CE903 Team 6

Final Report

Automata – Self-Driving Car

Team Members:

Juan Antonio Lizárraga Vizcarra

Monali Gothi

Ruth Lisa Wanjiku Maina

Ebubekir Kocaman

Muhammad Atiq Khan

Brijesh Asodariya

Mano Karukkampalayam Viswanathan

Devesh V. Fuse

Suyash Gupta

Kevin Machado Gamboa

Supervisor:

Dr Luca Citi

Date: 29 Apr 2021.

ABSTRACT

The following report details the development process for an autonomous driving agent, which operates in the CARLA simulation environment. Firstly, a series of RGB images were recollected by running an example autopilot script from the CARLA library, while also logging car control values (steering, throttle and braking). The data was then used to train multiple convolutional neural network architectures. Additionally, several independent detection modules were developed to detect lanes and objects in the scene. These could then be used to enhance the car control model input, to improve model learning. Although the detection modules perform satisfactorily, the experiments show that there are still plenty of improvements for a fully autonomous model. The strengths and limitations of the model are discussed, along with possible future lines of research.

Keywords: Autonomous driving, object detection, steering control, lane detection,

ACKNOWLEDGMENTS

We want to thank Dr Luca Citi for helping manage this project. Luca's expert guidance was critical in directing the group project, providing valuable feedback at crucial junctures, and helping us gain access to the virtual machine used for the simulator.

Additional thanks go to Dr Thakur Manoj, the module supervisor.

Table of Contents

List of figures

1. Introduction

1.1. Purpose

This document is the final report, it follows on from the work that had been completed within the requirement specification for the "Automata" project. The report starts with breaking down the "Automata" project into its component sections and completing an investigation into what has already been done within the industry and identifying what can be learnt from existing works and applied within the constraints of this project. The researched areas are autonomous vehicles, convolutional neural networks, transfer learning, object detection and masking, traffic sign detection and other related works. This is then followed up by the system design, where topics such as the requirements and architecture had been discussed. The methodology had also similarly been broken down into its component sections, this was done to ensure that teams could be created, and team members were able to work together effectively on specific tasks. The tasks within the methodology ranged from data collection, lane detection, object detection, traffic sign detection, model building, and exploratory analysis. The implementation is what came next, and the methods by which the tasks that were completed within the methodology, were completed within the project have been discussed. Section 6 is a continuation of the test cases that were created within the requirement specification document, however, within this report, they have been assessed to see whether the "Automata" project has been able to successfully pass them. A discussion had then been completed, to provide reasoning and an explanation of the results that were obtained. As well as also highlighting what needs to be done in the future to build upon what has been completed so far.

Due to this project being completed as a team, a section has also been provided to illustrate the steps that were taken to ensure that the team was able to work together effectively. The structure and methodology that were taken have been covered as well as the challenges and how they were overcome. The report comes to an end with the user documentation, with steps in how to get the CARLA simulator installed and how to run the scripts that were created.

1.2. Motivations

With the number of vehicles on the road within the UK totalling close to 38.8 million in September 2020 [1], it has meant that there are now more active road users within the UK than ever recorded before. In the UK individuals are required by law to ensure their vehicle, in 2017 insurance companies within the UK had paid out at an eyewatering amount of 13.4 billion pounds. Putting that into perspective that totals to around 29 million pounds being paid out every day [2]! A statistic that forms the main motivation for this project is not concerned with a monetary value, it was reported that in the year ending June 2018 that there were 26,610 people either killed or seriously injured [3]. The majority of accidents that occur on the road are a result of the driver, The United States Department of Transportation (USDOT) had predicted that the rise of driverless cars would see a drop of over 90% in fatal accidents on the road.

Autonomous vehicles provide an opportunity for the development of technology in an area that is a part of day-to-day life. Studies have found that one in six crashes that result in death or injury is related to fatigue, of this, an estimated 40% involve commercial vehicle drivers [4].

Autonomy within vehicles would allow for a significant reduction within these statistics as it would be able to minimise the risk that a collision occurs because of a mistake made by the driver.

“Automata”, is the name for the autonomous driving agent, and is the team's contribution to this field of research. The “Automata” is a software package that works within CARLA, which is an open-source simulator that is focused on autonomous driving research. Upon the release of the software created by “Automata”, it will serve as a foundation for further research into self-driving vehicles. As has been mentioned, due to the software being created within CARLA some restrictions are placed by the simulator itself, and these will be highlighted further below.

1.3. Objectives

The end goal for the project is to be able to create an autonomous driving agent, that can drive around within the CARLA simulator without any human intervention. Correctly being able to identify objects, vehicles and pedestrians in its surroundings. To achieve this aim, the functional requirements had been created, this was done to measure the overall success of the project and ensure that the main purpose that was set out to be completed was not forgotten. The functional requirements for this project have been presented within section 3.1 of this report.

The research that has been carried out throughout this project, allows for a solid foundation for projects relating to autonomous driving. The depth of the project allows for the combination of multiple methods that relate to lane detection/following, object detection (vehicle, pedestrians), traffic sign detection and a controller model. All working together in harmony would allow for a vehicle that would be able to drive without needing input from a driver.

Even though a model that incorporates multiple sensors in addition to a camera, may perform better in a real-world environment due to the time constraints with this project it was not deemed feasible to attempt to include such sensor values in the final model. This is a limitation that needs to be considered when attempting to improve the results that were achieved within this project.

2. Literature Review

2.1. Autonomous Vehicles

Autonomous vehicles can drive without human driver input. Autonomous driving has been in discussion since the 1900s [5]. With continuous research, there has been a drastic growth in the industry, especially after introducing (DAPRA) the U.S. Defense Advanced Research Projects Agency [6] and Google driverless car. It attracted a vast talent pool with different skillset into the growth of autonomous driving. In 2015 Tesla debut the autopilot feature that has enabled hands-free control for freeway driving and highway it has been developed by a single software Model S by Tesla cooperation. Model S manages lane changing and speed using integrated cruise control [7].

Autonomous vehicles are not yet available to the public especially after the Google self-driving car was found to be at fault in an accident after colliding with public transport bus in 2016 [8].

Recently limited autonomous functionalities have been integrated into a limited modern number of vehicles like Honda the Hybrid EX sedans [9] to assist with parallel parking, lane assistance monitoring vehicles position in the lane and parking assistance.

2.2. Convolutional Neural Networks

According to [10] the idea of Convolutional Neural Networks started to appear around the 1980s after publications regarding the functioning of visual cortices of monkeys and birds. Additionally, according to the same author, one of the first building blocks of CNN architectures for all the different versions available today was named LeNet-5. This was developed by Yann Le Cunn who is considered an important figure for his work in the AI field.

2.3. Transfer Learning

Transfer Learning is using knowledge of a pre-trained machine learning model as the starting point for a model on a second task which may be designed for a different but related problem. Transfer learning has a couple of advantages, yet the fundamental benefits are saving training time, better execution of neural organizations (in most cases), and not needing a lot of data. There are different approaches to transfer learning: training a model to reuse it, using a pre-trained model, feature extraction, etc. There are some pre-trained models which have been extensively used in literature: Inception-v3, ResNet, AlexNet, and GoogleNet trained on ImageNet dataset. These have been used for transfer learning and showed great execution in different applications [11].

Figure 1 shows the difference between training from scratch and transfer learning architecture.

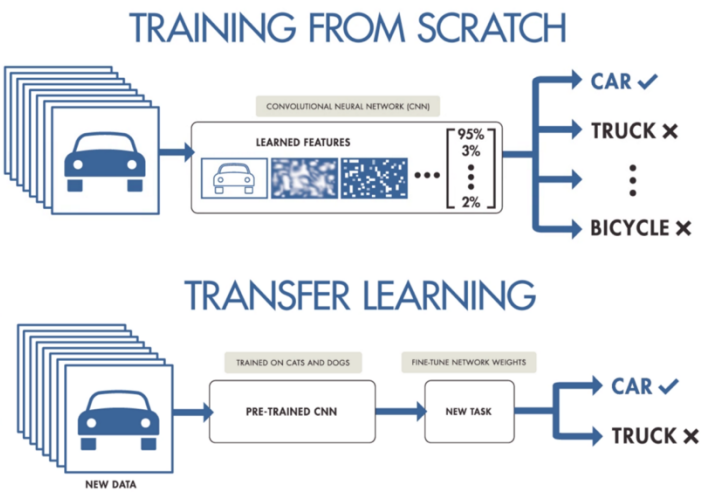


Figure 1 Difference between training from scratch and transfer learning [13]

2.4. Object Detection and Masking

Object identification is the process of tracking all the possible instances of real-world objects in pictures or recordings (possibly even in real-time) with the most extreme precision [12]. The object discovery method utilizes derived features and learning algorithms to recognize all the occurrences of an object category [12].

The technical development of object recognition began in the mid-2000s and the detectors at that time. They followed the low-level and mid-level vision and followed the technique for 'recognition-by-components'. This technique empowered object disclosure as an estimation of closeness between the object segments, shapes, and forms, and features. Some of the contemplated features were distance changes, shape settings and edges, amongst other things. Experiments went poorly and afterwards machine identification strategies began to come into the image to take care of this issue [12].

One of the earlier main obstacles for multi-scale object recognition was data that had "many sizes" and "diverse viewpoint proportions". In any case, after 2014, with developments in specialized data handling and processing, the issue was settled. This paved the way for the second period of item identification, where the assignments were tackled utilizing deep learning [12].

Object recognition is feasible thanks to machine learning and deep learning. The machine learning approach requires features to be characterized by an internal representation and afterwards utilizing models like Support Vector Machines (SVMs) to do the classification. While the deep learning approach makes it conceivable to locate objects without expressly characterizing the features to do the classification. The deep learning approach is significantly founded on Convolutional Neural Networks (CNNs) [12].

Machine Learning Methods for Object Detection:

- Scale-Invariant Feature Transform (SIFT)
- Histogram of Oriented Gradients (HOG) features
- Viola-Jones object detection framework

Deep Learning Methods for Object Detection:

- Region Proposals (R-CNN, Fast R-CNN, Faster R-CNN)
- You Only Look Once (YOLO)
- Deformable convolutional networks
- Refinement Neural Network for Object Detection (RefineDet)
- Retina-Net
- Single Shot MultiBox Detector (SSD)

R-CNN

It is of importance to improve the nature of candidate bounding boxes and to take a deep architecture to extract high-level features. Figure 2 shows the flowchart of R-CNN, which can be partitioned into three phases - Region proposal generation, CNN based deep feature extraction, classification, and localization [21].

The R-CNN uses selective search [19] to create about 2k locale proposals for each picture. Every district proposition is twisted or trimmed into a fixed resolution and the CNN module in [20] is used to obtain a 4096-dimensional feature as the final representation [21]. With pre-trained category specific linear SVMs for various classes, diverse areas recommendations are scored on a bunch of positive areas and background (negative) areas [21]. The scored areas are then changed with bounding box regression and filtered with a greedy non-maximum suppression (NMS) to deliver final bounding boxes for preserved object locations [21].

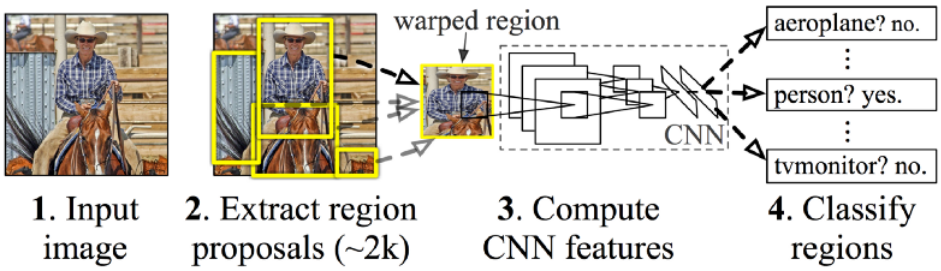


Figure 2: R-CNN [14]

Fast R-CNN

Girshick [15] presented a multi-task loss on classification and bounding box regression and proposed a novel CNN structure named Fast R-CNN [21]. The design of Fast R-CNN is shown in Figure 3, the entire picture is handled with Conv layers to create feature maps. At that point, a fixed-length feature vector is selected from every area proposal with a region of interest (RoI) pooling layer [21]. Each feature vector is then taken care of into a sequence of FC layers before final branching into two sibling output layers [21]. One output layer is liable for generating SoftMax probabilities for all C + 1 categories (C object classes in plus one 'background' class) and the other output layer encodes refined bounding box positions with four real-valued numbers. All parameters in these methods (apart from the generation of region proposals) are optimized via a multi-task loss in a start to finish way [21].

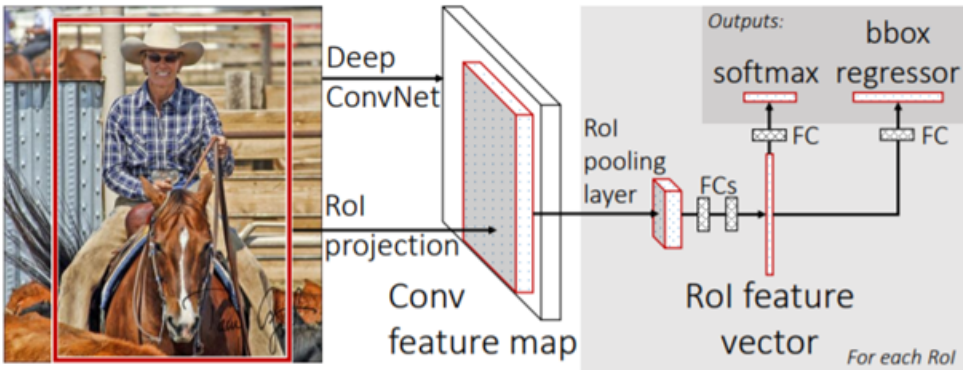


Figure 3: Fast R-CNN [15]

Faster R-CNN

Faster R-CNN is quite quicker than others in light of the fact that utilized another strategy Region Proposal Network (RPN) rather than Selective Search [24]. RPN primarily advises the Fast R-CNN where to look. Like Fast R-CNN, a single CNN accepts whole picture as info and produces a feature map [16]. On the feature map, RPN creates a bunch of rectangular object recommendations with "object-ness" scores as output. These qualities are then reshaped utilizing RoI pooling to predict classes and offset values for bounding boxes [16].

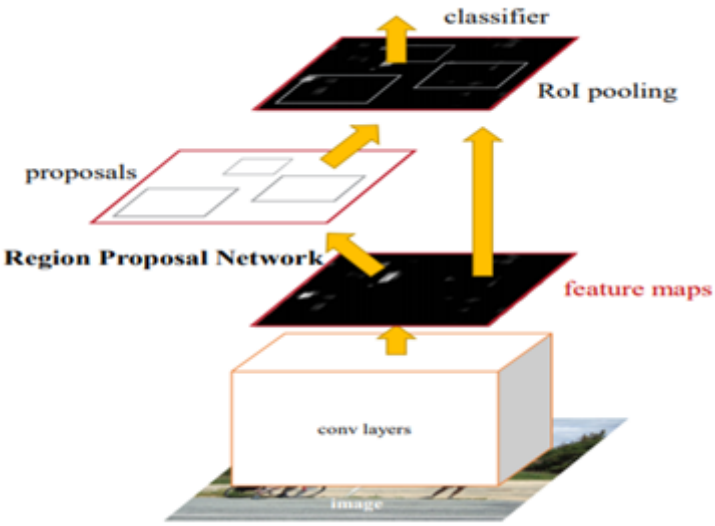


Figure 4: Faster R-CNN [16]

	R-CNN	Fast R-CNN	Faster R-CNN
Test time per image	50 seconds	2 seconds	0.2 seconds

Figure 5: Comparing of R-CNN Family

Mask R-CNN

Mask R-CNN is an extension of Faster R-CNN [26]. In this model, objects are grouped and localized using a bounding box and semantic division that classifies every pixel into a bunch

of categories [27]. This model extends Faster R-CNN by adding the expectation of division masks on every Region of Interest. The Mask R-CNN produces two outputs: a class name and a bounding box [27].

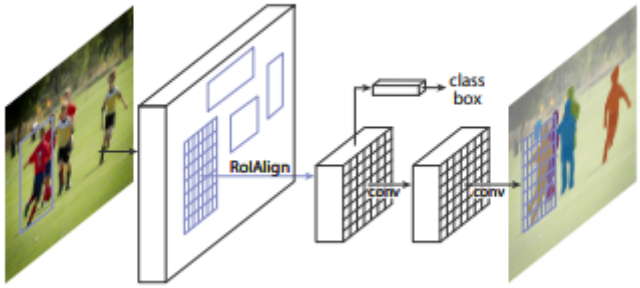


Figure 6: Mask R-CNN [27]

YOLO

YOLO has one neural network that predicts bounding boxes and class possibilities straightforwardly from whole pictures in a single evaluation [17]. The YOLO models treat 45 frames each second in real-time [26]. YOLO sees picture detection as a regression issue, which makes its pipeline very straightforward [26]. It's incredibly quick due to this basic pipeline [26]. R-CNN family has a complex pipeline and slower than YOLO [24]. In this model, 'You Look Only Once' at the picture to predict classes of objects and where they are [24]. YOLO separates the picture into grids, every grid cell predicts bounding boxes and confidence scores for these boxes [24]. The first form of YOLO is very quick however has downsides. The network struggles with little objects and this was tackled with later forms; Yolov2, Yolov3, Yolov4 [24].

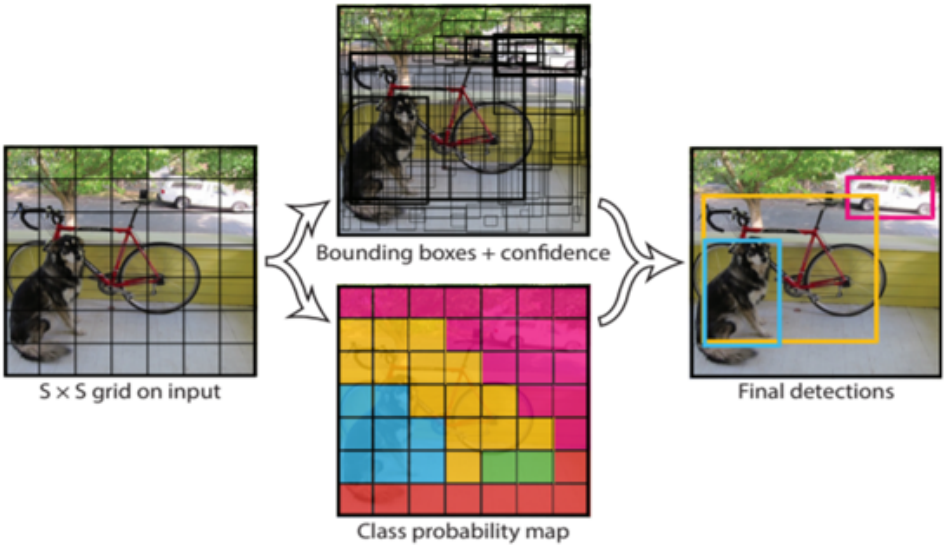


Figure 7: YOLO [17]

SSD (Single Shot Detector)

The SSD design was printed in 2016 by specialists from Google [23]. It presents an object identification model using a single deep neural network coupling regional proposals and feature extraction [23]. The SSD approach discretizes the output space of bounding boxes to a bunch of default boxes over various aspect proportions [25]. In the wake of discretizing, the technique scales per feature map area [25]. The Single Shot Detector network joins expectations from multiple feature maps with various resolutions to normally manage objects of varied sizes [25]. SSD is quicker than the past object discovery models like YOLO [24]. Input picture feeds into VGG-16 to extract feature maps with various scale [24].

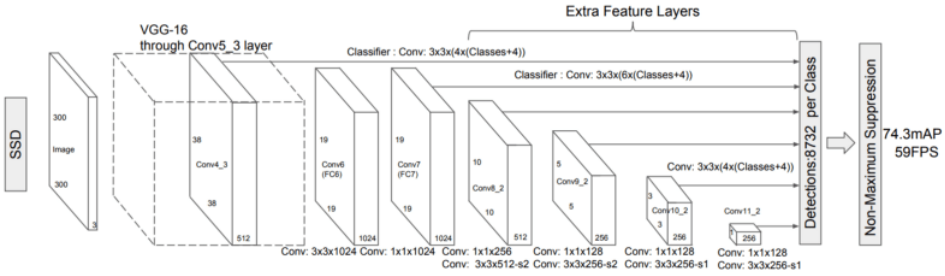


Figure 8: SSD [18]

2.5. Traffic Sign Detection

Traffic sign detection involves many Machine Learning techniques. These include Convolutional Neural Network (CNN) and Support Vector Machines (SVM). Methods include colour-based methods and shape-based methods. Once the data has been collected for detection, it is then pre-processed with enhancements applied. This is followed by segmentation according to attributes of the colour and shape. Traffic sign detection involves detecting potential pixel area of which may be of interest for detection. These are obtained through complex backgrounds. Images are also normalized to a specific size which then enters the recognition phase. Detection methods can be classed as statistical-based and SVM based. However, deep learning has advanced in this area in recent years. CNN can be trained automatically where features can be extracted. This improves the accuracy significantly which is much higher than traditional Machine Learning methods such as SVM. Despite this, real-world applications are limited with CNN due to the lack of available datasets. Larger datasets are required for successful CNN training and validation. Previous work implemented for CNN involves several steps in its making for traffic sign detection. Initially, the images are normalized between pixel values 0 and 0.5. This is done due to a performance increase when the raw data is set between 0 and 1. This is then passed onto the CNN model. VGG-16 used as a front-end network structure of the SSD algorithm. It consists of 5 stacked convolutional layers, 3 of which are fully connected and finally SoftMax. Each stacked convolution layer has several common convolutional layers. This forms a local CNN which is followed by a pooling layer. Output from this is then connected as an input to SoftMax. Once this is accomplished, image recognition is obtained. Figure 9 shows an outline of the CNN involved in traffic sign detection [28].

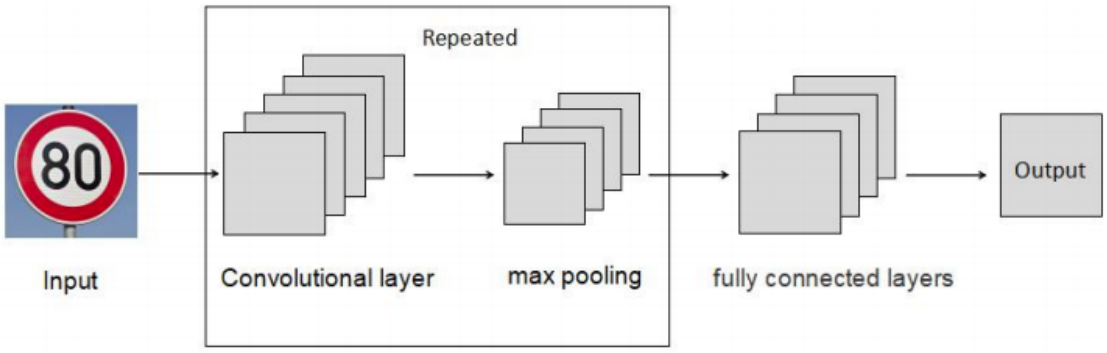


Figure 9: Structure of CNN model for traffic sign detection [28]

Another approach that exists is the use of SVM classifier with colour and texture feature extraction. Previous research done involves the use of RGBN, GABOR, HOG, LBP and SVM methods. This classification is achieved in 3 main steps.

1. Pre-processing of images is accomplished using RGBN colour space and ROI segmentation with the image threshold

2. Images are labelled, and a candidate area is identified

3. Traffic sign is detected with analysis of each candidate region by using region props

The captured images are in RGB coloured space. Due to this, they have sensitivity to environmental lighting conditions. To overcome this, the images are converted to a normalized RGB colour space (RGBN). RGBN colour space is created separately of various lighting conditions or intensities. These conversions will discard the effects of intensity variations. Traffic signs are measured with the use of yellow, red and blue colours from normalized RGB imagery. Figure 10 shows this method in a sample image.

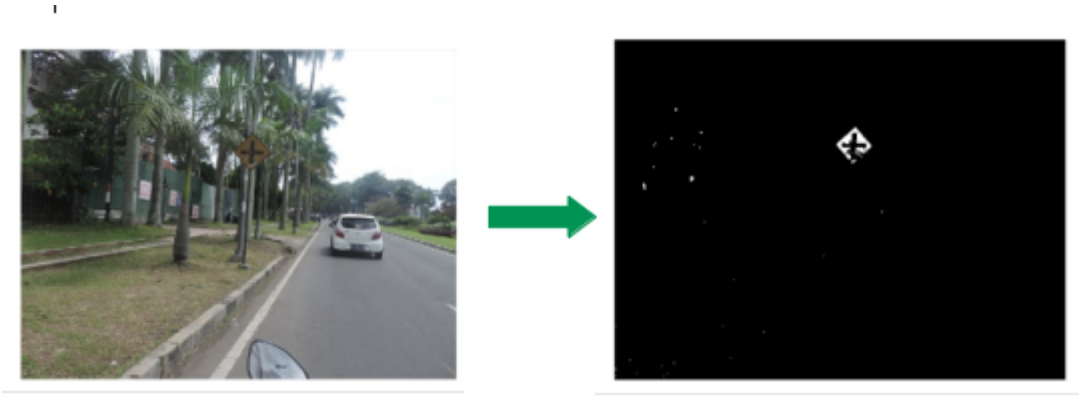


Figure 10: RGBN illustration [29]

After this, the image is labelled to identify the candidate regions where a traffic warning sign might exist. The area of each labelled region is calculated with the use of the region props method. Each ROI is analysed to find the region of the danger traffic sign. Small and large regions are considered as outliers and so are removed. This is done to reduce computation time and the fact that the data was not useful. Once a candidate region is chosen, a bounding box is drawn on the image which is thought to be a traffic sign. Figure 11 shows this applied to a sample image.

Figure 11: Candidate region for traffic sign detection [29]

SVM is a technique to which makes predictions, both for classification and regression. Due to the functionality and problem conditions, SVM is in a classed with the Artificial Neural Network (ANN). The basic concept for an SVM simply be explained as a search for the best hyperplane where serves as a separator of two classes in the input space [29].

2.6. Related Work

There has been significant attention in the area of self-driving vehicles in recent years as technology has been rapidly improving. Simulators such as CARLA allow for anyone to be able to create a model and run it within its simulator provided that their machine meets the hardware requirements [30]. In the work carried out by Jaafra, et al an advantage actor-critic approach with multi-step returns for autonomous driving has been proposed. Research that had been carried out highlighted that this type of reinforcement learning approaches tends to provide good convergence performance and faster learning this has made it a preferred reinforcement learning algorithm [31]. The experiment carried out by Jaafra, et al aimed to demonstrate whether the incorporation of a multi-step returns critic (MSRC) component in a deep reinforcement learning framework would allow for the agent to effectively be able to guide its learning strategy. They had hypothesised that a reduction in actor gradient variance would occur. The results were generally showing a higher performance, however, when the agent was shown new testing conditions it did show vulnerabilities.

A different approach had been taken within [32], in this paper Kadam, et al had used a deep deterministic policy gradient algorithm. The reason why this was selected was down to the fact that it can work in complex and continuous domains. The findings from this paper did show promising results however the architecture was only able to work in certain conditions and struggled when variable such as weather, and time of day were changed within CARLA as the sensors picked up different variables, e.g., shadows and reflections.

A different framework was proposed in [33], where a framework for an end-to-end Deep reinforcement learning pipeline was integrated with multiple recurrent neural networks. The proposed architecture was successful in the constraints of the project, where it was required to stay within the markings of the road. The future works that have been proposed is to extend

upon those works and apply them into a scenario where there are other agents present to see how they would interact with one another.

Autonomous research is a growing industry with massive impact in the urban environment the fusion of different systems to be controlled from a control area will require extensive research and testing hence why multiple top vehicle brand models have invested in integrated limited components like lane changing in Tesla models for a selected number of vehicles. With the need to continuously test algorithm before implementation similar open-source simulators used also in race cars have been used Autoware (Gazebo) [34], AirSim developed by Microsoft AI & research [35] and TORCS (OpenGL) [36]. Self-driving is highly technological driven compared to the socio-economic and environmental there is limited research on the impact it may have on the dimensions of sustainable transportation.

3. System Design

3.1. Requirements

FR001	The Controller can move the car programmatically.
FR002	The Lane Detection module can identify a valid path on the road.
FR003	The Model can follow a straight path
FR004	The Model can follow a curved path
FR005	The Object Detection module can identify other vehicles (cars, trucks, bikes).
FR006	The Object Detection module can identify pedestrians.
FR007	The Object Detection module can identify traffic lights.
FR008	The Traffic Sign Detection module can identify and classify traffic signs

3.2. Modules

Module: Autopilot (based on example CARLA script)
Workloads 1. Interfaces with CARLA's internal autopilot system 2. Adapted to take images from vehicle's RGB front camera 3. Logs vehicle control values (throttle, steering, brake) into csv file with corresponding image

Module: Controller
Workloads 1. Spawns a pre-defined vehicle in the CARLA server 2. Equips the vehicle with an RGB front camera 3. Defines an API to control the vehicle driving programmatically 4. Loads the necessary models to perform image processing and drive the car 5. Calculates vehicle control values based on input from RGB front camera 6. Destroys actors (vehicle and camera)

Module: Main
Workloads 1. Instantiates the Controller and links it to the CARLA server

Module: Lane Detection
Workloads 1. Takes an RGB image and produces a mask highlighting the presence of a lane.

Module: Object Detection
Workloads 1. Takes an RGB image and produces a mask highlighting the presence of objects traffic lights, vehicles, trucks, motorbikes, and persons.

Module: Traffic Sign Detection
Workloads 1. Takes an RGB image and classify it into a category from a collection of traffic sign categories

3.3. Architecture

The entire code base can be divided into two main modules: CARLA and AUTOMATA. CARLA contains all the simulator source code, along with the Python API library. It also houses the original autopilot script that we repurposed to perform data collection.

AUTOMATA contains our code. The main script instantiates all the necessary actors and connects to the CARLA server. The Controller module, in turn, includes the different detection modules and applies the vehicle control values to the CARLA vehicle.

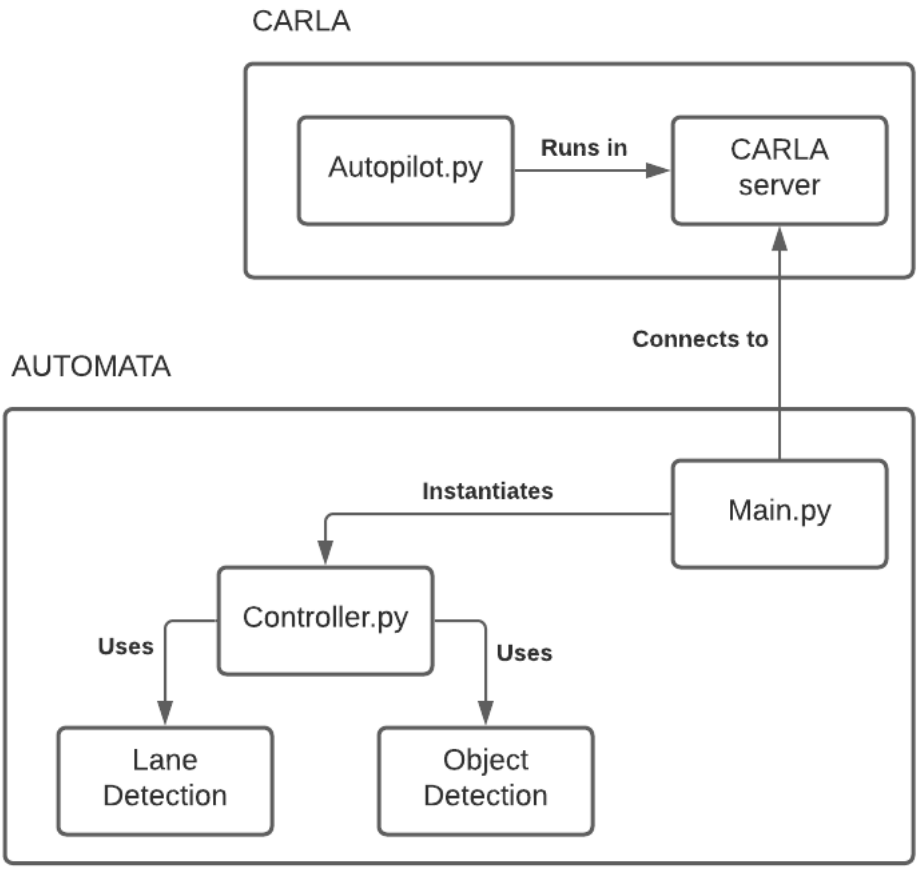


Figure 12: Entity-Relation diagram of Automata and CARLA modules

3.4. Data Flow

Capturing data for the automata model begins when interfacing with CARLA's RGB sensor. Despite the name, it outputs raw data in the form of a BGRA tensor. The alpha layer is first removed. The blue and red channels are then swapped to form the traditional RGB format. The image is then resized and fed into the lane detection model. The model's output, the lane mask, is concatenated to the RGB image. The enhanced image is then fed into the driving model that outputs the vehicle controls (throttle, steering and braking). CARLA's API for vehicle control is then utilized to apply the desired control values, which then move the car.

The data flow is summarized in the following figure:

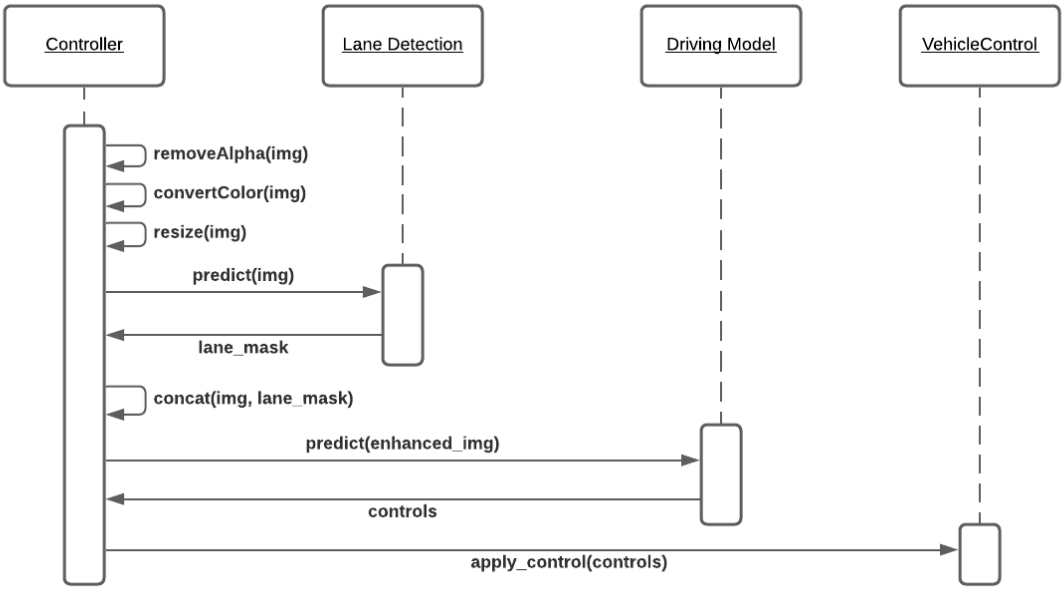


Figure 13: Sequence diagram displaying data flow for Controller driving

4. Methodology

4.1. Data Recollection

The CARLA build already comes with several example scripts which display different functionalities of the simulator. One of the provided scripts instantiates a car in the CARLA server and has the added option to enable autopilot functionality. The autopilot interfaces with the world and its actors (other vehicles, traffic lights, stop signs, etc.) in order to drive. The script comes with an option to save images.

However, since our use case required some custom fitting, the script was used as a base and slightly modified to accommodate for our data-collection needs. Firstly, in addition to saving images to disk, the Autopilot module also logs the steering, throttle, and brake values in a csv file. Furthermore, each row in the csv file has a corresponding image, whose id is also recorded for training purposes.

The model first recorded images and vehicle control values for a fixed rate (every 1,000 frames in the simulation). Initial experiments with a driving model demonstrated that the Controller underperformed when dealing with curves. As such, it was concluded that there wasn't sufficient data for curves. More importantly, obtaining vehicle control values for every segment of the curve was important as steering and braking greatly vary even for a single left or right turn. This resulted first in a reduction of the fixed rate (from 1,000 frames to 250) and the introduction of a separate rate for curves (25 frames). If the steering values exceeded a magnitude of 0.05, the controller would use the lower refresh rate for curves to gather more data about left and right turns. Otherwise, the vehicle would default to using the original fixed rate.

Data collection was something that was deemed as quite a worry originally as it was expected that it would take a long time and would also require a powerful computer with a dedicated GPU, but due to being able to have access to a virtual machine on the campus with a dedicated GPU it meant that we could run the data collection script without having to rely on an individual's computer. To collect the data for the models, the autopilot script had to be run for around 7 hours in which time a total of 10,840 images were generated. The specs for the remote computer were the following, Windows 10, Xeon 3.7Ghz CPU, 32GB RAM, and 5GB Nvidia Quadro P220 GPU.

4.2. Lane Detection

Two different approaches were implemented to perform lane detection in the Carla Simulator. The first method consisted of image processing. The algorithm starts with defining a triangular region of interest in the image to select mainly the street and applying a black mask to the rest of the image. Then, we convert the image into grey scale in order to find the edges inside our region of interest. Finally, in the end of this algorithm, the lanes are remarked. The results of this method are shown in Figure 14. This technique showed a poor performance especially when there was a mix between the sun and the shade.

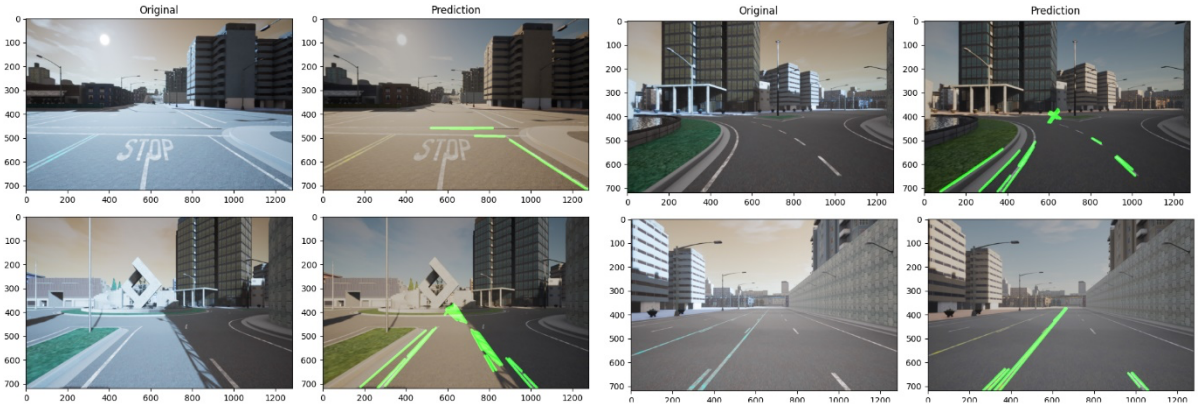


Figure 14: Results of Image Processing Lane Detection with Image Processing

The second method implemented for lane detection was a CNN based encoder-decoder model with 7 convolutional layers, and 4 deconvolutional (up-sampling) layers, took from Mvirgo[1]. The input and output for this model was both 80x160 for height and width respectably. The total number of trainable parameters was 181,687. According to Mvirgo, this model was trained using 21,054 images generated from 12 videos with different weathers traffic and road curves. The model achieved 0.0046 and 0.0048 training and validation loss after been trained with mean square error function.

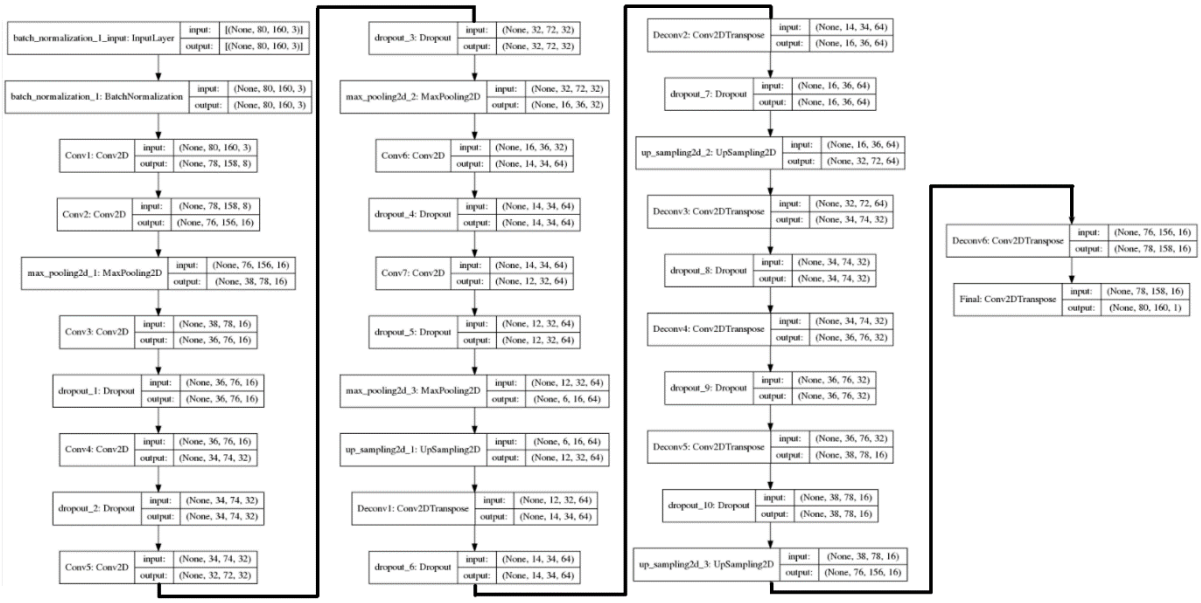


Figure 15: CNN encoder-decoder model for Lane Detection

The results achieved by the CNN encoder-decoder in the Carla Simulator can be seen in Figure 16.

[1] https://github.com/mvirgo/MLND-Capstone

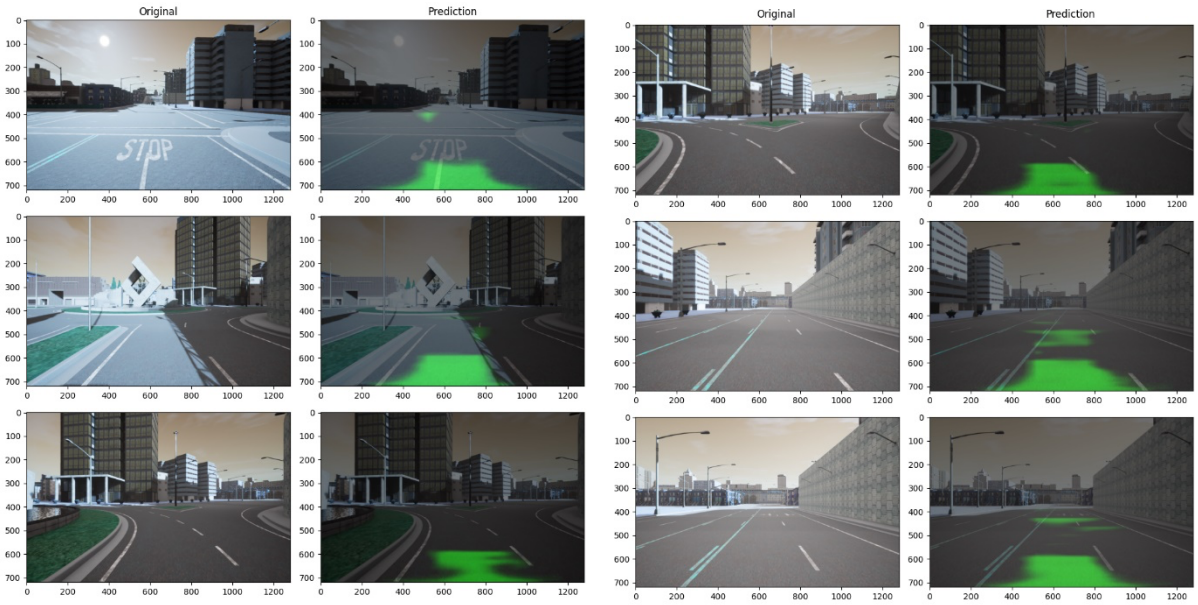


Figure 16: Results achieved with CNN encoder-decoder in Carla Simulator

4.3. Object Detection

In object detection, we are interested in finding all objects of different classes in an image. These detections are usually represented with bounding boxes. The most popular datasets for object detection are ImageNet[37] and the COCO[38]. The first model using the YOLOv3 for objection detection and the second model COCO dataset for object masking.

YOLOv3 produces the output with the Bounding box, Labels and the Confidence of the Object in the Image. The Output image is rescaled to a pixel (360,640). In Mask RCNN COCO model, the output is generated with the class name along with the mask to the object in the image.

These two models were faster in detecting the objects with good confidence. In testing with our own dataset, both the models precisely identify the various object.

Model confidence rate result. [Figure 17]

person is in [92, 299, 136, 405] with 98% confidence
person is in [186, 311, 215, 392] with 87% confidence
car is in [730, 286, 1035, 452] with 76% confidence

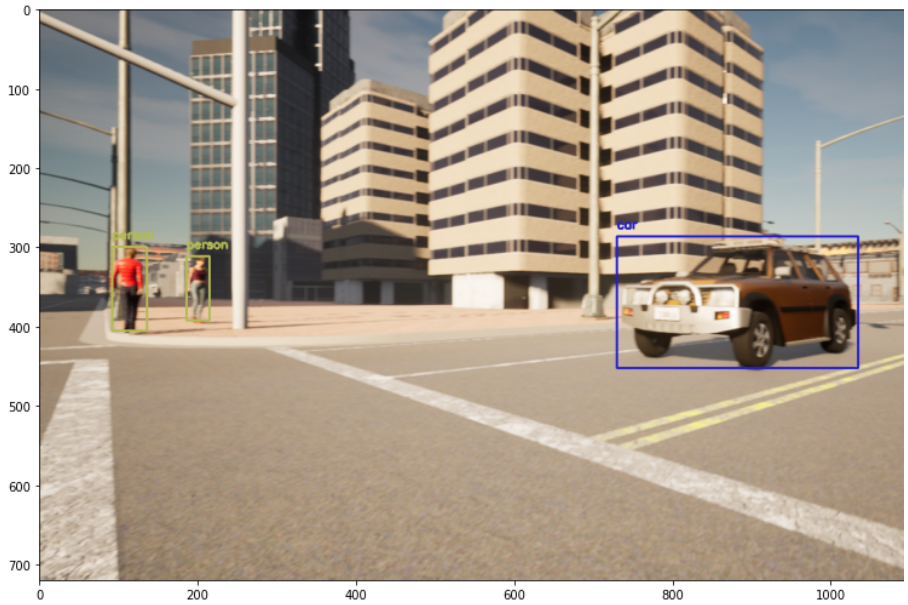


Figure 17: Image with vehicles and pedestrians

Model confidence rate result. [Figure 18]

person is in [812, 314, 845, 400] with 92% confidence
car is in [720, 350, 789, 378] with 90% confidence
bicycle is in [816, 367, 843, 417] with 85% confidence
motorcycle is in [488, 355, 529, 379] with 57% confidence

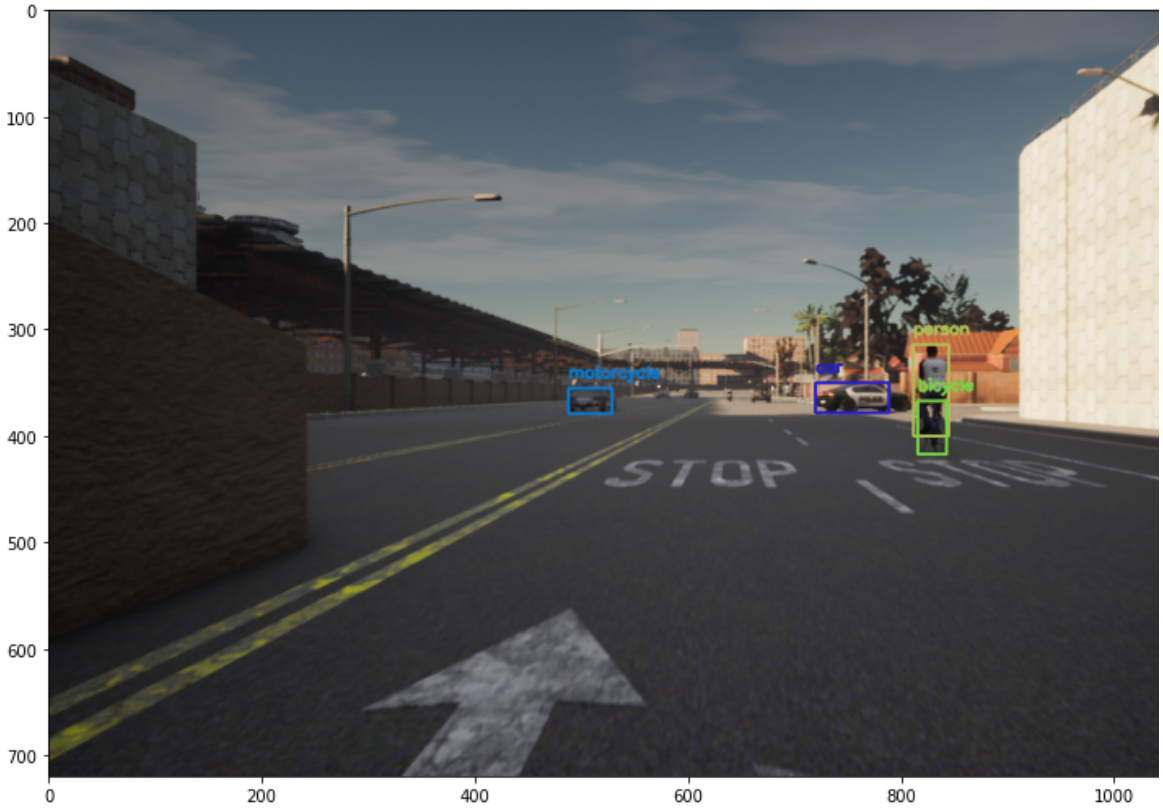


Figure 18: Another image with vehicles and pedestrians

A similar object detection technique was implemented using a MobileNet model. This model was pre-trained with the COCO dataset[2], which is large-scale object detection, segmentation, and captioning dataset. This model was set up to have a fast performance on mobile phones such as Pixel 3 on Android 10. This made the model feasible to run inside the Carla Simulator. The results achieved by this model can be seen in Figure 19.

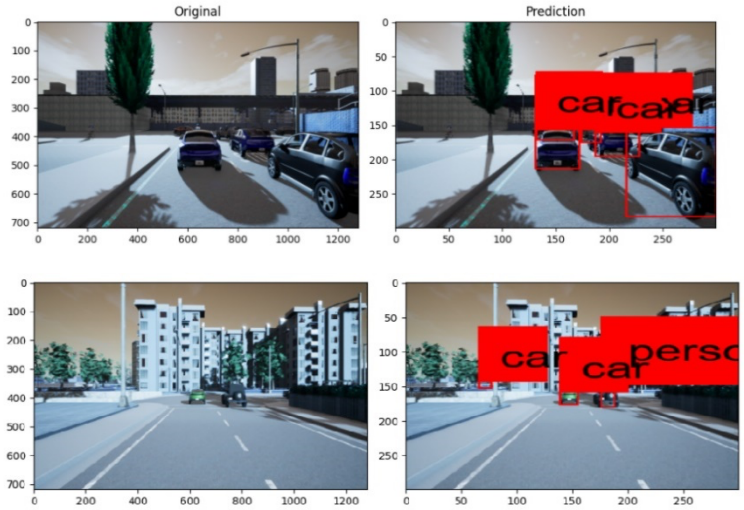


Figure 19: Object detection with MobileNet in Carla Simulator

4.4. Traffic Sign Detection

The traffic sign detection module was implemented with the creation of a CNN model. The dataset for this was obtained from [39]. The dataset contains train and test data for 43 different traffic signs. This approach involved several steps. At first, pre-processing of the images are done. This involves the images being resized and converted into a NumPy array. The next step in the implementation is splitting the data to train and test set with 80% of the data reserved for training and 20% for testing. One hot encoding is applied after this on the categorical data to ensure they become more expressive. After this is accomplished, model creation is done. A sequential model was used for this. Layers have been added to build the CNN. The first 2 Conv2D layers involve the use of 32 filters and a kernel size of 5,5. Within the MaxPool2D layer, the pool size is set to 2,2. This ensures that a maximum value would be selected of every 2x2 area of the image. This in consequence will reduce the image by a factor of 2. The dropout rate is set at 0.25 in the dropout layer. This ensures that 25% of neurons are removed with randomness involved. The 3 layers are applied a second time but with some modifications to the parameters. A flatten layer is then applied next to convert the 2-D data into a 1-D vector. 3 more layers are added to the model which are a dense layer, dropout layer and another dense layer. The second dense layer outputs 43 nodes as the traffic signs and this is divided to 43 categories in the dataset. This layer involves the use of the SoftMax activation function. This gives a probability value which then predicts the category from the available 43 categories for which it has the highest probability. After the model was created, it was compiled, and fit was applied. The batch size was set to 32 and the number of epochs was set at 5 for the model fit. Figure 20 shows a graph for the accuracy and loss of the training and validation dataset [40].

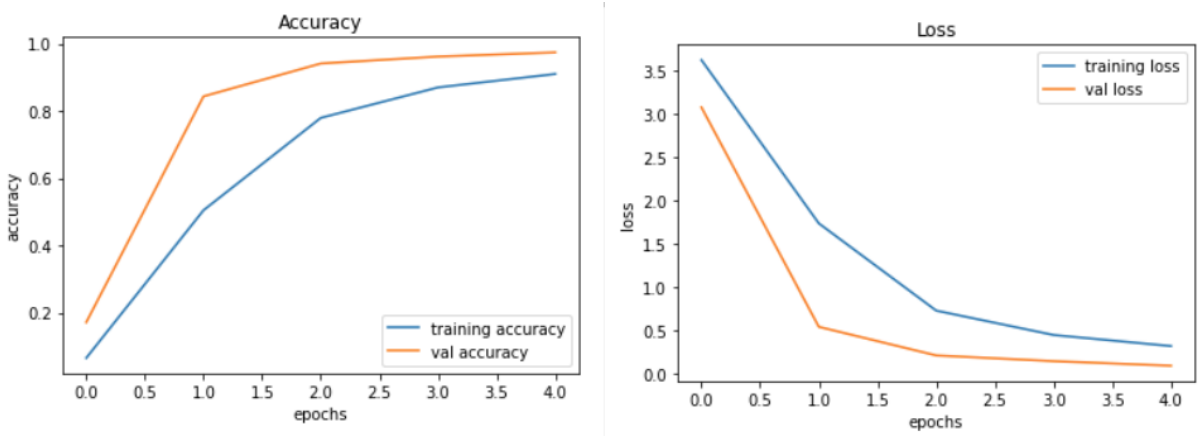


Figure 20: Accuracy and loss from the CNN model trained and validated

The next step was to test the model on a test dataset and obtain and accuracy. An accuracy of 94% was achieved. The final step was to classify the images into their respective category on chosen images. Figure 21 shows the results obtained for this.

Figure 21: Classification of images from the CNN model learnt

4.5. Model Building

Two separate model proposals were developed in parallel to evaluate model driving from different angles.

Single branch model with lane augmentation

The first approach relied on a sequential model with a single branch that took as an input a four-layered image: three layers for RGB and an additional fourth layer that contained the

mask output from the lane detection model. The first experiment for this model relied on 3 convolutional layers followed by dropout layers and a fully connected layer:

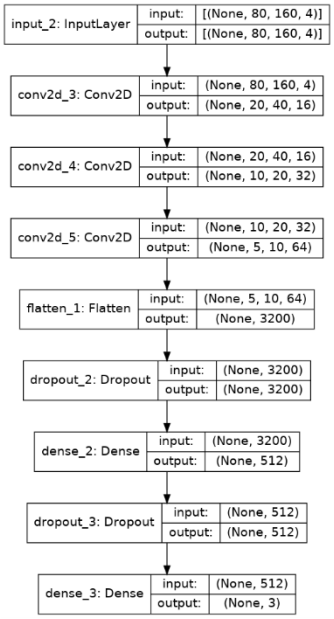


Figure 22: First architecture for the single branch experiment

However, a second model with more trainable parameters and layers was also developed, in the hopes that the total loss could be reduced. This model was built using the Xception architecture, which has been pre-trained to detect images using multiple convolutional layers. Using transfer learning, the model was adapted to predict the vehicle control values needed based on the augmented input image. It had the following architecture:

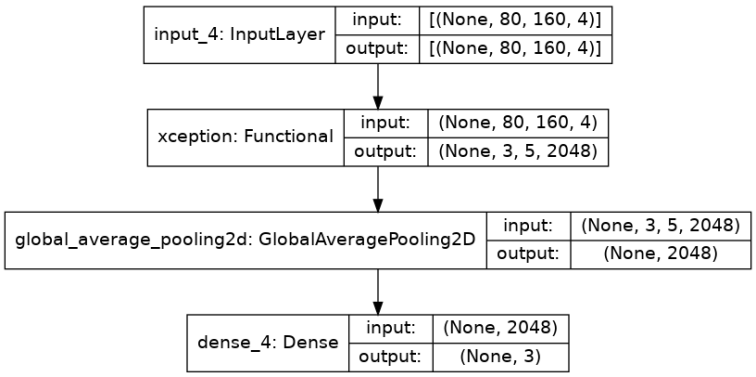


Figure 23: Second architecture for the single branch experiment

The models were trained for a maximum of 200 epochs, with early stopping callbacks that could halt training if validation loss did not improve for 5 epochs. In order to generate the validation set, 30% of the data generated by the autopilot was split into a separate set. Learning rate was set for 1e-4 and images were grouped into batches of 32.

The first model converged after 64 epochs and obtained a validation loss of 0.0112, while the second model stopped training after 42 epochs and achieved a validation loss of 0.0092.

Triple branch model

The models developed with this structure were based on the convolutional neural network proposed by NVIDIA corporation on [22] for autonomous driving. The first model architecture, shown in Figure 24 and available in the car_control_v5, was made of one common input layer and three branches, each of them for predicting steering, throttle and the brake respectively.

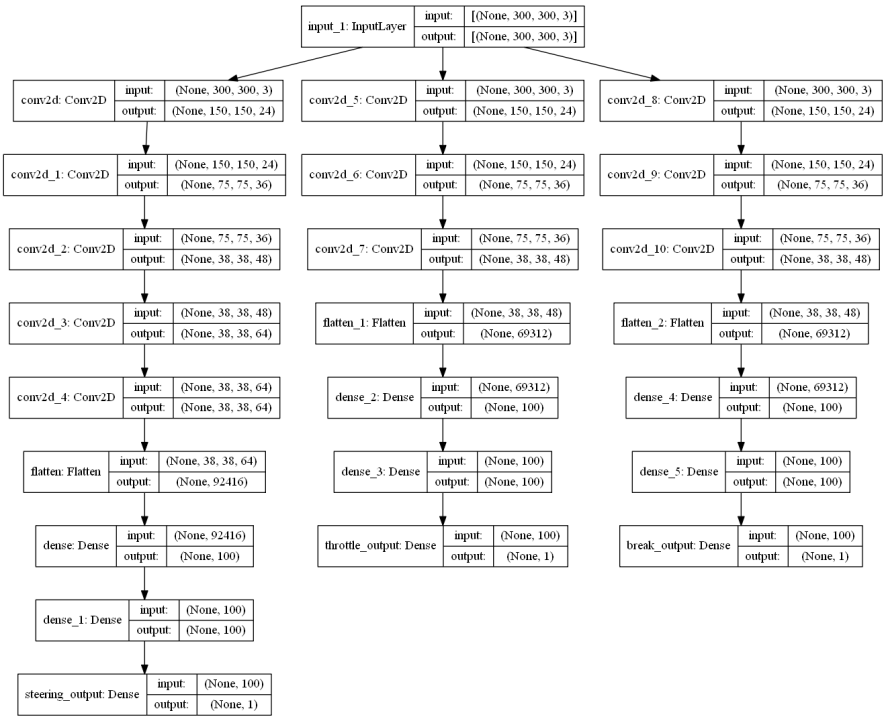


Figure 24: Three-branch model architecture version one

The steering branch contained 5 convolutional layers, two dense layers and one output with sigmoid activation function. For the throttle and break branches, both contained three convolutional layers two dense layers and an output layer also with sigmoid activation function. In total, this model contained 23,399,667 trainable parameters and was trained using 7000 examples (images), Adam optimizer and mean square error function. The loss achieved in training and validation were both around 0.26.

The second model architecture is illustrated in Figure 25 and available in car_control_v6. Unlike the first model, this contained one common trunk of 6 convolutional layers with batch normalization and a flatten layer in the end of these convolution blocks. From this point, each output was made of a single dense output layer, each with sigmoid activation function. The number of total trainable parameters were 286,443. This model was also trained using 7000 examples, Adam optimizer and mean square error function. The loss achieved in training and validation were 0.02 and 0.03 respectively.

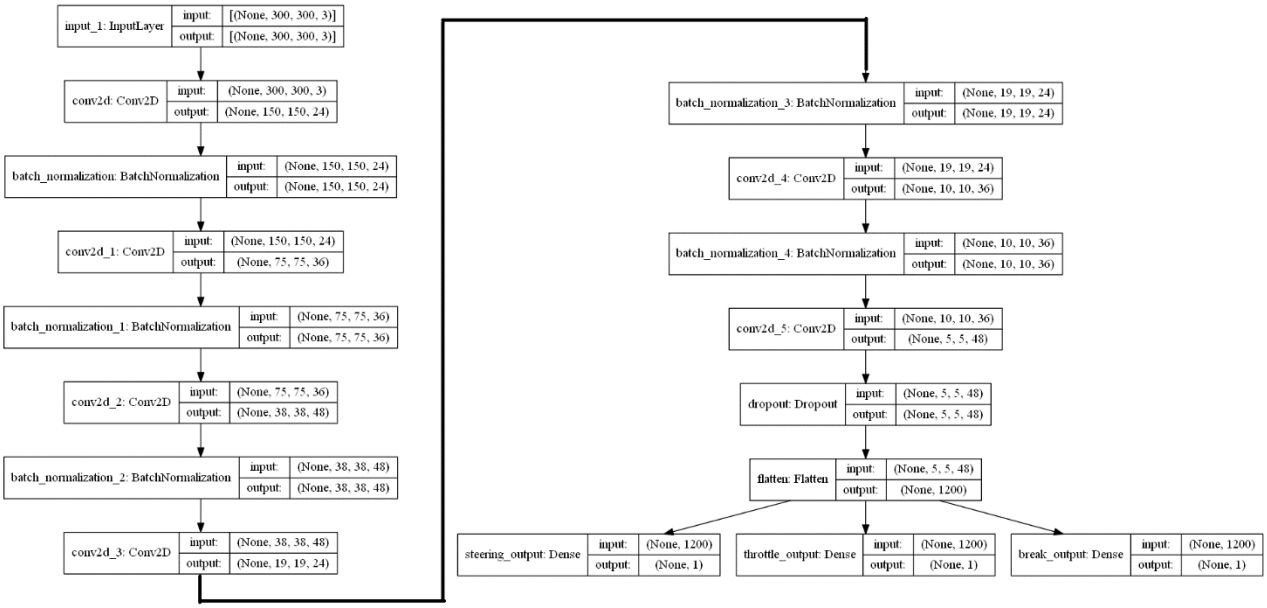


Figure 25: Three-branch model architecture version two

4.6. Exploratory Analysis

Exploratory data analysis is an important step that needs to be considered when using any data to train a model. What the team had originally thought was that due to the layout of the CARLA environment there were significantly more straight roads. What this meant was that originally the data that was obtained had a bias, where the spread of data generally meant that there was more collected data from when the vehicle was travelling in a straight line and the data that was available for when the vehicle was making a turn it was significantly smaller. To rectify this issue, a small adjustment in the data collection script had been made. What this meant was that the script was now taking data point entries more frequently when steering was applied.

Even when the data had been recollected and used again within the model, the agent would still not be able to travel in a straight line and would consistently turn either to the right or the left. To find the root cause of the issue troubleshooting steps had been taken, the first was the exploratory analysis of this data to see whether this newly collected data was to answer for the model's performance.

There were a total of 10840 entries within the dataset, a histogram is shown in the figure below. What this highlighted was the fact that the data was generally quite evenly spread between left, straight, and right. However, it could be seen that there were a few entries greater than 0.4 (right turning), but this was difficult to be seen from the histogram and so a boxplot had then been used to highlight any outliers that were present, this is shown just after.

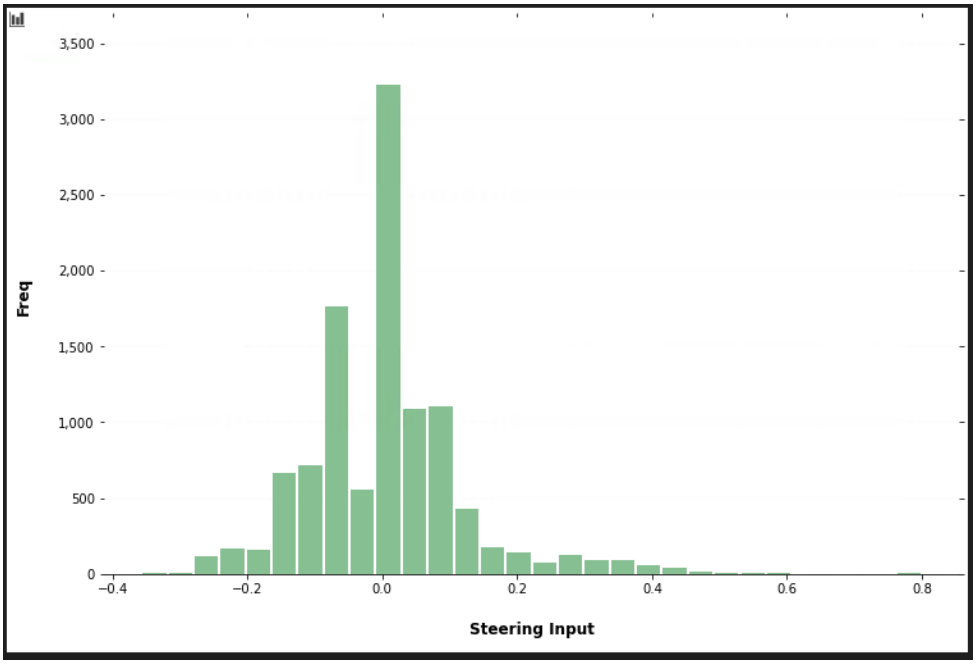


Figure 26 A histogram showing the steering input range from all 10840 entries

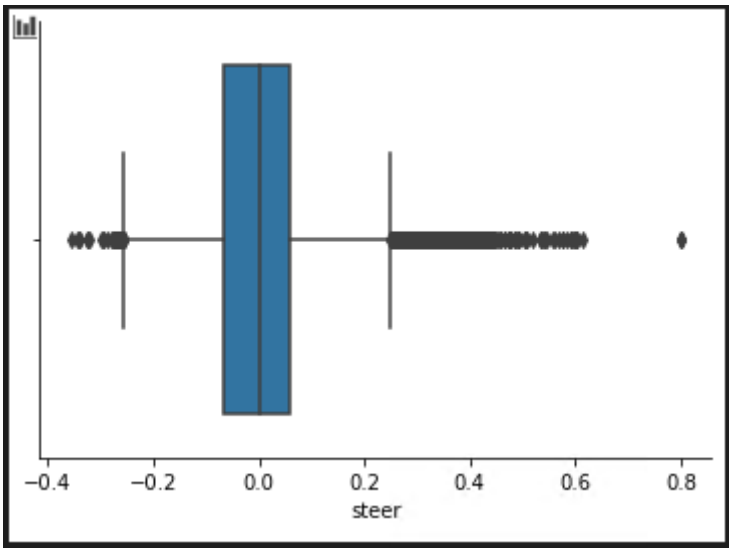


Figure 27 A boxplot showing the spread of steering input from all 10840 entries

In the boxplot shown above it was clearly visible that there was a bigger spread of outliers on the right-hand side, a few entries even extended up till 0.8. This was quite difficult to understand as this would be a very significant turning and could be as a result of the vehicle potentially making a U-turn within the road, however this was not able to be proven. The findings did illustrate that generally the main features from the data such as the min, max, upper and lower quartiles were evenly spread between the left and right. This showed us that there wasn't an issue within the data itself being skewed, but it was more to do with the models themselves.

A further investigation was then made to see how the vehicle's steering and speed is correlated with each other. A plot was created and this shown in the figure below.

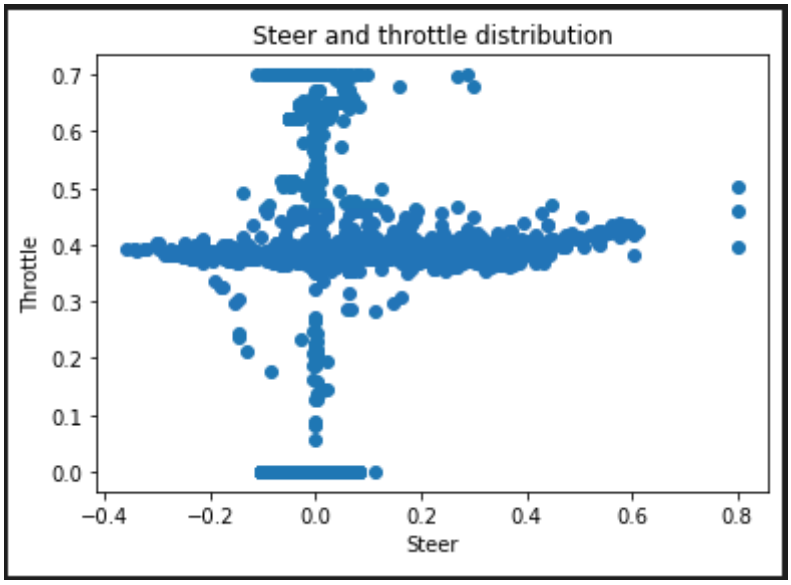


Figure 28 Plot showing a distribution of steering and throttle (10840 entries)

An interesting fact about the way the CARLA autopilot runs was learnt from this, as previously it was unbeknownst to us as to what speed the vehicle was travelling at normally. The steering and throttle distribution were able to easily illustrate to us, that whenever the vehicle was making a turning the throttle input would always drop to around 0.4. What this plot also then showed us was that within the dataset there were still entries where the vehicle was not moving at all, this did then ask the question as to how much of the data was there when the vehicle was stationary.

To remove the points within the dataset where there was no throttle being applied, we had removed any entry when there was braking taking place. This was due to the fact that the CARLA simulator would automatically remove any throttle input when the brake was applied. Once those entries had been removed the dataset had shrunk in size from the original 10,840 down to 8209. The results of which are presented below.

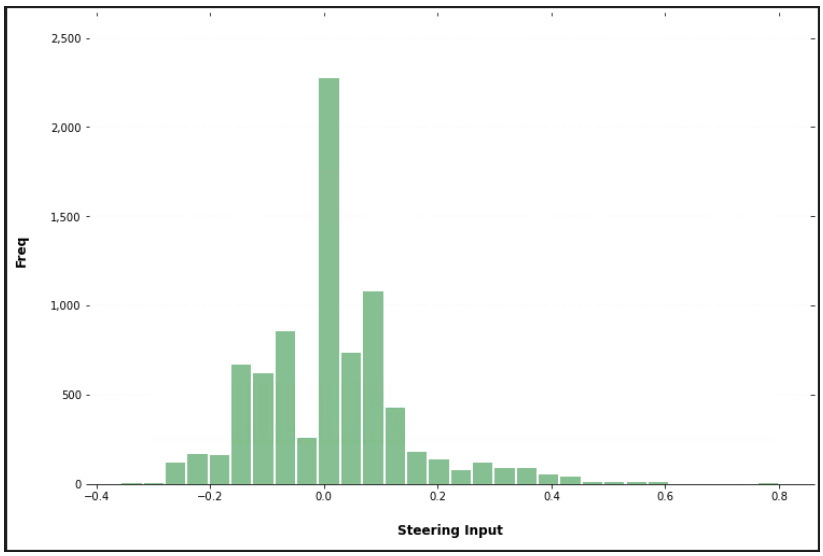


Figure 29 A histogram showing the steering input range from when the vehicle was not stationary (8290 entries)

The histogram shown in the figure above highlights that the main column affected by the removal of the stationary records was when the steering input was relatively straight.

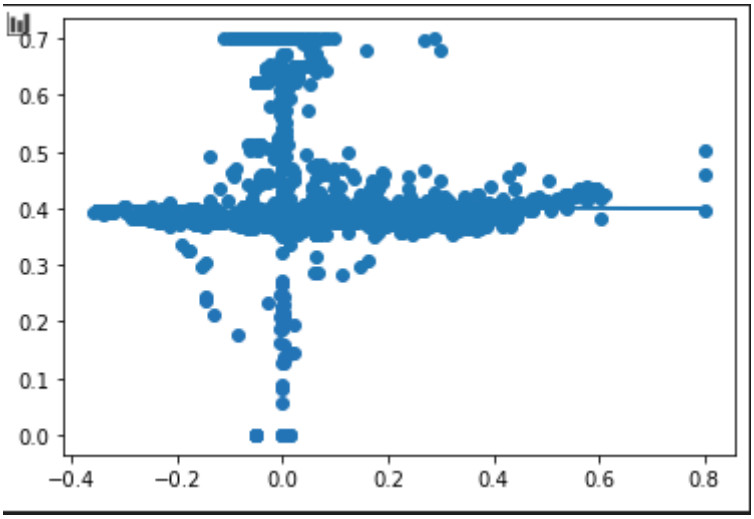


Figure 30 Plot showing a distribution of steering and throttle (8290 entries)

The main thing that we wanted to identify through these second set of diagrams was if the vehicle kept its wheels straight when stationary, as if this was the case then this could be the reason why the models may not have been performing as well we thought they would.

Overall, the findings from the EDA had highlighted to us that the data that had been collected was of a good quality and was not skewed. This then leaves the issue to be within the models themselves.

5. Implementation

5.1. Autopilot Model

As mentioned previously, the Autopilot module is based on the "manual_control.py" script provided by the CARLA documentation. Only slight modifications were needed to fully adapt the script to our data recollection needs.

In order to save the images, we use the carla.Image.save_to_disk() method. After an image has been written in the appropriate directory, an entry with the image id, steering, throttle and braking values is saved into a csv using Python's native csv library.

5.2. Lane Detection

The first lane detection model which uses image processing techniques was implemented mainly with the Python library OpenCV. The creation of a triangular region of interest and the black mask was achieved with cv.fillPoly and cv.bitwise_and respectively. Then, to detect the edges we used the Canny edge detection algorithm available in openCV[3]. Finally, the lanes were complemented using openCV Hough Line Transform[4]. An example of the implementation of this algorithm is available in experiments/lane_detection/exp2-lane_opencv.

For our second approach, the encoder-decoder model developed by Mvirgo was built using the Tensorflow Keras API. To build the network the module tf.keras.model.sequential was used, convolutional layers with tf.keras.layers.Convolution2D andtf.keras.layers.Dense for the fully connected layers. The implementation of this algorithm can be reviewed on the directory experiments/lane_detection/exp1-lane_network.

5.3. Object Detection

After collecting ten thousand images from the five towns in the CARLA simulator, the images are reshaped using the OpenCV randomly. After pre-processing, it is resized to (640,480) using the python-resize-image library, the image colour converted from BGR to RGB using cvtColor(). They are then transferred into the mask-rcnn coco model with the pre-trained weights. Detecting common objects in the town using the function model. detect () inferencing one image at a time in a batch.

It will return the bounding box coordinates, corresponding labels, and confidence scores for the detected objects in the scene.

The MobileNet model implemented for object detection was built with TensorFlow[5]. This model comes in a Tensorflow Lite format, which is a special format for mobile and embedded deployment

[3] Canny Edge Detection in OpenCV, available in https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_canny/py_canny.html
[4] openCV Hough Line Transform, available in: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_houghlines/py_houghlines.html
[M]odel available in: https://www.tensorflow.org/lite/guide/hosted_models

5.4. Traffic Sign Detection

For the implementation of the CNN model for traffic sign detection, Python 3 was used. Image pre-processing was done using the Python Imaging Library and NumPy. The data was split into train and test set via the use of the sklearn library where this had a method called train_test_split for data splitting. The data was one-hot encoded with the use of the keras' to_categorical method. The model was built using keras, specifically the Sequential class. The layer from keras which were used included Conv2D, MaxPool2D, Dense, Flatten and Dropout. After the creation of the model, it was saved as .h5 format. Matplotlib was used to display graphs of accuracy and loss from the training of the data via model.fit. Pandas was used to read a csv file to obtain sources for the test images to be predicted. The accuracy score was measured using sklearn's accuracy_score method. Finally, matplotlib was again used to plot image with their category being as the label of plotted images. These categories were obtained from a dictionary where the key was the number for the folder the images were in and the value being the description of the traffic sign, for instance, a "stop" traffic sign. The model was implemented by creating a Python class with a separate method for displaying the images with their corresponding category. This was done in a python file. A jupyter notebook was also created to set the parameters for the model and to executing the methods within the TrafficSign class along with executing the separate classify_images method.

5.5. Controller Model

In order to spawn the vehicle and set the appropriate attributes we need to make use of CARLA's Python API. It includes several utility methods to spawn actors in the CARLA server.

When initializing the controller, the lane detection and driving models are loaded into the class by using Keras' load_model() method.

In order to reshape and manipulate the raw image, we make use of numpy and OpenCV for Python. After the image has been pre-processed, we can then feed the image into the lane detection model. The result (the lane mask) is then concatenated to the original image also using numpy. The enhanced image is then fed into the driving model, which then outputs the vehicle control values.

These values are then applied to the vehicle by using the apply_control() method available to all CARLA vehicles.

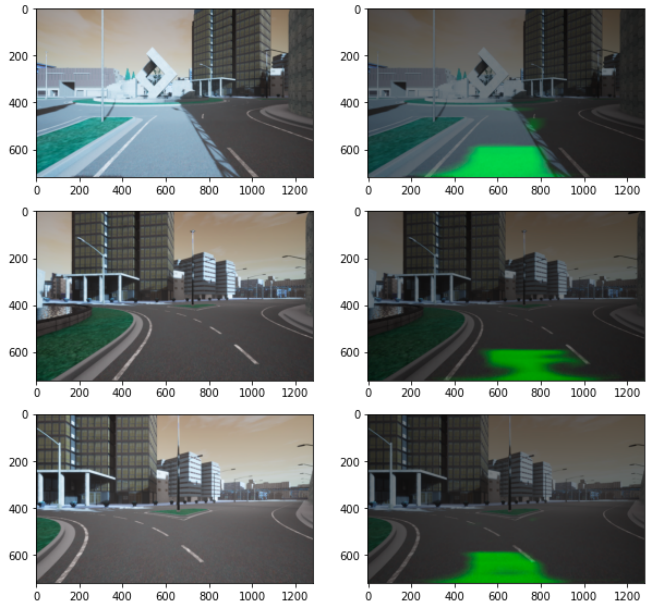
6. Testing

ID: TC001	Forwards movement and stopping
Objective: The test seeks to ensure that the model has appropriate control over the car acceleration and brakes. This basic output actions will serve as the basis for plan execution at later stages of development.	
Preconditions: The car should have space to move forwards.	
Inputs: 1. Instantiate a scene with the car and enough terrain to maneuver. 2. Instruct the car to move forwards 3. After the car has accelerated, instruct the car to stop	**Expected outputs:** 1. The car first accelerates, moves forwards and after a desired amount stops.
Result: The car is spawned on the scene, moves forwards and then comes to a complete stop. 	**[SUCCESS]**

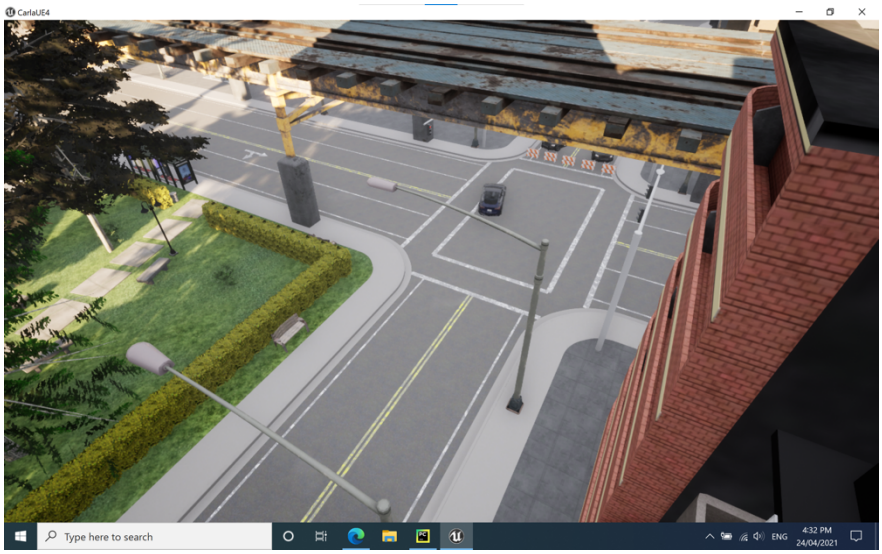
ID: TC002	Steering
Objective: The test seeks to ensure that the model has appropriate control over the car steering. This basic output actions will serve as the basis for plan execution at later stages of development.	
Preconditions: The car should be moving. It should also have enough lateral space to steer right and left.	
Inputs: 1. Instantiate a scene with the car and enough terrain to maneuver. 2. Instruct the car to go left 3. After the car has moved left, instruct the car to go right	**Expected outputs:** 1. The car first moves left and then right

Result: The car starts to steer to the left and then turns right. 	**[SUCCESS]**

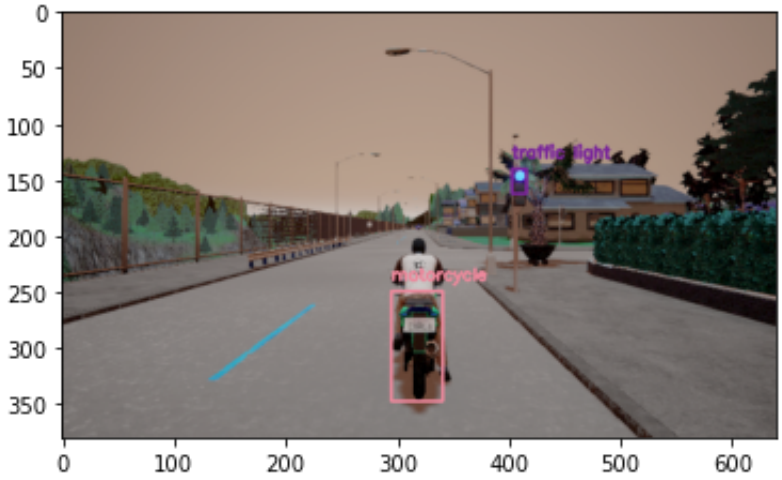
ID: TC003	Lane detection
Objective: The test verifies that the car can identify lanes in its immediate path.	
Preconditions: There are no other vehicles or actors on the scene.	
Inputs: 1. Recollect images from the autopilot 2. Select images from the dataset 3. Pre-process them for the lane detection module 4. Run lane detection prediction using the selected images	**Expected outputs:** 1. The lane ahead of the vehicle should be highlighted green.
Result: The lane detection properly paints the lanes ahead of the vehicle.	**[SUCCESS]**



ID: TC004	Straight path following
Objective: The test verifies that the car can stay on its appropriate lane, when the street is on a straight road.	
Preconditions: There are no other vehicles or actors on the scene. The road in front of the car has no curves.	
Inputs: 5. Instantiate a scene with the car and a clear straight path. 6. Instruct the car to perform path following	**Expected outputs:** 2. The car moves forwards and sticks to its own lane.
Result: The car slowly steers towards one side (usually to the left), invades the adjacent lane and slowly continues until it finds itself going off-road. The car usually gets stuck after crashing into wall in the surrounding buildings. 	**[FAIL]**

ID: TC005	Curved path following
Objective: The test verifies that the car can follow a curved path.	
Preconditions: There are no other vehicles or actors on the scene. The road in front of the car has a curve.	
Inputs: 1. Instantiate a scene with the car and a clear curved path. 2. Instruct the car to perform path following	**Expected outputs:** 1. The car follows the curve without going off-road.
Result: The car starts to follow the curve, but the steering isn't sufficiently aggressive. As such, the car ends up going off-road at some point of the curve.  	**[FAIL]**

ID: TC006	Object detection
Objective: The test verifies that the car can identify other vehicles and traffic lights.	
Preconditions: There are visible objects in the vehicle's field of vision.	
Inputs: 1. Recollect images from the autopilot 2. Select images from the dataset 3. Pre-process them for the object detection module 4. Run object detection prediction using the selected images	**Expected outputs:** 1. The vehicles, pedestrians and traffic lights should be highlighted with bounding boxes/masks.

Result: The object detection properly identifies and labels the surrounding objects in the vehicle's field of vision. 	**[SUCCESS]**

ID: TC007	Traffic sign detection
Objective: The test verifies that the car can detect traffic signs on the road.	
Preconditions: There is a visible traffic sign on the road in range of the car camera.	
Inputs: 1. Create an environment for the car with a traffic sign in sight 2. Select the image and preprocess it 3. Run traffic sign prediction for the saved model	**Expected outputs:** An image displayed of the traffic sign and a label indicating the category of the traffic sign
Result: The traffic sign detection properly identifies and labels the image with the correct traffic sign category. 	**[SUCCESS]**

7. Discussion and Further Work

Despite some unsuccessful test trials, the research and experiments conducted for this project still yield relevant findings. Most notably, the team was successfully able to mount and adapt several pre-trained models for lane and object detection into the CARLA simulation. Having multiple approaches for each of the detection modules also allowed the team to assess the overall effectiveness of each of the strategies.

For lane detection, the first approach (which relied on Hough transforms and image manipulation) was a very fast and cost-effective method of identifying the bounds of a lane. However, as one can observe from the experimental images, the method is not as functional with curved lanes and it is often deceived by shading artifacts. This can be mostly attributed to the complexity of the model. When compared to our second approach (which relied on convolutional neural networks), Hough transforms are a much simpler feature extraction technique. Nevertheless, for vehicles with limited processing power, this alternative might be quite useful for simple lane detection. Still, as explored during our literature review, even convolutional neural networks have been optimized to a very competent degree, making them viable even with limited processing power.

Our second approach for lane detection does have certain limitations. As observed from the experimental images, the mask definitely identifies the rough are that encompasses a lane. However, it is far from perfect. It sometimes does not fully identify the path and, in other instances, it even recognizes areas outside of the lane as valid. An improvement to this model, which we were unfortunately not able to test due to lack of time, was to define a mask for the CNN in a similar way to how we pre-process for the Hough transform approach. It would involve only making predictions for a predefined area ahead of the vehicle, in the form of a triangle. However, this could have certainty hindered the lane detection performance for curves, as these types of lanes do not conform to the triangular mask. Perhaps using a trapezoidal mask with a wider base could better predict curves, while also excluding nonsensical predictions outside of the area of interest. Regardless of these observations, the CNN-based lane detection module was still fairly effective, which is why this was considered a success for the project.

In terms of object detection, we also achieved considerable success with models that could accurately detect persons and other vehicles (such as cars, trucks, and bikes). Unfortunately, as most of these changes were implemented at a very late point of the development cycle, the detection model could not be properly fit into the car control pipeline. In order to do so, we would have followed a very similar approach to the lane detection module. We would first run the object detection model and generate a mask for each type of object. Then, we would append the mask as an additional layer for the car control model. Considering that the autopilot is fairly aggressive about braking when in the presence of a vehicle or a pedestrian, it is our hypothesis that such mask would have very direct effects on the braking of the vehicle. However, this is also a potential avenue of research or future work.

Given more time we would have also ideally experimented with more model architectures that used object detection. For our single branch model, this would have entailed a very similar workflow to the one described above – adding the extra information about object detection as an extra layer in the image input for the car control model. However, for our multi-branch approach we would have probably created separate convolutional layers for this detection component. This would have resulted in a much more complex model, with more potential to model intricate relationships between visual cues and vehicle controls.

On the topic of different architectures, comparing the single branch model to the multi-branch model was also a big focus of the research that is worth discussing. Our hypothesis during the development of both architectures suggested that the single branch model would have perhaps been too simple to mode the complex nature of driving solely from images. We believed that the multi-branch model would achieve a better performance; however, we also presumed the model could suffer from overfitting. To our surprise, the single branch model obtained a better performance, even with less training parameters. The nature of such behaviour is quite enigmatic, as there are multiple factors to consider. Perhaps the problem was not as complex as we had initially conceived, and it could indeed be modelled with a simpler architecture.

Another quite likely explanation for such a behaviour could be that each of the car controls (steering, throttle, and braking) is inherently related. A car would naturally not need to both brake and accelerate at the same time. Likewise, from our EDA, we observed that the autopilot commonly took curves with a predefined steering angle. Since much of these outputs are connected and dependent on one another, having a common architecture and developing feature extraction layers that are shared between the several controls could have positive results.

It is also possible that both the single- and multi-branch architectures are both stuck at a suboptimal local point in the learning space or that the pre-processing of the data isn't sufficient to develop actual useful driving patterns. As such, the better performance of the single branch architecture could very well just be the result of merely chance. With much more thorough image pre-processing and a greater focus on hyper-parameter tuning, the multi-branch architecture might be able to better model autonomous driving. This hyper-parameter tuning is left as another possible future avenue of research.

A matter worth discussing is the root cause for the model's inability to properly drive. This is quite a complex task as there are multiple failure points in the project pipeline. Our first hypothesis revolved around the data recollection. Since the model was mostly failing at taking curved paths, we believed that the data might have been improperly skewed. More specifically, we believed that there were not sufficient entries in our training data for these scenarios. As such, even a fairly complex model would not be able to learn how to steer left or right. From our experience driving real cars, we also believed that steering would greatly vary even for a same curve. We believed steering would gradually increase, until reaching a maximum point. After this, we thought the model would then progressively decrease until coming to the neutral state. As such, we decided to recollect much more data for curved paths.

However, as noted in our Methodology, this just introduced a new bias for the model to steer too aggressively. Despite this, we did notice improved steering on curves. Therefore, we concluded that the model was probably failing to learn due to the CNN architecture. This is the main reason why we tried to test different model approaches.

It is worth noting that there is an additional improvement, which we were unable to apply due to time constraints. In order to process the data and train the models in a reasonable amount of time the images, which are originally saved at a 1280x720 resolution, are downscaled. This applies to both our single- and multi-branch models. Shrinking the image to lower resolutions might result in a considerable loss of details. Since we weren't able to integrate object detection, we could not experiment much in this regard. But we theorized that small objects, such as traffic lights, might not even be perceptible once the image is downscaled. Ideally, the model should take the images at their full resolution and process them accordingly. This would ensure all details remained. However, this would severely slow down training and processing in real time. For a vehicle, which needs to act rapidly, this might not be viable. But with newer and faster CNN models, the complexity of the input could be preserved and these models could perhaps output predictions in time. This is another possible line of research for the future.

Another interesting observation that has substantial repercussions on the training of our model is the non-deterministic behaviour of the autopilot. At a junction, the autopilot can maintain a straight course or take a turn in one of many directions. Since the data collected lacks this contextual piece of information, for a given scenario (reaching a junction) there might be conflictive data for the model. In some instances, the data might indicate that the model should go straight; but it could also suggest that it should take a turn. It would have been interesting to analyse this behaviour and determine whether it influences our model. Furthermore, given multiple courses of action, could the model change directive. For example, the model might initially try to go straight, but at an intermediate point, the model might suggest taking a turn based on the data. Perhaps, a higher-level model that constrained the model actions would be suited here. This model would lock the vehicles actions until the desired result is obtained, in a similar fashion to transactions in a concurrent or multi-threaded system.

The team also discussed implementing a second level model which further processed the outputs of the underlying vehicle control model. It would round the suggested actions to a nearest decimal (for example, steering to the nearest tenth). The goal was to supress small noise in the outputs, like a minuscule steering suggestion which evidently should have been no steering at all. However, the team did not get around to implement this feature and test it to its fullest extent.

8. Project Management

8.1. Structure and Methodology

As had been discussed within the software requirement specification the development approach that had been taken for this project was an agile approach. This approach proved to be the most beneficial as it allowed for the team to be able to split into smaller subsections where everyone had their own specific task given to them. The development was broken down into weekly sprints, and by using Jira we were all able to monitor the progress of each individual as well as also making a note of what tasks were still remaining to be completed. The weekly meeting allowed for the team to address any issues that they may have been having as well as allowing for a discussion into the next steps and obstacles. In addition to this the team were able to communicate between meetings by using WhatsApp as a means to contact the team quickly with any questions or ideas that they may have had.

Overall, there were 5 sprints including the initial sprint to complete the SRS document. The SRS document had been broken down into its component headings, and each heading became a task within the first sprint. The tasks for the initial SRS document are as follows:

- SRS Document Sprint
  - System Modelling
  - Introduction
  - Requirements Specification
  - System Modelling
  - Test Management
  - Project Management

After the initial sprint to complete the SRS document was completed, the project itself was broken down into smaller tasks. Each week there would be a meeting to discuss the progress for everyone on their respective tasks, it was often the case that tasks could be extended onto the following week / sprint.

Within the first development sprint a total of 9 tasks had been put forward, 4 of which had been completed within the first week. The tasks that had been completed were model detection for street lanes, CARLA simulator car control and camera sensor simulator. For each of the completed task a comment was also left to evidence where the output was located.

Due to some of the tasks requiring a more work they were continued onto the following sprint, what was extended from the first to the second was the:

- Extended from Sprint 1 to Sprint 2
  - Model detection traffic signs
  - Algorithm to drive the car

- Research data generation and ingestion to model
- Training model with data collected
- Implement object detection (cars) in CARLA

Within the sprint 2, there were new tasks that had been suggested similar to the week prior to this there were multiple tasks that meant they were extended onto subsequent sprints. In this sprint what was completed was, an initial first model was trained using the preliminary data that had been collected. The tasks that had been started this sprint and extended onto sprint 3 were:

- Model detection traffic signs
- Research data ingestion into model
- Implement object detection (cars) in CARLA
- Construct models for CARLA #1
- Construct models for CARLA #2
- Construct models for CARLA #3
- Construct models for CARLA #4
- Construct models for CARLA #5
- Construct models for CARLA #6
- Recollect training data
- Data exploration on the current training dataset (find flaws, improvements)

The reason why there were multiple "Construct models for CARLA" was because on Jira it was not possible to assign multiple people onto a single task and by doing it in this manner it allowed for individual team members who were allocated this task to update their specific task with their findings.

In sprint 3 there was an introduction of one new task, and this was the recollection of training data. Similar to what had been done in the previous sprint there were some advancements within the tasks, however the same tasks that had been extended from sprint 2 into sprint 3 were now extended into the final sprint (sprint 4).

In the sprint 4, this was the final sprint where team members were able to inform us of their progress and generally everyone had completed the tasks that had been set to them. The completed tasks within this sprint were as follows:

- Model detection traffic signs
- Research data ingestion into model
- Implement object detection (cars) in CARLA

- Construct models for CARLA #1
- Construct models for CARLA #2
- Construct models for CARLA #3
- Construct models for CARLA #4
- Construct models for CARLA #5
- Construct models for CARLA #6
- Recollect training data
- Data exploration on the current training dataset (find flaws, improvements)

By using JIRA as a method to monitor what tasks were set for that specific sprint had meant that it was very easy to monitor what was being done each week. It ensured that the agenda for the weekly meetings could easily be set, ensuring that nothing was missed out. GitLab was also another tool that was used by the team as any team member who had been working on any part of the code was able to commit their works onto their own specific branch, the branches had been broken down into sub-teams who were working on different modules.

8.2. Challenges and Strategies

Hardware limitations

The CARLA simulator is a pretty demanding piece of software that requires dedicated hardware to run. Despite barely being able to run on a computer with integrated graphics, the simulation was clearly designed to run on a machine equipped with a dedicated GPU. For the first couple of development weeks, only two team members had access to machines capable of running CARLA, due to either hardware limitations or problems during the installation process.

The team was eventually able to contact the school and get access to a virtual machine with a dedicated GPU. This allowed users who didn't have direct access to appropriate hardware to develop models for CARLA and test their code at a later point of the project's development. Nevertheless, use of the virtual machines still had their downsides. The most notable limitation of the VMs was that their use was limited to a single user at a time. This meant that if several development teams wanted to use the computers, they would have to coordinate themselves. This proved to be especially troublesome when the machines were running data recollection scripts in the background, as these would take some time to complete.

Operating System Issues

CARLA is officially only supported on Windows and Linux. This hindered some of the project's initial progress as some of our team members use macOS as their primary operating system. And while there are threads detailing the complementary actions needed to build CARLA on a Mac machine, the overall process is quite cumbersome and has been reported to be unstable. As such, our macOS users had to rely on alternative means to run the simulator. For some, it meant setting up a dual-boot scheme with either a Windows or Linux partition. However, some team members were working on the new ARM-based M1 chipset, which is

still early on its lifecycle. Most notably, running a Windows or Linux partition is still not as stable as with earlier Mac architectures. For these team members, using the University VMs was the easier solution.

Problems with CARLA

- Some team members were not able to properly connect to the CARLA server, despite following installation instructions and using default parameters. After debugging issues involving the TCP ports, the team eventually arrived at a solution. In order to connect to a server on these faulty installations, the port number for both the client and server had to be manually specified. It also had to be a different port than the default option used by both the client and server. The team's conjecture is that the machines either had a firewall impeding the connection or another application was occupying CARLA's port number.
- Some team members were also not able to run some of the example scripts–namely, the spawn_npc.py script. This script was important as it was used to collect data for the object detection module. Despite the script appears to run properly, actors weren't spawned on the CARLA server. A solution to this issue was never found, but in response to this obstacle, recollecting the data that required this script was delegated to a different team member, who did not have issues spawning NPCs.
- The CARLA library must always be imported at the beginning of a Python script by locating the appropriate .egg file, created during the simulator's installation. Due to a lack of standards in the organization of the project's directory, some paths in our scripts did not point to the appropriate directory with the .egg file, after certain modules were reorganized. The problem was not immediately apparent as some team members manually added the .egg file to their PYTHONPATHs. This meant that even if the path in the script was wrong, the script would still load (having loaded the CARLA Python API from the PYTHONPATH). Eventually, some team members found the issue and corrected the path.

Problems with Python

For a reason unbeknownst to the team, the installation of Python on the virtual machines had a fatal flaw that prevented the installation of further Python libraries. After debugging the root issue, a team member identified that some DLLs required to enable SSL communication were misplaced in the installation. After correctly placing the files in the corresponding folder, the machines were then capable of downloading further libraries and development could carry on.

9. User Documentation

9.1. Installation

1. Download the appropriate build for CARLA 0.9.11 (Windows zip or Ubuntu tar.gz) from https://github.com/carla-simulator/carla/releases.
2. After downloading, navigate to the PythonAPI subdirectory and clone the following repo: https://cseegit.essex.ac.uk/2020_ce903/ce903_team06.git
3. Prepare the Python dependencies by creating a conda virtual env for Python 3.7
4. Install the appropriate libraries by running: `pip install -r requirements.txt`

9.2. How to run

To run Automata:

1. Run the CARLA server by executing CarlaUE4.sh or CarlaEU4.exe (depending on the OS build), located at the topmost level of the WindowsNoEditor folder.
2. On a separate terminal, navigate to WindowsNoEditor/PythonAPI/ce903_team06/automata
3. Run the Main.py script with: `python Main.py`
    - -model (-m): To select model number (1-6)
    - -spawn (-s): To select spawn point (0-255)

To collect more data for training:

1. Run the CARLA server by executing CarlaUE4.sh or CarlaEU4.exe (depending on the OS build), located at the topmost level of the WindowsNoEditor folder.
2. On a separate terminal, navigate to WindowsNoEditor/PythonAPI/ce903_team06/automata
3. Run the Autopilot.py script with: `python Autopilot.py`

10. References

[1] *Vehicle Licensing Statistics: 2020 Quarter 3 (Jul - Sep)*. Department for Transport, 2021, p. 1.

[2] "Topic: Car insurance in the UK", *Statista*, 2020. [Online]. Available: https://www.statista.com/topics/4560/car-insurance-in-the-uk/. [Accessed: 14- Apr- 2021].

[3] "Reported road casualties in Great Britain, provisional estimates: year ending June 2020", *GOV.UK*, 2021. [Online]. Available: https://www.gov.uk/government/statistics/reported-road-casualties-in-great-britain-provisional-estimates-year-ending-june-2020. [Accessed: 14- Apr- 2021].

[4] "The dangers of driving tired | Seton UK", *Seton UK*, 2019. [Online]. Available: https://www.seton.co.uk/legislationwatch/article/the-dangers-of-driving-tired/. [Accessed: 16- Apr- 2021].

[5] F. Duarte and C. A. Ratti, "Senseable City Lab :.:: Massachusetts Institute of Technology The Impact of Autonomous Vehicles on Cities: A Review," 2018, doi: 10.1080/10630732.2018.1493883.

[6] Unmanned vehicles come of age: The DARPA grand challenge https://www.computer.org/csdl/magazine/co/2006/12/04039240/13rRUzphDtw Accessed: 2021-04-23

[7] "Your Autopilot has arrived | Tesla UK." https://www.tesla.com/en_GB/blog/your-autopilot-has-arrived (accessed Apr. 25, 2021).

[8] "Google self-driving car caught on video colliding with bus | Self-driving cars | The Guardian." https://www.theguardian.com/technology/2016/mar/09/google-self-driving-car-crash-video-accident-bus (accessed Apr. 25, 2021).

[9] Car review: Honda Jazz Hybrid EX is an extremely fine piece of engineering | The Independent." https://www.independent.co.uk/life-style/motoring/honda-jazz-hybrid-ex-review-b1815685.html (accessed Apr. 26, 2021)

[10] A. Ajit, K. Acharya and A. Samanta, "A Review of Convolutional Neural Networks," 2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE), 2020, pp. 1-5, doi: 10.1109/ic-ETITE47903.2020.049.

[11] E. Baykal, H. Dogan, M. Ercin, S. Ersoz and M. Ekinci, "Transfer learning with pre-trained deep convolutional neural networks for serous cell classification", *Multimedia Tools and Applications*, vol. 79, no. 21-22, pp. 15593-15611, 2019. Available: 10.1007/s11042-019-07821-9.

[12] P. Vadapalli, *Ultimate Guide to Object Detection Using Deep Learning [2021]*, 2021.

[13] R. Suite, *Introducing Transfer Learning as Your Next Engine to Drive Future Innovations*, 2020.

[14] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2014.

[15] R. Girshick, "Fast R-CNN", in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440-1448.

[16] S. Ren, K. He, R. Girshick and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137-1149, 2016. Available: 10.1109/tpami.2016.2577031.

[17] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779-788.
[18] W. Liu et al., "SSD: Single Shot MultiBox Detector", in *European conference on computer vision, Springer*, Cham, 2016, pp. 21-37.
[19] J. Uijlings, K. van de Sande, T. Gevers and A. Smeulders, "Selective Search for Object Recognition", International Journal of Computer Vision, vol. 104, no. 2, pp. 154-171, 2013. Available: 10.1007/s11263-013-0620-5.
[20] A. Krizhevsky, I. Sutskever and G. Hinton, "ImageNet classification with deep convolutional neural networks", Communications of the ACM, vol. 60, no. 6, pp. 84-90, 2012. Available: 10.1145/3065386.
[21] Zhong-Qiu Zhao, Peng Zheng, Shoutao Xu, and Xindong Wu. Object detection with deep learning: A review. IEEE transactions on neural networks and learning systems, 2019.
[22] Bojarski, M., Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L., Monfort, M., Muller, U., Zhang, J., Zhang, X., Zhao, J., & Zieba, K. (2016). End to End Learning for Self-Driving Cars. ArXiv, abs/1604.07316.
[23] L. Hulstaert, "A Beginner's Guide to Object Detection", *Datacamp*, 2018.
[24] O. Akköse, "A Review of Object Detection Models", *Medium*, 2020.
[25] A. CHOUDHURY, "Top 8 Algorithms for Object Detection", *analyticsindiamag*, 2020.
[26] D. Mwiti, "A 2019 Guide to Object Detection", *Heartbeat*, 2019.
[27] K. He, G. Gkioxari, P. Dollar and R. Girshick, "Mask r-cnn", in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961-2969.
[28] W. Canyong. "Research and Application of Traffic Sign Detection and Recognition Based on Deep Learning". 2018 [Online] Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8410256 [Accessed: 27-Apr-2021]
[29] C. Rahmad, I. F. Rahmah, R. A. Asmara and S. Adhisuwignjo. "Indonesian Traffic Sign Detection and Recognition Using Color and Texture Feature Extraction and SVM Classifier". 2018 [Online] Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8350804 [Accessed: 27-Apr-2021]
[30] Jaafra, Y., Laurent, J.L., Deruyver, A. and Naceur, M.S., 2019. Seeking for Robustness in Reinforcement Learning: Application on Carla Simulator.
[31] Grondman, I., Busoniu, L., Lopes, G.A. and Babuska, R., 2012. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, *42*(6), pp.1291-1307.
[32] Kadam, R., Vidhani, V., Valecha, B., Bane, A. and Giri, N., 2021. Autonomous vehicle simulation using deep reinforcement learning. In *Machine Learning for Predictive Analysis* (pp. 541-550). Springer, Singapore.
[33] Sallab, A.E., Abdou, M., Perot, E. and Yogamani, S., 2017. Deep reinforcement learning framework for autonomous driving. *Electronic Imaging*, *2017*(19), pp.70-76.
[34] "Simulating Vehicles Using Autoware - Robotics Knowledgebase." https://roboticsknowledgebase.com/wiki/simulation/simulating-vehicle-using-autoware/
[35] "Home - AirSim." https://microsoft.github.io/AirSim/ .

[36] “torcs - 3D racing cars simulator game using OpenGL.” https://reposcope.com/package/torcs .
[37] “ImageNet.” http://www.image-net.org/ (accessed Apr. 23, 2021).
[38] “COCO - Common Objects in Context.” https://cocodataset.org/#home (accessed Apr. 23, 2021).
[39] Traffic Sign Detection dataset [Online] Available: https://kaggle.com/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign [Accessed: 25-Apr-2021]
[40] Traffic sign recognition using deep neural networks [Online] Available: https://towardsdatascience.com/traffic-sign-recognition-using-deep-neural-networks-6abdb51d8b70 [Accessed: 25-Apr-2021]

11. Appendix

11.1. Major Code Components

Autopilot.py

```
def record_data(img_id, image, control):
    image.save_to_disk(f'PythonAPI/ce903_team06/data/lane_detection/{img_id}')
    with open('PythonAPI/ce903_team06/data/lane_detection/_controls.csv', 'a', newline='') as f:
        csv.writer(f).writerow([img_id, control.steer, control.throttle, control.brake])

    return img_id + 1
```

```
frame = 0
while True:
    clock.tick_busy_loop(60)
    if controller.parse_events(client, world, clock):
        return

    if frame != 0 and (
            (frame % STRAIGHT_REFRESH_RATE == 0) or
            (abs(world.player.get_control().steer) > STEERING_THRESHOLD and frame % CURVE_REFRESH_RATE == 0)
    ):
        img_id = record_data(img_id,
                             CameraManager.last_img,
                             world.player.get_control())

    frame += 1
    world.tick(clock)
    world.render(display)
    pygame.display.flip()
```

Controller.py

```
def __init__(self, world, vehicle_id='vehicle.tesla.model3'):
    # Get vehicle blueprint
    blueprint_library = world.get_blueprint_library()
    vehicle_bp = blueprint_library.find(vehicle_id)

    # Pick a spawn point and spawn vehicle
    # spawn_point = random.choice(world.get_map().get_spawn_points())
    spawn_point = world.get_map().get_spawn_points()[3]
    self.vehicle = world.spawn_actor(vehicle_bp, spawn_point)

    # Set camera
    camera_bp = blueprint_library.find('sensor.camera.rgb')
    camera_bp.set_attribute('image_size_x', f'{IM_WIDTH}')
    camera_bp.set_attribute('image_size_y', f'{IM_HEIGHT}')

    # Get models
    self.lane_model = load_model('full_CNN_model.h5')
    self.control_model = load_model('car_control_v4')

    spawn_point = carla.Transform(carla.Location(x=2.5, z=1))
    self.sensor = world.spawn_actor(camera_bp, spawn_point, attach_to=self.vehicle)
    self.sensor.listen(lambda data: self.process_img(data))
```

```
def process_img(self, data):
    img = np.array(data.raw_data)
    img = np.reshape(img, (IM_HEIGHT, IM_WIDTH, 4))  # BGRA
    img = img[:, :, :3]  # BGR
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  # RGB

    # Prepare image for lane detection model
    img = cv2.resize(img, (160, 80))
    img = np.array(img)
    img = img[None, :, :, :]

    # Generate lane mask and concatenate to original
    lane = self.lane_model.predict(img)[0] * 255
    enhanced_img = np.concatenate((img[0], lane), axis=2)
    enhanced_img = enhanced_img[None, :, :, :]

    control = self.control_model.predict(enhanced_img)[0]
    control = [float(x) for x in control]
    control[2] = round(control[2])
    self.apply_control(control)
```

```
def apply_control(self, ctrl):
    self.vehicle.apply_control(carla.VehicleControl(steer=ctrl[0], throttle=ctrl[1], brake=ctrl[2]))
```

Main.py

```
import carla
from Controller import Controller


def handler(sign, frame):
    for actor in actor_list:
        actor.destroy()
    sys.exit(0)


actor_list = []
signal(SIGINT, handler)

# Instantiate client
client = carla.Client('127.0.0.1', 2000)
client.set_timeout(2)

world = client.get_world()
controller = Controller(world)
actor_list.append(controller)

while True:
    pass
```

11.2. Team Minutes

Date & time of meeting: January 28th, 2021 - 9:15 GMT	
Group members present: Juan, Atiq, Ebubekir, Kevin, Lisa, Mano, Monali, Suyash, Brijesh	Absent: Devesh
Name of chairperson for this meeting: Juan	Name of secretary for this meeting: Lisa
Review of all actions agreed at the previous meeting: - None	
Agenda Items: - Agile vs Traditional RE discussion. - Meeting with Professor Luca.	
Issues/problems to be reported to the project supervisor: N/A	

Date & time of meeting: January 28th, 2021 - 13:30 GMT	
Group members present: Juan, Atiq, Ebubekir, Kevin, Lisa, Mano, Monali, Suyash, Brijesh	Absent: Devesh
Name of chairperson for this meeting: Kevin	Name of secretary for this meeting: Lisa
Review of all actions agreed at the previous meeting: N/A	
Agenda Items: - Project introduction (autonomous self-driving car) **Next course of action** - Work on SRS draft - Research possible simulation environments for task	
Issues/problems to be reported to the project supervisor: N/A	

Date & time of meeting: January 30th, 2021 - 14:00 GMT	
Group members present: Juan, Atiq, Ebubekir, Kevin, Lisa, Mano, Monali, Suyash, Brijesh, Devesh	Absent: N/A
Name of chairperson for this meeting: Juan	Name of secretary for this meeting: Lisa
Review of all actions agreed at the previous meeting: - CARLA simulation was able to run in a machine - SRS document was created, and divisions were created based on project requirements	
Agenda Items: - Decide what to discuss in the next meeting with Luca - Divide the sections of the SRS document **Next course of action** - Start work on SRS document	
Issues/problems to be reported to the project supervisor: - Refer to SRS comments	

Date & time of meeting: February 4th, 2021 - 9:30 GMT	
Group members present: Juan, Atiq, Ebubekir, Kevin, Lisa, Mano, Monali, Suyash, Brijesh, Devesh	Absent: N/A
Name of chairperson for this meeting: Juan	Name of secretary for this meeting: Lisa
Review of all actions agreed at the previous meeting: - CARLA simulation was able to run in a machine - SRS document was created, and divisions were created based on project requirements	
Agenda Items: - Decide what to discuss in the next meeting with Luca - Gantt chart and split the project into implementation phase. - SRS review **Next course of action** - Discuss with Luca this afternoon and Finalize the SRS.	
Issues/problems to be reported to the project supervisor: - Refer to SRS comments	

Date & time of meeting: February 11th, 2021 - 13:00 GMT	
Group members present: Juan, Atiq, Ebubekir, Kevin, Lisa, ,Mano, Suyash, Brijesh, Devesh, Monali	Absent: N/A
Name of chairperson for this meeting: Kevin	Name of secretary for this meeting: Juan
Review of all actions agreed at the previous meeting: - Review changes to SRS document	
Agenda Items: - Discuss possibility of using virtual machines Next course of action - Start working on setting up the environment	
Issues/problems to be reported to the project supervisor: - Hardware limitations on machines to run Carla	

Date & time of meeting: February 13th, 2021 - 14:00 GMT	
Group members present: Juan, Atiq, Ebubekir, Kevin, Lisa, ,Mano, Suyash, Brijesh, Devesh, Monali	Absent: N/A
Name of chairperson for this meeting: Juan	Name of secretary for this meeting: Kevin
Review of all actions agreed at the previous meeting: - CARLA simulation was able to run in a machine - SRS document was created, and divisions were created based on project requirements	
Agenda Items: - SRS review - Requirements review **Next course of action** - Source of rewards (Decision/Design choice)	
Issues/problems to be reported to the project supervisor: - Refer to SRS comments - In the case Jira is not available, keep working on Gitlab - Instead of testing on another simulator, test on a different environment - It will be open to include different types of sensors such as lidar, etc.	

Date & time of meeting: February 16th, 2021 - 14:00 GMT	
Group members present: Juan, Atiq, Ebubekir, Kevin, Lisa, Devesh, Monali	Absent: Mano, Suyash, Brijesh
Name of chairperson for this meeting: Kevin	Name of secretary for this meeting: Lisa & Atiq
Review of all actions agreed at the previous meeting: - Review changes to SRS document	
Agenda Items: - Start working on setting up the environment - Carla Simulator set-up – How many people can run the simulator? - Task split on algorithm (fitting the algorithm into Carla) - Algorithms setup (Traffic lights, pedestrians, Pedestrian crossing, Lane changing) **Next course of action** - Researching algorithms that are available (On GitLab Board)	
Issues/problems to be reported to the project supervisor: - Hardware limitations on machines to run Carla	

Date & time of meeting: February 18th, 2021 - 14:00 GMT	
Group members present: Juan, Kevin, Atiq, Devesh, Ebubekir, Lisa, Mano, Monali, Suyash	Absent: Brijesh
Name of chairperson for this meeting: Monali	Name of secretary for this meeting: Juan
Review of all actions agreed at the previous meeting: - Review division of tasks	
Agenda Items: - Show code progress (Controller script) - Discuss potential approaches to model training **Next course of action** - Migrate issues and commits to new official repo	
Issues/problems to be reported to the project supervisor: N/A	

Date & time of meeting: February 25th, 2021 - 14:00 GMT	
Group members present: Juan, Kevin, Atiq, Devesh, Ebubekir, Lisa, Mano, Monali, Suyash	Absent: Brijesh
Name of chairperson for this meeting: Atiq	Name of secretary for this meeting: Juan
Review of all actions agreed at the previous meeting: - Show code progress (Controller script) - Discuss potential approaches to model training	
Agenda Items: - Progress on getting the virtual machine - Discuss possible detection models (Luca suggestion: https://github.com/bonlime/keras-deeplab-v3-plus, may be slow) **Next course of action** - Implement camera on controller - Try to incorporate models	
Issues/problems to be reported to the project supervisor: N/A	

Date & time of meeting: March 1st, 2021 - 14:00 GMT	
Group members present: Juan, Kevin, Atiq, Ebubekir, Lisa, Mano, Monali	Absent: Devesh, Suyash, Brijesh
Name of chairperson for this meeting: Atiq	Name of secretary for this meeting: Juan
Review of all actions agreed at the previous meeting: - Implement camera on controller - Try to incorporate models	
Agenda Items: - Update on VM (log-in credentials, users) - Show progress on lane detection **Next course of action** - progress on lane detection	
Issues/problems to be reported to the project supervisor: Accessing Carla simulator	

Date & time of meeting: March 6th, 2021 - 14:00 GMT	
Group members present: Juan, Kevin, Ebubekir, Lisa, Mano, Monali, Devesh	Absent: Suyash, Atiq, Brijesh
Name of chairperson for this meeting: Kevin	Name of secretary for this meeting: Juan
Review of all actions agreed at the previous meeting: - progress on lane detection - Accessing Carla simulator	
Agenda Items: - Demo how to run code for pending members - Discuss work on lane detection - Plot work for the next week **Next course of action** - Generate test data from autopilot - Build model to drive controller based on lanes - Start work by implementing object (Car) detection	
Issues/problems to be reported to the project supervisor: N/A	

Date & time of meeting: March 25th, 2021 - 14:00 GMT	
Group members present: Juan, Kevin, Ebubekir, Lisa, Mano, Monali, Devesh, Atiq, Brijesh, Suyash	Absent: N/A
Name of chairperson for this meeting: Kevin	Name of secretary for this meeting: Juan
Review of all actions agreed at the previous meeting: - Generate test data from autopilot - Build model to drive controller based on lanes - Start work by implementing object (Car) detection	
Agenda Items: - Update on virtual machines - Discuss progress on autonomous driving - Set meetings for Spring break **Next course of action** - Generate more data with new distribution for curves - Develop more models for autonomous driving - Keep researching more projects	
Issues/problems to be reported to the project supervisor: N/A	

Date & time of meeting: April 5th, 2021 - 15:00 GMT	
Group members present: Juan, Lisa, Monali, Devesh, Atiq, Suyash	Absent: Kevin, Mano, Ebubekir, Brijesh
Name of chairperson for this meeting: Atiq	Name of secretary for this meeting: Juan
Review of all actions agreed at the previous meeting: - Generate more data with new distribution for curves - Develop more models for autonomous driving - Keep researching more projects	
Agenda Items: - Progress on experimentation - Status of the new data set **Next course of action** - Finishing object detection - Training on new data set - Generate more data for different towns	
Issues/problems to be reported to the project supervisor: N/A	

Date & time of meeting: April 12th, 2021 - 15:00 GMT	
Group members present: Lisa, Juan, Kevin, Atiq, Mano, Monali, Ebubekir	Absent: Suyash, Devesh, Brijesh
Name of chairperson for this meeting: Juan	Name of secretary for this meeting: Kevin
Review of all actions agreed at the previous meeting: - Training on new data set - Generate more data for different towns	
Agenda Items: - Discuss driving models - Share kaggle dataset for experimentation - Consider shortcomings of the data set - Discuss issues with spawn_npc and poorly performing models **Next course of action** - Second layer for model that rounds values from predictor - Try to recollect data on a fixed refresh rate with shorter windows - Unblock object detection team	
Issues/problems to be reported to the project supervisor: - Issue with spawn_npc for VM in Lisa's account.	

Date & time of meeting: April 19th, 2021 - 15:00 GMT	
Group members present: Juan, Lisa, Atiq, Brijesh, Kevin, Mano, Monali, Ebubekir	Absent: Suyash, Devesh
Name of chairperson for this meeting: Juan	Name of secretary for this meeting: Kevin
Review of all actions agreed at the previous meeting: - Second layer for model that rounds values from predictor - Try to recollect data on a fixed refresh rate with shorter windows - Unblock object detection team	
Agenda Items: - Atiq's exploratory analysis of the data - Progress on models and new ideas - Final report - Questions for Luca's meeting **Next course of action** - Luca questions - Model isn't driving perfectly - tips/suggestions - Outline final report (get approval for format/sections) - Discuss what happens if model doesn't drive (report findings/lessons?) - Personal appraisal (format, what should be included?) - NVM - Can we use other datasets for our project? - Dates for the Assessment and presentation	
Issues/problems to be reported to the project supervisor: N/A	

Date & time of meeting: April 20th, 2021 - 14:00 GMT	
Group members present: Juan, Kevin, Devesh, Atiq, Brijesh, Lisa, Monali, Mano, Ebubekir, Suyash (internet problems, left)	Absent: N/A
Name of chairperson for this meeting: Juan	Name of secretary for this meeting: Lisa
Review of all actions agreed at the previous meeting: - Final report - Questions for Luca's meeting	
Agenda Items: - System performance for model - Individual presentation of work done and next steps - Final report overview, structure and expectation. - Presentation and demonstration dates – TBC - Personal appraisal structure - Open question to Luca on the performance of our algorithms **Next course of action** - Start first draft of final report	
Issues/problems to be reported to the project supervisor: N/A	

Date & time of meeting: April 26th, 2021 - 15:00 GMT	
Group members present: Kevin, Juan, Atiq, Lisa, Ebubekir, Mano, Monali	Absent: Devesh, Suyash, Brijesh
Name of chairperson for this meeting: Juan	Name of secretary for this meeting: Lisa
Review of all actions agreed at the previous meeting: - Final report overview, structure and expectation. - Personal appraisal structure and ranking.	
Agenda Items: - Close sprits and tasks opened in the Git/Jira - Review report **Next course of action** - Luca's meeting tomorrow. - Git code to be merged to the Master file.	
Issues/problems to be reported to the project supervisor: N/A	

Date & time of meeting: April 27th, 2021 - 14:00 GMT	
Group members present: Mano, Atiq, Suyash, Devesh, Kevin, Lisa, Ebubekir, Juan, Monali, Brijesh	Absent: N/A
Name of chairperson for this meeting: Juan	Name of secretary for this meeting: Lisa
Review of all actions agreed at the previous meeting: - Final report review and changes. - Git code to be merged to the Master file.	
Agenda Items: - Review the Final report with Luca. - **Git branch to include a read me - important.** - Presentation. Week 31 **Next course of action** - Submit zip files before the deadline. - Prepare presentation for **week 31 - 7th May @ 3 pm**	
Issues/problems to be reported to the project supervisor: N/A	