SIM-TO-REAL CONDITIONAL END-TO-END SELF-DRIVING VEHICLE THROUGH VISUAL PERCEPTION

A THESIS
SUBMITTED TO
THE DEPARTMENT OF ELECTRONICS AND ELECTRICAL COMMUNICATIONS
FACULTY OF ENGINEERING, CAIRO UNVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
BACHELOR OF ENGINEERING

ABDELRAHMAN HUSSEIN EL-GAMMAL
AHMED SHERIF SALAH
AMR MEDHAT IBRAHIM
FAYROUZ YEHIA ZAKARIA
NERMEEN MOHAMED ELEWA
MOHAMED TAREK SHAABAN

JULY 2019

Abstract

Self-driving vehicles, or cars in particular, is the sleeping giant of this century, with more and more cars on the road, and more fatalities due to human error, the need of automating the driving task increases. The advancements being made in the hardware capabilities, along with the huge amount of data available, deep learning can be extensively used overcoming the performance and the reliability of any rule based method to perform such task. Moreover, using deep learning or AI in a hand-engineered context shall limit its capabilities to our human interpretation of the provided data, which is in our case a visual of the road in front of the vehicle, therefore we adopted an end-to-end approach to cover the driving task starting from an image of the road in front of the vehicle till the control signals sent to the vehicle's motors, all in one computation without any pre or post processing. And since deep learning requires huge amounts of data, collecting data from the real world might not be the best option due to the measurements inaccuracy, driving style variations, time consumed, possible dangers, and more reasons that eliminated collecting a real-world dataset, thus, we made use of the simulators available to have a consistent driving style with various scenarios, accurate measurements, without time or power limitations, and completely safe. Afterwards, the driving policy learned in the simulated

environment can be modularly transferred to the real environment, and perform almost as well as it did in the simulator on real physical roads.

Acknowledgments

This dissertation would not have been possible without the support of many people. First and foremost, we would like to thank our advisors and role models Dr. Hassan Mostafa, Dr. Hossam Hassan, Dr. Ibrahim Sobh, and Eng. Mahmoud El Khateeb.

Dr. Hassan Mostafa and Dr. Hossam Hassan, we would like to thank you both for your patience, guidance, support the great supervision you offered us through the whole project, and equipment you helped us access and use easily

Dr. Ibrahim Sobh and Eng. Mahmoud El Khateeb, you both provided perfect technical guidance, through the whole project, that helped us reach our desired objective, address various complex problems we faced, perform several advanced techniques. We would specially thank you for the tremendous amount of support and guidance you provieded us..

Moreover, we want to thank Dr. Filipe Codevilla, Dr. Mattias Muller, and Eng. Loay Amin, who have helped us with their relevant experience on several problems we faced through our project.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

There are various benefits self-driving cars have to offer on different aspects. Most importantly, they could make roads much safer. For example, the leading cause of most accidents in our daily life is the human error. According to the statistics provided by World Health Organization (WHO) [1], most of the road fatalities are caused by the human error as shown in the following figure 1, therefore self-driving cars can a more reliable approach for reducing these human errors.

Moreover, there is a study made by Eno Centre for Transportation [2], this study found out that if ten percent of all cars were self-driving, as many as 211,000 accidents would be prevented annually. Some 1,100 lives would be preserved, and the economic costs of automobile accidents would be reduced by more than $20 billion. An additional benefit could be decreasing or even eliminating traffic congestion can be achieved by self-driving cars by following a consistent behavior during traffic jams, turning all cars on the road into a fleet of cars moving similarly with interconnection and intercommunication among them.

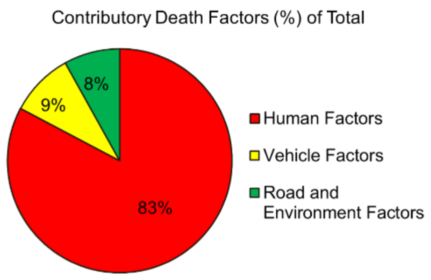


Figure 1. Roads Fatalities Factors

Another crucial aspect is the amount of time and effort spent during driving daily, but with self-driving cars drivers can take over the whole driving task, letting drivers make use of their time. Also, self-driving cars could also come in handy in emergency situations. For example, if a driver lost consciousness, a vehicle equipped with self-driving technology could take them to safety.

Taxonomy of Driving Automation

It describes the level of automation in a driving system, there are some things we need to take into consideration while defining the taxonomy of self-driving car and the level of automation. The driver attention needed for example, does the driver need to keep attention on the steering wheel all the time? The driver action needed, for example does the driver need to steer? Does the driver need to control the speed? Or does the driver need to change the lanes or can the car stay in the current lane without any intervention? What exactly do we need to expect when we say that the car can drive autonomously? All these questions lead to the autonomous driving taxonomy. The categorization standards that we will discuss in this topic are be suggested by the Society of Automotive Engineers (SAE), but we need to describe the driving task before classifying the levels of automation.

The driving task consists of two main tasks, lateral control and longitudinal control. Lateral control which refers to steering and navigating laterally on the road, keeping a constant distances from the boundaries of the road. While longitudinal control is the task where we control the position and velocity of the car along the roadway, via throttle and brakes.

More tasks could be considered, like object and event detection and response (OEDR). OEDR is essentially the ability to detect objects and events that immediately affect the driving task and to react to them appropriately. Moreover, one more task to be considered is planning, which is primarily concerned with the long and short term plans needed to travel to a destination or execute maneuvers such as lane changes and intersection crossings. Some more miscellaneous tasks that people perform while driving can be considered as well. These include actions like signaling with indicators, interacting with other drivers, etc.

Levels of Automation

These levels are commonly-used to describe levels of driving automation, defined by the SAE Standard J3016 [3].

Level 0 – No Automation: It is a full human perception, planning and control. In this level, there is no driving automation whatsoever and everything is done by the driver.

Level 1 – Driving Assistance: In this level the autonomous system assists the driver by performing either lateral or longitudinal control tasks, either but not both.

For example, adaptive cruise control, in adaptive cruise control or ACC, the system can control the speed of the car, but it needs the driver to perform steering. So it can perform longitudinal control but needs the human to perform lateral control. Similarly, lane keeping assist systems, in lane keeping assistance, the system can help you stay within your lane and warn you when you are drifting towards the boundaries.

Level 2 – Partial Driving Automation: In this level the system performs both the control tasks, lateral and longitudinal in specific driving scenarios.

Some simple examples of level two features are GM Super Cruise, and Nissan's Pro Pilot Assist [4]. These can control both your lateral and longitudinal motion but the driver monitoring of the system is always required. Nowadays, many automotive manufacturers offer level two automation products including Mercedes, Audi, Tesla and Hyundai.

Level 3 – Conditional Driving Automation: In this level, the system can perform Object and Event Detection in Response to a certain degree in addition to the control tasks. However, in the case of failure the control must be taken up by the driver.

An example of level three systems, would be the Audi A Luxury Sedan, which was an automated driving system that can navigate unmonitored in slow traffic.

Level 4 – High Driving Automation: In this level, we arrive at highly automated vehicles, where the system is capable of reaching a minimum risk condition, in case the driver doesn't intervene in time for an emergency. Level four systems can handle emergencies on their own, but may still ask drivers to take over to avoid pulling over to the side of the road unnecessarily. With this amount of automation, the passengers can check their phone or watch a movie knowing that the system is able to handle emergencies and is capable of keeping the passengers safe. However, level four still permits self-driving systems with a limited operational design domain (ODD).

For example, as of fall 2018, only Waymo has deployed vehicles for public transport with this level of autonomy. The Waymo fleet [5] can handle the driving task in a defined geographic area with a nominal set of operating conditions, without the need for a human driver.

Level 5 – Full Driving Automation: In this level the system is fully autonomous and its ODD is unlimited. Meaning that it can operate under any condition necessary. Level five is the point where our society undergoes transformational change. With driverless taxis shuttling people in packages wherever we need them. Unfortunately, we don't have any examples for level five yet.

Autonomous Cars Industry

Self-driving cars will be without a doubt the standard way of transportation in the future. Major companies are willing to spend millions of dollars in their development, as its future market is predicted to worth trillions. Self-driving cars are now a feasible due to many different computational technological advancements. We introduce briefly some of the most important companies that had some major contribution in the field of autonomous driving and navigation.

CMU Navlab [6]: The Carnegie Mellon University Navigation Laboratory Navlab (CMU) group builds computer-controlled vehicles for automated and assisted driving. Since 1984, they have built a series of robot cars, vans, SUVs, and buses. Navlab11 [7], their latest vehicle in the Navlab family, a robot Jeep Wrangler equipped with a wide variety of sensors for short-range and mid-range obstacle detection.

Waymo's Self-Driving Cars: Waymo got started in 2009 as the Google Self-Driving Car Project [8] and was spun off as a subsidiary of Google's own parent company Alphabet Inc. in 2016. Waymo car is equipped with cameras to enhance the vision along with a LIDAR. LIDAR is technology that uses lasers to generate a highly accurate representation of the area around a self-driving car. Unlike a human driver, LIDAR is able to generate a complete 360 degree view of the world around the vehicle. A Waymo car is able to plot a route from one location to another and then react, in real time, to the flow of traffic, using the information from maps, the LIDAR system, and other sensors. Sensors designed to detect objects as far as three football fields away in all directions including pedestrians, cyclists and vehicles. Waymo had tested its system in six states and 25 cities across the U.S over a span of more than 9 years. Waymo announced that it has been running Level 4 autonomous cars, with no human behind the wheel.

Tesla Model S: Tesla started equipping Model S with hardware to allow for the incremental introduction of self-driving technology: a forward radar, a forward-looking camera, 12 long-range ultrasonic sensors positioned to sense 16 feet around the car in every direction at all speeds, and a high-precision digitally-controlled electric assist braking system. This combined suite of features represents the only fully integrated autopilot system involving four different feedback modules: camera, radar, ultrasonic, and GPS. These mutually reinforcing systems offer real-time data feedback from the Tesla fleet, ensuring that the system is continually learning and improving upon itself. Autopilot allows Model S to steer within a lane, change lanes with the simple tap of a turn signal, and manage speed by using active, traffic-aware cruise control. Digital control of motors, brakes, and steering helps avoid collisions from the front and sides, as well as preventing the car from wandering off the road.

Renault NEXT TWO [9]: Driverless operation is based on a system that monitor and analyze the vehicle's environment using the following: a radar fitted in the front bumper and a camera on the central rear-view mirror. The radar detects the vehicle in front and calculates its speed. The camera is used to correctly position the vehicle in its lane. The system also features an ultrasound belt wrapping around the vehicle. These monitoring systems are coordinated by a control unit that communicate the powertrain components and guard against contradictory instructions.

Mercedes-Benz: Mercedes S class has option for autonomous steering, lane keeping, acceleration/braking, accident avoidance, and driver fatigue detection. Partially automated driving is available to drivers of new

Mercedes-Benz E and S class models. The current S-Class falls under Level 2 autonomy. These systems can assist in steering, acceleration, and deceleration, but the driver remains in charge of monitoring the driving environment at all times. In 2020, Mercedes-Benz will step up to the plate with its own Level 3 technology for the next generation S-Class.

Objective

We propose an approach for driving autonomy using an end-to-end approach illustrated in figure 2, the main objective is to design and operate a vehicle controller on a hardware prototype of an autonomous vehicle, navigating from a starting point to a destination point, on a route defined by a higher level planner, using only visual data captured by an ordinary camera (frames) placed on the front of the vehicle, relying on data extracted only from simulated environments, interacting with its surrounding environment autonomously.

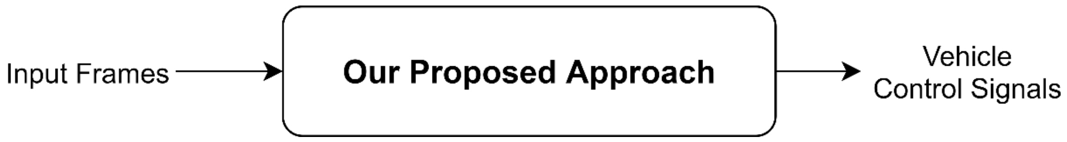


Figure 2. An illustration of our proposed end-to-end approach.

Chapter 2

Background

Machine Learning

Machine Learning (ML) is considered a subset of artificial intelligence (AI) which enables the system to automatically learn from experience and deal with new problem and tasks effectively without being explicitly programmed. The existence of complex tasks in the real world which we can't handle with traditional rule based programming accelerates the research in the area of machine learning to build a reliable system which is able to perform these complex tasks with high immunity to random possible variations. Machine Learning simply builds a mathematical model based on given information known as training data and use this model to preform predictions or decisions on relevant data that it hasn't been exposed to before.

History: In 1959, Arthur Samuel coined the term "Machine Learning" while at IBM and wrote the first computer learning program which was the game for checkers. As time passes, machine learning researches increased but considered only an application for artificial intelligence. In 1957, Frank Rosenblatt designed the first neural network for computers (the perceptron) simulating the process of a human brain. In 1990s machine learning recognized as a separate field and started to flourish. In 2000s with the huge computational technological advancements, more machine learning researches were done and machine learning becomes a trending topic in the research area.

Types of Machine Learning: The types of machine learning differ in their objective, inputs, outputs, and the approach to perform required tasks. We are going to cover the most used and essential types of machine learning briefly in the following subsections.

Supervised Learning: Supervised learning is the type of machine learning meant to map input data to output data. We build a machine learning model and train it using labeled data, it correlates the main features in the input data to the output labels, and gains the ability to perform future predictions on relevant new unseen inputs with high accuracy.

Supervised Learning has many types and approaches. The most well-known types are regression and classification. Regression is used to predict continuous values of outputs depending on the current input to the model with help of what we call hypothesis function as shown on figure 3. On the other hand, classification is being used to determine the category of specific input. Classification could be binary (categorize input into two types only), or Multi-class Classification (categorize input into multiple options) as shown in figure 4 [10].

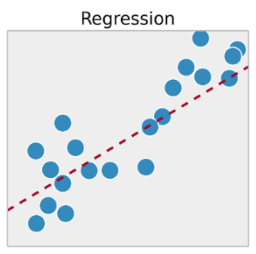


Figure 3. Linear regression illustration.

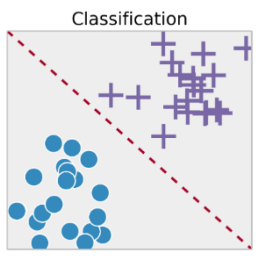


Figure 4. Logisticc regression illustration.

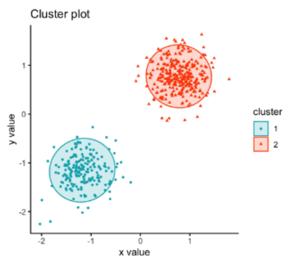


Figure 5. Clustering illustraion.

Unsupervised Learning: Unsupervised Learning is type of machine learning which can learn by itself without the need to labeled dataset. It searches for common correlations in given data, estimates a model that is able to analyze new and unseen input data. Unsupervised Learning is commonly used in clustering data into clusters which is determined without human interference based on the given unlabeled data as shown in figure 5 [11].

Reinforcement Learning: Reinforcement learning is the type of machine learning where the agent learns by himself without any given data as it learns from interaction with surrounding environment. Depending on the effects of specific actions the agent perform, feedback signals is sent to the agent to tell him how good/bad these actions were. Given appropriate amount of time, the agent will be able to learn patterns and logical triggers to his actions so that the least amount of negative feedback will be sent to him as shown in figure 6.

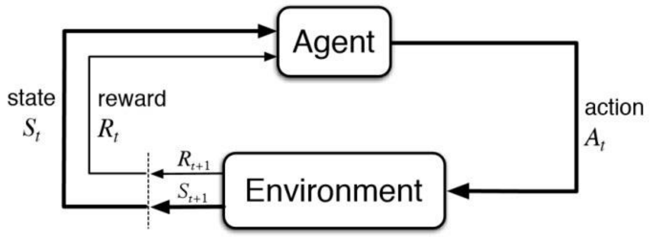


Figure 6. Reinforcement learning concept illustration.

Reinforcement learning, due to its generality, is studied in many other disciplines, such as game theory, control theory, and operations research.

Deep Learning

Deep Learning [12] is a class of machine learning which uses multiple layers to extract complex high dimensional features from raw input data. Deep learning is able to deal effectively with complex problems such as analysis of images, videos and time series events, taking into consideration spatial, temporal dependencies, or both. The term “deep” in deep learning refers to the number of layers through which the data is transformed. Deep learning methods can handle efficiently supervised learning problems, unsupervised learning problems as well as reinforcement learning problems.

Deep Learning Approaches: Deep Learning approaches is based mostly on artificial neural networks (ANN). Neural networks, in general, are built to simulate the behavior of the human brain– specifically, pattern recognition and the passage of input through various layers of simulated neural connections.

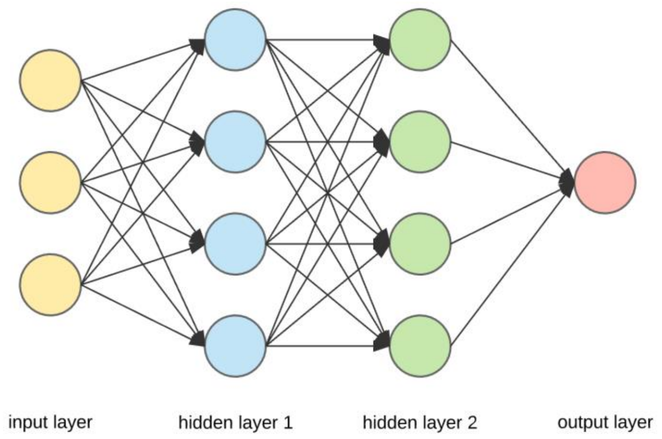


Figure 7. An illustration of a standard two layers neural network.

Neural networks are based on a collection of interconnected layers of nodes called perceptrons, responsible for processing information passing through different layers as shown in figure 7. Neural networks have many types based on the problem they are addressing. We are interested in the following three types;

Deep Neural Networks: A deep neural network (DNN) is a neural network with more than two hidden layers. As we increase the depth of the neural network, the ability to detect higher level features increases. The main advantage of DNNs to traditional machine learning approaches is that we don’t need to separately extract features from the raw input as the DNN can handle the task of feature extraction efficiently correlating the most affecting features to perform the required task. DNNs are trained using back propagation algorithm which is simply calculating the derivatives of a layer with respect to the previous layer starting from the

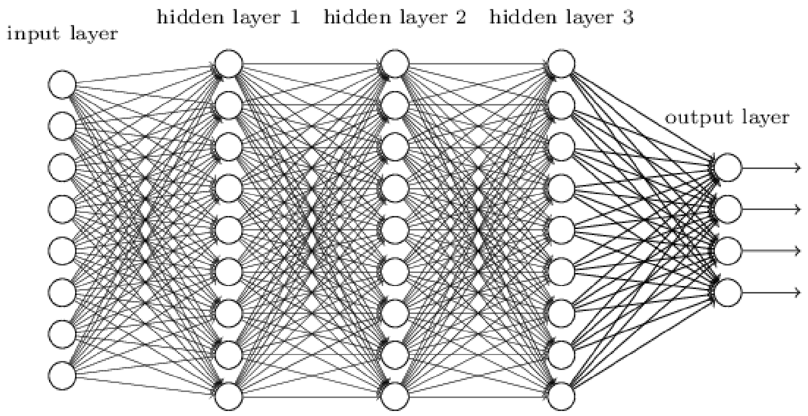


Figure 8. An illustration of multiple layers deep neural network.

output layer to the input layer. An illustration of a DNN structure is shown in figure 8.

Convolutional Neural Networks: A convolutional neural network (CNN) is a class of deep neural networks, most commonly applied to analyzing visual imagery. CNNs are regularized versions of multilayer perceptrons. Multilayer perceptrons usually refer to fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer. The "fully-connectedness" of these networks makes them prone to overfitting data. Typical ways of regularization include adding some form of magnitude measurement of weights to the loss function. However, CNNs take a different approach towards regularization: they take advantage of the hierarchical pattern in data and assemble more complex patterns using smaller and simpler patterns. Therefore, on the scale of connectedness and complexity, CNNs are on the lower extreme. They are also known as shift invariant or space invariant artificial neural networks

(SIANN), based on their shared-weights architecture and translation invariance characteristics [13][14].

Convolutional networks were inspired by biological processes [15][16][17][18] in that the connectivity pattern between neurons resembles the organization of the animal visual cortex. Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field. CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns the filters that in traditional algorithms were hand-engineered. This independence from prior knowledge and human effort in feature design is a major advantage. They have applications in image and video recognition, recommender systems [19], image classification, medical image analysis, and natural language processing [20]. A simple illustration of the general structure of a conventional CNN is shown in figure 9.

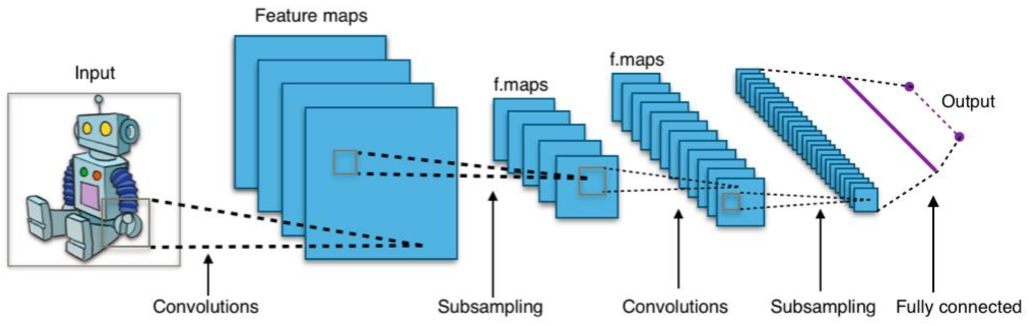


Figure 9. An illustration of a conventional convolutional neural network.

Recurrent Neural Networks: Recurrent neural network (RNN) is a class of neural networks which is used to analyze sequential data. The input to RNN is correlated with previous inputs forming a time series input which passes through the network affecting the final output as shown in figure 10. RNNs are suitable for problem with high temportal dependencies such as speech analysis, recognition, language models, machine translation, etc.

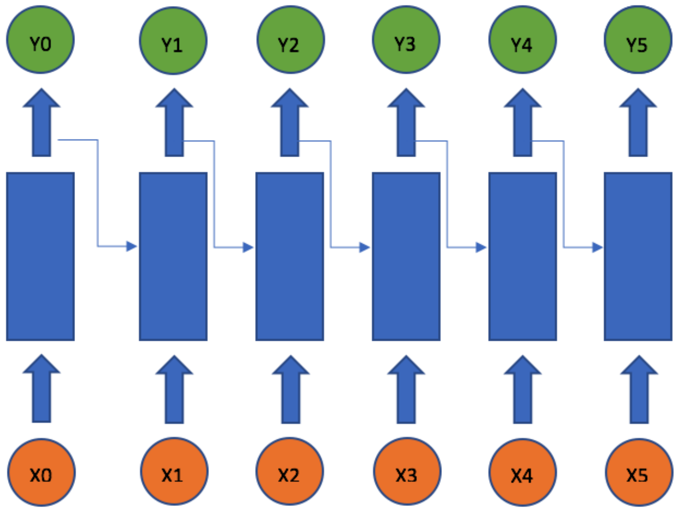


Figure 10. An unrolled illustration of a conventional RNN.

Deep Learning in Self Driving Cars

Self-Driving car is defined as a car which is capable of sensing and approximating its surrounding environment and navigating with little or no human interference. Deep learning had a major contribution in developing self-driving cars. Autonomous driving has two approaches to, either a hand-engineered modular pipelined approach, or an end-to-end deep learning based approach.

Modular Approach: The main idea of this approach is to split the task of autonomous driving into multiple modules performing smaller and specific tasks. Combing all these modules together gives the vehicle the ability to take decisions on its own without human interference. We are discussing briefly the main modules existing in this approach.

Localization: Localization means that the vehicle is able to detect its own position with very high accuracy. HD maps are used for localization with the help of GPS system.

Planning: Planning is meant to feed the vehicle with both the long term planning and short term planning. Planning module is important for the vehicle as it affective directly the behavior of the vehicle at every moment.

Perception: Perception module is the eyes for the vehicle. Several sensory data can be combined to provide a robust representation of the surrounding environment, like cameras, LIDAR, RADAR, and another sensors. CNNs are used in this module heavily to perform different tasks as lane detection, object detection and localization, and more.

Control: Given sensor data and planned trajectories, a control module is necessary to control the vehicle in a way that let it follow its trajectory as well as interacting with the surrounding environment accurately.

End-to-End Approach: End-to-End approach aims to eliminate any hand-engineered pipelining, unleashing the abilities of deep learning to form its own model of the environment, and an approximate robust relation between the surrounding environment and the corresponding control signals. The driving model learns from thousands of frames associated with control signals how to deal in different situations without the need to program it explicitly. The resulting driving policy of this approach is a replica of the driver's behavior existing in the provided training dataset.

Simulation

Simulation is an essential part of testing, evaluating and developing self-driving vehicles, because it allows us to ensure that our vehicle will operate safely before we even step foot in it, recently it has been useful for collecting data required for developing self-driving vehicles. Simulation has been used to verify that the vehicle's controller handles all various scenarios appropriately, most importantly, we can test our vehicle in situations that cannot be tested in real environment as it would be too dangerous to test on actual roads and very risky on the other drivers and pedestrians. Generally, to test the reliability of such a safety critical system, such as an end-to-end self-driving vehicles, it must first be tested and evaluated in a simulated environment.

There exists a number of simulators that are capable of testing and evaluating self-driving cars. The task is to find a suitable simulator that is compatible with our objective. The most important requirement is that the simulator provides a high-fidelity realistic driving environment. In the next subsections, we will discuss briefly several self-driving simulators.

AirSim: Airsim [21] is an open source simulation tool for drones, cars and other several vehicles, based on Unreal Engine 4 developed by Microsoft as a platform for AI research. AirSim provides realistic environments, vehicle dynamics, and multi-modal sensing for researchers building autonomous vehicles that use AI to enhance their safe operation in the open world. It provides realistic and high-fidelity environments that makes it a suitable visual and physical simulator for autonomous vehicles as shown in figure 11. AirSim is inspired by the goal of developing reinforcement learning algorithms for the autonomous agents that can operate in the real world.

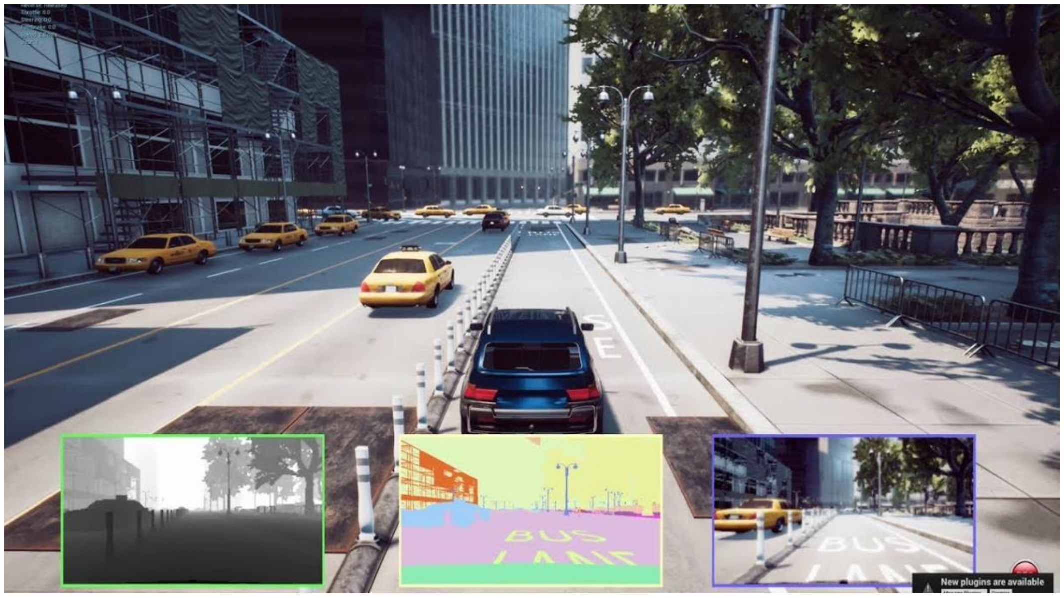


Figure 11. A screen capture of a scene taken from AirSim, showing several live sensor data it has to offer.

Gazebo Simulator: Gazebo [22] is an open source 3D dynamic multi-robot environment. Gazebo, is designed to accurately reproduce the dynamic environments a robot may encounter. Gazebo offers the ability to accurately and efficiently simulate populations of robots in complex indoor and outdoor customized environments. All simulated objects have mass, velocity, friction, and numerous other attributes that allow them to behave realistically when pushed, pulled, knocked over, or carried. The physics engine used in Gazebo is designed to simulate the dynamics associated with rigid bodies as shown in the simulation environment in Gazebo shown in figure 12.

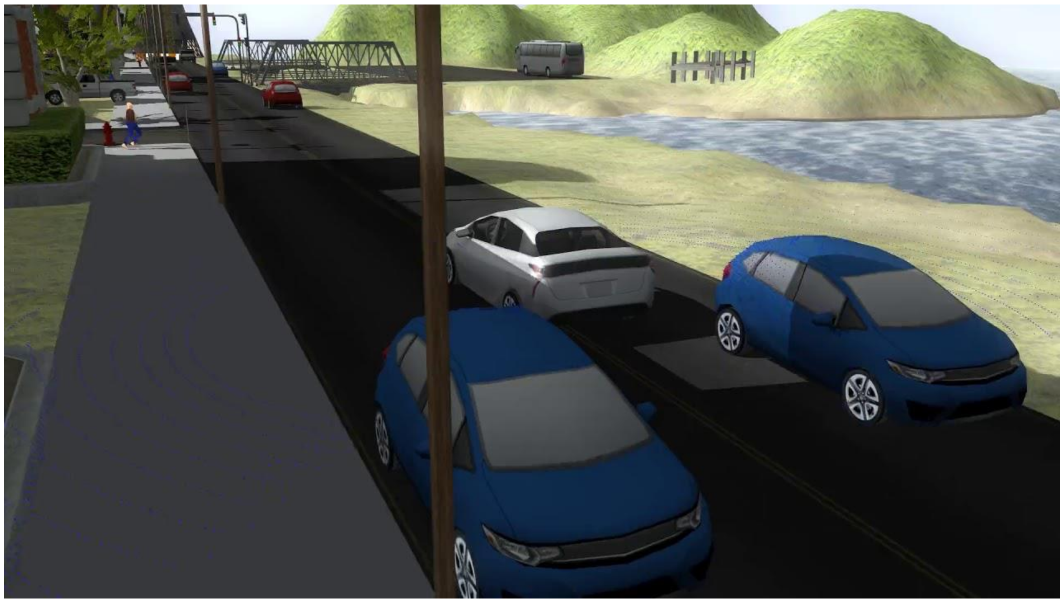
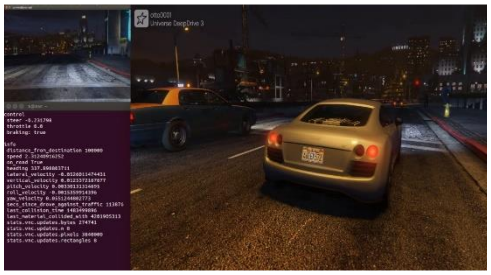


Figure 12. A scene from Gazebo simulator.

DeepDrive: DeepDrive [23] is an open sourced simulation tool for self-driving cars, based on Unreal Engine to be a platform for AI research. It features a tensorflow baseline agent and a self-driving benchmark. The simulation environment is shown in figure 13. OpenAI has enabled its DeepDive Project to transform GTA V into a self-driving car simulator. OpenAI has released the open source integration of artificial intelligence training software Universe within the DeepDive Project to bring GTA V to the self-driving world. The game provides artificial intelligence (AI) agents the access to a 3D world through Universe. The agents can watch the action live and examine the behavior of people within the GTA environment to develop an intelligent self-driving model. DeepDrive repurposes GTA V as a self-driving car simulator; it also provides pre-trained self-driving agents and the datasets used to train them. GTA V is an extensive, detailed world about a fifth of the size of Los Angeles, easily modified to any size or city as shown in figure 14. Not only does it have winding city streets, but also mountains, deserts, and highways that you can explore in 257 different cars, and it has 14 different weather simulations.



```
control
 steer -0.231798
 throttle 0.0
 braking: true

info
 distance_from_destination 100000
 speed 2.31240916252
 on_road True
 heading 337.898803711
 lateral_velocity -0.0526011474431
 vertical_velocity 0.0125372167677
 pitch_velocity 0.00330131314695
 roll_velocity -0.0015359914396
 yaw_velocity 0.0551244082773
 secs_since_drove_against_traffic 113876
 last_collision_time 1483499896
 last_material_collided_with 4201905313
 stats.vnc.updates.bytes 274741
 stats.vnc.updates.n 8
 stats.vnc.updates.pixels 3840000
 stats.vnc.updates.rectangles 8
```

Figure 13. A scene taken from DeepDrive.

Figure 14. A high definition detailed scene from DeepDrive.

CARLA Simulator: CARLA (Car Learning to Act) is an open source simulator developed to support autonomous vehicle research, in particular the development, training and validation of autonomous urban driving systems [24] that is founded by Intel. CARLA is able to simulate advanced and realistic weather conditions, urban environments and non-player characters. CARLA provides open digital assets (urban layouts, buildings, vehicles) that were created for this purpose and can be used freely. The simulation platform supports flexible specification of sensor suites, environmental conditions, and full control of all static and dynamic actors. The simulator is built with Python as a layer on top of Unreal Engine 4 (UE4). CARLA has several towns, urban traffic environments as shown in figure 15 wherein vehicles may be rendered and tested. CARLA is used to study the

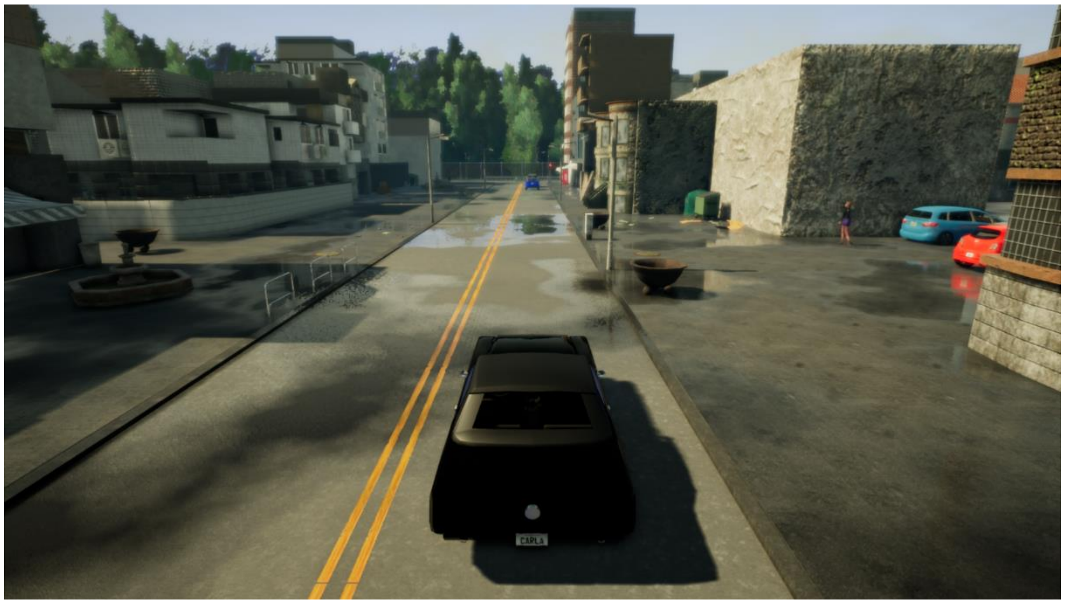


Figure 15. A standard scene from CARLA simulator.

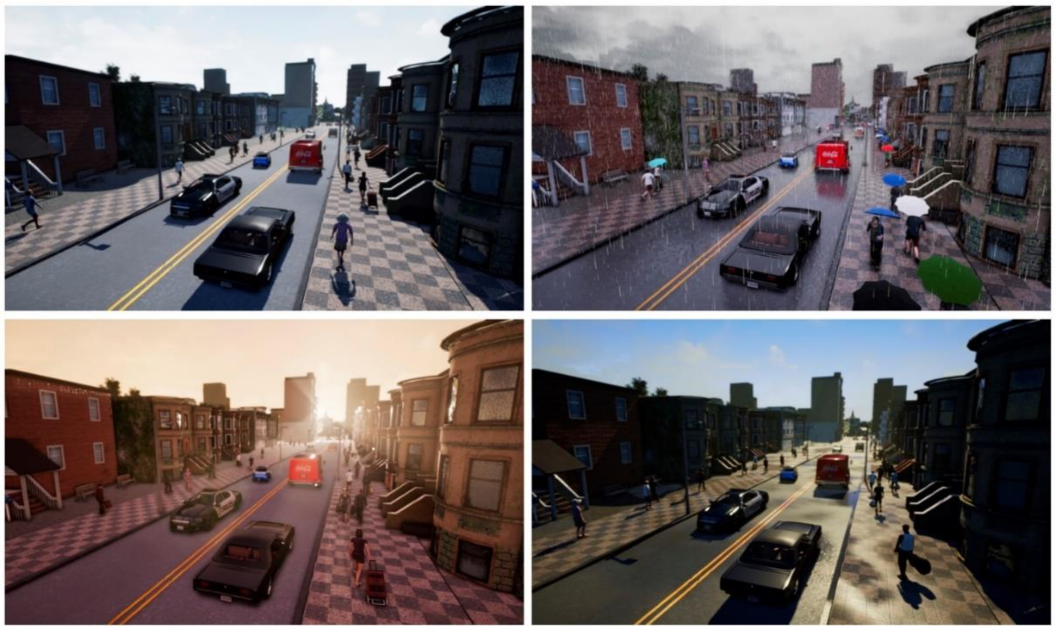


Figure 16. The same scene from CARLA simulator, taken from different daytimes and weather conditions.

performance of several approaches to autonomous driving. These approaches are evaluated in controlled scenarios of increasing difficulty, and their performance is examined via metrics provided by CARLA, illustrating the platform's utility for autonomous driving research. It includes urban layouts, a multitude of vehicle models, buildings, pedestrians, intersections, cross traffic, traffic rules, street signs, etc. A wide range of environmental conditions can be specified, including weather and time of day. A number of such environmental weather conditions are illustrated in figure 16. The environment of CARLA is composed of 3D models of static objects such as buildings, vegetation, traffic signs, and infrastructure, as well as dynamic objects such as vehicles and pedestrians. All models are carefully designed to reconcile visual quality and rendering speed. All 3D models share a common scale, and their sizes reflect those of real objects. CARLA 0.8.4 includes 40 different buildings, 16 animated vehicle models, and 50 animated pedestrian models. This way they have designed two different towns: Town 1 with a total of 2.9 km of drivable roads, used for training, and Town 2 with 1.4 km of drivable roads, used for testing that is shown in figure 17. CARLA allows flexible configuration of the agent's sensor suite. The version we used includes sensors that are limited to RGB cameras and to pseudo-sensors that provide ground-truth depth and semantic segmentation. It provides three sensing modalities, depth and semantic segmentation are pseudo-sensors that support experiments that control for the role of perception. Additional sensor models can be plugged in via the API. The number of cameras and their type and position can be specified by the client. Camera parameters include 3D location, 3D orientation with respect to the

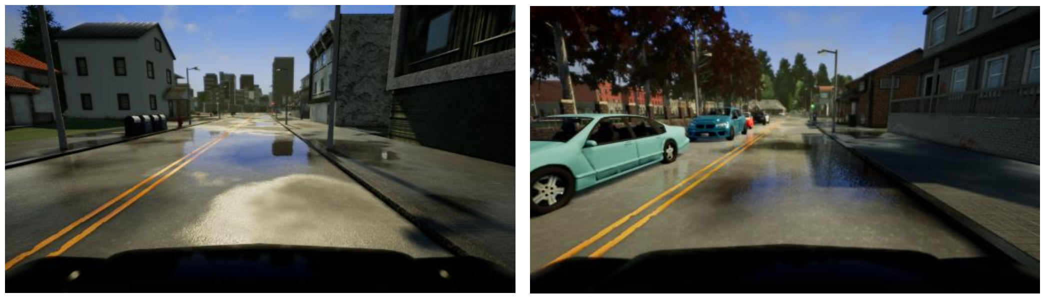


Figure 17, a scene from town 1 on the left, and another from town 2 on the right.

car's coordinate system, field of view, and depth of field.

Datasets

Deep learning mainly relies on huge amounts of training data. Getting the right data means gathering or identifying the input data that describes the environment sufficiently to be related to the outputs you want to predict; i.e. data that contains signals that describes events you care about. Some of the popular related datasets are listed below;

Berkeley DeepDrive BDD100k: This is one of the largest dataset for self-driving AI and contains over 100,000 videos of over 1,100-hour driving events across different times of the day, and varying weather conditions. The annotated images within the dataset come from New York and San Francisco areas.

Baidu ApolloScape: This large dataset defines 26 distinct semantic items such as cars, bicycles, pedestrians, street lights, etc. This dataset contains various categories of data set like scene parsing, car instance, lane segmentation, etc.

Comma.ai: This dataset contains more than seven hours of highway driving. The data set contains measurements like speed, acceleration, steering angles and GPS coordinates.

Oxford’s Robotic Car: This dataset contains over 100 repetitions of the same route through Oxford, UK, captured over one year with different combinations of weather, traffic and pedestrians, along with long-term changes such as construction and roadwork.

KUL Belgium Traffic Sign Dataset: This dataset contains 10,000+ annotations of traffic signs from thousands of physically distinct traffic signs in Belgium, particularly in the Flanders region. Belgium TSC dataset is built for traffic sign classification purposes. Is a subset of Belgium TS dataset and contains cropped images around annotations for 62 different classes of traffic signs. Belgium TSC is split in a training part with 4591 images and a testing part with 2534 images.

LISA: Laboratory for Intelligent & Safe Automobiles, UC San Diego Datasets: This dataset includes LISA Vehicle Detection Dataset, LISA Traffic Sign Dataset, and LISA Traffic Light Dataset.

KITTI Dataset: The KITTI dataset has been recorded from a moving platform while driving in and around Karlsruhe, Germany .It includes camera images, laser scans, high-precision GPS measurements and IMU accelerations from a combined GPS/IMU system.

CityScapes: This dataset focuses on semantic understanding of urban street scenes, from 50 different cities, during all different seasons, on different daytimes, several weather conditions, and varying scene layout. It contains 5000 frames, with 30 different segmentation classes annotated finely and coarsely.

But using a real dataset didn’t meet our requirements, and all of them were taken from a set of environments completely different than ours, moreover, collecting a dataset using a fleet of real vehicles on real roads can be time consuming, expensive, and very risky on other drivers and pedestrians, on the other hand, a dataset collected completely from a simulated environment is almost cost-free, risk-free, and completely customizable to include all different scenarios.

The dataset that we used to train our driving model was extracted by Intel Labs from CARLA Simulator

- The dataset contains 600,000 frames of RGB image stored at a resolution of 200x88.
- Each frame has its corresponding 28 measurements for like steering, gas, brake, speed, acceleration, etc.

Chapter 3

Methodology

Imitation Learning

A basic imitation learning [25] consists of discrete environment observations o_t and actions a_t, in our case these observations are raw simulated images (RGB) taken from a camera placed on our simulated vehicle, and these actions are performed by an expert driver that could be a human expert or a rule based autopilot. The goal is to train a controller that mimics the expert's actions on the training data and beyond in familiar –but not similar– data. The simulated data generated by the expert that is supposed to train the controller is a set of observation-action pairs as follows;

$$\beta = \{\langle o_i, a_i \rangle\}_{i=1}^{N}$$

The basic assumption that the expert is performing the tasks of interests perfectly, which are; navigating in straight urban two-lane roads, taking left and right turns, taking decisions based on the higher level command provided by a higher level planner on intersections, and interacting with other vehicles and pedestrians. This is a supervised learning problem, which the controller's parameters $\mathcal{W}$ of a function approximator $F(o; \mathcal{W})$ must be optimized to fit the mapping of the observations o_t to the actions a_t as follows;

$$\underset{\mathcal{W}}{minimize} \sum_i \ell(F(o_i, \mathcal{W}), a_i)$$

Moreover, the expert's actions in our case are logical and explained by the observations, this can be represented as a function E that maps the controller's actions a_t to the observations as follows;

$$a_i = E(o_i)$$

An approximator will be able to fit sufficiently given enough data as long as the expert's actions are explained enough by the observations, this condition will exist in certain simple tasks like lane following and simple turns. Considering more complex tasks and scenarios, this condition will break down immediately, like intersections for example, the action taken by the expert can never be explained by the observation as it depends on a higher level decision, which is in our case called the higher level command that is provided by the higher level planner. The same observations could lead to totally different actions depending on the higher level command or the defined route in general. Therefore if a controller or a model is trained without considering that higher level command, it would take an arbitrary decision in each intersection, which results in a non-useful system in the defined urban environment. A system like this will have no communication with the higher level planner to define a route at all.

Therefore, the higher level command must be included explicitly along with the observations in the function E to map the actions a_t with the observations o_t and the higher level command c_t simultaneously as follows;

$$a_i = E(o_i, c_i)$$

The learning objective now can be reformulated to include the higher level command c_t as follows;

$$\underset{\mathcal{W}}{\text{minimize}} \sum_i \ell(F(o_i, \mathcal{W}), E(o_i, c_i))$$

Therefore, the dataset becomes triplets of observations, actions, and higher level commands as follows;

$$\beta = \{\langle o_i, c_i, a_i \rangle\}_{i=1}^{N}$$

And the final general learning objective becomes as follows;

$$\underset{\mathcal{W}}{\text{minimize}} \sum_i \ell(F(o_i, c_i, \mathcal{W}), a_i)$$

Afterwards, the controller-environment relation in association with the higher level command can be represented as follows in figure 18.

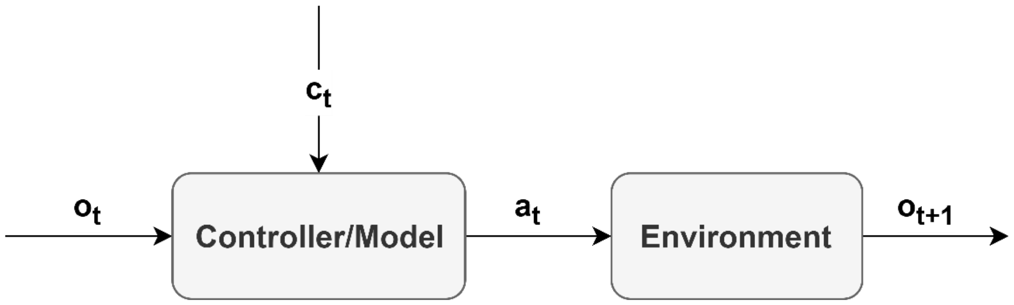


Figure 18. An illustration of our controller-environment interactions.

Model Architecture

In our case, our observations o_t includes images and low dimensional measurement vectors, and the controller *F* is a deep neural network taking o_t as an input alongside with the higher level command c_t and predicts the suitable action a_t. The actions space could be continuous or discrete according to the nature of each action. In our case, the actions a_t consists of three main signals, steering which is continuous, throttle which is discrete, and brakes which is also discrete, while the higher level command c_t is a categorical variable indicating which direction to take, either left, right, straight, or follow-lane that is represented by a one-hot vector. Through the simulation phase the input image is raw RGB without any preprocessing.

We addressed the problem of embedding the higher level command into the learning objective through two different approaches with two different architectures. Also the convolutional neural network (CNN) used in both architecture was strongly inspired by NVIDIA's PilotNet [26], modified and embedded in a larger network that is inspired by CoIL [27], that will be discussed in detail through this section. All our experiments and trials were implemented with Keras using Tensorflow [28] Backend.

Architecture #1

This architecture is illustrated in figure 19, and table 1. The model takes the images alongside with the measurements vector and the higher level command as inputs, and all three of them are processed independently. The image is processed by a CNN while the measurements vector and the higher level command are processed by two separated fully connected networks (FCN), and their outputs are concatenated to form a joint vector that is then processed by another FCN called the control FCN to output the steering, throttle, and brakes signals all at once.

This architecture aims to a learning objective that considers the higher level command as a trainable input as follows;

$$\underset{\mathcal{W}}{minimize} \sum_i \ell(\mathcal{F}(o_i, c_i, \mathcal{W}), a_i)$$

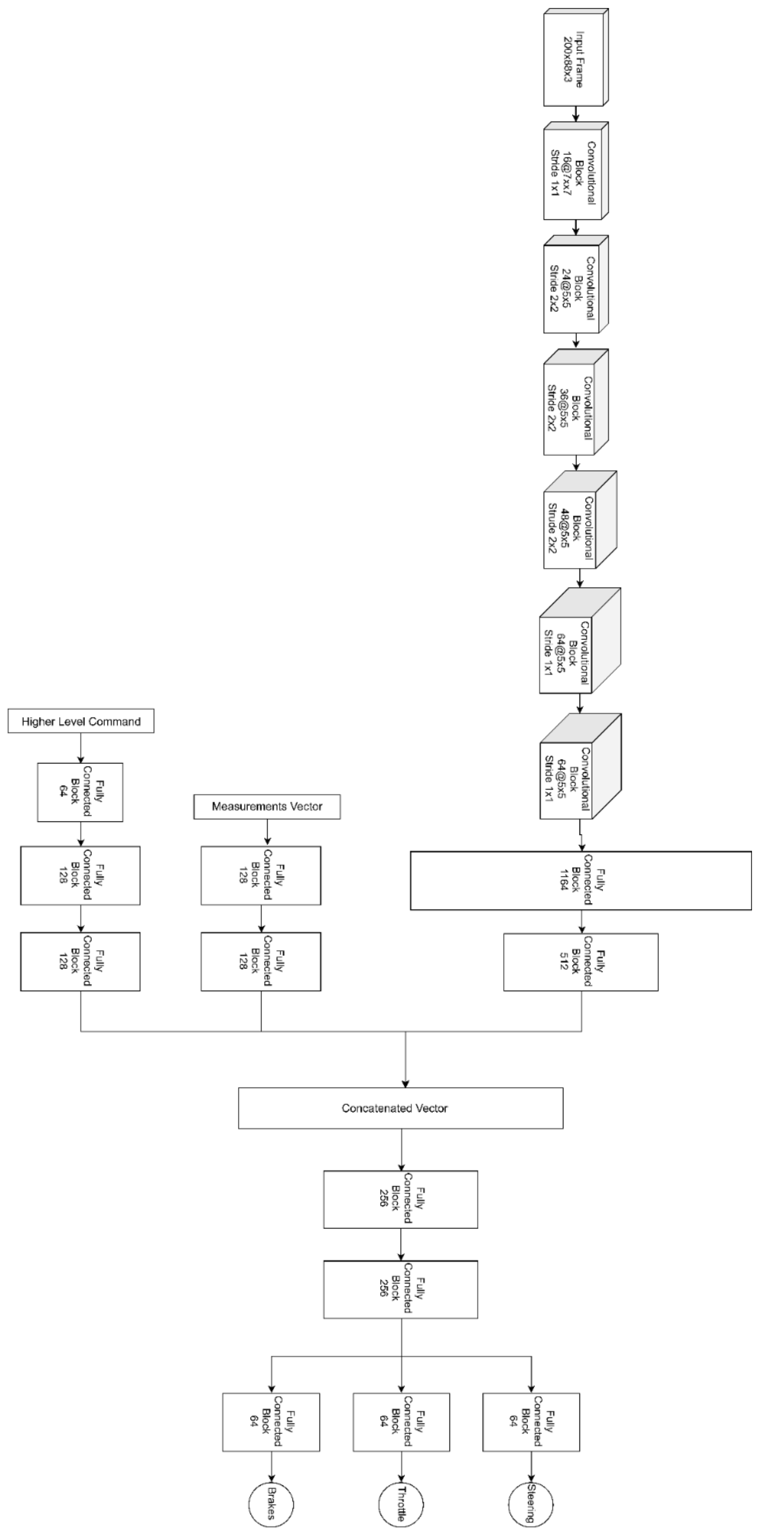

Input Frame 200x88x3
Convolutional Block 16@7xx7 Stride 1x1
Convolutional Block 24@5x5 Stride 2x2
Convolutional Block 36@5x5 Stride 2x2
Convolutional Block 48@5x5 Stride 2x2
Convolutional Block 64@5x5 Stride 1x1
Convolutional Block 64@5x5 Stride 1x1
Fully Connected Block 1164
Fully Connected Block 512
Higher Level Command
Fully Connected Block 64
Fully Connected Block 128
Fully Connected Block 128
Measurements Vector
Fully Connected Block 128
Fully Connected Block 128
Concatenated Vector
Fully Connected Block 256
Fully Connected Block 256
Fully Connected Block 64
Fully Connected Block 64
Fully Connected Block 64
Brakes
Throttle
Steering

Figure 19. A detailed figure of our command-input architecture.

Table 1. A detailed description of our command-input architecture.

Input	Layer/Block	Input Shape	Output Shape	Number of Parameters
Frame	Convolutional Block 1	200x88x3	200x88x16	2,444
	Convolutional Block 2	200x88x16	98x48x24	9,720
	Convolutional Block 3	98x48x24	47x19x36	21,780
	Convolutional Block 4	47x19x36	22x8x48	43,440
	Convolutional Block 5	22x8x48	20x6x64	27,968
	Convolutional Block 6	20x6x64	18x4x64	37,184
	Fully Connected Block 1	4,608	1164	5,369,532
	Fully Connected Block 2	1164	512	598,528
Speed Measurement	Fully Connected Block 3	1	128	768
	Fully Connected Block 4	128	128	17,027
Concatenated Vector	Fully Connected 5	640	256	165,120
	Fully Connected 6	256	256	66,816
	Fully Connected 7	256	64	16,704
	Fully Connected 8	64	1	65
	Total Number of parameters			6,410,631

Architecture #2

This approach is illustrated in figure 20, and table 2. The model in this approach takes the image alongside with the measurements vector as inputs, without the higher level command, similarly, they are processed independently as in approach 1. The main difference that the higher level command is no longer a trainable input, instead it will be used to influence the control FCN that processes the joint vector that results of concatenating the CNN's output with the measurements FCN's output in a special manner. There will be certain number of control FCNs that takes the joint vector and outputs the output signals, that number depends on the number of directions or categories that the higher level planner is intended to produce, in our case as mentioned before there is only 4 possible values for that higher level command c_t, therefore there will be 4 control FCNs or 4 control branches processing the joint vector, each for every direction.

This architecture can be modeled as having 4 experts willing to driving all the time, and are able to interact with the environment sufficiently all the time except at the intersections, one of them for example tends to take the left exit of the intersection, called after the higher level command direction; the left head, and similarly for the right and the straight head, while the follow-lane head is an expert in driving and turning in ordinary situations without intersections, and it is the one that controls the vehicle most of the time.

This architecture aims to a learning objective that doesn't include the higher level command as a trainable input at all, as follows;

$$\underset{\mathcal{W}}{minimize} \sum_i \ell(\mathcal{F}(o_i, \mathcal{W}), a_i)$$

But, it does affect the whole model, as the inputs are fed forward into the model including all the branches or heads, the higher level command selects which branch or head to backpropagate through during training time. The resulting model after training should have a CNN that is trained on all the training dataset including all branches, a measurements FCN that is also trained on all the training dataset, and 4 branches, each trained on its corresponding subset of the training dataset according to the higher level command, but all of them share the same CNN and measurements FCN.

This model architecture was trained in a very special manner, as a custom loss function was used to guide the backpropagation in the branch corresponding to each direction or higher level command, also the training process itself needed to be performed carefully to avoid domination of a certain branch over the others.

An extra branch was added called the speed auxiliary branch, it is responsible of predicting the corresponding speed from the output of the CNN only, although the actual speed is included as an input in the measurements vector, but the predicted speed is drawn from an early stage in the network that hasn't been exposed to the input one yet, this connection forced an implicit correlation between the input frame and the output speed that could be used later to add more reliability to the overall model while driving.

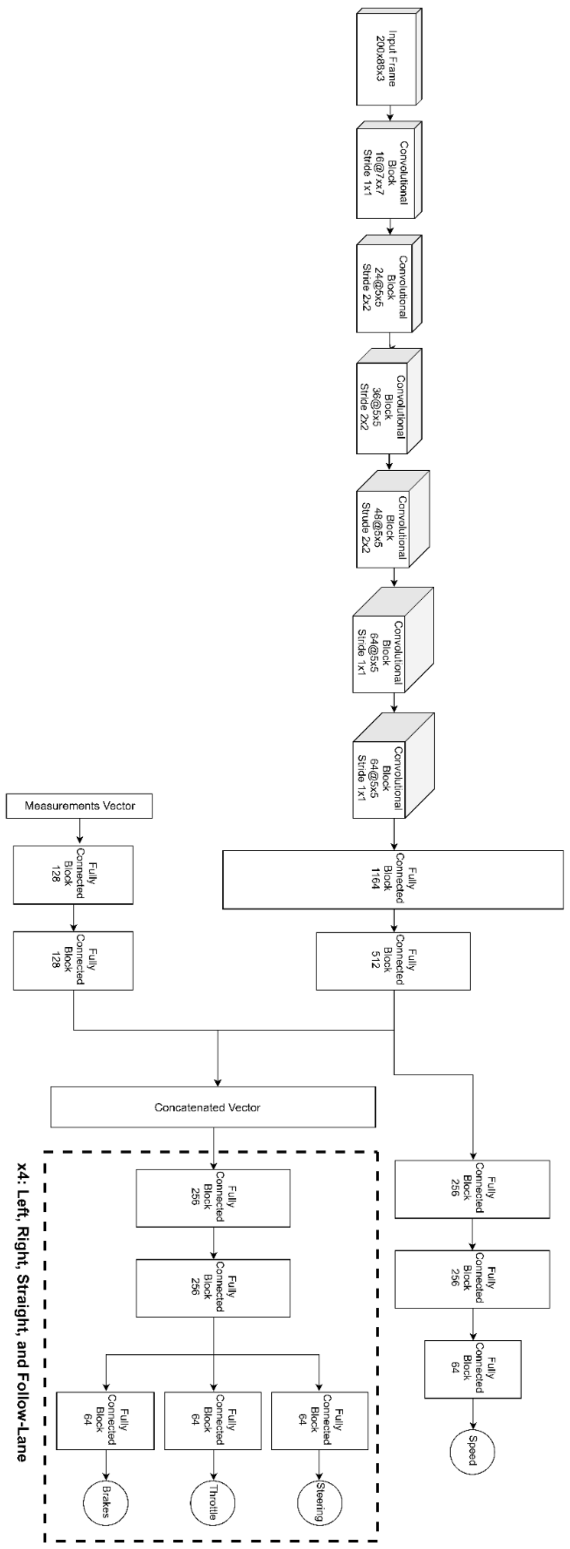

Input Frame 200x88x3
Convolutional Block 16@7xx7 Stride 1x1
Convolutional Block 24@5x5 Stride 2x2
Convolutional Block 36@5x5 Stride 2x2
Convolutional Block 48@5x5 Strude 2x2
Convolutional Block 64@5x5 Stride 1x1
Convolutional Block 64@5x5 Stride 1x1
Measurements Vector
Fully Connected Block 128
Fully Connected Block 128
Fully Connected Block 1164
Fully Connected Block 512
Concatenated Vector
x4: Left, Right, Straight, and Follow-Lane
Fully Connected Block 256
Fully Connected Block 256
Fully Connected Block 64
Fully Connected Block 64
Fully Connected Block 64
Brakes
Throttle
Steering
Fully Connected Block 256
Fully Connected Block 256
Fully Connected Block 64
Speed

Figure 20. A detailed figure of our command-input architecture.

Table 2. A detailed description of our branched architecture.

Input	Layer/Block	Input Shape	Output Shape	Number of Parameters
Frame	Convolutional Block 1	200x88x3	200x88x16	2,444
	Convolutional Block 2	200x88x16	98x48x24	9,720
	Convolutional Block 3	98x48x24	47x19x36	21,780
	Convolutional Block 4	47x19x36	22x8x48	43,440
	Convolutional Block 5	22x8x48	20x6x64	27,968
	Convolutional Block 6	20x6x64	18x4x64	37,184
	Fully Connected Block 1	4,608	1164	5,369,532
	Fully Connected Block 2	1164	512	598,528
Speed Measurement	Fully Connected Block 3	1	128	768
	Fully Connected Block 4	128	128	17,027
Concatenated Vector	Fully Connected 5	640	256	165,120
	Fully Connected 6	256	256	66,816
	Fully Connected 7	256	64	16,704
	Fully Connected 8	64	1	65
	Total Number of parameters			7,456,785

The convolutional block mentioned in both architectures in figures 20 and 21 consists of a 2D convolutional layer, followed by a relu activation, batch normalization, and dropout, the batch normalization-dropout order was strictly followed to avoid variance shift and to make use of their advantages combined. An illustration showing the operations' order is shown below in figure 21. Similarly, the fully connected block has the same structure, but a fully connected layer instead of a convolutional layer, as shown also in figure 21.

The output nodes had different activations according to their function; the steering output node had a tanh activation function as the steering angle was normalized to have values ranging from -1 to 1, and was considered a regression output with mean absolute error loss function, while the throttle and the brakes output nodes had sigmoid activation functions as their values ranged only from 0 to 1, also with a mean absolute error functions as well.

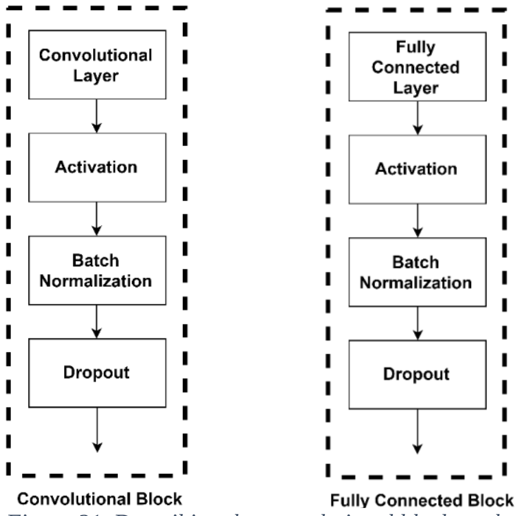


Figure 21. Describing the convolutional block on the left, and the fully connected block on the right

Training

Due to the differences between both architectures, the training phase was completely different for each, as the second one had very special requirements that had to be handled carefully in order to train the 4 branches simultaneously. Both of them at the end of the training phase had reasonable losses;

- $\text{Loss}_{\textbf{steering}}$ = 0.015 ~ 0.030
- $\text{Loss}_{\textbf{throttle}}$ = 0.050 ~ 0.100
- $\text{Loss}_{\textbf{brakes}}$ = 0.050 ~ 0.100

Although all losses were relatively small, but the most expressive and important one is the steering loss, as it varies rapidly and frequently, while the throttle and the brakes in the training dataset were almost fixed to certain values most of the time, so their losses weren't as expressive as the steering loss.

Architecture #1

Training this model was completely straight forward, the training dataset was fed to the network in batches of size 32 samples, completely shuffled and random, including the higher level command. The training dataset was balanced to reach 600,000 samples, 150,000 samples for each direction, then the model was trained and evaluated after 20~25 epochs. The final loss values on the end of the training phase were on average close to the values mentioned before, but it doesn't define how good the model is in its task.

Architecture #2

This model had to be trained carefully as it have 4 control branches sharing the same CNN mainly. After various experiments, hyperparameters tuning, and training methods, the best way to train this model is by training it with batches of random samples of each direction separately, a batch of size 32 samples from the left direction for the left branch, followed by a batch of the same size from right direction for the right branch, while the speed branch being fed all the training data regardless of the higher level command, and so on till the whole dataset is fed to the model.

The training phase sequence is illustrated in figures 22 and 23, indicating which parts of the model were trained using each subset of the training dataset.

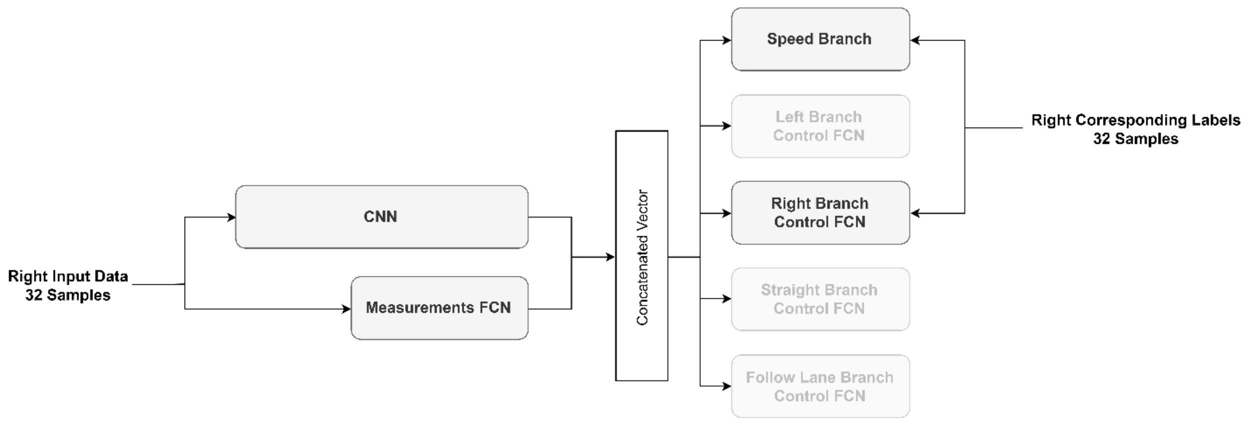


Figure 23. An illustration showing the right branch training batch.

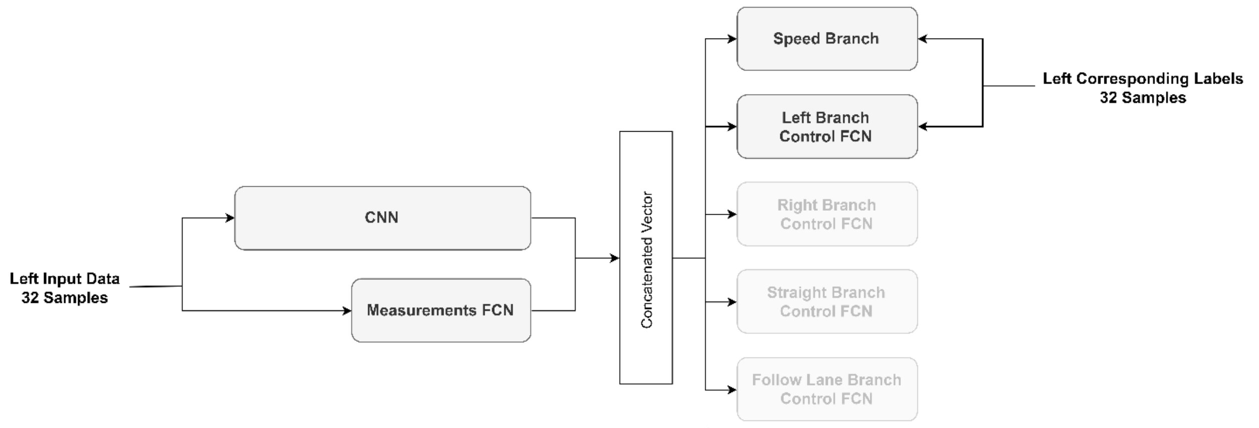


Figure 22. An illustration showing the left branch training batch.

Evaluation

Evaluating the models' performance in such task included inspecting their performance on the test set, but it wasn't enough to measure how good either of them, therefore an evaluating metric had been defined to do so, which is the success rate, given a total number of defined trips between two arbitrary points on the simulator's map, the higher level planner defined the shortest path as a sequence of higher level commands, afterwards the trained model was left to control the vehicle according to its visual view and the higher level command, the success rate is the number of the successful trips it performed to the total number of defined trips.

The first model wasn't very promising at all, it successfully performed the basic task of driving in straight roads and taking simple turns, but the higher level command had no influence at all on its decision at the intersections, taking arbitrary decisions that made it fail to reach its defined destination.

On the other hand, the second model performed much better than the first, and in order to improve this model, we had to understand and inspect various internal properties after training, like visualizing internal activations [29] and saliency maps [30]. The internal activations following the convolutional blocks after training showed exactly which features did the model rely on in predicting its final output, illustrated in figure 24. Afterwards, for further understanding of how the model predict its outputs, the saliency maps were carefully inspected, we observed that on normal conditions on straight road segments, that the model focused and concentrated on the center of the image, as shown in figure 25.

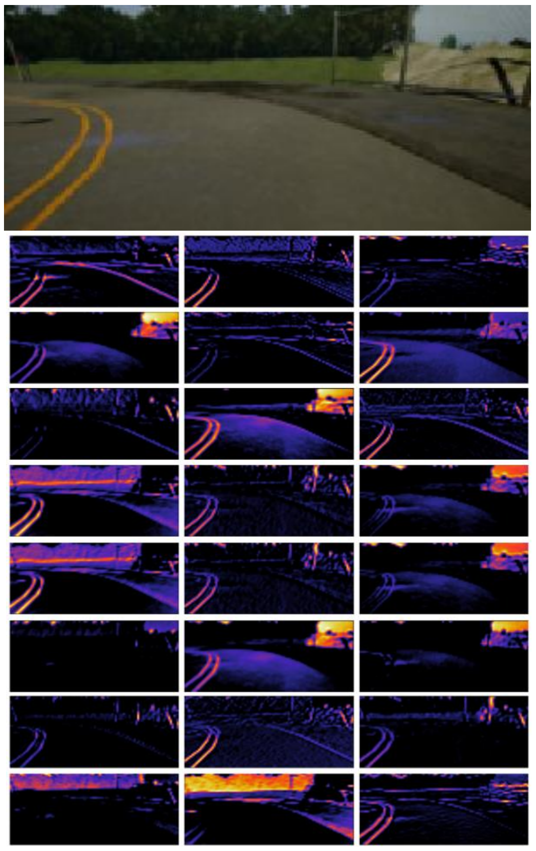


Figure 24. Showing an input frame on top, followed by its corresponding activations after the first convolutional block, after the training phase, showing how much the network extracted the importance of the road boundaries and lanes implicitly.

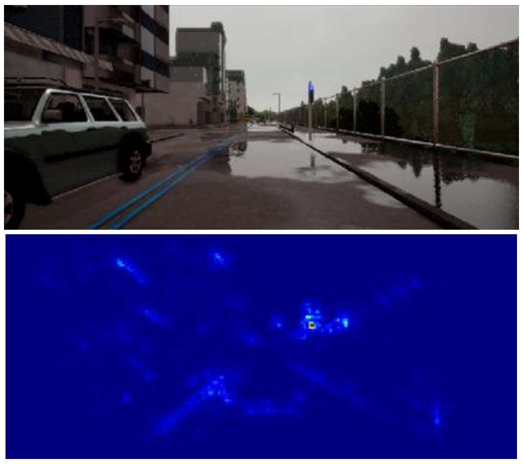


Figure 25. Showing an input frame on top, followed by its corresponding saliency map, showing how much the model learned to focus on the centerof the scene.

Path Planning

In order to generate the higher level command we mentioned, we had implement a routing algorithm that is able to find a suitable route between our starting point and destination point, and then convert this route into a sequence of higher level commands.

The routing module's input is a source point in (latitude, longitude) co-ordinates, as well as a destination point in the same co-ordinates, and a grapph-based map of our campus that we managed to find on OpenStreetMaps, the routing module finds the shortest path between the two points as a sequence of co-ordinates, which we are then converted to a new approximate (x, y) co-ordinate system. Using the new (x, y) co-ordinates, we reformulated the problem. We need to find a sequence of higher level commands (left, right, forward, follow-lane). We managed to do so by assuming a vector normal to the ground, and by taking the cross product of a vector from the previous point till the current point, with a vector from the previous point till the next point, then taking the dot product of the resultant vector of the cross product with the one normal to the ground, we had a value which indicates whether the next point falls on the left or the right of the currunt point. Moreover, a threshold was sat by try and error to compromise the road's bending and turns to generate the forward command.

Challenges

Transferring the learned driving policy from the simulated environment to a real environment is the main objective from the beginning, but due to the huge difference between the two domains, as shown in figure 26, the trained CNN failed completely to generalize and extract the features required for the latter parts of the network.

To address this problem we had to transform either domains to the other or find an intermediate domain that both can reach, afterwards the driving policy can be learned on that domain and abstracted to be used on both of them similarly without fine-tuning.

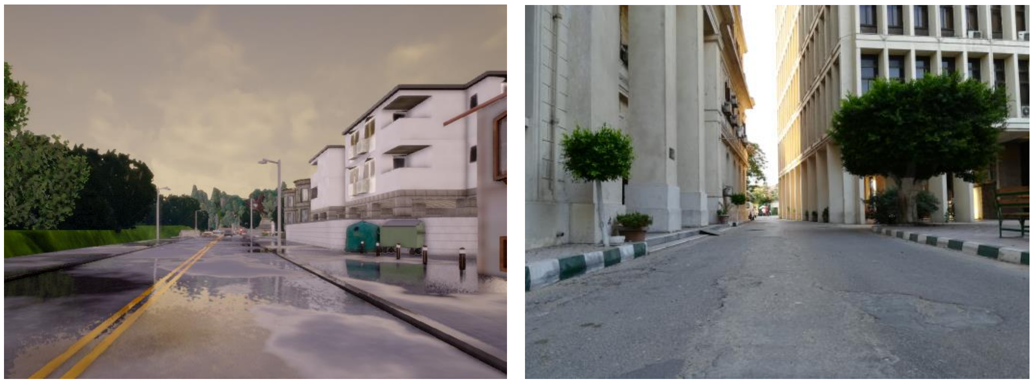


Figure 26. Showing the difference between our simulated environment on the left, and our real environment on our campus on the right.

Domain Transformation and Semantic Segmentation

To transfer the learned driving policy from the simulated domain to the real domain, one of these two solutions must be followed in order to narrow the gap between the two domains; either domain transformation or semantic segmentation. Both approaches offer immunity to the scene structure in both domains, like the layout of roads, buildings, cars, pedestrians, surface appearance (materials, lighting), and the properties of the camera. The selected approach will be used as a perception module, and its role is to filter out nuisance factors and preserve the information needed for planning and control.

Domain Transformation

Regarding domain transformation, using generative models, the simulated domain can be converted to the real domain, and vice versa, as shown in figure 27, and several approaches in the literature have done this task accurately, like DLID [31], CADA [32], CyCADA [33], but their major drawback is their processing time, training time, and training possible instability. Therefore, we haven’t selected domain transformation to solve our problem.

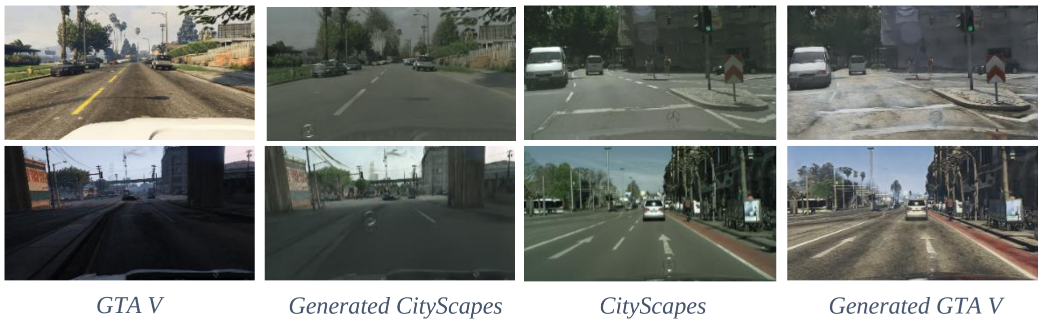


Figure 27. The image pairs on the left taken from a simulated domain, and converted to a real one, while the pair on the right taken from a real domain, and converted to a simulated one.

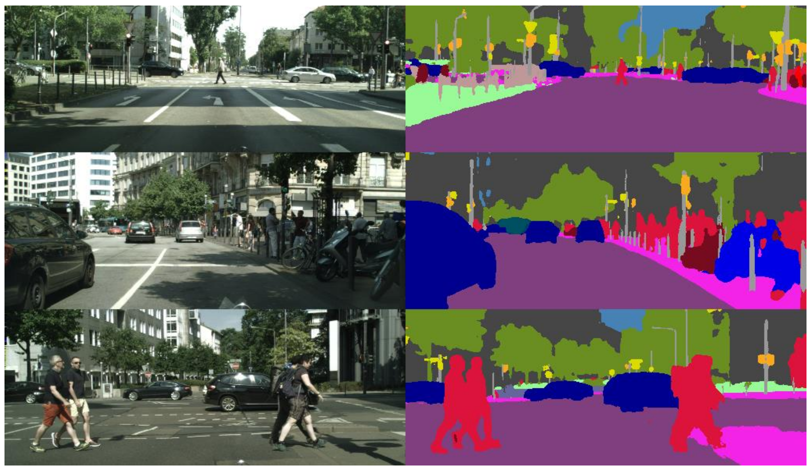


Figure 28. Per-pixel semantic segmentation.

Semantic Segmentation

On the other hand, semantic segmentation is the task of classifying each and every pixel in an image to a class as shown in figure 28. Here you can see that pedestrians are red, the road is purple, the vehicles are blue, street signs are yellow, etc. Semantic segmentation is different from instance segmentation, which is that different objects of the same class will have different labels and identites.

Using per-pixel binary segmentation of the image into “road” and “no road” regions. It abstracts away texture, lighting, shading, and weather, leaving only a few factors of variation: the geometry of the road, the camera pose, and the shape of the objects occupying the road. Such segmentation contains sufficient information for following the road and taking turns, and it abstracts enough to support transfer. Moreover, binary segmentation takes far less processing time, and much more stable in training. Therefore, we adopted binary segmentation as a perception module in order to transfer the driving policy to the real environment.

After reviewing several semantic segmentation models in the literature, we implement the perception module with an encoder-decoder deep convolutional neural network like ENet [34], PSPNet [35], and ERFNet [36], we settled on using ERFNet, although it wasn’t the fastest choice in terms of processing time, it was the most generalized one, as it was trained on real data, and generalizes on simulated data [37], which fits our purpose exactly. The model architecture details are shown in table 3, and implemented with PyTorch [38].

The training dataset for ERFNet we chose was CityScapes’ standard dataset, taken from several cities around Germany, with a total number of 5,000 labeled frames, we edited the whole dataset to remove all the labels except the road for our binary segmentation model. Moreover, we also had to edit ERFNet to fit our purpose for binary segmentation instead of semantic segmentation.

After training ERFNet on CityScapes' dataset, it was able to generalize and perform well on both on simulated data, and real data, from our campus as well as it does on CityScapes' test set, as shown in figures 29 and 30.

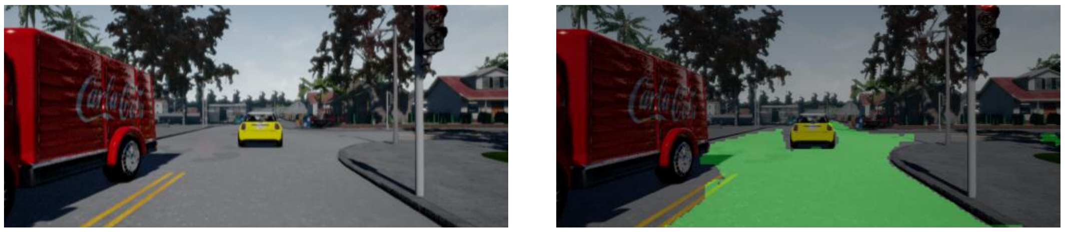


Figure 29. The simulated input frame to our perception module shown on top, followed by the predicted per-pixel binary segmentation, with "road" labels illustrated in green, and "no road" otherwise.

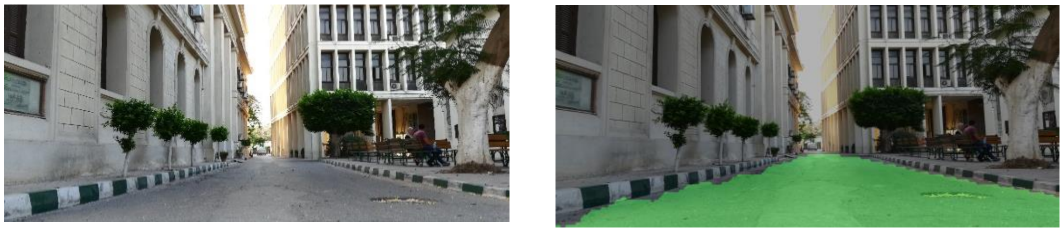


Figure 30. The r input frame to our perception module shown on top, followed by the predicted per-pixel binary segmentation, with "road" labels illustrated in green, and "no road" otherwise.

Layer	Type	Out Channels	Out Resolution
1	Downsampler Block	16	512 x 256
2	Downsampler Block	64	256 x 128
3-7	5 x Non-bt-1D	64	256 x 128
8	Downsampler Block	128	128 x 64
9	Non-bt-1D (dilated 2)	128	128 x 64
10	Non-bt-1D (dilated 4)	128	128 x 64
11	Non-bt-1D (dilated 8)	128	128 x 64
12	Non-bt-1D (dilated 16)	128	128 x 64
13	Non-bt-1D (dilated 2)	128	128 x 64
14	Non-bt-1D (dilated 4)	128	128 x 64
15	Non-bt-1D (dilated 8)	128	128 x 64
16	Non-bt-1D (dilated 16)	128	128 x 64
17	Deconvolution	64	256 x 128
18-19	2 x Non-bt-1D	64	256 x 128
20	Deconvolution	16	512 x 256
21-22	2 x Non-bt-1D	16	512 x 256
23	Deconvolution	2	1024 x 512

Table 3

Simulation to Reality

The main challenge in our case is how our trained driving model will react when performing in a real environment; a whole new environment and perceptual distribution. Our purpose is with minimum fine tuning, we could successfully deploy our model in the real world, bridge the gap between the simulated domain and the real domain, and drive smoothly with no need for collecting more data from the new target environment to fine-tune the driving model. We were able to do so using semantic segmentation, which eliminates unnecessary visual details and unifies the two domains.

In this section, we transfer a driving policy trained in simulated environment to a a real-world envrionment, with no fine tuning of the driving model. Our final pipeline we need to do so, is illustrated in figure 31.

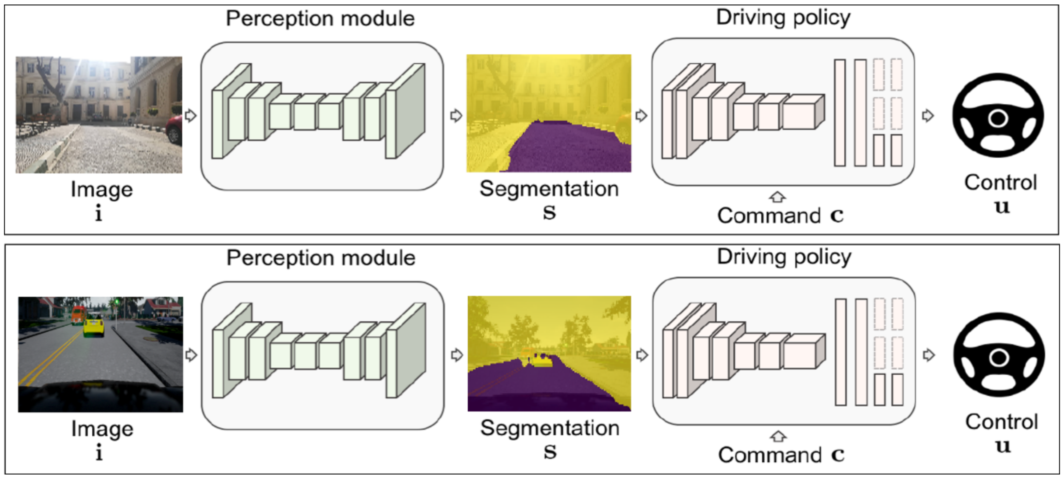


Figure 31. Our abstraction pipeline, operating in a real environment as well as the simulated environment.

Real Environment

After deploying the trained driving model on our physical prototype, we tested the driving policy on our university campus, which is an outdoor predefined environment that includes intersections and roads with no traffic lights nor traffic signs. An image taken from the vehicle's view on our campus is shown in figure 32.

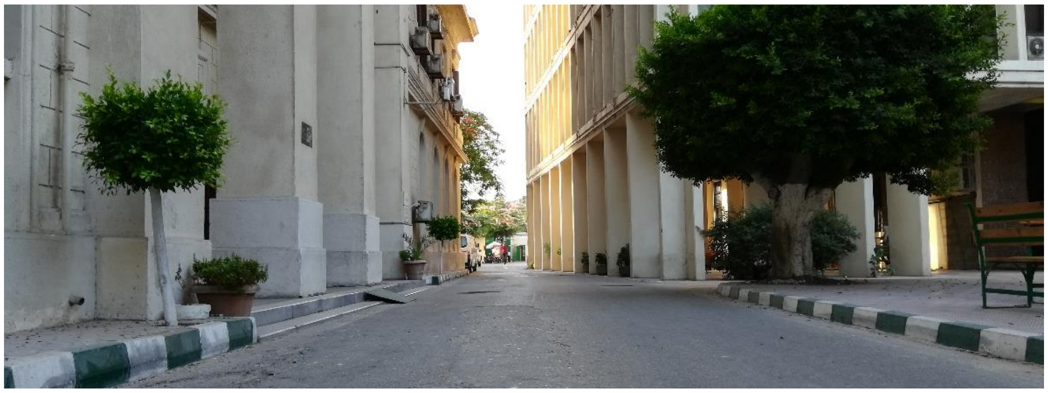


Figure 32. An image from our target environment, which is our campus.

Challenges

We faced some failures in the semantic segmentation view, as it segments the RGB frame that is taken from the camera mounted on our physical prototype, as shown in figure 32, the reason of the semantic segmentation model failure is that the model is trained on CityScapes’ dataset, that includes lanes in all of its frames, while our campus doesn’t. We proposed an experimental solution to enhance the semantic segmentation model by adding graphical lanes to our testing environment as boundaries to the RC car. This solution is firstly experimented by editing some images graphically to test the lanes’ impact on the semantic segmentation result. The addition of lanes made the semantic segmentation much better in most cases as shown in figure 33.

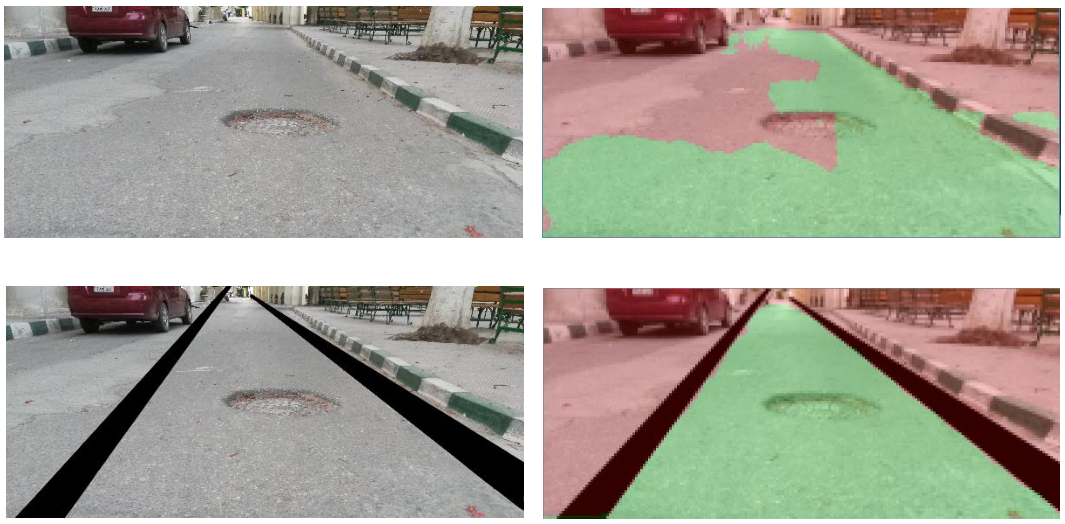
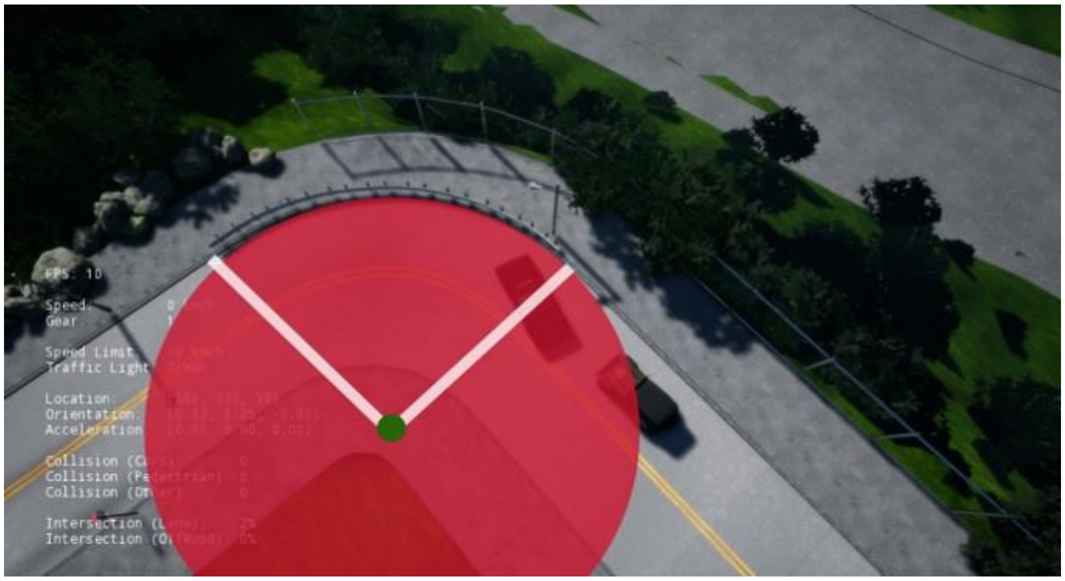


Figure 33. An illustration show the effect of the road lanes on the perception modules.

One more challenge we had to encounted, is the inconsistancy of our semantic segmentation results without road lanes, therefore, we had to build our own track, that is built with the same road geometry in turns and intersections used in our simulated environment. Due to the difficulty of building a large track spanning a large area, instead, we build a miniature version, designed to expose our vehicle to various senarios, like straight roads, simple turns, and taking descisions at intersections. An illustration of the road geometry defined in our simulated environment map as well as an image of our track is shown in figures 34 and 35 respectively.



```
FPS: 10

Speed:          0 [illegible]
Gear            [illegible]

Speed Limit     [illegible]
Traffic Light   [illegible]

Location:       [illegible]
Orientation:    [illegible]
Acceleration    [illegible]

Collision (C[illegible])          0
Collision (Pe[illegible])         0
Collision (Ot[illegible])         0

Intersection (L[illegible])       2%
Intersection (O[illegible])       0%
```

Figure 34. The roads geometry defined in CARLA simulator.

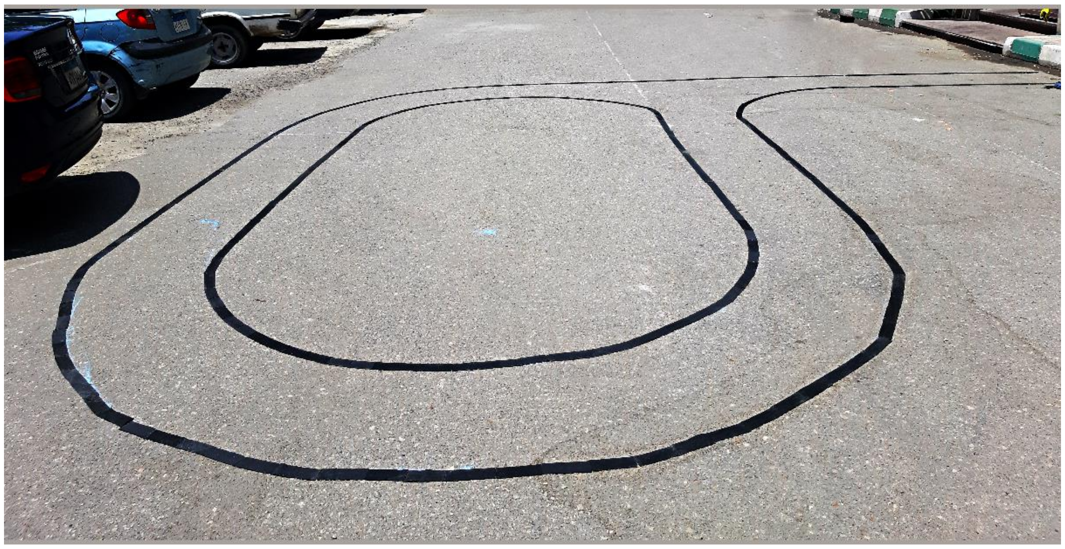


Figure 35. An image of the road we designed, according to the geometry of CARLA simlator's maps relative to our physical prototype's size.

The final challenge we faced, that the trained ERFNet on CityScapes' alone lacked consistency on our track, as it classified regions outside our track as "road", which is not wrong, but it doesn't meet our purpose. So, we had to fine-tune it using data captured from our track. In addition, we faced some processing time issues due to the size of the original ERFNet, and due to our simpler binary segmentation task, we managed to use an optimized version of ERFNet called ERFNet-Fast, which is much smaller than the original one, that fits our purpose perfectly. The full ERFNet-Fast architecture is illustrated in detail in table 4.

To fine-tune ERFNet-Fast, we needed much data from our track, covering several weather, daytime, shadow, and lighting conditions, which is not easy to collect. Therefore, we collected and labeled 600 unique frames, afterwards, we augmented them to reach 14,000 frames, using various augmentation techniques like changing brightness, contrast, channel order, and blur randomly, as well as adding sun flare, artificial shadows, and random objects. A sample of our augmented dataset is shown in figure 36.

Layer	Type	Out Channels	Out Resolution
1	Downsampler block	16	100 x 44
2-6	5 x Non-bt-1D	16	100 x 44
7	Downsampler block	64	50 x 22
8	Non-bt-1D (dilated 2)	64	50 x 22
9	Non-bt-1D (dilated 4)	64	50 x 22
10	Non-bt-1D (dilated 8)	64	50 x 22
11	Non-bt-1D (dilated 16)	64	50 x 22
12	Deconvolution	16	100 x 44
13-14	2 x Non-bt-1D	16	100 x 44
15	Deconvolution	2	200 x 88

Table 4

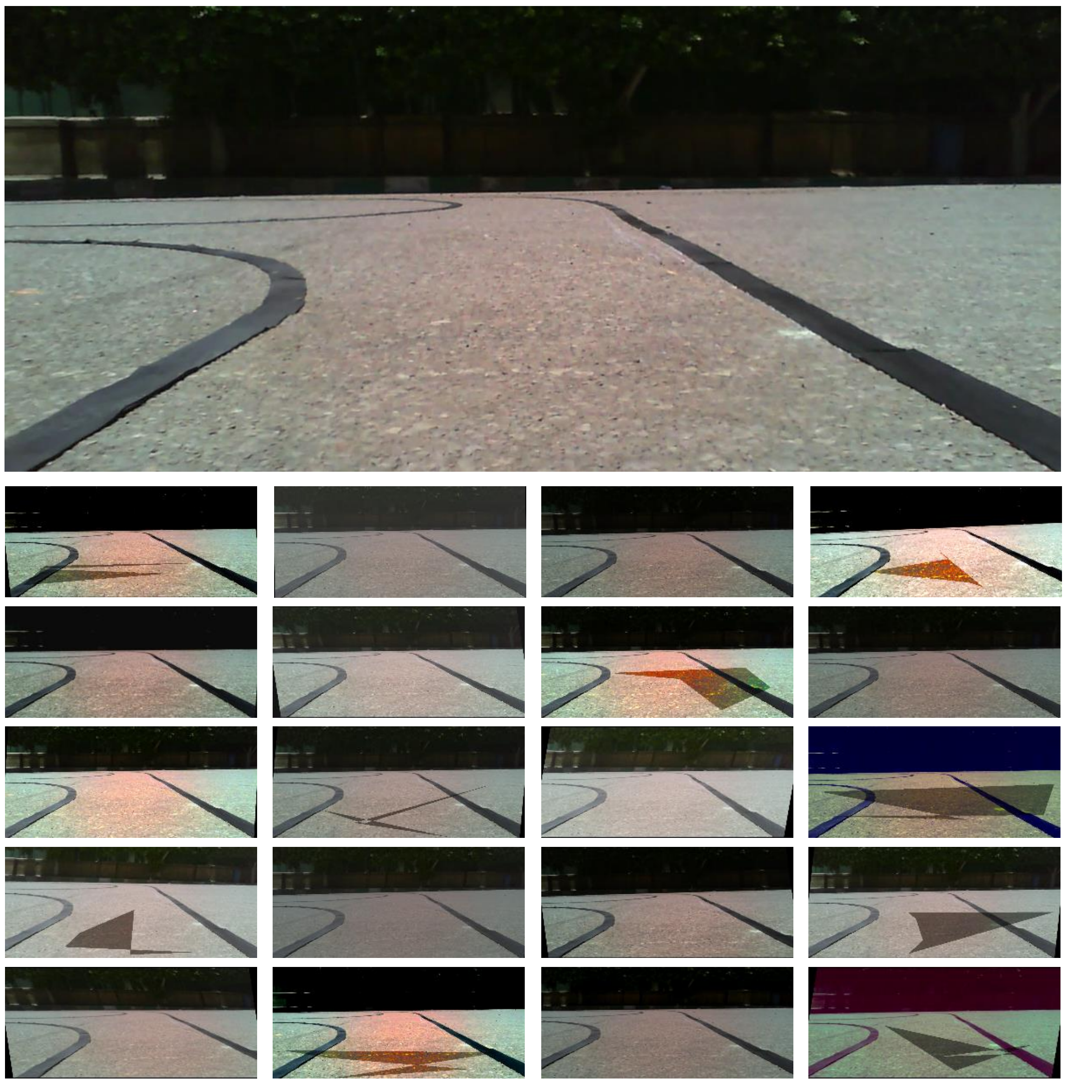


Figure 36. An original image from our track on top, followed by 20 different randomly augmented images.

Chapter 4

Physical Setup

Overview

As explained in earlier chapters, the objective is to utilize state-of-the-art deep learning algorithms on a hardware embedded platform to model a self-driving vehicle. Here is an overview of the hardware system used is shown in figure 37.

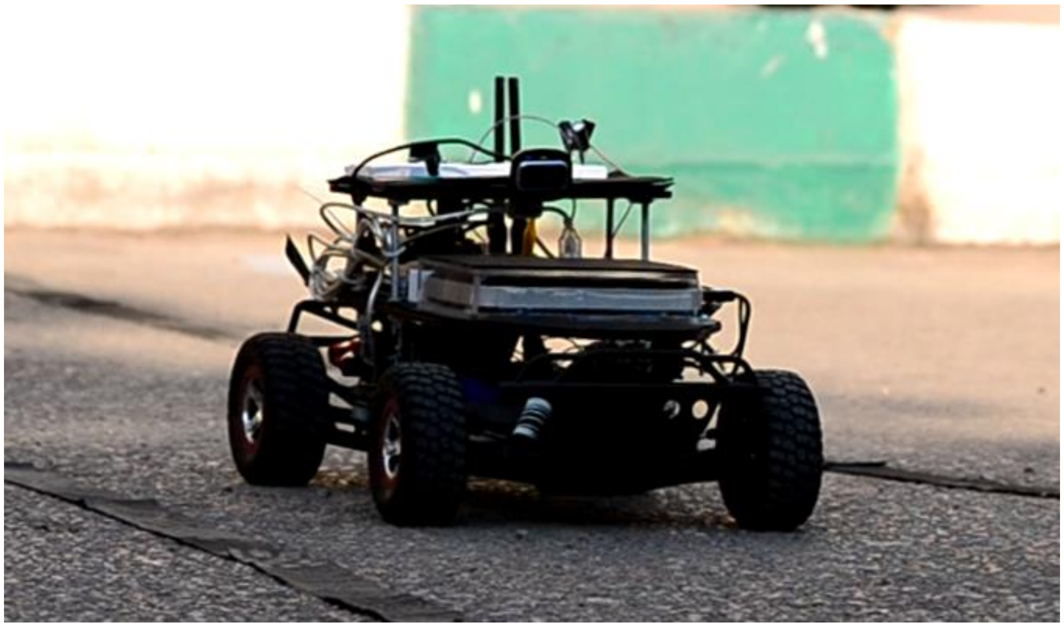


Figure 37. Our physical prototype.

Figure 38. NVIDIA Jetson TX2.

Main Computing Platform

This project uses the NVIDIA Jetson TX2 Developer Kit [39], depicted in figure 38, which exposes the hardware capabilities and interfaces of the Jetson TX2 AI supercomputer on a module. It is proved to be a fast, most power-efficient embedded AI computing device by the time this thesis is published. Moreover, it is compatible with several deep learning frameworks, computer vision, GPU computing, multimedia processing, and others. So, it was a suitable embedded platform to deploy our deep learning algorithms.

The Jetson TX2 is pre-flashed with a Linux environment and includes support for many common APIs. Many important libraries are easy to install and use like CUDA Toolkit for Ubuntu and Linux for Tegra (L4T), OpenCV [40], TensorRT and CuDNN [41].

Main Vehicle Components

The main mechanical hardware vehicle platform used is the Traxxas Slash 2WD RC vehicle [42]. Here is an anatomy of the vehicle as depicted in figure 39. The main components on the chassis are:

- DC Motor.
- Electronic Speed Controller (ESC).
- Steering Servo Motor.
- Battery Cells.
- Receiver Box.

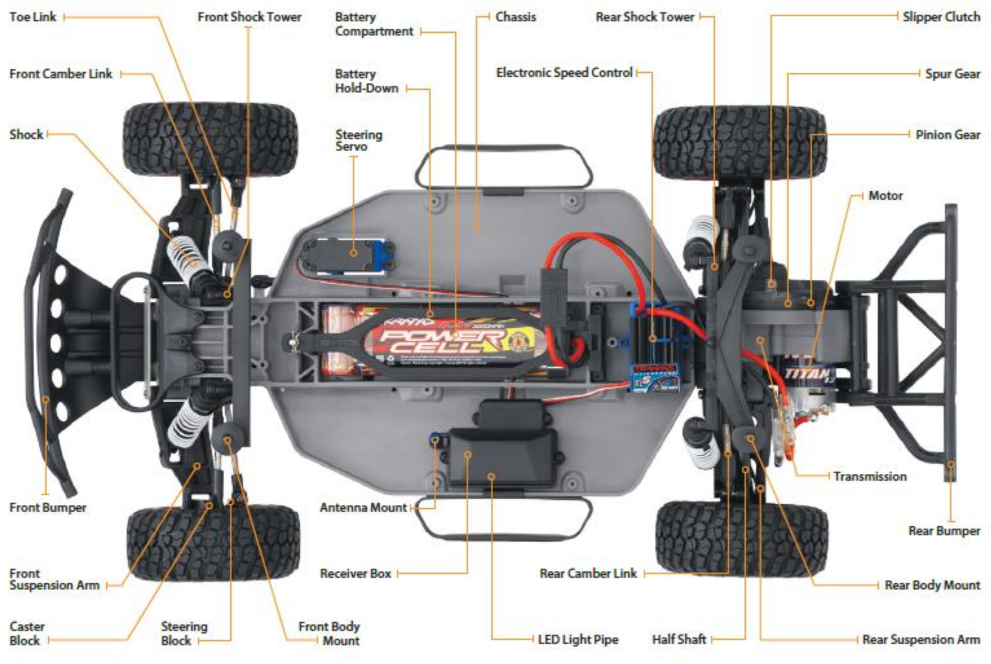


Figure 39. Our physical prototype chasis.

This RC car is mainly used for races, radio-controlled by a Traxxas TQ 2.4 GHz transmitter. The control signals are sent from the TQ transmitter to the receiver box which passes them to both the ESC and the steering servo. The ESC is programmed to control the speed of the DC motor and monitor the battery voltage in order to provide a smooth control experience. In order to utilize this amazing vehicle platform for our objective, the radio system has been removed and both the ESC and the steering servo are connected to a low level embedded micro-controller to provide an ease programming capability.

One main advantage of using this vehicle is the mechanical stability provided in its suspension system, front and rear bumper, and the Ackerman steering system.

Layers & Connections

We have upgraded the vehicle with two layers full of variant components with different functionalities. On the base layer, the ESC and the DC Motor are powered by a NiMH battery. Meanwhile, an embedded microcontroller is used to generate PWM signals to control both the steering servo and the ESC. This microcontroller is connected to the Jetson TX2 which is mounted on the first layer, alongside with the main power supply source which is an Energizer XP20000 Power Pack that provides enough power for Jetson TX2, a USB hub to expand the USB ports available on Jetson TX2, the embedded microcontroller, and a USB camera is also mounted and connected to the Jetson as in figure 40.

The overall process pipeline is summarized as follows:

1. The camera captures a frame infront of the vehicle.
2. The Jetson TX2 processes the input frame and outputs its corresponding predictions.
3. The predictions are converted to control signals, then passed to the microcontroller.
4. The microcontroller generates the corresponding PWM signals to control both the steering servo and the ESC.
5. The ESC outputs the corresponding electrical signals to control the speed of the DC motor.

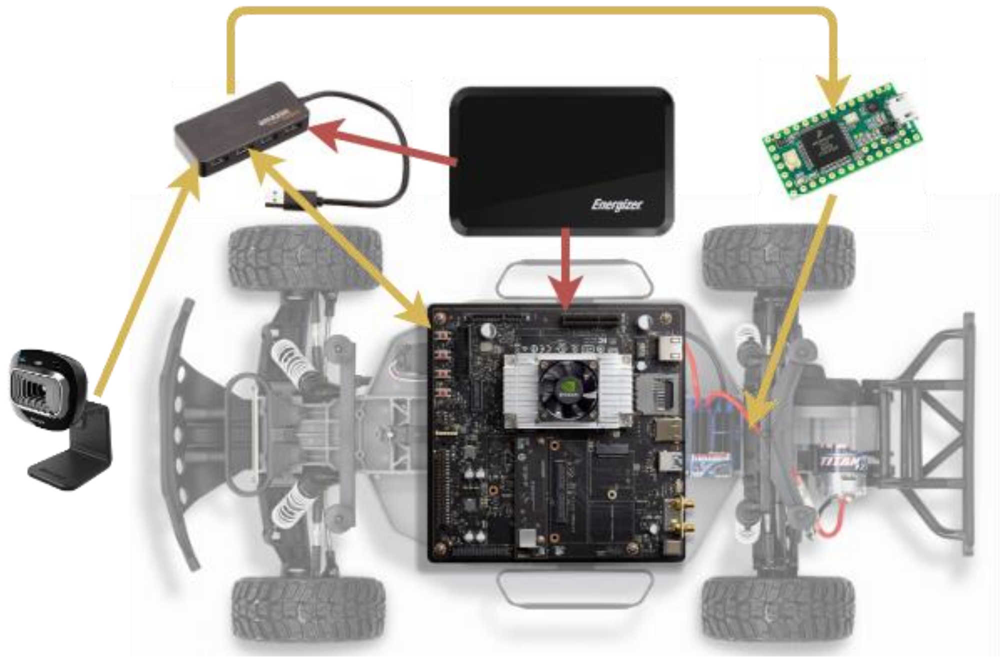


Figure 40. The interconnections and intercommunication on our physical system.

Power Management

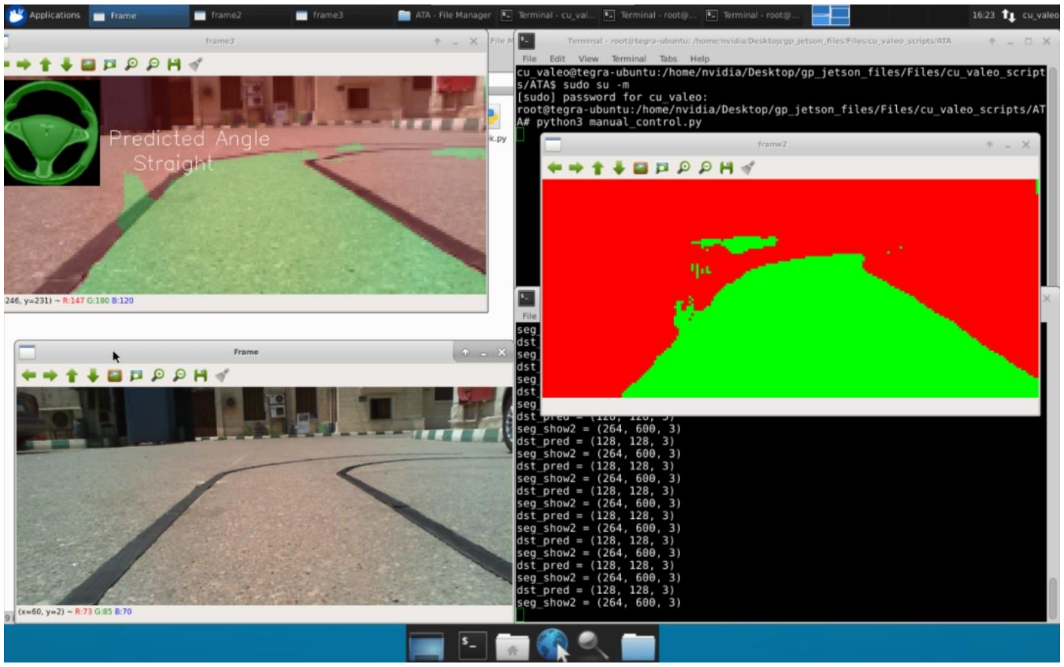
For the NVIDIA Jetson TX2 Developer Kit to run AI models efficiently, it was powered by the 19V 3A outlet of the power pack mentioned before. The Jetson TX2 Kit consumes around 60W at full CPU/GPU utilization, and the system is shipped with a 19V 4.74A power supply. The power pack also provides USB hub with 5V 2A and the PWM microcontroller with 5V 1A.

Firmware

The low-level microcontroller is programmed to provide both the ESC and the steering servo with the required PWM signals upon receiving the prediction signals from the Jetson TX2. This firmware provides an abstraction and ease of high-level control encoded in discrete numbers which are mapped to different PWM control signals. The encoded numbers are then used in communication with Jetson TX2 to be encapsulated in other abstraction layers in the main software application layer.

Wireless Control & Observation

As explained previously, the stock radio system has been removed from the vehicle. But in order for the live debugging process to be more robust and efficient, a wireless connection has been set up on the Jetson TX2 to provide live feedback on the vehicle readings during run-time. A wireless connection hosted on a remote server utilizes the Virtual Network Computing (VNC), which is a connection system that allows you to access the target device remotely. So, a VNC server has been set up on the Jetson TX2. Meanwhile, a command or an observation station is connected remotely to the server, observing and analyzing the scene alongside with different readings and control actions as depicted in Figure 41.



```
cu_valeo@tegra-ubuntu:/home/nvidia/Desktop/gp_jetson_files/Files/cu_valeo_script
s/ATA$ sudo su -m
[sudo] password for cu_valeo:
root@tegra-ubuntu:/home/nvidia/Desktop/gp_jetson_files/Files/cu_valeo_scripts/AT
A# python3 manual_control.py
```

```
seg_show2 = (264, 600, 3)
dst_pred = (128, 128, 3)
seg_show2 = (264, 600, 3)
dst_pred = (128, 128, 3)
seg_show2 = (264, 600, 3)
dst_pred = (128, 128, 3)
seg_show2 = (264, 600, 3)
dst_pred = (128, 128, 3)
seg_show2 = (264, 600, 3)
dst_pred = (128, 128, 3)
seg_show2 = (264, 600, 3)
dst_pred = (128, 128, 3)
seg_show2 = (264, 600, 3)
dst_pred = (128, 128, 3)
seg_show2 = (264, 600, 3)
```

Figure 41. A screen capture from our remote workstation, with complete access to our physical prototype readings and views.

Chapter 5

Experiments and Results

Simulation Benchmark

To prove our end-to-end approach effictiveness, as well as testing our driving policies, we used CARLA simulator’s experiment suites, that offer several trips with gradual increasing difficulty, to unit test each task the driving policy should perform. In this subsection, we benchmarked the original driving policy, trained on raw RGB simulated frames. Our evaluating metric is the success rate, which is the number of successful trips to the total number of defined trips. We compared our results over 50 trips to that of CoIL in both towns 1 and 2, in ordinary weather conditions, using the same experiment suite they used, as shown in table 5.

Moreover, we were able to benchmark our model over 100 trips in town 1, and surprisingly achieved a success rate of **85.85%**, over a total distance of 22.4 km. Also, it was able to achieve a success rate of **84.42%** over 190 different trips, in 2 different weather conditions, in town 1, covering a distance of 41 k

Model	Success Rate	
	Town 1	Town 2
CoIL Command Input	78%	52%
CoIL Branched	88%	64%
Ours Branched	**94%**	**73%**

Table 5. Results in the simulated environment. We compare the presented method to baseline approaches, according to the success rate.

Environment	IoU		
	Maximum	Minimum	Average
CityScapes	**0.95**	0.46	**0.89**
CARLA	0.91	**0.51**	0.76

Table 6. Maximum, minimum, and average IoU values of our perception module on CityScapes' and CARLA test sets.

Semantic Segmentation Evaluation

The most common metric used in semantic segmentation literature is the intersection over union (IoU). IoU meausre allows us to evaluate how similar our predicted labels to the ground truth.

We managed to measure the IoU over two datasets extracted from CARLA simulator, and CityScapes. The maximum, minimum, and average IoU values we achieved at the end of the training phase are illustrated in table 6.

Physical Benchmark

On our track, by design, it is capable of benchcmarking our driving policy in all kinds of tasks it should perform, as mentioned before. We ran several testing sessions in various weather conditions and daytimes. The final results we reached over 10 successive testing sessions are illustrated in detail in table 7. Moreover, the higher level command that should be provided by a higher level planner, was provided manually through a remote controller.

Task	Success Rate
Straight Roads	100%
Left Turns	90%
Right Turns	60%
Approaching an in Intersection Taking a Left Exit	60%
Approaching an in Intersection Taking a Right Exit	40%
Approaching an in Intersection Taking a Straight Exit	100%

Table 7. The success rate achieved by our physical prototype on 10 successive testing sessions.

Chapter 6

Conclusion

We have tested and evaluated an end-to-end approach trained using simulated data, on a a simulated environment, targeting level 5 autonomy, achieving a state-of-the-art success rates in different driving sceanrios, without any pre of post processing. Moreover, we managed to encapsulate and abstract our driving policy through training it on an intermediate domain, that could be reached from several domains and environments. We managed to convert our real-world environment on our campus to the intermediate domain of ours, and successfully transferred our driving policy to our physical prototype.

We proved that end-to-end approaches provide implicit feature extraction, through inspecting the saliency maps of our simulation driving policy. Although end-to-end approaches, given enough data, could provide an extraordinary solution to several complex tasks, but its main drawback that we have no control on the features it relies on, making it extremely hard to debug and inspect in most applications.

The abstraction we demonstrated, showed minimum fine-tuning on the perception module, which is much more easier than fine-tuning the driving policy, as the data it needs can be easliy collected and labeled.

References

[1] Road traffic injuries from WHO, https://www.who.int/news-room/fact-sheets/detail/road-traffic-injuries

[2] Preparing a Nation for Autonomous Vehicles: Opportunities, Barriers and Policy Recommendations, 2010.

[3] J3016-201401 Standard for Terms Related to On-Road Motor Vehicle Automated Driving Systems.

[4] NISSAN'S PROPILOT ASSIST, https://www.nissanusa.com/experience-nissan/news-and-events/nissan-propilot-assist.html

[5] Mayank Bansal, Alex Krizhevsky and Abhijit Ogale. ChauffeurNet: Learning to Drive by Imitating the Best and Synthesizing the Worst. At Google Brain & Waymo, 2018.

[6] Toward autonomous driving: the CMU Navlab. I. Perception, IEEE, 1991

[8] "Google's self-driving-car project becomes a separate company: Waymo". The Associated Press, 2016.

[7] Yata, Teruko & Thorpe, Chuck & Dellaert, Frank. (2002). Static Environment Recognition Using Omni-camera from a Moving Vehicle.

[9] Renault Innovation, https://group.renault.com/en/innovation-2/

[10] Stuart J. Russell, Peter Norvig (2010) Artificial Intelligence: A Modern Approach.

[11] Unsupervised Learning: Foundations of Neural Computation.

[12] Schmidhuber, J. (2015). "Deep Learning in Neural Networks: An Overview.

[13] Hubel, DH; Wiesel, TN (October 1959). "Receptive fields of single neurones in the cat's striate cortex". J. Physiol.

[14] LeCun, Yann; Bengio, Yoshua; Hinton, Geoffrey (2015). "Deep learning". Nature.

[15] Weng, J; Ahuja, N; Huang, TS (1993). "Learning recognition and segmentation of 3-D objects from 2-D images". Proc. 4th International Conf. Computer Vision: 121–128.

[16] Schmidhuber, Jürgen (2015). "Deep Learning". Scholarpedia.

[17] Homma, Toshiteru; Les Atlas; Robert Marks II (1988). "An Artificial Neural Network for Spatio-Temporal Bipolar Patters: Application to Phoneme Classification". Advances in Neural Information Processing Systems.

[18] Waibel, Alex (December 1987). Phoneme Recognition Using Time-Delay Neural Networks. Meeting of the Institute of Electrical, Information and Communication Engineers (IEICE). Tokyo, Japan.

[19] Alexander Waibel et al., Phoneme Recognition Using Time-Delay Neural Networks IEEE Transactions on Acoustics, Speech, and Signal Processing, Volume 37, No. 3, pp. 328. - 339 March 1989.

[20] LeCun, Yann; Bengio, Yoshua (1995). "Convolutional networks for images, speech, and time series". In Arbib, Michael A. (ed.). The handbook of brain theory and neural networks (Second ed.). The MIT press. pp. 276–278.

[21] Shah, Shital et al. "AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles.".

[22] Nathan Koenig and Andrew Howard. Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator. In Intelligent Robots & Systems Conference 2004, Japan.

[23] Chen, Chenyi et al. "DeepDriving: Learning Affordance for Direct Perception in Autonomous Driving." 2015 IEEE International Conference on Computer Vision (ICCV) (2015): 2722-2730.

[24] Dosovitskiy, Alexey et al. "CARLA: An Open Urban Driving Simulator." CoRL (2017).

[25] P. Englert, A. Paraschos, J. Peters, and M. P. Deisenroth. Model-based imitation learning by probabilistic trajectory matching. In ICRA, 2013.

[26] Bojarski, Mariusz et al. "End to End Learning for Self-Driving Cars.".

[27] Bojarski, Mariusz et al. "End to End Learning for Self-Driving Cars.".

[28] TensorFlow: A system for large-scale machine learning, 2006.

[29] Bojarski, Mariusz et al. "Explaining How a Deep Neural Network Trained with End-to-End Learning Steers a Car.".

[30] Simonyan, Karen et al. “Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps.”.

[31] Sumit Chopra , Suhrid Balakrishnan , Raghuraman Gopalan, “DLID: Deep learning for domain adaptation by interpolating between domains” (2013).

[32] Long, Mingsheng et al. “Conditional Adversarial Domain Adaptation.” NeurIPS (2018).

[33] Hoffman, Judy et al. “CyCADA: Cycle-Consistent Adversarial Domain Adaptation.” ICML (2018).

[34] Paszke, Adam et al. “ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation.”

[35] Zhao, Hengshuang et al. “Pyramid Scene Parsing Network.” 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2017): 6230-6239.

[36] Romera, Eduardo et al. “ERFNet: Efficient Residual Factorized ConvNet for Real-Time Semantic Segmentation.” IEEE Transactions on Intelligent Transportation Systems 19 (2018): 263-272.

[37] Mueller, Matthias et al. “Driving Policy Transfer via Modularity and Abstraction.” CoRL (2018).

[38] Paszke, Adam et al. “Automatic differentiation in PyTorch.” (2017).

[39] NVIDIA Jetson TX2 Developer Kit Specifications, https://developer.nvidia.com/embedded/jetson-tx2-developer-kit

[40] Bradski, G., 2000. The OpenCV Library. Dr. Dobb's Journal of Software Tools.

[41] Chetlur, Sharan et al. "cuDNN: Efficient Primitives for Deep Learning.".

[42] Traxxas Slash 2WD User Manual, https://traxxas.com/sites/default/files/58034-58024-OM-N-EN-R02_0.pdf