# Language Conditioned Self-Driving Cars

Using Environmental Object Descriptions For Controlling Cars

by

Nithish Balachandar Moudhgalya

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

ARIZONA STATE UNIVERSITY

May 2021

## ABSTRACT

Self-Driving cars are a long-lasting ambition for many AI scientists and engineers. In the last decade alone, many self-driving cars like Google Waymo, Tesla Autopilot, Uber, etc. have been roaming the streets of many cities. As a rapidly expanding field, researchers all over the world are attempting to develop more safe and efficient AI agents that can navigate through our cities. However, driving is a very complex task to master even for a human, let alone the challenges in developing robots to do the same. It requires attention and inputs from the surroundings of the car, and it is nearly impossible for us to program all the possible factors affecting this complex task. As a solution, imitation learning was introduced, wherein the agents learn a policy, mapping the observations to the actions through demonstrations given by humans. Through imitation learning, one could easily teach self-driving cars the expected behavior in many scenarios. Despite their autonomous nature, it is undeniable that humans play a vital role in the development and execution of safe and trustworthy self-driving cars and hence form the strongest link in this application of Human-Robot Interaction. Several approaches were taken to incorporate this link between humans and self-driving cars, one of which involves the communication of human's navigational instruction to self-driving cars. The communicative channel provides humans with control over the agent's decisions as well as the ability to guide them in real-time. In this work, the abilities of imitation learning in creating a self-driving agent that can follow natural language instructions given by humans based on environmental objects' descriptions were explored. The proposed model architecture is capable of handling latent temporal context in these instructions thus making the agent capable of taking multiple decisions along its course. The work shows promising results that push the boundaries of natural language instructions and their complexities in navigating self-driving cars through towns.

TABLE OF CONTENTS

# LIST OF TABLES

LIST OF FIGURES

FIGURE Page

# Chapter 1

# INTRODUCTION

Starting from mid 20th century, the field of Artificial Intelligence (AI) has diversified applications in today's world. Its impact and development have increased exponentially in the last few decades alone. The goal of AI has always been to perform tasks logically and rationally in an attempt to make the job of humans easier. One particular branch of AI, namely Machine Learning (ML) took off during 1952, with the introduction of alpha-beta pruning algorithm for playing the game of ”Checkers”. Following this, inspired by the biological structure of the human brain, ”perceptrons” were introduced as a mathematical model that encompasses the functionality of a single neuron. In the 1970s, this branch separated out from AI as an independent field of work that depends on statistics and probability theory in its algorithms. The goal shifted from creating AI that mimics human intelligence to perform logical tasks to being a learning tool for developing AI. Over a decade of hard work and efforts brought the brainchild of today's ML research - ”Deep Learning”, and the idea wherein stacked layers of perceptron networks seemingly provide better results as opposed to conventional algorithms.

Humans have always been observed as one of the most complex beings on this planet. Over millions of years, with evolution and natural selection, we have developed the skills and biological structures that help us perform complex tasks with ease. Furthermore, human intelligence and physical dexterity have provided us with the ability to perform tasks that require coordination between several components in our body - limbs, eyes, ears, etc. Humans learn over several years to master different skills and tasks like singing, dancing, oration, narration, driving, arts, and science, etc. Out of these, driving is considered one of the toughest skills to master for many humans owing to the complexity of its nature. Driving requires not just our visual attention on the road, but also towards cars beside and behind us on the streets, signs on the sides, traffic rules, speed variations, and in-car attributes like indicators and wipers. On top of these, there also several scenarios that arise while driving like brake checking, tailgating, jaywalking, traffic signal malfunctions, uncharted roadworks, change in weather (rain or snow), etc. that could affect the navigation and well as driving decisions.

AI has several applications that it has become an inseparable asset of our daily lives, such as search engines, email spam filters, e-commerce recommendations, social networking, navigational aids, and some complex applications like autopilot in flights, self-driving cars, market prediction, etc. Self-driving cars have fascinated AI researchers and enthusiasts for several decades now. The basic idea behind them can be stated as the ability of an AI agent or model to use information from its surroundings and an onboard computer to process the information and steer through the roads whilst navigating to its destination. The simplicity of the task description eludes the latent complexity of this task, which makes it potentially hard even for many humans. The challenges in developing a fully functional self-driving car include identifying objects around the car, determining the object trajectories for those in motion, multi-modal data stream input, and finally, the onboard computer that needs to process all these quickly and take decisions.

According to the Society of Automotive Engineers (SAE), self-driving cars have 6 sub-levels. Level 0: The automated system issues warnings and may momentarily intervene but has no sustained vehicle control. Level 1 (”hands-on”): The driver and the automated system share control of the vehicle. Examples include Cruise Control, Adaptive Cruise Control, and Parking Assistance. The driver must be ready to retake full control at any time. Level 2 ("hands-off"): The automated system takes full control of the vehicle: accelerating, braking, and steering. The driver must monitor the driving and be prepared to intervene immediately at any time if the automated system fails to respond properly. Level 3 ("eyes off"): The driver can safely turn their attention away from the driving tasks. The vehicle will handle situations that call for an immediate response, like emergency braking. The driver must still be prepared to intervene when called upon by the vehicle to do so. Level 4 ("mind off"): No driver attention is ever required for safety, for instance, the driver may safely go to sleep or leave the driver's seat. However, self-driving is geo-fenced or only under certain circumstances. Outside of that, the vehicle must be able to safely abort the trip. Level 5 ("steering wheel optional"): No human intervention is required at all. Today, Waymo and Tesla, the leaders in the development of self-driving cars, are at Level 2, which shows how far along are we in developing the true AI agent that can cruise through the streets without human intervention.

Driving is a real-world problem that required loads of data to learn the basic rules and policies of driving on the roads. Even if we develop a system that can tackle these tasks, there is always a concern about its safety or reliability. No matter what the developments in this field are, the potential ability for humans to be able to communicate with self-driving cars to guide them or instruct them has always been a hot research focus. Interactive self-driving cars provide a reliable sensation since humans can instruct them on what has to be done. Such AI agents can use this communicative channel to take sudden decisions that could involve a simple pulling over to the side of a highway to driving with low visibility in snowstorms. In the last few years alone, researchers have developed means to enable humans the opportunity to control what self-driving cars need to do.

# Chapter 2

# BACKGROUND INFORMATION

This chapter presents work and literature regarding imitation and conditional imitation learning. The chapter introduces imitation learning and its applications, followed by the pitfalls or challenges faced by it, and lastly explores conditional imitation learning and its literature as a solution.

## 2.1 Imitation Learning

For years, the robotics field has drawn inspiration from designs and executions from humans and other animals. The inexplicable beauty of nature has baffled them and inspired them in creating complex robots that are capable of performing several tasks. Researchers have spent years trying to develop AI agents that can mimic the actions of biological beings and execute their tasks with equal capabilities. The decision and behavior of biological and intelligent biological beings (experts) are affected by various factors both external and internal to them. It is nearly impossible for AI researchers to encode all those factors into the decision and control systems of AI agents since these factors may be latent as well. Furthermore, the advancement in robotics has enabled the possibility to train AI agents that can perform tasks that require motor skills as well. Defining the protocols and algorithms to train robots to perform such tasks is nearly impossible.

Humans have themselves been products of imitation. From their birth, they observe the actions around them and try to reenact those actions to their best ability. Humans have had millions of years of natural evolution to perfect their mechanical structure and cognitive systems that make learning new actions far easier than for a robot. The biological structure of our body and brain gives us an inherent advantage in being able to perform a complex task that involves the co-operation of several components of our system - vision, auditory, motor skills, logical and rational thinking, etc. The number of scenarios that could arise in a complex application is simply too large to estimate and account for while creating AI agents. In such cases, having some demonstration from experts could be seen as gaining prior knowledge of the application. The evidence all around us supports the undisputed efficacy of imitation as a learning procedure and hence gave rise to the simplest yet promising ideas in AI agents today - Imitation Learning.

It is much easier for a human teacher or expert to transfer their knowledge through demonstrations than to articulate it in a way that the learner will understand [Raza *et al.* (2012)]. With inspiration and basis stemmed in neuroscience, imitation learning is an important part of machine intelligence and human-computer interaction and has from an early point been viewed as an integral part of the future of robotics [Schaal (1999)]. The AI model learns from the demonstrations of the expert but when subjected to any situation, it can approach to solve a quantifiable goal in a manner the human may or may not. To allow such solutions, imitation learning should provide the learner opportunity to explore as well and exploit the demonstration. In many scenarios, the robot actions are required to be natural and human-like, such as picking up a teapot or cup, self-driving cars, etc. In these cases, the AI model needs to not just imitate the expert but learn a policy that can help it solve new situations as well.

To define the process of imitation learning mathematically, lets refer to the state of system at any given instant of time $t$ as $x_t$ and the actions taken by the expert as $a_t$. Any demonstration $D_i$ used to train imitation learning models can be represented as shown in Equation (2.1).

$$D_i = \{(x_0, a_0), (x_1, s_1)....(x_t, a_t)....(x_n, a_n)\} \tag{2.1}$$

The goal is to use the demonstration data to learn the mapping between the states and their actions, or in other words the policy $\pi$. Equation (2.1) depicts the policy learned by the model. Figure 2.1 depicts the mapping process pictorially.

$$a_t = \pi(x_t) \tag{2.2}$$

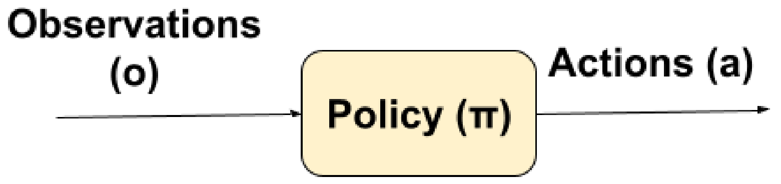


Figure 2.1: Imitation Learning

Imitation learning has been used to teach robots of different degrees of freedom (DOF) and skeletons to perform a wide range of tasks. Some examples are navigational problems, which typically employ vehicle-like robots, with relatively lower degrees of freedom. These include flying vehicles [Sammut *et al.* (1992); Abbeel *et al.* (2007); Ng *et al.* (2006)], or ground vehicles [Silver *et al.* (2008); Ratliff *et al.* (2007); Chernova and Veloso (2008); Ollis *et al.* (2007)]. Other applications focus on robots with higher degrees of freedom such as humanoid robots [Mataric (2000); Asfour *et al.* (2008); Calinon and Billard (2007)] or robotic arms [Kober and Peters (2010, 2014); Mülling *et al.* (2013)]. High DOF humanoid robots can learn discrete actions such as standing up, and cyclic tasks such as walking [Berger *et al.* (2008)]. Imitation learning extracts information about the behavior of experts and the surroundings and learns a mapping between the observations and demonstrated expert behavior.

Traditional machine learning algorithms do not scale to high dimensional agents with high degrees of freedom [Kober and Peters (2010)].

## 2.2 Conditional Imitation Learning

Imitation Learning for complex tasks like self-driving cars is faced with many challenges. First, complex behaviors can be seen as a trajectory of dependent micro-actions which violates the independent and identically distributed (i.i.d.) assumption that most machine learning practices employ. Hence the learned policy must be able to adapt its behavior based on previous actions and make corrections if necessary[Hussein *et al.* (2017)]. Second, it is essential that the learned policy must be able to adapt to variations in the task and environment. The complex nature of self-driving as an application dictates that agents must be able to reliably perform the task even under varied circumstances. Lastly, tasks that entail human-robot interaction adds safety as a chief concern in self-driving cars [De Santis *et al.* (2008); Ikemoto *et al.* (2012)].

The task of self-driving cars can be subdivided into two prime components - the navigation planner and the controller. The controller is responsible for driving the car by manipulating its control parameters such as throttle, steering, and brakes. The controller specializes in taking the environment state & observations and predicting the appropriate values for these control parameters based on its policy learned from expert demonstrations. The navigation planner is responsible for planning the route that the agent has to follow in order to complete a high-level goal. Simple imitation learning fails to combine these two components and even if it does, the agents face the challenges mentioned above. Hence, conditional imitation learning was introduced to not only teach the agent the mapping between observations and actions but also train them to adapt according to t some conditional signal or factor. In other words, the conditional signal could work as a piece of additional information that the agent can use to adapt to various situations [Codevilla *et al.* (2018)].

The difference between conditional imitation learning and simple imitation learning can be easily understood by looking at the equations. In conditional imitation learning, the demonstrations are comprised of additional conditions or variables $c_t$ that affect the actions taken by expert at any instant of time as shown in Equation (2.3).

$$D_i = \{(x_0, c_0, a_0), (x_1, c_1, a_1), ..., (x_t, c_t, a_t), ...(x_n, c_n, a_n)\} \tag{2.3}$$

The goal is the same as before, to learn a policy $\pi$, however the difference is in the fact that the learned policy depends on not just the state but also on the conditional variable as shown in Equation (2.4). The provision of conditional signals into the policy makes it easier to adapt the models to different tasks and scenarios. Figure 2.2 depicts the mapping process pictorially.

$$a_t = \pi(x_t, c_t) \tag{2.4}$$

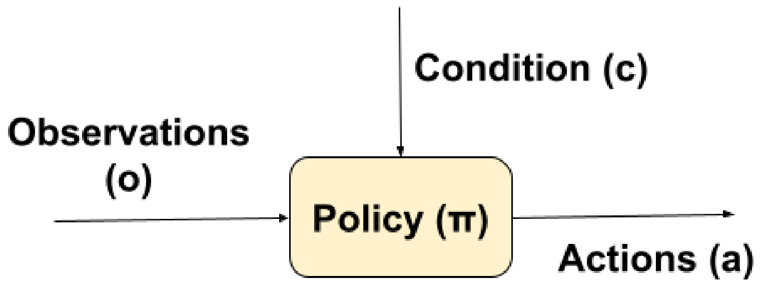


Figure 2.2: Conditional Imitation Learning

The agent is trained to learn a policy from expert demonstrations as before but this time it has to learn the policy conditioned upon a command or control signal.

The inclusion of conditions in ¡state, action¿ pair helps develop an agent that can control the car efficiently whilst following the command. Thus given any navigational planner, once we get a high-level command, the agent can learn to follow the command and control the car. The introduction of conditional imitation learning opened up new avenues of research in self-driving cars. However, this approach has its own demerits. The conditions provided to the agent need to be concise and precise. The navigational planner can thus only provide simple intimations of what actions to take such as - "left", "right" or "straight".

To enhance the applicability of conditional imitation learning for self-driving cars, the gap between simple commands and natural language instructions like the ones we provide to human driver needs to be bridged [Liang *et al.* (2018); Roh *et al.* (2020)]. The scope of such a system would be unlimited as it provides a way to convert the high-level complex commands of the navigational planner to control signals that the controller network uses to drive the cars. The use of natural language instructions makes it effective in integrating real-life navigation systems such as GPS and Google Maps into self-driving cars and thus enabling a new era of conversational and interactive cars. In this work, the possibilities of using natural language statements in providing high-level commands to the car that it can safely follow and complete the route planned by the user are explored.

Chapter 3

# CARLA SIMULATOR

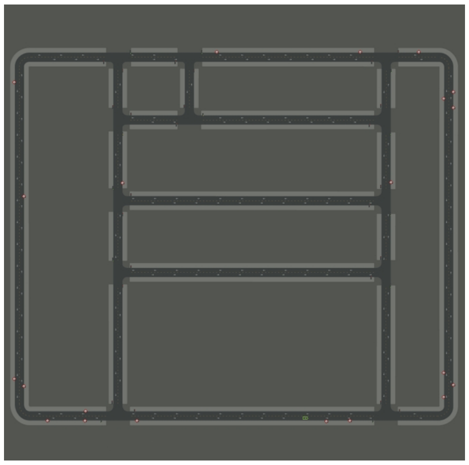
CARLA is an open-source simulator for autonomous driving research, developed from the ground up to support the development, training, and validation of autonomous urban driving systems [Dosovitskiy *et al.* (2017)]. Being an open-source simulator, it provides access to not just its codes and protocols but also to several assets like buildings, layouts, and vehicles. The simulation platform supports flexible specifications of sensor suites and environmental conditions. CARLA can be used to study the performance of three approaches to autonomous driving: a classic modular pipeline, an end-to-end model trained via imitation learning, and an end-to-end model trained via reinforcement learning.

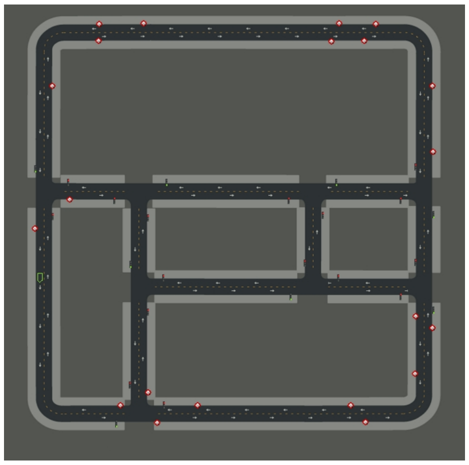
## 3.1 Towns and Weather

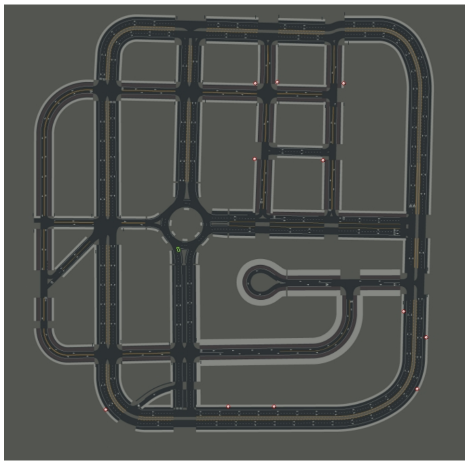
CARLA provides over 8 different environment setups as towns. Each town has its own properties and purpose, giving self-driving cars an insight into different scenarios. Figure 3.1 depicts the structural layouts of all these towns.

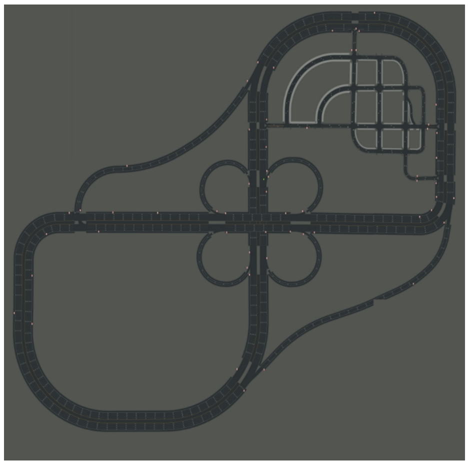
- Town01 - A basic town layout with all ”T junctions”.
- Town02 - Similar to Town01, but smaller.
- Town03 - The most complex town, with a 5-lane junction, a roundabout, unevenness, a tunnel, and much more.
- Town04 - An infinite loop with a highway and a small town.

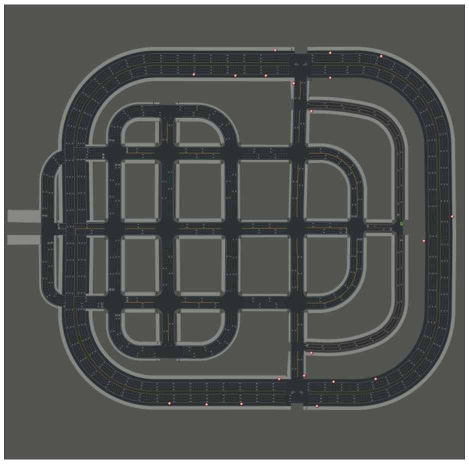
- Town05 - Squared-grid town with cross junctions and a bridge. It has multiple lanes per direction. Useful to perform lane changes.
- Town06 - Long highways with many highway entrances and exits.
- Town07 - A rural environment with narrow roads and barns.
- Town10 - A city with different environments such as an avenue or a promenade, and highly realistic textures.

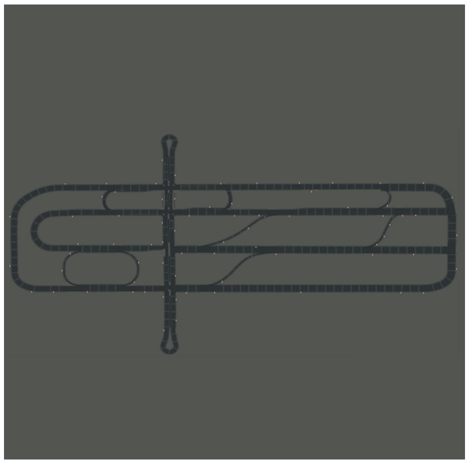

(a) Town01

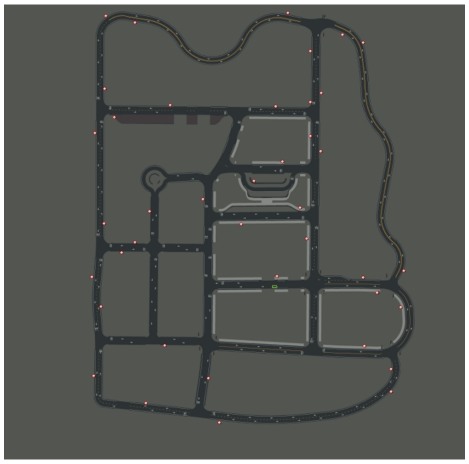

(b) Town02


(c) Town03


(d) Town04


(e) Town05


(f) Town06


(g) Town07

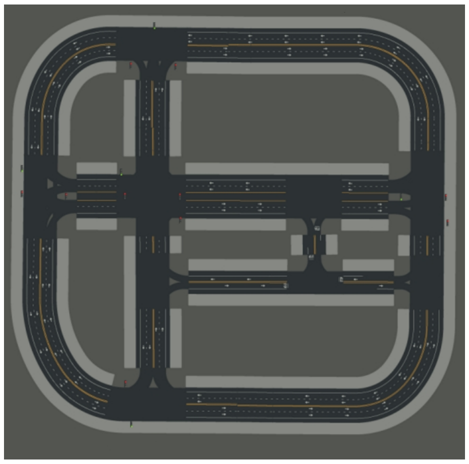
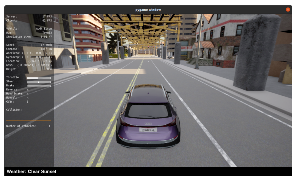
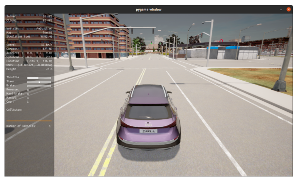

(h) Town10

Figure 3.1: CARLA Towns

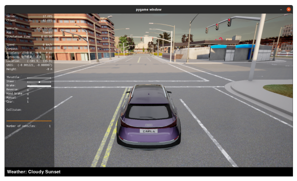
CARLA provides around 15 preset weather conditions that can be used off the self by researchers to change the perception of surroundings. This range covers noon setups with clear/cloudy/wet/rainy conditions and sunsets with clear/cloudy/wet/rainy. CARLA also provides a way to modify the weather conditions in the simulated environment through client APIs to control not just the sun's altitude and speed, but also precipitation percentage, cloud cover, etc. This way users can take complete control over the conditions in which they prefer to collect data, train their models and test the generalization of their agents. Figure 3.2 shows some of these weather conditions available in CARLA in its latest version.



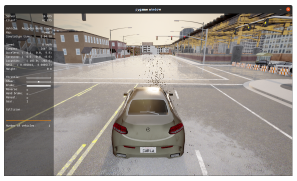
(a) Clear Noon



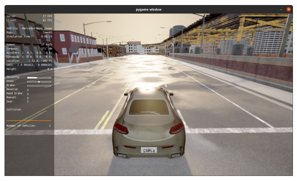
(b) Cloudy Noon



(c) Cloudy Sunset



(d) Hard Rainy Sunset



(e) Soft Rainy Sunset

Figure 3.2: CARLA Weather Setups

## 3.2 Sensors

CARLA also provides a plethora of sensors that can be spawned with self-driving cars. The sensors provide an easy interface to stream data to the client in various formats that can be processed and used for training models. In most self-driving cars, the goal is to use cameras and LIDAR alone to take control of decisions. Some of these details are explained below.

### 3.2.1 RGB Camera

The "RGB" camera acts as a regular camera capturing images from the scene. Several effects can be applied to mimic realism like Vignette(darkens the border of the screen), Grain jitter(adds some noise to the render), Bloom(intense lights burn the area around them), Auto exposure(modifies the image gamma to simulate the eye adaptation to darker or brighter areas) or Lens flares(simulates the reflection of bright objects on the lens). Some sample images captured using this sensor are shown in Figure 3.3.

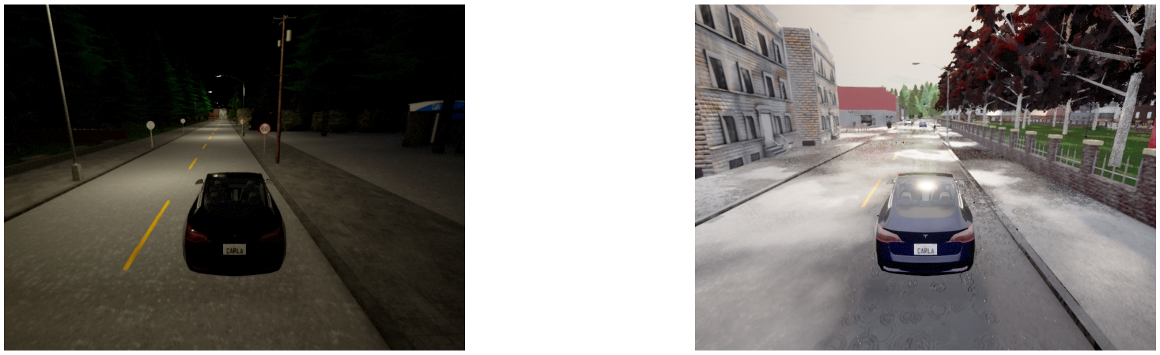


Figure 3.3: RGB Images

### 3.2.2 Semantic Segmentation Camera

This camera classifies every object in sight by displaying it in a different color according to its tags (e.g., pedestrians in a different color than vehicles). When the simulation starts, every element in the scene is created with a tag. There are a total of 25 tags in CARLA today that cover, buildings, trees, roads, traffic signs, poles, and much more distinguishable segments of the image. Some sample images captured using this sensor are shown in Figure 3.4.

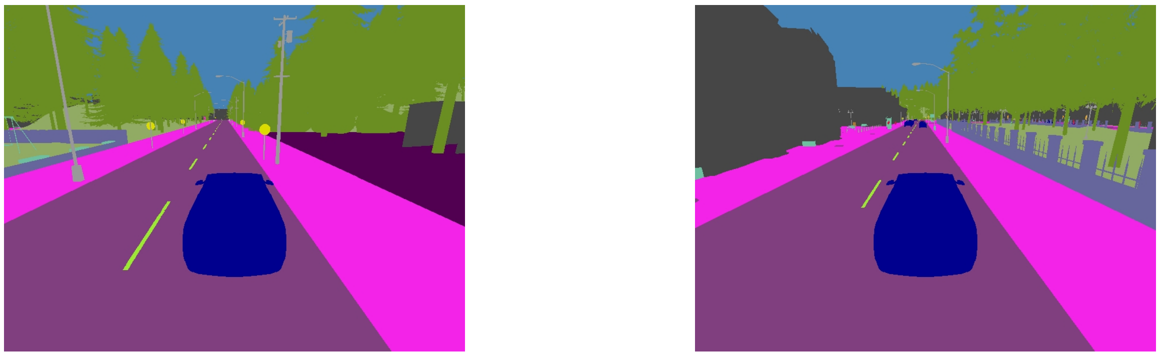


Figure 3.4: Semantic Segmentation Images

### *3.2.3 Depth Camera*

The camera provides raw data of the scene codifying the distance of each pixel to the camera (also known as depth buffer or z-buffer) to create a depth map of the elements. The internal system in CARLA converts the distances into the range [0, 1] and codifies this image as grayscale. Some sample images captured using this sensor are shown in Figure 3.5.

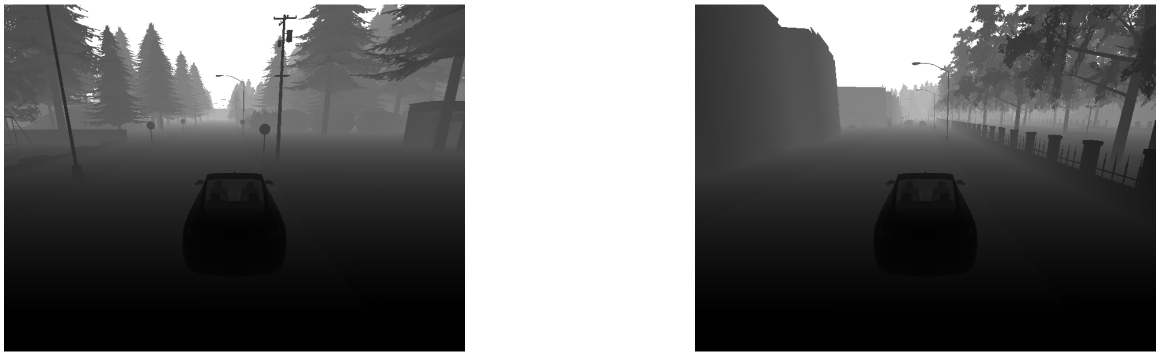


Figure 3.5: Depth Images

### *3.2.4 LIDAR Sensor*

A LIDAR measurement contains a package with all the points generated during a 1/FPS interval. During this interval, the physics are not updated so all the points in a measurement reflect the same ”static picture” of the scene. A sample image captured using this sensor is shown in Figure 3.6.

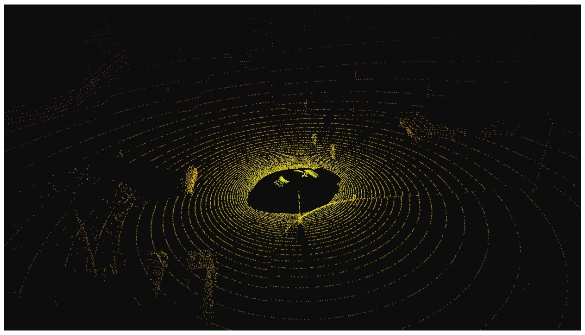


Figure 3.6: LIDAR Image

### *3.2.5 GNSS Sensor*

This sensor reports the current GNSS position of the vehicle. This is calculated by adding the metric position to an initial geo reference location defined within the OpenDRIVE map definition. The GNSS position is used by the servers to calculate the trajectory waypoints for the autopilot.

### *3.2.6 IMU Sensor*

Provides measures that accelerometer, gyroscope, and compass would retrieve for the car. The data is collected from the car’s current state. These readings give an idea of the acceleration of the vehicle along the axes and its angle of tilt helps determine the physics of the wheels.

## 3.3 CARLA Traffic Manager and Autopilot

CARLA provides a server-side engine called the Traffic Manager (TM). The role of the TM is to calculate the future trajectories of every object subscribed to it in the simulation and implement them. In other words, TM is responsible for giving life to all objects in the simulation. The TM runs an indefinite loop that does the following 3 steps repeatedly - storing the current state of the simulation, calculating movement for every registered actor, and applying commands to all actors at once. The loop takes care of updating the motion of the registered actors based on the environment state. Figure 3.7 shows a detailed architecture diagram of the TM.

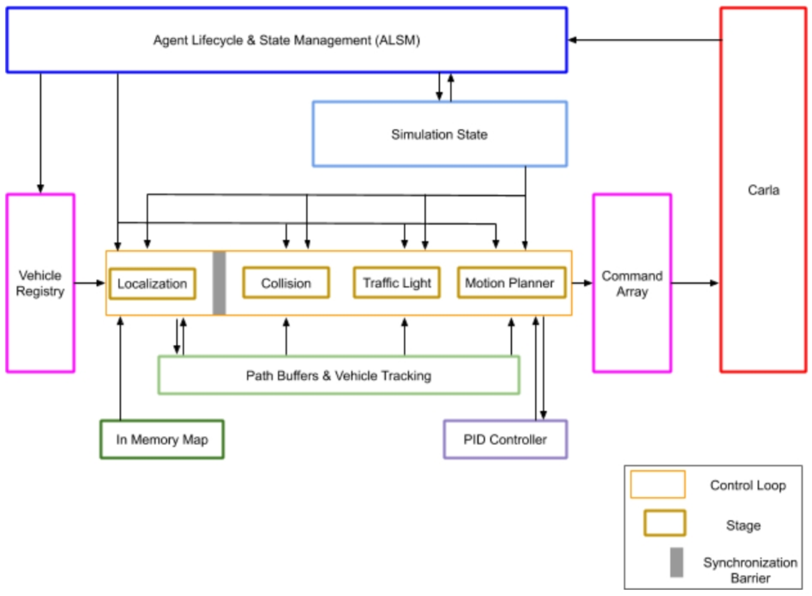


Figure 3.7: Traffic Manager Architecture

The server-side autopilot is a part of the TM in CARLA. The vehicle once registered to the TM, is controlled by it to navigate endlessly through the town. The only disadvantage of the TM autopilot is that it cannot be controlled by the client to execute pre-determined routes.

# Chapter 4

# END-TO-END DRIVING USING HIGH-LEVEL COMMANDS

Two progressive experiments were conducted, in an effort to control self-driving cars with high-level commands. Details of the first experiment are discussed in this chapter in-depth. Section 4.1 discusses the motivation behind this approach, followed by Section 4.2 that describes the new dataset. Section 4.3 illustrates the model architecture designed to learn the policy using conditional imitation learning whose results are compared and explored in Section 4.4. Finally, before moving to the next step, Section 4.5 discusses the limitations of this approach.

## 4.1 Motivation

Conditional Imitation Learning provided a way of factoring the user's objective into the model that can affect its control predictions. However, previous works only use a single command signal indicating the prime objective of the user's intent [Codevilla *et al.* (2018)]. The self-driving cars could learn to navigate the city, following the user's objective but had several restrictions in its implementation. The architecture in Codevilla *et al.* (2018) creates identical branches for every prime command signal, thus creating additional parameters and making it non-scalable in the future. Furthermore, the user instruction needs to be a single command signal that is supplied at every instant based on the current position of the car with respect to the goal location.

In this approach, the first issue is addressed by creating a new architecture that can process the observations along with the commands using a single branch itself. To tackle the second issue, high-level natural language instructions were added to the input and NLP methods were employed to embed this information which can be used in conjunction with the observations to control the car. To train this architecture, a new dataset had to be constructed and the inference bed modified to provide higher-level commands instead of a single command signal based on referential position vectors of the destination and current car position.

## 4.2 Dataset

The dataset plays a vital role in any ML task but it has one of the most significant impacts in imitation learning since the dataset is comprised of expert demonstrations. The quality of the dataset determines the quality of the policy learned by the agent as well. In this experiment, the dataset was created using only Town02 from CARLA. This town is usually used by all previous works to test the efficiency of the agents in navigating through the unseen cities. Furthermore, being a smaller town, the dataset could be collected much faster and have a stratified distribution among the different demonstrations and scenarios. The expert model used in this setup was the server-side autopilot that comes inbuilt with CARLA TM Figure 3.7.

To collect the dataset, a user agent is spawned amidst the town at a random position with the RGB Camera, Semantic Segmentation Camera, Depth Camera, GNSS Sensor & IMU Sensor (refer Section 3.2) and is registered to the TM in CARLA to run in autopilot mode. The various components of this dataset are listed in Table 4.1. Apart from the data collected from these sensors, the control parameters (throttle, speed, and brake) used by the autopilot to drive the car are also recorded and stored along with the current speed of the car.

| | |
|---|---|
| RGB Image | Brake |
| Semantic Segmntation Image | Accelerometer Readings |
| Depth Image | Gyroscope Readings |
| Speed | Compass Readings |
| Steer | Traffic Signal State |
| Location | GNSS Readings |
| High Level Command | Throttle |

Table 4.1: Dataset 1 Components

Table 4.2 lists the various possible phrases that are merged together to construct the high-level navigational natural language instructions for the self-driving cars in this dataset.

| **Prefix/Suffix** | **LEFT** | **RIGHT** | **STRAIGHT** |
|---|---|---|---|
| [ at the next intersection, from the next intersection, at the next junction, from the next junction, at the light, from the light ] | [ take a left, turn left, make a left ] | [ take a right, turn right, make a right ] | [ keep going straight, drive straight, head straight ] |

Table 4.2: High Command Segments for Dataset 1

The agent, once spawned, keeps roaming the town continuously and the data frames are collected at an FPS rate of 10. However, not all these frames are useful demonstrations to train natural language instructions, and hence every time the agent makes a decision at a junction or road ends, the episode is saved comprising of 150 frames in total with padding on both ends to provide the context of the decisions taken by the autopilot. Each episode is tagged with simple natural language instruction. Some of the high-level commands used in this dataset are shown in Table 4.3.

| **STRAIGHT** | **LEFT** | **RIGHT** |
|---|---|---|
| keep going straight from the next junction | take a left at the next intersection | at the light, make a right |
| drive straight along this lane | take a left at the next light | turn right from the next junction |
| head straight on this street | when you reach the next junction, turn left | from the next intersection, take the right |

Table 4.3: Dataset 1 High Command Examples

## 4.3 Model Architectures

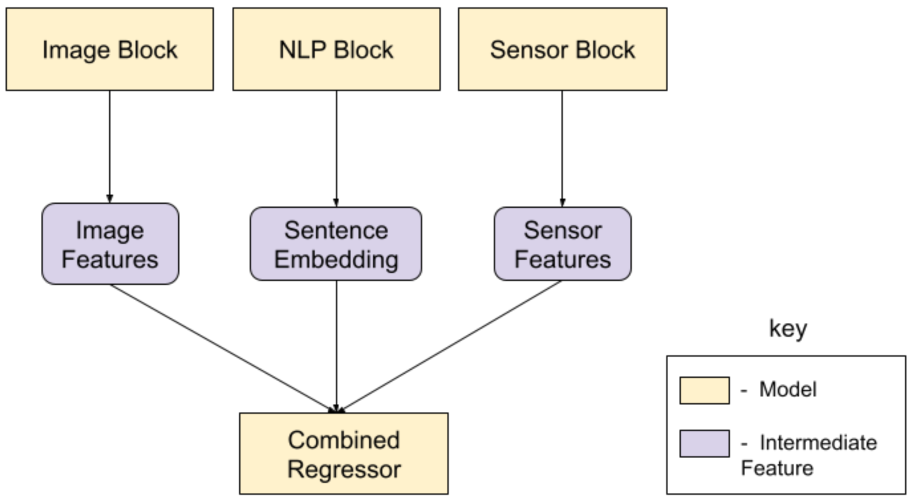


Figure 4.1: ModelV1 Abstract Architecture

The dataset collected above requires the need to modify the architecture proposed in Codevilla *et al.* (2018) to adapt to high-level instructions. The model architecture proposed in this approach (henceforth referred to as ModelV1), has 4 components in total - 3 main blocks that process different observatory variables and one that combines the outputs from all these blocks and makes predictions of control values. The overall architecture of this model is shown in Figure 4.1.

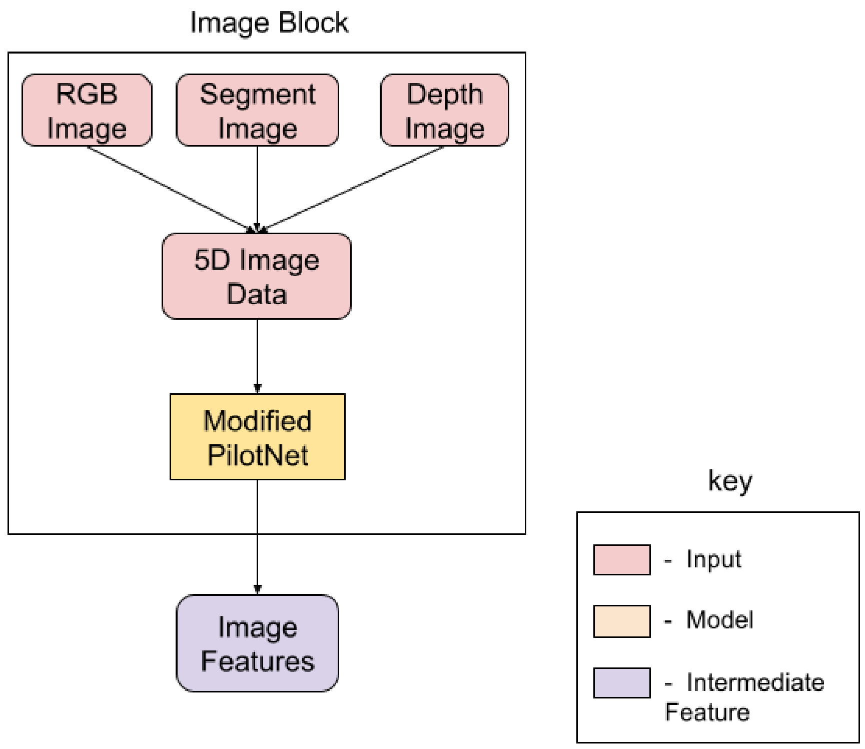


Figure 4.2: ModelV1 Image Block Architecture

**Image Block -** This is the first and most vital block of ModelV1. In any self-driving car, the current state of the environment is captured mostly through cameras and LIDAR sensors. In these experiments, the state is represented by the images that can be observed at any given instant of time in the simulation. The image block is responsible for taking the image state of the system and extract features from it that can be used in lower layers of the model to learn the expert policy. The architecture of this block is inspired by PilotNet [Bojarski *et al.* (2016)], with a small change to its initial structure. To encompass the details from semantic segmentation and depth measures, the tags and depth values are extracted and added as additional channels to the original RGB image thus rendering a 5-dimensional tensor as the input to this block. These 5D images, once processed by the Image Block, are embedded into a vector of size 512.

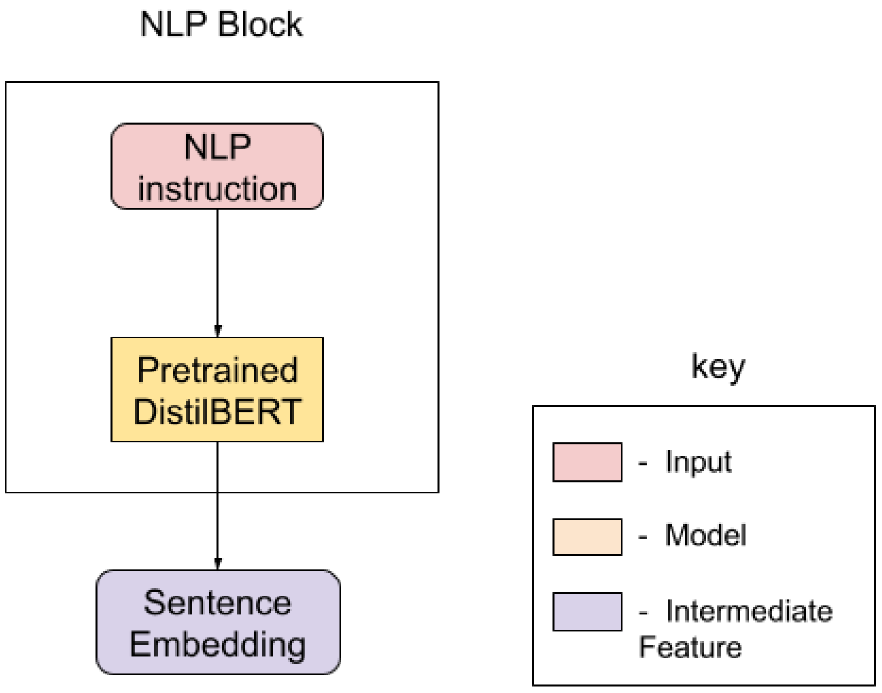


Figure 4.3: ModelV1 NLP Block Architecture

**NLP Block -** The second most important component of ModelV1 is the NLP block. It is one of the main improvisations over the existing works allowing the self-driving cars to follow natural language commands rather than just a signal. The NLP block was constructed using one of the models from the transformer family (DistilBERT) [Sanh *et al.* (2019)]. The transformer models have an inherent flaw of aligning the sentence embeddings for most of these high-level commands together, mainly due to the high similarity index between the commands, especially for left and right, since they usually differ by just one work. To make these models robust and adaptive to the commands in this dataset, these networks were pre-trained on the commands alone to classify the sentences into 3 categories namely STRAIGHT, LEFT, or RIGHT. The NLP block in ModelV1 embeds the high-level statements into vectors of size 512.

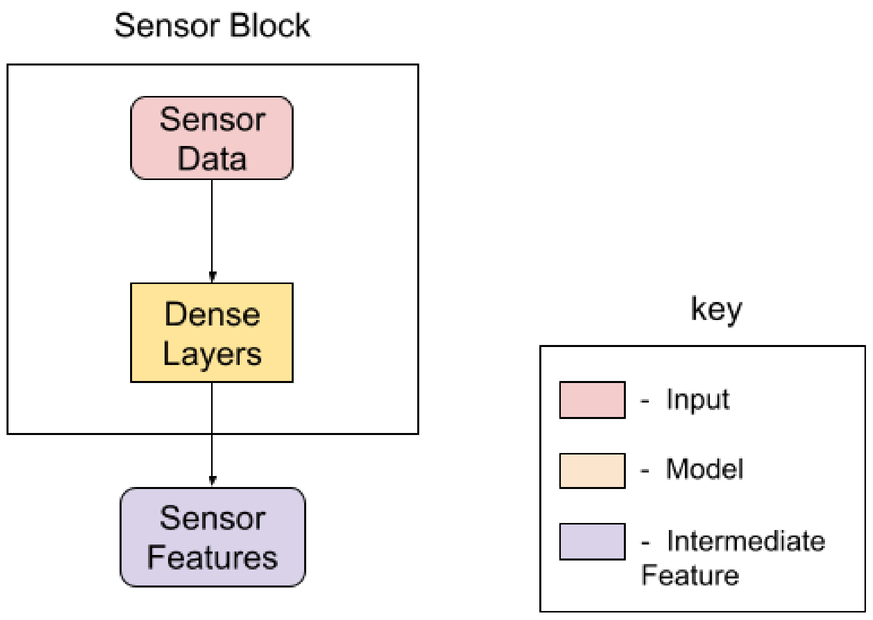


Figure 4.4: ModelV1 Sensor Block Architecture

**Sensors Block -** The final input component of ModelV1 is the Sensors Block. The main task of this block is to ingest the measurement data like speed, accelerometer values, traffic light state, etc, and embed these into a single vector of size 32.

**Combined Regressor -** The last component of ModelV1 is the combined regressor. This component is responsible for combining the feature vector embeddings from all the previous components and use them to predict the control values for the car. The regressor has a stack of Dense layers that process the features from all the earlier components and directly predict the steer, throttle, and brake values of the

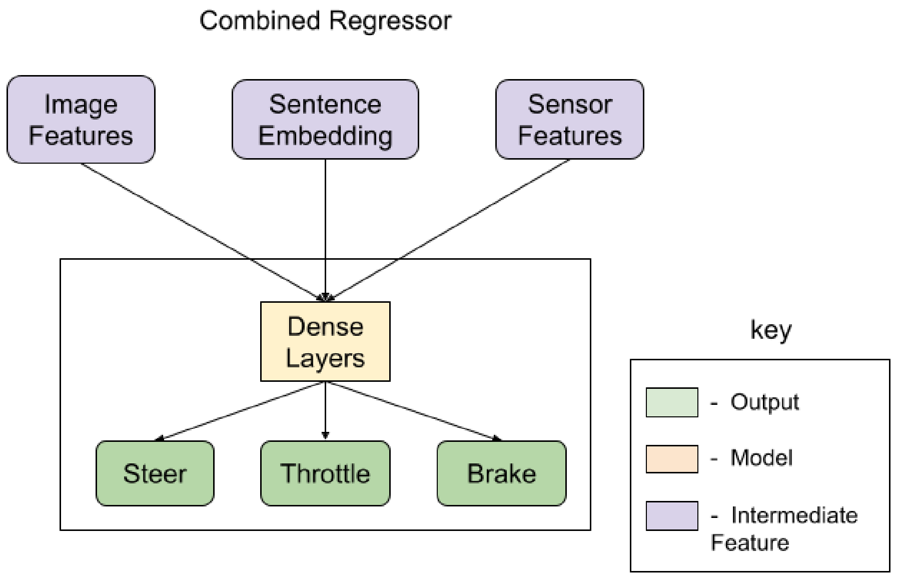


Figure 4.5: ModelV1 Combined Regressor Block Architecture

car.

## 4.4 Results

An essential assumption made while training this network was that the policy is markovian - the actions depend only on the current observations of the system and not exploit temporal properties. In other words, any temporal relation between frames needs to be eliminated since the policy that the agent learns needs to just map the observations with their corresponding actions. Also, the pre-trained NLP block would be frozen in all training processes to ensure its sentence embeddings don't change due to loss from control values. Figure 4.6 shows the cosine similarity of the sentence embeddings generated for all the possible statements that can be generated in this dataset. We can observe small islands of high similarity sub matrices in this diagram that shows that on fine-tuning the embeddings have been trained to distinguish the commands based on the prime objective.

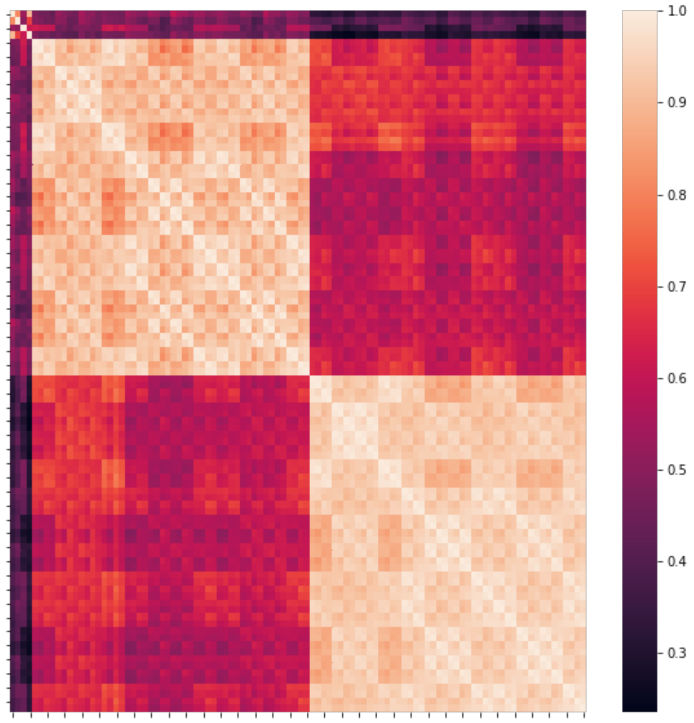


Figure 4.6: Cosine Similarity Of Sentence Embeddings

The agent was trained and tested on datasets collected from town02. The inference testbed had to measure the success rate of the agent in completing a given command. To estimate this, the agent was tested in two scenarios - single level commands and multi-level commands.

### *4.4.1 Singe Level Command Inference*

In this inference testbed, the agent was spawned randomly in the town and a single command was given for it to execute in the nearest proximity. The agent had to take the high-level command and drive through the roads and execute them without colliding or violating the traffic rules. Figure 4.7 shows the success rates of this agent against each type of high-level command that was given. It can be observed that STRAIGHT commands have a higher success rate than the others, which can

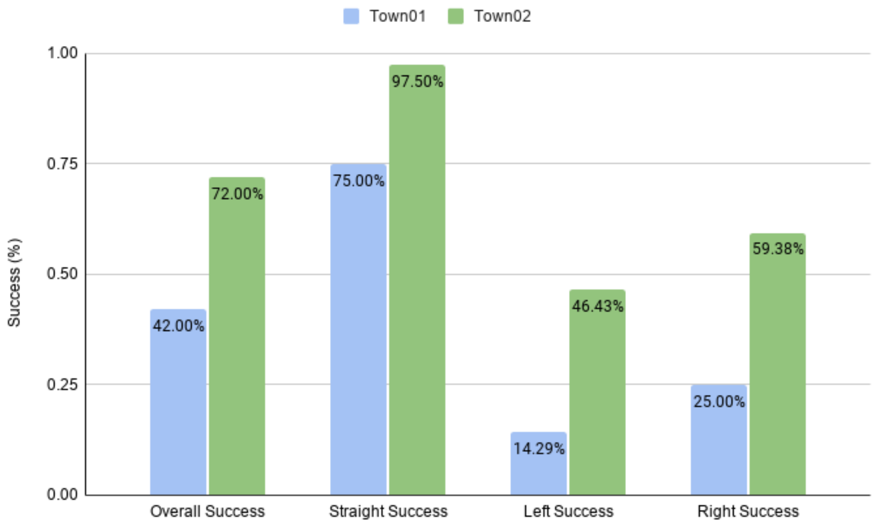


Figure 4.7: Success Rates Across Both Towns

be attributed to the fact that these scenarios don't require the agent to drastically change its steering value.

| **Model** | **Town** | **Overall Success** |
|---|---|---|
| ModelV1 | Town02 (seen) | 0.72 |
| | Town01 (unseen) | 0.42 |
| Goal Conditional Imitation Learning | Town02 (seen) | 0.2 |
| | Town02 (unseen) | 0.26 |
| Non-conditional Imitation Learning | Town01 (seen) | 0.24 |
| | Town02 (unseen) | 0.3 |
| Branched Conditional Imitation Learning | Town01 (seen) | 0.88 |
| | Town02 (unseen) | 0.64 |

Table 4.4: Comparing Success Rates

From Table 4.4, it is evident that this model achieves high accuracy, as good as the existing work, with more complex commands and more scalable architecture. In several episodes, the agent has shown poor lane following skills which can be attributed to the lack of features extracted by the Image Block. Due to the poor lane following abilities, from Table 4.5, it is observed that the average distance covered by this model between successive infractions is too small.

| Model | Town | Average Distance Per Infraction (km) |
|---|---|---|
| ModelV1 | Town02 (seen) | 1.03 |
| | Town01 (unseen) | 0.745 |
| Goal Conditional Imitation Learning | Town01 (seen) | 5.76 |
| | Town02 (unseen) | 0.89 |
| Non-conditional Imitation Learning | Town01 (seen) | 1.87 |
| | Town02 (unseen) | 1.22 |
| Branched Conditional Imitation Learning | Town01 (seen) | 2.34 |
| | Town02 (unseen) | 1.18 |

Table 4.5: Average Distance Between Infractions Comparison

### 4.4.2 *Multi Level Command Inference*

In this inference testbed, the agent was spawned randomly as before but the humans are in the loop. The commands provided to the agent are real-time and the agent has to follow them as it goes through the town. Since the inference bed is not completely automated, the efficiency cannot be measured with a simple success/failure metric. Hence, the average distance traveled by the agent in both towns was calculated. Figure 4.8 shows this information. In town01, which is new and unseen, the agent seems to cover distances as well as the town it was trained on. This is because, humans have the ability to control the car even if it goes astray in the middle but tentatively asking it to get back on lane, a valid proof for why Human-Robot

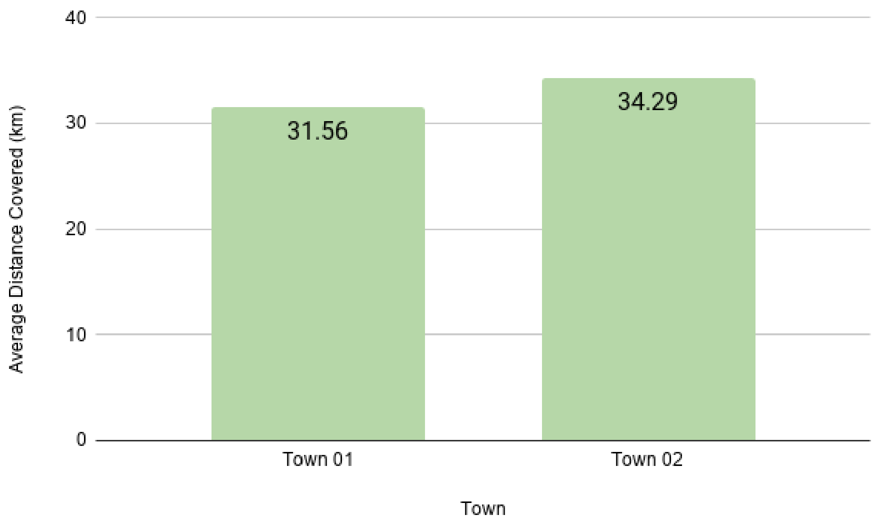


Figure 4.8: Average Distance Covered over 50 runs

Interaction can be beneficial for the development and deployment of self-driving cars.

Table 4.6 shows the success rate of each type of turn that the agent takes in all the episodes. It can be observed as usual, that STRAIGHT succeeds most of the time. When it comes to LEFT/RIGHT turns, the agent seems to miss many of them, though the ones they do take, succeed with high accuracy.

| **Town** | **Overall success** | **Straight success** | **Left misses** | **Right misses** | **Left success** | **Right success** |
|---|---|---|---|---|---|---|
| Town02 (seen) | 0.790 | 0.960 | 0.225 | 19.000 | 0.823 | 0.880 |
| Town01 (unseen) | 0.677 | 0.920 | 0.265 | 24.000 | 0.760 | 0.810 |

Table 4.6: Multiple Level Command-Wise Success and Miss Comparison

## 4.5 Limitations

Though this architecture had great merits in its structure and efficiency, it failed to capture a few details like the duration of lane following prior to making the desired turn instructed by the user. This can be attributed to the fact that each episode in the dataset had only 150 frames, which omits the long lane following scenarios. Also, the nature of this dataset forces the inference bed to supply the high-level commands for actions at the right time in the simulation. Failure to do so, might set the agent off track since it is forced to follow the human instruction but at the wrong time, and eventually end up colliding with other objects. Another big limitation of this network was that the agent failed to maintain lanes while driving. This could be attributed to the fact that there are very few data frames where the agent has seen the importance of lane following and hence the agent tends to focus more on the turn perfections rather than lane follow. To combat these issues and add more interactive agents, another experiment was conducted which is discussed in detail in the next chapter.

Chapter 5

# ADDING ENVIRONMENTAL OBJECT DESCRIPTIONS TO DRIVING INSTRUCTIONS

This chapter discusses the second experiment conducted to make self-driving cars a more interactive and controlled AI agent. To do so, descriptions of environment objects were added to the natural language instructions given to the self-driving cars. The chapter begins with exploring the motivation behind this approach in Section 5.1, followed by the construction of yet another dataset in Section 5.2. The model architecture is explored in Section 5.3 and finally, results are compared and tabulated in Section 5.4.

## 5.1 Motivation

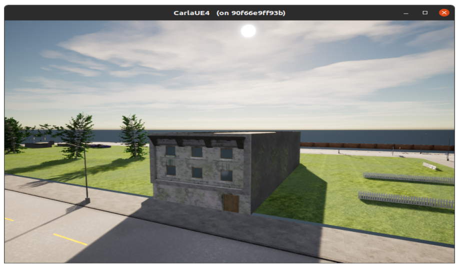
As seen in the earlier chapter, the first experiment was conducted in an attempt to provide a more scalable and better system that could process high-level natural language instructions from the user/human. However, the system failed to surpass the existing benchmarks. Furthermore, no system exists today that could take high-level descriptive instructions to navigate the cars, as humans usually do in cabs. The goal of this approach was to take the setup in the previous chapter and add more complex natural language statements that require the self-driving cars to not just focus on the prime objective of the command but also on the temporal implication hidden behind the descriptive portion.

As humans, we can relate to a scenario, where we are traveling with a stranger in a bike or car and navigate them with our instructions. On many occasions, descriptive instructions are easier to follow for us, such as, "take a left near the tall brown building". These instructions are more complex and harder to achieve since the agent has to not only realize that the user wants to take a left but also find the exact left that the user intends to take based on the object described from the scene. This goes against most conventional works in self-driving cars, wherein the main focus of the vision blocks of the model, apart from identifying other cars, passengers, and traffic signs, is the road. The works so far have always trained their agents to identify the roads and follow the lane whilst making decisions based on human commands. However, in this case, the vision model needs to also focus on the surroundings, which broadens its requirements and the features extracted. Furthermore, the agent needs to be able to adapt to new environments pretty well where the buildings or objects could be totally different as well.

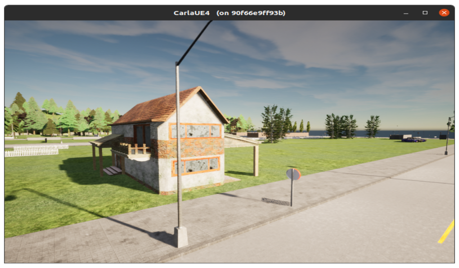
## 5.2 Dataset

The new requirements proposed in this experiment require the need to construct a new dataset, one that allows longer episode lengths and has destined navigational plans for each episode. This dataset comprises several episodes, each of which has a start and end location and a navigational planner that pre-calculates the route. To simplify the task, episodes with route plans that had single turns were only collected. Since the server-side autopilot in CARLA does not allow client code to modify the routes or to perform navigational planning, a client-side PID controller-based agent was used to collect this dataset. The Kp, Kd, and Ki values of the PID controller determined the behavior of the agent if it was "cautious", or "normal" or "erratic". The data was collected using the "normal" behavior agent to complete single turn episodes which could span from as small as 50 frames to 600 frames.

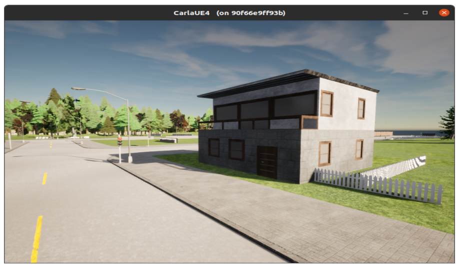
The PID controller agent plans the global route from source to destination initially and then adds waypoints for a local planner to drive the car smoothly. The advantage of using this agent to collect data was that the local planner uses a low-level command signal that needs to be executed to reach the next waypoint. The low-level commands currently are [LANE FOLLOW, LEFT, RIGHT, CHANGE LANE LEFT, CHANGE LANE RIGHT, STOP]. These low-level commands depict exactly what the agent is performing at each timestep. An essential effort taken in this approach is to describe the environmental object. Since these descriptions cannot be obtained through the CARLA engine, these descriptions were collected manually. Any building description would have 4 main attributes - color, size, texture, and type and thus using this template to maintain uniformity, the building descriptions were collected. Figure 5.1 shows some samples of buildings and their descriptions generated.



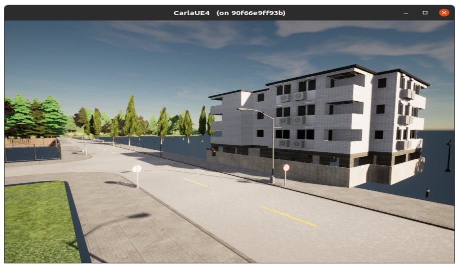
(a) Big Grey Concrete Building



(b) Small Red Roofed House



(c) Small Grey Gallery House



(d) Big White Building

Figure 5.1: Dataset 2 Building Descriptions Sample

| | |
|---|---|
| RGB Image | Brake |
| Semantic Segmntation Image | Low level command |
| Depth Image | Stop Signs (if within 25m) |
| Speed | Speed Limit (if within 25m) |
| Steer | Traffic Signal State (if within 25m) |
| High Level Command | Throttle |

Table 5.1: Dataset 2 Components

| **Prefix/Suffix** | **LEFT** | **RIGHT** | **STRAIGHT** |
|---|---|---|---|
| [ at the next intersection, from the next intersection, at the next junction, from the next junction, at the light, from the light, at the (building_description), from the (building_description) ] | [ take a left, turn left, make a left ] | [ take a right, turn right, make a right ] | [ keep going straight, drive straight, head straight ] |

Table 5.2: High Command Segments for Dataset 2

The components collected in this dataset are listed in Table 5.1. The GNSS and IMU sensor measurements are not taken since they are not always available in real-time. Furthermore, the dataset has a slightly different setup for constructing the high-level commands from the one used in the previous approach. The different sets of command segments can be seen in Table 5.2. The high-level commands can still be constructed randomly by combining the building descriptions and prime objective statements as before, however, to ensure that the agent can follow simple instructions

| **STRAIGHT** | **LEFT** | **RIGHT** |
|---|---|---|
| keep going straight from the big white building | take a left at the next intersection | at the gas station, make a right |
| drive straight from the tall brown glassed building | take a left at the big red roofed house | turn right from the next junction |
| head straight on this street | from the small grey gallery house, turn left | from the small white garage, take a right |

Table 5.3: Dataset 2 High Command Examples

as well, 20% of the training data is constructed by using simple commands as seen in the previous chapter. Table 5.3 shows some of the possible commands this dataset can contain. Figure 5.2 depicts some of the image data contained in this dataset. It is a little different from the previous approach since the position of cameras has changed from overlooking the car to driver seat view to make sure the self-driving cars can see exactly what humans can see in that place.

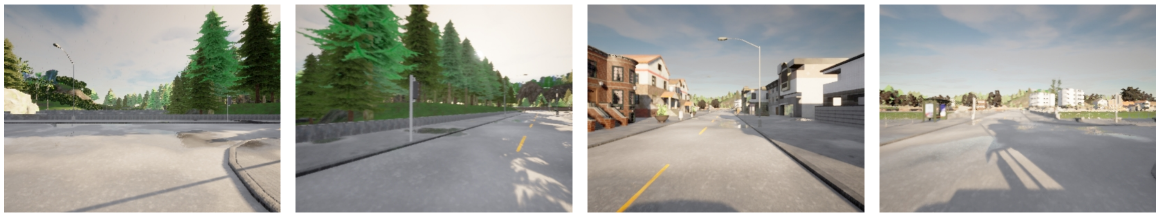


Figure 5.2: Dataset 2 RGB Image Sample

The construction of high level commands for the episodes can be interpreted from Figure 5.3. The rtajectory followed by the car for that episode is shown by the dashed line. Thus the prime objective of the agent in that episode is to take a left. To identify the building description to use for this episode, we follow the following procedure.

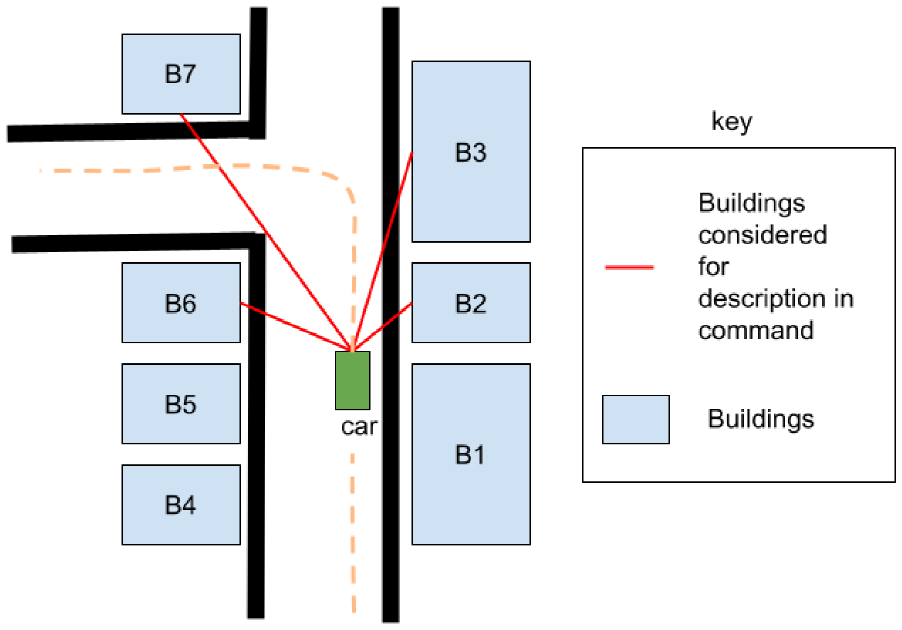


Figure 5.3: Generating High-Level Commands

1. Place the car at a waypoint 15m away from the junction.
2. Identify all the buildings visible in the field of view of the RGB camera in driver seat.
3. Choose the closest building based on a distance estimator

To follow the protocols of previous works, the dataset was collected from both Town01 and Town02. The agent usually was trained on Town01 and tested on Town02. However, as we can see from Table 5.4, the distribution of datapoints amongst the low-level commands is too sparse in the Town01 dataset due to its longer road lengths. Also, the building density in town01 is low, thus making most houses and buildings out of sight from the driver seat camera. Owing to these reasons, the Town02 dataset was used to train the model, as it had a lower ratio of distribution between low-level commands.

| **TOWN** | **LANE_FOLLOW** | **STRAIGHT** | **LEFT** | **RIGHT** |
|---|---|---|---|---|
| Town 01 | 371,695 | 88,941 | 118,412 | 121,505 |
| Town 02 | 659,143 | 151,368 | 172,406 | 169,563 |

Table 5.4: Dataset Distribution Among Low Commands

## 5.3 Model Architecture

In this section, the model architecture devised to tackle the new dataset is described. There are two main models developed in this approach, one is a semantic segmentation model trained to identify 7 types of tags in the scene and the second is the end-to-end self-driving network, henceforth referred to as ModelV2. To train semantic segmentation, a UNet architecture [Ronneberger *et al.* (2015)] is trained using the RGB data as input and the pixel-wise tags extracted from the semantic segmentation images collected as a part of this dataset. Figure 5.4 shows the UNet architecture and Figure 5.5 shows the outputs of the trained segmentation UNet model. It can be observed from these images that the trained segmentation model is able to distinctly identify objects in the environment and tag them with almost complete accuracy even in unseen towns.

ModelV2 has a total of 7 components - 3 input blocks to process 3 different modalities of data and 4 output heads to predict control values, speed limits, traffic signal states, and instantaneous low commands. Figure 5.6 shows the abstract model architecture explained below.

**Vision Block -** The first component of ModelV2 is the vision block, depicted in Figure 5.7. The block is a modified version of Resnet50 architecture [He *et al.* (2016)], with the first convolution layer alone modified to take 4 channeled images. The vision block is loaded with weights pre-trained using imagenet dataset [Deng *et al.* (2009)]

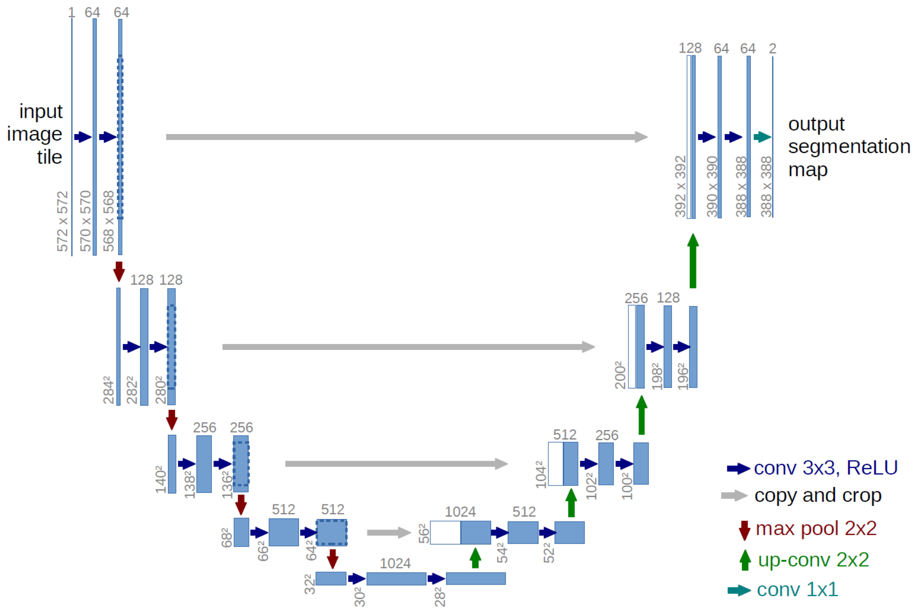
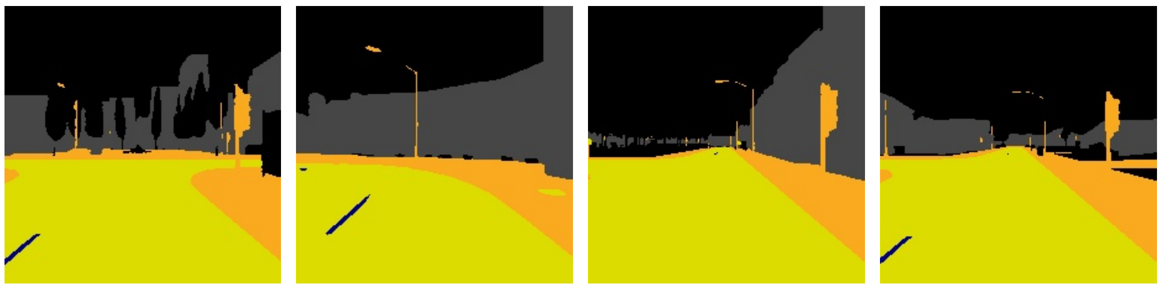


Figure 5.4: UNet Architecture



(a) Example 1 (b) Example 2 (c) Example 3 (d) Example 4

Figure 5.5: UNet Model Semantic Segmentation Predictions

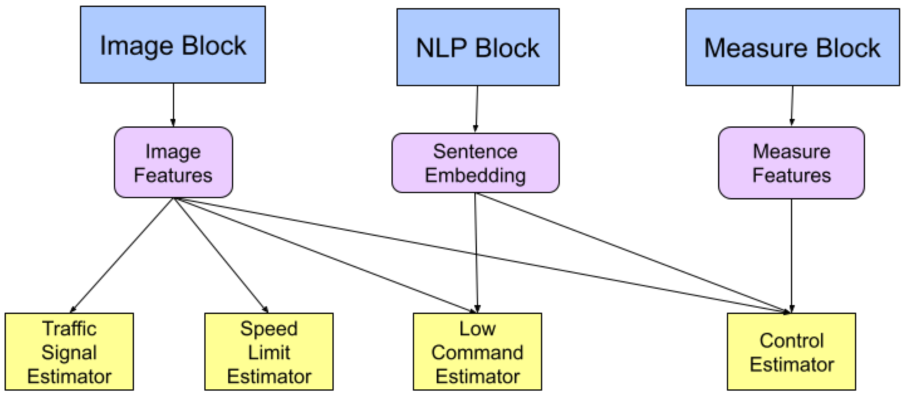


Figure 5.6: ModelV2 Abstract Architecture

and then fine-tuned completely using this dataset. The vision block extracts features from the image as a 1000 dimensional vector. These features play a vital role in not just predicting control values but also other outputs as described in the later sections.

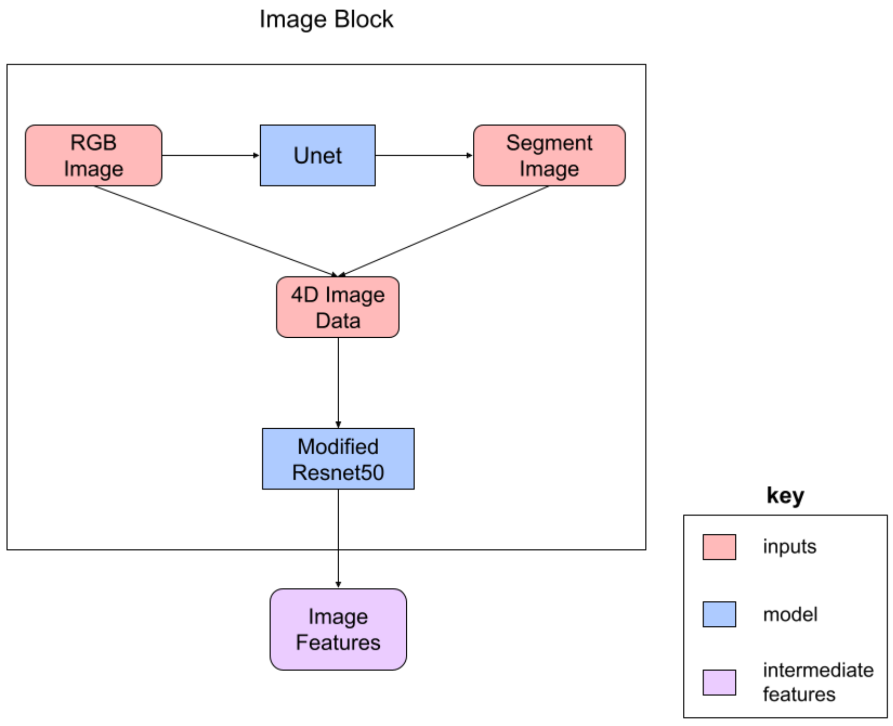


Figure 5.7: ModelV2 Image Block

**NLP Block -** The second component of ModelV2 is the NLP block, depicted in Figure 5.8. In ModelV1, this block was constructed using pretrained DistilBERT model fine-tuned on command types. However, in this model architecture, the NLP block is comprised of Glove Embeddings [Pennington *et al.* (2014)] and 2 stacked GRU layers. The sentences are embedded using GRU layers to avoid fine-tuning as done in the previous approach and also to learn from scratch the sentence embedding that can be used to identify the instantaneous low command much more effectively. The NLP block ingests the high-level instruction and converts it into a 128-dimensional vector.

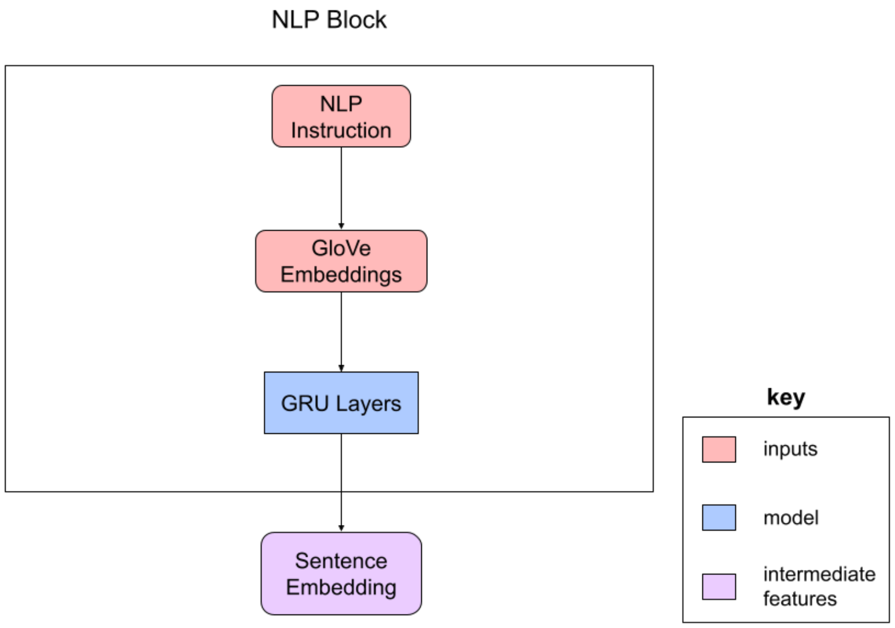


Figure 5.8: ModelV2 NLP Block

**Measure Block -** The third component and the last of input blocks in ModelV2 is the measure block, depicted in Figure 5.9. It takes in the current speed of the car and embeds that into higher dimensions by running it through some stacked Dense layers. The measure block gives out a 32-dimensional vector as the measurement feature.

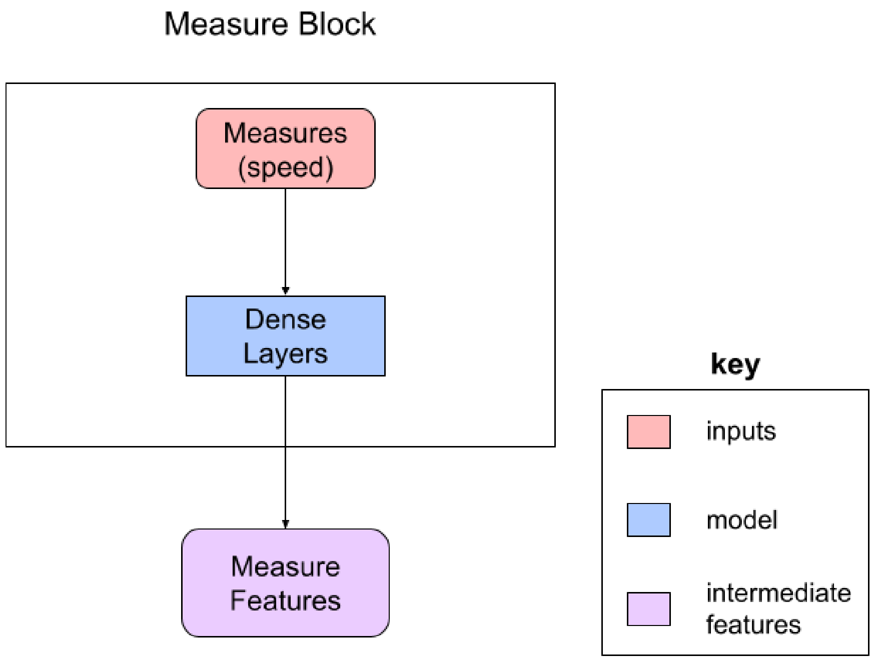


Figure 5.9: ModelV2 Image Block

**Low Command Estimator -** The fourth component and the first output component of ModelV2 is Low Command Estimator, depicted in Figure 5.10. It takes the sentence embedding from NLP Block and Image features from Vision Block and predicts the instantaneous low command to follow. This is one of the most important auxiliary outputs as it indirectly maps the observation and user instruction together in estimating the next right step on an abstract level.

**Control Estimator -** The fifth component and the second output component of ModelV2 is the Control Estimator, depicted in Figure 5.11. It takes the image features from Vision Block, sentence embedding from NLP Block, and measure features from Measure Block and predicts the steer, brake, and throttle values. This is the prime output that is essential in driving the car. The control values have to be as accurate as possible to ensure the car navigates through the city properly, irrespective of whether

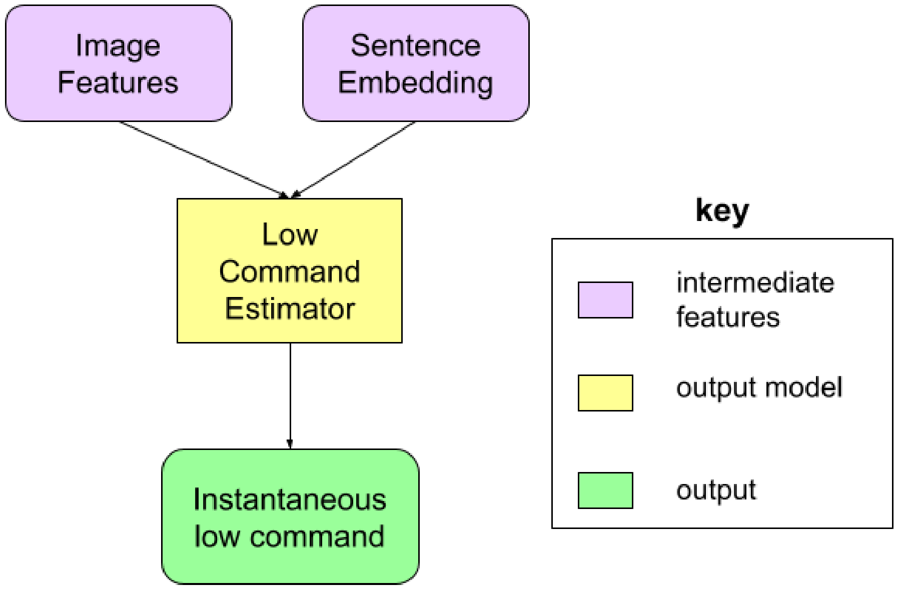


Figure 5.10: ModelV2 Low Command Estimator

it follows human instructions or not.

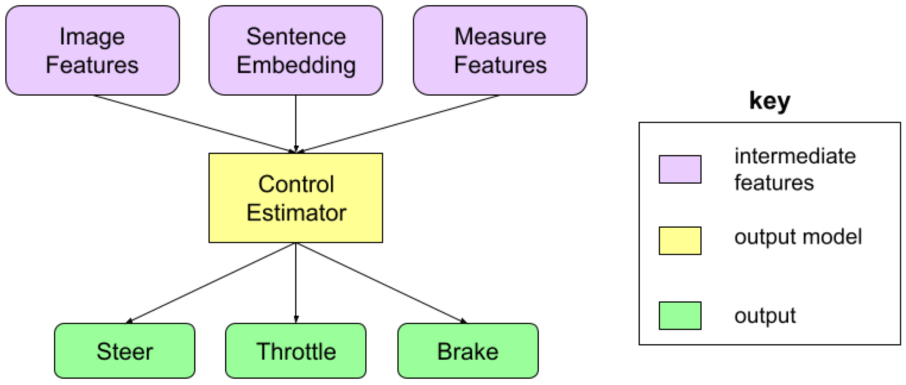


Figure 5.11: ModelV2 Control Estimator

**Traffic Signal Estimator -** The sixth component and the third output component of ModelV2 is the Traffic Signal Estimator, depicted in Figure 5.12. It takes the image features from Vision Block and estimates if a traffic light is within 25m of the car and visible. If yes, then another branch predicts the value/state of the traffic signal affecting the self-driving car.

**Speed Limit Estimator -** The seventh component and the last output component of ModelV2 is the Speed Limit Estimator, depicted in Figure 5.12. It also takes the image features from Vision Block and estimates if a speed limit sign is within 25m of the car and visible. If yes, then another branch predicts the value written on the speed limit board affecting the self-driving car.

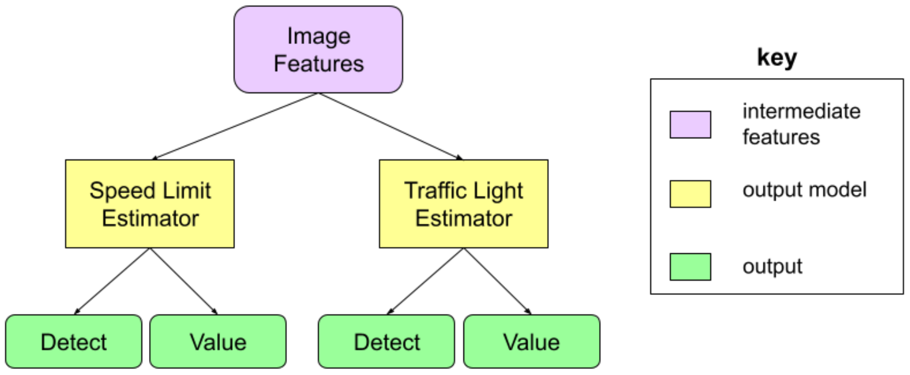


Figure 5.12: ModelV2 Speed Limit and Traffic Light Estimator

## 5.4 Results

The experiments were conducted by training and testing the agent on either of the towns. As before, the Markovian assumption is employed [Bojarski *et al.* (2016); Codevilla *et al.* (2018)] and hence the model learns a policy to map the observations (RGB image, NLP command, and speed of car) to the actions at each timestep. The trained models are tested by running on an inference bed that calculates not just the success rate of the model in that specific town but also measures of an autonomous driving benchmark like route completion and infraction scores. Furthermore, to measure the efficiency of the agent in imitating the expert, the expert is also run side-by-side with the trained agent and check similarity in command sequences.

During inference, to complete an episode successfully, the agent had to navigate from the location it was spawned, follow the lane till the junction of decision making, use human-supplied natural language descriptive instruction to make a turn and then stop at the destination. The episodes would have scenarios wherein the agent would have to skip junctions before making the turn in order to satisfy the described building. Figure 5.13 shows the success rates when the dataset collected from Town02 was used for training and the agent was tested on both towns - Town02 (seen) and Town01(unseen). It can be observed that the agent successfully completes almost all episodes in Town02 but fails exactly by half in Town01, which it had not seen in its training. Figure 5.14 shows the success rates when the dataset collected from Town01 was used for training and the agent was tested on both towns - Town02 (unseen) and Town01(seen). As opposed to conventional expectations, the agent seemed to fail in both towns.

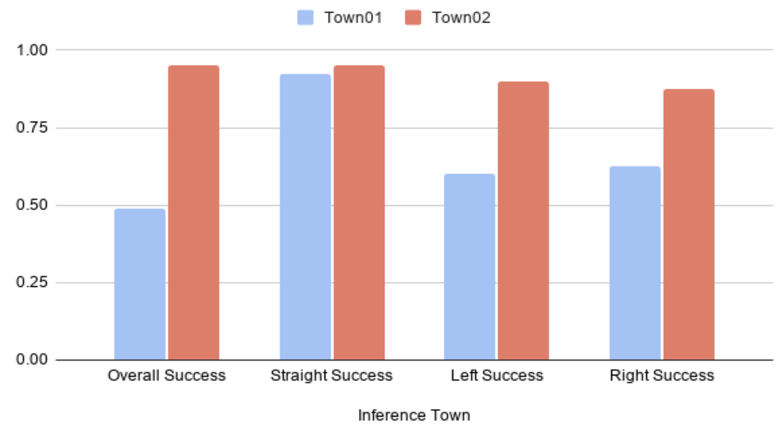


Figure 5.13: Success Rates by Training on Town02

Another way to compare the efficiency of ModelV2 with the existing baseline is by measuring the average distance between infractions. Table 5.5 shows the average distance traveled by the agent when trained on town02 and town01. It can be noted

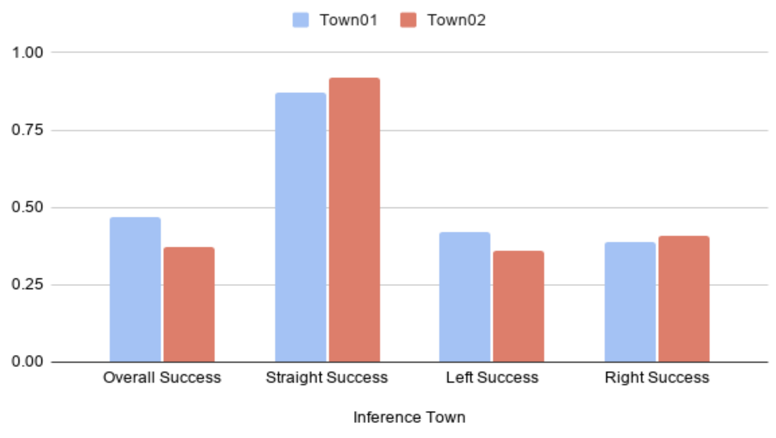


Figure 5.14: Success Rates by Training on Town01

that the agent seems to cover long distances in the town it was originally trained on since it can perform lane following with high accuracy. However, the agent's ability to follow lanes in the unseen town, though comparable isn't better than the baselines.

| Model | Town 02 | | Town 01 | |
|---|---|---|---|---|
| | Inference Town | Average Distance Per Infraction (km) | Inference Town | Average Distance Per Infraction (km) |
| ModelV2 | Town 01 (unseen) | 0.947 | Town 01 (unseen) | 1.712 |
| | Town 02 (seen) | 3.042 | Town 02 (seen) | 0.862 |
| Goal Conditional Imitation Learning | Town 01 (seen) | - | Town 01 (seen) | 5.76 |
| | Town 02 (unseen) | - | Town 02 (unseen) | 0.89 |
| Non-conditional Imitation Learning | Town 01 (seen) | - | Town 01 (seen) | 1.87 |
| | Town 02 (unseen) | - | Town 02 (unseen) | 1.22 |
| Branched Conditional Imitation Learning | Town 01 (seen) | - | Town 01 (seen) | 2.34 |
| | Town 02 (unseen) | - | Town 02 (unseen) | 1.18 |

Table 5.5: Average Distance (km) Between Infractions

To further understand and explore the results obtained, metrics used in the autonomous driving benchmark were slightly tweaked to fit the inference scenarios. The agent's ability to complete a given episode is measured by comparing its route completion and weighting that with infraction penalty to obtain an overall score for that episode. Table 5.6 outlines the average metrics scored by the agent in these experiments. It can be observed that the best scores are achieved when trained and tested on town 02, followed by training and testing on town01. Both cases of unseen towns perform comparably the same, which might seem like the agent fails to adapt. On closer evaluation, we can see that the route completion values are still high in these setups, however, the infractions are the main reason the agents fail.

| Train Town | Inference Town | Route Completion | Collision Infraction | Lane Invasion Infraction | Total Infraction | Overall Score |
|---|---|---|---|---|---|---|
| Town 02 | Town 01 | 78.913 | 0.715 | 0.642 | 0.412164 | 32.52509773 |
| Town 02 | Town 02 | 96.292 | 0.979 | 0.991 | 0.982081 | 94.56654365 |
| Town 01 | Town 01 | 75.358 | 0.926 | 0.953 | 0.908209 | 68.44081382 |
| Town 01 | Town 02 | 63.147 | 0.634 | 0.761 | 0.579121 | 36.56975379 |

Table 5.6: Autonomous Driving Metrics

In Figure 5.15, the command similarity of the trained agent with that of the expert agent is plotted. The graph also aligns very well with the results from the previous table wherein the model performs very well in towns it was trained on but fails to follow commands at the right moment in unseen towns.

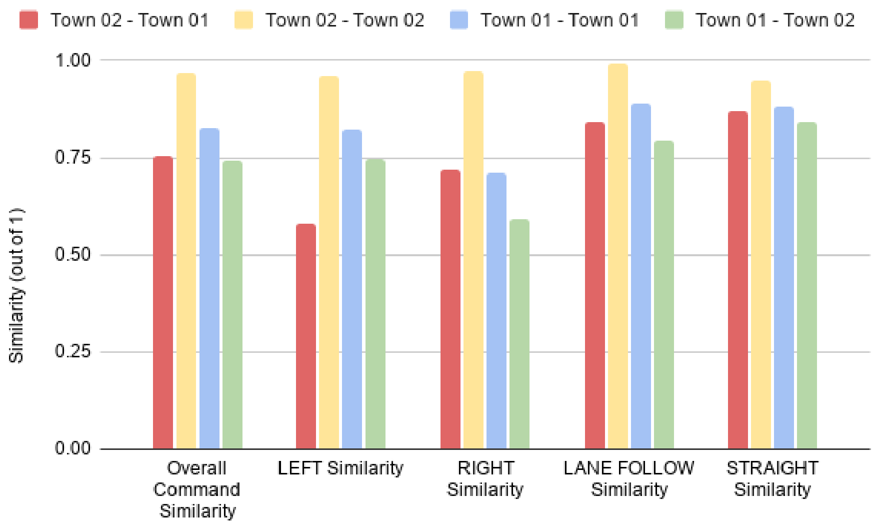


Figure 5.15: Instantaneous Low Command Similarity (Train - Inference) setups

As seen from Figure 5.6, the model has 2 output blocks responsible for identifying the traffic lights and speed limits on the streets. To explore the efficiency of the model in doing that, GradCams were used to follow the gradients through the architecture all the way to the input image, and a saliency map was created to highlight the regions of interest.

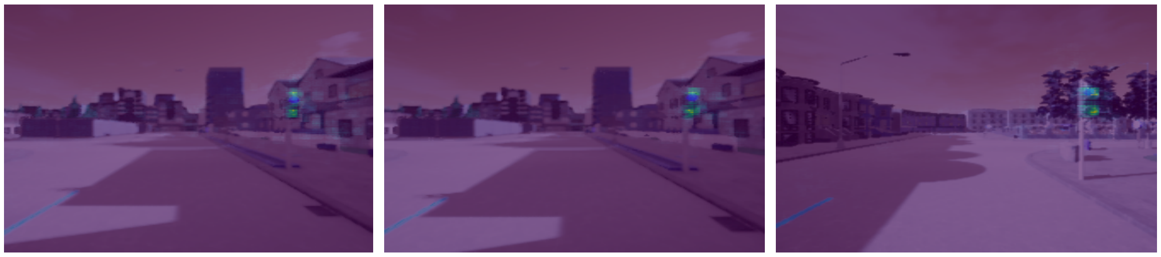


Figure 5.16: Saliency Maps based on Traffic Signal Predictions

Figure 5.16 shows the saliency maps generated using traffic signal as output. It can be observed, in the case of traffic signals, the model seems to completely focus on the traffic signal lights to detect their presence and to identify their state it seems to aptly focus on the two main lights - Red and Green.

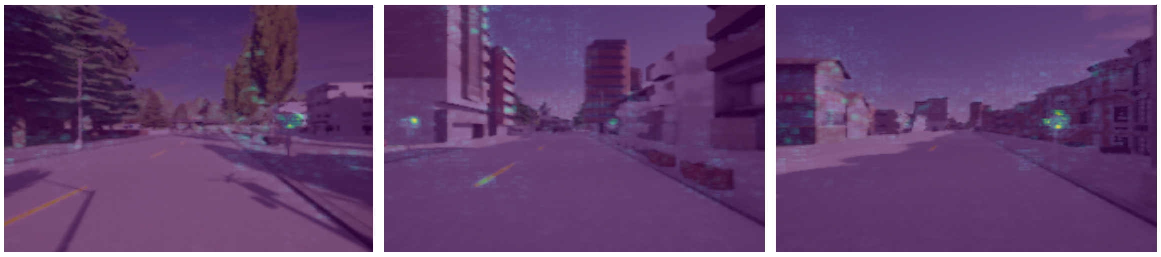
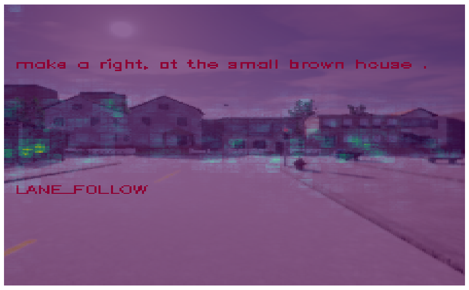


Figure 5.17: Saliency Maps based on Speed Limit Predictions

Figure 5.17 shows the saliency maps generated using speed limits as output. In the case of speed limits, it can be observed that the model identifies the boards quite accurately to detect the speed limit signs and to predict the value, they seem to focus on the numbers on the board as expected.

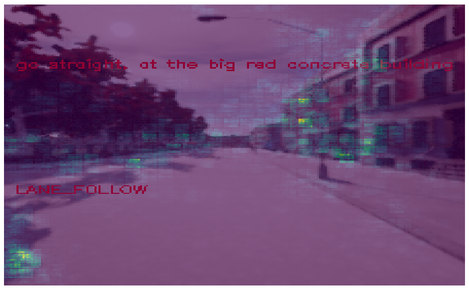
Finally, to check which portion of the image the model focuses on to take turn decisions, the model grads based on Low Command Estimator Outputs. Figure 5.18

shows the saliency maps generated using this output. These saliency maps do not provide a complete understanding of how the model's decision-making abilities but most certainly provide partial answers.

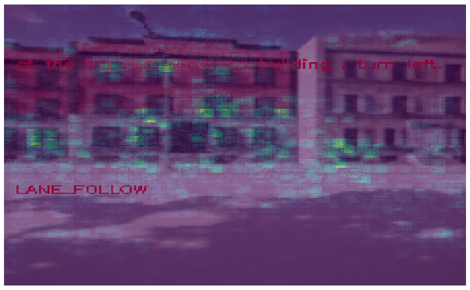


(a) Example 1

(b) Example 2



(c) Example 3



(d) Example 4

Figure 5.18: Saliency Maps based on Low Command Predictions

It can be observed that the model has focused on the buildings described in the command while trying to make a decision. In the first two samples, the agent focuses on a small brown building by the left side of the road. In the last two, the agent focuses on the red concrete building, either on the side or straight ahead when making the decision to turn. This shows that the model has learned a way to combine the information from the NLP block and the vision block to make the right decisions.

# Chapter 6

# INFERENCE AND DISCUSSION

This chapter focuses on combining the observations and results obtained through various stages seen so far and understanding their causality. First, results from both approaches are compared, and then the comparisons between these approaches and existing baselines to reason out the tendencies of success and failure in these approaches.

## 6.1 Comparing ModelV1 and ModelV2

The main difference between these approaches arises from the way the episodes are collected and the nature of high-level command given to the self-driving car. ModelV1 attempts to extrapolate the work in Codevilla *et al.* (2018) by adding an NLP block to process natural language commands and removes the redundant branches. However, ModelV2 attempts to extend this further by making changes to the command, making it more descriptive of the environment rather than just giving directional instructions. Comparing the success rates from Figs. 4.7 and 5.13, we can see that ModelV2 clearly outperforms ModelV1. Also, comparing these models with respect to the distance between infractions from Tables 4.5 and 5.5, ModelV2 seems to travel longer than even baseline models in some cases, which depicts that ModelV2 can perform lane following and commands far better than ModelV1.

The architecture proposed in ModelV2 has outperformed ModelV1 architecture in most metrics however, its efficiency in roaming the city continuously with new commands at every junction hasn't been tested. In multi-level command sequence, ModelV1 has proven to be good, making at least 5 turns in every episode. The multi-level experiments provide proof that interaction between humans and self-driving cars in navigational instructions proves to be a successful avenue of research. The results inherently show that with humans guiding and instructing the car, the agents can navigate through any city, whether it has seen them or not since the humans can correct the self-driving cars if required.

## 6.2 Comparing Proposed Approaches with Baseline

Each proposed approach has different advantages over the baseline approach. ModelV1 removes the redundant branches and adds high-level commands to the model instead of a simple signal. This way this architecture provides a scalable and modular architecture that can process new commands and scenarios in the future with the advent of more functional simulators. However, this approach fails to surpass the baseline results due to several reasons. Firstly, from Tables 4.4 and 4.5, we can see that ModelV1 achieves comparable success rates on the same tows as it was trained, however, fails to generalize well to new unseen town, and this model has a smaller distance between infractions mainly due to its tendency to veer off the lane to adapt to user commands. It can however be observed that this model outperforms other baselines like simple imitation and goal conditional models in success rates.

ModelV2 takes the work to next level of complexity to make the high-level commands more descriptive of the environment. The new structure of command could serve very well as a single level and double level command sequence which instructs the agents to make multiple turns in the same instruction sequentially. To incorporate these additions and complexities, the model architecture is improved in ModelV2. Also, to avoid pretraining of NLP block, it is substituted with GLoVE embeddings and GRU layers. We can draw several conclusions by comparing the results from Table 5.5 and Figs. 5.13 and 5.14. Firstly it can be observed that ModelV1 achieves comparably high success rates on the same tows as it was trained, however fails to generalize well to a new unseen town. This model also achieves a higher average distance between infractions than baseline models in the town it was trained on but has smaller values for unseen towns. Though the vision block improved lane following in this model, the above results can be justified due to the incapability of this architecture in identifying the right turns or junctions that lead to more collisions.

From Table 5.6, it can be noted that the route completion on average is low when the agent is in an unseen town. Also, the collision infractions are much higher in these cases causing the cars to succeed in far fewer episodes. Another main reason for the failure of this model to generalize across towns is that the layout of these towns is very different. Town01 has lower building density and more free driveways to houses that due to some weather settings can be deceived as roads. The vision block extracts features that are necessary to control the car but also identify traffic lights and speed limits. On top of this, the model is trained in an end-to-end fashion thus making it hard to identify what portions of the instructions the agent is focusing on while making control and low command predictions. Also, the building descriptions for these towns are very different. These inherent reasons make it hard for ModelV2 to generalize across towns. It can however be observed that this model outperforms other baselines like simple imitation and goal conditional models in success rates and has a comparable distance between infractions. This proves that the agent is capable of making better decisions than any approach given it has at least seen the town a few times.

# Chapter 7

# CONCLUSION AND FUTURE WORKS

This chapter, based on the observation and inferences drawn, focuses on summarizing the work and all its merits. It then goes on to describe the scope of future works that could potentially make this approach the very next step towards interactive and human-friendly self-driving cars.

## 7.1 Conclusion

The work presented takes the next step to develop self-driving cars that follow human instructions expressed in natural language either in simple pretext or in a complex descriptive manner. The ModelV1 took a step towards a scalable and generalized architecture for end-to-end self-driving cars and achieved comparable to state-of-the-art results in most cases. ModelV2 extended this approach to complicated descriptive instructions that involved building descriptions in them. It showed phenomenal and promising results in a dense town but failed in generalization and sparse town setups. The main reason behind its failure being the difference in town setups and more importantly the complicated nature of the episode (skipping intersections, navigating through deadends, etc) represented by a single statement. Furthermore, driving has always been seen as a temporal task, the actions taken in previous time step do affect the actions taken in the next, however assuming that these effects would be encompassed by the observations alone, or in other words, assuming a Markovian approach, forced the model to draw information for successful driving from the current state alone. With this in retrospection, ModelV2 is the next step towards interactive self-driving cars controlled and guided by humans in natural language.

## 7.2 Future Works

The experiments conducted in this work entails the potential of the proposed approach in various facets. There is scope of improvement and potential for valuable application in future for this research and these are discussed in this section.

- First, the actual ability of the model in navigating through unseen towns can be tested much better if the towns had the same buildings in them which makes the language model more adaptive to the unseen towns based on its learning. The results attained from such experiments would help analyze the comparable performance of the proposed architectures across similar environmental setups thus giving us a wholesome picture of reasonable scalability.

- Second, the use of complex language conditioning in this work proves that we can develop architectures to process much more complicated instructions given by the user. Ordinal instructions can be seen as a natural navigational command provided by most GPS systems today. Developing systems capable of handling them would be really useful in integrating real-time GPS systems with self driving cars. Several approaches can be taken to aid this, it could be a complex deep neural network with gated attention or neural turing machines capable of performing counting tasks along with navigation.

- Third, the work paves way for challenging new types of self-driving modes and instructions. One of the most popular instruction given is ”Follow the red ¡model, make¿ car till you reach my office.”. In this case, the agent needs to perform navigation through the streets by following the same actions taken by another car that it needs to identify in front or beside it.

- Fourth, any real time control system requires to be agile and quick to respond to the user/human requirements. The delay in time taken by the agent process the instruction given by user and navigate accordingly needs to be small enough to compete with real world scenarios.

- Fifth, the work has shown immense capability for the model to robustly adapt to incorrect instructions given by humans and take the desired turns when possible. Metrics to test these scenarios and adapt to them would be required to also add on to the robustness of this approach.
- Sixth, the architecture has shown vulnerabilities in terms of its capacity in generalizing to unseen towns. Efforts and work is required to try different approaches to learn a policy that can be more scalable to such scenarios.
- Seventh, the effectiveness of any self-driving car can be verified by trying it on a real robot, and hence testing the proposed architecture on our RC cars would settle the question of adaptability.
- Last, being a very dynamic and booming research topic, several works propose different approaches to self-driving cars like trajectory prediction, hierarchical imitation or reinforcement learning or using deep learning to only identify features that can be used by physics engines to drive the car. There is a need to find a way to compare all these approaches on a single benchmark metric that can help determine which of these could potentially be the right way ahead.

# REFERENCES

Abbeel, P., A. Coates, M. Quigley and A. Y. Ng, “An application of reinforcement learning to aerobatic helicopter flight”, Advances in neural information processing systems **19**, 1 (2007).

Asfour, T., P. Azad, F. Gyarfas and R. Dillmann, “Imitation learning of dual-arm manipulation tasks in humanoid robots”, International Journal of Humanoid Robotics **5**, 02, 183–202 (2008).

Berger, E., H. B. Amor, D. Vogt and B. Jung, “Towards a simulator for imitation learning with kinesthetic bootstrapping”, in “Workshop Proceedings of Intl. Conf. on Simulation, Modeling and Programming for Autonomous Robots (SIMPAR)”, pp. 167–173 (2008).

Bojarski, M., D. Del 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”, arXiv preprint arXiv:1604.07316 (2016).

Calinon, S. and A. Billard, “Incremental learning of gestures by imitation in a humanoid robot”, in “Proceedings of the ACM/IEEE international conference on Human-robot interaction”, pp. 255–262 (2007).

Chernova, S. and M. Veloso, “Teaching collaborative multi-robot tasks through demonstration”, in “Humanoids 2008-8th IEEE-RAS International Conference on Humanoid Robots”, pp. 385–390 (IEEE, 2008).

Codevilla, F., M. Miiller, A. López, V. Koltun and A. Dosovitskiy, “End-to-end driving via conditional imitation learning”, in “2018 IEEE International Conference on Robotics and Automation (ICRA)”, pp. 1–9 (IEEE, 2018).

De Santis, A., B. Siciliano, A. De Luca and A. Bicchi, “An atlas of physical human–robot interaction”, Mechanism and Machine Theory **43**, 3, 253–270 (2008).

Deng, J., W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database”, in “2009 IEEE conference on computer vision and pattern recognition”, pp. 248–255 (Ieee, 2009).

Dosovitskiy, A., G. Ros, F. Codevilla, A. Lopez and V. Koltun, “Carla: An open urban driving simulator”, in “Conference on robot learning”, pp. 1–16 (PMLR, 2017).

He, K., X. Zhang, S. Ren and J. Sun, “Deep residual learning for image recognition”, in “Proceedings of the IEEE conference on computer vision and pattern recognition”, pp. 770–778 (2016).

Hussein, A., M. M. Gaber, E. Elyan and C. Jayne, “Imitation learning: A survey of learning methods”, ACM Computing Surveys (CSUR) **50**, 2, 1–35 (2017).

Ikemoto, S., H. B. Amor, T. Minato, B. Jung and H. Ishiguro, “Physical human-robot interaction: Mutual learning and adaptation”, IEEE robotics & automation magazine **19**, 4, 24–35 (2012).

Kober, J. and J. Peters, “Imitation and reinforcement learning”, IEEE Robotics & Automation Magazine **17**, 2, 55–62 (2010).

Kober, J. and J. Peters, “Policy search for motor primitives in robotics”, in “Learning Motor Skills”, pp. 83–117 (Springer, 2014).

Liang, X., T. Wang, L. Yang and E. Xing, “Cirl: Controllable imitative reinforcement learning for vision-based self-driving”, in “Proceedings of the European Conference on Computer Vision (ECCV)”, pp. 584–599 (2018).

Mataric, M. J., “Getting humanoids to move and imitate”, IEEE Intelligent Systems and their Applications **15**, 4, 18–24 (2000).

Mülling, K., J. Kober, O. Kroemer and J. Peters, “Learning to select and generalize striking movements in robot table tennis”, The International Journal of Robotics Research **32**, 3, 263–279 (2013).

Ng, A. Y., A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger and E. Liang, “Autonomous inverted helicopter flight via reinforcement learning”, in “Experimental robotics IX”, pp. 363–372 (Springer, 2006).

Ollis, M., W. H. Huang and M. Happold, “A bayesian approach to imitation learning for robot navigation”, in “2007 IEEE/RSJ International Conference on Intelligent Robots and Systems”, pp. 709–714 (IEEE, 2007).

Pennington, J., R. Socher and C. D. Manning, “Glove: Global vectors for word representation”, in “Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)”, pp. 1532–1543 (2014).

Ratliff, N., D. Bradley, J. A. Bagnell and J. Chestnutt, “Boosting structured prediction for imitation learning”, (2007).

Raza, S., S. Haider and M.-A. Williams, “Teaching coordinated strategies to soccer robots via imitation”, in “2012 IEEE International Conference on Robotics and Biomimetics (ROBIO)”, pp. 1434–1439 (IEEE, 2012).

Roh, J., C. Paxton, A. Pronobis, A. Farhadi and D. Fox, “Conditional driving from natural language instructions”, in “Conference on Robot Learning”, pp. 540–551 (PMLR, 2020).

Ronneberger, O., P. Fischer and T. Brox, “U-net: Convolutional networks for biomedical image segmentation”, CoRR **abs/1505.04597**, URL `http://arxiv.org/abs/1505.04597` (2015).

Sammut, C., S. Hurst, D. Kedzier and D. Michie, “Learning to fly”, in “Machine Learning Proceedings 1992”, pp. 385–393 (Elsevier, 1992).

Sanh, V., L. Debut, J. Chaumond and T. Wolf, “Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter”, arXiv preprint arXiv:1910.01108 (2019).

Schaal, S., “Is imitation learning the route to humanoid robots?”, Trends in cognitive sciences **3**, 6, 233–242 (1999).

Silver, D., J. Bagnell and A. Stentz, “High performance outdoor navigation from overhead data using imitation learning”, Robotics: Science and Systems IV, Zurich, Switzerland **1** (2008).