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

AUTHOR'S DECLARATION

This dissertation is submitted to the University of Bristol in accordance with the requirements of the degree of Master of Science in the Faculty of Engineering. It has not been submitted for any other degree or diploma of any examining body. Except where specifically acknowledged, it is all the work of the Author.

SIGNED: ***Jones Mabea Agwata*** DATE: **SEPTEMBER, 2018**

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 cameras in favour of LiDAR has become more plausible. As compared to cameras that suffer drawbacks such as visibility issues in extreme weather, LiDAR is extremely robust and accurate in various conditions.

This project will be 65% type I (Software Development) and 35% type II (Theoretical).

Dedication and acknowledgements

Here goes the dedication.

TABLE OF CONTENTS

Page

List of Tables **ix**

List of Figures **xi**

1 Introduction **1**
1.1 Aims and Objectives 2
1.2 Deliverables 2
1.3 Report structure 3

2 Background **5**
2.1 Components of an AV 5
2.2 Industrial Approaches 8
2.2.1 Camera 9
2.2.2 Camera and Radar 10
2.2.3 LiDAR, Camera and Radar 10
2.3 Object Detection 10
2.3.1 Object Detection using Classic Computer Vision Methods 10
2.3.2 Object Detection using Deep Learning 10
2.4 Legal, Ethical and Economic Considerations 14

3 Implementation **15**
3.1 Datasets 15
3.1.1 Context Classification 15
3.1.2 LiDAR Performance Evaluation 16
3.2 Can We Automatically Detect The Context in Driving Scenes? 16
3.2.1 Image Context Detection 17
3.2.2 PointCloud Context Detection 18
3.2.3 Implementation Issues 19
3.3 Do Some Object Detection Methods work better in Different Contexts? 20
3.3.1 VoxelNet 20

3.3.2 AVOD 22
3.3.3 Evaluation 23
3.3.4 Is VoxelNet Performance Valid for Different Datasets? 23
3.4 Conclusion 26

4 Results and Analysis **27**
4.1 Can We Automatically Detect Scene Contexts? 27
4.1.1 Metrics 27
4.1.2 Analysis 27
4.2 Do Object Detection Models Perform Better in Different Contexts? 28
4.3 Is VoxelNet Performance Valid For Different Datasets? 31
4.4 Analysis 31

5 Discussion and Future Work **33**

6 Conclusion **35**

A Appendix A **37**

Bibliography **39**

List of Tables

Table		Page
21	Modes of Operation of Common Architectures	14
31	Velodyne HDL-64E and VLP-16 Specifications.	24
41	Comparison of context detection methods	28
42	Penalised Point Cloud classifier	28
43	Baseline	29
44	Car AP	29
45	Pedestrian AP	30
46	Inference Time on Single NVIDIA P100 GPU	30
47	Memory	31
A1	Velodyne LiDAR Family	37

List of Figures

Figure **Page**

21 Velodyne LiDAR family. From left: Velodyne HDL-64E , HDL-32E , VLP-16 6
22 Components of Waymo's Self Driving Car, Source:[35] 7
23 AV Leaderboard. Navigant Research. Source:[27] 9
24 NN vs CNN Source: `https://cs231n.github.io/convolutional-networks` 11
25 Faster R-CNN overview. Source:[26] 12

31 Urban scene 17
32 Non-Urban scene 17
33 Semantic Histograms 18
34 Ambiguous scene 20
35 VoxelNet Architecture 23
36 AVOD Architecture 23
37 SIL Data Frame CSV 24
38 Veloview Frame of SIL data 24
39 L-CAS Annotation Tool 25

41 Temperature 30
42 Power 30

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). This is necessary for mapping the environment around the vehicle in order to safely navigate. In addition to the high cost of these sensors, fusing their input and running object detection models 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, companies are exploring different ways to reduce the number of sensors while still achieving a high level of navigational accuracy and safety. Extensive research has been undertaken to assess the performance of these sensors in different weather contexts such as in rain, fog and snow. However, the performance of these sensors in different area contexts, that is urban or non-urban, has not been exhaustively explored. Urban environments tend to have more dynamic objects and scenarios as compared to the static ones in most non-urban areas. As such, it can be argued that depending on the area context, the sensor configuration could be modified.

You probably need to present a succinct critique comment for each of the top 4 pieces of related work (which 'coincidentally' are limitations you try to address in your aims). Why do we need to test with different datasets? different models? Why is this important scientifically and also for companies developing lidar-onlu systems?

In order to investigate this, state of the art object detection models will be considered. Firstly, VoxelNet [37]], a LiDAR only model that uses point clouds as input. Secondly, Aggregated View Object Detection(AVOD) [14], a multimodal model that fuses image and point cloud data. Both model implementations were available on GitHub and were modified in order to align with the

aims of this project.

1.1 Aims and Objectives

Following the motivations in the presented discussion , the performance of 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:

- **Image and LiDAR Context Classifier**. Available as Jupyter interactive notebooks including pre-trained models.
- **Custom VoxelNet Model** Modified VoxelNet model including interactive notebooks for training, testing and validating the model.
- **Custom AVOD Model** Modified AVOD model including interactive notebooks for training, testing and validating the model.
- **Validation Dataset** Point Cloud dataset obtained from the University of Bristol Smart Internet Lab working on connected and AVs. Tools to convert the dataset into a trainable input for VoxelNet will be provided as well as some annotated sample frames.
- **Evaluation report.** The following topics will be discussed.
    1. A review of related research and implementations tackling object detection in AVs.
    2. Economic, ethical and legal analysis of the implementation and its potential impact on the development of AVs.
    3. Evaluation of context classifiers and object detection models in different contexts.
    4. Validation of VoxelNet using the Smart Internet Lab dataset.

1.3 Report structure

This report will consist of five main chapters.

- Chapter 2 discusses the different components of AVs, current implementations in the industry, a background on the research that has been undertaken in the field of object detection.
- Chapter 3 details the project execution and the methods undertaken to achieve the objectives.
- In Chapter 4, the results following evaluation of the methods will be discussed and analysed.
- Finally, a concluding chapter discusses the major findings, whether the objectives were achieved and a justification of their implications.

CHAPTER 2

BACKGROUND

The idea of autonomous vehicles(AVs) is not new. As early as 2005, DARPA had invested heavily in the creation of unmanned trucks and organised for the Urban Challenge [5] to allow for different teams to showcase their unmanned vehicles. However, due to the challenges such as low computational power and underdeveloped AI and ML systems, the resulting implementations were not practical and had a high fault rate of 1 fault in 100 miles compared to the human fault rate of around 1 in 100 million miles. Nonetheless, from this challenge, it was clear that the prospect of AVs was plausible and indeed possible.

Currently, AVs are divided into five levels of autonomy as defined be the Society of Automotive Engineers(SAE):

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 involvment.

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.

LiDAR units require complex optical systems that are expensive to build. As a result, they are the most expensive sensors in AVs with the top end such as Velodyne HDL-64E shown in figure 21 costing more than 50,000$. In a bid to reduce the cost of LiDAR units, different companies are exploring different design methods that are cheaper but still able to offer the same performance as the top end LiDAR units such as solid state LiDAR[1].

Figure 21: Velodyne LiDAR family. From left: Velodyne HDL-64E , HDL-32E , VLP-16

- **Cameras** - Cameras mounted on the vehicle are used for classification and identification of various objects on the road. Cameras can also be used to create 3D maps of the surrounding environment.By combining two cameras, a stereo image can be captured that provides depth information. Alternatively, by combining a camera and IR Laser sensor for depth estimation, RGB-D [12] images are obtained and mapped in a point cloud.

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

[1]See appendix **??**

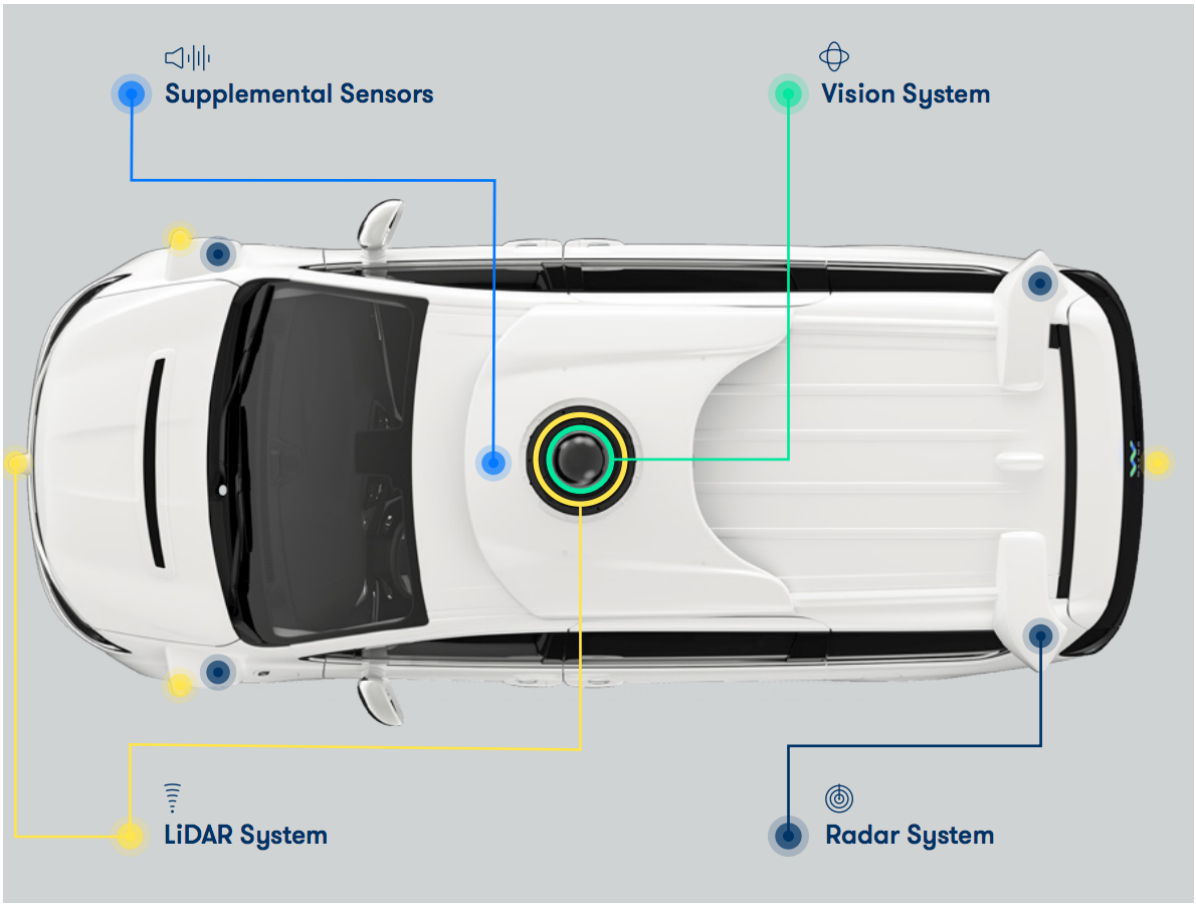


Figure 22: Components of Waymo's Self Driving Car, Source:[35]

- **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 work 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 poor weather. However, the reflectivity of these radio waves depends on the nature of objects, their size, absorbtion characteristics and the transmitting power. As such, it is may not be effective for detecting objects with low absorbtion characteristics such as pedestrians and animals.

- **Processing Unit** - In order to process all the data from sensors on the vehicle, AVs require powerful processing units to to do so in real time. 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)[32] or combinations with each other.

AVs have three main modes of operation namely,

- **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.

- **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 be able to accurately follow the plans. This involves calculation of energy and forces. 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

In order to be viable, AVs need to meet a few constraints as discussed in [16].

- **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 [21], 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). As such, an AV system should be able to react within 100ms, that is, faster than the human response time.

- **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.

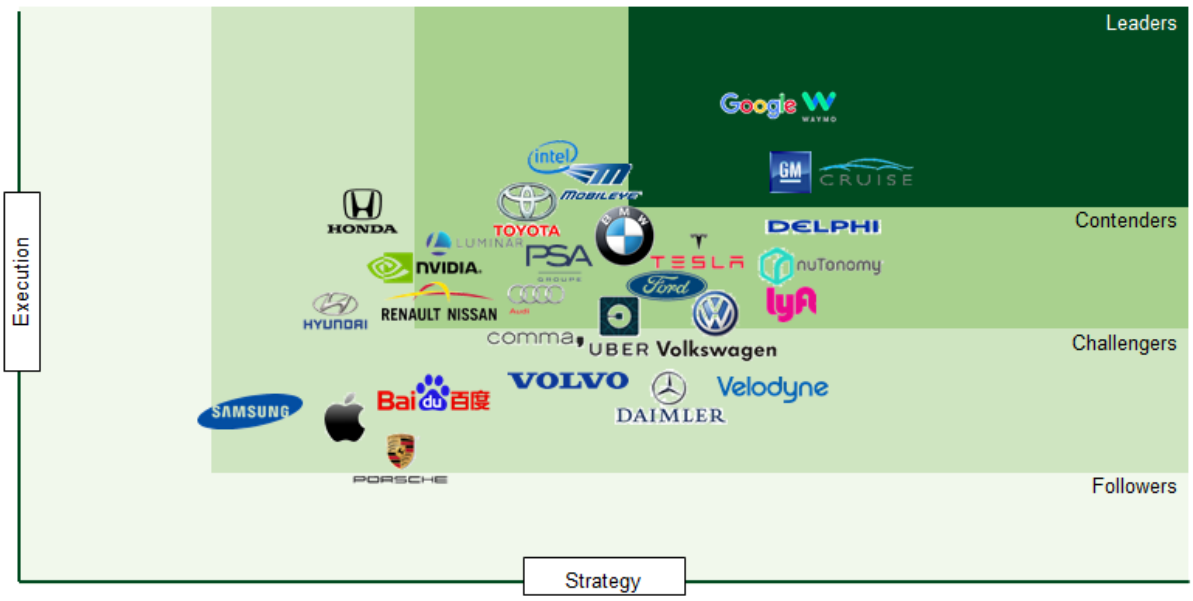


Figure 23: AV Leaderboard. Navigant Research. Source:[27]

- **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. 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. As such the configuration of these systems has to be carefully considered to ensure a reasonable driving range.

- **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. As such, additional cooling systems have to be installed in the vehicle in order to ensure this.

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

2.2.1 Camera

Mobileye[20], is one of the top contenders in the development of camera only AVs. It was recently acquired by Intel and believes 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.[13]

2.2.2 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.

2.2.3 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.

2.3 Object Detection

In this section, related research on object detection methods for AV perception will be discussed.

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[18] 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)[9] or Haar Features[34] that were used in classifiers. 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)[36], NARF[33] 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.

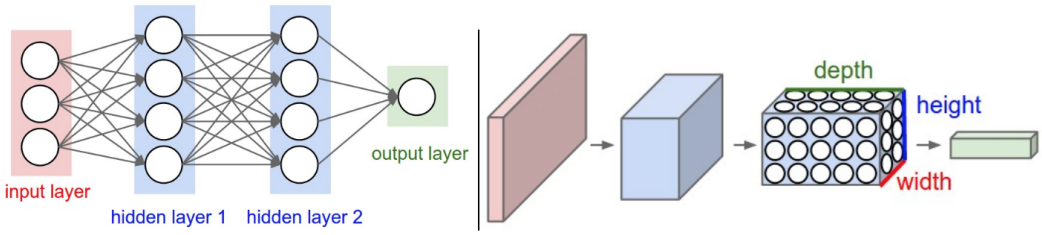
2.3.2 Object Detection using Deep Learning

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 backpropagation 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 24: NN vs CNN Source: `https://cs231n.github.io/convolutional-networks`

Most architectures consist of five main layers:

- **Input Layer** Raw image data is fed through this layer.
- **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[] 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, 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. It also required the CNN, classifier and regressor to be trained separately. Fast R-CNN[11] 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[26] sought to remove the bottleneck imposed by the using selective search to propose regions. 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 were then passed onto Fast R-CNN to classify the objects.

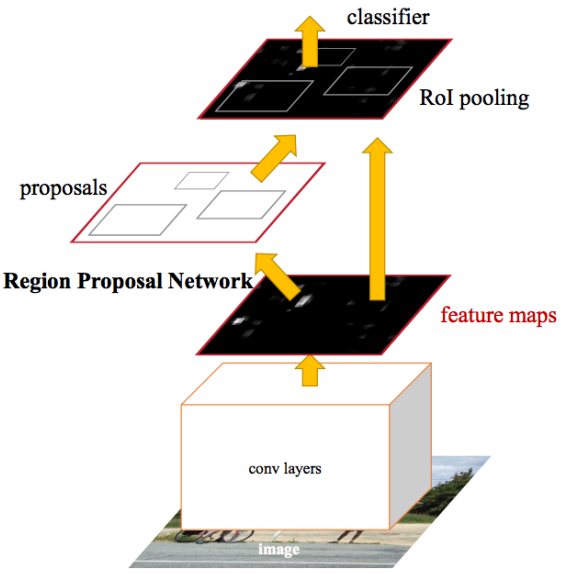


Figure 25: Faster R-CNN overview. Source:[26]

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 [6] 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.

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 [7] 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[8] 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[24] 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[19] 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++[25] 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[23].

2. **Voxel Based**
   By dividing the pointclouds into 3D voxel grids, points within these voxels can be sampled to perform feature extraction. VoxelNet [37] 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 [15] 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

2.4 Legal, Ethical and Economic Considerations

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. It has been argued that AVs will reduce this figure by

According to [10], there are no public laws that cover the use of independent autonomous vehicles in public spaces. In his article, he cites the fundamental issue as "understanding and accepting the effect of AVs as an independent action by a machine". Due to the lack of public laws on their use, he proposes extending fundamental rights such as the right to life into the framework for creating laws that cover emerging technologies that have an effect on the public, that are otherwise not accounted for in traditional laws.

Bearing this in mind, it is important to note that It has been predicted that upon mainstream

This creates a realistic baseline that can then be extended in evaluating the performance of AV systems. As such, if the risk of automation is lesser than the risk of human vehicle control then the AV would be beneficial. Consequently, the AVs are not to be considered as perfect systems but instead should be treated as systems that minimise the overall risk to the passenger and also their surroundings.

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 these 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 & Rural Affairs [1] 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.

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[31] and Random Forest classifiers [30], 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 33 was calculated.

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:

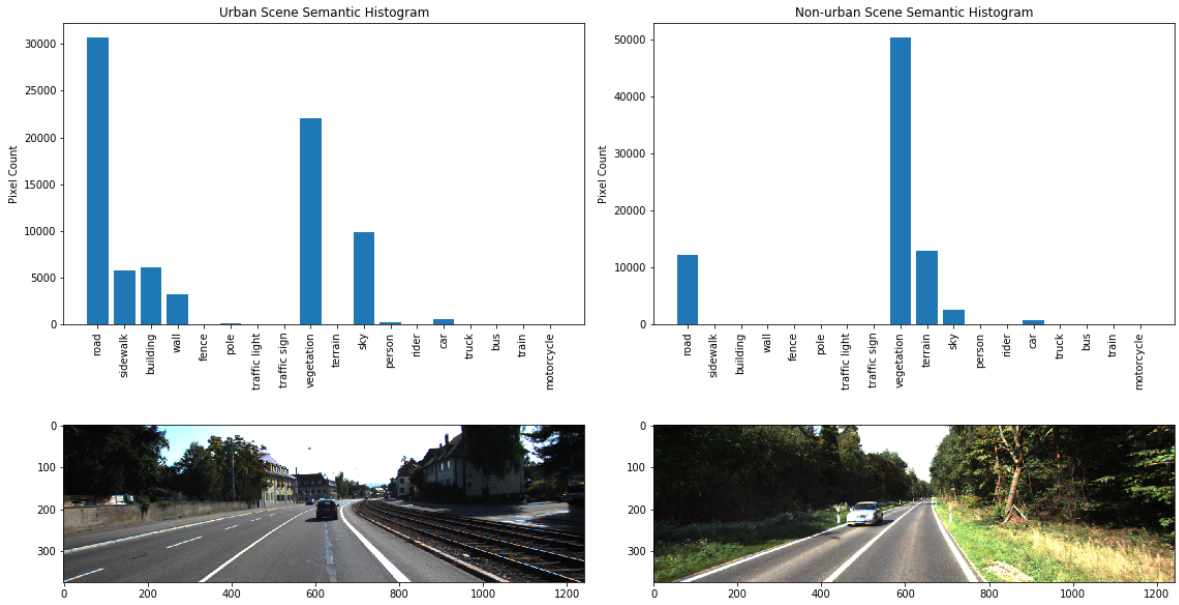


Figure 33: Semantic Histograms

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

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)[36], NARF[33] 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)[28], Signatures of Histograms of Orientations(SHOT)[29].

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[22] 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 34: 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, is it possible that LiDAR-only models can perform better or at par with multimodal methods in urban areas?

For this task,the state-of-the-art VoxelNet(LiDAR only) and AVOD(LiDAR+image) proved to be the most suitable candidates. This was influenced by the fact that they both have 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 leading researchers at Apple Inc. This 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`

3.3.1.1 Architecture Overview

VoxelNet is composed of three fundamental blocks.

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

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 [17]. This stemmed from the fact that there were few frames with pedestrians as compared to the car class and as a result the class imbalance was high in that negative anchors ≫ positive anchors.

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 consists of three 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.

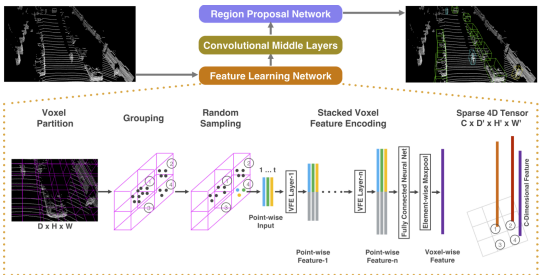


Figure 35: VoxelNet Architecture

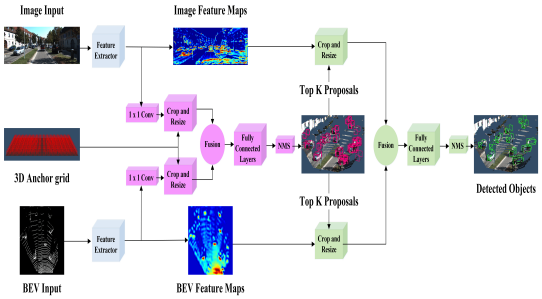


Figure 36: AVOD Architecture

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. 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 37

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.

[2]See table 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	6.640221595576416	0.30467360853659851	75.89747013826725	6.640221728458581	63.0	5.0	23.0	76.188	3433080909.0	3433080909.0	5.0
0.03370168805122375	8.045639038085938	-1.274315118789673	0.0337016894267474	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	11.215099667664233	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	-0.847423791885376	0.0818424105905633	16.16960230547166	-0.8474238034857464	2.0	12.0	29.0	16.192	3433080925.0	3433080925.0	-3.0

Figure 37: SIL Data Frame CSV

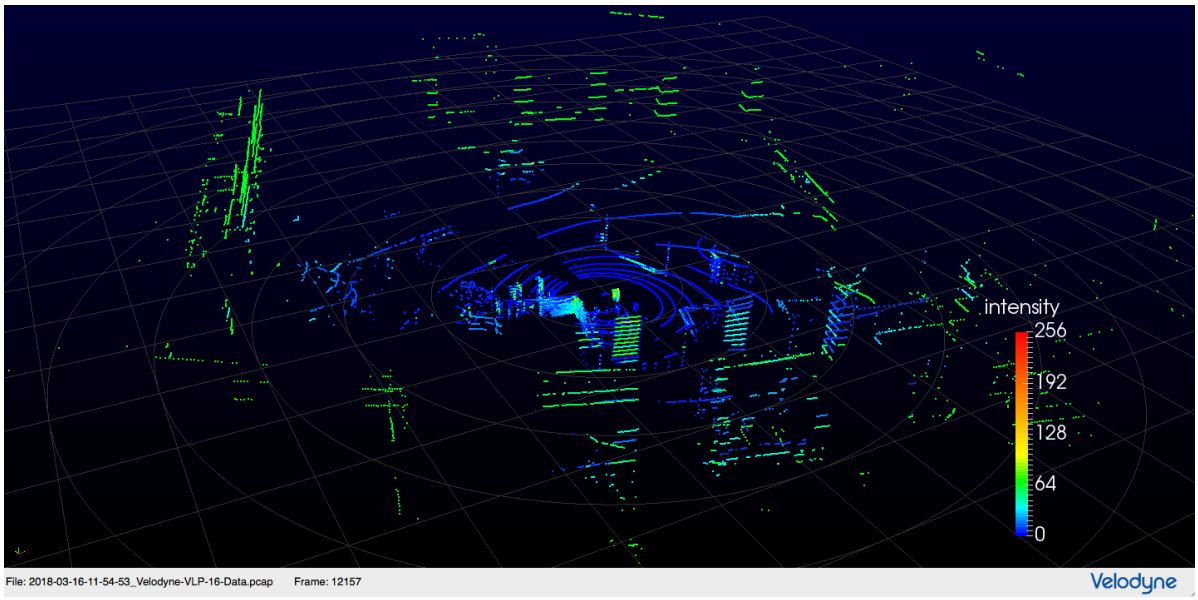


Figure 38: Veloview Frame of SIL data

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 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. 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].

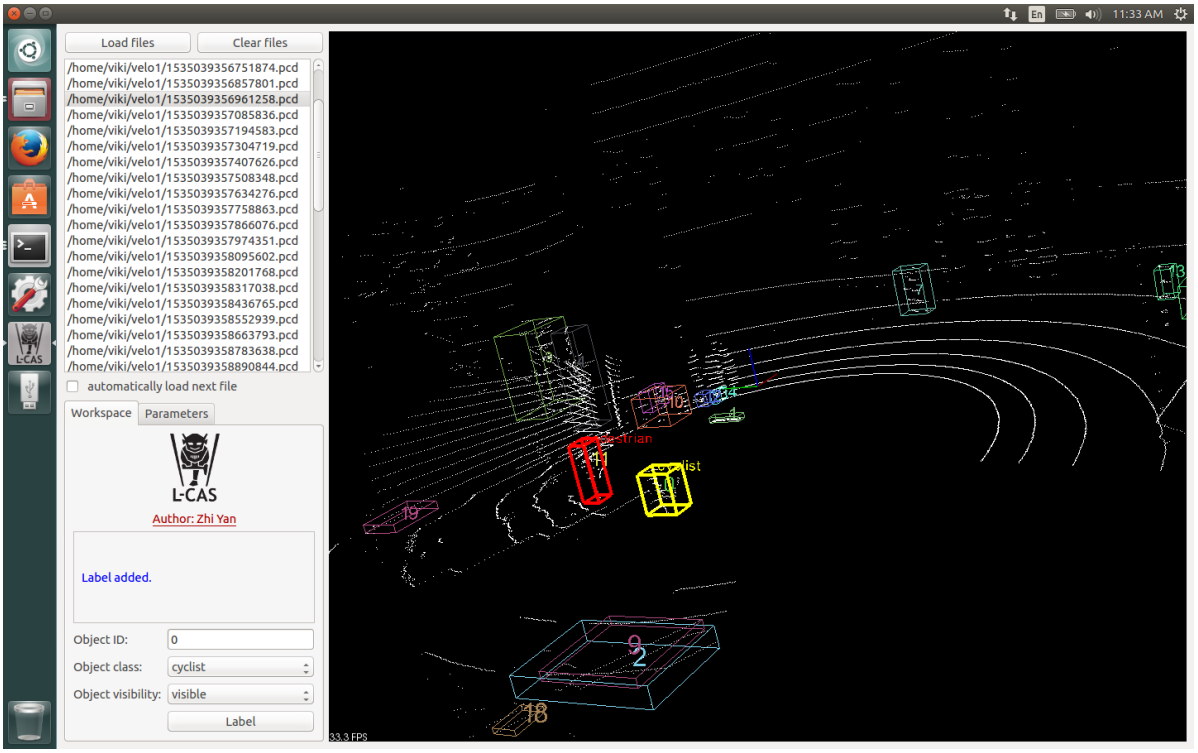


Figure 39: L-CAS Annotation Tool

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

[3]See listing A.1

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 two neural networks that could potentially perform better in different contexts. Finally, I was able to identify a potential challenge with LiDAR only models and we were able to test whether VoxelNet would still perform given a different dataset.

CHAPTER 4

RESULTS AND ANALYSIS

Following the described methodology in the previous chapter, this chapter seeks to interpret and analyse the results obtained while evaluating

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

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 classifier, 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, the majority classifier 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?

Metrics

The models were evaluated using a similar method as the KITTI evaluation protocol. The average precision(AP) of the intersection over union(IoU) of the bounding boxes was used as the main metric. 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. Only IoU values ≥0.7 were considered for the car class and ≥0.5 for the pedestrian class in calculating the AP.

GPU

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

Baseline

Each model was run on the original KITTI validation dataset in order to obtain the baseline performance shown in 43.

	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 43: Baseline

4.2.0.1 Running Each Model Exclusively

To establish if 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 44: 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 45: Pedestrian AP

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 46: Inference Time on Single NVIDIA P100 GPU

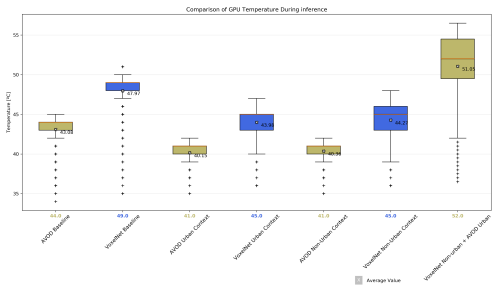


Figure 41: Temperature

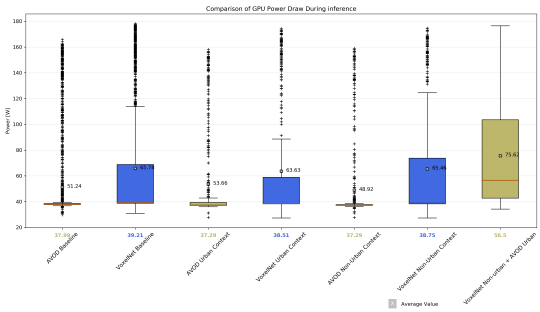


Figure 42: Power

Analysis

Car Car AVOD outperformed VoxelNet on 2D and 3D detections in both contexts. However, VoxelNet in terms of bird-eye-view performed slightly better if not relatively similar to AVOD's.

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.

GPU It is important to bear in mind that for the car class, the IoU threshold for detections is 0.5 as compared to the pedestrian IoU threshold of 0.5. As mentioned earlier while describing the model architectures, due to the fact that AVOD fuses the top proposals from the image and LiDAR point cloud feature maps, the predicted bounding boxes tend to be tighter and thus have a higher IoU. Given by the observation the pedestrian class result, it is plausible to argue that the

performance of VoxelNet would have been higher or close to AVOD's given that the IoU threshold was slightly reduced.

Running Both Models Simultaneously

Having identified that the the models perform differently in different contexts, I set out to further investigate whether it is possible to have both models running on a single GPU based on their performance in difference contexts. As such, I run AVOD using data from the urban context and VoxelNet on data from the non-urban context dataset.

Table 47: Memory

	Min	**Max**
Memory(MB)	4608.500000	14320.000000

4.2.0.2 Analysis

4.3 Is VoxelNet Performance Valid For Different Datasets?

Metrics

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

4.4 Analysis

CHAPTER 5

DISCUSSION AND FUTURE WORK

Following the analysis, we have identified that indeed it is possible to detect and characterise images with relatively high accuracy and point clouds to a much lesser extent. In addition, we have determined that the performance of multimodal and LiDAR only models are different in different contexts. Finally, we were able to realise the impact of using point clouds from different LiDAR sensors. In this section, I wish to further discuss the implications of these findings on the development of production autonomous vehicles. This chapter will be concluded by a discussion on how this work can be further improved and any points that may have not been fully explored.

determining context

performance different

validation

Despite seeing the higher temperature and power usage, this occured 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 suspended in the background. Therefore the usage would be much lesser than actually having running both. This can be implemented using a pipeli

multiple sensor for small object detection lidar for the non-urban context

Context Detection models using rnn End to End Pipeline model

CHAPTER 6

CONCLUSION

VLP-128 Solid state lidar

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

BIBLIOGRAPHY

[1] *Urban and rural area definitions for policy purposes in england and wales: Methodology (v1.0)*, GOV.UK, (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] M. BUEHLER, K. IAGNEMMA, AND S. SINGH, *The DARPA urban challenge: autonomous vehicles in city traffic*, vol. 56, springer, 2009.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

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

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

[12] P. HENRY, M. KRAININ, E. HERBST, X. REN, AND D. FOX, *Rgb-d mapping: Using depth cameras for dense 3d modeling of indoor environments*, in In the 12th International Symposium on Experimental Robotics (ISER, Citeseer, 2010.

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

[14] J. KU, M. MOZIFIAN, J. LEE, A. HARAKEH, AND S. WASLANDER, *Joint 3d proposal generation and object detection from view aggregation*, arXiv preprint arXiv:1712.02294, (2017).

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

[16] 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.

[17] 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).

[18] 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.

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

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

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

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

[23] 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).

[24] 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.

[25] 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.

[26] 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.

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

[28] 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.

[29] 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.

[30] 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.

[31] 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.

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

[33] 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.

[34] P. VIOLA AND M. JONES, *Rapid object detection using a boosted cascade of simple features*, in Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on, vol. 1, IEEE, 2001, pp. I–I.

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

[36] 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.

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