Car Driving Without Cameras

By

JONES MABEA AGWATA

Department of Computer Science

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of MASTER OF SCIENCE in the Faculty of Engineering.

SEPTEMBER 2018 | CSMSC-18

Executive Summary

The need for robust object detection in 3D point clouds has greatly increased with the ongoing push for autonomous vehicles (AVs). Most of these systems use Light Detection and Ranging (LiDAR), cameras or a combination of both in order to perform object detection. LiDAR presents objects as point clouds in a 3D space thus offering critical shape information of objects in view. However, this representation is sparse. As a result, LiDAR-based detection performs poorly as compared to multimodal methods that have helped overcome this. Nonetheless, multimodal methods are often complex to set up and synchronise with the cost of components running into thousands of pounds. In light of this, reducing the number of sensors while still maintaining or even improving the accuracy has become a topic of interest by industry players who are developing AVs. With the cost of LiDAR declining following recent improvements in solid-state LiDAR technology, dropping the number cameras in favour of LiDAR has become more plausible as cameras suffer from drawbacks such as visibility issues in extreme weather.

This project will be 65% type I (Software Development) and 35% type II (Theoretical) and will explore the use of multimodal methods and LiDAR only methods in different contexts with the aim of reducing the number of sensors in AVs.

The main contributions and achievements of this project are:

- A cross-discipline review of issues affecting AV and their adaptation. (Chapter 2)
- Developing an image and point-cloud context classifier. (Chapter 3)
- Open sourcing and annotating sample frames from Smart Internet Lab dataset that can be used for training and developing object detection models. (Chapter 3)
- Demonstrated impact of Tensorflow graph parameters and FLOPs on GPU memory usage and Temperature. (Chapter 4)
- Establishing a bottleneck in LiDAR only object detection models that have only been validated on a single dataset.(Chapter 4)

DEDICATION AND ACKNOWLEDGEMENTS

First and foremost, I'd like to acknowledge God's hand throughout the Masters course and the wisdom provided by Him in various aspects. I'm extremely grateful to my supervisors, Dr. Luis Gonzalez and Dr. Raul Santos-Rodriguez who have offered indispensable insight that has been truly beneficial to completing this project. In addition, I'd like to thank Dr. Sion Hannuna for providing me with necessary resources and advice. Of course, how can I forget the companionship of coffee throughout the draining coding and debugging periods.

Most importantly, this project is dedicated to my parents, Zablon Mabea and Petronilla Mabea, who have consistently and selflessly supported and encouraged me throughout my life.

Table of Contents

List of Tables

List of Figures

Figure **Page**

CHAPTER 1

INTRODUCTION

Accelerated by recent advancements in technology, the prospect of Autonomous Vehicles (AVs) driving in public roads is becoming more and more a reality. As this is an emerging field, there are numerous variations of implementations by different companies. Arguably, a key characteristic of these implementations is a large number of perception sensors including cameras, radars and Light detection ranging sensors(LiDAR) that are necessary for mapping the environment around the vehicle in order to safely navigate. Most of these sensors are quite expensive and fusing their input and processing it requires powerful processing units such as GPUs. This process consumes a lot of power and generates a lot of heat. In an effort to reduce the cost of such systems, companies are exploring different ways to reduce the number of sensors while still achieving a high level of navigational accuracy and safety.

Compared to humans who are able to easily adapt to different situations, AVs need to be trained and tested in different possible situations. Different regions in the world pose varying challenges, such as poor visibility in Scandinavian countries due to heavy snow, and on the other hand heavily populated regions with heavy traffic such as New York pose challenges such as high occlusion of objects. An effective way of testing and evaluating AV systems is using a scenario-based approach. This has been the basis of Software-in-Loop and Hardware-in-Loop tests that test these systems in a virtual real-time environment to understand how they would react. This is done by modelling conditions and evaluating them on the corresponding software and hardware [46]. As a stepping stone, the scenario-based approach can be used to develop AV systems that are tailor-made for certain

environments/regions and later on generalised as the systems become more adaptable to different environments. There are some companies and institutions that adopt this approach and have deployed autonomous prototypes in different areas. For example, Uber currently has some AVs in Steel City, Pittsburgh whereas Waymo has deployed some in Phoenix, Arizona. Other companies are still developing their AVs within test tracks[24]. Using these approaches such companies are able to collect a large amount of data for testing and evaluating future models. However, this data is not always released to the public therefore limiting the variation and amount of data that is available to other parties working on AVs.

Currently, a lot of cutting-edge research on AV systems relies on a small subset of data. Most research is based on the KITTI [13] dataset that was collected to aid the development of AV systems by providing the necessary data to develop models to achieve different task. Such tasks included, stereo, optical, depth, odometry, sceneflow and object detection. As a result it is highly favoured in the development of novel computer vision algorithm. In equal light, relying on few datasets creates a bottleneck in terms of validation as these algorithms may end up performing sub par when evaluated on different datasets or even worse in a real-time setting.

Following this argument, a viable scenario that could be beneficial to explore is how different sensors and corresponding object detection models perform in urban and non-urban contexts. In doing so, companies developing AVs can fit different sensors in AVs that work in different contexts with the aim of maximising their performance in that context while reducing the cost by removing redundant sensors. The most reliable and accurate combination of sensors in terms of and reliability is camera and LiDAR. However, there has been increased interest in developing LiDAR only systems that are able to exploit highly accurate depth information from LiDAR such as Doxel[1] that is an autonomous robot that uses LiDAR to monitor construction sites. As such it may be possible to develop LiDAR only units that can be used in a specific context.

1.1 Aims and Objectives

Following the motivations in the presented discussion , the performance of single sensor ()LiDAR only) and multimodal(LiDAR and Camera) models in different contexts will be investigated with the aim of reducing the number of sensors in AVs. To achieve this aim, the following objectives will need to be fulfilled:

1. **Detect and characterise the context of images and point clouds.**

2. **Evaluate the performance of single sensor and multimodal models in different contexts.**
3. **Validate performance of the single sensor model on a custom dataset**

1.2 Deliverables

The deliverables are categorized in terms of the related objectives:

1. **Context Detection**
    - **Image and LiDAR Context Classifier** - Available as a software deliverable in the form of Python scripts and Jupyter interactive notebooks. This will include pre-trained models to reproduce the results.
2. **Custom single sensor and multimodal object detection models**
    - **LiDAR only object detection model** modified to run on custom context datasets and external datasets. This will include tools to preprocess and convert external datasets. Available as Python scripts and Jupyter interactive notebooks.
    - **LiDAR+Camera object detection model** modified to run on custom context datasets. This will include tools to preprocess these datasets. Available as Python scripts and Jupyter interactive notebooks.
3. **Single sensor validation**
    - Point Cloud dataset obtained from the University of Bristol Smart Internet Lab working on connected AVs. Tools to convert the dataset into compatible input for LiDAR only object detection model and annotated sample frames for training and testing will be provided.
4. **Report containing the following discussions:**
    - A review of related research and implementations tackling object detection in AVs.
    - Evaluation and analysis of context classifiers and object detection models in different contexts.
    - Validation results of single sensor model using external dataset from the Smart Internet Lab on connected and autonomous vehicles.

1.3 Report structure

The report will be split into different chapters each discussing various stages encountered while working on this project.

- Chapter 2 begins by discussing the background on the development of AVs, the different components in AVs, current implementations in the industry and finally related research on the research that has been undertaken in the field of object detection.
- Chapter 3 details the project execution by discussing the tools, methods and the criteria for various model evaluations.
- In Chapter 4, the results from various evaluations will be presented. This will be complemented with an analysis of the various evaluation metrics.
- Finally, the concluding chapter will establish whether the objectives were achieved and a justification of their implication. Furthermore, a discussion on how this work can be further improved in future implementations will be presented. Lastly, a personal reflection on lessons learnt during the execution of the project will be detailed.

CHAPTER 2

BACKGROUND

Despite recent interest in autonomous vehicles(AVs), the idea of autonomous vehicles(AVs) is not new. In fact, there had been attempts to autonomously navigate using cameras as early as 1990. Dan Pomerleau, a leading researcher from Carnegie Mellon, had developed a neural network, ALVINN(Autonomous Land Vehicle in Neural Network)[29], that could steer a van using camera input and avoid obstacles using a laser range finder . His system used artificial neural networks to categorise road and non-road segments and detect objects in them[28].This was one of the pioneering research in this field. Despite the success, a major limitation was the low computational power of computers at that time that led to long training times for artificial neural networks. Due to this limitation, interest in AVs slowly waned off as interest in the internet revolution peaked.

It was not until around 2004 when the AV landscape started seeing major development. At around this period, the US congress had passed regulation to replace 33% of military vehicles with AVs[18]. Despite major military investment, there was not much progress and therefore Defense Advanced Research Projects Agency(DARPA) opted to revolutionise the process by opening up the challenge to the public. In February 2003, DARPA hosted the Grand Challenge with a prize of 1$ million dollars to develop an AV that could finish a 142 mile course that was set up in Mojave Desert to simulate harsh difficult conditions. Of 160 competitors, there was no winner. In the 2005 Grand Challenge, Stanford University had the fastest car to complete the course and consequently won the challenge. Subsequent challenges were held and more AVs were able to complete

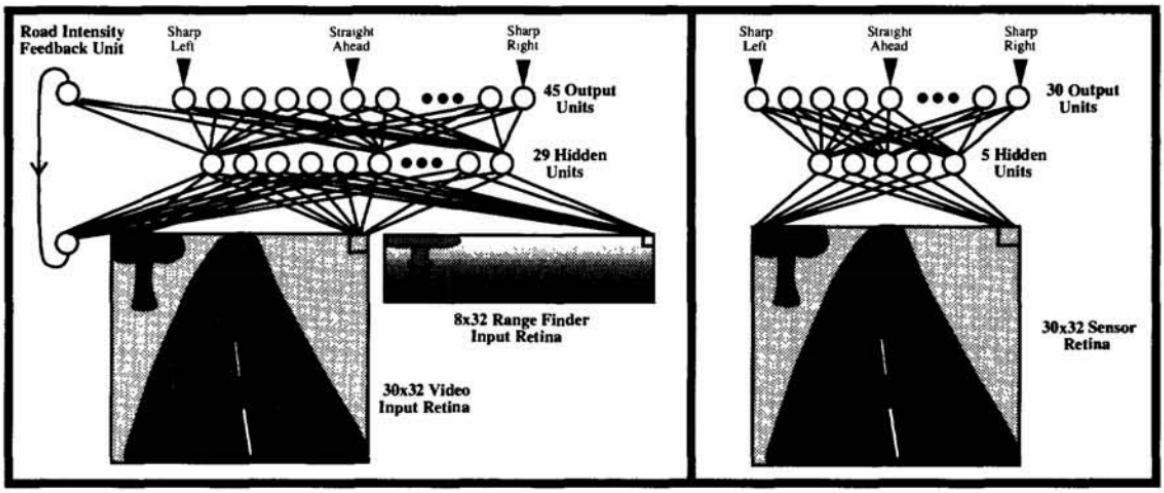


Figure 21: ALVINN architectures. Left: With camera and laser range finder. Right: camera only. Source: [28]

the course aided by advances in AI methods. Following these challenges, more research was undertaken to create more reliable and accurate sensors. Notably, Velodyne came about as a result of the Grand Challenge and is currently the leading supplier of LiDAR units.

With different types of autonomous systems being deployed in vehicles nowadays, the Society of Automotive Engineers(SAE) defined five levels of autonomy to differentiate them:

SAE level	**Description**
0	No AV control systems. An example is blind spot indicators.
1	Basic driver assistance built into vehicle design. An example is the cruise control function.
2	Basic AV control systems. Driver required to monitor the environment and take back control if need be. A good example is the Tesla Autpilot.
3	AVs can safely navigate and drive within mapped environment. Driver required to monitor the environment and take back control if need be.
4	Highly autonomous control capable of handling most conditions but the driver has the option to take control. Renault ISymbioz is an example of a level 4 AV.
5	Completely autonomous with zero human intervention. None yet are available.

2.1 Components of an AV

- **LiDAR** - LiDAR provides highly detailed 3D information about the environment around the vehicle and objects in it. LiDAR operates by sending out pulses of lasers and recording the reflections of the pulses from objects. By comparing this with the time taken for the lasers to be reflected(time of flight) and their direction, the distance of these objects can be calculated and mapped in a point cloud. LiDARs have proven to be extremely important for safe navigation due to

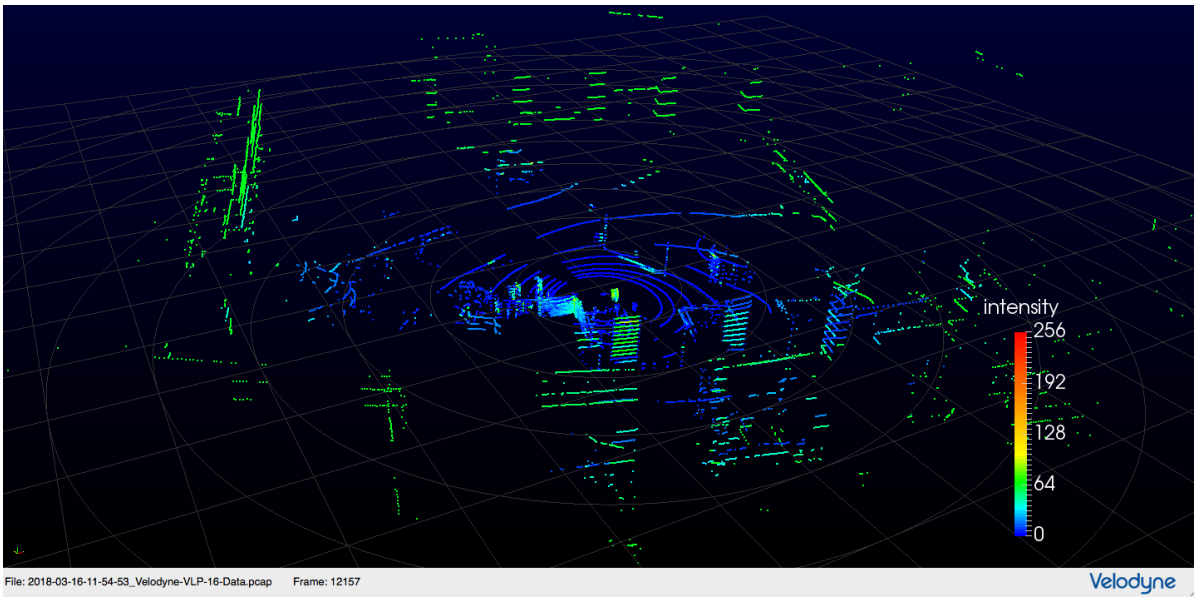


Figure 21: LiDAR Point Cloud

their high level of accuracy. However due to their low angle resolution they are not able to capture contours of objects. In addition they do not provide texture information and therefore classifying objects can be quite difficult.

- **Cameras** - Grayscale and color cameras are normally used to detect and classify different objects, markings and signs on the road. Grayscale images tend to offer better edge information than color images. This can be useful for better edge detection. Point clouds can also be obtained by using two cameras to obtain a stereo image that has depth information or by combining a camera and infrared laser to obtain RGB-D images such as in the Kinect sensor. Less commonly, thermal cameras have been used in AVs to overcome weaknesses of color and grayscale cameras in poor lighting conditions.

Figure 22: Velodyne LiDAR family. From left: Velodyne HDL-64E , HDL-32E , VLP-16

- **Position Estimators** - Position estimators are a group of sensors used for navigation of the vehicle. These include GPS systems, odometers and gryometers.

- **Distance Sensors** - Distance sensors such as radars and sonars are important for gauging the distance of objects on the road. Radars are the most commonly used distance sensors and they function by transmitting radio waves and recording the reflected radio waves from objects. As compared to cameras and LiDARs, radars work well in a variety of low visibility scenarios such as in heavy snow/rainfall. However, the reflectivity of these radio waves depends on the nature of objects, their size, absorption characteristics and the transmitting power. As such, it is may not be effective for detecting objects with low absorption characteristics such as pedestrians and animals.

- **Processing Unit** - In order to process all the data in real-time from sensors on the vehicle, AVs require powerful processing units. Most ML/AI algorithms used for detecting and identifying objects from LiDAR and camera data demand large amounts of processing power. This is achieved through the use of CPUs,

GPUs, Field Programmable Gate Arrays(FPGA)[4], Application Specific Integrated Circuits(ASICs)[39] or combinations with each other.

These components are used in different operations. In most AV implementations, these operations can be grouped into three categories.

Perception - This is the first step which involves processing the input from the sensors. In this mode tasks such as object detection and tracking, lane detection, traffic sign detection and recognition are performed.

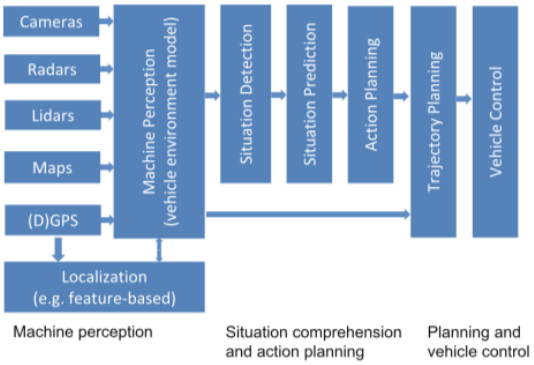


Figure 23: Detailed operation pipeline. Source:[10]

Planning - This is the next step after detection and recognition tasks are performed . In this stage route and trajectory planning algorithms are run to plan how the vehicle should navigate in the immediate environment as well as a route to a target location. These algorithms are required to handle complex situations to ensure safety of the passengers and other road users.

Control - This stage involves the execution of plans created in the planning stage. This stage is crucial as the actuators involved in steering and movement have to be able to accurately follow the planned routes and actions. At this stage the trajectories and movement of other road users and objects have to be calculated in order to anticipate and avoid any accidents.

2.2 Industrial Approaches

Despite improvements in traffic safety over the years, over 2.2% of deaths globally are caused by road accidents. Of these accidents, 94% are as a result of human error. A major motivation for the development of AVs is their potential to drastically reduce this by removing the human element. However due to the multidisciplinary nature of AVs there are various factors that need to be considered before public adoption of AV

2.2.1 Legal and Economic Concerns

Modelling Risk

Gasser et. al[12], define safety as the lack of unreasonable risks where risk is the product of the probability of an accident and its severity. From this, it is clear that there are different degrees of severity each with a various weighting; a fatality is more costly than damage to property. In their study they were able to establish that AVs will have the potential of preventing some accidents, however, they will introduce some risk as a result of their automation. They further went on to establish the relationship with severity as seen in figure 21. From this relationship they speculated that there is no uniform distribution of risk and in addition, there may be a reduction in severe accidents but an increase in lesser severe accidents. This relationship can be expressed as a ratio that could express the probability of acceptance of a said autonomous system.

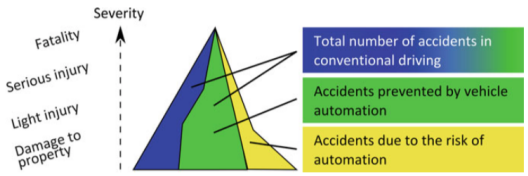


Figure 21: Diagram illustrating distribution between types of risk and severity. Source:[12]

$$P_{acceptance} = \frac{R_{added\,risk}}{R_{avoided\,risk}}$$

However, this ratio does not necessarily guarantee approval as there are some cases of certain systems such as nuclear plants that the risks outweigh the benefits but due to socio-economic and/or political factors tend to get accepted. Nonetheless, it forms a pragmatic ratio that could be used as a rule of thumb.

Liability

In terms of safety, a lot of arguments against fully autonomous vehicles have been based on trolley problems whereby the AV has to choose between two fatal scenarios, either killing the passenger or by standers in the case of an accident. However this argument has quite a few vital flaws as outlined in [11]. It was established that most tragic choices are avoidable by undertaking enough preventive measures. This has been the basis for most AV systems whereby they practise minimal risk policy. Waymo define an "operational design domain" whereby their AVs can safely operate. This includes time of day, speed ranges, traffic laws and regulations among other factors[45]. This ensures that the vehicle is able to maintain minimal risk operation. Coupled with the sensors

continually monitoring vehicles and objects around them, they are able to continuously preempt potentially dangerous situations and avoid them. In addition, backup systems are installed to increase the level of redundancy in case some systems fail.

As mentioned before, there is a risk associated with the introduction of these AVs that may result in accidents. As a result, shifting from human agents to an autonomous agent will result in creation of a large number of legal questions in terms of liability and safety regulations. Currently there is no legal framework that cover the use of fully autonomous vehicles. Manufacturers developing AVs will need to develop robust and reliable systems to a satisfactory degree so as not to open themselves to product liability suits. A leading cause of product liability suits is accidents when using their automotative products. In addition, this can further extend to the use of a defective design on the vehicle even when there is no accident by the concept of strict liability. Another consideration that can lead to litigation is the breach of warranty. According to [47], a warranty is "an affirmation or promise concerning a product or its performance, features, or characteristics, such as those concerning the safety of a product". In light of this, companies developing AVs need to consider the following as defined in [47]:

- Design systems that are highly accurate and able to recognize other road users and hazards.
- The systems should be able to use enough data from sensors at real-time and process to a high precision.
- Develop sufficient crash avoidance strategies.
- Ensure adequate information security measures are undertaken.

Nonetheless, it is also important to create a collective scientific examination process that investigates how the traffic system that is used currently may contribute to accidents [12]. This has been the basis for developing traffic systems to be used in case of 100% AV adoptions such as [42].

2.2.2 Hardware Considerations

In light of the legal requirements, [20] established a few metrics that could be used in order to achieve viable AV systems.

Performance

At the moment there are no clear regulations as to how fast the perception, planning and control pipeline should be. However according to research by Brown et al. [3], humans

take around 600 ms to respond and brake when expecting an interruption, however this figure shoots to 850 ms when an unexpected situation arises. In addition, Newell et al [26], established that the fastest human response time is between 100-150ms. These figures can be used as a baseline while developing a pipeline. In this pipeline, there are two factors to consider, namely the frame rate (frequency of the data from sensors) and the processing latency (time taken to process data). In consideration of this, [20] estimated that an AV system should be able to run processed within 100ms to achieve better performance than a human driver.

Storage

AVs should be able to store maps of different areas with fine granularity for accuracy during localisation. As a result, the storage of these maps can run into tens of Terabytes. Despite recent advances in cloud technology and the emergence of 5G connectivity that is significantly fast. Downloading these maps would take significant time and would also render the car unusable in case of no internet connectivity. As such, the vehicles should have enough storage to store these maps locally.

2.2.2.1 Power

Most of the major industry players have moved to electric-vehicles for their AV systems. Depending on the equipment and sensor configurations used in these systems, the power usage can range from 500 watts to 1.5 kilowatts. Total power consumption includes the computing, storage and cooling overheads. Given that the AVs have limited battery capacity, heavy power consumption by these systems can lead to poor driving ranges hence making the cars less viable. From their evaluation, a system with 1 CPUs and 3GPUs operating fully can lead to a 6% reduction in the driving range and 11.5% if the whole system is considered. Furthermore, they established that cooling such a system would result in a 77% overhead of the total power usage. Therefore, the configuration of these systems has to be carefully considered to ensure a reasonable driving range.

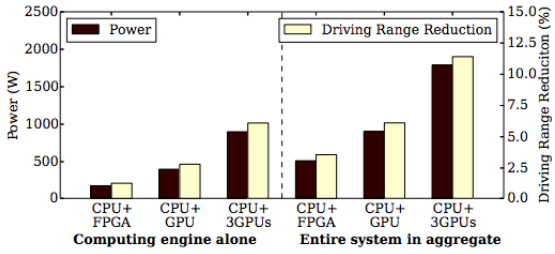


Figure 21: Graph illustrating reduction of power range against the processing unit configurations. Source:[20]

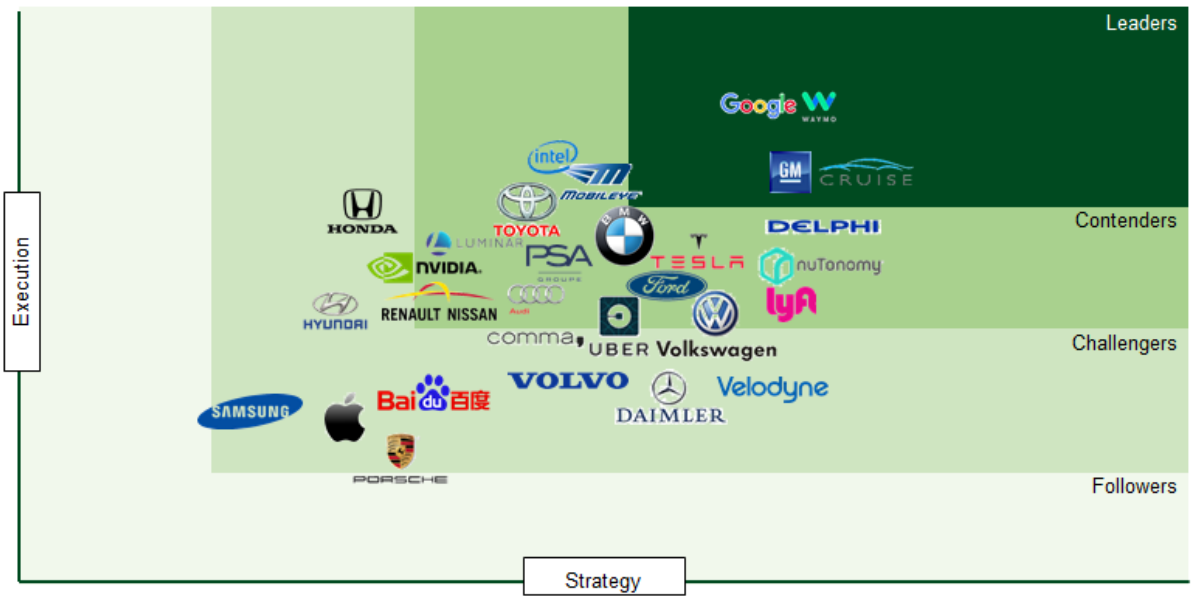


Figure 21: AV Leaderboard by Navigant Research. Source:[34]

Thermal

Processing components in the AV systems such as CPUs and GPUs require a significant amount of energy to cool. This is necessary to ensure that they operate within their recommended thermal operating range. Failure to do so could result in the failure of these systems. Depending on the type of component, the range is normally below 80°Celcius. Additional cooling systems are therefore necessary to ensure optimal operating temperatures.

2.2.3 Perception

Following the discussion above, the next subsections will review how different industry players are implementing their AV systems with special focus to the issue of perception. Figure 21 illustrates the current progress of various companies working on AVs.

Camera Only

Mobileye[25], is one of the top contenders in the development of camera only AVs. They were recently acquired by Intel and believe that AVs should be able to accurately and safely navigate using cameras only given the fact that humans are able to do so with vision only. Previously, MobilEye was a supplier of vision systems for Advanced Driver Assistance Systems and had a partnership with Tesla to supply their vision systems prior

being acquired by Intel. Given their extensive background in developing these vision systems, the partnership with Intel aims to develop a complete autonomous driving package[16].

Camera and Radar

Another major contender that is using the camera and radar configuration is Tesla. Elon Musk, the founder, believes that LiDAR is not necessary for AV perception following a similar argument as MobilEye. However, this argument was dispelled following a recent crash whereby due to bright sunlight, a Tesla vehicle on 'autopilot' mode was unable to detect a white vehicle thus resulting into an accident leading to loss of life. The invariance of LiDAR to different lighting conditions could have helped in detecting the vehicle.

2.2.4 LiDAR, Camera and Radar

This configuration is used by most of the leading industry players such as Uber, Waymo and GM Cruise. This configuration has proven to be robust and accurate. However, in order to process the amount of fused data, they require large amounts of computational power which in turn may lead to increased power usage.

2.2.5 LiDAR

LiDAR-only systems have not been explored by major companies working on AVs for driving on public roads. However,various autonomous robots have been developed to assist in other industries such as the Neato house cleaning robot. Doxel, a start up aiming to solve problems in the construction industry has also created an autonomous robot that autonomously scans and captures the progress of different construction sites and further process them using neural networks.

2.3 Object Detection

Object detection has seen a rapid increase in innovation over the past few years. From the early days of the sliding window method used in classic computer vision algorithms to the development of deep learning algorithms that are able to learn complex features

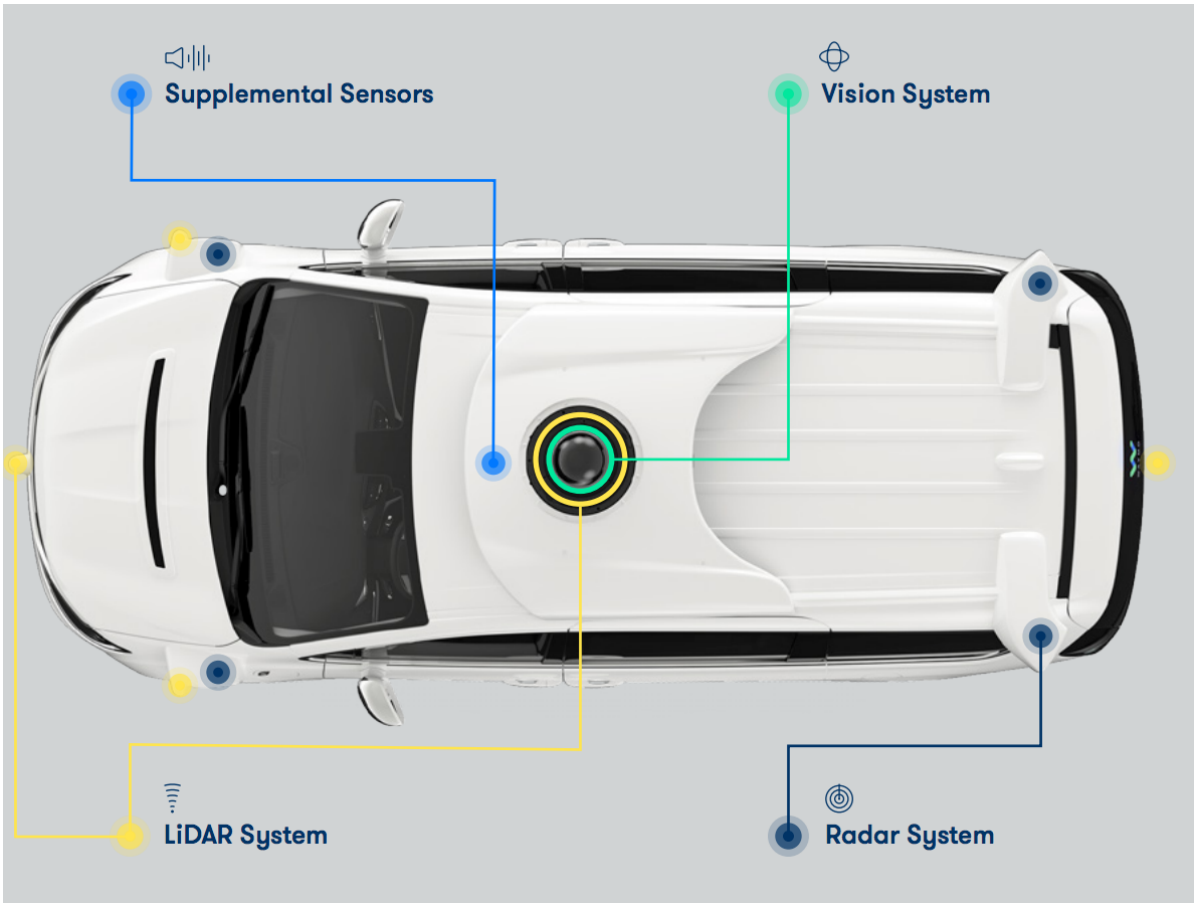


Figure 21: Waymo's Self Driving Car Configuration, Source:[45]

automatically. In this section, I will explore different object detections methods that have been used and their underlying methods of operation.

2.3.1 Object Detection using Classic Computer Vision Methods

Classic computer vision methods have advanced over the years from simple algorithms such as SIFT[22] and SURF[2] which use descriptors to detect interesting points in images for object detection to much higher level representations such as the Histogram of Gradient(HOG)[8] or Haar Features[43] that were used in classifiers applied on a sliding window.

However, these methods have been unreliably slow and not as accurate. In addition these features are only able to model 2D data. 3D methods such as Intrinsic Shape Signatures(ISS)[49], NARF[41] and Uniform Sampling have since been developed but are lacking in terms speed. Furthermore, these feature detectors have to be hand-crafted

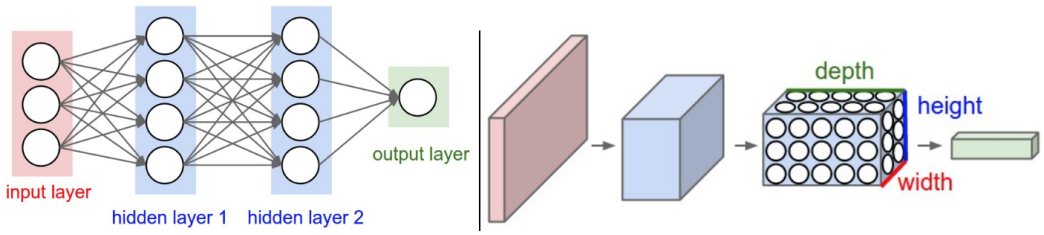
manually which is a tedious process. As a result, they are rarely used in real-time AV applications due to their speed and accuracy.

2.3.2 Object Detection using Deep Learning

The advancement of neural networks has offset the reliance of classical methods of object detection by allowing for features to be automatically learnt by the network without the need for manual feature extraction. This is attributed to the networks having numerous number of deep layers that are able to capture these features.

Convolutional Neural Networks(CNN)

The idea of CNNs has been there for over two decades now and are inspired by the idea of receptive fields in the primary visual cortex. Currently, they are widely used in image object detection tasks and various CNN architectures have been developed. As compared to regular neural networks which have all neurons connected to each other between hidden layers, neurons in CNNs are connected to a small region in the layer before it(receptive field) and have three dimensions(width,height and depth). By doing so this reduces the number of parameters to be tuned during back-propagation as well as stored.



Left: A regular 3-layer Neural Network. Right: A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels).

Figure 21: NN vs CNN Source: `https://cs231n.github.io/convolutional-networks`

Common architectures implement five main layers.

- **Input Layer** Forms the first stage of the CNN where the input data is fed in.
- **Convolution Layer** This layer calculates the output of neurons connected to a receptive field. The size of the output is determined by the following equation.

$A_{out} = (A_{in} - K + 2P)/S + 1$
where A_{in} is the input size, A_{out} is the output size, K is the kernel size, P is the padding size and S is the stride of the kernel.

- **Activation Layer** In this layer the output of the convolution is passed through an activation function such as sigmoid, ReLU or TanH.
- **Pooling Layer** This layer downsamples the output of the activation layer along the width and height dimensions.
- **Fully connected layer** Finally, this layer computes the probability scores for different classes by connecting all neurons from the pooling layer.

Region Proposal Networks(RPN)

R-CNN[15] brought forth the idea of region-based CNNs with the aim of increasing the accuracy of CNNs. By using selective search, regions of interest(ROI) in input images are proposed. These regions are then resized are then fed through a CNN to generate a bounding box and classification of the image. Finally a classifier and regressor are run on the output. However, as this method requires a forward pass of the CNN for each proposal, its speed was quite slow. In addition, the CNN, classifier and regressor have to be trained and tuned separately. Fast R-CNN[14] iterated on R-CNN to further improve the speed and accuracy by using ROIPooling whereby instead of running each proposal in a CNN, regions that were overlapping were pooled and fed through the CNN thus sharing the computation. Furthermore, this model was end-to-end and thus combined the classifier and regressor by adding a softmax layer and linear regression layer after the CNN. Faster R-CNN[33] sought to remove the bottleneck imposed by the using selective search to propose regions. This was solved by using a sliding window with various anchor boxes on features maps generated by a forward pass of a CNN. Bounding boxes on certain regions of the image could be generated. These regions are then passed onto Fast R-CNN to classify the objects.

Depending on the input, these two network architectures have formed the back bone of most 2D and 3D object detection applications.

Monocular Vision Monocular vision is derived from a single camera. This representation is 2D and lacks depth information. Mono3D by Chen et al [5] is a state of the art object detection model that uses monocular images where they were fed through a CNN and the resulting 2D object proposals were extruded into 3D bounding boxes by performing segmentation.

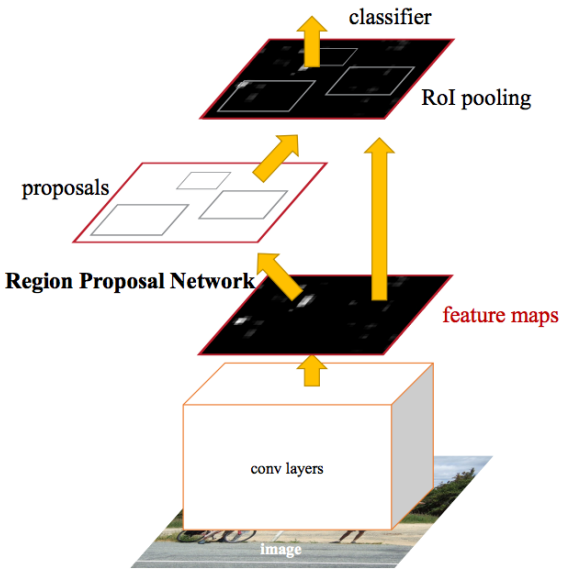


Figure 22: Faster R-CNN overview. Source:[33]

Stereo Vision Stereo vision involves calibrating two cameras intrinsically and extrinsically by matching similar points in images from both cameras. Using the resulting stereo image, it is possible to estimate the depth of objects in the image. However the accuracy of this depends on a few factors such as the resolution, camera quality, lens distortion and the calibration algorithm. 3DOP [6] is one of the state-of-the-art models that use stereo images in CNNs to detect objects.

LiDAR+Camera By fusing input from a monocular image and point clouds, it is possible to extract depth information from the images. Chen et al further reiterated their monocular approach into MV3D[7] a multiview 3D object detection network. In their work, they were able to fuse LiDAR and camera input to create a bird's eye view of the surrounding environment and further perform object detection of features.

LiDAR Point cloud object detection is particularly challenging as established 3D object detection methods for images cannot be applied to point clouds. Point clouds are a set of geometric points in a euclidean space. They are unordered in nature and this presents a particular problem to deep learning methods as they need to be invariant to permutations of the input set. Depending on how they manipulate the point clouds, current point cloud based object detection models can be split into three groups:

1. **Direct Manipulation**

Pointnet[31] developed by Qi et al was able to overcome this challenge on small sets of point clouds. By using point clouds as direct input into Recurrent Neural Networks[23] They were able to create a global point cloud signature that could classify and segment 3D objects in the point cloud. They further improved this model into Pointnet++[32] which recursively applied PointNet on nested partitions of the point clouds using a hierarchical neural network. In doing so they were able to capture the local structures of the point clouds as a result of their metric nature and thus achieve better performance than PointNet. However this approach has high memory and computational requirements. For both of these publications, the source code was released for public use. Consequently, they have formed a fundamental foundation for further development of point cloud object detection methods such as F-PointNet[30].

2. **Voxel Based**
   By dividing the pointclouds into 3D voxel grids, points within these voxels can be sampled to perform feature extraction. VoxelNet [50] is a state of the art architecture that uses this method to extract these features through Voxel Feature Extraction layer. The extracted features are then fed through a RPN to generate bounding boxes and classify objects within them.

3. **2D-Based**
   This involves projecting the pointcloud into a 2D image plane coordinate system which can then be trained on existing image based CNNs such as VGG-16. VeloFCN [19] was one of the first methods to implement this approach using a fully convolutional network. However, VeloFCN lacked bird eye and front view representations.

Network Model	**Mode**	**Code**	**Inference Time**
VoxelNet	*LiDAR*	Unnoficial	***100ms***
AVOD	*LiDAR+Image*	Yes	***100ms***
MV3D	*LiDAR+Image*	Unnoficial	360ms
MONO3D	*Mono Image*	Yes	4.2s
3DOP	*Stereo Image*	Yes	3s

Table 21: Modes of Operation of Common Architectures

CHAPTER 3

IMPLEMENTATION

Following the prior discussion, this section seeks to address three challenges affecting affecting object detection models for AVs. This will be necessary to achieve the objectives outlined. Firstly, the necessary datasets to be used for the different evaluation tasks will be outlined . The second task will be to determine whether it is possible to detect the context of images/point clouds from their features. Upon successful classification of images/pointclouds into their respective contexts, the performance of multimodal and LiDAR in different contexts will be assessed. Finally, the performance of the LiDAR only model will be validated using an external dataset to determine if indeed LiDAR based object detection models are extrapolable to different datasets.

3.1 Datasets

3.1.1 Context Classification

1. **Cityscapes** - Cityscapes is a large-scale dataset used to develop pixel and instance level semantic segmentation models. It is composed of a scenes from 50 different cities captured using stereo cameras. In 5000 scenes, the images have been annotated on a pixel-level. 20000 scenes have also been coarsely annotated. In addition to this, a benchmark suite to evaluate segmentation models is available.

2. **KITTI Object Detection**- KITTI dataset was collected with the aim of encouraging the development of computer vision and robotic algorithms for AVs.The

data was captured by a car was fitted with a number of sensors including a high resolution greyscale and colour cameras, Velodyne HDL-64 LiDAR and a GPS/IMU inertial navigation system. The vehicle was driven around a city, rural areas and highways. This dataset is available as part of a benchmark suite for various AV object detection models.The dataset consists of a total of 14,999 images and corresponding point clouds with 7,481 as labelled training files and 7,518 being unlabelled testing files. The testing set are used for evaluating results through the official test server. However, the test server is not available for general public use.

3.1.2 LiDAR Performance Evaluation

1. **University of Bristol Smart Internet Lab(SIL)** - This dataset was captured from a Velodyne VLP-16 LiDAR on stationary vehicle positioned at the Millennium Square in Bristol as part of the connected and autonomous vehicles project. Available as LiDAR network capture files(.pcap).

2. **KITTI Object Detection**

3.2 Can We Automatically Detect The Context in Driving Scenes?

Determining the context from images and point clouds where no prior information about their location nor context has been provided is quite difficult. This is often the case in a lab setting where the data is obtained from public database. This was indeed the case with images and point clouds from the KITTI dataset that did contain such information and since the frames were discontinuous as they shuffled into training and testing sets, it was difficult to infer the context using the sequence of prior frames. Notably, this was also evident in the Cityscapes dataset. From this observation, it was necessary to create a context detector for images and furthermore point clouds that can be used in a lab setting.

To begin with, I visually classified 3749 images from the KITTI dataset to be used as training and test data for the context detector. Of these 1029 were non-urban and the remaining 2720 urban. The context of the image scenes were determined using the following criteria defined by the UK Department for Environment, Food and Rural Affairs [9] Urban regions are characterised by:

1. High density of road users often including pedestrians, cyclists and vehicles.
2. Presence of numerous buildings that tend to be large and multi-story.
3. Presence of multiple transport facilities such as trams.

On the other hand, non-urban regions are characterised by:

1. Low/medium density of road users mostly vehicles.
2. Few or no buildings, mostly isolated dwellings.
3. Long stretches of motorways surrounded by vegetation.

Following this, the classified data was split on a 80:20 ratio to be used as the training and test set respectively.

Figure 31: Urban scene

Figure 32: Non-Urban scene

3.2.1 Image Context Detection

Images provide a rich representation of a scene that can be exploited through image segmentation. Various image segmentation models have been developed to break down images into coherent regions.Semantic segmentation involves classifying these regions into various classes on a pixel level. Classic computer vision techniques for semantic segmentation utilised Texton Forests[38] and Random Forest classifiers [37], however, deep learning techniques have overtaken them in terms of performance. Developed by Google, Deeplab-V3 is one of the best performing open-source models on the Cityscapes semantic segmentation benchmark table.

Implementation Details

Extracting Semantic Histograms Using the Deeplab-V3 network, I performed semantic segmentation on the classified images. For each image, a semantic histogram containing the number of pixels in each semantic class as seen in figure 31 was calculated.

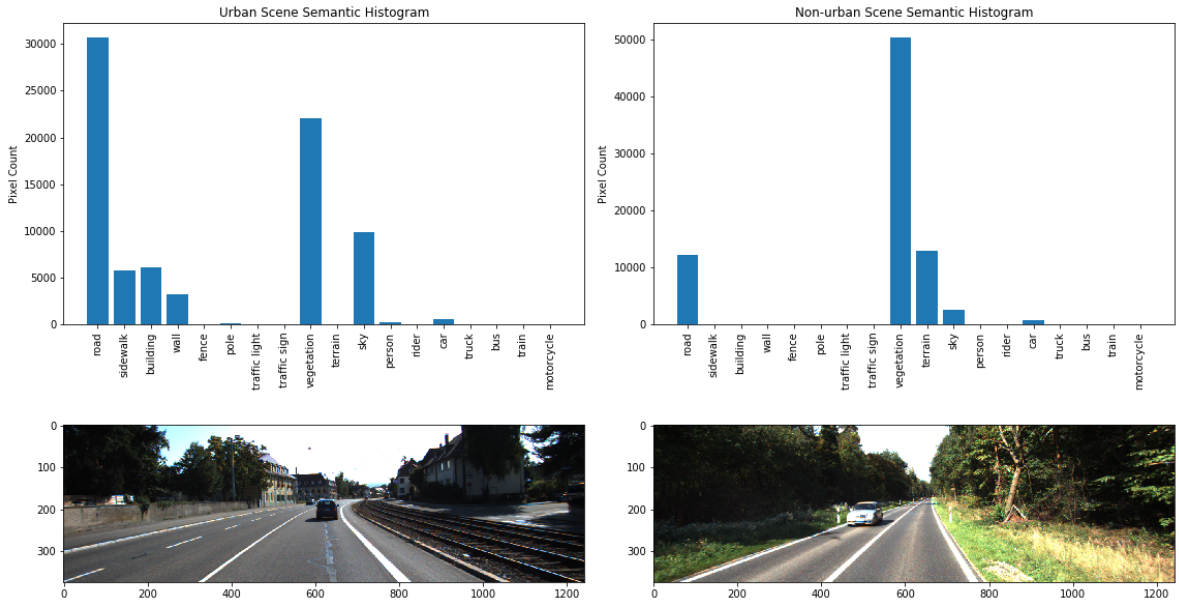


Figure 31: Semantic Histograms

Training - By matching the semantic histograms with the context label of the image, I was able to train a Support Vector Machine(SVM) using the scikit-learn library to classify the image context. Linear and radial basis functions were tested on the SVM to evaluate their performance.

Evaluation - The trained SVM classifier was then used to classify keypoint descriptors that were extracted from the test point clouds. Given that a single point clouds can contain various features, majority voting was used to establish the dominant feature class.

Penalising misclassifications - In a realistic setting, I speculated that it would be more costly to classify urban as non-urban than non-urban. This is because non-urban settings are less dynamic than urban settings. In order to extend this as part of the AV system, it was important to create a penalising factor to the classifier as defined below:

$$R_{urban} = \lambda_{11}P(urban|x) + \lambda_{12}P(non-urban|x)$$
$$R_{non_urban} = \lambda_{21}P(urban|x) + \lambda_{22}P(non-urban|x)$$
$$Context = \begin{cases} \text{Urban}, & \text{if } R_{urban} < R_{non-urban} \\ \text{Non-urban}, & \text{otherwise} \end{cases}$$

where λ is a 2x2 matrix with $\lambda_{11}$and λ_{22} set to 0 and λ_{12} set to 1.6 and λ_{21} set to 1.

3.2.2 PointCloud Context Detection

As compared to point clouds from RGB-D or stereo cameras, point clouds from LiDAR lack rich features. This is due to the sparse nature of the point clouds and as a result, the resulting representation tends to have a large number of 'holes'. Despite the availability of point clouds segmentation models, there are no datasets with annotations of outdoor scenes such as Cityscapes. As such using semantic histograms to characterise scene contexts in this case would not be suitable. A different approach was therefore used by first detecting the key points in the cloud, extracting and classifying features from them and later on matching these features in other point clouds.

1. **Keypoint extraction**
   Similar to classic image processing, key points are salient points that are not affected by any form of transformation or distortion. As such, they are distinctive and repeatable in different points-of-view. They can either be global or local on a given surface with the former being preferred to extract 3D shape information and the latter for 3D object detection. Currently, only a few 3D keypoint detectors have been proposed. That is Intrinsic Shape Signatures(ISS)[49], NARF[41] and Uniform Sampling of point clouds in voxels.

2. **Feature description**
   Keypoints extracted act as features that offer a robust representation of a local or global surface. In order to provide a description of features, a descriptor is used to encode the relevant surrounding of the keypoints. These include include Fast Point Feature Histograms(PFH)[35], Signatures of Histograms of Orientations(SHOT)[36].

3. **Feature Matching**
   Once different feature descriptors have been obtained. They can used for object

detection and segmentation using nearest-neighbour functions. For high dimension data as in the case of point-clouds, randomised kd-trees and FLANN are usually used to achieve decent performance.

3.2.2.1 Implementation Details

The above methods were implemented in Python using PyDriver[27] framework and Point Cloud Library. Point Cloud Library is an open-source project for 2D/3D image and point cloud processing. The library contains various software implementations of keypoint extractors, descriptors and feature matching functions. However, the implementations are written in C++ and the available Python wrappers are only for Python 2.7. PyDriver is a framework contains a suite of object detection, classification and tracking functions. In addition, it contains Cythonized wrappers to the Point Cloud Library functions necessary for keypoint detection, extraction and feature matching and can run on Python 3.x. Code is available at `https://github.com/lpltk/pydriver`

2000 images from the classified context database were used as training samples, and the remaining 1749 as test samples.

Training - Keypoints within the ground truth bounding boxes were extracted using ISS with a salient and NonMaxSuppression(NMS) radius of 0.25. SHOT feature descriptor with a radius of 2 was applied on the keypoints to create feature descriptors. Each feature was assigned with the context class of the point cloud. An AdaBoost classifier was then trained on the feature types.

3.2.3 Implementation Issues

Despite being able to visually classify a large number of images, some images were difficult to classify as they exhibited characteristics from both contexts. Furthermore, due to the fact that the images was not contiguous , I was unable to infer their context by examining preceding image frames. As such, this affected the performance of both classifiers.

Figure 31: Ambiguous scene

3.3 Do Some Object Detection Methods work better in Different Contexts?

Having been able to successfully classify and create a dataset of urban and non-urban images and corresponding point clouds, the next step was evaluating how a LiDAR based and image+LiDAR based object detection model would perform in different contexts. Multimodal methods have been argued to perform better in 3D object detection tasks whereby objects are partially occluded or relatively small. Such characteristics are common in urban areas as compared to non-urban areas which have fewer and more distinct objects. In light of this argument, I assessed whether LiDAR-only models can perform better or at par with multimodal methods in different contexts.

For this task, VoxelNet(LiDAR only) and AVOD(LiDAR+image) proved to be the most suitable candidates. As seen in table 21 both models had the least inference time that would similar inference times, and both were open-source with the only exception being that the VoxelNet implementation was unofficial[1]. Both of these models were written in Python and run on TensorFlow Deep Learning Framework.

3.3.1 VoxelNet

VoxelNet is an end to end point cloud based 3D object detection network that was recently published by researchers at Apple Inc. The network is one of the top performing LiDAR only models on the KITTI benchmark. However, the code was never released and only unofficial implementations are currently available online. Code is available at `https://github.com/qianguih/voxelnet`

[1]Had similar evaluation results as the unreleased original on KITTI benchmark.

3.3.1.1 Architecture Overview

To better understand how it works it is important to understand the architecture. VoxelNet is composed of three fundamental blocks; a feature learning layer, CNNs and finally a RPN.

1. **Feature Learning Layer**
   In this layer, the point clouds are divided into equal 3D voxels. Points within these voxels are then grouped. For the car detection, 35 points are sampled and for pedestrian and cyclist detections, 35 points are sampled from within the voxels. These sampled points are then fed through multiple FCNs that learn and aggregate features from the points. The result is a 4D tensor that represents these features per voxel.

2. **Convolutional Middle Layers**
   The 4D tensor containing these features is then passed through multiple convolutional middle layers. In doing so the different layers learn different features in an expanding receptive fields. This is allows the network to learn complex shape information at different scales.

3. **Region Proposal Network**
   Finally, the feature maps from the convolutional middle layers are fed through a RPN to classify and generate oriented 3D bounding boxes for the objects.

Loss Function

$$L = \alpha \frac{1}{N_{\text{pos}}} \sum_i L_{\text{cls}}(p_i^{\text{pos}}, 1) + \beta \frac{1}{N_{\text{neg}}} \sum_j L_{\text{cls}}(p_j^{\text{neg}}, 0) + \frac{1}{N_{\text{pos}}} \sum_i L_{\text{reg}}(\mathbf{u}_i, \mathbf{u}_i^*)$$

where:

- L_{cls} is the binary cross entropy loss that is calculated for bot positive and negative classes.
- α is a weighting parameter for the positive anchors normalised binary cross entropy loss.
- β is a weighting parameter for the negative anchors normalised binary cross entropy loss.
- L_{reg} is the SmoothL1 regression loss.

3.3.1.2 Implementation Details

Training - To encourage generality, the model was trained on 3732 point clouds from the original KITTI training set. VoxelNet was trained both car and pedestrian classes on two NVIDIA P100 GPUs with a batch size of 4 for 160 epochs with α = 1 and β= 10. Adam optimizer was used with an initial learning rate of 0.01 for the epochs≤80, 0.001 for 80≤epoch≤120 and 0.0001 for epochs≥ 120.

Changes to Loss Function - In an effort to improve performance of pedestrian and cyclist detection, I implemented the focal loss function as defined in [21]. Focal loss was introduced with the aim of solving high foreground-background imbalance in single stage detectors. While running VoxelNet on the cyclist class, there were few positive anchors for cyclists as there were very few frames with cyclists and as such the negative anchors ≫ positive anchors. Therefore, implementing the focal loss would focus on hard negatives.

3.3.2 AVOD

AVOD is a state of the art multimodal network that offers improved performance for small object classification by fusing LiDAR point clouds and images to perform detection. Code is available at `https://github.com/kujason/avod`

3.3.2.1 Architecture Overview

AVOD's architecture is comprised of two distinct stages:

1. **Feature Extractors** To create a BEV map, LiDAR point clouds are projected into the XY plane and discretised to create BEV 2D grid. In each cell a height feature is defined by the maximum height of the points in the cell and it's reflectance defines the intensity feature in five slices between [0,2.5] of the Z axis. For each cell, the normalised point cloud density is calculated using $min(1.0, \frac{log(N+1)}{log16})$ where N is the number of points.

   For the image and BEV map similat feature extractors consists of an encoder and decoder are applied. The encoder is a CNN that creates a downsampled feature map resulting in higher representation. The decoder is a Frustum Point Net that upsamples the feature map to original size resulting in higher resolution. The

output from the image and point cloud feature is fused by passing it through a 3x3 convolution layer.

2. **Multimodal Fusion Region Proposal Network**
   Feature maps from the point cloud and image feature extractors are fused in a RPN to generate proposals. The top proposals are then passed through a second RPN that performs object classification and bounding box regression to generate oriented 3D bounding boxes for the objects.

Implementation Details

Training - Similar to VoxelNet's training method, the model was trained on 3732 point clouds and images from the original KITTI training set. The AVOD-FPN model was trained for both car and pedestrian classes for 120000 steps on two NVIDIA P100 GPUs. The adam optimiser with an exponential decay learning rate was chosen with an initial learning rate of 0.0001 and a decay rate of 0.8 each 30000 steps.

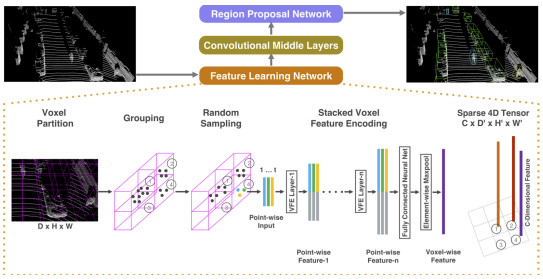


Figure 31: VoxelNet Architecture

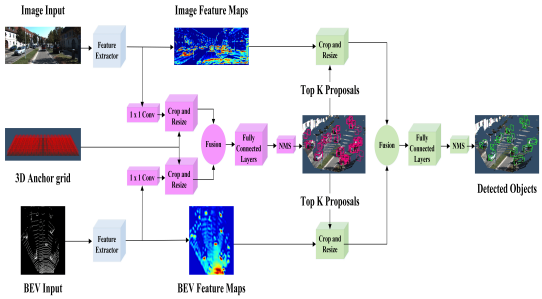


Figure 32: AVOD Architecture

3.3.3 Evaluation

Once both models were trained, they were evaluated on 2058 images/point clouds from the context dataset with equal number of class samples(1029 each). This was necessary to avoid bias as the urban samples were significantly more than the non-urban samples. For each context, each model was run on a single NVIDIA P100 GPU and the results of 3 runs averaged.

3.3.4 Is VoxelNet Performance Valid for Different Datasets?

As compared to image object detection methods that are can perform well regardless of the input image resolution, LiDAR point clouds can vary in different scenes. This

can depend on various factors but is largely determined by the LiDAR sensor and it's frequency in terms of points per second. Coupled with the sparse nature of point clouds, this implies that LiDAR only models may not be able to generalise to input from different LiDAR sensors with different frequencies. I therefore set out to understand how LiDAR sensors with different frequencies affect the performance of VoxelNet In this section I aim to validate the VoxelNet model using the SIL dataset that was obtained from a Velodyne VLP-16 sensor with a lower frequency of points than that of KITTIs Velodyne HDL-64E[2].

Preprocessing

Prior to using the SIL data, I had to preprocess it to the necessary input format. VoxelNet only required four input fields, **XYZ values** and **reflectance** of the points, whereas the dataset was in the form of network capture files(.pcap). To convert them into this format, I exported the capture filed into CSV format for each of the frames. This was done in VeloView using the CSV export function. The result was a CSV file with 13 fields as seen in figure 31

Points_m_XYZ:0	Points_m_XYZ:1	Points_m_XYZ:2	X	Y	Z	intensity	laser_id	azimuth	distance_m	adjustedtime	timestamp	vertical_angle
0.01759156584739685	5.304835319519043	-1.421434164047241	0.01759156507648251	5.304835470033373	-1.421434195703044	1.0	0.0	19.0	5.492	3433080897.0	3433080897.0	-15.0
0.07194533199071884	20.6107349395752	0.3597639203071594	0.07194532898218986	20.61073481953913	0.3597639062981624	20.0	1.0	20.0	20.614	3433080900.0	3433080900.0	1.0
0.02194887027144432	5.988438129425049	-1.382549166679382	0.02194887062991644	5.98843819471877	-1.382549179997394	1.0	2.0	21.0	6.146	3433080902.0	3433080902.0	-13.0
0.02645202539861202	6.88900899887085	-1.339097499847412	0.02645202482367224	6.889008789131341	-1.339097529552591	1.0	4.0	22.0	7.018	3433080907.0	3433080907.0	-11.0
0.3046736121177673	75.89746856689453	6.640221595764416	0.3046736085365985	75.89747013826725	6.640221728458581	63.0	5.0	23.0	76.188	3433080909.0	3433080909.0	5.0
0.03370168805122375	8.045639038085938	-1.274315118789673	0.033701689426747 4	8.045638637731509	-1.274315152217721	1.0	6.0	24.0	8.146	3433080911.0	3433080911.0	-9.0
0.3095385730266571	70.94056701660156	8.710489273071289	0.3095385870564042	70.94056833290217	8.71048945053951	63.0	7.0	25.0	71.474	3433080913.0	3433080913.0	7.0
0.04397721216082573	9.691121101379395	-1.189932227134705	0.04397721326242784	9.691120843322894	-1.18993226900786	1.0	8.0	26.0	9.764	3433080916.0	3433080916.0	-7.0
0.3213215470314026	70.80862426757812	11.2150993347168	0.3213215462835961	70.80862345723331	11.21509966766423	63.0	9.0	26.0	71.69200000000001	3433080918.0	3433080918.0	9.0
0.05691538378596306	12.07773017883301	-1.056676268577576	0.05691538501887421	12.07773041569957	-1.056676225072608	3.0	10.0	27.0	12.124	3433080920.0	3433080920.0	-5.0
0.08184240758419037	16.16960144042969	-0.8474237911885376	0.0818424105905633	16.16960230547166	-0.8474238034857464	2.0	12.0	29.0	16.192	3433080925.0	3433080925.0	-3.0

Figure 31: SIL Data Frame CSV

Out of the 13 fields, I extracted the XYZ values and intensity(reflectance) of the points and saved them as Numpy binary files. This resulted in a smaller file sizes(from around 4MB to 70KB) that are also quicker to read than normal CSV files.

Finally, I modified VoxelNet's configuration file to include the specifications for the VLP-16 LiDAR sensor. This was essential as the network was initially trained on KITTI point clouds that were obtained from a Velodyne HDL-64E with different specifications as seen in table 31. These specifications are necessary for manipulating the point clouds and therefore directly affect the performance of the network. The modified entries were the angular and vertical resolution, maximum and minimum XYZ values and the height of the VLP-16 LiDAR sensor on the vehicle. Given that there was no image data for

[2]See table 31

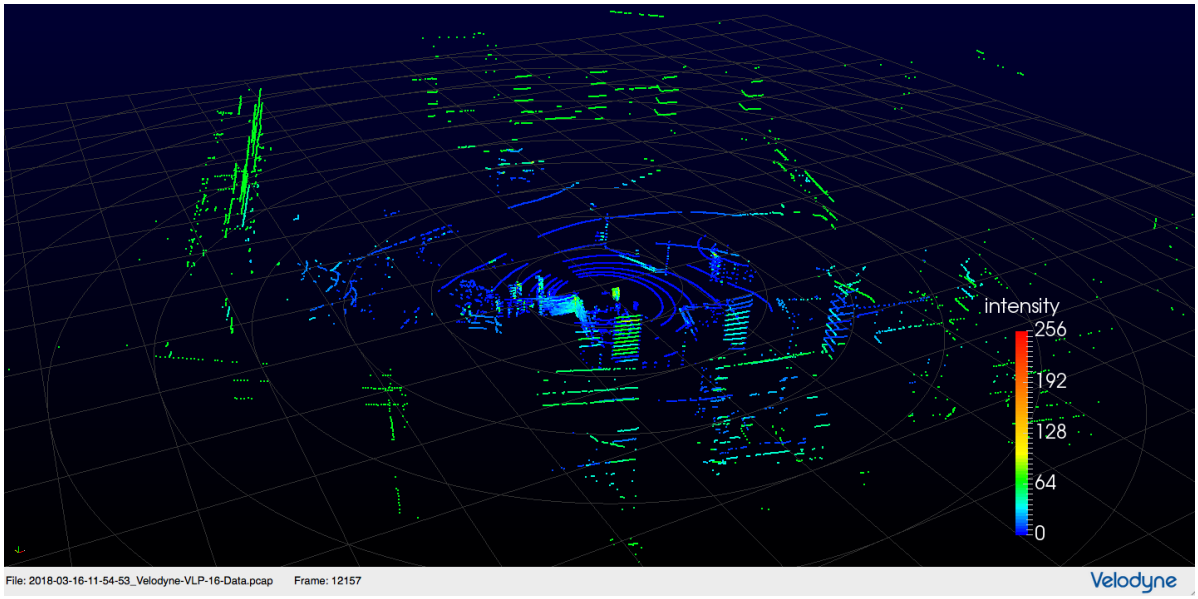


Figure 32: Veloview Frame of SIL data

visualisation, the image size in the configuration file was set to 0 to prevent unnecessary tensor memory allocation for images that are used for visualisation.

LiDAR	Hor FOV	Ver FOV	Range	Angular Resolution	Points/second	Channels
HDL-64E	360°	26.9°	120m	~0.4°	~2.2 Million	64
VLP-16	360°	± 15°	100m	0.1°	600,000	16

Table 31: Velodyne HDL-64E and VLP-16 Specifications.

Annotating Ground Truth Labels

The SIL dataset did not contain any ground truth labels for the point clouds as compared to the KITTI dataset. As a result, there was no metric to assess the performance of the network. To solve this, objects from the dataset needed to be annotated to obtain some ground truth bounding boxes.

L-CAS Annotation Tool

Developed by the Lincoln Centre for Autonomous Systems Research, this tool provides a semi-automatic labelling function that clusters and highlight regions of interest that may contain objects[48]. The user can then label them depending on the object class such as car, cyclist, pedestrian. The result is then stored in a file containing the objects and their positions in the point cloud. To use this tool, the .pcap capture files were converted into .pcd format using the ROS Velodyne point cloud package[3].

[3]See listing A.1

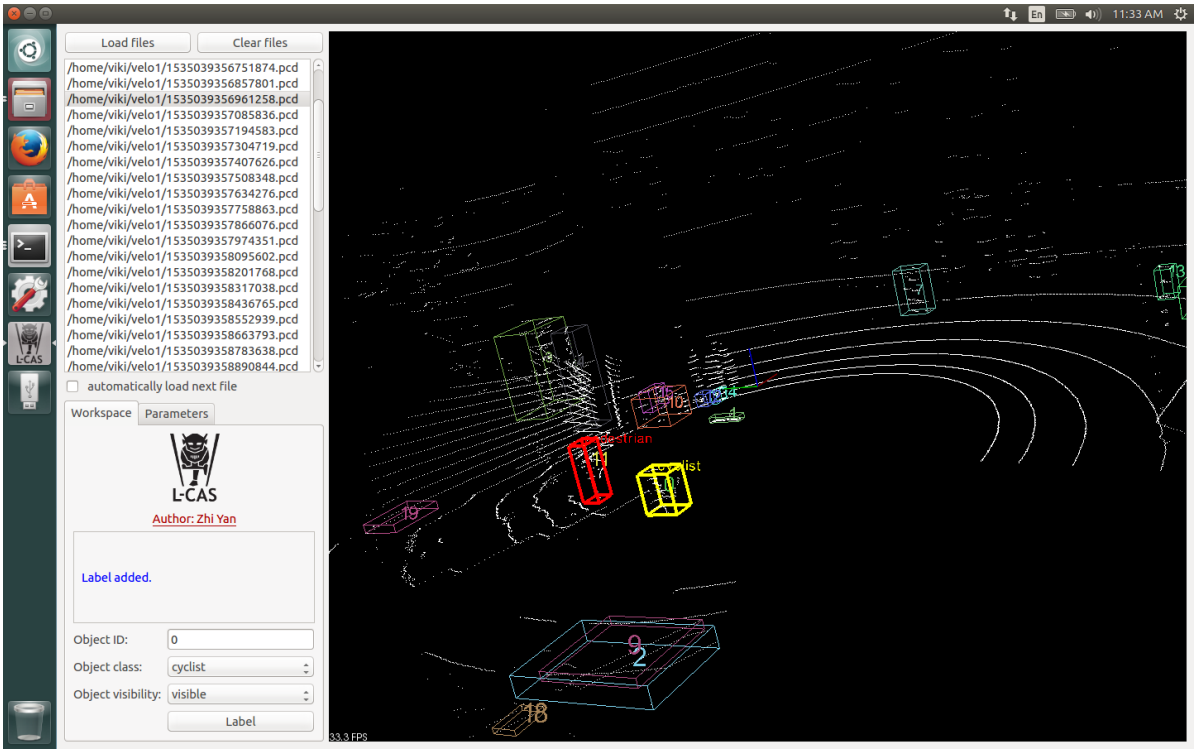


Figure 33: L-CAS Annotation Tool

3.3.4.1 Validation

Using the annotation tool, I annotated 500 frames to be used for validation. The output was a list of files containing object labels and their bounding boxes that would serve as the ground truth for validation. I then ran VoxelNet on these frames to generate the predicted bounding boxes for different object classes.

3.4 Conclusion

In the beginning of this section I set out to answer three important questions. Following a methodological approach I was able to establish tools to detect and characterise the context of images and point clouds. Building up on this, I was able to identify AVOD(LiDAR+image) and VoxelNet(LiDAR only) as the suitable object detection models to test whether they perform better in different contexts. Finally, I was able to identify a potential challenge with LiDAR only models that require the specifications of the LiDAR sensor in order to ensure correct projection and manipulation of the point clouds. As such, using point clouds from different sensors may result in incorrect predictions.

CHAPTER 4

RESULTS AND ANALYSIS

4.1 Can We Automatically Detect Scene Contexts?

4.1.1 Metrics

In evaluating the classifiers, it was crucial to understand the predictive power of the two context classifiers. As such, precision, recall and f1-score were selected as the main performance metrics.

Precision is the percentage of positive predictions that were correct.

$$Precision = \frac{TP}{TP + FP}$$

Recall is the percentage of positive samples that were correctly predicted.

$$Recall = \frac{TP}{TP + FN}$$

F1-score is the balanced mean of the precision and recall.

$$2 \cdot \frac{precision \cdot recall}{precision + recall}$$

4.1.2 Analysis

Tables 42 and 41 form the basis of this analysis.

I established that determining the context from images is much more accurate than from point clouds. As highlighted before, images offer higher representational power and resolution than sparse point clouds. As seen in figure 41, the f1-score of the image classifier outperformed the point cloud classifier by more than 30% .

	Context Detection using PointCloud Feature Matching				Context Detection using Image Segmentation			
Context	precision	recall	f1-score	support	precision	recall	f1-score	support
urban	**0.52**	0.45	0.48	206	0.81	**0.9**	**0.85**	193
non-urban	0.51	**0.58**	**0.55**	206	**0.9**	0.81	**0.85**	218
avg / total	0.51	0.51	0.51	412	0.86	0.85	0.85	411

Table 41: Comparison of context detection methods

With regard to the point cloud context classifier, majority voting resulted in around 50:50 precision for both classes. The recall for the non-urban was 7 percent more than urban which was below 50%. By implementing the penalty function in the point cloud classifier, recall for the urban class improved by 17% as seen in table 42.

Penalised Point Cloud Context Classifier				
context	**precision**	**recall**	**f1-score**	**support**
urban	**0.52**	**0.62**	**0.54**	206
non-urban	0.46	0.33	0.38	206
avg / total	0.47	0.47	0.46	412

Table 42: Penalised Point Cloud classifier

4.2 Do Object Detection Models Perform Better in Different Contexts?

4.2.1 Metrics

KITTI Evaluation Protocol

The models were evaluated using the official KITTI evaluation protocol. Average precision(AP) is the area under a recall-precision curve that was calculated after predictions. Predicted bounding boxes were considered as positive matched if they were above a predefined intersection over union(IoU). IoU is calculated as:

$IoU = \frac{Overlap area}{Union area}$

Where the overlap area is the region that is shared between the predicted bounding box and the ground truth bounding box and the Union area is the total area of the predicted

bounding box and ground truth box. This was calculated for 2D bounding boxes, 3D bounding boxes and Bird-Eye View bounding boxes. The IoU threshold was set at ≥0.7 for the car class and ≥0.5 for the pedestrian class.

GPU

As mentioned earlier, the resources of an AV are limited and therefore it is encouraged to implement efficient models. In light of this, I monitored the GPU statistics to better understand how different models affect the GPU. During inference, GPU statistics were collected at an interval of 1 second using NVIDIA System Management Interface(nvidia-smi). The following metrics were collected.

- **Power Draw** - Watts
- **Memory Used** - MB
- **Temperature** - °C
- **Clock Speed** - MHz

For uniformity purposes, all the experiments were performed on single NVIDIA P100 GPU with 16GB High Bandwidth Memory.

Baseline

A baseline, was established by running each model was run on the original KITTI validation dataset. The results can be seen in table in 41. In addition, the temperature and power metrics were collected and can be seen in the last boxplot of figures 41 and 42.

	Class	**Car**			**Pedestrian**		
Model	**Difficulty**	**Easy**	**Medium**	**Hard**	**Easy**	**Medium**	**Hard**
AVOD	***2D Bounding Box***	**86.987999**	**77.10569**	**68.012459**	56.214417	51.969517	46.578247
	BEV Bounding Box	**86.00412**	**74.355942**	**65.62397**	**50.94368**	**49.857227**	**44.704525**
	3D Bounding Box	**75.447769**	**63.742085**	**54.334251**	**48.757492**	**44.84359**	**43.014023**
VoxelNet	***2D Bounding Box***	65.951279	56.683437	55.972511	**66.309807**	**60.556213**	**53.479141**
	BEV Bounding Box	81.242409	70.672707	65.347595	50.308723	46.726196	41.24551
	3D Bounding Box	40.000149	34.892159	31.376112	34.161884	30.576441	28.982103

Table 41: Baseline

Running Each Model Exclusively

To investigate whether the performance of each model varies in different contexts, each model was run exclusively on a single GPU with data from each context dataset.

	Context	Urban			Non-Urban		
Model	**Difficulty**	**Easy**	**Medium**	**Hard**	**Easy**	**Medium**	**Hard**
AVOD	***2D Bounding Box***	**86.987999**	**77.10569**	**68.012459**	**89.186172**	**79.754562**	**78.550514**
	BEV Bounding Box	86.00412	74.355942	65.62397	87.11274	**76.859818**	**75.720955**
	3D Bounding Box	**75.447769**	**63.742085**	**54.334251**	**75.430656**	**64.199951**	**62.902302**
VoxelNet	***2D Bounding Box***	69.597939	65.98201	59.284454	77.520409	67.733231	62.281651
	BEV Bounding Box	**86.34037**	**76.187859**	**68.103294**	**88.635025**	75.361214	69.502548
	3D Bounding Box	73.632263	58.473991	50.744587	68.620865	49.455608	45.809917

Table 42: Car AP

	Context	Urban			Non-Urban		
Model	**Difficulty**	**Easy**	**Medium**	**Hard**	**Easy**	**Medium**	**Hard**
AVOD	***2D Bounding Box***	**78.626633**	**77.407684**	**70.029228**	76.096977	70.946465	71.205276
	BEV Bounding Box	**80.880447**	**80.393105**	**79.296593**	77.806831	77.654358	71.792923
	3D Bounding Box	**80.649719**	**80.142876**	**79.056778**	77.538368	71.868484	71.564285
VoxelNet	***2D Bounding Box***	72.129387	65.245384	65.508232	**84.218407**	**76.52832**	**76.593803**
	BEV Bounding Box	74.766678	73.944069	74.138588	**89.830276**	**80.201164**	**80.114815**
	3D Bounding Box	71.177628	64.247559	64.18145	**87.206612**	**77.964493**	**78.036652**

Table 43: Pedestrian AP

Analysis

The qualitative results of the models are illustrated in tables 42, 43 and 44 .Results for the urban context are highlighted in pink whereas non-urban results are highlighted in blue. GPU results are available in form of boxplots on figures 42 and 41. For this evaluation, the first six boxplots form the basis of this analysis. The median is highlighted at the bottom of the x-axis in gold and blue while the average is in black font next to the 'X' marker.

Car - AVOD outperformed VoxelNet on 2D and 3D detections in both contexts. However, VoxelNet performed slightly better in bird-eye-view detections as compared to AVOD.

Pedestrian - As compared to the car class, there was a distinct performance boundary whereby AVOD dominated all detections in the urban context whereas VoxelNet did in the non-urban context.

Context	**VoxelNet**			**AVOD**		
	min	**max**	**mean**	**min**	**max**	**mean**
Urban	0.113	3.813	0.129	**0.095**	**2.457**	**0.112**
Non-urban	0.113	**2.224**	0.127	**0.096**	2.506	**0.113**

Table 44: Inference Time on Single NVIDIA P100 GPU

In terms of inference time, AVOD was slightly faster than VoxelNet in both contexts Notably, the minimum time for VoxelNet was AVOD's mean time. Nonetheless both were able to achieve a frame rate of about **9 FPS**.

GPU - As seen in figures 42 and 41,AVOD generally ran at a lower temperature and consumed less power as compared to VoxelNet. Generally, VoxelNet caused higher temperatures more than 75% of the time. The spread of temperature was smaller for AVOD as compared to VoxelNet's. In terms of **power usage**, a similar pattern was seen whereby the AVOD's usage stabilised within a very small range most of the time as compared to VoxelNet which exhibited a significantly higher range fluctuation. With regard to the context, running the models on a specific context resulted in lower median temperatures as seen in figure 41. For VoxelNet, the median temperature reduced by around 4°Celsius whereas for AVOD the median temperature was reduced by around 3°Celsius. Additionally, VoxelNet had a smaller spread while running on the urban context as compared to VoxelNet. AVOD did not exhibit any significant changes in terms of spread while running in different contexts. AVOD's and VoxelNet's median power usage did not exhibit any significant reduction. However, AVOD running in a non-urban context resulted in a very small spread with a slightly lower mean temperature as compared to running in an urban context. An opposite effect was seen in VoxelNet's spread where running in a non-urban context resulted in a larger spread and slightly higher mean temperature.

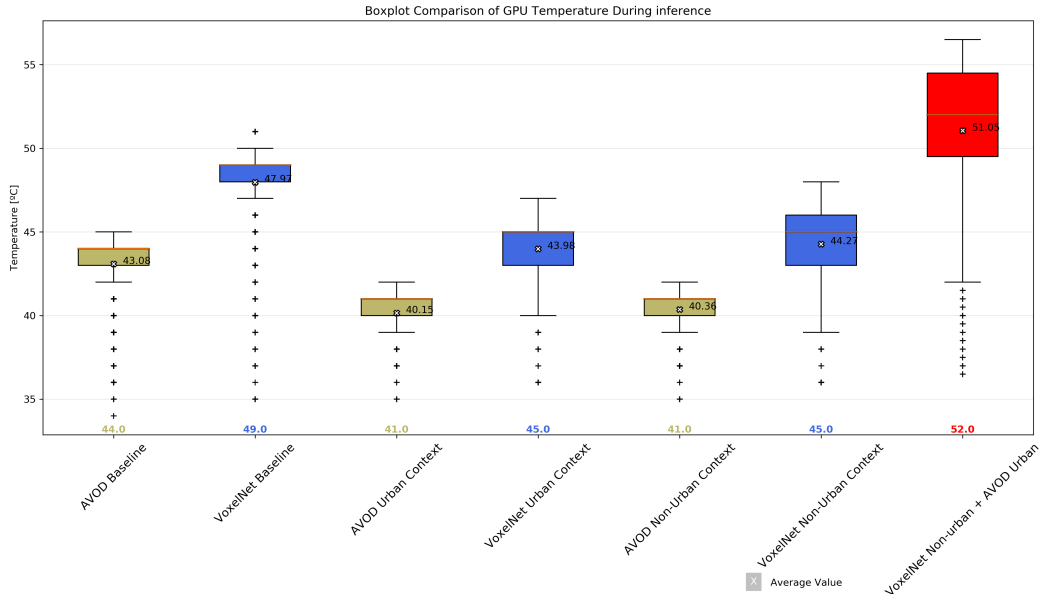


Figure 41: Temperature boxplots of evaluation runs.Voxelnet(blue), AVOD(gold), AVOD+VoxelNet(Red)

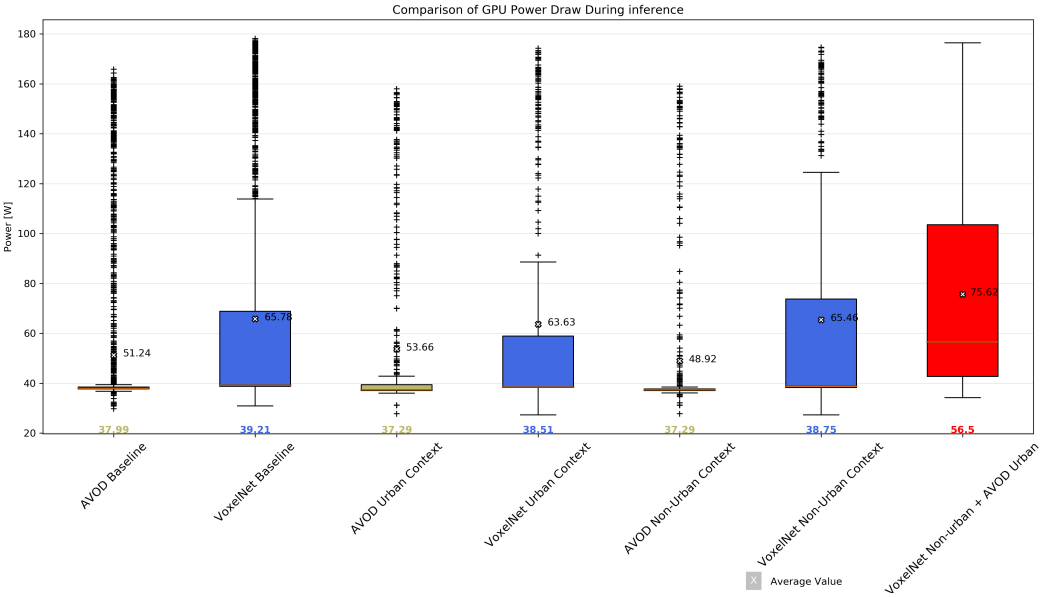


Figure 42: Power boxplots of evaluation runs. Voxelnet(blue), AVOD(gold), AVOD+VoxelNet(Red)

Running Both Models Simultaneously

Having identified that AVOD is better at detecting cars and pedestrians in urban contexts and VoxelNet in non-urban contexts, I set out to further investigate whether it is possible

to have both models running optimally on a single GPU. This was done by running both models simultaneously. Each model was started on an initial memory fraction of 40% (6.4GB) and was allowed to further allocate more memory if need be. This was done by setting the ***gpu_mem_fraction*** to **0.4** and ***allow_mem_growth*** to **True** before running the respective Tensorflow graphs.

Table 45: Memory

	Min	**Max**
Memory(MB)	4608.50	14320.00

Analysis

The results are available in form of box-plots on figures 42 and 41 as well as table 45. For this test, the last box-plot forms the basis of this analysis.The median is highlighted at the bottom of the x-axis in red while the average is in black font next to the 'X' marker.

Temperature - Running both models simultaneously resulted in significantly higher temperatures as compared running them exclusively. The average temperature was 7°Celsius hotter than the maximum mean temperature from running the models exclusively. In addition, the temperature exhibited a significantly greater spread than running either model on any context.

Power - A similar trend was seen power-wise. The mean power usage was 10 watts greater than the maximum mean power from running the models exclusively. In addition, the spread of power was much larger than running either context. The median power usage was also much higher by around 16 Watts.

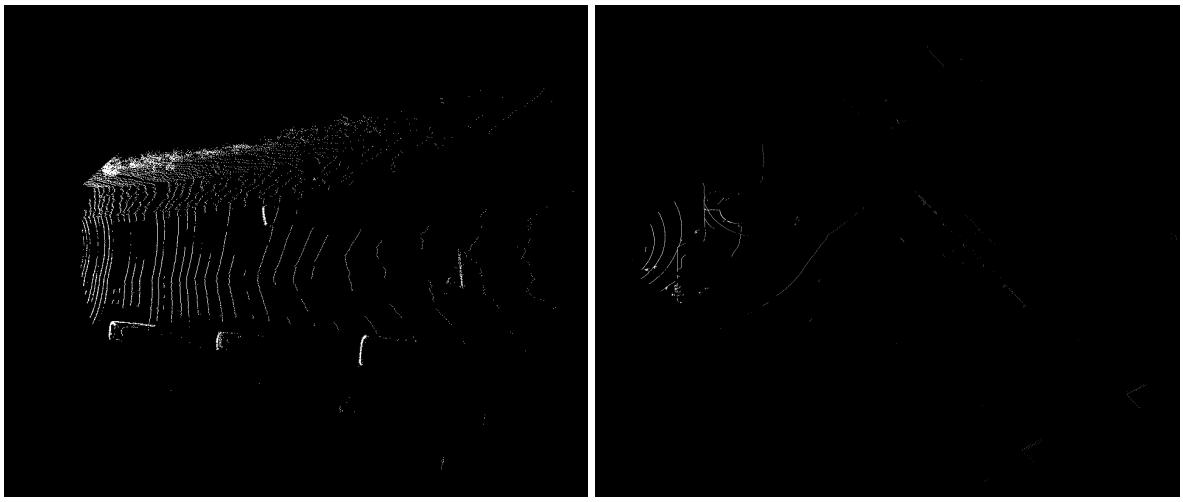
Memory - Running both models resulted in a total memory usage of 14.3 GB out of the total memory of 16GB. Both models were able to run optimally with no memory exhaustion errors.

4.3 Is VoxelNet Performance Valid For Different Datasets?

4.3.1 Metrics

Given that the data was collected from a stationary vehicle in an urban area with mostly pedestrians, VoxelNet was evaluated on the pedestrian class only. However, as there was no corresponding image data for the point clouds, only the 3D bounding boxes and Bird-Eye View bounding boxes with IoU ≥ 0.5. were used in calculating the AP.

4.3.2 Analysis



KITTI bird eye view SIL bird eye view

Figure 41: Bird Eye View Difference between SIL and KITTI

VoxelNet was **not** able to predict any objects on the SIL dataset. As highlighted before, the VLP-16 LiDAR sensor has a lower frequency than that of the HDL-64E. This therefore affected the point cloud density and is visible in figure 41. From this figure, we can see that more SIL point cloud density was significantly lower than that of KITTI. SIL point clouds had an average point cloud density per frame that was lower than that of KITTI's by 42% with a median of 0.0048 in 1000 samples as seen in figure 42 as compared to SIL's of 0.0028. In fact, close to 100% of the KITTI point cloud density was higher than that of SIL. Nonetheless, the KITTI point cloud density fluctuated along a larger range as compared to the SIL dataset.

The point cloud density was calculated as:

$$density = \frac{N_i}{S_i}$$

where: N is the number of points in point cloud frame i and S is the XY size of the frame i.

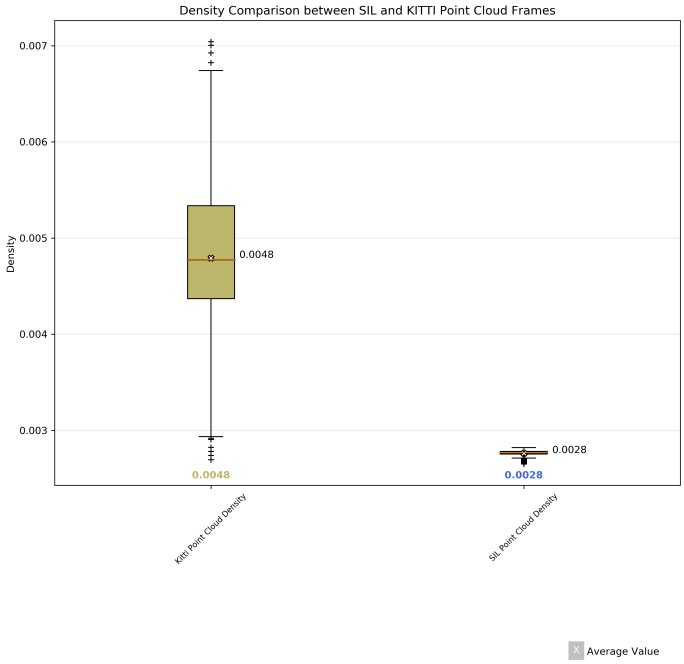


Figure 42: Point Cloud Density

Following this evaluation, I established that there was a positive correlation between the performance of the LiDAR only models and the point cloud density of the input. As mentioned in the precious section, point clouds are sparse and as a result suffer from poor contour retrieval for objects. An increase in the point density could help alleviate this by allowing for more shape information to be recovered.

4.4 Discussion

From the analysis it is clear that predicting the context using images is far much accurate than using point clouds. This performance can be further complimented by use of map information from leading map providers such as TomTom and Google or even in-house maps from companies that have developed their own such as Google.

With regard to multimodal models that fuse camera and LiDAR, AVOD has proven to perform much better in terms of detection in urban contexts with more dynamic objects. Aided by input from LiDAR and cameras, occluded objects can be detected much better. In addition, the bounding boxes are much tighter leading to higher IoUs. Nonetheless, it cannot go without mentioning the importance of sensor placement. Multimodal methods do not always guarantee that objects will be detected. Recently, and UBER AV was involved in accident where a pedestrian was knocked down at night when crossing the road. Despite the vehicle being fitted with a LiDAR sensor that should've allowed the pedestrian to be detected, it did not do so. It was later established that the placement of the single LiDAR on the roof resulted in a 300ft blind spot radius[40]. As such, it may be more suitable to have LiDAR units all around the car and cheaper but high quality camera units as the object detection models have proven to be highly invariant to resolution. Especially with the development of solid-state LiDARs that are cheaper, scalable and robust as they lack expensive optics and mechanical parts as seen in current implementations like the Velodyne HDL-64E.

Finally, the impact of point cloud density on the performance of VoxelNet was realised. Following critical evaluation, it was clear that VoxelNet is not sensor-agnostic and as such requires the model to be modified to include the specifications of the LiDAR sensor.

Graph Operations

As mentioned, Tensorflow defined neural networks as computational graphs with nodes, edges and operations.

Node - Computations happen in nodes and may have multiple inputs. The data is in form of tensors.

Edges - Define how tensor moves within the graph,branching and any looping states.

Operation - This is a form of computation that is defined in the Tensorflow library. Can include multiplication, addition or even abstract operations such as assertions.[1]

The number of nodes, edges and graphs greatly influence the performance of the model while running. Floating point operations vary depending on the operation. Matrix multiplications form the bulk of many operations and the FLOPS are calculated as:

[1]Full list of operations can be found here: `https://gist.github.com/dustinvtran/cf34557fb9388da4c9442ae25c2373c9`

$$AB_{flops} = xq \cdot (2y - 1)$$
where $Shape_A = (x, y)$ and $Shape_B = (p, q)$

The total number of parameters is calculated as:

$$\Sigma_{j=0}^{N} \Pi_{i=0}^{rank} V_i$$
Where N is the total number of trainable variables, V_i is the shape of variable V for tensor rank i.

These values can be calculated using the Tensorflow profiler. An example snippet can be seen in appendix A.2.

AVOD

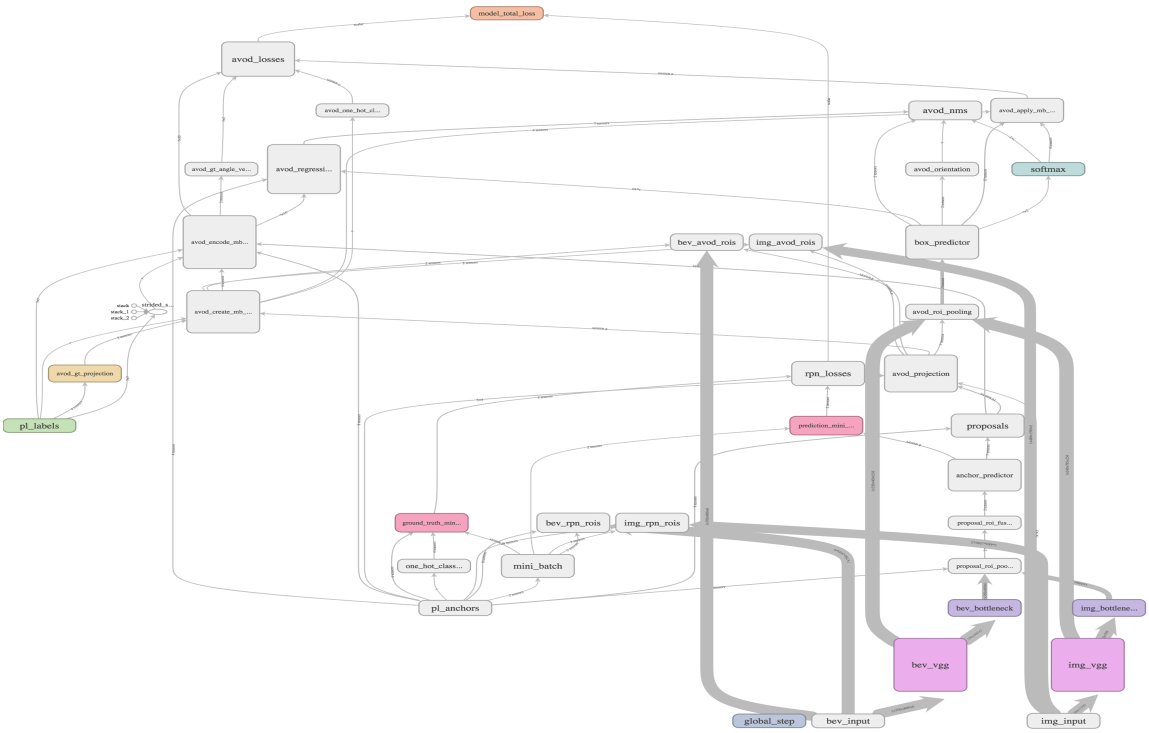


Figure 41: AVOD Tensorflow Graph

- **Total Floating Point Operations** - 696,694,670,729(≈ 696 Billion)
- **Number of Parameters** - 16,682,424(≈16.6 Million)

VoxelNet

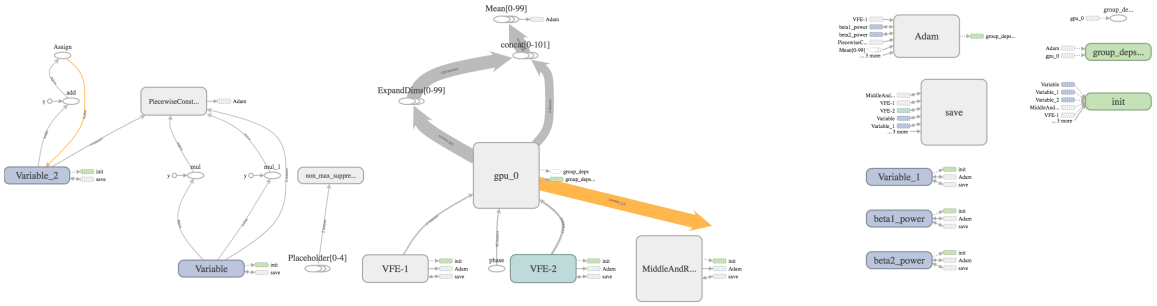


Figure 42: VoxelNet Tensorflow Graph

- **Total Floating Point Operations** - 1,227,846,793,165(≈1.27 Trillion)
- **Number of Parameters** - 6661552(≈6.6 Million)

Effect on Power and Temperature

The increased power and temperature usage of VoxelNet as compared to AVOD is attributed to the fact that it has ≈ 531 billion more FLOPs than AVOD. This represents a 43% increase in complexity compared to AVOD. The increased operations are due to the fact that most operations operations on the point cloud are matrix multiplications to manipulate the point clouds.

On the other hand, AVOD had ≈10 million more parameters than VoxelNet, this is more than a 60% increase. This can be attributed to the increased number of variables to hold the input for the image and point cloud RPNs as seen in figure 41.

Despite seeing the higher temperature and power usage, this occurred when both models were running simultaneously. However this would not be the case in an actual implementation as one model will only be used while the other one will be inactive. This can be implemented using a pipeline system whereby both Tensorflow graphs can be loaded on a single GPU, however, depending on the context, the input can be fed into either graph while the other one remains dormant and vice versa. Therefore the usage would be much lesser than actually having both running.

CHAPTER 5

CONCLUSION AND FUTURE WORK

This project developed two novel method to detect and characterise contexts from images and point clouds respectively. The methods have been tested on the KITTI dataset and image context detection has proved to be quite accurate as compared to point cloud context detection. I have shown that cutting edge multimodal and LiDAR only methods perform differently in urban and non-urban environments with multimodal methods performing much better in urban environments and LiDAR only performing close to multimodal methods in non-urban contexts. Furthermore I was able to establish that it is possible to run a multimodal method(AVOD) in urban contexts and a LiDAR only model(VoxelNet) simultaneously on a single GPU optimally. This opens up the possibility of a pipeline model whereby depending on the context, it may be possible to switch to a different model without being affected. Figure 51 illustrates a prototype of such a system. The GPU metrics and graph statistics were able to provide an insight into the correlation between the temperature generation and FLOPS as well as between the number of parameters and memory usage whereby higher FLOPs resulted in higher temperatures and similarly, higher parameters resulted in higher memory usage.

The downsides of LiDAR only methods in terms of generalising to different LiDAR sensors was also established whereby the frequency of point returned per second affected the point cloud density and consequently VoxelNet did not produce any predictions. This highlighted the need to either standardise the use of sensors or to develop sensor-agnostic object detection models that can generalise to different sensors and datasets. Alternatively, depth filling methods can be utilised to increase the denseness of these

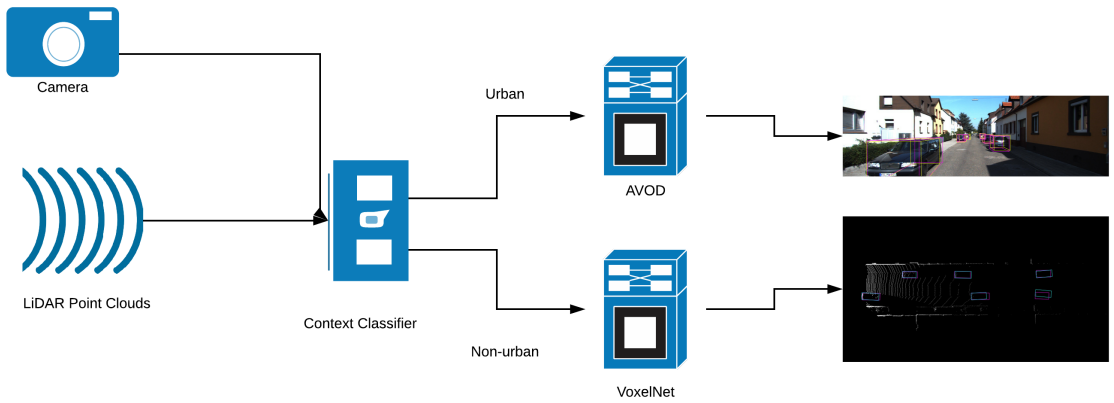


Figure 51: Context-Dependent Model Pipeline Prototype

point clouds before being used as input[17].

A final contribution was to make publicly available the SIL dataset with annotated samples that can be used in developing new object detection models and testing existing ones to improve their generalisation.

5.1 Future Work

Despite being able to fulfill the objectives set out, this project can be further improved in future iterations. Firstly, the process of context detection can be implemented using deep learning methods to enable automatic feature extraction instead of using manual feature extraction using histograms and descriptors. This could be similar to the feature extractor layers in AVOD that are able to extract features automatically from both image and point clouds. In doing so, the performance of the LiDAR point cloud detection could be improved as well. Furthermore, using a deep learning methods, it is possible to extend the implementation into an end to end model that can be trained all at once.

Another avenue could be exploring the use different accelerators other than the GPUs. [20] investigated the use of ASIC, FPGAs and GPU for different tasks such as detection, tracking and localisation and established that using ASICs and FPGA could could restrain the driving range reduction to around 5%. This could be further be improved by using models with lightweight architectures with high accuracy but have low memory and computational power requirements. A recently published model that tries to achieve this is [44].

5.2 Personal Reflection

This project greatly helped me understand various perspectives on the development of AVs. Beginning from the technical, legal and even economic aspects that form the driving force for different design choices. This project evolved from simply trying to replicate the VoxelNet model that was not publicly released to later on realising a gap in the assessment of object detection models in different contexts and further to realising the impact of different LiDAR sensors on the performance of these models. This process helped me understand more about how the development of such systems tend to be quite complex and involve the collaboration of different disciplines to better understand how they can be fully integrated.

In terms of programming, the graph model abstraction of the Tensorflow framework presented a steep learning curve for me as I was used to imperative programming frameworks. Debugging was quite difficult but over time, I was able to understand how to manipulate the graphs. This was especially the case while implementing the focal loss function that resulted in inf and nan errors. Another time consuming element was hyper parameter tuning for the networks. Most models that were released in public did not include the best parameters for training these networks, as such a lot of time was spent in finding these parameters. CometML greatly helped in tracking the parameters that were the best performing and I could automatically create a GitHub pull request for them.

Working with point clouds also tended to be quite difficult as there are not many Python libraries that can manipulate point clouds and those that are available do not support as many features. As a result, a lot of time was spent in the research of different methods to handle the point clouds for the task of context detection and object detection. This resulted in using a lot of boiler plate code using different libraries. If there was enough time, this could've been improved by compiling these functions into one library that could provide these functions in a simpler manner.

APPENDIX A

LiDAR	Hor FOV	Ver FOV	Range	Angular Resolution	Points/second	Channels
VLS-128	360°	+15°to -25°	300m	0.11°	~9.6 Million	128
HDL-64E	360°	26.9°	120m	~0.4°	~2.2 Million	64
HDL-32E	360°	+10°to -30°	80m-100m	0.1°	~1.39 Million	32
VLP-32C	360°	+15°to -25°	200m	0.1°	~1.2 Million	32
VLP-16	360°	± 15°	100m	0.1°	600,000	16

Table A1: Velodyne LiDAR Family

```
#Code to read .pcap files and convert them to .pcd

#Load existing pcap file and create a sensor publisher
$ roslaunch velodyne_pointcloud VLP-16-points.launch pcap:=(path to .pcap file)

#Receiver to  convert sensor messages to .pcd (In separate terminal window)
$ rosrun pcl_ros pointcloud_to_pcd input:=/velodyne_points _prefix:=(path to save .pcd file)

```

Listing A.1: .pcap -> .pcd

```
def calculate_metrics(chkpoint):
  """
  Profiles a Tensorflow Graph to obtain the parameters and FLOPS
  params:
  chkpoint: Tensorflow checkpoint.
```

```
  returns:
  stats: FLOPS proto object
  params: Parameters proto objects

"""
  with tf.Graph().as_default():
    #GPU options.
    gpu_options = tf.GPUOptions(per_process_gpu_memory_fraction=0.4,
                                visible_device_list='0',
                                allow_growth=True)
    #create session config proto object
    config = tf.ConfigProto(
              gpu_options=gpu_options,
              device_count={"GPU": 1},
              allow_soft_placement=True,
              log_device_placement=False
             )
    #Include to initialise variables.
    run_metadata = tf.RunMetadata()
    # Run session with specified configs.
    with tf.Session(config=config) as sess:
        saver = tf.train.import_meta_graph(chkpoint+'.meta', clear_devices=True)
        saver.restore(sess, chkpoint)
        opts =tf.profiler.ProfileOptionBuilder.float_operation()
        stats = tf.profiler.profile(sess.graph, run_meta= run_metadata, cmd='op'
    , options=opts)
        opts = tf.profiler.ProfileOptionBuilder.trainable_variables_parameter()
        params = tf.profiler.profile(sess.graph, run_meta=run_metadata, cmd='op'
    , options=opts)
    return stats,params
```

Listing A.2: Calculating FLOPS and Parameters

A.1 Code Listing

The full code listing is contained on a Github repository that can be accessed through the following link:

`https://github.com/jonesmabea/Thesis/tree/master/src`

BIBLIOGRAPHY

[1] *Artificial intelligence for construction productivity*, 2018.

[2] H. BAY, T. TUYTELAARS, AND L. VAN GOOL, *Surf: Speeded up robust features*, in European conference on computer vision, Springer, 2006, pp. 404–417.

[3] I. BROWN, *Human factors: the journal of the human factors and ergonomics society*, Driver Fatigue, 36 (1994), pp. 298–314.

[4] S. D. BROWN, R. J. FRANCIS, J. ROSE, AND Z. G. VRANESIC, *Field-programmable gate arrays*, vol. 180, Springer Science & Business Media, 2012.

[5] X. CHEN, K. KUNDU, Z. ZHANG, H. MA, S. FIDLER, AND R. URTASUN, *Monocular 3d object detection for autonomous driving*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 2147–2156.

[6] X. CHEN, K. KUNDU, Y. ZHU, H. MA, S. FIDLER, AND R. URTASUN, *3d object proposals using stereo imagery for accurate object class detection*, IEEE transactions on pattern analysis and machine intelligence, 40 (2018), pp. 1259–1272.

[7] X. CHEN, H. MA, J. WAN, B. LI, AND T. XIA, *Multi-view 3d object detection network for autonomous driving*, in IEEE CVPR, vol. 1, 2017, p. 3.

[8] N. DALAL AND B. TRIGGS, *Histograms of oriented gradients for human detection*, in Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on, vol. 1, IEEE, 2005, pp. 886–893.

[9] F. DEPARTMENT FOR ENVIRONMENT AND R. AFFAIRS, *Urban and rural area definitions for policy purposes in england and wales: Methodology (v1.0)*, GOV.UK, (2018).

[10] K. DIETMAYER, *Predicting of machine perception for automated driving*, in Autonomous Driving, Springer, 2016, pp. 407–424.

[11] B. H. FRIED, *What does matter? the case for killing the trolley problem (or letting it die)*, The Philosophical Quarterly, 62 (2012), pp. 505–529.

[12] T. M. GASSER, *Fundamental and special legal questions for autonomous vehicles*, in Autonomous Driving, Springer, 2016, pp. 523–551.

[13] A. GEIGER, P. LENZ, AND R. URTASUN, *Are we ready for autonomous driving? the kitti vision benchmark suite*, in Conference on Computer Vision and Pattern Recognition (CVPR), 2012.

[14] R. GIRSHICK, *Fast r-cnn*, in International Conference on Computer Vision (ICCV), 2015.

[15] R. GIRSHICK, J. DONAHUE, T. DARRELL, AND J. MALIK, *Rich feature hierarchies for accurate object detection and semantic segmentation*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2014, pp. 580–587.

[16] INTEL, 2018.
https://www.intel.co.uk/content/www/uk/en/automotive/go-automated-driving.html.

[17] J. KU, A. HARAKEH, AND S. L. WASLANDER, *In defense of classical image processing: Fast depth completion on the CPU*, CoRR, abs/1802.00036 (2018).

[18] P. LAW, *Law 106-398,„Äúfloyd d*, Spence National Defense Authorization Act for Fiscal Year, (2001).

[19] B. LI, T. ZHANG, AND T. XIA, *Vehicle detection from 3d lidar using fully convolutional network*, arXiv preprint arXiv:1608.07916, (2016).

[20] S.-C. LIN, Y. ZHANG, C.-H. HSU, M. SKACH, M. E. HAQUE, L. TANG, AND J. MARS, *The architectural implications of autonomous driving: Constraints and acceleration*, in Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems, ACM, 2018, pp. 751–766.

[21] T.-Y. LIN, P. GOYAL, R. GIRSHICK, K. HE, AND P. DOLLÁR, *Focal loss for dense object detection*, IEEE transactions on pattern analysis and machine intelligence, (2018).

[22] D. G. LOWE, *Object recognition from local scale-invariant features*, in Computer vision, 1999. The proceedings of the seventh IEEE international conference on, vol. 2, Ieee, 1999, pp. 1150–1157.

[23] L. MEDSKER AND L. JAIN, *Recurrent neural networks*, Design and Applications, 5 (2001).

[24] A. MILLER, *Some of the companies that are working on driverless car technology*, 2018.

[25] MOBILEYE, *Enabling autonomous - mobileeye*, 2018. https://www.mobileye.com/future-of-mobility/mobileye-enabling-autonomous/.

[26] A. NEWELL AND S. K. CARD, *The prospects for psychological science in human-computer interaction*, Human-computer interaction, 1 (1985), pp. 209–242.

[27] L. PLOTKIN, *Pydriver*, bachelor's thesis (Studienarbeit), Karlsruhe Institute of Technology, Germany, Mar. 2015.

[28] D. POMERLEAU, *Rapidly adapting artificial neural networks for autonomous navigation*, in Advances in neural information processing systems, 1991, pp. 429–435.

[29] D. A. POMERLEAU, *Alvinn: An autonomous land vehicle in a neural network*, in Advances in neural information processing systems, 1989, pp. 305–313.

[30] C. R. QI, W. LIU, C. WU, H. SU, AND L. J. GUIBAS, *Frustum pointnets for 3d object detection from rgb-d data*, arXiv preprint arXiv:1711.08488, (2017).

[31] C. R. QI, H. SU, K. MO, AND L. J. GUIBAS, *net: Deep learning on point sets for 3d classification and segmentation*, Proc. Computer Vision and Pattern Recognition (CVPR), IEEE, 1 (2017), p. 4.

[32] C. R. QI, L. YI, H. SU, AND L. J. GUIBAS, *Pointnet++: Deep hierarchical feature learning on point sets in a metric space*, in Advances in Neural Information Processing Systems, 2017, pp. 5105–5114.

[33] S. REN, K. HE, R. GIRSHICK, AND J. SUN, *Faster r-cnn: Towards real-time object detection with region proposal networks*, in Advances in neural information processing systems, 2015, pp. 91–99.

[34] N. RESEARCH, *Navigant research leaderboard: Automated driving vehicles*, Jan 2018.
https://www.navigantresearch.com/research/navigant-research-leaderboard-automated-driving-vehicles.

[35] R. B. RUSU, N. BLODOW, AND M. BEETZ, *Fast point feature histograms (fpfh) for 3d registration*, in Robotics and Automation, 2009. ICRA'09. IEEE International Conference on, Citeseer, 2009, pp. 3212–3217.

[36] S. SALTI, F. TOMBARI, AND L. DI STEFANO, *Shot: Unique signatures of histograms for surface and texture description*, Computer Vision and Image Understanding, 125 (2014), pp. 251–264.

[37] J. SHOTTON, A. FITZGIBBON, M. COOK, T. SHARP, M. FINOCCHIO, R. MOORE, A. KIPMAN, AND A. BLAKE, *Real-time human pose recognition in parts from single depth images*, in Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on, Ieee, 2011, pp. 1297–1304.

[38] J. SHOTTON, M. JOHNSON, AND R. CIPOLLA, *Semantic texton forests for image categorization and segmentation*, in Computer vision and pattern recognition, 2008. CVPR 2008. IEEE Conference on, IEEE, 2008, pp. 1–8.

[39] M. J. S. SMITH, *Application-specific integrated circuits*, vol. 7, Addison-Wesley Reading, MA, 1997.

[40] H. SOMERVILLE, *Uber's use of fewer safety sensors prompts questions after arizona...*, Mar 2018.

[41] B. STEDER, R. B. RUSU, K. KONOLIGE, AND W. BURGARD, *Narf: 3d range image features for object recognition*, in Workshop on Defining and Solving Realistic Perception Problems in Personal Robotics at the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), vol. 44, 2010.

[42] R. TACHET, P. SANTI, S. SOBOLEVSKY, L. I. REYES-CASTRO, E. FRAZZOLI, D. HELBING, AND C. RATTI, *Revisiting street intersections using slot-based systems*, PloS one, 11 (2016), p. e0149607.

[43] P. VIOLA AND M. JONES, *Rapid object detection using a boosted cascade of simple features*, in Computer Vision and Pattern Recognition, 2001. CVPR 2001. Pro-

ceedings of the 2001 IEEE Computer Society Conference on, vol. 1, IEEE, 2001, pp. I–I.

[44] Y. WANG, T. SHI, P. YUN, L. TAI, AND M. LIU, *Pointseg: Real-time semantic segmentation based on 3d lidar point cloud*, arXiv preprint arXiv:1807.06288, (2018).

[45] WAYMO, *Waymo safety report*, 2018. https://waymo.com/safety/.

[46] T. WINKLE, *Development and approval of automated vehicles: considerations of technical, legal, and economic risks*, in Autonomous Driving, Springer, 2016, pp. 589–618.

[47] S. S. WU, *Product liability issues in the us and associated risk management*, in Autonomes Fahren, Springer, 2015, pp. 575–592.

[48] Z. YAN, T. DUCKETT, AND N. BELLOTTO, *Online learning for human classification in 3d lidar-based tracking*, in In Proceedings of the 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vancouver, Canada, September 2017.

[49] Y. ZHONG, *Intrinsic shape signatures: A shape descriptor for 3d object recognition*, in Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on, IEEE, 2009, pp. 689–696.

[50] Y. ZHOU AND O. TUZEL, *Voxelnet: End-to-end learning for point cloud based 3d object detection*, arXiv preprint arXiv:1711.06396, (2017).