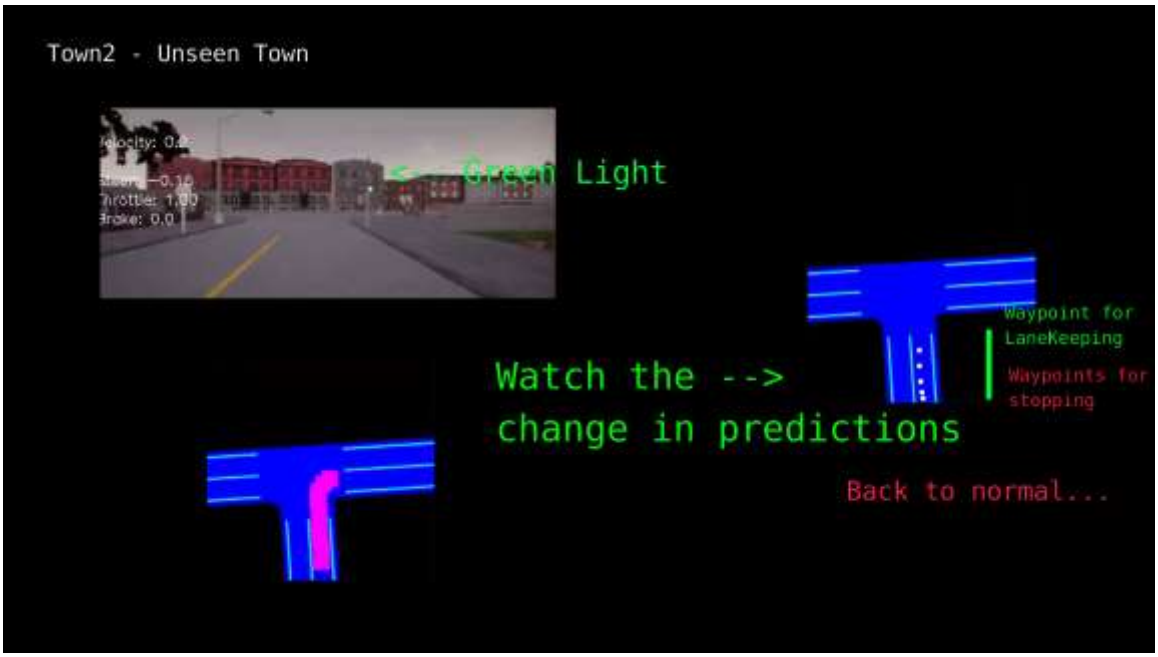


Figure 32: Stopping for a Traffic light – after seeing the green light

From the above examples, we can see that the waypoints are well apart for lane-keeping and as it reached the traffic light/junction, the waypoints start to come closer indicating that ego-vehicle has to slow down. In the other example [figures 29 and 30], we can see that the temporal waypoints are very short in fact cluttered at the same location, indicating that the vehicle has to stop completely. Controls for such stopping can be designed with our control unit based on the cluttered waypoints. Once the light turns green [figure 31], the predicted waypoints are again straight and temporally well-apart indicating that the ego can resume the travel.

We tested the same task at varying weather conditions and predicted waypoints were as expected. We have to mention that the control unit has to be tuned properly to get the needed results.

Below are the results we got when testing at rainy weather.

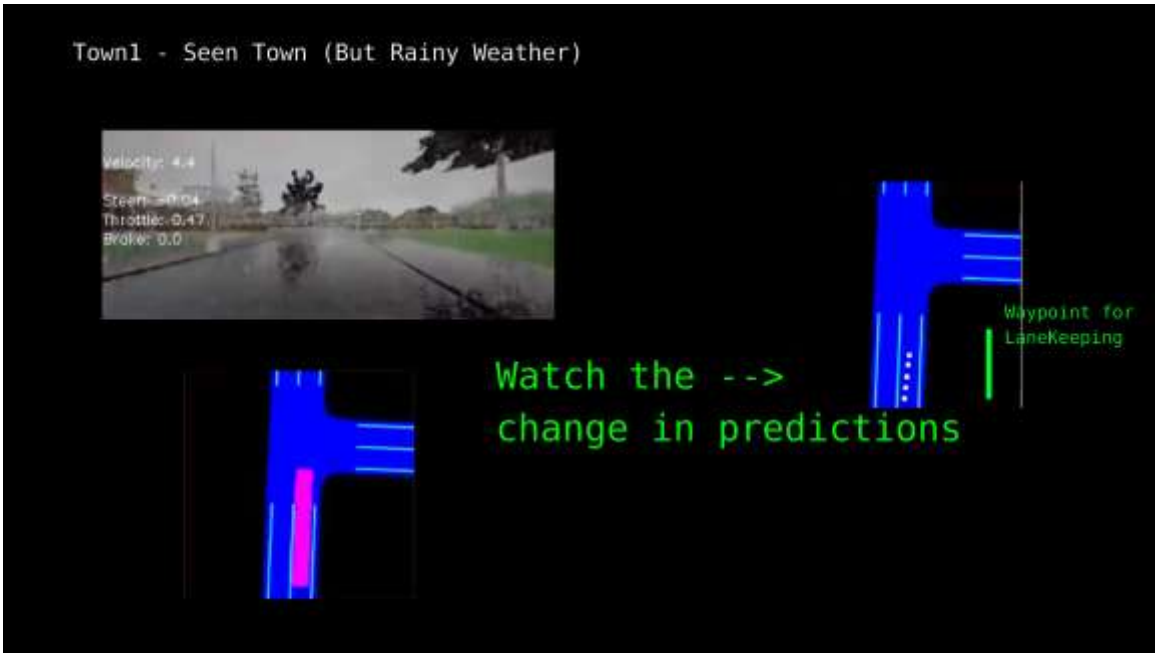


Figure 33: Stopping for a Traffic light – before seeing the red light

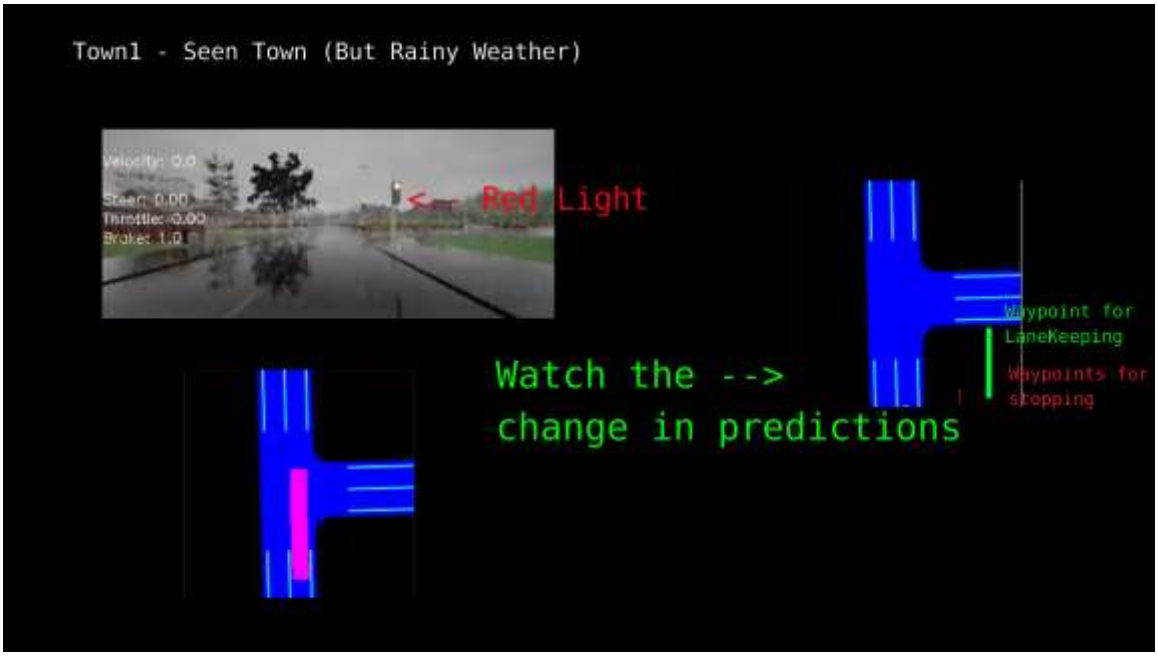


Figure 34: Stopping for a Traffic light – after seeing the red light (a)

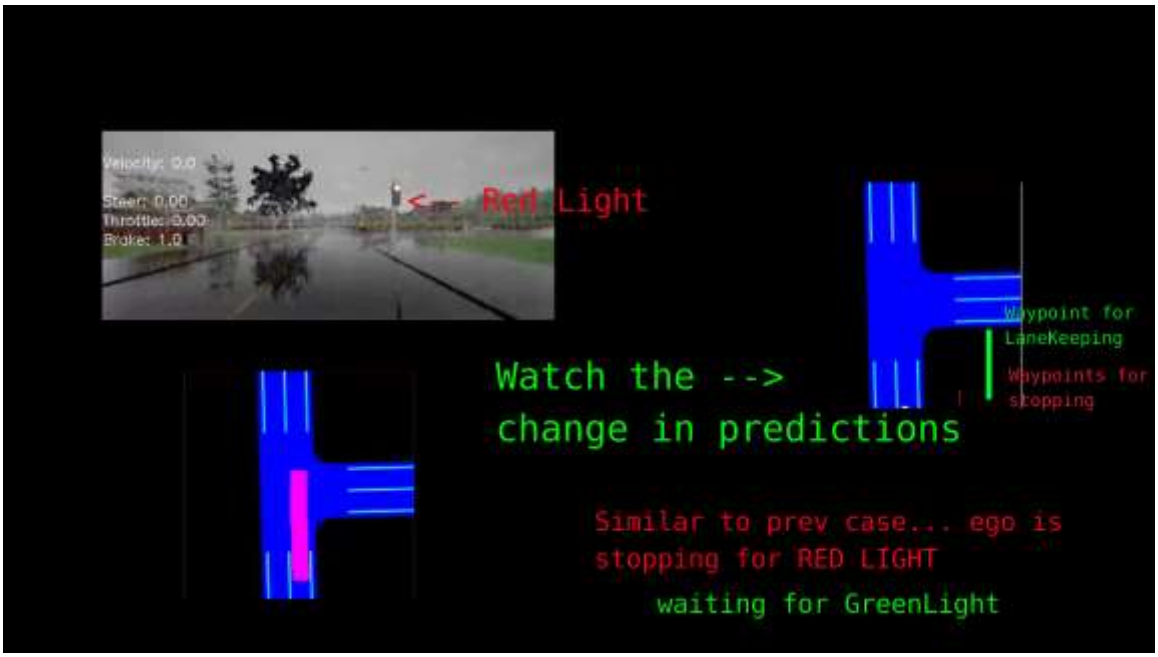


Figure 35: Stopping for a Traffic light – After seeing the red light (b)

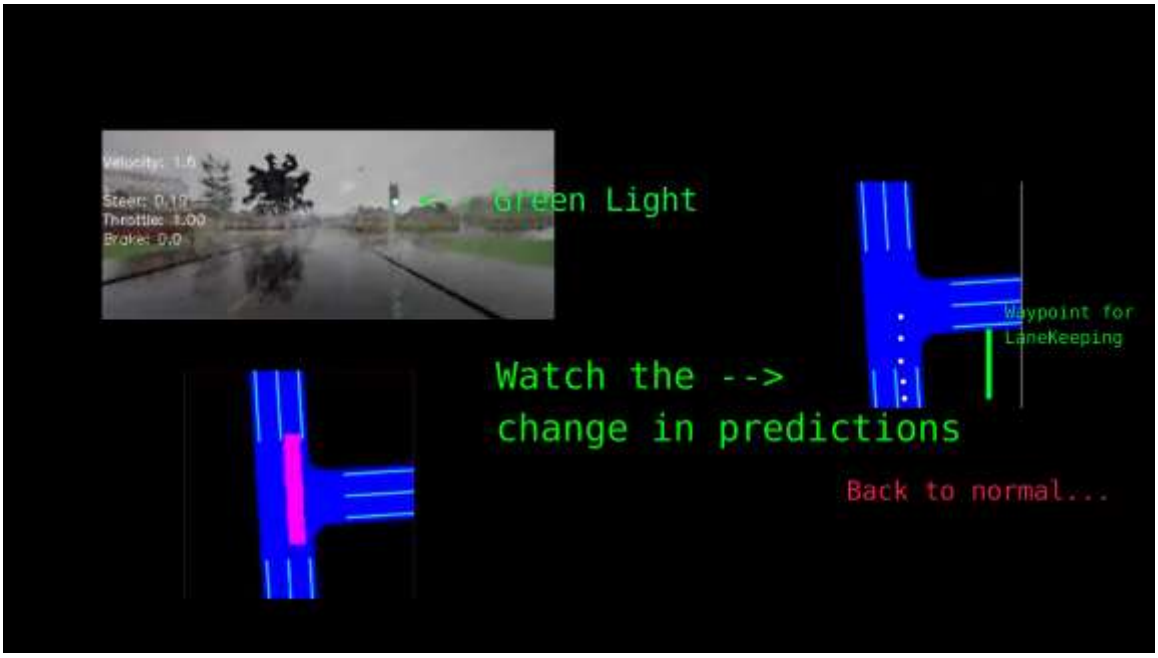


Figure 36: Stopping for a Traffic light – after seeing the green light

When tested at the rainy weather, similar to the previous case, the predicted waypoints were cluttered when seeing the red traffic light and changes back to well-spaced waypoints when the traffic light turns green.

3.7 Pedestrian

In this task, we evaluate our agent on handling J-walking pedestrian. For this, we created scenarios and made pedestrians cross the road at different timings based on the ego location. Here, the objective is to predict waypoints that would avoid collision with the pedestrians either by slowing down or by completely stopping.

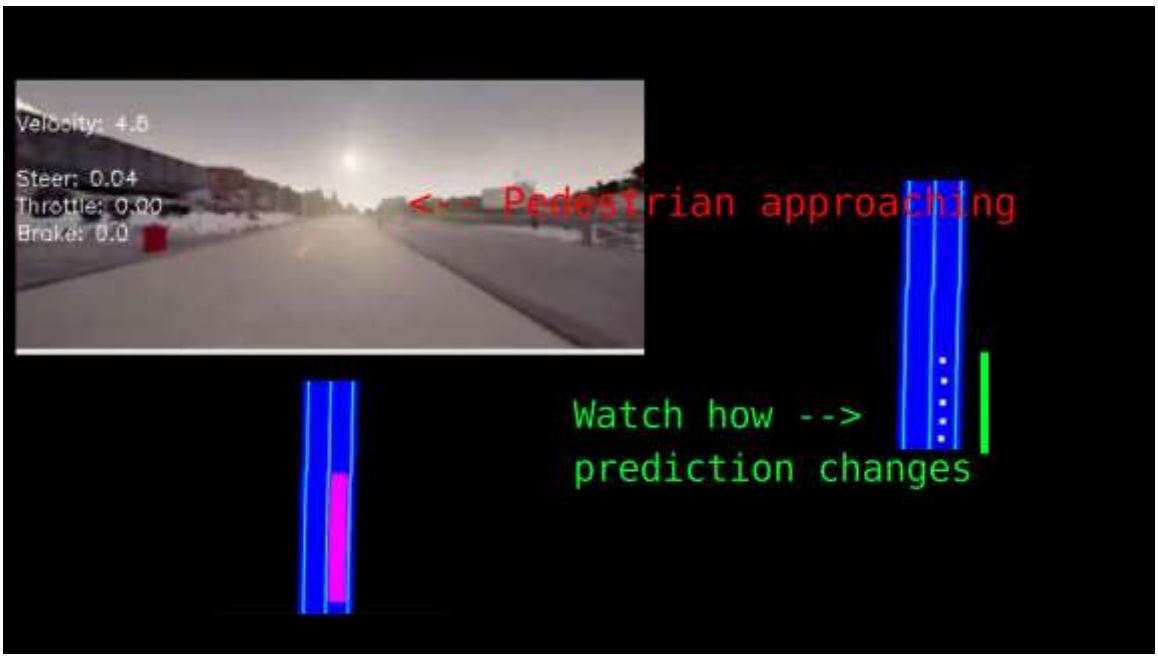


Figure 37: Slowing down for J-walking pedestrian – before pedestrian J-walks

In this next example, we made the pedestrian to J-walk such the ego vehicle has to slow down in order to avoid the collision. In figure 36, when there is no pedestrian, the predicted waypoints are straight and well-spaced (similar to lane-keeping). Once the pedestrian is detected [in figure 38] the predicted waypoints are shorter and comparatively closer to each other. These shorter waypoints are processed by the control unit to find appropriate throttle/brake to reduce the ego speed.

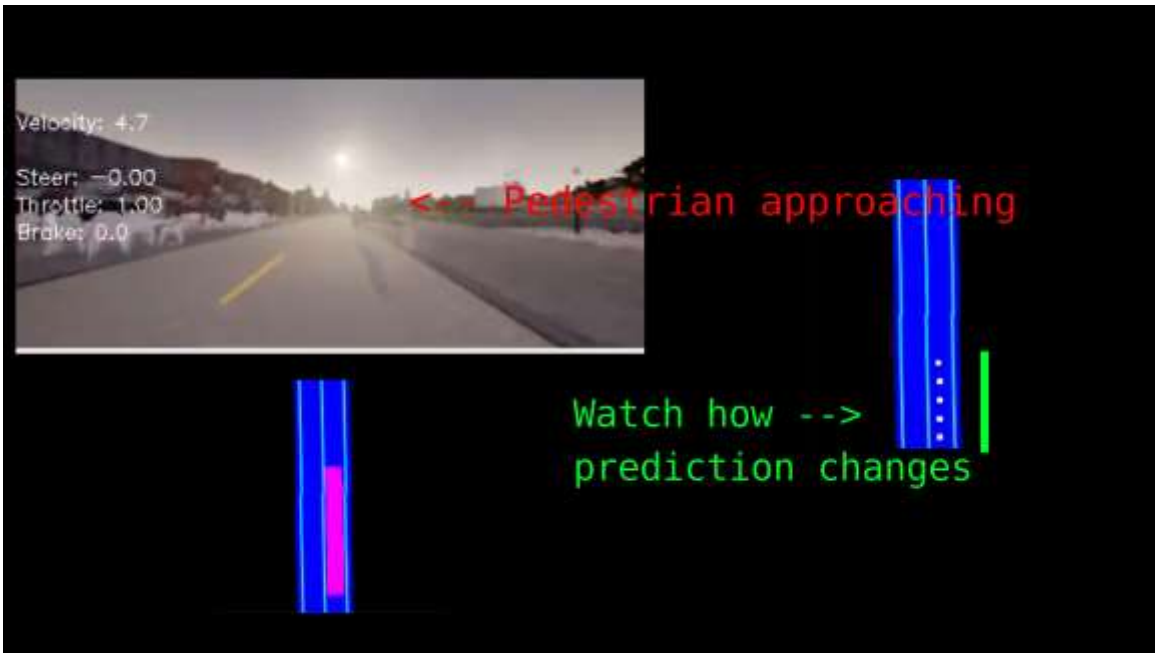


Figure 38: Slowing down for J-walking pedestrian – when pedestrian J-walks (a)

From figure 37, when there is no pedestrian, the predicted waypoints are similar to lane-keeping. Note the length of the waypoints is mark using a green line, which we would use as a reference for the upcoming time instances.

Figure 39: Slowing down for J-walking pedestrian – when pedestrian J-walks (b)

We can see that, length of the red line is much shorter than the green line, indicating that the ego vehicle has to slow down. The process of slowing down is handled by the control unit based on the waypoints (length).

In the previous scenario, we only saw the ego-vehicle was able to slow down for J-walking pedestrian. However, in this scenario, we made the pedestrian cross the road, right in front of the ego-vehicle. Here the ego must stop completely in order to prevent the collision.

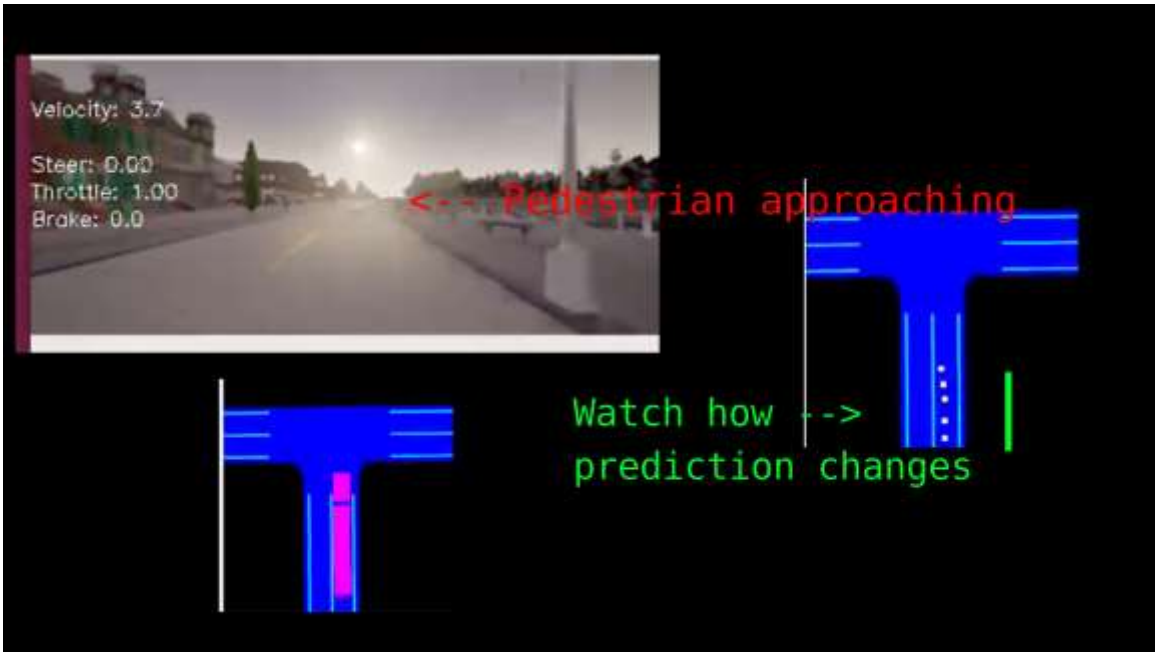


Figure 40: Stopping for J-walking pedestrian – before pedestrian J-walks

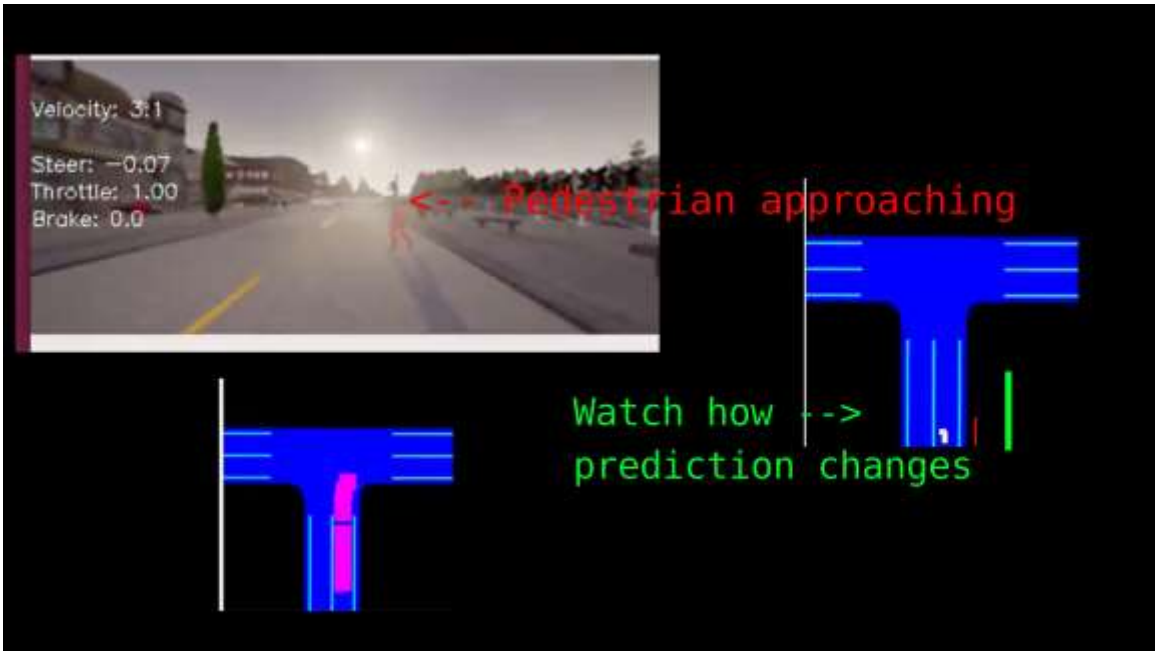


Figure 41: Stopping for J-walking pedestrian – when pedestrian J-walks (a)

Figure 42: Stopping for J-walking pedestrian – when pedestrian J-walks (b)

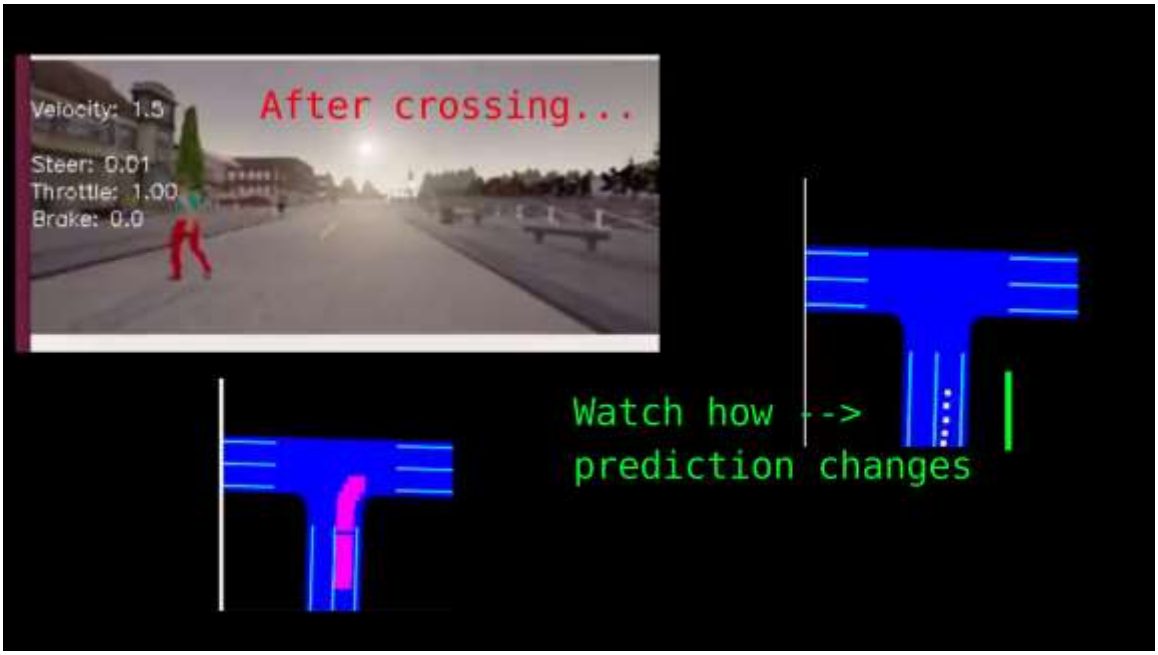


Figure 43: Stopping for J-walking pedestrian – after pedestrian J-walks

If we notice, we can see the pedestrian in figure 40 is crossing slower than in figure 37. We can also see from figure 41, the predicted temporal waypoint very short and cluttered than in figure 38. These cluttered waypoints are handled by the control unit – which would stop our ego vehicle. Similarly to the previous case, once the pedestrian finishes crossing our lane, the predicted waypoints are back to normal as before i.e. straight and well-spaced [figure 42]. We repeated all the tasks at multiple weather conditions for multiple times.

3.8 Inference

From all the above tests, we can infer from that our agent is able to predict desired waypoints

- to lane keep,
- to turn at the corners,
- to take decisions at the intersections,
- to obey traffic lights and
- to avoid collision with pedestrians.

In addition, the predicted waypoints are comparatively shorter (length-wise) while approaching the corners or the road intersections indicating that the vehicle has to slow down.

Chapter 4. Environment Generation:

In this chapter, we have showcased our work on generating photo-realistic environments for simulations and pre-deployment testing purposes. In particular, we have discussed motivation and need behind this work and briefly on existing state-of-art techniques, and finally proposed a reasonable workflow to generate a 3D photo-realistic environment from Lidar and Google Earth images. We have also showcased the rendered images of the environment generated using our approach.

4.1 Motivation

The main reason behind this work was that we needed a simulation environment to conducted experiments and to evaluate our algorithms before deploying them in the real world. However, the needed software and tools were expensive. Therefore, we decided to develop a custom workflow to generate the environment using existing open-sourced software and the tools we had in our lab. For this work, we decided to use lidar to scan the environment (for its accurate measurement) and to use google images (to get photo-realisticness). One can also develop the environment using lidar or google image (independently), however, the generated environment would come with the cost i.e. the

environment would either lack photo-realisticness or accuracy respectively. Therefore, in our approach, we wanted to combine the accuracy of the lidar and photo-realisticness of the google images to generate our final environment.

4.2 Proposed methodology

The procedure we propose focuses on connecting the existing state-of-art tools in photogrammetry domains such as SLAM, Meshing, and StructureFromMotion. We found the rendered environment to be reasonable. The workflow [42] is as follows, first, we generated the point cloud of the entire route that we want to map. We fused all the instantaneous point cloud to create a single dense point cloud using SLAM. Then, we meshed all those point clouds to generate the 3D mesh of the environment. Here, the generated mesh would lack the color details as it was generated only using a lidar point cloud. Therefore, we generated another colored point cloud from google earth images using the StructureFromMotion technique. Finally, we combined the colored point cloud as the textures to the 3D mesh generated from the Lidar point cloud. As a result, the final generated mesh is accurate and photo-realistic.

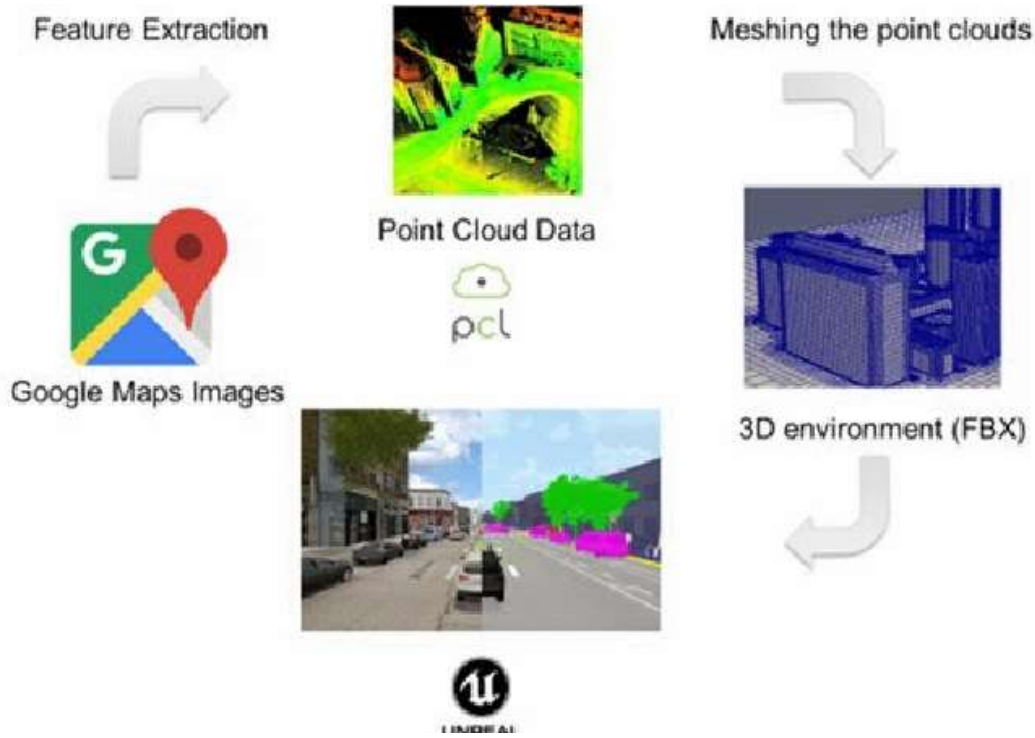


Figure 44: Our proposed workflow for generating environment

For this work, we only used existing open-sourced tools such as MeshLab, PCL, Meshroom, Autoware, Blender, and similar tools. Images of the rendered environment are shown in the results session of this chapter. We also imported and conducted various experiments with the generated environment in a simulator such as Carla (Unreal).

The procedure to replicate our work is discussed in the following sections. The major steps of our workflow are as follows, pre-processing, SLAM, mesh generation, SfM, depth map, and texture generation. We collected our own point cloud data using Velodyne lidar and used google earth studio to collect images of the same route. We also used

OpenStreetMap data to get the road map, which we used to overlay over our environment for better results.

4.2.1 Pre-processing

In preprocessing, we mainly try to trim down than redundant point cloud data to improve the speed in the next steps. This is mainly to remove the dense point clouds of the object that were significantly closer to the lidar. Due to design, the point clouds collected near the lidar would be cluttered and points collected at a far distance would be far apart. Our objective is to remove those redundant points and to sample a reasonable amount of data cloud for the SLAM step. Note that when we do SLAM, point cloud from each scan would be interlaced therefore without preprocessing the generated point cloud would very dense (computationally expensive). For sampling, we used a Poisson sampling disk [22] approach to remove those highly cluttered point clouds. For our work, we removed almost 40-45% of lidar point cloud data. We have to mention that, the collected lidar scan would include point clouds of other moving objects such as vehicles, pedestrians, etc… Ideally, those point clouds have to be removed before the SLAM step. One can use existing state-of-art methods [21] to detect point cloud of other vehicles, cyclists, and pedestrians, and can remove from each lidar scan. One can skip this step if the collected data contains only the static objects. However, we noticed this only issue only in the later part, therefore, we manually removed those vehicles and pedestrians in the meshing step. The tools we primarily used for this step are Voxel and Poisson sampling disk, in particular, we used

MeshLab and Point Cloud Library. One can also skip this step and directly move to SLAM if preferred.

4.2.2 SLAM

In this step, we used the SLAM technique to interlace all the individual preprocessed lidar cloud to generate a single point cloud of the entire route. There are various ways to achieve this, however, we preferred NDT based SLAM implementation of Autoware for its ability to handle large data. Once the point cloud of the entire route was generated, we again tried to reduce the point cloud. This step is optional and can have an appreciable effect while finding vertices normal for mesh generation. For reducing the point cloud, we used the voxel sample approach.

4.2.3 Meshing

In this step, we used the meshing technique over the SLAM-ed point cloud to generate the 3D mesh. The main task for this step is to identify the appropriate vertex normals for our point cloud. Slight variation in this step could have a significant effect on the final mesh, therefore, one might have to heuristically select the parameter based on resolution and size, and might have to repeat this step for better results. Once the vertex normals are calculated, one can any surface meshing algorithm to construct the mesh. We found better surfaces with the ball-pivoting surface reconstruction algorithm [23]. The

parameters for ball-pivoting should be chosen based on point cloud density, size of the route/environment, and other similar factors. The generated mesh would have some faulty triangles and edges such as single point triangles, duplicate edge. These faulty triangles have to be removed else one might face issues when importing them in simulators. We use Meshlab for this step. Then, we used a quadratic edge decimation [24] technique to significantly reduce the triangle and edge count without much loss in the surface. To remove the spiky surface we used smoothing filters such as gaussian and laplacian filter for this. For this task, we used Meshlab [32], Blender, Point Cloud Library.

The generated mesh is a reasonably accurate replica of the environment as we have used lidar for the measurement. The next step is to add the color (photo-realisticness) to the mesh. For this, one can collect images using drones, however, we used google earth studio to collect the images of our environment. We nearly collected around 4,000 satellite image for this process. We acknowledge that the generated point cloud would be affected by the quality of google maps and occlusion due to trees and buildings.

In the upcoming part, we have only discussed the important step while skipped some intermediate steps such as feature extraction, point correspondences, and feature matching which can be handled by using Meshroom. We used Meshroom for this part, one might also use Pix4D, OpenMVG, VisualSFM, and other photogrammetry software to achieve the same result. The overall objective is to find the extract of the features from all

images and to match them to estimate the 3D structure of the environment. We think, this would be a good place to justify our goal. Although one might be able to generate the environment using only the images with SfM techniques, the generated environment would not be accurate (as our lidar mesh) and would be affected by occlusion and map quality. For this reason, we use the SfM technique with our images, only to incorporate photo-realisticness to our previous Lidar mesh.

4.2.4 Structure from Motions (SfM)

In this step, we used the matched feature from the previous step to generate a 3D point cloud (with color information). We used SfM libraries [25, 26, 27, 28, 29] from the meshroom to generate the point cloud. We heuristically identified the parameter needed for the SfM algorithm. One might also use other photogrammetry tools such as Pix4D, VisualSFM. Meshroom is used because of its cuda enabled libraries. One can mesh this point could to generate the environment if needed, however, it would have the previously mentioned drawbacks. To mention, we use this point cloud only to add photo-realisticness to our lidar mesh. To enhance texturing results, we created a dense point cloud using [28, 29, 30], which can be implemented using libraries from the meshroom. One can also further enhance the results by generating the Depth map. For this step, we primarily used Meshroom, Pix4D, and Blender.

4.2.5 Texturing

In this step, we added the textures from the SfM point cloud to the lidar mesh, using Least Square Conformal Maps [30]. We used meshroom's implementation of the Least Square Conformal Map algorithm and also used Blender to fix the scale and origin. The result of the final map is shown in figures 44 - 50. Note that images of the rendered environment shown below are from the SfM technique.

Figure 45: Rendering of Environment (a)

Figure 46: Rendering of Environment (b)

Figure 47: Rendering of Environment (c)

Figure 48: Rendering of Environment (d)

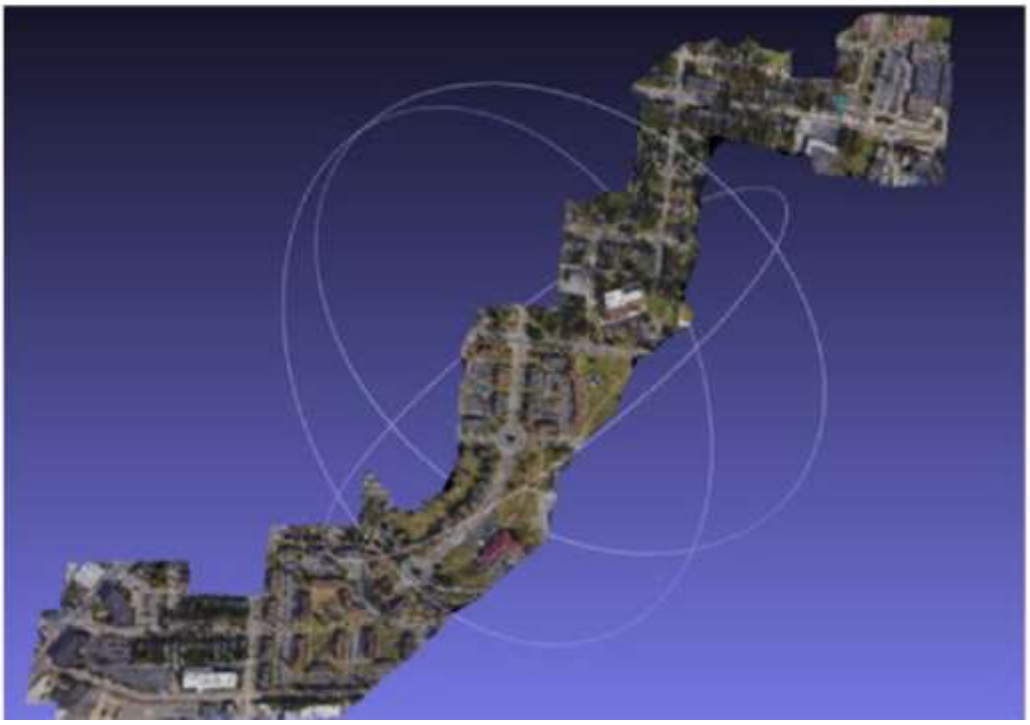


Figure 49: Rendering of Environment (e)

Figure 50: Lidar Mesh – Different place (a)

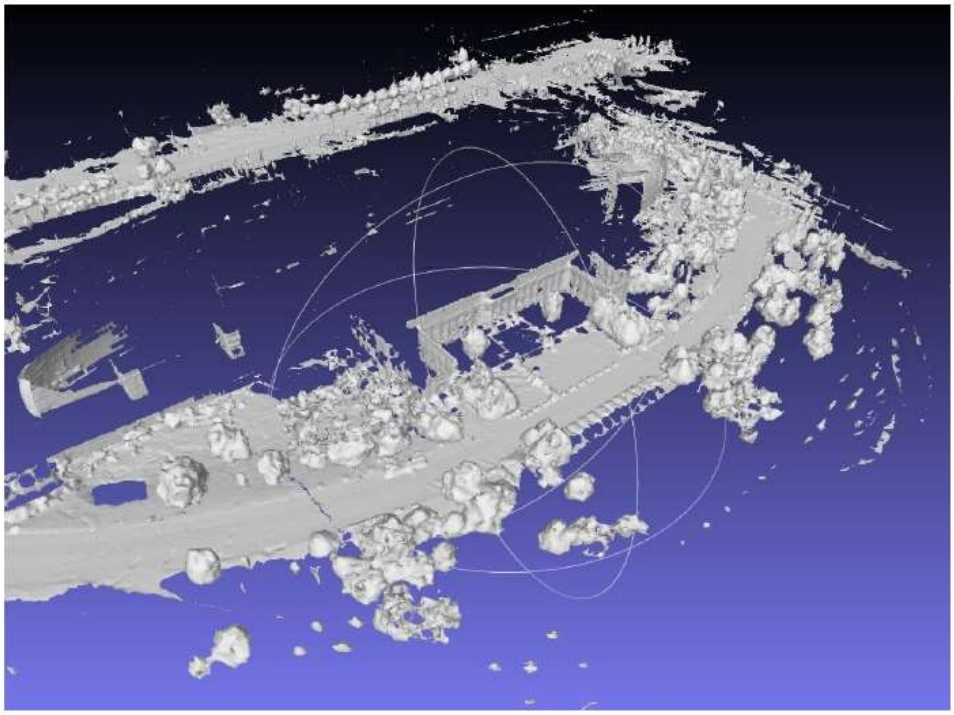


Figure 51: Lidar mesh (b)

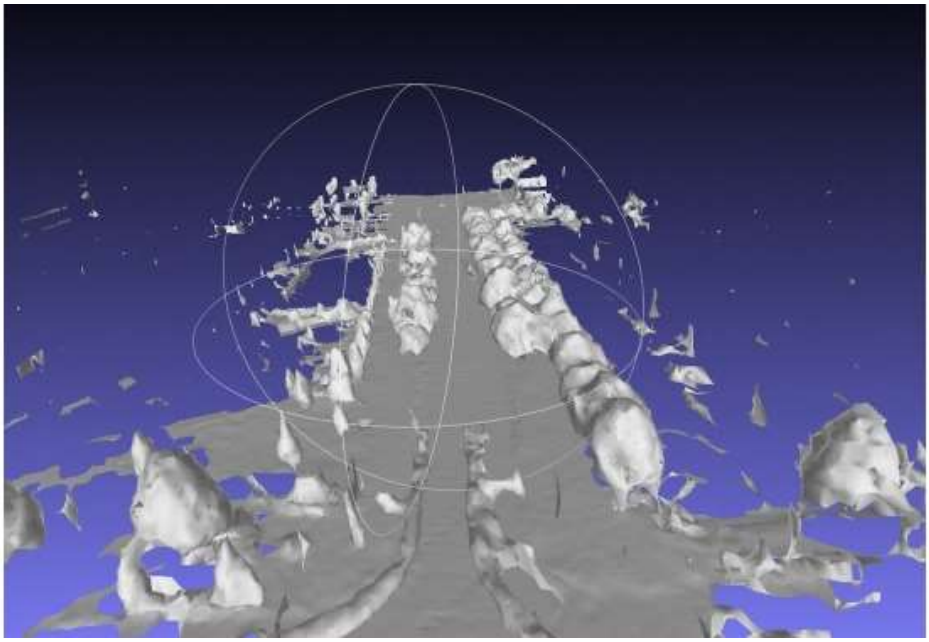


Figure 52: Lidar mesh (c)

Chapter 5. Results and Conclusions

In this chapter, we have briefly reviewed the overall work carried out and results that we got, and finally, concluded with the future work and scope.

In the first part of this thesis, we delve into addressing the autonomous driving problem. In chapter 1, we explained the current trends available to solve the autonomous driving problem along with their shortcomings, then emphasized the need to think the autonomy from the human-diving perceptive and proposed an alternative approach to address the problem. We also justified the reasoning of our approach with the simulation results. In Chapter 2 and Chapter 3, we discussed the architecture of our perception and control unit, then on training and evaluation methods. We also showed the simulation results for tasks such as lane-keeping, decision-taking at the intersection, obeying traffic lights, and avoiding collision with pedestrians at different simulation conditions.

Having said that, there are a lot of ways one can further improve this work. One can improve the agent to work at various traffic conditions and also various weather conditions. In current settings, our model would fail at dense traffic or at extreme weather conditions. One could also explore the recurrent base neural network and see whether it

could have an improvement in predicting desired waypoints for our ego by internally estimating the future locations of other vehicles and pedestrians. One could also explore the controls aspect of our problem. Since autonomy is a wide topic and can be seen and addressed in multiple directions. One could also explore a different spot for ‘mid’ in the ‘end-to-end’ pipeline and compare the results with the existing approaches.

In the second part of this thesis, we delve into generating environments for simulation and research purposes. In Chapter 4, we discussed the need for such simulation environments and explained the issues behind them. Then, we proposed a workflow that uses Lidar and google earth images to address the issue. Further, we focused on using only the open-sourced software and tools to achieve this. We also explained the step-by-step in-detailed procedure of our workflow. Finally, we showcased the rendered environment generated using our approach. Even here, there is a lot to improve. One can work on improving the speed of the overall process. One can incorporate state-of-art sensor fusion techniques to improve the final results.

Appendix - Image collection - Simulator

Below are the images of the top views of the town.

Figure 53: Town 1 – Top View (a) [41]

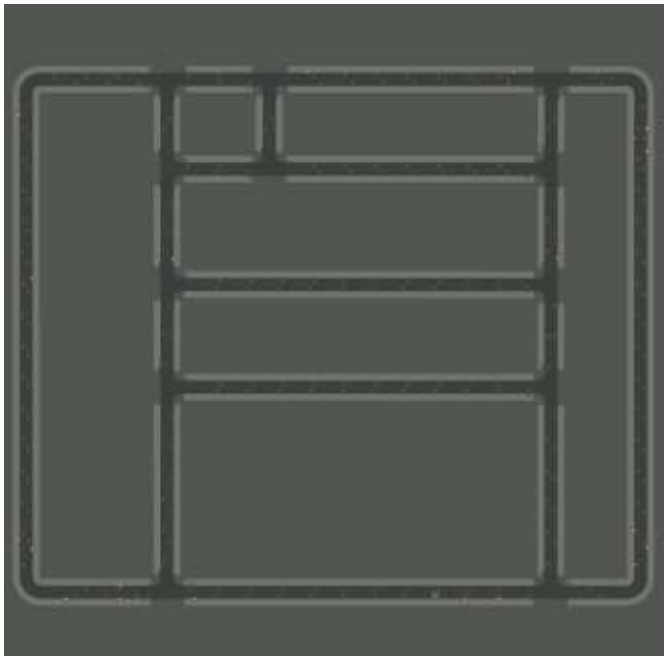


Figure 54: Town 1- Top view (b) [41]

Figure 55: Town 2 – Top View (a) [41]

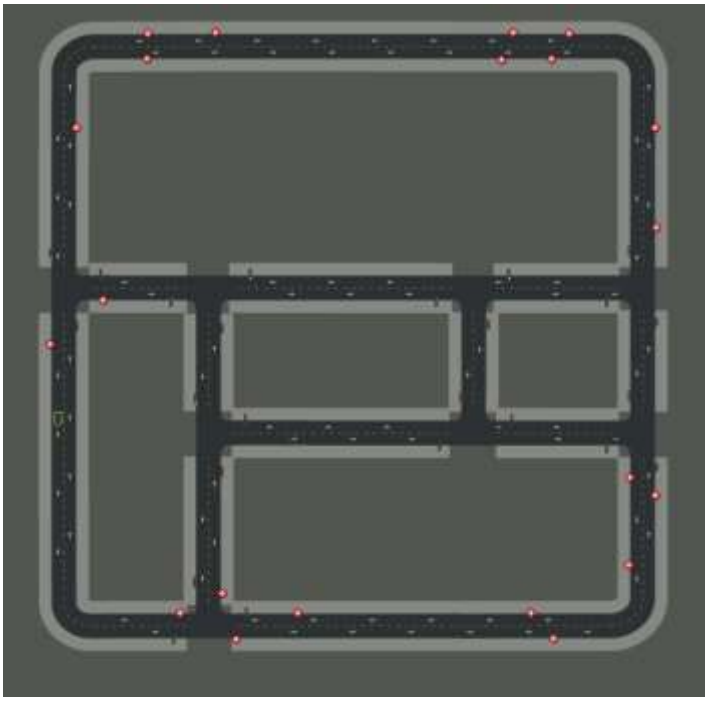


Figure 56: Town 2 – Top View (b) [41]

Bibliography

[1] M. Bansal, A. Krizhevsky, A. Ogale, “ChauffeurNet: Learning to Drive by Imitating the Best and Synthesizing the Worst”, June 2019, Robotics: Science and Systems XV, https://arxiv.org/abs/1812.03079.

[2] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. “CARLA: An open urban driving simulator”. In CoRL, 2017.

[3] S. Hecker, D. Dai, and L.V. Gool. “Learning driving models with a surround-view camera system and a route planner”, 2018, arXiv-preprint: arXiv:1803.10158,

[4] D. Chen and B. Zhou and V. Koltun and P. Krähenbühl, “Learning by Cheating”, In CoRL2019, arXiv:1912.12294, https://arxiv.org/abs/1912.12294.

[5] F. Codevilla, M. Muller, A. Lopez, V. Koltun, and A. Dosovitskiy. “End-to-end driving via conditional imitation learning”. In ICRA, 2018.

[6] F. Codevilla, E. Santana, A. Lopez, and A. Gaidon. “Exploring the limitations of behavior cloning for autonomous driving”. In ICCV, 2019.

[7] A. Sauer, N. Savinov, and A. Geiger. “Conditional affordance learning for driving in urban environments”. 2018, arXiv-preprint: arXiv:1806.06498.

[8] W. Sun, A. Venkatraman, G. J. Gordon, B. Boots, and J. A. Bagnell. “Deeply AggreVaTeD: Differentiable imitation learning for sequential prediction”. In ICML, 2017.

[9] K. He, X. Zhang, S. Ren, and J. Sun. “Deep residual learning for image recognition”. In CVPR, 2016.

[10] Y. Pan, C. A. Cheng, K. Saigol, K. Lee, X. Yan, E. A. Theodorou, and B. Boots. “Agile autonomous driving using end-to-end deep imitation learning”. In RSS, 2018.

[11] D. A. Pomerleau. “ALVINN: An autonomous land vehicle in a neural network”. In Neural Information Processing Systems, 1988.

[12] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, et al. “End to end learning for self-driving cars”. 2016, arXiv-preprint: arXiv:1604.07316.

[13] M. Bojarski, P. Yeres, A. Choromanska, K. Choromanski, B. Firner, L.D. Jackel, U. Muller, “Explaining how a deep neural network trained with end-to-end learning steers a car”. In CoRR, 2017.

[14] X. Chen, H. Ma, J. Wan, B. Li, T. Xia, “Multi-view 3d object detection network for autonomous driving”. In CVPR 2017.

[15] S. Chen, S. Zhang, J. Shang, B. Chen, N. Zheng, “Brain-inspired cognitive model with attention for self-driving cars”. 2017, arXiv-preprint: arXiv:1702.05596.

[16] S. Hecker, D. Dai, L. Van Gool, “Failure prediction for autonomous driving models”. In IEEE Intelligent Vehicles Symposium, 2018.

[17] S. Hecker, D. Dai, L. Van Gool, “Learning Accurate, Comfortable and Human-like Driving”. In IEEE Intelligent Vehicles Symposium, 2019.

[18] S. Tsugawa, "Vision-based vehicles in Japan: machine vision systems and driving control systems," in *IEEE Transactions on Industrial Electronics*, vol. 41, no. 4, pp. 398-405, Aug. 1994.

[19] Google Earth Studio, Google.inc (https://www.google.com/earth/studio/)

[20] OpenStreetMap. OSM (www.openstreetmap.org)

[21] B. Wu, A. Wan, X. Yue, K. Keutzer, “SqueezeSeg: Convolutional Neural Nets with Recurrent CRF for Real-Time Road-Object Segmentation from 3D LiDAR Point Cloud”, 2017.

[22] G. Ranzuglia, M. Callieri, M. Dellepiane, P. Cignoni, R. Scopigno, “Efficient and Flexible Sampling with Blue Noise Properties of Triangular Meshes,” IEEE Trans. on Visualization and Computer Graphics, Vol. 18, Num. 6, page 914--924, 2012

[23] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, G. Taubin, "The ball-pivoting algorithm for surface reconstruction," in IEEE Transactions on Visualization and Computer Graphics, vol. 5, no. 4, pp. 349-359, Oct.-Dec. 1999.doi: 10.1109/2945.817351

[24] M Tarini, N Pietroni, P Cignoni, D Panozzo, E Puppo, “Practical quad mesh simplification”, Computer Graphics Forum 29 (2), 407-418, 2010

[25] J. Cheng, C. Leng, J. Wu, H. Cui, H. Lu, “Fast and Accurate Image Matching with Cascade Hashing for 3D Reconstruction”. In CVPR 2014.

[26] P. Moulon, P. Monasse, R. Marlet, “Global Fusion of Relative Motions for Robust, Accurate and Scalable Structure from Motion”. In ICCV 2013

[27] R. Hartley and A. Zisserman, “Multiple view geometry in computer vision”, Cambridge, 2000.

[28] R. Toldo, R. Gherardi, M. Farenzena, A.Fusiello, “Hierarchical structure-and-motion recovery from uncalibrated images”. In CVIU 2015.

[29] H. Hirschmüller, “Accurate and efficient stereo processing by semi-global matching and mutual information”. In CVPR 2005.

[30] H. Hirschmüller, “Stereo processing by semi-global matching and mutual information”, 2008.

[31] B. Lévy, S. Petitjean, N. Ray, J. Maillot, “Least Squares Conformal Maps for Automatic Texture Atlas Generation“, Siggraph, 2002

[32] P. Moulon, P. Monasse, R. Marlet. “Adaptive Structure from Motion with a contrario model estimation”. In ACCV 2012

[33] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, G. Ranzuglia, “MeshLab: an Open-Source Mesh Processing Tool”, Sixth Eurographics Italian Chapter Conference, page 129-136, 2008.

[34] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, “Deepdriving: Learning affordance for direct perception in autonomous driving,” in Proceedings of the IEEE International Conference on Computer Vision, 2015, pp. 2722–2730.

[35] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, V. Pratt, et al., “Towards fully autonomous driving: Systems and

algorithms," in Intelligent Vehicles Symposium (IV), 2011 IEEE. IEEE, 2011, pp. 163–168.

[36] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer, et al., "Autonomous driving in urban environments: Boss and the urban challenge," Journal of Field Robotics, vol. 25, no. 8, pp. 425–466, 2008

[37] Car POV image https://www.pinterest.com/pin/417497827929315599/

[38] Map image https://www.devims.com/blog/google-maps-navigation-and-traffic-data-now-in-india/

[39] Carla-simulator-image (carla.org)

[40] H. Ergun, Y. C. Akyuz, M. Sert, J. Liu, "Early and Late Level Fusion of Deep Convolutional Neural Networks for Visual Concept Recognition", 2016, International Journal of Semantic Computing, 2016, Vol.10, No. 03, pp.379-397.

[41] Town Images, Carla Simulator - docs - data collector (https://github.com/carla-simulator/)

[42] X. Li, A.C. Arul Doss, B. Aksun Guvenc, L. Guvenc, "Pre-Deployment Testing of Low Speed, Urban Road Autonomous Driving in a Simulated Environment," SAE Technical Paper 2020-01-0706, 2020, doi:10.4271/2020-01-0706