Deep Learning for Mobile Robots

Fraser Loneragan

22243455

Supervisor: Prof. Dr. Thomas Bräunl

GENG5511/GENG5512 Engineering Research Project

Submitted: 6th June 2022

Word Count: 7922

Faculty of Engineering and Mathematical Sciences
School of Engineering, Electrical, Electronic and Computer Engineering

Declaration

I, **Fraser Loneragan**, certify that:

This thesis is my own work, and I have appropriately referenced any sources used in preparing it.

This thesis does not contain any material which has been submitted under any degree in my name at any other tertiary institution.

In the future, no part of this thesis will be used in a submission in my name, for any other degree or diploma in any university or other tertiary institution without the prior approval of the University of Western Australia and where applicable, any partner institution responsible for the joint-award of this degree.

This thesis does not infringe on any copyright, trademark, patent or any other rights for any material belonging to another person.

Date: 5th Jun. 2022

Signed: [illegible]

Abstract

In 2022, fully autonomous, driverless cars are still unavailable to the public. One of the primary benefits of autonomous vehicles is their potential to prevent or reduce the number of vehicle accidents caused by human error. However, the development of driverless car software is complex due to the development costs, safety, laws, and regulations associated with a licensed car for road usage. Mobile robots developed and tested in a laboratory environment are not constrained in such ways and offer a low-cost development platform to develop and test driverless car software. The Artificial Intelligence (AI), Machine Learning (ML), and Deep Learning (DL) models used in ADAS systems are typically computationally expensive in terms of speed and memory requirements. Today, some SBCs, such as the Jetson Nano, come with GPU cores, which is significant because GPU cores can be used for fast training and inferencing of DL models. This is due to the increase in parallel computing that can be achieved by a GPU (i.e., they often consist of 100s or even 1000s of cores) compared with a CPU that typically consists of 4 or 8 cores. This project aims to expand upon deep learning and autonomous driving algorithms developed by past UWA students with a Jetson Nano instead of a Raspberry Pi. The new system retains the same EyeBot I/O board but uses the Jetson Nano for computation rather than a Raspberry Pi. The Jetson Nano can run more complex DL neural networks at higher frame rates than the Raspberry Pi due to the utilization of GPU cores on the Jetson Nano. The literature review conducted for this project identified a gap. To date, there are no publicly available Australian traffic sign datasets. Therefore, one of the goals of this project was to develop a dataset and a model to detect and classify Australian traffic signs. The DL models used in this project include an end-to-end lane keeping model using PilotNet, a traffic sign recognition model using MobileNets-SSD, and a Convolutional Neural Network (CNN) model to recognise speed limit signs. The project successfully demonstrated the robot could navigate the track, and recognise Australian traffic signs and speed limits at an acceptable speed of 19.2 frames per second (FPS). In addition, this project has provided a platform using the UWA developed EyeBot I/O board and RoBIOS libraries to allow future students to expand and improve the robot's autonomous capability using the Jetson Nano and an initial labelled dataset of Australian traffic signs.

Table of Contents

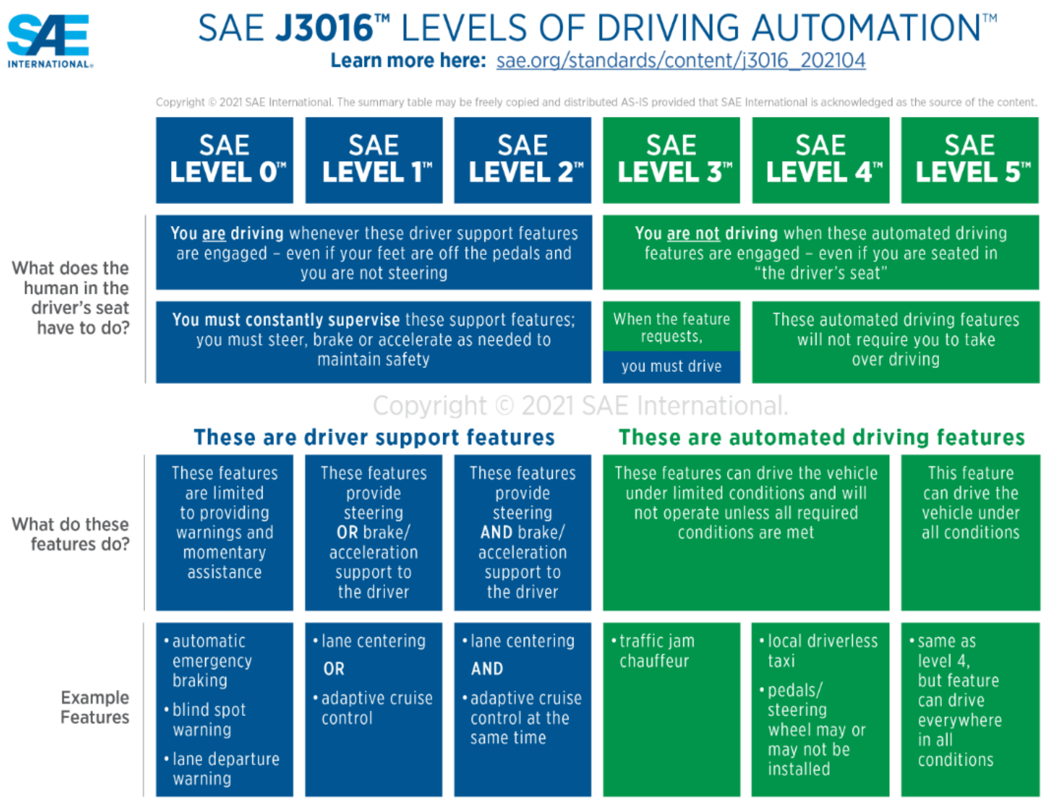
List of Figures

List of Tables

1 Introduction and Background

1.1 Introduction

Today, most car manufacturers include level 0 to level 2 Advanced Driver Assistance Systems (ADAS) in their vehicles. These levels are defined by the Society of Automotive Engineers (SAE) as the following [1]:



SAE J3016™ LEVELS OF DRIVING AUTOMATION™

Learn more here: sae.org/standards/content/j3016_202104

Copyright © 2021 SAE International. The summary table may be freely copied and distributed AS-IS provided that SAE International is acknowledged as the source of the content.

	SAE LEVEL 0™	SAE LEVEL 1™	SAE LEVEL 2™	SAE LEVEL 3™	SAE LEVEL 4™	SAE LEVEL 5™
What does the human in the driver's seat have to do?	You **are driving** whenever these driver support features are engaged – even if your feet are off the pedals and you are not steering			You **are not driving** when these automated driving features are engaged – even if you are seated in "the driver's seat"		
	You must constantly supervise these support features; you must steer, brake or accelerate as needed to maintain safety			When the feature requests, you must drive	These automated driving features will not require you to take over driving	
	These are driver support features			**These are automated driving features**		
What do these features do?	These features are limited to providing warnings and momentary assistance	These features provide steering **OR** brake/ acceleration support to the driver	These features provide steering **AND** brake/ acceleration support to the driver	These features can drive the vehicle under limited conditions and will not operate unless all required conditions are met		This feature can drive the vehicle under all conditions
Example Features	• automatic emergency braking • blind spot warning • lane departure warning	• lane centering **OR** • adaptive cruise control	• lane centering **AND** • adaptive cruise control at the same time	• traffic jam chauffeur	• local driverless taxi • pedals/ steering wheel may or may not be installed	• same as level 4, but feature can drive everywhere in all conditions

Copyright © 2021 SAE International.

Figure 1-1 Latest SAE levels of driving automation [1]

One of the primary benefits of autonomous vehicles is their potential to prevent or reduce the number of vehicle accidents caused by human error [2]. Cars with these features are reported to have 14% reduced property damage liability claims [2]. The opposing view is that humans may misuse, disuse, or abuse automation technology [3]. Unfortunately, Tesla's "Autopilot" feature has been misused numerous times and has caused fatalities on several occasions [4, 5].

Developing driverless car software is difficult due to the development costs, safety, laws, and regulations associated with a licensed car for road usage. However, mobile robots developed and tested in a laboratory environment are not constrained in such ways and offer a low-cost and fast development platform for driverless car software and systems. The University of Western Australia (UWA) has been performing work in this area for many years. This project is based on previous UWA work undertaken but uses an improved hardware platform and a newly developed Australian traffic sign detection model and dataset.

1.2 Background and History

The following section describes the background and history of the relevant development of autonomous robots at UWA.

1.2.1 Carolo Cup

The Carolo Cup is a mobile robot autonomous driving international competition organized by the Technical University of Braunschweig, Germany [7]. It is a student competition that features three challenges: parking, free drive, and dynamic obstacle avoidance.

Past UWA students have developed autonomous driving software and were able to adapt an EyeBot robot to do both lane keeping and traffic sign recognition based on the Carolo Cup rules [6, 8, 9]. Their lane keeping model used PilotNet, traffic sign recognition used colour segmentation for regions of interest (ROI) generation, and MobileNet for image classification [6]. However, the primary limitation in their project was the limited computing power of the Raspberry Pi 3B. For example, common object detection models such as SSD-MobileNet-V2 run at 1 FPS on the Raspberry Pi 3B compared to 39 FPS on the Jetson Nano (refer to Appendix 7.2). Therefore, their traffic sign recognition model was explicitly developed around this limitation, and the overall implementation with traffic sign recognition and lane keeping was able to run at a speed of 5 FPS [6].

The winning team from the Carolo-Master-Cup in 2021 was Team KITcar. They used a Jetson Nano (4GB) for computation, a camera combined with several distance sensing sensors, and no LIDAR for their entry [10]. The team's navigation stack is shown in Figure 1-2, and Dr. Drift car in Figure 1-3.

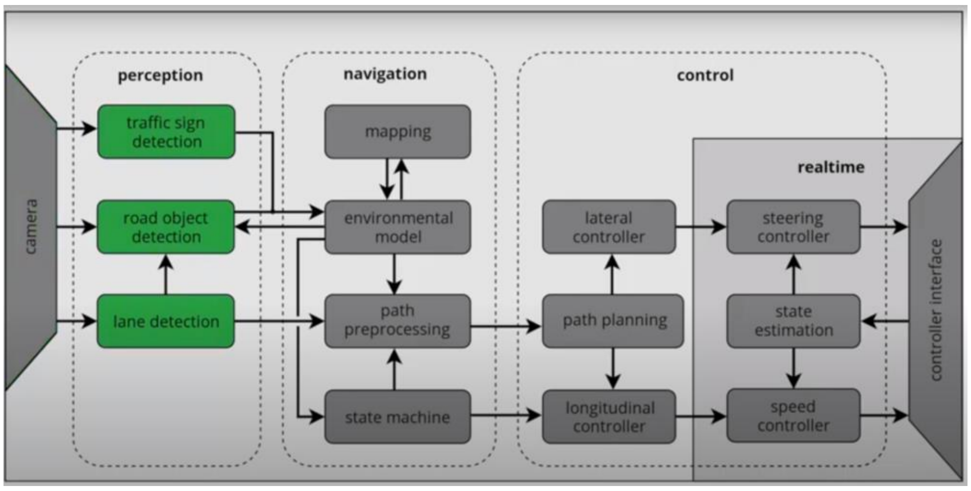


Figure 1-2 Team KITcar Navigation Stack [10]

Figure 1-3 Carolo Cup winning Dr. Drift, made by Team KITcar [11]

1.2.2 Track

The track available for testing was a scaled-down version of a Carolo Cup track, as shown in Figure 1-4. The track consists of an intersection, parking bays, and a pedestrian crossing. This track is used as the controlled environment in this project to test and develop the robot's autonomous driving software.

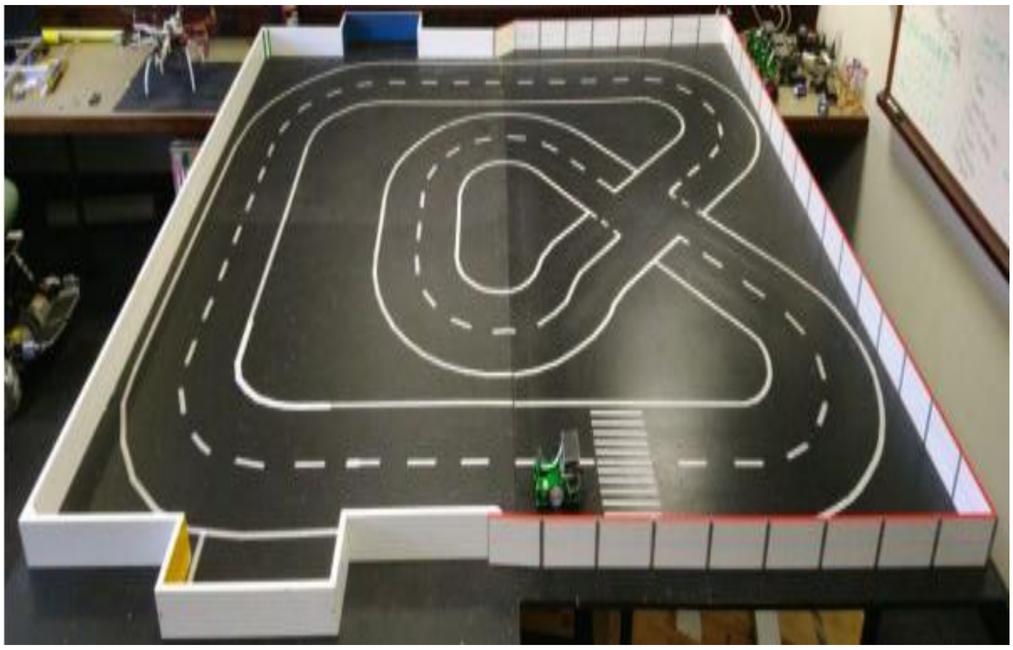


Figure 1-4 Scaled (1:2) Carolo cup track available to train and test on in the Robotics Laboratory [12]

1.2.3 EyeBot

Prof. Bräunl developed the EyeBot family of mobile robots with the latest revision, Eyebot 7, developed in 2017 [13, 14]. These robots typically consist of:

1. Two rear wheels and motors and a free to rotate front wheel to allow for steering.
2. Raspberry Pi Model 3B or Model 4B for the onboard computational processing.
3. 3 Position Sensitive Devices (PSDs) facing left, right, and in front.
4. Touchscreen LCD screen.
5. A single front-facing camera.
6. An I/O board, as shown in Figure 1-6 is loaded with the latest RoBIOS-7 (Robot Basic Input / Output System) OS, which provides commands to interface with connected PSD sensors, camera, and motors.

An EyeBot soccer robot is shown in Figure 1-5, and the Eyebot I/O board in Figure 1-6.

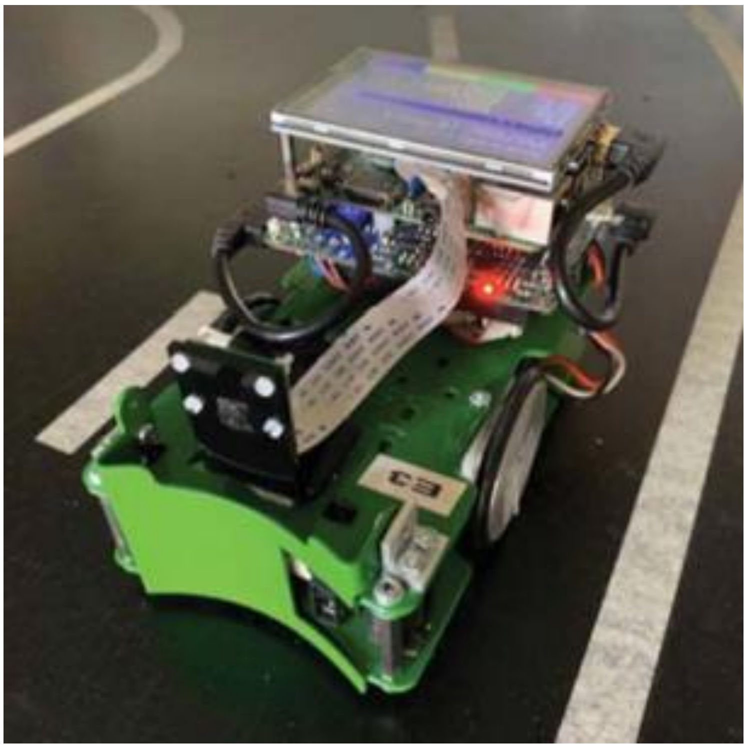


Figure 1-5 EyeBot SoccerBot [6]

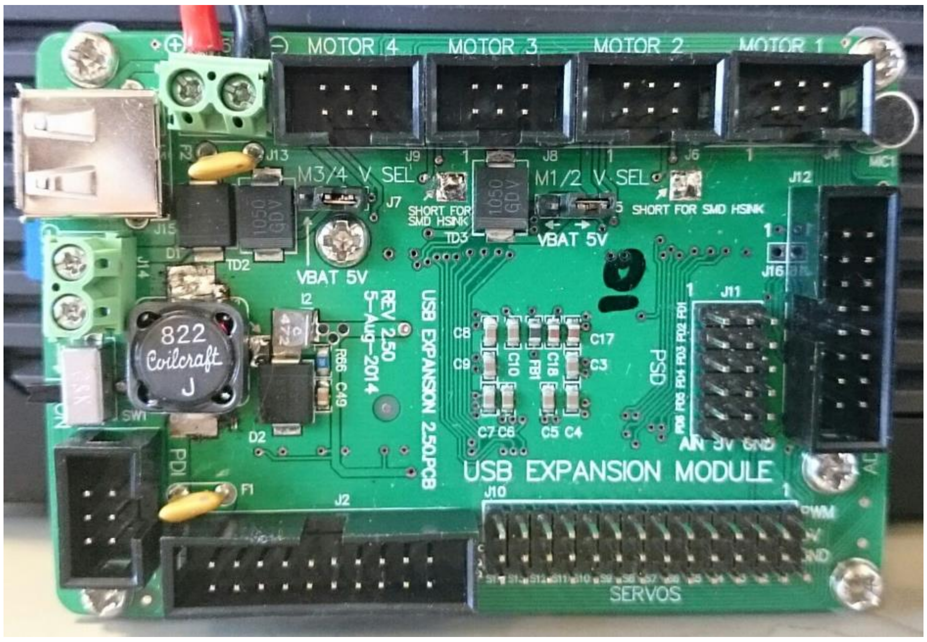
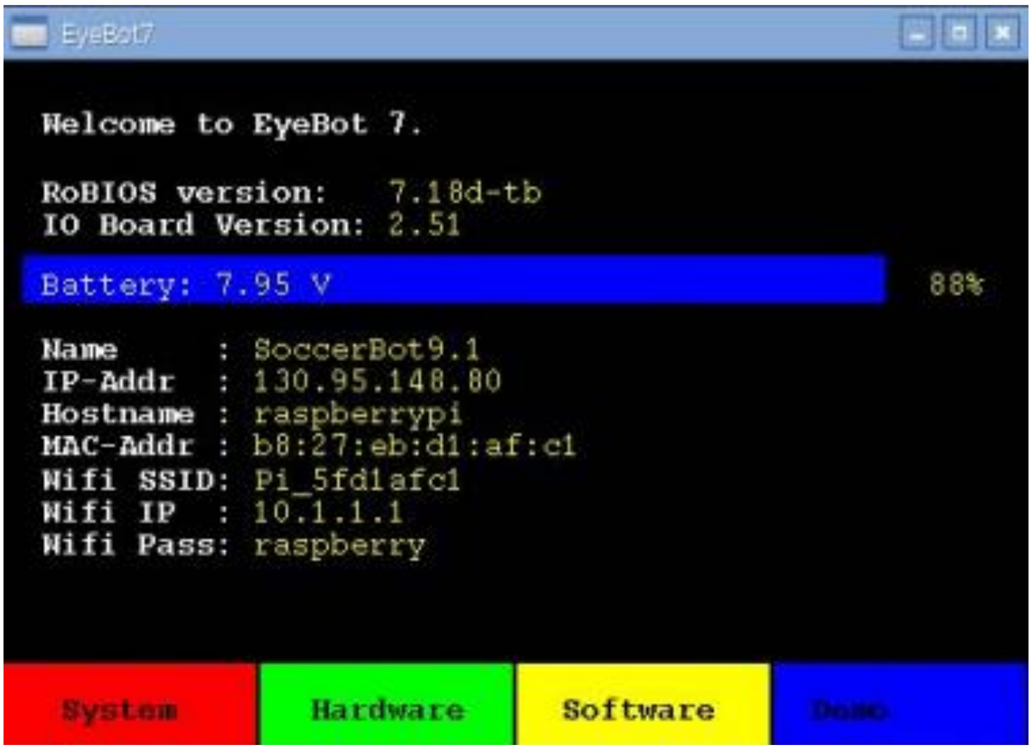


Figure 1-6 EyeBot I/O Board [14]

1.2.4 EyeBot User Interface and RoBIOS Library

The EyeBot user interface and Robot Basic Input Output System (RoBIOS) library were developed by Prof. Bräunl and past UWA students [14]. The Graphical User Interface

(GUI) and library commands provide an easy-to-use system for end-users to interact with sensors and motors connected to the I/O board. A screenshot of the GUI is shown in Figure 1-7 [14].


```
Welcome to EyeBot 7.

RoBIOS version:    7.18d-tb
IO Board Version:  2.51

Battery: 7.95 V                                  88%

Name      : SoccerBot9.1
IP-Addr   : 130.95.148.80
Hostname  : raspberrypi
MAC-Addr  : b8:27:eb:d1:af:c1
Wifi SSID:  Pi_5fd1afc1
Wifi IP   : 10.1.1.1
Wifi Pass:  raspberry
```

Figure 1-7 EyeBot User Interface [14]

1.2.5 EyeSim VR

The EyeBot family of robots has an associated simulator called EyeSimVR, based on the Unity 3D game engine and uses the same RoBIOS libraries with virtual Eyebot robots [15]. It was used for training and testing the PilotNet model as the simulator allowed different tracks to be imported, including Carolo Cup tracks and access to the camera feed of the virtual robot as it navigates the track in the simulator as shown in Figure 1-8. Furthermore, although not used in this project, it is possible to create 3D traffic sign objects in Blender and import these into the simulator. This could potentially be used to train a Traffic Sign Recognition (TSR) model from images collected in the simulator.

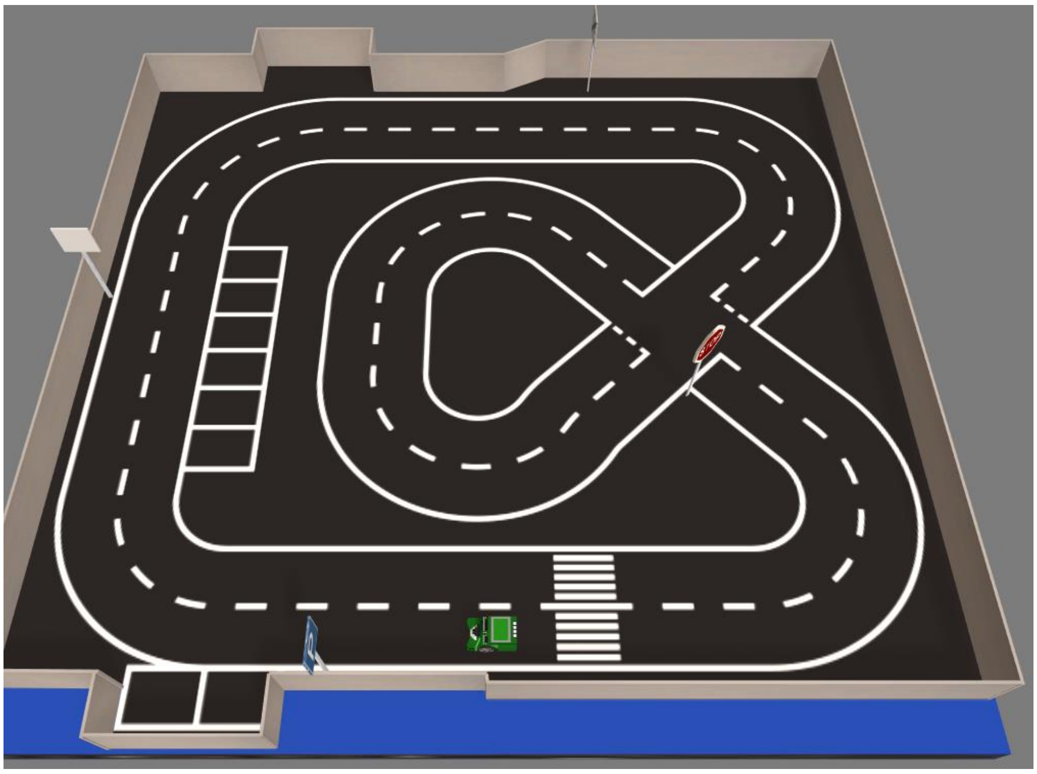


Figure 1-8 EyeSim VR Carolo Cup track and traffic signs

2 Literature Review

2.1.1 Current Levels of Driving Automation

Currently, Level 5 (full automation) has yet to be achieved. The highest level of driving automation achieved to date is level 4 by the Waymo Chrysler Pacifica [16]. This system uses LIDAR primarily for environment detection and computer vision as a backup. The reason why it is not considered Level 5 is due to working only in limited conditions. One of these conditions is only being able to operate in Phoenix, Arizona, although there are plans for deployment in San Francisco, California [17]. The highest SAE level achieved so far for a mass-production car sold to the public is level 3. This was for the Honda Legend Sedan using the Traffic Jam Pilot system [18]. Level 3 systems assume the authority for all manoeuvres in the determined scenario unless, through self-assessment, the system cannot continue and will alert the driver to take control of the vehicle [19]. Generally, most production cars today come with levels 0, 1, or 2 ADAS. The reason cars do not come with level 3 or higher ADAS is due to the cost of the sensors (mainly LIDAR), development costs associated with driverless car software and safety concerns. Examples of level 1 ADAS include adaptive cruise control, emergency brake assist, lane-keeping, and lane-centering. Examples of level 2 ADAS include highway assist, autonomous obstacle avoidance, and autonomous parking [19]. Tesla's Autopilot feature, similar to highway assist, is considered Level 2 automation [16].

2.2 LIDAR vs Computer Vision

LIDAR and Computer Vision are two techniques that autonomous cars can use to gather information about the road and the surrounding environment to make driving decisions. A LIDAR sensor pulses light to the environment and records what is reflected, mapping the surrounding environment [20]. Computer Vision uses the video feed from cameras connected to the car to gather information of its surroundings. Both systems are viable and not mutually exclusive; for example, the Waymo driverless car that achieves level 4 SAE autonomy uses both LIDAR and computer vision [20]. However, LIDAR sensors are significantly more expensive than cameras, and successful autonomous robots have been developed using camera systems only (e.g., Team KitCar's Carolo Cup winning entry [15]). In addition, lane keeping, TSR, and speed limit recognition are typically done using a camera sensor. For these reasons, the project used a camera sensor only.

2.3 Artificial Intelligence, Machine Learning, Deep Learning and Deep Neural Networks

Abbass defines artificial intelligence as the automation of the decision-making process [8]. This is important in Autonomous driving as the vehicles need to make their own decisions based on sensory input. Both Machine Learning (ML) and Deep Learning (DL) are used in Artificial Intelligence systems. Machine learning consists of techniques and algorithms which enable computers to learn from data. DL utilizes multi-layered neural networks to make classifications from data [21]. Neural Networks combine multiple nonlinear processing layers, using simple elements operating in parallel and are based on the biological nervous system [22]. They consist of many layers: including an input layer, multiple hidden layers (i.e., giving rise to the name Deep Neural Network (DNN)), and an output layer, as shown in Figure 2-1 [22]. Learning occurs through the adjustment of weights whereby errors in the output layer are back-propagated through each hidden layer to the input layer, amending the weights in each layer [23].

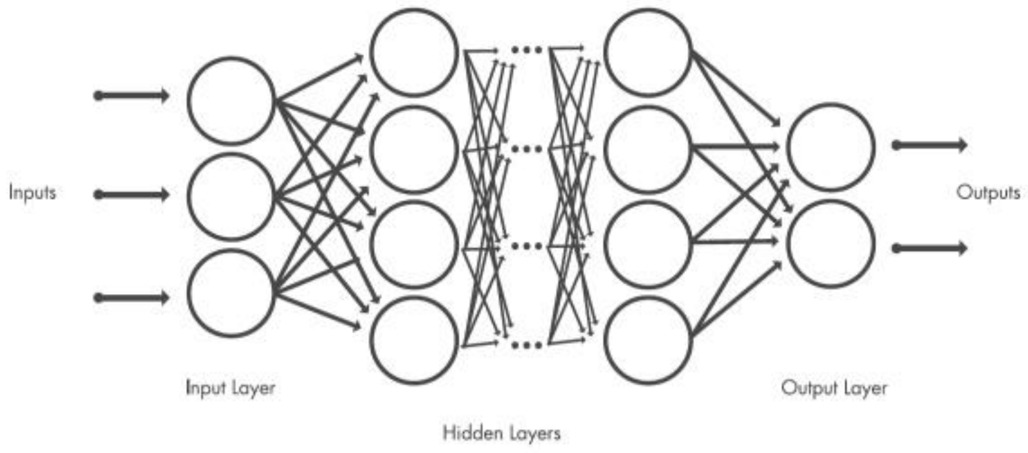


Figure 2-1 Deep Learning model diagram [22]

2.4 Convolutional Neural Networks

Convolutional Neural Networks are a type of artificial neural network which is highly effective for computer vision tasks. They feature several types of hidden layers, most notably convolutional layers, which apply numerous filters to convolve the entire image to generate feature maps. The significant advantages of the convolution filter are reducing the number of parameters and identifying correlation between neighbouring pixels, which is vital for the learning process [24].

2.4.1 TensorFlow, Keras and TensorFlow Lite

TensorFlow, Keras, and TensorFlow Lite are tools used to train and inference Deep Learning and Machine Learning models. TensorFlow is an open-source deep learning framework created by Google Brain and supports both multi-CPU and GPU-based executions [25]. Keras is a high-level Application Programming Interface (API) that interfaces with TensorFlow and makes building deep learning models easier and with less code [25]. TensorFlow Lite is a set of tools included in TensorFlow, enabling developers to run machine learning models on mobile, embedded, and edge devices [26]. It has several advantages over TensorFlow’s protocol buffer model format, such as reduced size (i.e., small code footprint) and fast inference (i.e., data is directly accessed without an extra parsing and unpacking step). Therefore, enabling TensorFlow Lite to be used efficiently on devices with limited computational and memory resources [26].

2.4.2 NVIDIA PilotNet

NVIDIA is a corporation primarily known for its development and production of high-end consumer and professional graphics cards. They are also heavily involved in both Deep Learning and AI, often using hardware developed-in-house for training and inferencing. They have created their own end-to-end driving neural network architecture called PilotNet. A schematic of the architecture is shown in Figure 2-2. This software is a Convolution Neural Network that learns to control a vehicle by using pixels from images taken with a front-facing camera and mapping them directly to steering commands [27]. An example of the saliency map generated by NVIDIA is shown in Figure 2-3.

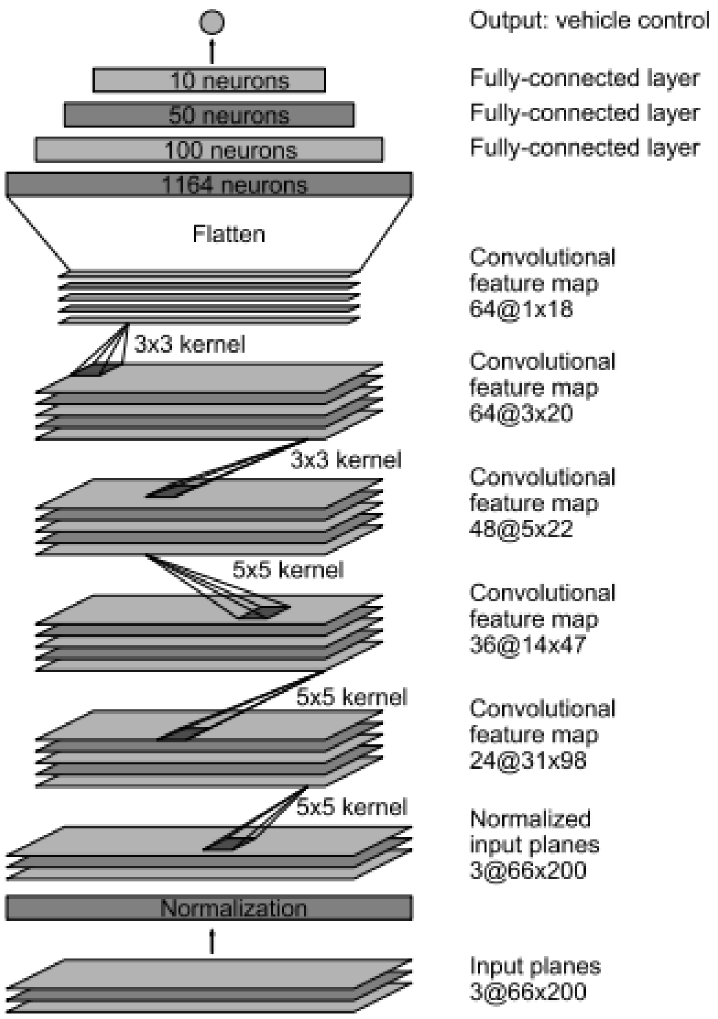


Figure 2-2 NVIDIA PilotNet CNN architecture [27]

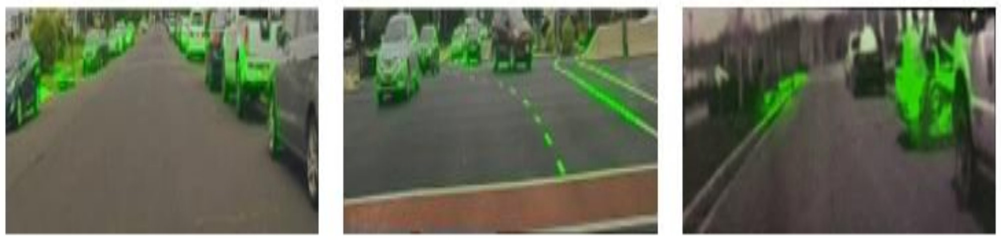


Figure 2-3 PilotNet saliency example [27]

2.5 Object Detection

There are two main methods used for detecting objects. The first uses image classification as an input and will output a prediction for the object base on the classified image. This method is used when the position and size of the object are not relevant and when there can only be one object in each image. The second method directly detects and predicts the object. It also determines the object’s location with a bounding box [28]. Object detection models can also detect multiple objects in one image. Examples of fast object detection DL models include Regions with Convolutional Neural Networks (R-CNN), Fast R-CNN, You Only Look Once (YOLO), and Single Shot Detectors (SSD). R-CNN and Fast R-CNN firstly identify objects that are expected to be found and then detect and classify objects only in those regions using a CNN. At the expense of accuracy for efficiency, SSD and YOLO eliminate proposal generation and the subsequent pixel or feature resampling stages and encapsulate all computation in a single step [29]. Speed is important for TSR as there is a small window when the traffic sign is in view of the camera when the vehicle is in motion. Therefore, TSR must run at an acceptable real-time speed that can detect traffic signs even at high speeds such as 100 km/hr. Thus, the speed of an object detection model is often just as vital as its accuracy in the context of TSR.

2.5.1 SSD, MobileNet and MobileNet-SSD

MobileNet is a class of efficient convolutional neural networks used for mobile vision applications such as image classification [30]. They are based on depth-wise separable convolutions and are specifically designed to execute on embedded systems [30]. Single Shot Detectors (SSDs) consist of a trained image classification network as a backbone model and an SSD head, as shown in Figure 2-4. NVIDIA has trained a MobileNet-SSD object detection model using PyTorch that combines the SSD-300 (300x300 image input) Single-Shot MultiBox Detector with a MobileNet backbone, as shown in Figure 2-5. It is a popular network architecture for real-time object detection on mobile and embedded devices and was used by NVIDIA to successfully classify 19 objects in the Common Objects In Context (COCO) dataset [31].

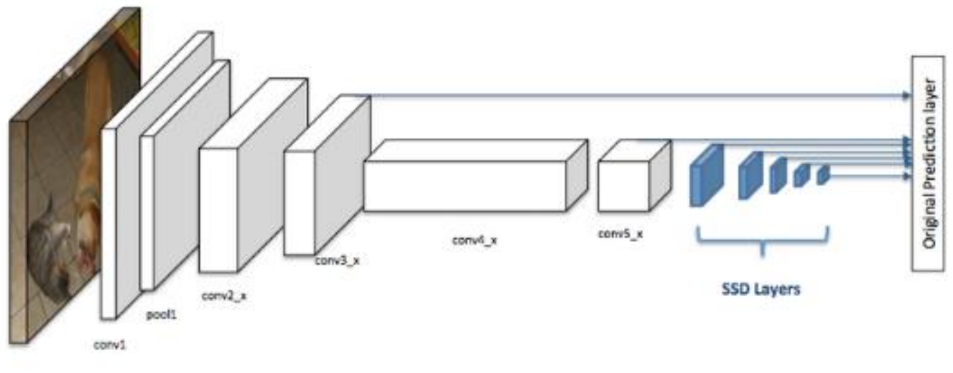


Figure 2-4 Typical SSD model with a backbone and head [29]

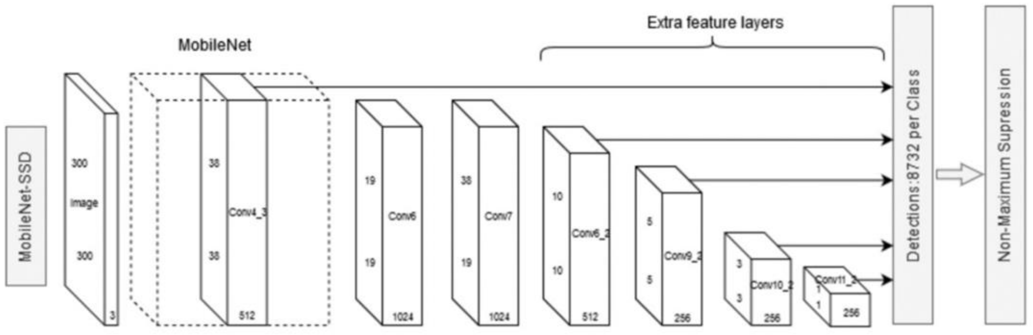


Figure 2-5 NVIDIA MobileNet-SSD model [31]

2.6 PyTorch, Open Neural Network Exchange and NVIDIA TensorRT

PyTorch is an open-source machine learning framework that helps developers create machine learning models [32]. It is like TensorFlow and reduces the amount of code required to write and develop Machine Learning models. Open Neural Network Exchange (ONNX) provides an open-source format for AI, deep learning, and traditional machine learning models [33]. Finally, TensorRT is a software development kit built on the CUDA

GPU programming language allowing high-performance deep learning inferencing [34]. The TensorRT processing flowchart is shown in Figure 2-6.

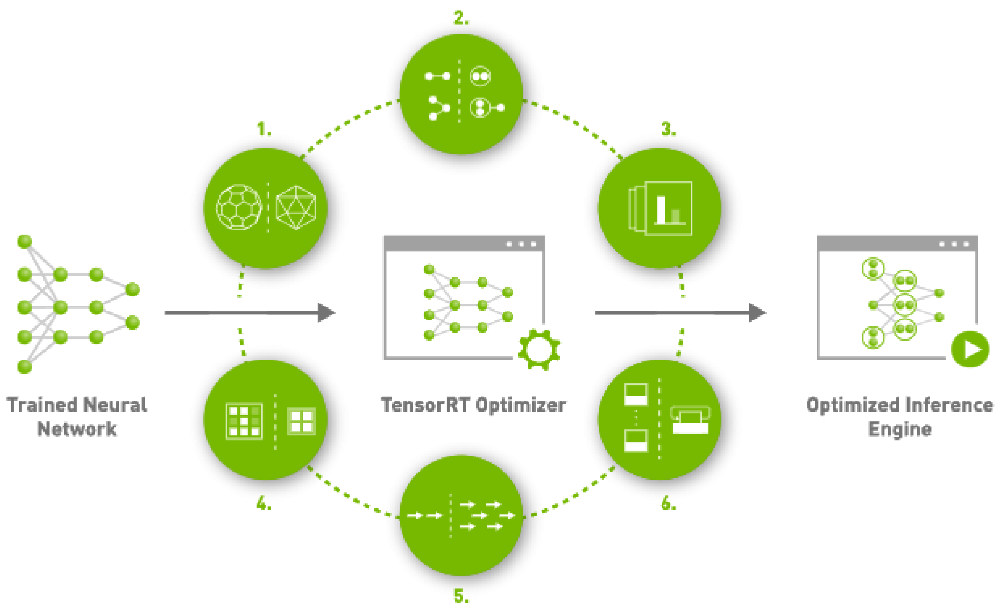


Figure 2-6 TensorRT flowchart [34]

2.7 Transfer Learning

Transfer learning uses existing DNNs trained for one application to be used for another application. This is done by taking the domains, tasks, and distributions used in training and testing for one application to be modified and trained on a different application [35]. In transfer learning, a typical task is to uncover the common features that can benefit each task. There are three types of transfer learning [12]:

1. Inductive Transfer Learning aims to learn a new task in the same domain.
2. Transductive Transfer Learning: learning aims to learn the same task in a new domain.
3. Unsupervised Transfer Learning, where both domains and tasks are different.

This type of learning is beneficial when data in the target domain is limited, which is the case for this project for TSR.

2.8 Domain Randomization:

Gathering data for DL models in a simulator is faster, more scalable, and lower cost than using a physical robot. Unfortunately, discrepancies between simulators and the real world make transferring learnt behaviours from simulators to the real world challenging. Some of these discrepancies between the real world and simulators are the physics and the appearance. Domain randomisation is a way to increase the generalisation of models trained in a simulator. This is done by randomising environment variables and appearance of the simulator domain, which can ensure models generalize through the learning process by making variability high, hence improving performance in the real world [36].

2.9 Traffic Sign Detection and Datasets

2.9.1 Traffic Sign Datasets

There is a considerable amount of data on traffic signs throughout the world. This includes the German Traffic Sign Recognition benchmark dataset [37], German traffic sign detection benchmark [38], Russian Traffic Sign Images dataset [39], Swedish traffic signs dataset [40], Belgian Traffic Sign dataset [41], American traffic sign dataset [42] and the Mapillary (European) traffic sign dataset [43]. Examples of the German traffic sign benchmark are shown in Figure 2-7 and the Mapillary traffic sign dataset in Figure 2-8. However, there is a gap in this area in Australia where there are no Australian traffic sign datasets freely available to the public. Researchers at QUT have made an Australian traffic sign dataset but have not released the dataset publicly for privacy concerns [44].

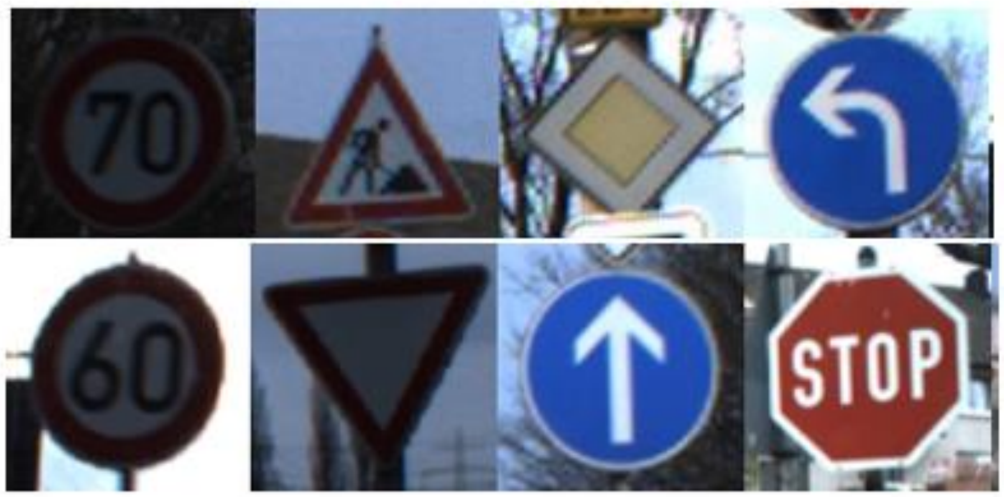


Figure 2-7 German Traffic Sign Recognition Benchmark data examples [45]

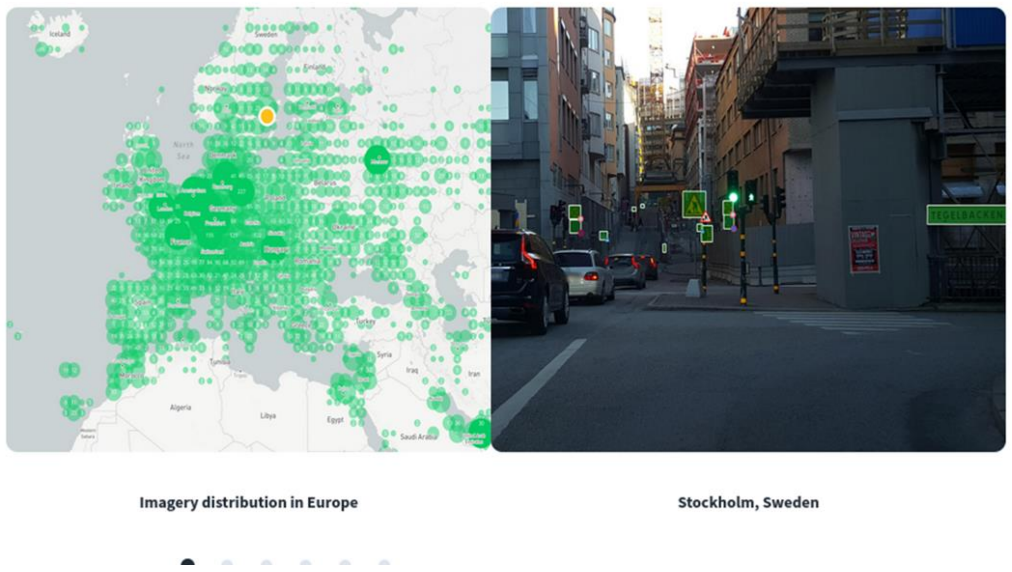


Figure 2-8 Mapillary traffic sign dataset [43]

2.9.2 Traffic Sign Recognition (TSR)

NVIDIA has developed several SignNet models (i.e., traffic sign DNNs) to detect and classify traffic signs and traffic lights. They have collected data in the United States, European Union, and Japan. Data collection used eight cameras spread around the car, but model inference can be done via one front-facing camera, as shown in Figure 2-9. Some of the limitations that they note are [46]:

1. Data was collected mainly during daylight, overcast, twilight, and non-rain conditions. Therefore, performance at night-time and in the rain is likely to suffer.
2. Classification performance depends on the size of the traffic sign in the image frame. NVIDIA recommend traffic signs be 20 pixels or more in size to be accurately predicted.
3. SignNet will not perform well in regions where data was not collected, such as Australia.

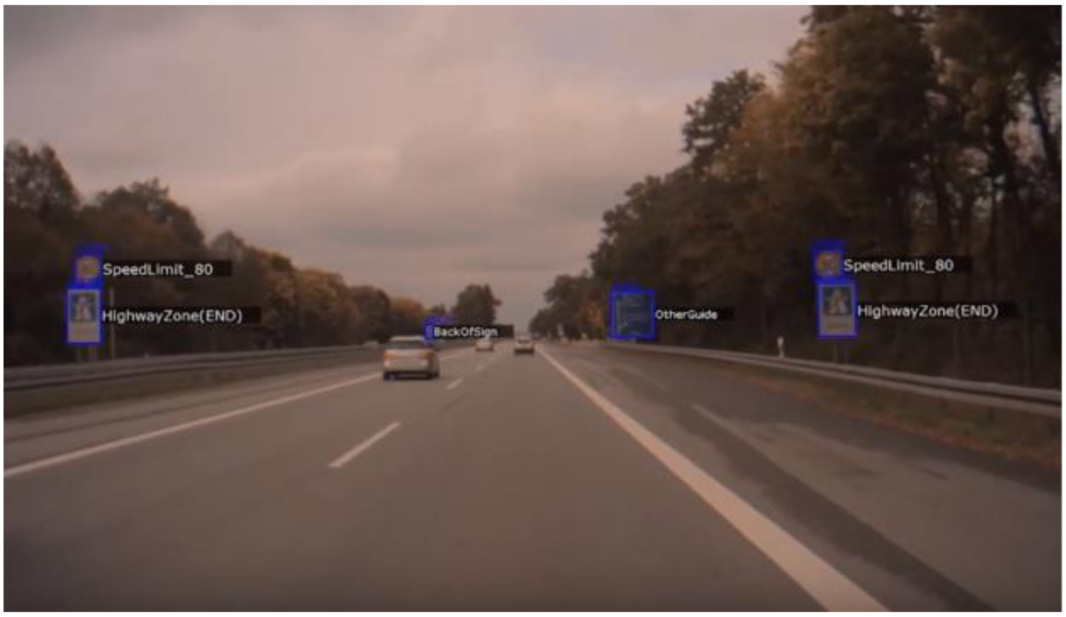
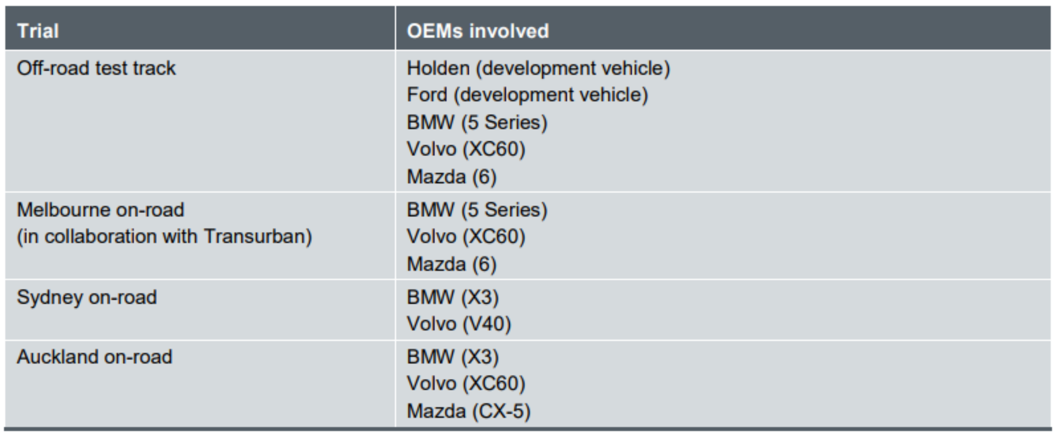


Figure 2-9 NVIDIA SignNet traffic sign detection example [47]

In addition to NVIDIA, researchers at the Auckland University of Technology have made their own TSR model with their own New Zealand traffic sign dataset using SSD and YOLO object detection models [48]. These models rely on labelled datasets to classify traffic signs. The literature review did not find any datasets for Australian traffic signs; therefore, a key requirement for this project was to develop a dataset and method for classifying Australian traffic signs. Despite the lack of a dataset, some car manufacturers include TSR systems in their vehicles sold in Australia. Austroads have tested the TSR systems of several cars operating in Australia and New Zealand, as shown in Figure 2-10. Key findings from their report include [49]:

1. Static speed limit detection was nearly 100% accurate.
2. Speed limit detection would detect 40 km/hr school zones as a standard 40 km/hr speed limit but does not recognise the school zone hours to which the speed limit applies.
3. The TSR systems tested cannot recognise traffic signs other than speed limit signs.
4. The TSR systems tested have difficulties recognising electronic speed limits.



Trial	OEMs involved
Off-road test track	Holden (development vehicle) Ford (development vehicle) BMW (5 Series) Volvo (XC60) Mazda (6)
Melbourne on-road (in collaboration with Transurban)	BMW (5 Series) Volvo (XC60) Mazda (6)
Sydney on-road	BMW (X3) Volvo (V40)
Auckland on-road	BMW (X3) Volvo (XC60) Mazda (CX-5)

Figure 2-10 Vehicles with TSR tested by Austroad [49]

An explanation for TSR being able to recognise speed limit signs in Australia nearly 100% accurately with no available dataset is likely due to the transferability of speed limit datasets from other regions in the world, as speed limits across the globe look similar. Interestingly, the report notes that the TSR systems tested in Australia can only detect and classify speed limit signs. This is no longer valid as Mazda, for example, advertise that their TSR can recognize 'STOP', speed limit, and 'NO ENTRY' signs on their Australian website [50]. Finally, the report notes TSR systems have difficulties in detecting and classifying electronic speed limits. This is a problem as electronic speed limits are becoming more prevalent in Australia, especially on highways and locations where temporary roadwork is occurring. Detecting and classifying electronic speed limit signs was not addressed in this project but could be a key area for future research projects.

3 Design Process

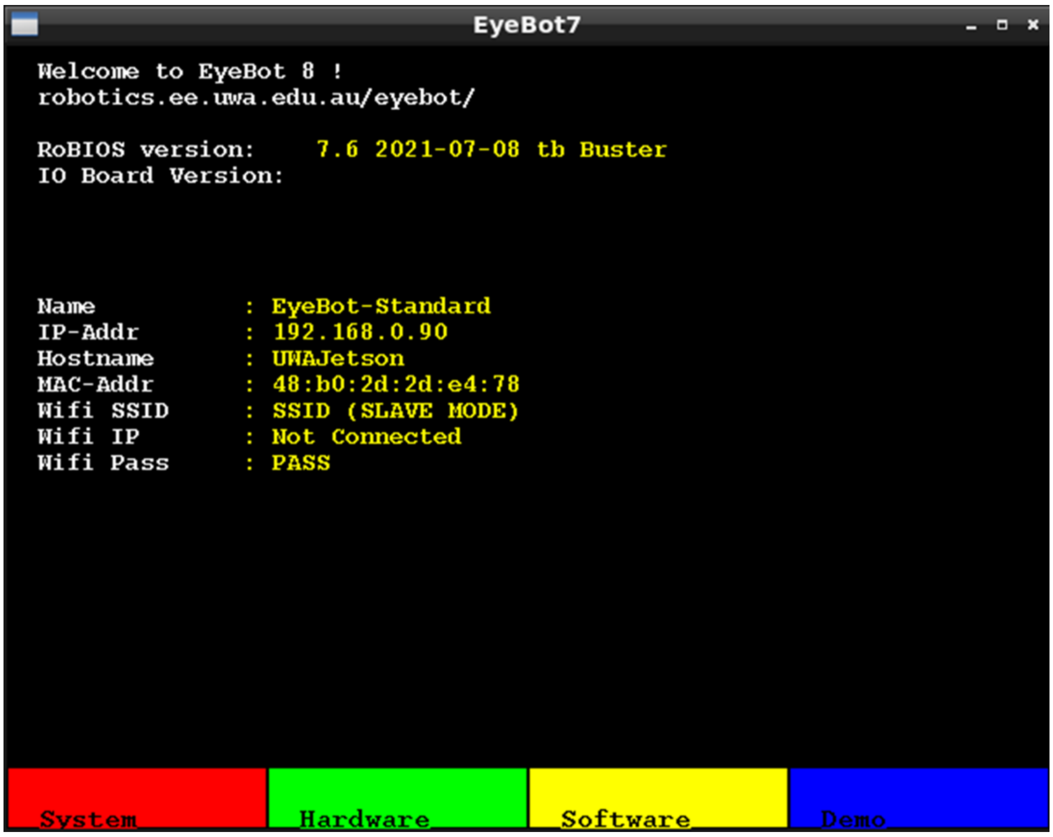
3.1 Raspberry Pi vs Jetson Nano

The three relevant considerations that led to the choice of Single Board Computer (SBC) for this project in order of priority were fast inferencing, price, and memory. The Jetson Nano includes 128 Maxwell GPU cores, compared with the Raspberry Pi 4B, which only has a CPU and no GPU cores. The GPU cores can be utilized for fast inferencing of complex DNNs. The Jetson Nano and the Raspberry Pi have similar performing CPUs and an equivalent cost of around $80 [51, 52]. The last consideration was the memory, and the Jetson Nano comes with 2GB of ram compared to the 4GB of ram on the Raspberry Pi 4B.

Ultimately the decision was made to use the Jetson Nano over the Raspberry Pi due to the GPU cores. Furthermore, the prices were similar, and the memory limitations could be addressed using swap space if necessary. The Jetson Nano 4GB was not considered because the cost at the time was too high at ~$200, although it has been reduced in price to $140 [53].

3.2 Porting EyeBot User Interface and RoBIOS Libraries

A key concern was the compatibility of the EyeBot user interface, RoBIOS libraries, and the Jetson Nano. Porting this software to the Jetson Nano involved ensuring the correct packages were installed such as x11 and changing all file paths to reflect the different Linux home directories. Porting was successful and is shown in Figure 3-1.



```
Welcome to EyeBot 8 !
robotics.ee.uwa.edu.au/eyebot/

RoBIOS version:    7.6 2021-07-08 tb Buster
IO Board Version:



Name           : EyeBot-Standard
IP-Addr        : 192.168.0.90
Hostname       : UWAJetson
MAC-Addr       : 48:b0:2d:2d:e4:78
Wifi SSID      : SSID (SLAVE MODE)
Wifi IP        : Not Connected
Wifi Pass      : PASS
```

Figure 3-1 EyeBot User Interface and RoBIOS library on Jetson Nano

3.3 Hardware Constraints - Jetson Nano 2 GB Developer Kit

3.3.1 Power Requirement – Jetson Nano

The Raspberry Pi 3B and 4B and the Jetson Nano 2GB Developer kit require 5V and 2.5A for power. Therefore, the regulated voltage output from the EyeBot I/O board (i.e., when powered by a 7.2V battery), which power the Raspberry Pi's should be good enough therefore to also power the Nano. However, the Nano needs more current when peripherals like the USB camera are connected. It was, therefore, required to add a mobile power bank to suitably power the Jetson Nano. The chosen power bank was a 10000 mAh power bank from Altronics that can supply 5V and a maximum of 3.6A is shown in Figure 3-2. At a nominal load of 2.5A, the battery life is approximately 4 hours which was reasonable for testing purposes.

Figure 3-2 Mobile power bank used (5V 3.6A 10000mAh) [54]

3.3.2 Camera Limitations

There were some limitations due to using a single mono camera, mainly the dead zone (area under the camera in which the ground is out of view) and the lack of ability to determine the distance objects are from the robot. The different fields of view (FOV) of the Logitech C920 camera (refer [55]) used in this project are:

- Diagonal FOV = 78°
- Horizontal FOV = 70.42 °

- Vertical FOV = 43.30 °

The vertical FOV resulted in the robot having a dead zone of approximately 16 cm as tested using a white piece of paper as shown in Figure 3-3.

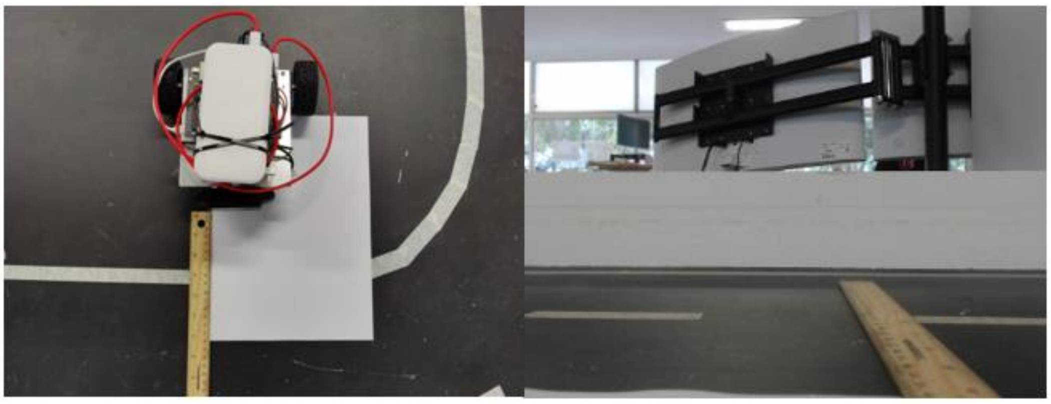


Figure 3-3 16 cm camera dead zone based on white paper test

Too much dead zone can be of concern in situations where the turn is too tight and the road lines move out of view of the camera. The dead zone can be improved by using a multicamera setup where one camera is pointed to the ground for Lane Keeping and another is pointed towards the horizon for TSR, or by using a camera with a greater vertical FOV. Additionally, to determine the distance objects are from the camera a stereo camera can be used instead of a mono camera or by pairing a mono camera with a LIDAR sensor. The Logitech C920 camera was used due to its availability, and easy installation in the space below the robot.

3.4 Lane Keeping

Lane keeping using PilotNet has been implemented and tested by various past UWA students [6, 9, 55]. This project implements a PilotNet model that uses red, green, and blue (RGB) images as inputs and will output steering values in degrees/sec and is trained using images from the EyeSim simulator. Data collection in the simulator was done manually by driving the robot around the track using an Xbox controller. Every steering input using the Xbox controller's L1 and R1 buttons (refer Figure 3-4) saves the current camera image and the steering angle. Four tracks were used during training, shown in Figure 3-5, and 12 laps of each track were driven (i.e., six laps in each direction in the left lane). The model can then be trained to predict steering in degrees/sec from images captured on the test track, as shown in Figure 3-6.

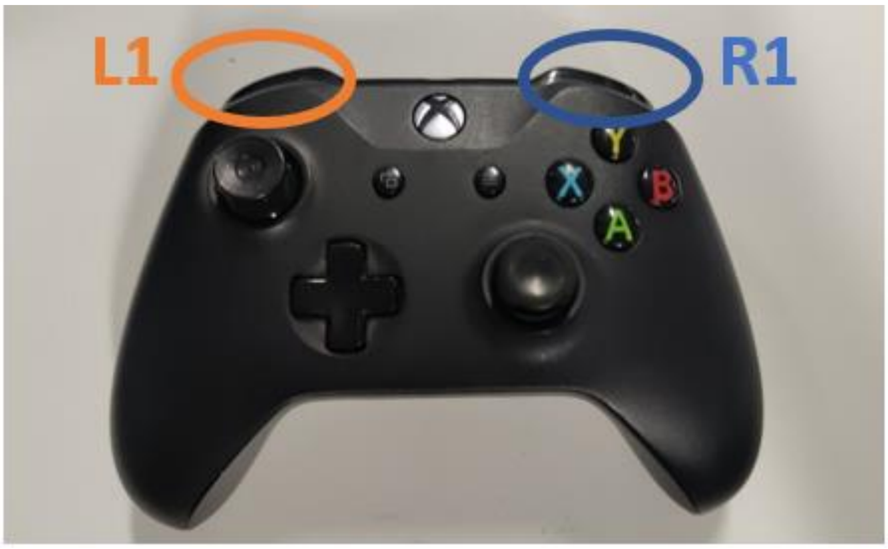


Figure 3-4 Xbox controller, with L1 and R1 buttons labelled

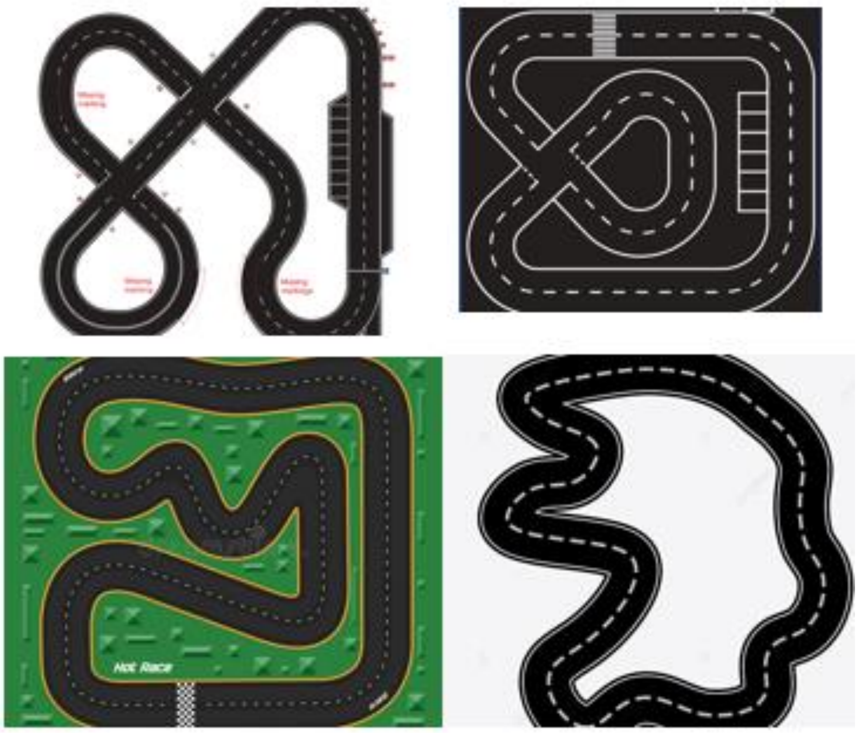


Figure 3-5 Tracks in EyeSim used for training

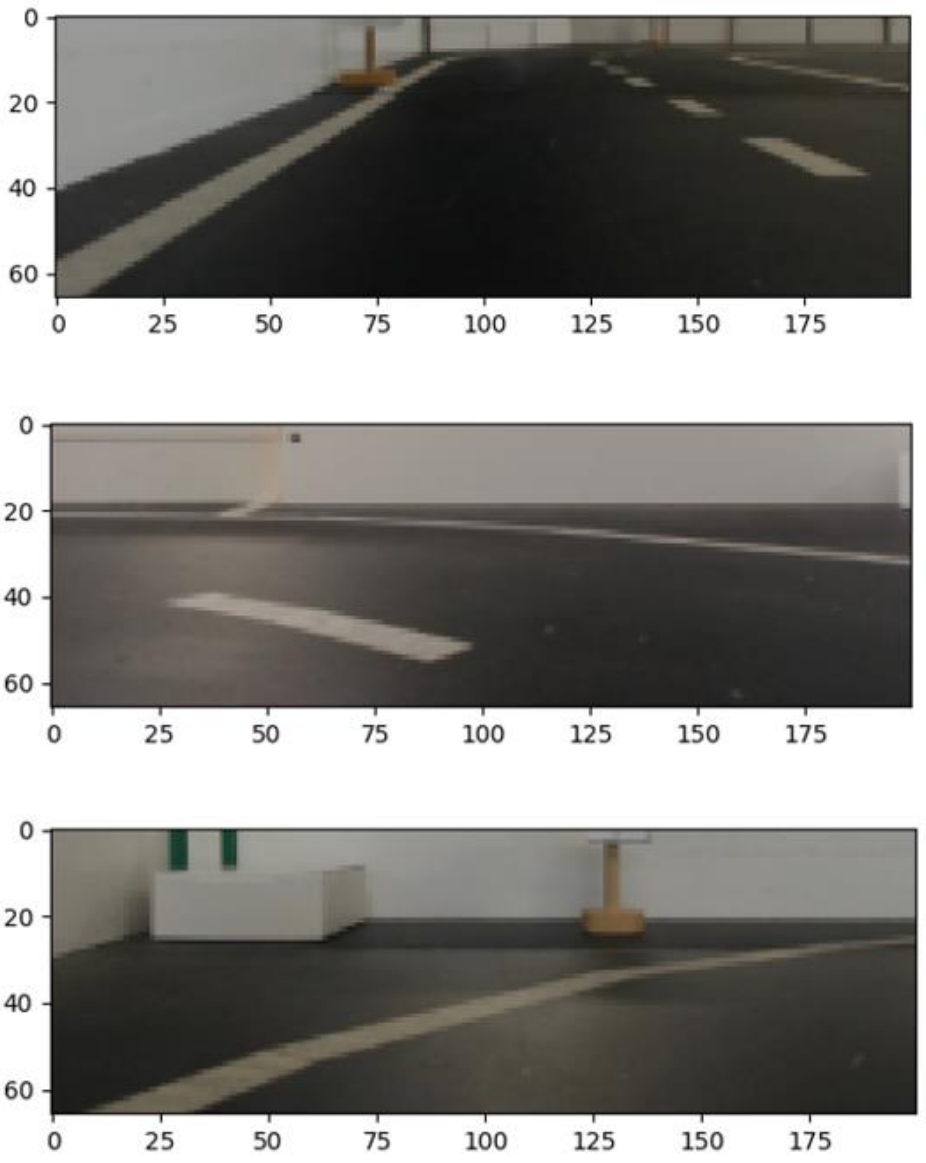


Figure 3-6 200x66 example images: straight, left, and right in which the predicted steering angles were -3, 13 and -12 respectively (degrees/sec)

3.4.1 Domain Randomization and Data Augmentation

Domain randomization and data augmentation techniques have been used to improve the performance of the PilotNet model trained using the simulator to the real test track, similar to past research work [12]. The domain randomisation techniques include randomising the colour of squares, size of squares, orientation of squares, and brightness of road marking lines on the track. This technique was used to create three variations of each track, as shown in Figure 3-7. The data augmentation techniques include flipping the image to double the dataset and adjusting blur, brightness, and contrast to increase the images' variability, as shown in Figure 3-8 and Table 3-1 [12].

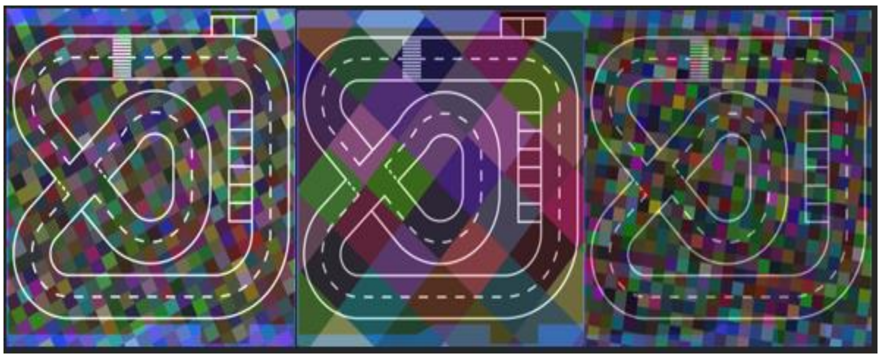


Figure 3-7 Randomisation of the colour of squares, number of squares, and orientation of squares, as well as the brightness of the road markings

Table 3-1 Data Augmentation parameters.

Parameter	**Value**
Brightness	[-0.2, 0.2]
Blur	Constant bilateral blur
Contrast	[-0.5, 0.5]
Zoom	[0, 0.2]
Flip	All images

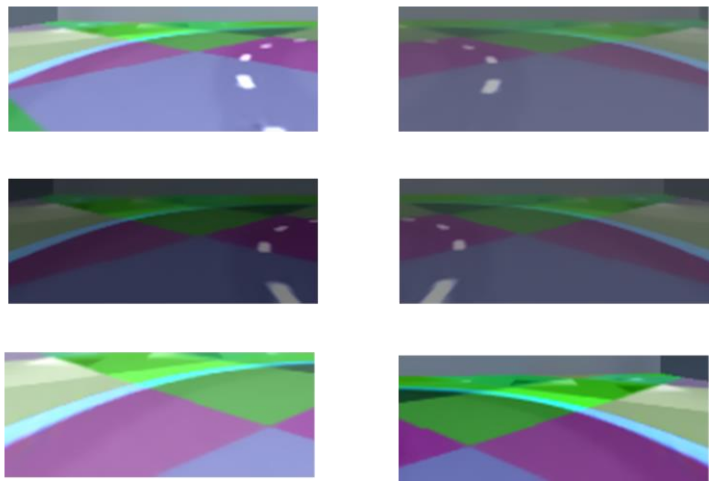


Figure 3-8 Data Augmentation example images

3.5 Traffic Sign Recognition

Because this project is not intended to compete in the Carolo Cup, it was decided to use Australian traffic signs rather than German ones. As discussed in the Literature Review, there is no existing traffic sign dataset for Australia available to the public; therefore, a new dataset was required. Unfortunately, typical ways of training traffic sign recognition using a suitably large, labelled (i.e., ~50,000 labelled images) dataset was impossible. This project, therefore, used a pre-trained MobileNet-SSD object detection model and applied transfer learning to recognise new classes of objects based on a limited dataset. Data collection was undertaken by driving the robot around the track using the PilotNet model. As the robot navigated the track, it would periodically save images from the robot's perspective, which included mock Australian signs shown in Figure 3-9. These images were then manually labelled using an open-source program called LabelImg [56], as shown in Figure 3-10.

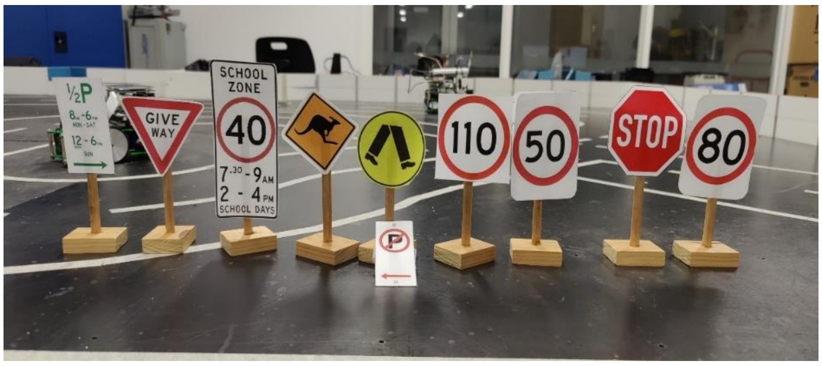


Figure 3-9 Australian Traffic sign subset

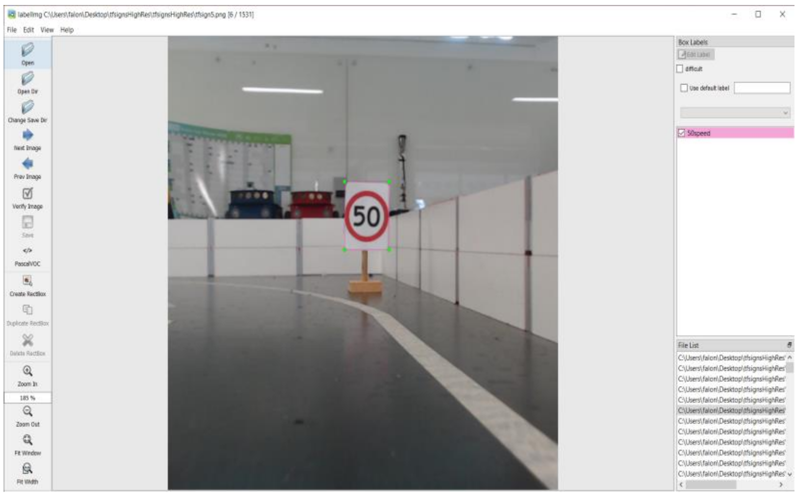


Figure 3-10 Labelling traffic signs using LabelImg

3.6 Speed Limit Recognition

One problem that arose from the traffic sign recognition was detecting and classifying different speed limits. The initial method used would tend to classify the speed limit with the most instances in training which at the time was the 80 km/h speed limit. The initial method could, however, differentiate between 40 and 80 km/h speed limits, likely due to the different shape and format of the speed limit sign. Therefore, speed limit detections were generalised to be speed limit signs regardless of the speed limit, as shown in Figure 3-11. A third model using computer vision techniques which isolated the digits in a speed limit and passed these images to a digit classifier trained on the MNIST dataset (i.e., a large labelled dataset of hand-drawn digits from 0-9) was implemented as shown in Figure 3-12 [57]. The CNN model code used is shown in Appendix 7.1. The computer vision techniques used to isolate the digits are:

1. Grayscale conversion;
2. Detect circle and convert all pixels to black which are not in the circle;
3. Binarize the image using Otsu thresholding;
4. Detect contours with a minimum and maximum size;
5. Padding to retain aspect ratio;
6. Resize the image to 28x28.

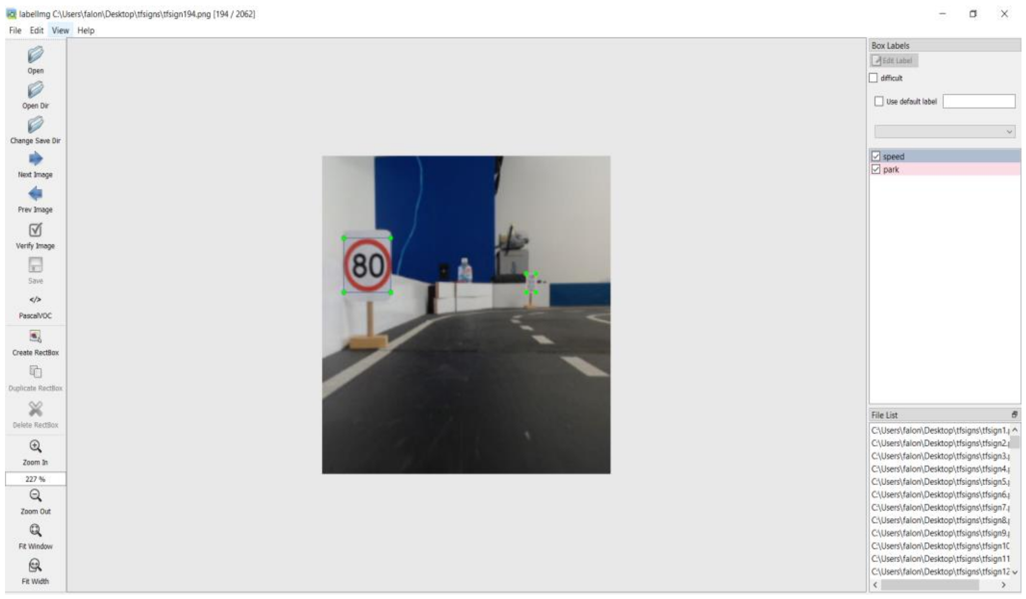


Figure 3-11 Example of labelling all speed limits the same

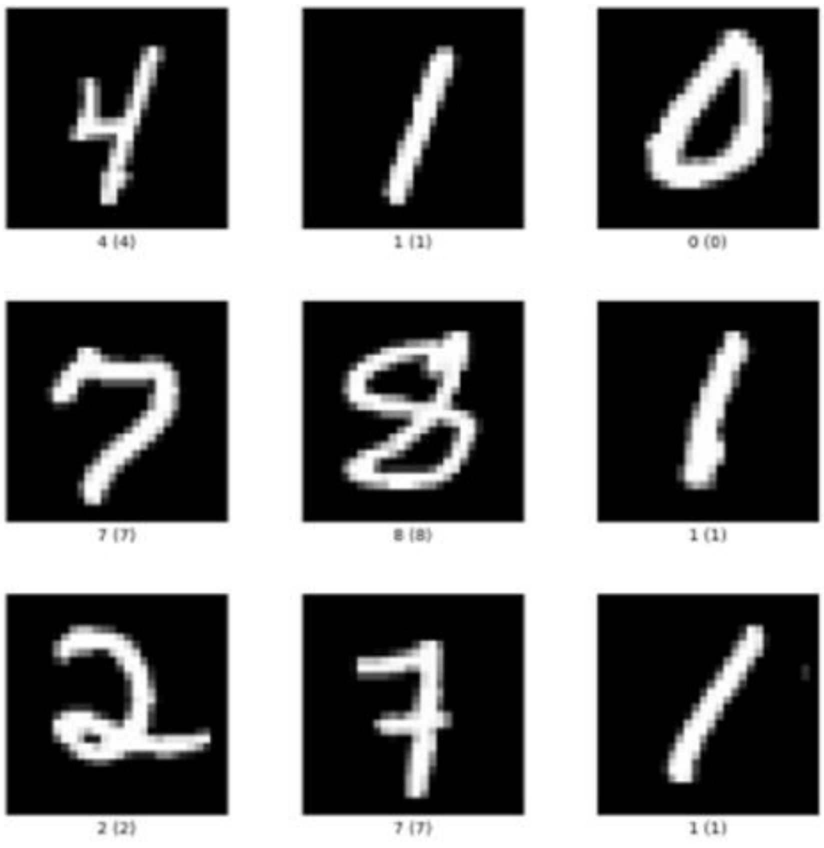


Figure 3-12 MNIST example data [58]

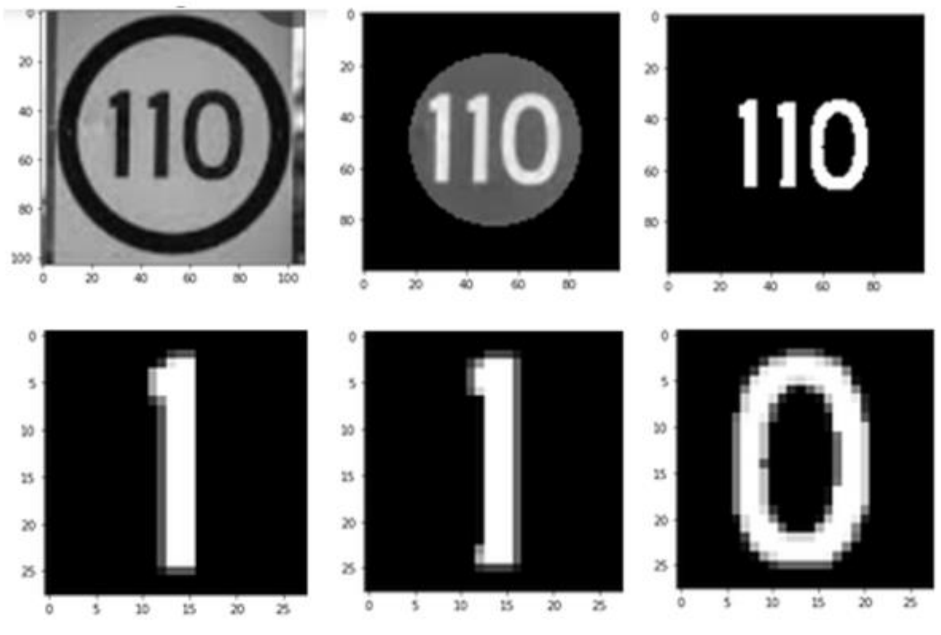


Figure 3-13 Ordered left-right, top-bottom computer vision steps specified earlier, to isolate the digits

3.7 Evaluating Success

3.7.1 Autonomous Rate (AR)

The autonomous driving rate is used as a measure for autonomous driving systems. NVIDIA defined the autonomous driving rate as follows [27].

$$A.\mathrm{R.} = \left[1 - \frac{\mathrm{N(interventions)} \times 6\ seconds}{\mathrm{elapsed\ time\ [seconds]}}\right] \times 100\% \quad (1)$$

Their formula is biased toward slow-moving vehicles even if track performance is the same. Previous UWA students, however, considered this aspect and developed a formula that was independent of the speed of the vehicle [6] and the formula was as follows:

$$A.\mathrm{R.} = \left[1 - \frac{\mathrm{N(interventions)}}{N\mathrm{(track\ segments)} \times \mathrm{N(completed\ circuits)}}\right] \times 100\% \quad (2)$$

Where:

1. Intervention is defined as any deviation outside the white lanes (i.e., front middle wheel crosses the line). This project does not distinguish between a near miss and failure like the past UWA students, as the car width used in this project is slightly larger than the width of the track, and therefore, near misses cannot be determined (refer Figure 3-21).
2. Track segment is defined the same as the previous UWA students, whereby the number of track segments is the number of individual driving tasks, for example, transitioning from straight to turn and turn to straight, as shown in Figure 3-14.

This project used the autonomous driving rate defined by the past UWA students (refer to equation (2)).

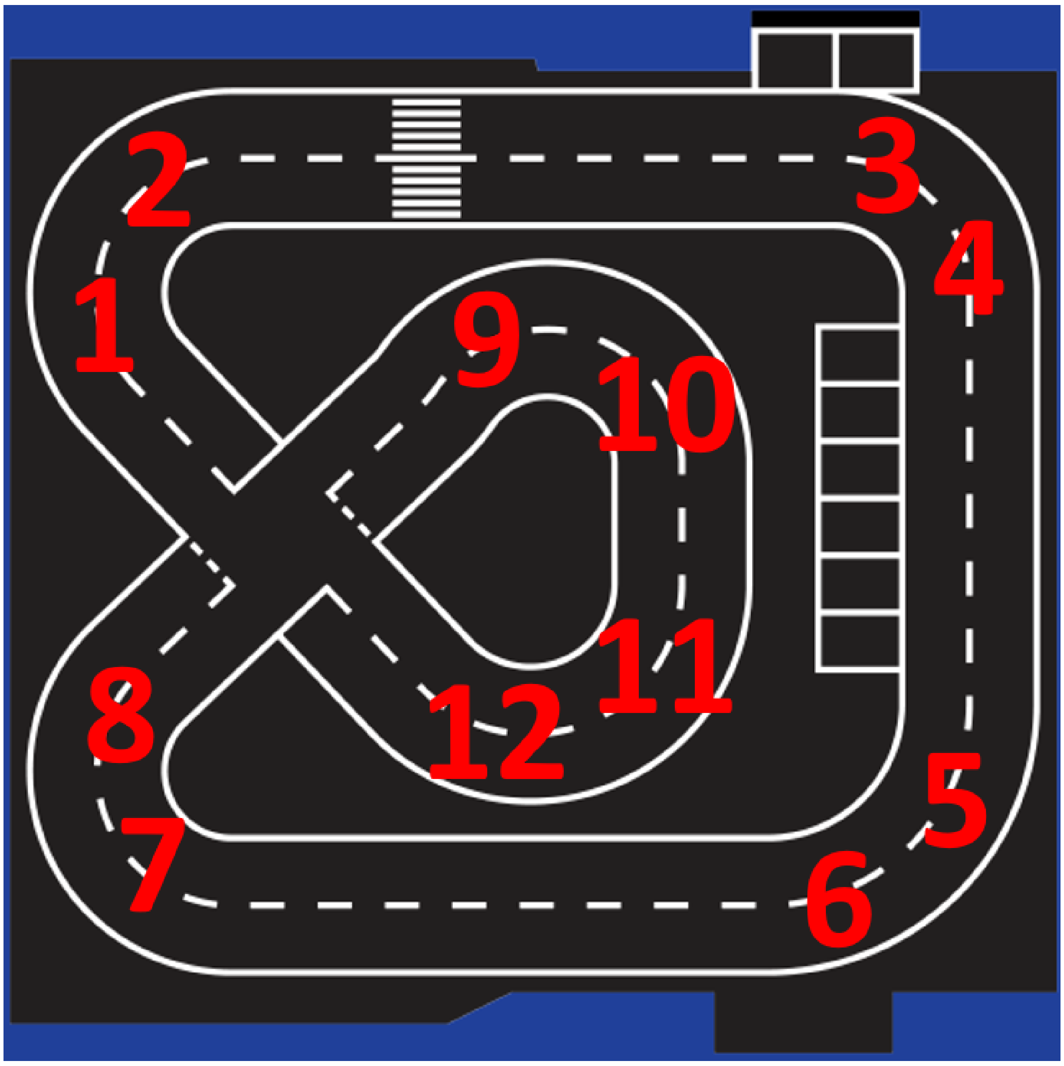


Figure 3-14 Track segmentation example

3.7.2 Average Precision (AP)

A popular method of measuring object detection model performance is using Average Precision. Average Precision requires definitions for True Positive (TP), False Positive (FP), True Negative (TN), and False Negative (FN). These are defined using the Intersection over Union (IoU) formula as defined in Equation (3) and visualised in Figure 3-15 [59].

$$IoU = \frac{truth \cap predicted}{truth \cup predicted} \quad (3)$$

Where:

1. If the correct object is detected and IoU >= 0.5, then classify it as TP.
2. If the correct object is detected and IoU < 0.5, then classify it as FP.

3. If an object is present and the model failed to detect it, classify it as FN.
4. Lastly, TN is every part of the image where the model did not predict an object, and no object was there, which is not helpful and therefore ignored.

Examples of TP, FP, and FN are shown in Figure 3-16.

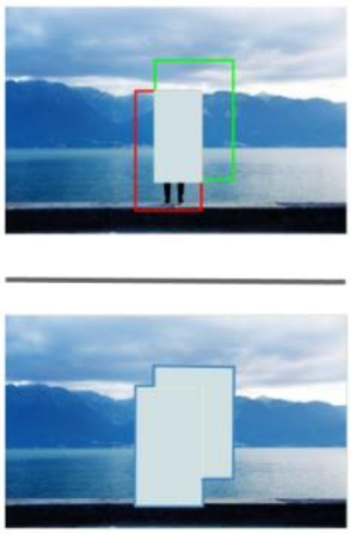


Figure 3-15 Intersection over Union visualized [60]

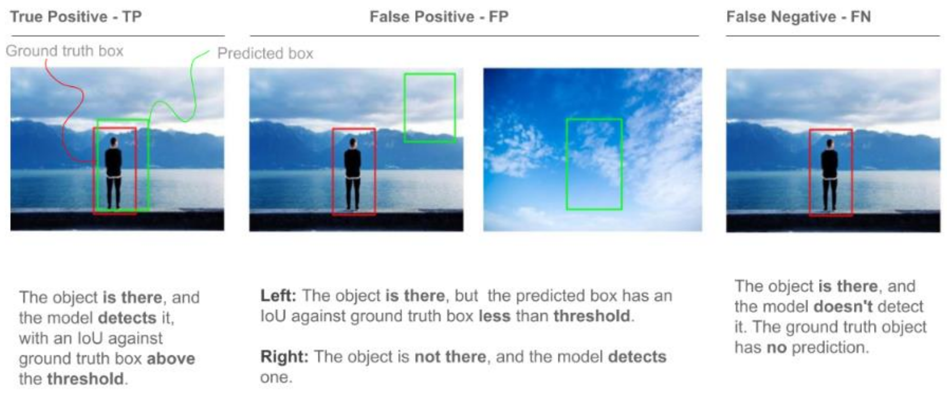

True Positive - TP
Ground truth box
Predicted box
False Positive - FP
False Negative - FN
The object **is there**, and the model **detects** it, with an IoU against ground truth box **above** the **threshold**.
Left: The object **is there**, but the predicted box has an IoU against ground truth box **less** than **threshold**.
Right: The object is **not there**, and the model **detects** one.
The object **is there**, and the model **doesn't** detect it. The ground truth object has **no** prediction.

Figure 3-16 TP, FP, FN object detection example [60]

Precision is the accuracy of positive predictions defined in equation (4). Recall is the ratio of positive instances correctly detected as defined in equation (5) [61].

$$Precision = \frac{TP}{TP+FP} \quad (4)$$

$$Recall = \frac{TP}{TP+FN} \quad (5)$$

Both precision and recall are suitable measurements for model performance. However, a trade-off exists between precision and recall. Choosing a threshold for determining a class to be positive or negative will inversely affect precision and recall, as shown in Figure 3-17. AP encapsulates precision and recall and summarizes the precision-recall curve by AP across recall values from 0 to 1.

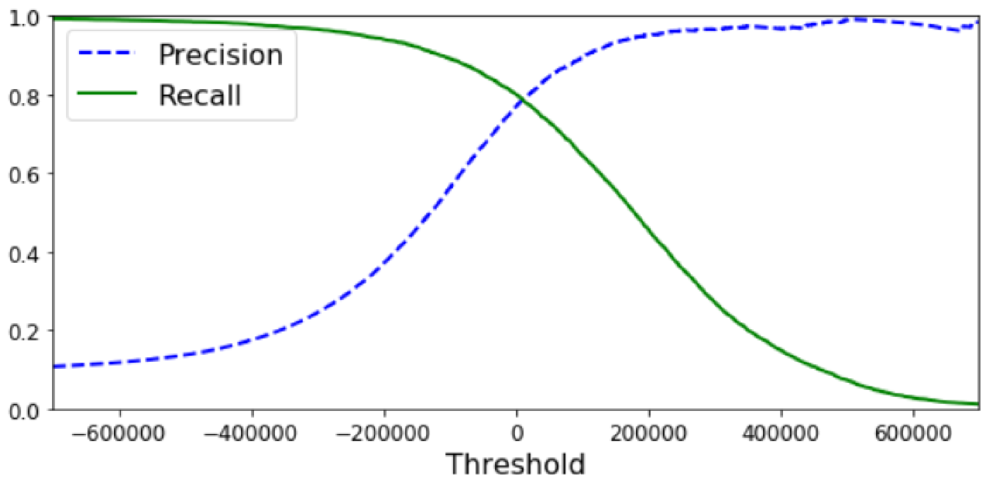


Figure 3-17 Precision Recall trade-off visualised [61]

3.7.3 Evaluating Speed Limit Recognition Performance

This project uses an ad-hoc method to recognise speed limits. Detections from the TSR, classified as speed limits, are passed to the speed limit recognition model. Many images can be given to the model before a determination is made. This is because pictures can be too blurry, or too far away. Therefore, to assess the speed limit recognition, a test was developed to measure the angle and distance at which the speed limit could be recognised, as shown in Figure 3-18.

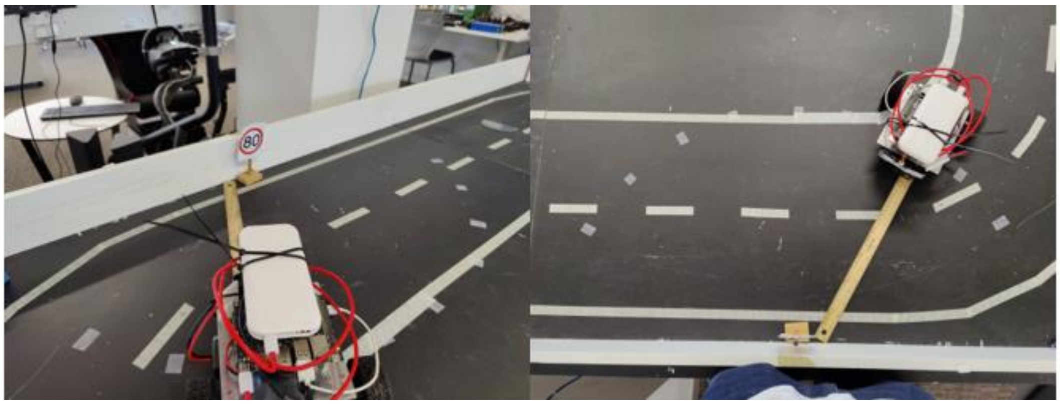


Figure 3-18 Speed Limit Recognition test method

3.7.4 Evaluating Detection Range of TSR

Finally, the range for detecting traffic signs at a consistent 60% confidence threshold was also measured, as shown in Figure 3-19.

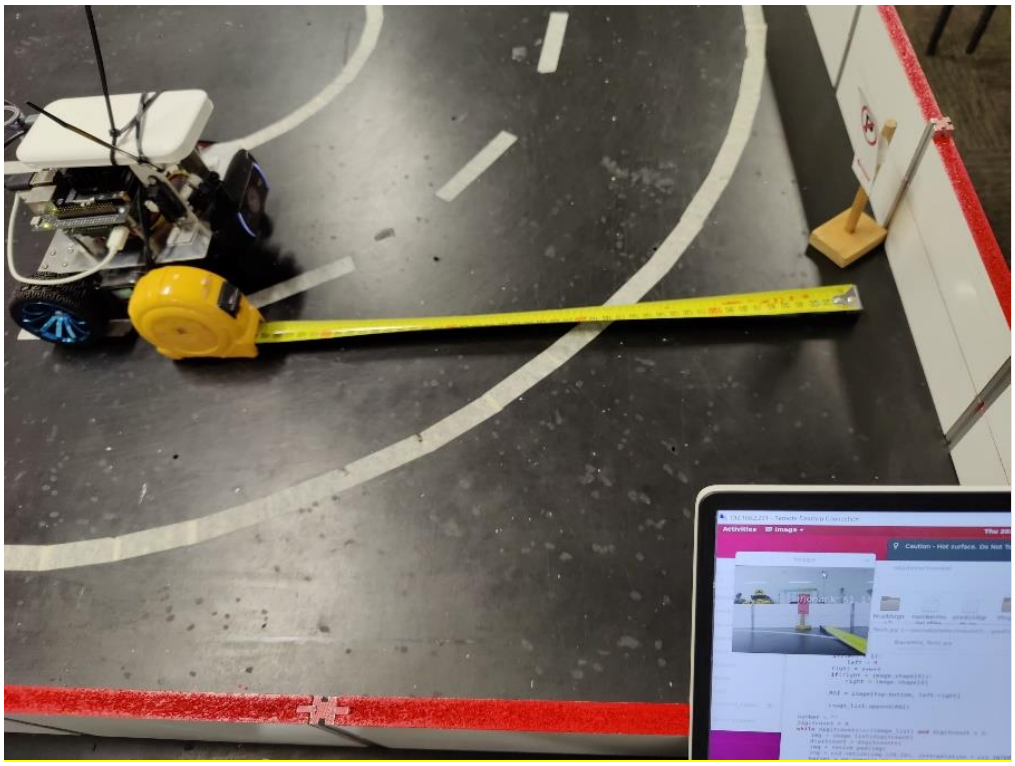


Figure 3-19 Traffic sign detection range assessment method

3.8 Final Design

The robot's final hardware design and size are shown in Figure 3-20 and Figure 3-21. In addition, a summary of the hardware components, DNN models, and important software is provided in Table 3-2. A flow chart showing the interaction of the software systems is shown in Figure 3-22.

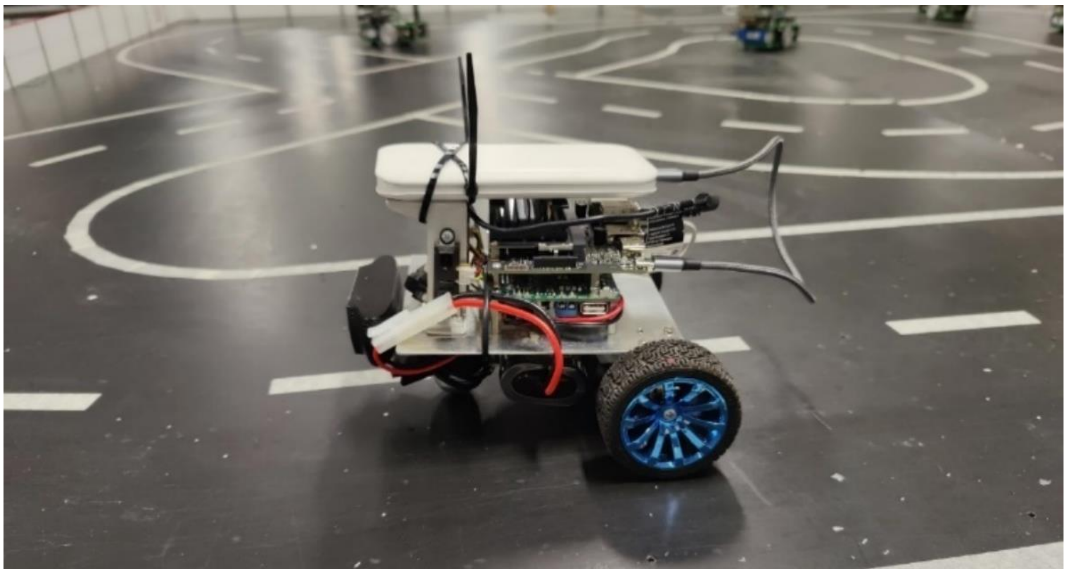


Figure 3-20 Final design of robot

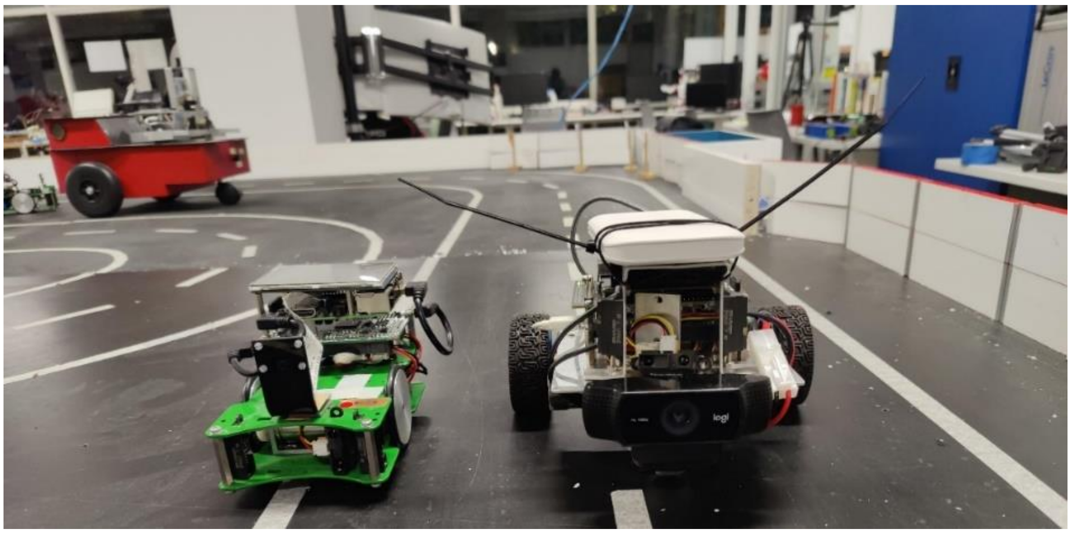


Figure 3-21 Size comparison of EyeBot SoccerBot used in previous projects (left) vs robot (right)

Table 3-2 System Components and Deep Learning Software Networks

Hardware/Software	Component
Hardware	EyeBot 7 I/O board
	10000 mAh mobile power bank
	Jetson Nano 2GB Developer Kit
	Logitech C920 USB Webcam
	7.2V battery
	3x PSD sensors
	2 Aslong 6V motors and wheels
	Metal prototype frame
DNN models	PilotNet – Lane Keeping
	MobileNet-SSD - TSR
	CNN – Speed Limit Recognition
Important Software Versions	Jetpack 4.6
	TensorRT 8.0.1
	CUDA 10.2
	Python 3.6
	TensorFlow and Keras 2.6.0

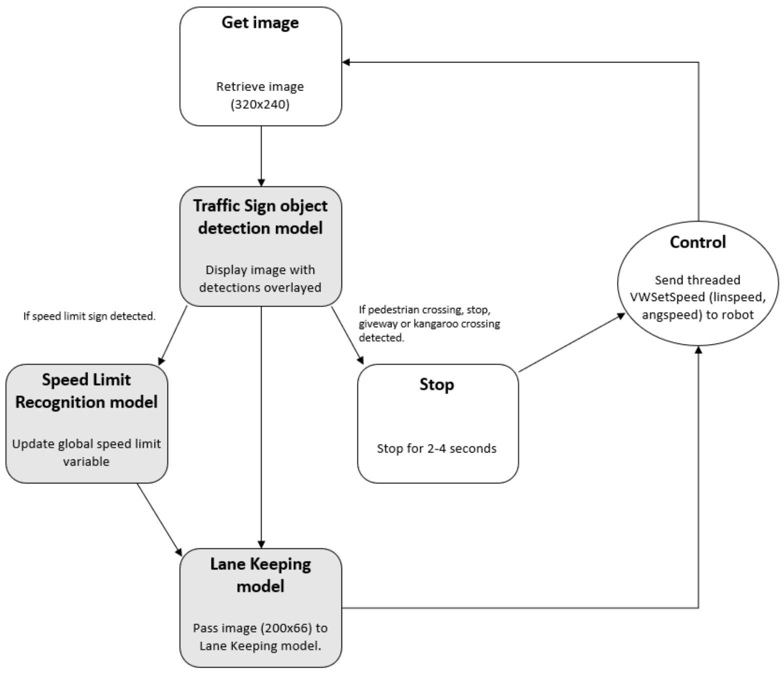

Get image
Retrieve image (320x240)
Traffic Sign object detection model
Display image with detections overlayed
If speed limit sign detected.
If pedestrian crossing, stop, giveway or kangaroo crossing detected.
Speed Limit Recognition model
Update global speed limit variable
Stop
Stop for 2-4 seconds
Control
Send threaded VWSetSpeed (linspeed, angspeed) to robot
Lane Keeping model
Pass image (200x66) to Lane Keeping model.

Figure 3-22 Control Loop diagram

4 Results

4.1 Lane Keeping

4.1.1 Training Loss and Accuracy

The dataset used to train the PilotNet model consisted of 30,542 images. It was trained over 100 epochs at a learning rate of 0.001. Figure 4-1 shows the training and validation loss over those 100 epochs.

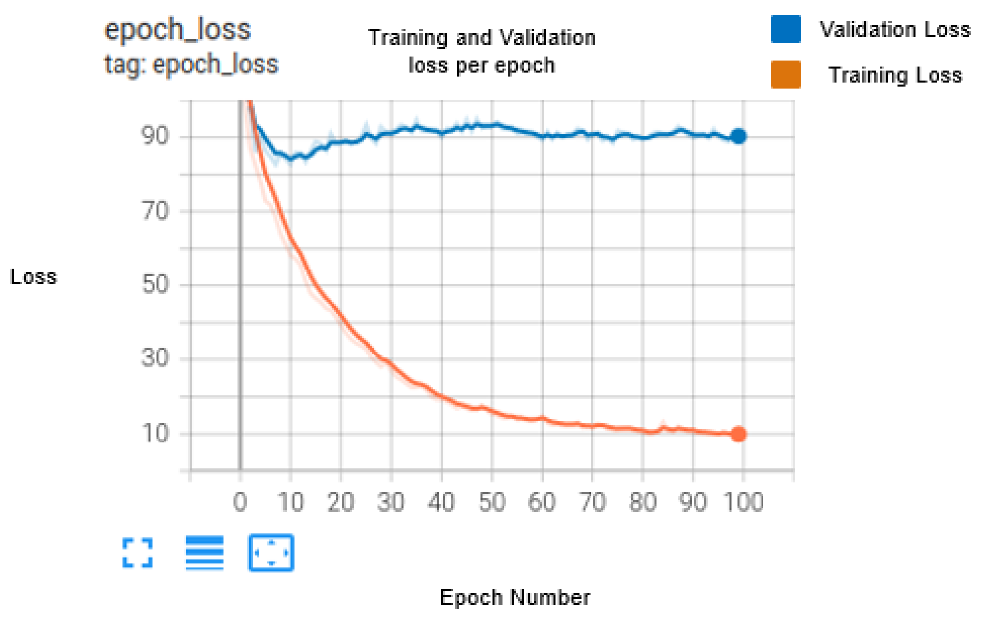


Figure 4-1 Training results

The training loss converges to a much lower value than the validation loss (refer Figure 4-1). This indicates that the model overfitted the training data and had a significantly higher loss for data it hadn’t seen (i.e., validation images). Some methods to reduce the overfitting were attempted, such as reducing the initial learning rate and the learning rate during training and early stopping; however, this didn’t improve the overfitting and made the loss greater. The problem is likely caused by the dataset developed for this project. Possibly too much data augmentation was performed, and better validation loss convergence could have been achieved if only some images were augmented. Alternatively, the model could be changed from a continuous regression output to several discrete outputs such as left, slight left, straight, slight right, and right.

4.1.2 AR Test Results

The robot's autonomy was tested both on the simulator and the scaled version of the Carolo test track in the robotics laboratory. The test matrix used to determine AR in the simulator is summarised in Table 4-1. Testing was always done using the outer lane because corners on the inside lane can be too tight. The tracks used in the testing are shown in Figure 4-2 to Figure 4-5.

Table 4-1 AR Simulator Test Matrix

Test Number	Track	Number of Segments	Lane	Track seen in training?
1	1	12	Left	Yes
2	1	12	Right	Yes
3	2	10	Left	Yes
4	2	10	Right	Yes
5	3	16	Left	No
6	3	16	Right	No
7	4	10	Left	No
8	4	10	Right	No

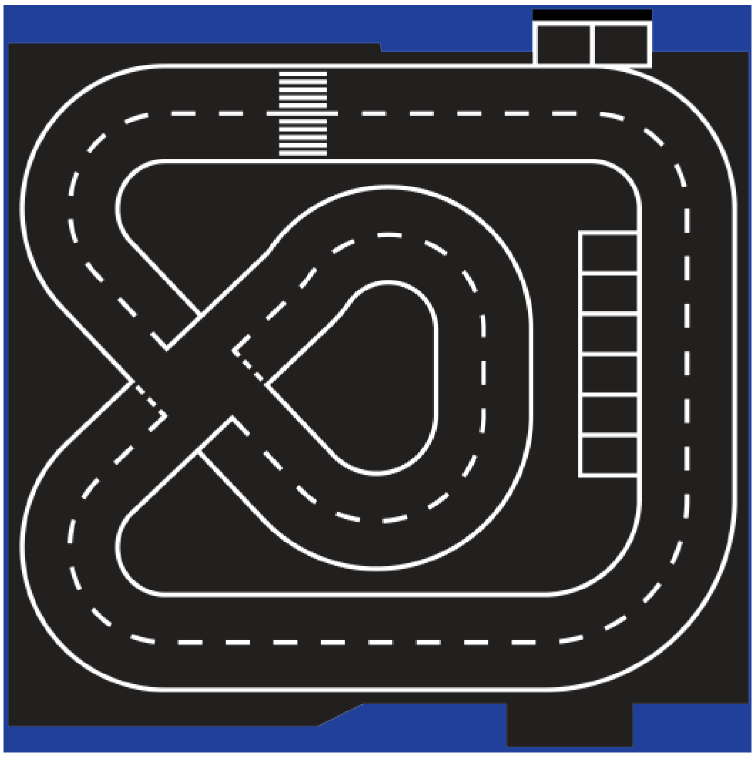


Figure 4-2 Track 1, 12 track segments in total (seen)

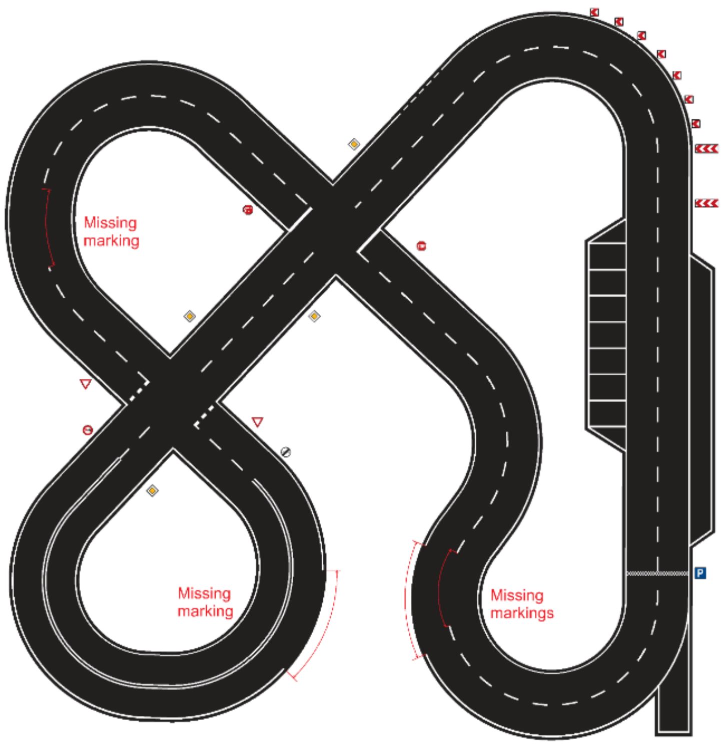


Figure 4-3 Track 2, 10 track segments in total (seen)

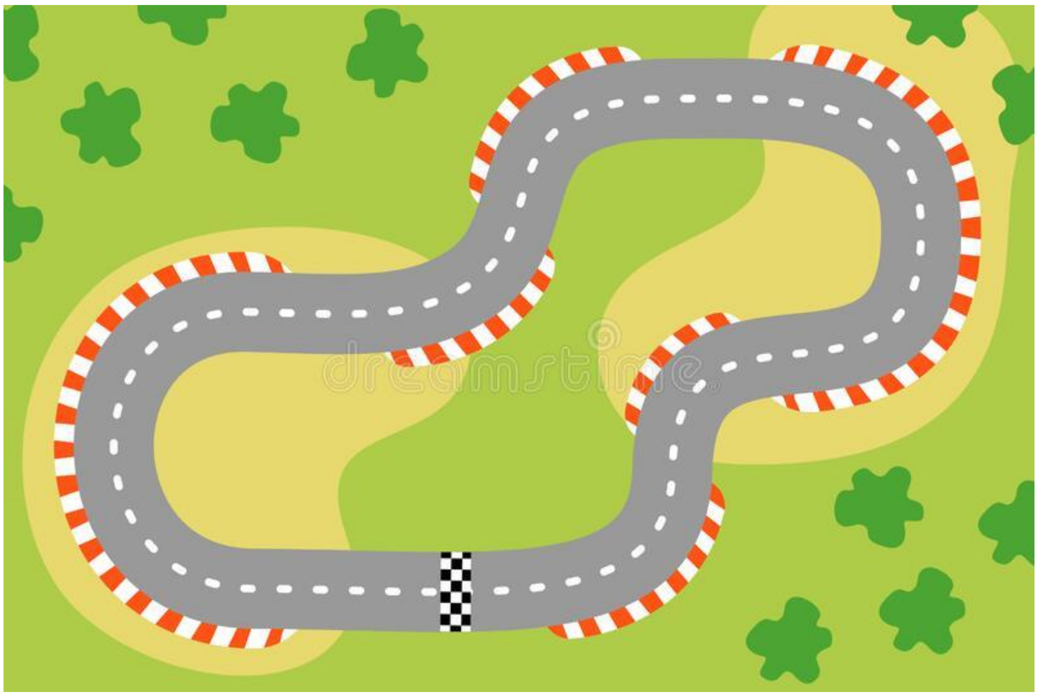


Figure 4-4 Track 3, 16 track segments in total (unseen)

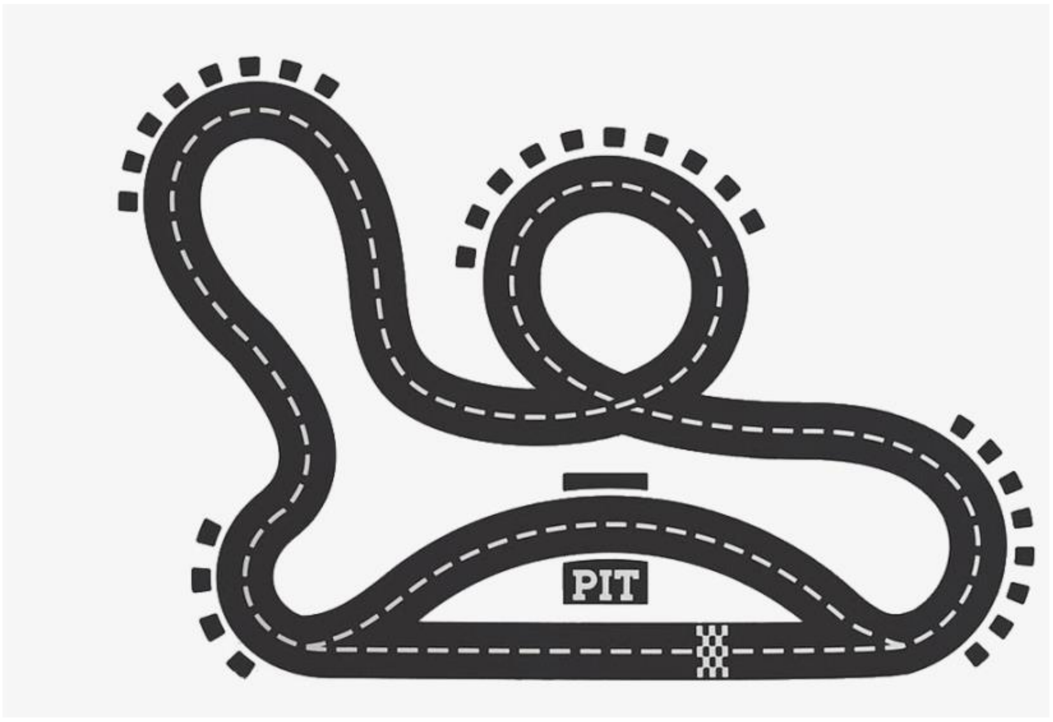


Figure 4-5 Track 5, 10 track segments in total (unseen) (pit lane is ignored)

The AR results for the simulator testing are summarised in Table 4-2.

Table 4-2 PilotNet Simulator Autonomy Rate Tests for Seen and Unseen Test Tracks

Test Number	Track Segments	Lap 1 Interventions	Lap 2 Interventions	Lap 3 Interventions	Total Interventions	A.R
1	12	0	0	0	0	100%
2	12	0	0	0	0	100%
3	10	1	1	1	3	90%
4	10	0	0	0	0	100%
5	16	1	1	1	3	93.8%
6	8	0	0	0	0	100%
7	10	3	3	3	9	70%
8	10	0	2	0	2	93.3%

The average AR for the tracks that the PilotNet model was trained on using the simulator was 97.5%. This compares with 89.3% for the unseen tracks. This reduction is less than 10% and could be attributed to the overfitting of the data. Using more tracks, a larger dataset, increasing domain randomisation, and including data that corrects the robot in situations where it begins to deviate off the track would improve the generalisation and, therefore, the AR on unseen tracks.

Testing for the Carolo real test track was done at different times to evaluate the effect of light in the robotics laboratory, and the test matrix is summarised in Table 4-3. Note the testing was conducted at UWA during May and always in the outside lane.

Table 4-3 PilotNet Carolo Real Test Track Matrix

Test Number	Number of Segments	Lane	Time of Test
1	12	Left	Night
2	12	Right	Night
3	12	Left	Midday
4	12	Right	Midday
5	12	Left	4:30 pm
6	12	Right	4:40 pm

The results for testing on the real Carolo test track are summarised in Table 4-4.

Table 4-4 PilotNet Carolo Real Test Track AR Results

Test Number	Track Segments	Lap 1 interventions	Lap 2 interventions	Lap 3 interventions	Total interventions	A.R
1	12	1	0	0	1	94.4%
2	12	0	3	1	4	88.9%
3	12	1	2	0	3	91.7%
4	12	1	2	2	5	86.1%
5	12	1	2	2	5	86.1%
6	12	4	4	4	12	66.7%

These results can be compared with previous student work that had AR scores of 100%, 100%, and 66.7% for constant speeds of 50mm/sec, 100mm/sec, and 150mm/sec, respectively, when considering failures only (and not near misses) [6]. The results for this project are lower compared to the previous testing (i.e., the average AR for midday and night is 90.2%), however the difference isn’t unreasonable (<10%), considering time for this project was split amongst other tasks and not solely for training a PilotNet model. It should be noted that the speed used in the testing for this project was not constant but used a formula as defined in equation (6) which allowed the robot to slow down around corners and speed up on straights.

$$Vehicle\ Speed\ (mm/s) = 130 - 2.5 * (predicted\ angle\ speed) \quad (6)$$

Additionally, the results show a decrease in AR depending on the time of day. AR at night-time is highest, followed closely by midday, and then finally, a significant drop in AR occurs at 4.30 pm when the sun's angle influences the image of the track the robot sees. The robotics laboratory has two sets of windows facing west and east, as shown in Figure 4-6. This source of natural light in the robotics laboratory changes the performance of the PilotNet model at different times of the day due to the reflection from the track. More randomisation of the training images' brightness and contrast could help the model's generalisation in these conditions. Alternatively, mixing the data collected in the simulator with data collected on the real robot may improve the model's performance in challenging lighting conditions as the model will be exposed to some of them during training. An investigation could be done by analysing the saliency map of the input in each layer of the model, but this was outside of the scope of this project. This is because a saliency map can reveal the pixels which have high activation and therefore are important to the model. It can then be inferred if the reflected light has high activation in the model or if it is only the road lines. An example is shown in Figure 4-7.

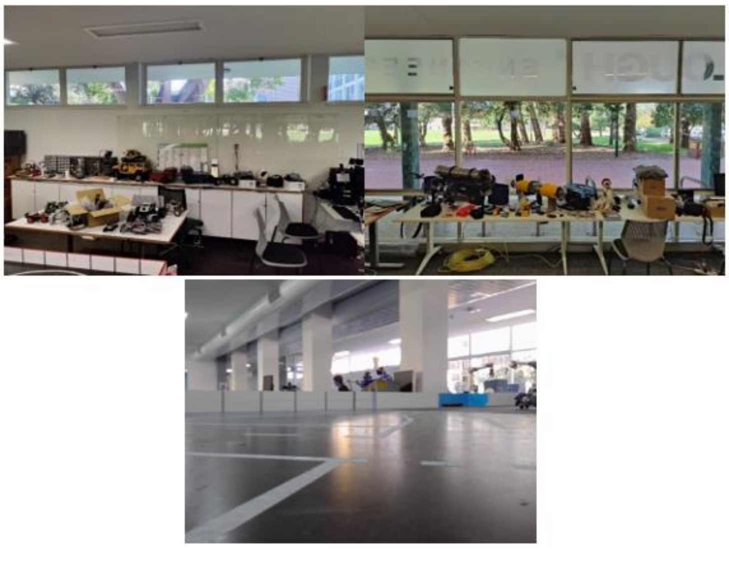


Figure 4-6 Windows facing the East, West and image taken from robot during afternoon

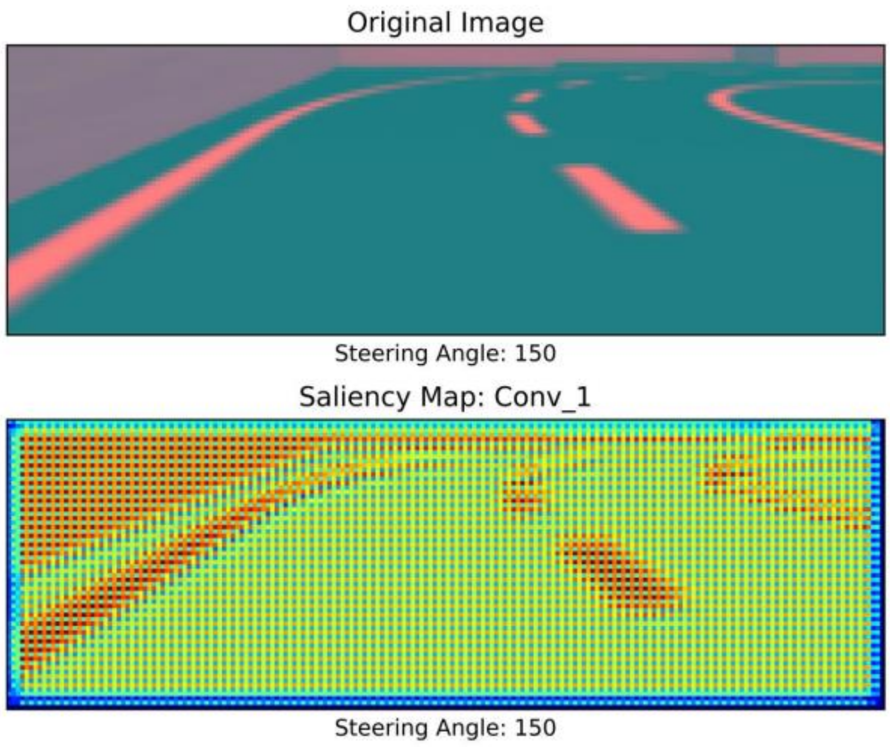


Figure 4-7 Saliency map example in TensorFlow 1 [55]

4.2 Traffic Sign Recognition

4.2.1 Dataset Development

The dataset for the traffic signs was developed by driving the robot around the Carolo track, taking pictures, and labelling them as described in section 3.5. A total of 1715 traffic sign images were labelled for training out of 3140 images. The distribution of traffic signs in the dataset are shown in Figure 4-8.

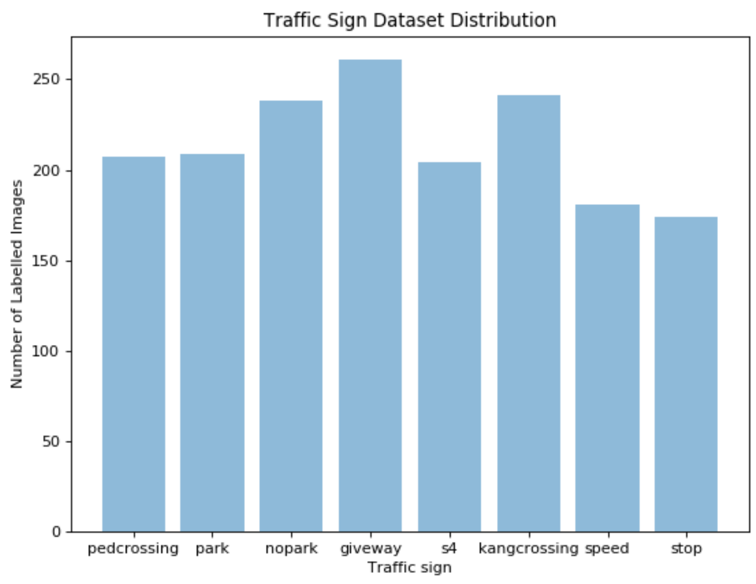


Figure 4-8 Australian traffic sign dataset class distribution (Note s4 refers to the SoccerBot, which was included in the training of the TSR)

4.2.2 Training Loss

The training validation loss of the model trained using Transfer Learning is shown in Figure 4-9.

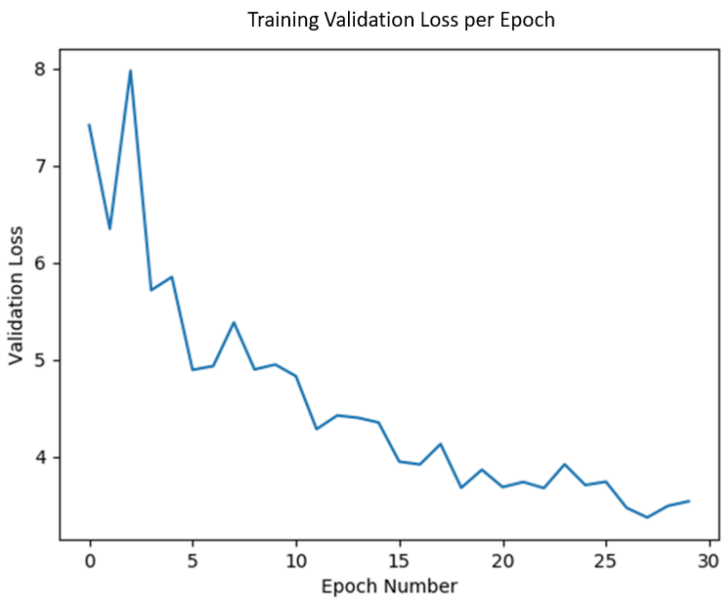


Figure 4-9 Validation loss during training

This figure shows that the model's training achieved the lowest validation loss and, therefore, the highest level of recognition after 26 epochs; hence this was the final model used.

4.2.3 Measuring Traffic Sign Recognition Average Precision

The average precision (AP) of the original MobileNet-SSD model was 0.676 when it was trained to recognise 19 objects in the COCO dataset [62]. The AP for each class is provided in Table 4-5. After applying Transfer Learning to this model with the Australian traffic sign dataset, the average precision across all classes was 0.634, as shown in Table 4-6. This is only slightly lower than the AP of the original model of 0.675. However, this is lower than AP results from researchers at the Auckland University of Technology. They measured an AP score of 0.9014 and 0.9770 using their SSD and YOLOv5 TSR models, respectively [48]. Unlike this project, which was on embedded hardware, their project used a powerful desktop computer. Similarly to this project, their dataset was small; they had 2182 labelled traffic signs and only eight classes of traffic signs [48]. Nevertheless, their AP results were high and could be used as a future benchmark for UWA projects.

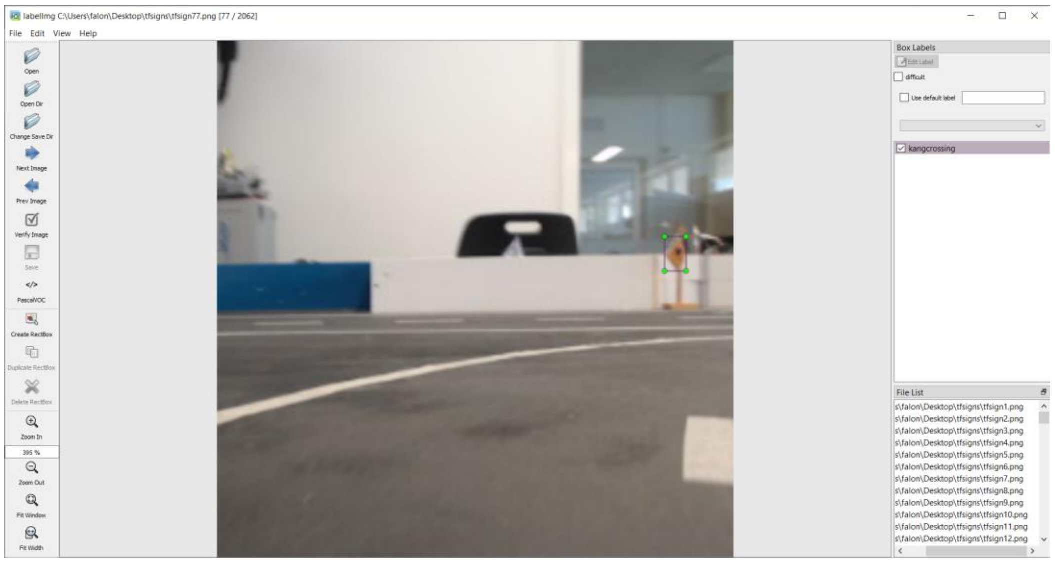
Comparing Table 4-6 and Figure 4-8 indicates that traffic signs with low occurrences in the dataset had a higher AP. For example, speed limit and pedestrian crossing had low occurrences in the dataset but high AP. In comparison, traffic signs with high occurrence in the dataset, such as give way and kangaroo crossing, had lower AP. This is contrary to the expectation that traffic signs with high occurrence in the dataset would be more accurately detected and classified. A possible explanation for this could be due to the additional images for these classes were of lower quality, and perhaps they were being labelled from a far distance in the training data, as shown in Figure 4-10. Otherwise, there are examples in Table 4-5 where some objects had low AP, including bottles and boats, compared to the mean AP, indicating a degree of variability is expected in this type of detection method.

Table 4-5 Original AP results for MobileNet-SSD model before Transfer Learning [62]

Object Class	Average Precision
Aeroplane	0.674
Bicycle	0.791
Bird	0.612
Boat	0.562
Bottle	0.347
Bus	0.774
Car	0.728
Cat	0.836
Chair	0.514
Cow	0.624
Dining table	0.706
Dog	0.785
Horse	0.820
Motorbike	0.796
Person	0.704
Sheep	0.607
Sofa	0.755
Train	0.823
TV monitor	0.646
Average Precision across all classes	0.676

Table 4-6 AP for TSR after Transfer Learning

Traffic Sign	Average Precision
Give way	0.419
Kangaroo crossing	0.535
No parking sign	0.461
Parking sign	0.561
Pedestrian Crossing	0.840
Soccer Bot (S4)	0.666
Speed limit sign	0.827
Stop sign	0.761
AP across all classes	0.634



```
s\falon\Desktop\tfsigns\tfsign1.png
s\falon\Desktop\tfsigns\tfsign2.png
s\falon\Desktop\tfsigns\tfsign3.png
s\falon\Desktop\tfsigns\tfsign4.png
s\falon\Desktop\tfsigns\tfsign5.png
s\falon\Desktop\tfsigns\tfsign6.png
s\falon\Desktop\tfsigns\tfsign7.png
s\falon\Desktop\tfsigns\tfsign8.png
s\falon\Desktop\tfsigns\tfsign9.png
s\falon\Desktop\tfsigns\tfsign10.png
s\falon\Desktop\tfsigns\tfsign11.png
s\falon\Desktop\tfsigns\tfsign12.png
```

Figure 4-10 Example of traffic sign which is labelled from a long distance

4.2.4 Maximum Detection range

The maximum detection range of each traffic sign is summarised in Table 4-7. Interestingly the 80 speed limit and 40 speed limit signs significantly differed in detection range. This is potentially due to the 40 speed limit being smaller within the overall school zone speed limit sign. The size of the traffic sign in the image affects the model's ability to detect and classify accurately. Therefore, a better test in the future would be to consider the minimum number of pixels a traffic sign needs to be before 60% confidence threshold detection is made; this would mean it could be compared to other TSR models designed to detect full-scale traffic signs on public roads.

Table 4-7 Traffic Sign Detection Range – Consistent 60% Confidence Threshold.

Traffic sign	Distance (cm)
Stop	47
Give way	27
Pedestrian crossing	36
Kangaroo crossing	35
No parking	45
Parking	73
80-speed	45
40-speed	36

4.3 Speed limit recognition

4.3.1 Dataset

The dataset used for detecting the speed limits was the MNIST dataset [63]. The number of images used in training per class is shown in Figure 4-11. The validation accuracy and loss for classifying the speed limit digits are shown in Figure 4-12 and Figure 4-13.

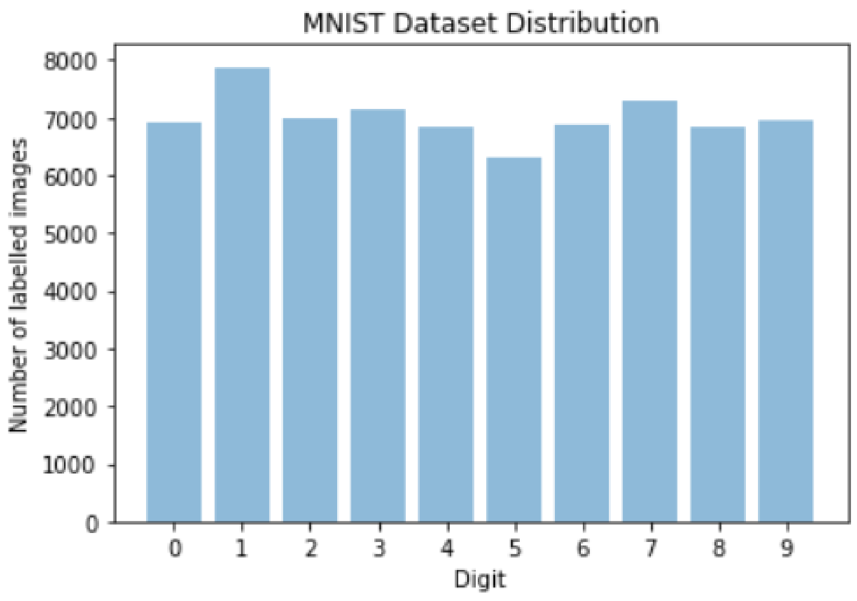


Figure 4-11 MNIST dataset distribution

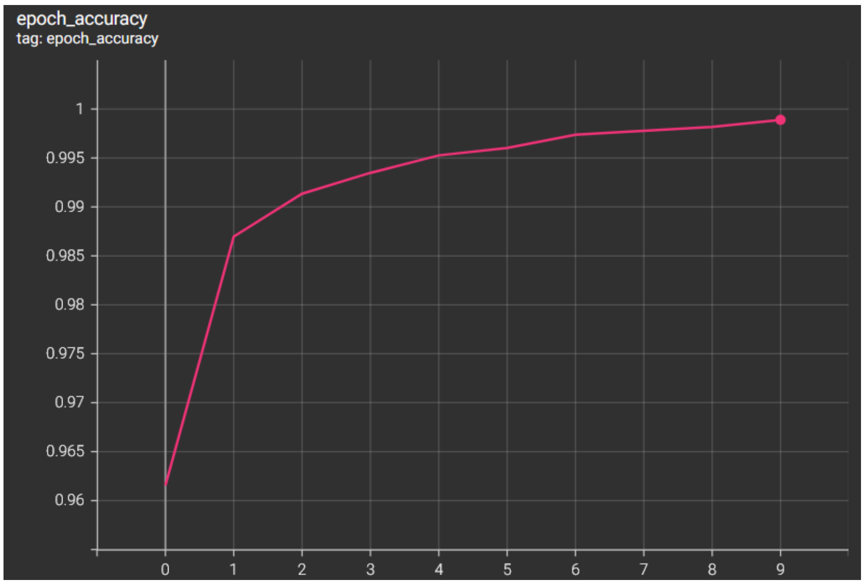


Figure 4-12 Validation accuracy over 10 epochs

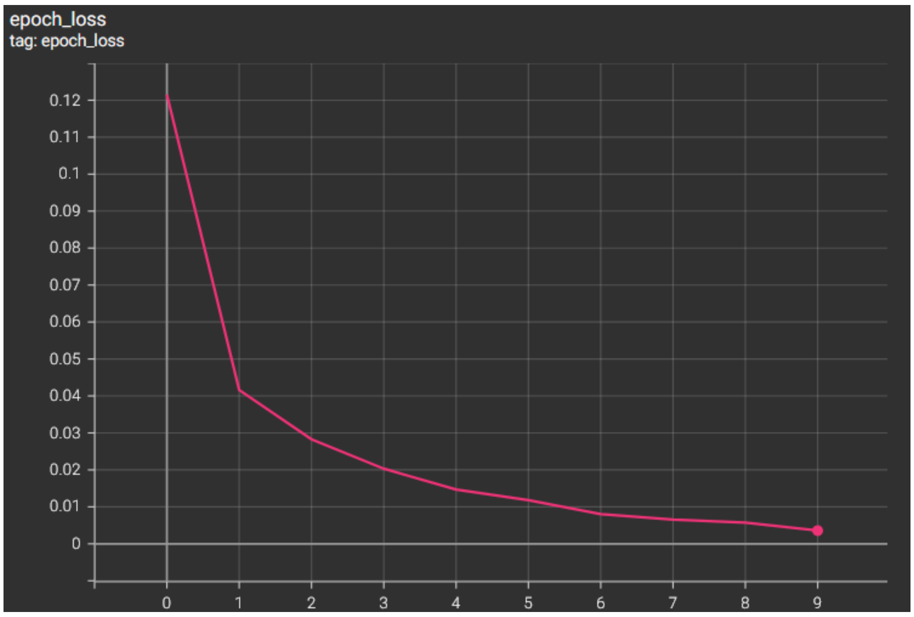
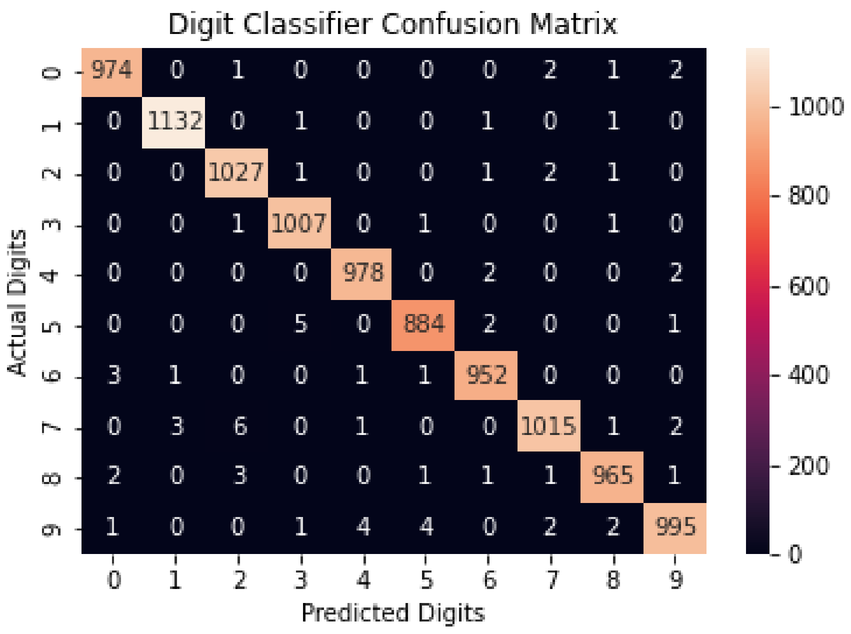


Figure 4-13 Validation loss over 10 epochs

The figures show a high level of accuracy and a low level of loss after ten epochs. The confusion matrix for the speed limit digit classification is shown in Figure 4-14.



Actual \ Predicted	0	1	2	3	4	5	6	7	8	9
0	974	0	1	0	0	0	0	2	1	2
1	0	1132	0	1	0	0	1	0	1	0
2	0	0	1027	1	0	0	1	2	1	0
3	0	0	1	1007	0	1	0	0	1	0
4	0	0	0	0	978	0	2	0	0	2
5	0	0	0	5	0	884	2	0	0	1
6	3	1	0	0	1	1	952	0	0	0
7	0	3	6	0	1	0	0	1015	1	2
8	2	0	3	0	0	1	1	1	965	1
9	1	0	0	1	4	4	0	2	2	995

Figure 4-14 Confusion matrix for speed limit digit classification

The test accuracy achieved using this method was 99.3% which is similar to other image classifiers researchers have trained using this dataset [64, 65]. The confusion matrix shows the most incorrect predictions when the model attempts to predict a digit with ground truth seven instead of predicting digit two. This occurred six times out of 1027 classifications (refer Figure 4-14) and is likely due to the shape of a '2' being similar to a '7'.

An alternative method for classifying the speed limits would be to use the AS1744 series of fonts (i.e., the same font used to create speed limit signs in Australia) to create a dataset by making 1000s of slightly varied digits (from 0-9) in this font. This should be considered in future work and then compared to the performance of the digit classifier trained on the MNIST dataset.

4.4 Speed Limit Manual Testing

Manual testing was done to assess the speed limit recognition; the results are shown in Table 4-8. Speed limit signs were placed against a wall, and the robot moved to different angles from the centreline of the sign, as shown in Figure 4-15 and the maximum distance was measured for detection at each angle.

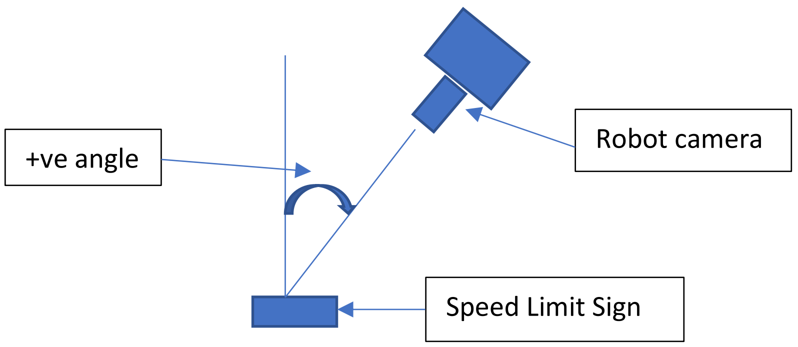


Figure 4-15 Speed limit angle and distance recognition testing

The model could not recognise any speed limits at angles greater than plus or minus 60°. Interestingly, the model could detect speed limits at positive 45° but not at negative 45°. It is unclear why this was the case, perhaps the testing was skewed in some way, or the lighting of the speed limit from the left and right direction could be different.

Table 4-8 Manual Speed Limit Recognition Testing (ND = Not Detected)

Speed Limit (km/h)	Angle (°)								
	+60	+45	+30	+15	+0	-15	-30	-45	-60
110	ND	24 cm	30 cm	30 cm	33 cm	28 cm	30 cm	ND	ND
80	ND	17 cm	31 cm	31 cm	34 cm	35 cm	29 cm	ND	ND
50	ND	15 cm	31 cm	31 cm	34 cm	35 cm	29 cm	ND	ND
40	ND	27 cm	31 cm	29 cm	32 cm	30 cm	26 cm	ND	ND

4.5 Framerate

The framerate for the TSR model only was, on average, 28 FPS, and the average framerate when the system was constantly detecting a 110-speed limit sign was 21 FPS. The overall framerate of the entire system with lane keeping, TSR, and speed limit recognition was, on average, 19.2 FPS. This is higher than the framerate of the previous project with TSR and PilotNet on the Raspberry Pi 3B, which was 5 FPS [6]. A robot with a high FPS will have high reactivity and is, therefore, more likely to recover from mistakes and drive better as it has more frames of data to either recognise a traffic sign or send a turn or drive command to the robot. Therefore, a framerate of 19.2 FPS can be considered a significant improvement.

4.5.1 Combined AR – Lane Keeping, TSR and Speed Limit Recognition

The total AR for testing the robot on the Carolo test track is summarised in Table 4-9. The test included four traffic signs requiring stopping (stop, give way, kangaroo crossing, pedestrian crossing) and four speed limit signs. These signs were then placed on the track by a second (i.e., unbiased) person. The vehicle's speed is determined by the formula defined in Equation (6). The AR results are shown in Table 4-9, where traffic signs and speed limits are considered track segments and failure to detect these signs in the context of the circuit is deemed an intervention in equation (2). Images of the track with traffic signs are shown in Figure 4-16 and Figure 4-17. The test was done in the outside left lane at night and the turns added with the traffic signs resulted in 20 track segments per lap.

$$Vehicle\ speed\ (mm/s) = 80 + (speed\ limit) - 2.5\ \times (predicted\ angle\ speed) \quad (6)$$

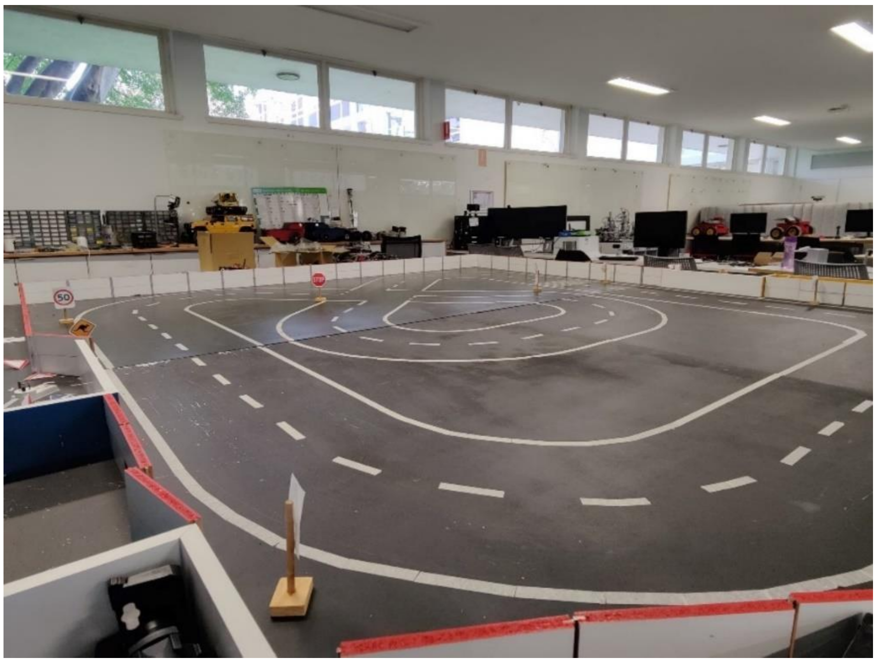


Figure 4-16 Test track with traffic signs, angle 1

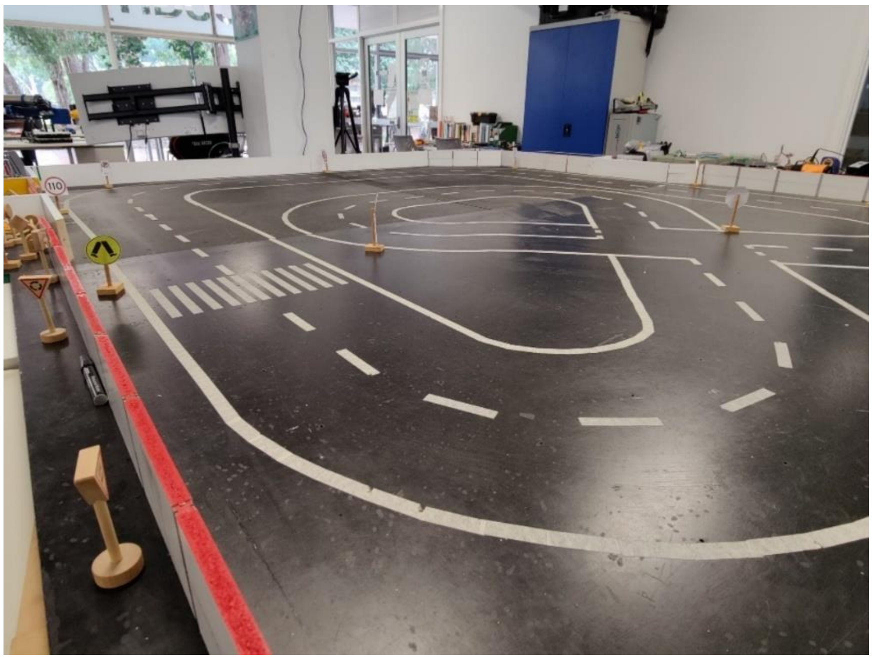


Figure 4-17 Test track with traffic signs, angle 2

Table 4-9 Total AR for PilotNet, TSR, and speed limit detection on Carolo Cup track

Track Segments	Lap 1 Interventions	Lap 2 Interventions	Lap 3 Interventions	Total Interventions	AR
20	3	2	4	9	85%

The testing showed that the overall autonomy achieved was 85%. 7 of 9 interventions were due to traffic signs not being detected. These errors were primarily due to the angle and distance the robot approached the traffic sign causing the traffic sign to be out of range of the camera or out of the camera's FOV. A wide-angled camera (such as a Fisheye lens camera) or multiple cameras could be used to solve this issue.

5 Conclusions

The research conducted for this project demonstrated the following:

1. Replacing the Raspberry Pi with the Jetson Nano resulted in a significant improvement in the overall system speed. The overall system, which included PilotNet for lane-keeping, MobileNet-SSD for traffic sign recognition, and a CNN for determining speed limits, ran at an average of 19.2 FPS. This is a significant improvement compared to the previous UWA project, which ran at an average of 5 FPS. The robot with the Jetson Nano and EyeBot I/O board will provide an improved hardware platform for future research.
2. The Literature Review identified that, unlike Europe and America, there are currently no publicly available Australian traffic sign datasets. As a result, a dataset, and a model to detect and recognise Australian traffic signs was successfully developed as part of this project. The model achieved an average precision of 0.634 which was slightly lower than the average precision before Transfer Learning. This, however, was also lower than the average precision recently achieved by researchers at the Auckland University of Technology, who scored 0.901 and 0.977 on New Zealand traffic signs. This area requires further work to understand why the New Zealand team achieved a better result and to apply this to future TSR systems developed at UWA. In addition, the TSR system could be used on the UWA shuttle bus, which will start testing on public roads soon.
3. As defined by equation (2), the overall autonomy rate achieved was 85%. Most interventions were caused by the TSR not recognising certain traffic signs due to not

being in the camera’s view when at a close enough distance (i.e., due to turning). Further work in this area is required to improve the system by addressing its shortcomings.

5.1 Future Work

The following section discusses potential for future work.

5.1.1 Traffic Sign Recognition

The following area should be considered for improving the TSR system:

1. Review the New Zealand team’s work [48] concerning their TSR system and determine whether there are improvements in their work that can be used and adapted to the UWA robot.
2. Use the AS1744 series of fonts (i.e., the same font as the Australian traffic signs) to create a dataset by making 1000s of slightly varied digits (from 0-9) in this font.
3. Stereo cameras such as the Oak-D cameras could be considered as an alternative to a single front-facing camera. They are a middle ground between computer vision and LIDAR as the left, and right cameras can be used to gather depth information in an image. It is especially compelling as this can be used in conjunction with TSR to determine the distance traffic signs and other objects are from the robot, as shown in Figure 5-1.
4. Develop a TSR system that can recognise electronic speed limits, for example in Figure 5-2.

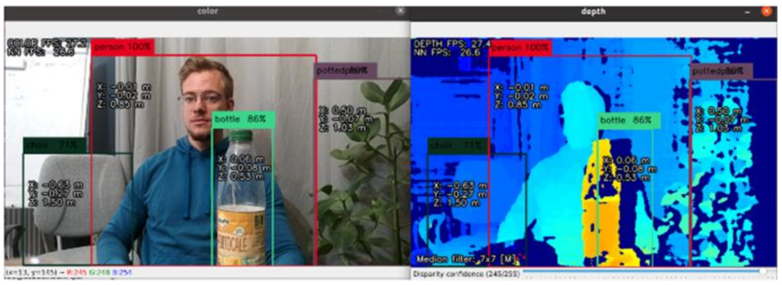


Figure 5-1 Luxonis object detection example integrated with stereo camera to allow for detections to have their position relative to the camera estimated [66]

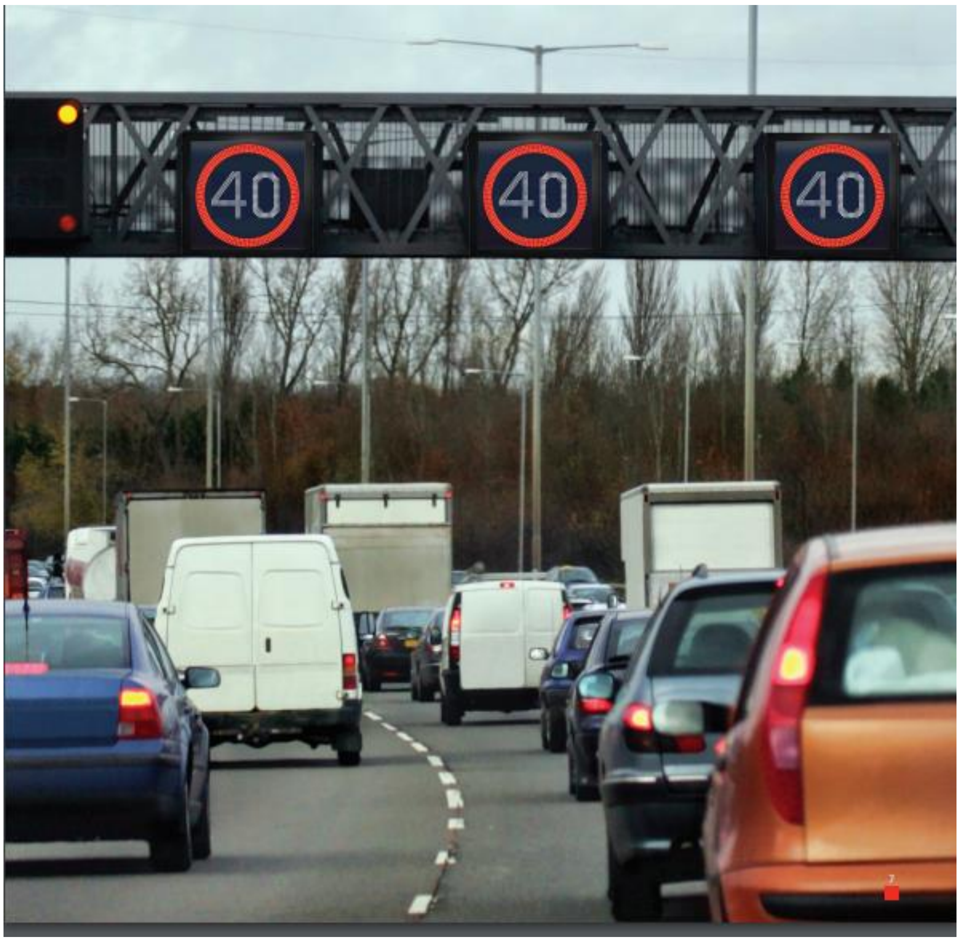


Figure 5-2 Electronic speed limit example [67]

5.1.2 Lane Keeping, Predicted Trajectory. Intersection Logic and Pedestrian Crossing

NVIDIA has adapted its PilotNet model to predict a car's trajectory rather than its steering angle [68]. This has several advantages, including the ability to plan trajectories and therefore be able to avoid obstacles dynamically. Future work could also involve intersection control, as shown in Figure 5-3, whereby autonomous driving could decide to navigate left, straight, or right. Another task could be to implement a pedestrian tracking feature such that the robot, when it stops at a pedestrian crossing, can safely cross when it detects no pedestrians are currently crossing, as shown in Figure 5-4 and as implemented in Team KitCar's video demonstration [10].

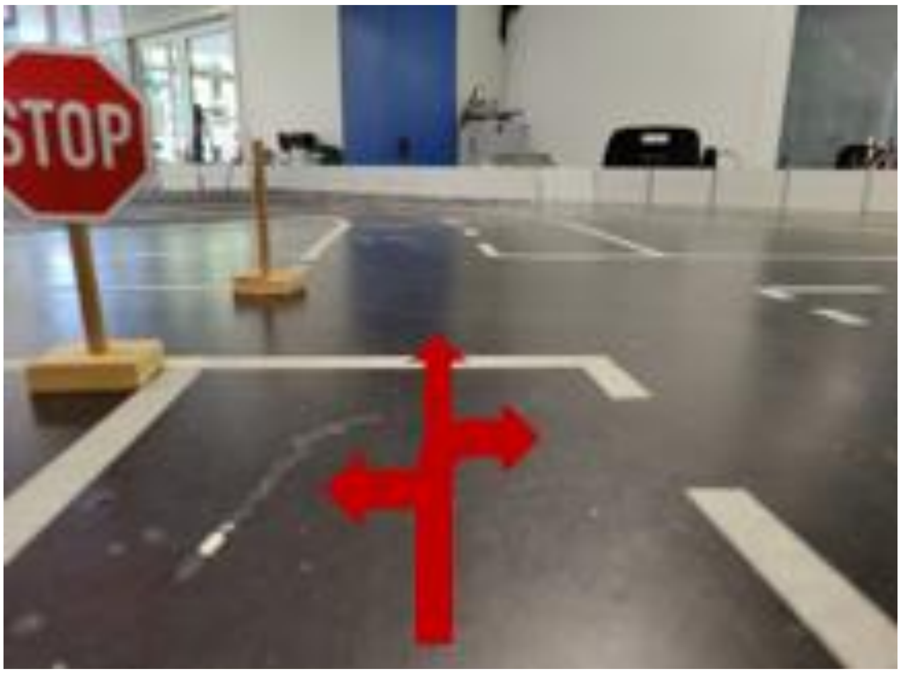


Figure 5-3 Intersection logic

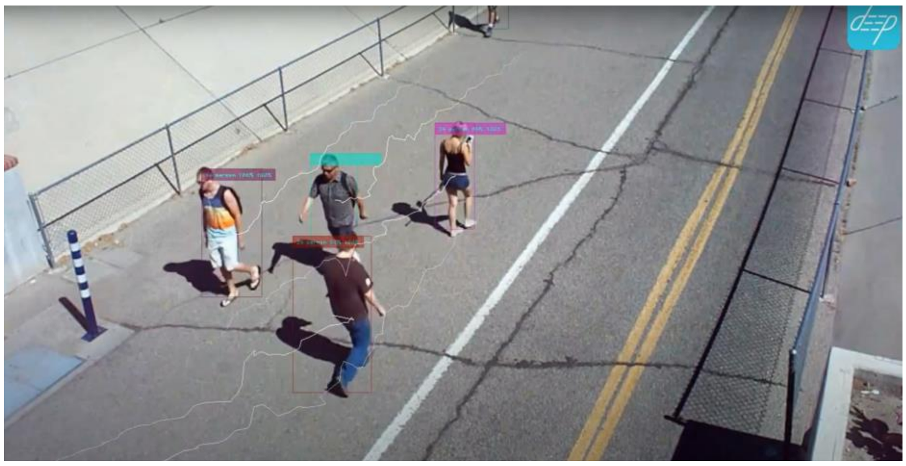
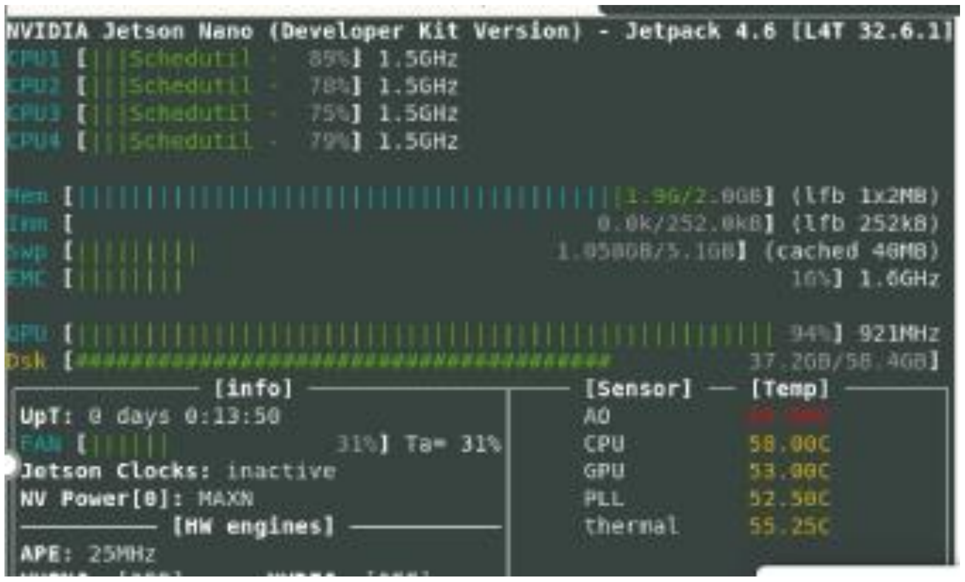


Figure 5-4 Pedestrian tracking for a pedestrian crossing

5.1.3 Hardware Considerations

When the three DNN models are running, approximately 3GB of memory is used as shown Figure 5-5. All 2GB of memory on the Jetson Nano is filled with 1GB of swap space being used. Furthermore, due to the lack of USB ports and lack of onboard WIFI, which causes one USB port to be used by a WIFI adaptor, there is no touchscreen LCD screen on the robot. Both issues could be addressed by upgrading to the Jetson Nano (4GB) version which has 2GB more memory and one extra USB port.



```
NVIDIA Jetson Nano (Developer Kit Version) - Jetpack 4.6 [L4T 32.6.1]
CPU1 [|||Schedutil -   85%] 1.5GHz
CPU2 [|||Schedutil -   78%] 1.5GHz
CPU3 [|||Schedutil -   75%] 1.5GHz
CPU4 [|||Schedutil -   79%] 1.5GHz

Mem [||||||||||||||||||||||||||||||||||||||||||1.9G/2.0GB] (lfb 1x2MB)
Imm [                                   0.0k/252.0kB] (lfb 252kB)
Swp [|||||||||                        1.058GB/5.1GB] (cached 40MB)
EMC [||||||||                                    16%] 1.6GHz

GPU [||||||||||||||||||||||||||||||||||||||||||||||| 94%] 921MHz
Dsk [########################################          37.2GB/58.4GB]
┌──────────── [info] ─────────────┬────── [Sensor] ── [Temp] ─────┐
│UpT: 0 days 0:13:50              │     AO          [illegible]    │
│FAN [||||||           31%] Ta= 31%│     CPU         58.00C         │
│Jetson Clocks: inactive          │     GPU         53.00C         │
│NV Power[0]: MAXN                │     PLL         52.50C         │
│──────── [HW engines] ───────────│     thermal     55.25C         │
│APE: 25MHz                       │                                │
```

Figure 5-5 Screenshot of the resources used when the main script is running

6 References

1. Engineers, S.o.A., *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehichles*. 2021.
2. Hussain, R., J. Lee, and S. Zeadally, *Autonomous Cars: Social and Economic Implications.* IT professional, 2018. **20**(6): p. 70-77.
3. Parasuraman, R. and V. Riley, *Humans and Automation: Use, Misuse, Disuse, Abuse.* Human factors, 1997. **39**(2): p. 230-253.
4. Fung, B., *The driver who died in a Tesla crash using Autopilot ignored at least 7 safety warnings.* The Washington post, 2017.
5. *A Tesla driver is charged in a crash involving Autopilot that killed 2 people*. 2022, The Associated Press.
6. Burleigh, N., J. King, and T. Braunl, *Deep Learning for Autonomous Driving*. 2019, IEEE. p. 1-8.
7. *Carolo-Master-Cup@Home Regulations 2021*. 2021 [cited 2021 10/9/2021]; Available from: https://www.tu-braunschweig.de/fileadmin/Redaktionsgruppen/Institute_Fakultaet_5/Carolo-Cup/Master-Cup_Regulations_210120.pdf.
8. King, J., *SIFT-like Keypoint Cluster-Based Traffic Sign Recognition with Deep Learning*, in *School of Electrical, Electronic and Computer Engineering*. 2019, University of Western Australia.
9. Burleigh, N., *Autonomous Driving on a Model Vehcile Lane Detection and Control*, in *School of Electrical, Electronic and Computer Engineering*. 2019, University of Western Australia. p. 37.
10. *Carolo-Master-Cup@Home: Team KITCar*. 2021; Available from: https://www.youtube.com/watch?v=6_i5glNFQeY.
11. *Our Cars*. 2021; Available from: https://kitcar-team.de/auto.php.
12. Wege, F., *Domain Adaptation and Meta-Learning for End-to-End Autonomous Driving*, in *School of Electrical, Electronic and Computer Engineering*. 2020, University of Western Australia. p. 84.
13. Braunl, T., *EyeBot: a family of autonomous mobile robots*. 1999, IEEE. p. 645-649 vol.2.
14. Bräunl, T., et al. *EyeBot 7 User Guide*. 2018 [cited 2021 17/8/2021]; Available from: https://robotics.ee.uwa.edu.au/eyebot/EyeBot7-UserGuide.pdf.
15. Bräunl, T. *EyeSim VR - EyeBot Mobile Robot Simulator*. [cited 2021 30/8/2021]; Available from: https://robotics.ee.uwa.edu.au/eyesim/.
16. Braylon, A. *Tesla could beat Waymo in the race to self-driving future*. 2019 [cited 2021 11/9/2021]; Available from: https://www.wheelsjoint.com/tesla-could-beat-waymo-in-the-race-to-self-driving-future/.
17. Bellan, R. *Waymo is expanding its driverless program in Phoenix*. 2022; Available from: https://techcrunch.com/2022/05/18/waymo-is-expanding-its-driverless-program-in-phoenix/.
18. Slovick, M. *World's First Level 3 Self-Driving Production Car Now Available in Japan*. 2021 [cited 2021 14/9/2021]; Available from: https://www.electronicdesign.com/markets/automotive/article/21158656/electronic-design-worlds-first-level-3-selfdriving-production-car-now-available-in-japan.
19. Galvani, M., *History and future of driver assistance.* IEEE instrumentation & measurement magazine, 2019. **22**(1): p. 11-16.

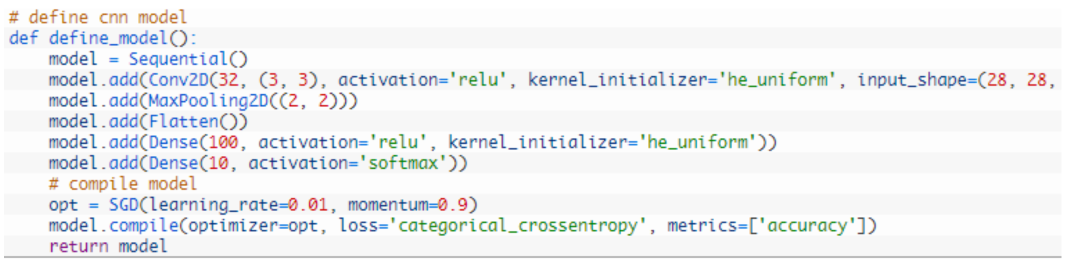
20. Waymo. *Waymo Driver*. Available from: https://waymo.com/waymo-driver/.
21. Jeffcock, P. *What's the Difference Between AI, Machine Learning, and Deep Learning*. 2018 [cited 2021 10/9/2021]; Available from: https://blogs.oracle.com/bigdata/post/whatx27s-the-difference-between-ai-machine-learning-and-deep-learning#:~:text=Machine%20learning%20is%20a%20subset,to%20solve%20more%20complex%20problems.
22. MathWorks. *Introducing Deep Learning with MATLAB.* Available from: https://au.mathworks.com/campaigns/offers/next/deep-learning-ebook.html.
23. Bräunl, T., *Embedded robotics : mobile robot design and applications with embedded systems*. 3rd ed. 2008, Berlin: Springer.
24. Sinha, R.K., R. Pandey, and R. Pattnaik, *Deep Learning For Computer Vision Tasks: A review.* 2018.
25. Sarkar, D., R. Bali, and T. Ghosh, *Hands-On Transfer Learning with Python : Implement Advanced Deep Learning and Neural Network Models Using TensorFlow and Keras*. 2018, Birmingham: Packt Publishing, Limited.
26. TensorFlow. *TensorFlow Lite*. Available from: https://www.tensorflow.org/lite/guide#:~:text=TensorFlow%20Lite%20is%20a%20set,%2C%20embedded%2C%20and%20edge%20devices.
27. Bojarski, M., et al., *End to End Learning for Self-Driving Cars.* 2016.
28. Developer, A. *How single-shot detector (SSD) works?* ; Available from: https://developers.arcgis.com/python/guide/how-ssd-works/.
29. Liu, W., et al., *SSD: Single Shot MultiBox Detector*. 2016, Springer International Publishing: Cham. p. 21-37.
30. Howard, A.G., et al., *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications.* 2017.
31. Franklin, D. *Hello AI World*. 2021 [cited 2021 October 8]; Available from: https://github.com/dusty-nv/jetson-inference/blob/master/docs/pytorch-ssd.md.
32. PyTorch. *From Research to Production*. Available from: https://pytorch.org/.
33. *Onnx*. 2022; Available from: https://github.com/onnx/onnx.
34. NVIDIA. *NVIDIA TensorRT*. [cited 2022 January 30]; Available from: https://developer.nvidia.com/tensorrt#:~:text=TensorRT%2C%20built%20on%20the%20NVIDIA,high%20performance%20computing%2C%20and%20graphics.
35. Pan, S.J. and Q. Yang, *A Survey on Transfer Learning.* IEEE transactions on knowledge and data engineering, 2010. **22**(10): p. 1345-1359.
36. Tobin, J., et al., *Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World.* 2017.
37. Stallkamp, J., et al., *Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition.* Neural networks, 2012. **32**: p. 323-332.
38. Houben, S., et al. *Detection of traffic signs in real-world images: The German traffic sign detection benchmark*. IEEE.
39. Shakhuro V.I, K.A.S., *Russian Traffic Sign Images Dataset.* Computer Optics, 2016. **2**(40): p. 294-300.
40. Larsson, F. and M. Felsberg. *Using Fourier Descriptors and Spatial Models for Traffic Sign Recognition*. in *Image Analysis*. 2011. Berlin, Heidelberg: Springer Berlin Heidelberg.
41. Timofte, R. *BelgiumTS Dataset* 2014; Available from: https://btsd.ethz.ch/shareddata/.

42. Mogelmose, A., M.M. Trivedi, and T.B. Moeslund, *Vision-Based Traffic Sign Detection and Analysis for Intelligent Driver Assistance Systems: Perspectives and Survey.* IEEE transactions on intelligent transportation systems, 2012. **13**(4): p. 1484-1497.
43. Ertler, C., et al., *The Mapillary Traffic Sign Dataset for Detection and Classification on a Global Scale.* 2019.
44. Milford Michael, G.S., Mount James, Keir Andrew, *How Automated Vehicles Will Interact with Road Infrastructure Now and in the Future* 2020.
45. *The German Traffic Sign Recognition Benchmark*. 2010; Available from: https://benchmark.ini.rub.de/gtsrb_news.html.
46. NVIDIA. *SignNet*. Available from: https://docs.nvidia.com/drive/archive/driveworks-3.0/signnet_mainsection.html.
47. Alarcon, N. *DRIVE Labs: Classifying Traffic Signs and Traffic Lights with SignNet and LightNet DNNs*. 2019; Available from: https://developer.nvidia.com/blog/drive-labs-signnet-and-lighnet-dnns/#:~:text=To%20achieve%20this%20goal%2C%20the,detection%20and%20traffic%20sign%20detection.
48. Zhu, Y. and W.Q. Yan, *Traffic sign recognition based on deep learning.* Multimedia tools and applications, 2022. **81**(13): p. 17779-17791.
49. Roper, Y.R., Mark; Chakich, Zoran; McGill, William; Nanayakkara, Vinnuka; Young, David; Whale, Russell, *Implications of Traffic Sign Recognition (TSR) Systems for Road Operators*. 2018.
50. Mazda. *Traffic Sign Recognition (TSR)*. [cited 2022 May]; Available from: https://www.mazda.com.au/imagination-drives-us/safety-traffic-sign-recognition/.
51. RS. *NVIDIA Nvidia Jetson Nano 2GB Developer Kit without Wi-Fi Development Kit*. 2022; Available from: https://au.rs-online.com/web/p/processor-development-tools/2049969.
52. Components, R. *Raspberry Pi 4 B 4GB*. 2022; Available from: https://au.rs-online.com/web/p/raspberry-pi/1822096.
53. RS. *NVIDIA Jetson Nano Development Kit 945-13450-0000-100*. 2022; Available from: https://au.rs-online.com/web/p/processor-development-tools/1999831.
54. Altronics. *USB C Power Delivery Battery Bank 10000mAh*. Available from: https://www.altronics.com.au/p/d0507b-power-delivery-usb-c-qc3-battery-bank-10000mah/.
55. Ryan, A., *End-to-End Learning for Autonomous Driving Robots*, in *School of Electrical, Electronic and Computer Engineering*. 2020, University of Western Australia. p. 42.
56. *LabelImg*. Available from: https://github.com/tzutalin/labelImg.
57. Li, D., *The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web].* IEEE signal processing magazine, 2012. **29**(6): p. 141-142.
58. TensorFlow. *mnist*. Available from: https://www.tensorflow.org/datasets/catalog/mnist.
59. Khandelwal, R. *Evaluating performance of an object detection model*. 2020; Available from: https://towardsdatascience.com/evaluating-performance-of-an-object-detection-model-137a349c517b.

60. *Evaluating Object Detection Models: Guide to Performance Metrics*. 2019; Available from: https://manalelaidouni.github.io/Evaluating-Object-Detection-Models-Guide-to-Performance-Metrics.html.
61. Upadhyay, A. *Precision/Recall Tradeoff*. 2020; Available from: https://medium.com/analytics-vidhya/precision-recall-tradeoff-79e892d43134.
62. *pytorch-ssd*. 2020; Available from: https://github.com/qfgaohao/pytorch-ssd.
63. LeCun, Y.C., Corinna; Burges, Christopher. *The MNIST Database of handwritten digits*. 1998; Available from: http://yann.lecun.com/exdb/mnist/.
64. Lecun, Y., et al., *Gradient-based learning applied to document recognition.* Proceedings of the IEEE, 1998. **86**(11): p. 2278-2324.
65. Cireşan, D.a.M., Ueli and Schmidhuber, Juergen, *Multi-column Deep Neural Networks for Image Classification.* arXiv, 2012.
66. Luxonis. *DepthAI's Documentation*. 2020; Available from: https://docs.luxonis.com/en/latest/.
67. Aldridge. *ESLS/VSLS Electronic/Variable Speed Limit Signs*. Available from: https://www.trafficltd.com.au/wp-content/uploads/2022/03/ESLS-VSLS-brochure-V4.1-copy.pdf.
68. Bojarski, M., et al., *The NVIDIA PilotNet Experiments.* 2020.
69. Brownlee, J. *How to Develop a CNN for MNIST Handwritten Digit Classification*. 2019; Available from: https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-from-scratch-for-mnist-handwritten-digit-classification/#:~:text=Learning%20with%20Anaconda-,MNIST%20Handwritten%20Digit%20Classification%20Dataset,digits%20between%200%20and%209.
70. Franklin, D. *Jetson Nano Brings AI Computing to Everyone*. 2019; Available from: https://developer.nvidia.com/blog/jetson-nano-ai-computing/.

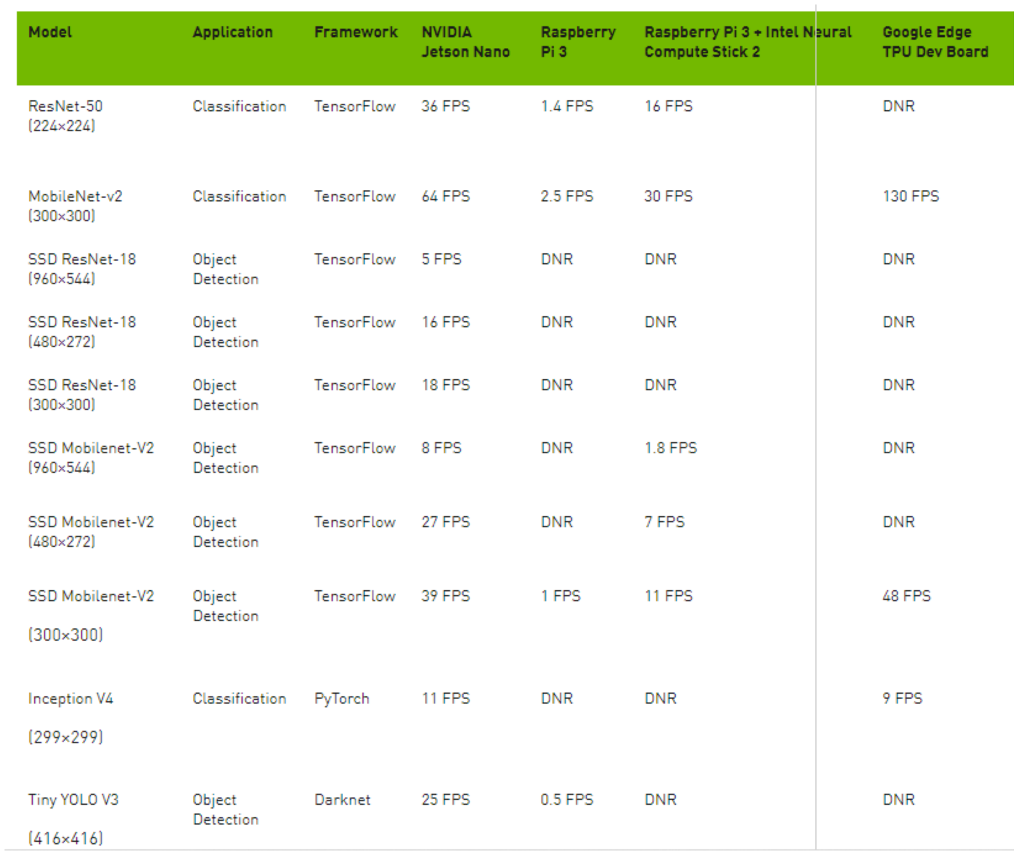
7 Appendix

7.1 CNN model code using TensorFlow 2 and Keras API [69].



```
# define cnn model
def define_model():
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', input_shape=(28, 28,
    model.add(MaxPooling2D((2, 2)))
    model.add(Flatten())
    model.add(Dense(100, activation='relu', kernel_initializer='he_uniform'))
    model.add(Dense(10, activation='softmax'))
    # compile model
    opt = SGD(learning_rate=0.01, momentum=0.9)
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

7.2 NVIDIA Deep Learning Inference Performance Tests across popular SBC [70].



Model	Application	Framework	NVIDIA Jetson Nano	Raspberry Pi 3	Raspberry Pi 3 + Intel Neural Compute Stick 2	Google Edge TPU Dev Board
ResNet-50 (224×224)	Classification	TensorFlow	36 FPS	1.4 FPS	16 FPS	DNR
MobileNet-v2 (300×300)	Classification	TensorFlow	64 FPS	2.5 FPS	30 FPS	130 FPS
SSD ResNet-18 (960×544)	Object Detection	TensorFlow	5 FPS	DNR	DNR	DNR
SSD ResNet-18 (480×272)	Object Detection	TensorFlow	16 FPS	DNR	DNR	DNR
SSD ResNet-18 (300×300)	Object Detection	TensorFlow	18 FPS	DNR	DNR	DNR
SSD Mobilenet-V2 (960×544)	Object Detection	TensorFlow	8 FPS	DNR	1.8 FPS	DNR
SSD Mobilenet-V2 (480×272)	Object Detection	TensorFlow	27 FPS	DNR	7 FPS	DNR
SSD Mobilenet-V2 (300×300)	Object Detection	TensorFlow	39 FPS	1 FPS	11 FPS	48 FPS
Inception V4 (299×299)	Classification	PyTorch	11 FPS	DNR	DNR	9 FPS
Tiny YOLO V3 (416×416)	Object Detection	Darknet	25 FPS	0.5 FPS	DNR	DNR