A Deep Learning Approach to the Detection and Tracking of Moving Objects in 2D Point Clouds

Hunter Schofield

A THESIS SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTERS OF SCIENCE

GRADUATE PROGRAM IN EARTH AND SPACE SCIENCE
YORK UNIVERSITY
TORONTO, ONTARIO

AUGUST 2022

© Hunter Schofield, 2022

Abstract

The detection and tracking of moving objects (DATMO) are crucial tasks that any autonomous vehicle must perform. Autonomous vehicles must detect and track all obstacles to ensure safety within the environment while also completing their tasks efficiently. In autonomous driving research, LiDAR is becoming increasingly popular due to its high resolution and accuracy. There are many state-of-the-art DATMO methods using LiDAR, however, most methods are designed for 3D LiDAR sensors. Methods that work for 2D LiDAR sensors are not as robust as their 3D counterparts or require too many computational resources to run efficiently on less powerful robots. This research presents two robust solutions to the DATMO problem based on deep learning techniques that can scale to meet a variety of hardware constraints. The first solution, detect while track (DWT), combines a convolutional neural network (CNN) with a multiple hypothesis tracking (MHT) approach and Kalman filter. The second solution, pixel predictions for future-oriented bounding boxes (PIXFOR), combines a CNN with a recurrent network architecture to solve both detection and tracking problems in a single forward pass. Both methods are experimentally validated on an unmanned ground vehicle (UGV) operating on an intersection scenario and a highway scenario using 2D point clouds collected from simulation and hardware environments. The run time performance of both methods is also validated different hardware platforms to show that the methods can scale to meet different hardware constraints. When compared to state-of-the-art DATMO methods, the newly proposed methods outperform in the object detection and tracking tasks, while operating at a faster run time on equivalent hardware.

Acknowledgements

Foremost, I would like to acknowledge my supervisor, Professor Jinjun Shan. His guidance throughout my program has been invaluable, and I am grateful for all the opportunities that I have had working in his lab. Both the quality of my research and my academic skills have flourished under Professor Shan's supervision.

I would also like to thank my lab mates Mingfeng Yuan, Yibo Liu, Marc Savoie, Hao Wang, Hassan Alkomy, Samira Eshghi, Huarong Zhao and Dr. Shuo Zhang for their guidance and assistance that they have offered. I am very fortunate to work alongside such smart and talented people.

Finally, I would like to thank my mom and dad, my friends and family, and especially my girlfriend, Violette McGreggor, for the personal support that they have provided me on my journey towards this degree. This journey has been difficult at times, and I am so thankful for the love that you have all shown me.

Table of Contents

Abstract i

Acknowledgements ii

Table of Contents v

List of Tables vi

List of Figures viii

List of Algorithms ix

List of Symbols x

1 Introduction 1

1.1 Motivation . . . 1

1.2 Problem Definition, Objectives, and Contributions . . . 3

1.3 Experimental Platform . . . 6

1.4 Organization of Thesis . . . 7

2 Object Detection in Point Clouds 9

2.1 Point Cloud Acquisition . . . 9

2.2 Traditional Approaches . . . 10

2.2.1 Segmentation and Clustering . . . 10

2.2.2 Correspondence Grouping . . . 14

2.3 A Deep Learning Approach . . . 16

2.3.1 Point Cloud Object Detection Networks . . . 16

2.3.2 Training a BEV Network . . . 17

2.3.3 PIXOR . . . 26

2.4 Newly Proposed Deep Learning Approaches . . . 32

2.4.1 PIXOR2D . . . 32

2.4.2 Extending PIXOR with Recurrent Layers . . . 34

2.5 Section Conclusions . . . 37

3 **Object Tracking in Point Clouds** **38**
3.1 Object Tracking Overview . . . 38
3.2 Approaches to the DATMO Problem . . . 39
3.2.1 Kalman Filtering . . . 39
3.2.2 Extended Kalman Filter . . . 40
3.2.3 Data Association Techniques . . . 41
3.2.4 Occupancy Grid and Obstacle Map Approach . . . 43
3.3 The Detect While Track Approach . . . 44
3.3.1 Problems with Traditional Approaches . . . 44
3.3.2 Improving Traditional Approaches with PIXOR . . . 46
3.4 PIXFOR for DATMO . . . 51
3.5 Section Conclusions . . . 53

4 **Results** **54**
4.1 Data Environments and Metrics . . . 54
4.1.1 Simulation Environment Introduction . . . 54
4.1.2 Hardware Environment Introduction . . . 55
4.1.3 Data Quality . . . 57
4.1.4 Evaluation Metrics . . . 58
4.2 Simulations . . . 60
4.2.1 PIXOR2D Results . . . 60
4.2.2 PIXFOR Results . . . 64
4.3 Experimental Verification . . . 68
4.3.1 PIXOR2D Results . . . 68
4.3.2 DWT Results . . . 72
4.3.3 PIXFOR Results . . . 74
4.4 Run Time Analysis . . . 77
4.5 Comparison with State of the Art Methods . . . 77

5 **Conclusions and Future Work** **79**
5.1 Conclusions . . . 79
5.2 Future Work . . . 80

Bibliography **82**

List of Tables

1 Memory Requirements of 3D PIXOR Network 32

2 Memory Requirements of 2D PIXOR Network 34

3 2D PIXOR Network Detection Performance 60

4 PIXFOR Tracking Performance at $t = 0$ 67

5 PIXFOR Tracking Performance at $t = 2$ 67

6 2D PIXOR Network Detection Performance on Hardware 68

7 Tracking Results of DWT 72

8 PIXFOR Tracking Performance at $t = 0$ 75

9 PIXFOR Tracking Performance at $t = 2$ 75

10 Run Time Analysis of PIXFOR and PIXOR2D Networks 77

11 Object Tracking Results of DWT, PIXFOR and other DATMO Methods 78

List of Figures

1 Velodyne HDL-64E 64 Line LiDAR [1] 2
2 Ibeo LUX 4L 4 Line LiDAR [2] 2
3 Quanser QCar Experimental Platform [3] 6
4 Quanser QCar with Triangle Box 7
5 OptiTrack Flex 13 Camera [4] 7
6 Depiction of 2D LiDAR Scanning an Arbitrary Shaped Object Adapted From [5] . . 9
7 2D Point Cloud Top View with LiDAR Field of View Adapted From [6] 10
8 Ground Segmentation Depicting Road Points as Green and Obstacles as Red [7] . . . 10
9 KITTI Data Set Frames of Reference [8] 18
10 PIXOR Network Architecture 28
11 Modified 2D PIXOR Network 33
12 Oscillating Bounding Boxes in Sequential Predictions 35
13 Recurrent PIXOR Architecture 36
14 Intersection Scenario Example Depicting Prediction Errors of the Nearest Neighbour Search and L-Shape Feature Extractor 45
15 The Detect While Track Block Diagram 50
16 PIXFOR Architecture for DATMO 53
17 4 Car Intersection Scenario Simulation 54
18 6 Car Highway Scenario Simulation 55
19 Intersection Experimental Environment with 4 QCars 56
20 Highway Experimental Environment 57
21 Position Errors for Light and Heavy 2D PIXOR Network on Simulated Data 61
22 Yaw Error of Light PIXOR2D Network 62
23 Yaw Error of Heavy PIXOR2D Network 63
24 PIXFOR Highway Current and Future Time Step Position Errors 65
25 PIXFOR Velocity Error 65
26 PIXFOR Intersection Current and Future Time Step Position Errors 66
27 Yaw Error of Current and Future PIXFOR Network Predictions 67
28 Position Errors for Light and Heavy 2D PIXOR Network on Hardware Data 69

29 Yaw Error of Light PIXOR2D Network on Hardware Data 70
30 Yaw Error of Heavy PIXOR2D Network on Hardware Data 71
31 DWT Method Solving Bounding Box Shape and Merging Issues 72
32 Bounding Box Center Position Error of DWT Method 73
33 Bounding Box Yaw Error of DWT Method 74
34 PIXFOR Current and Future Position Errors 75
35 PIXFOR Current and Future Yaw Errors 76

List of Algorithms

1 3-d Tree Construction 13

2 Euclidean Cluster Extraction 13

3 Classification Target Generation 20

4 Line Segment Minimum/Maximum y-Coordinate 22

5 Non Maximum Suppression 30

6 Candidate Matching Algorithm 49

List of Symbols

$\hat{x}_{k|k-1}$ Kalman Filter *A Priori* State Estimation

$\hat{x}_{k|k}$ Kalman Filter *A Posteriori* State Estimation

$\hat{Y}$ Set of Predictions in a Data Set

$\mathcal{L}_{BCE}$ Binary Cross Entropy Loss

$\mathcal{L}_{BFL}$ Binary Focal Loss

$\mathcal{L}_{CE}$ Cross Entropy Loss

$\mathcal{L}_{FL}$ Focal Loss

$\mathcal{L}_{SL1}$ Smooth L1 Loss

Ω^k Set of Hypotheses at Time K

$\tilde{A}_k$ Kalman Filter State Transition Model

$\tilde{B}_k$ Kalman Filter Control Input Model

$\tilde{C}_k$ Kalman Filter Observation Model

$\tilde{G}^{ijk}$ PIXFOR Ground Truth Tensor

$\tilde{S}_k$ Kalman Filter Pre-Fit Residual Covariance Matrix

ζ Specific Measurement Cluster

B_k Set of Predictions at Time k

b_p Regression Prediction Vector

b_t Ground Truth Regression Target Vector

C^{ijk} Bounding Box Corner Prediction Tensor

C_θ^{ij} Yaw Angle Cosine Prediction Matrix

C_k Set of Candidates at Time k

C_p Bounding Box Corner Prediction Matrix

C_t Ground Truth Bounding Box Corner Matrix

c_t ConvLSTM Cell State at Time t

D_i Discretized Input Shape Tensor

F_1 Network F_1 Score

f_t ConvLSTM Forget Gate at Time t

G^{ijk} PIXOR2D Ground Truth Tensor

h_t ConvLSTM Hidden State at Time t

i_t ConvLSTM Input Gate at Time t

IoU Intersection Over Union

K_k Kalman Filter Gain

L^{ij} Object Length Prediction Matrix

L_t^{ij} Ground Truth Classification Target Matrix

M^{ijk} Mesh Tensor

$MOTA$ Multiple Object Tracking Accuracy

$MOTP$ Multiple Object Tracking Precision

n Track Vector

N_t^{ij} Input Track Matrix at Time t

N_k Set of Tracks at Time k

o_t ConvLSTM Output Gate at Time t

P Network Precision

$P_{k|k-1}$ Kalman Filter *A Priori* State Estimation Covariance Matrix

$P_{k|k}$ Kalman Filter A Posteriori State Estimation Covariance Matrix

Q_k Kalman Filter Process Noise Covariance Matrix

R Network Recall

R_k Kalman Filter Observation Noise Covariance Matrix

S_θ^{ij} Yaw Angle Sin Prediction Matrix

S_k Set of Super Candidates at Time k

u_k Kalman Filter Control Input Vector

v_k Kalman Filter Observation Noise

W^{ij} Object Width Prediction Matrix

w_k Kalman Filter Process Noise

X^{ij} Center Position x Coordinate Prediction Matrix

X_k Input Point Cloud at Time k

x_k Kalman Filter State

Y Set of Ground Truths in a Data Set

Y^{ij} Center Position y Coordinate Prediction Matrix

y_k Kalman Filter Pre-Fit Residual

$Z(k)$ Multiple Hypothesis Measurements at Time k

Z^k Multiple Hypothesis Measurements Time Series

z_k Kalman Filter Observation

1 Introduction

1.1 Motivation

Due to limitations of human drivers such as visibility restrictions or slow reaction timing, autonomous vehicles have started to become an increasingly popular area of study [9, 10, 11]. Driving is one of the most common tasks that humans collectively participate in, but these limitations can make this task dangerous [12]. Autonomous vehicles that are equipped with sensors that can perform well in all driving conditions can alleviate these dangers. Furthermore, the applications of autonomous vehicles are not limited to just road driving vehicles. There are many environments in which autonomous vehicles can be designed to operate that make them more practical than humans, often because they can operate more efficiently such as in a warehouse [13], or because the environment itself is too dangerous for a human, such as in rescue and relief operations [14]. Despite the application, a common task that any autonomous vehicle needs to perform is environmental perception. The detection and tracking of moving objects (DATMO) is one of many critical perception tasks that many autonomous vehicles need to perform. There are many approaches to solving the detection problem for a variety of sensors including cameras [15, 16], radar [17, 18], LiDAR [19, 20], as well as a combination of each, however, the tracking problem requires a sensor that can provide detailed depth information, and thus, best suited for radar or LiDAR.

In practice, the DATMO problem can be easily solved when high quality dense information about the environment is supplied. For example, the model based tracking approach proposed by [21] uses data collected from a Velodyne HDL-64E 3D LiDAR, depicted in Figure 1.

Alternatively, many approaches use multiple sensors for data collection, such as the geometric model free approach proposed by [22] which uses six 4-line LiDARs collected from the IBEO LUX 4L, depicted in Figure 2.

While these sensors are capable of providing a lot of information about their surrounding environment, there are many scenarios where they are not feasible. For instance, the Velodyne HDL-64E requires 15 V at 4 A from the power supply, which requires 60 W to operate. Similarly the Ibeo LUX 4L requires an average of 7 W of power to operate, so for six of these, the sensing platform of the

Figure 1: Velodyne HDL-64E 64 Line LiDAR [1]

Figure 2: Ibeo LUX 4L 4 Line LiDAR [2]

geometric model free approach would require 42 W. These power requirements are suitable for larger platforms that can be equipped with large batteries, such as full scale autonomous vehicles for road driving, however, these are not sufficient for smaller autonomous vehicles, such as those operating in industrial environments like warehouses.

Smaller autonomous vehicles usually perceive their environment with sensors that require less overall power, but also don't provide as dense information. Never the less, these are still important platforms, that are being increasingly used in industry. A common sensor used on these types of vehicles is a 2D LiDAR, which is very similar to it's 3D cousin in functionality, but only provide points within a plane. Unfortunately, many approaches to the DATMO problem rely on 3D geometry within a point cloud, and do not scale well, or at all, to a 2D point cloud. As such, when it comes to smaller autonomous vehicles, there are few approaches to the DATMO problem that are suitable. The work approached by [23] works well in the fixed perspective case, however, this method does not maintain tracks well in environments where the perspective of the moving objects changes rapidly with respect to the LiDAR. Motivated by this gap, this research aims to find a deep learning solutions to the detection and tracking of moving vehicles that are scalable, allowing them to operate under the constraints of various hardware platforms; from road driving vehicles, to smaller robots working in industrial settings.

1.2 Problem Definition, Objectives, and Contributions

As the name suggests, the detection and tracking problem can be subdivided into two distinct problems; the detection problem, and the tracking problem. The detection problem is concerned with identifying all objects in a specific scenario. A sub task of the detection problem is the classification task in which each object that is detected is also assigned to a category that identifies the object (e.g. car, truck, pedestrian, etc.). The tracking problem is concerned with identifying the objects throughout some time interval, and assigning an identifier to the objects, usually called a *track*, that can be used to continually identify that object throughout future time steps.

When it comes to solving the DATMO problem in point clouds, there are two main philosophies, *detect before track* (DBT) and *track before detect* (TBD). As the name suggests, DBT methods perform

the task of object detection first, then track those objects through time. TBD approaches track data points through time, then use probabilistic methods to identify which tracks should be clustered as objects.

DBT methods, such as those introduced by [24], use a point clustering approach followed by geometric model fitting to solve the detection problem. Another detection method is to identify the bounding box of an object using a deep learning framework. The self-embedded single stage detector (SE-SSD) is one such deep learning framework which can detect an object's bounding box in real time [25]. Unfortunately, the SE-SSD method depends on a 3D point cloud as an input, and would need to be greatly modified to support a 2D point cloud. However, there are other deep learning frameworks, such as PIXOR, which can also predict bounding boxes in real time, and can be easily modified to solve the detection problem in 2D point clouds [26]. Once the detections have been made, recursive filtering can be employed, such as an extended Kalman filter (EKF) to track the object's motion.

TBD methods, like the geometric model free approach presented in [22], use a discretized grid to distinguish between static and dynamic points in the point cloud. Once these points are determined, an extended Kalman filter is used to track the discretized cell grids. The grid cell tracks are then iterated over to predict which ones contain objects, and which ones are static and belong to the environment. In the following chapters, DBT and TBD approaches, as well as underlying methods that enable them, will be explored in depth.

The main problem with these methods is they often rely on an input that can densely represent the environment. As mentioned in the previous section, sensors that can provide this information are usually expensive, large, and require a lot of power to use. Furthermore, the separation of the detection and tracking tasks into two steps can create an implicit delay, and can subject the method to edge cases that can cause the method to fail. As previously mentioned, the ultimate goal of this work is to develop deep learning solutions to the DATMO problem that can scale to meet the constraints of different hardware, while being robust to problems that can arise in traditional DATMO methods. To achieve the goal, the follow objectives have been set.

i Solve the detection problem by modifying an existing convolutional neural network (CNN) for object detection in point clouds, such as PIXOR.

ii Combine a traditional tracking approach, such as Kalman filtering, with the predictions from the CNN.

iii Extend the CNN object detector by adding a recurrent layer to improve on the traditional tracking approaches by solving the DATMO problem in a single forward pass.

The following thesis will outline how these goals are met, but it is important to identify the limitations under which they are achievable. The key limitation of the proposed methods depends on the hardware, specifically the LiDAR, on which they are implemented. Different LiDAR sensors have different operating ranges, and so, the dimensions of the returned point clouds will be different. The methods proposed throughout this thesis have no dependency on specific point cloud dimensions, however, the precision of the measurements made will scale with the input size of the dimension. The degree to which the precision will scale is unknown since different LiDAR sensors will provide different information densities, but it can be assumed linear if the same point density to area ratio is maintained before and after scaling.

Another critical limitation of the proposed methods is the hardware platform on which they are implemented. While the proposed methods are designed to be scalable, there are some minimum hardware requirements. While it is not mandatory to operate the proposed methods on a GPU, it is recommended to do so on a GPU device with at least 200 Mb of contiguous memory available to achieve the same run-time performance outlined in this thesis.

Under these limitations, the following contributions will be made to achieve the goal and objectives identified above.

i This thesis will propose a new convolutional network architecture to detect objects within 2D point clouds, with a configurable scaling parameter to allow the memory usage of the network to be easily controlled so that it can be implemented on a variety of hardware platforms while still operating in real-time.

ii This thesis will propose a new multiple hypothesis tracking framework to solve the detection and tracking of moving objects task in 2D point clouds This method is capable of operating in real-time and is able to achieve a tracking performance that is comparable to state of art frameworks that operate on 3D point clouds. Being built with the newly proposed convolutional network as a

backbone, this detection and tracking framework is easy to tune to meet the constraints of various hardware platforms.

iii This thesis will propose a new recurrent convolutional network architecture that is capable of detecting and tracking moving objects in real-time within 2D point clouds. This method is a pure deep learning method and achieves a better tracking performance than the newly proposed multiple hypothesis tracking method, while also having the ability to predict object locations and tracks at future time steps. Like the newly proposed multiple hypothesis tracking method, this recurrent method is easily tuneable to meet the hardware constraints of many platforms.

1.3 Experimental Platform

The work presented in future chapters is evaluated both in a simulation environment, and on a scaled hardware platform that emulates a full scale autonomous vehicle, the Quanser QCar, depicted in Figure 3.

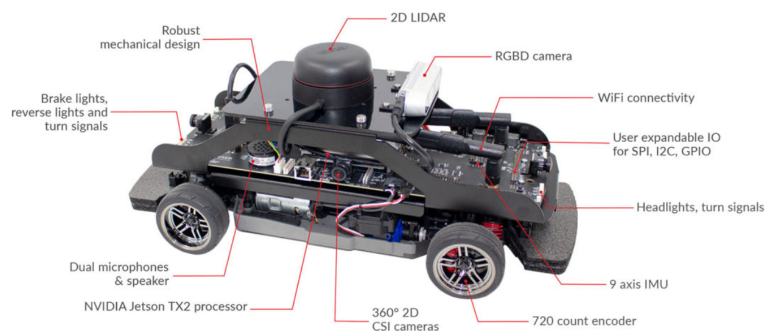


Figure 3: Quanser QCar Experimental Platform [3]

The primary LiDAR sensor used to collect the data for this work is the Rplidar-A2 located at the top of the QCar. Due to the position of the LiDAR on the vehicle, it is difficult for two of these vehicles to observe each other. This is because the plane of the LiDAR lies above the other QCars. To alleviate this problem, triangle and square boxes were constructed that can be mounted on the vehicle. These boxes minimally obstruct the field of view of the LiDAR on the vehicle itself, while providing a key feature for other vehicles to detect. Figure 4. shows the QCar with the triangle box addition.

Another part of the experimental platform is the motion capture system which is used to record the ground truth information about the hardware platform when running experiments. The motion capture system is constructed using 16 OptiTrack Flex 13 cameras, depicted in Figure 5., which are capable

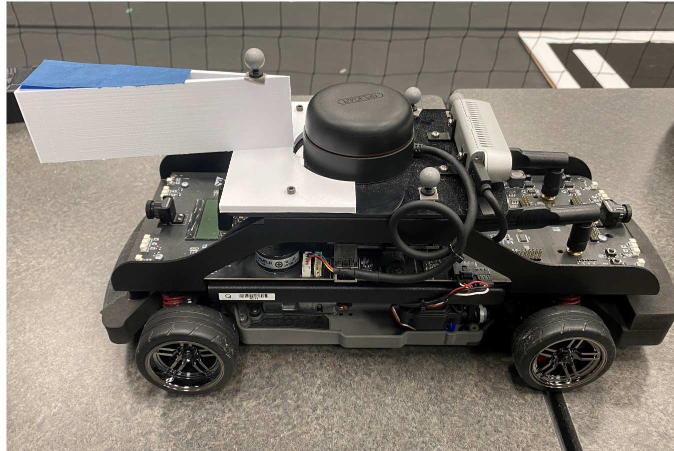


Figure 4: Quanser QCar with Triangle Box

of detecting specialized markers mounted on the QCars. By observing the markers, a rigid body that is representative of a QCar can be constructed to obtain the ground truth pose in real time.

Figure 5: OptiTrack Flex 13 Camera [4]

1.4 Organization of Thesis

The contents of this thesis are organized as follows.

Chapter 2: Object Detection in Point Clouds - Provides information about what point clouds are, how they are collected from LiDAR sensors, and various approaches for object detection within them, including both traditional and novel deep learning approaches.

Chapter 3: Object Tracking in Point Clouds - Defines the tracking problem and outlines data association and track management. Once the tracking problem is introduced, this section details some traditional and state of the art approaches for solving the tracking problem. This section also pro-

poses novel tracking methods, extending on concepts introduced from Chapter 2.

Chapter 4: Results - Gives a detailed introduction to the simulation and hardware environments used to evaluate the performance of the newly proposed methods. Provides qualitative and quantitative results of the newly proposed methods in both the aforementioned simulation and hardware environments.

Chapter 5: Conclusions and Future Work - Summarizes the newly proposed methods discussed in the thesis and begins discussion on how to extend the current status of the work done for future developments.

2 Object Detection in Point Clouds

2.1 Point Cloud Acquisition

A LiDAR is a type of laser range finder (LRF) which determines the distance to objects by emitting a laser beam, detecting the reflection, and measuring the time taken between the emission and detection. In a single pass, this process allows the sensor to identify a single point, p_i, in it's environment, relative to the lidar frame of reference, **{L}**. To obtain more useful information, most LiDAR manufactures implement the emitter and detector on top of a rotating base that rotates at a constant rate, ω. Using an integrated chip to precisely time laser emissions, a LiDAR is able to achieve a roughly constant angular resolution, α. Now, the LiDAR is able to obtain a set of points, $P = \{p_i, p_{i+1}, ...\}$, in **{L}** that represent the environment. In this case, we call P a 2D point cloud since only one laser emitted at a static angular offset. An Illustration of this process is provided in Figure 6. and Figure 7.

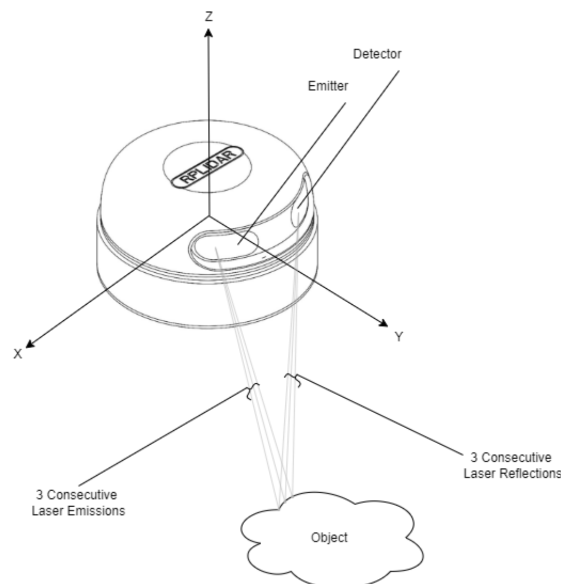


Figure 6: Depiction of 2D LiDAR Scanning an Arbitrary Shaped Object Adapted From [5]

This LiDAR design can be easily extended into three dimensions. For non solid-state LiDARs, this is done by using multiple emitters that are offset from each other by some vertical angular resolution, v. The response from each of these emitters is usually referred to as a 'line', and LiDARs with more lines can provide a highly dense point cloud.

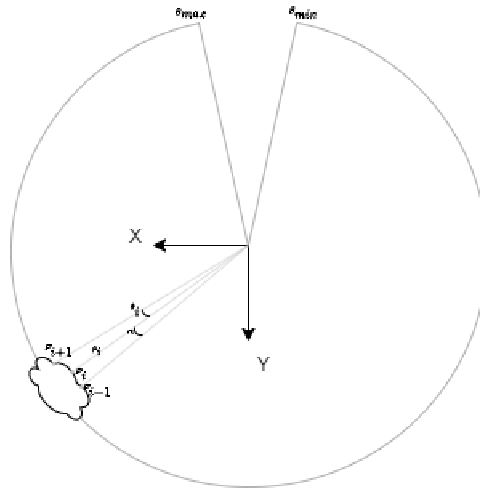


Figure 7: 2D Point Cloud Top View with LiDAR Field of View Adapted From [6]

2.2 Traditional Approaches

2.2.1 Segmentation and Clustering

Traditionally, object detection is performed in 3D point clouds and focuses on iterative methods for identifying certain geometries. There are two main strategies commonly employed in point cloud object detection, segmentation and clustering. In segmentation, each point in the point cloud is associated with an identifier that categorizes that point as part of some object. A popular example of segmentation in point clouds is ground segmentation where a point cloud is segmented into two categories, points that represent the ground plane, and points that do not. An example of point cloud segmentation to distinguish the ground from obstacles is provided in Figure 8.

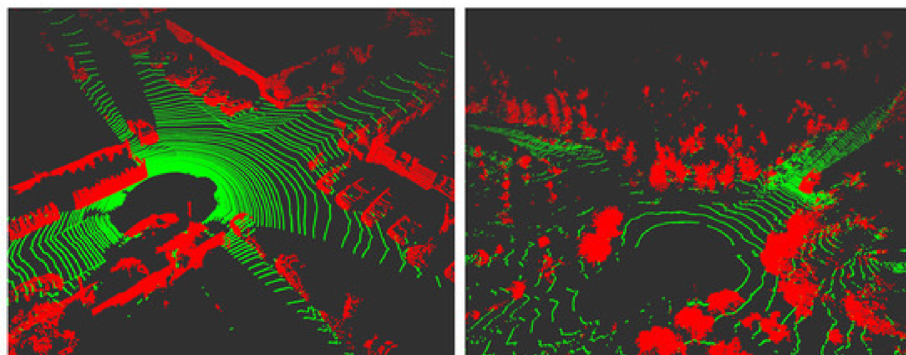


Figure 8: Ground Segmentation Depicting Road Points as Green and Obstacles as Red [7]

A common way to perform segmentation in point clouds is through the RANSAC [27] algorithm, which stands for random sample and consensus. This method is an iterative model fitting approach for finding a best fit in a data set that has many outliers. To solve the problem of ground segmentation for some point cloud, P, the RANSAC algorithm is as follows. In most cases, the ground is considered a flat plane, so begin by defining a plane in Cartesian coordinates.

$$ax + by + cz + d = 0 \tag{1}$$

In order to solve this equation for the variables a, b, c and d, we need at least three points on the plane. The first step is to select a random set of three points from the point cloud. Using these three points, the plane equation can be solved.

$$\begin{aligned}
a &= (y_2 - y_1)(z_3 - z_1) - (z_2 - z_1)(y_3 - y_2) \\
b &= (z_2 - z_1)(x_3 - x_1) - (x_2 - x_1)(z_3 - z_2) \\
c &= (x_2 - x_1)(y_3 - y_1) - (y_2 - y_1)(x_3 - x_2) \\
d &= -ax - by - cz
\end{aligned} \tag{2}$$

Next, a threshold distance, D, is defined. Then, every point in the point cloud is iterated, and the distance between each point and the plane is computed. If the distance between a specific point and the plane is less than D, then this point is added to the set of inlier points, I.

$$I = \left\{ (x, y, z) \in P \mid \frac{ax + by + cz + d}{\sqrt{a^2 + b^2 + c^2}} < D \right\} \tag{3}$$

This process is repeated for some number of iterations, N, and the plane parameters are chosen as the parameters which yield the largest inlier set. That is, for some plane parameter space $S \in \mathbb{R}^4$ that is obtained by randomly sampling three points in P over N iterations, the plane parameters are chosen as those which maximize the cardinality of I.

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \arg\max_{s \in S} |I| \tag{4}$$

With this plane equation, the input point cloud can be separated into a ground plane (inlier set, I), and a non ground plane (outlier set, $O = P \setminus I$).

By itself, this ground segmentation does not solve the problem of object detection, however, it is an important first step. This is because the input point cloud is now refined into an outlier set that can be parsed with a clustering algorithm to obtain bounding boxes around any object that deviates from the ground plane. There are many different types of clustering algorithms that typically fall into two categories; hierarchical or partitioning. When working with point clouds, clustering algorithms are usually hierarchical as these methods begin with an initial cluster (the whole point cloud) and find an optimal amount of sub clusters. Partitional methods such as K-means clustering [28] usually require additional knowledge about the data, such as how many clusters exist that need to be identified. Since this is unknown for the task of object detection, it is better to use a hierarchical approach.

One such traditional hierarchical approach to clustering in point clouds is the Euclidean cluster extraction method. The Euclidean cluster extraction method begins by formatting the input point cloud, O, into a k-d tree data structure [29], where $k = 3$ since the input point cloud is in 3 dimensions. This structure is useful for object detection because the average search run time is $O(\log n)$ with a worst case run time of $O(n)$.

The k-d tree is a type of binary tree where each node represents a k-dimensional point. However, instead of having the left node represent a smaller value than its parent and the right node represent a larger value as with a binary tree, the k-d tree cycles through the dimensions used at each layer of the tree. This means for the 3D case, if the root defines an x-axis aligned plane, then the child nodes will be split based on the y-axis, the grand children based on the z-axis, before cycling back to the x-axis for the great grand children, and so on. A 3-d tree can be constructed from O using the recursive algorithm defined in algorithm 1.

Upon completion of this algorithm a k-d tree representation, $\hat{O}$, is obtained from the input point cloud O, with $k = 3$. Using this point cloud representation, the Euclidean cluster extraction algorithm can be performed as defined in algorithm 2, adapted from [30], to obtain a set of point clusters, C, which will represent the objects in the environment.

Now, a set of clusters, C, which represent objects in the environment is obtained. However, these

Algorithm 1: 3-d Tree Construction

```
Input: O
Output: Ô

depth ← get current depth
axis ← depth mod 3
median ← int(|O| / 2)
sort O by axis

initialize new node in tree
node.location ← O[median]
node.left← 3-d tree construction for O on interval [0, median]
node.right ← 3-d tree construction for O on interval (median, |O|]
return node
```

Algorithm 2: Euclidean Cluster Extraction

```
Input: Ô, d_th
Output: C

C ← ∅
Q ← ∅
P̂ ← ∅
for p ∈ Ô do
    add p to Q
    add p to P̂
    for p_q ∈ Q do
        search for the set P^k of point neighbours of p_q within some distance d_th
        for p^k ∈ P^k do
            if p^k ∉ P̂ then
                add p^k to Q
                add p^k to P̂

    if |Q| > 0 then
        add Q to C
        Q ← ∅

return C
```

clusters can be further refined. Additional parameters can be set in the Euclidean cluster extraction algorithm to only add clusters to C if they contain at least some minimum, or at most some maximum quantity of points. This allows the filtering of small clusters that can arise due to sensor noise, or large clusters which are more representative of the environment (e.g. a tree, wall, etc.) but not the objects of interest in the environment.

Together, the algorithms identified in this section are good at quickly identifying objects in point clouds. However, they are more general object detectors and cannot detect specific objects unless a detailed model is provided to compare each cluster against. Even then, the accuracy of the object detection when a model is provided is not the greatest since this algorithm does not account for sensor noise well. Furthermore, this method cannot solve the problem of object recognition, where an object is both located and identified. To solve this problem, the clustering algorithm identified in this section can be enhanced by coupling it with a method known as correspondence grouping.

2.2.2 Correspondence Grouping

Correspondence grouping is a type of point cloud clustering method where each cluster has a point to point correspondence with a 3D descriptor that represents an object in the current point cloud environment. There are two popular algorithms for correspondence grouping, 3D Hough voting [31], and geometric consistency [32]. In this section, the 3D Hough voting methodology will be introduced since it can perform well even for objects that are partially occluded, and is the default method used by the popular point cloud library (PCL).

The 3D Hough voting algorithm works by voting for features such as edges and corners in the parameter space of the shape that is being detected using the generalized Hough transform (GHT). In GHT, feature voting occurs for a specific position, orientation and scale factor for the shape or object being sought. The method is described as follows for 3D point clouds. Assume a model exists of some object to detect within a 3D point cloud with a unique reference point, C^M and a set of feature points F^M. First, a vector is computed between the model reference point and each feature.

$$V_i^M = C^M - F_i^M \tag{5}$$

To make this vector invariant to rotaion and translation, it needs to be transformed in the coordinates computed on F_i^M.

$$V_{i,L}^M = R_{G,L}^M \cdot V_i^M \tag{6}$$

Where $R_{G,L}^M$ is a rotation matrix with each row representing a unit vector in the local reference frame of the feature point F_i^M. Finally, each feature F_i^M is associated with it's vector $V_{i,L}^M$. Now that the model is defined, the point cloud is searched for corresponding scene features ($F_j^S \leftrightarrow F_i^M$). This is done by employing the previously defined Euclidean clustering algorithm, but with an additional criterion that enforces each output cluster must have similar associations between F_i^S and $V_{i,L}^S$.

With this, a specific model can be accurately identified in a 3D point cloud environment. This method can be further refined to get a more precise pose of the object by employing a few iterations of the popular iterative closest point (ICP) [33] algorithm between the identified scene cluster and the model.

The traditional point cloud object detection methodologies identified in this section are good at solving the general detection on average, but their performance is limited. For example, detections might be missed in cases where the point cloud environment contains an object that is not well represented by the model being used to identify those types of objects. This can often occur if an object is far from the LiDAR when the point cloud is collected, resulting in a sparse representation of the object. Furthermore, while the Hough voting method works better than geometric consistency in partially occluded environments, the performance is still limited. These types of issues are hard to solve because they often have a small number of occurrences in a data set, and the characteristics of any single occurrence is often unique. That is, it is unlikely for two sparse object representations, or two possible object occlusions to be the same, which makes it difficult for model based methods to perform the detection task.

This is the motivation for pursuing a deep learning approach in the work outlined in this thesis. Since large data sets can contain samples for most possible object representations, a deep neural network can be employed to learn how to detect objects even in these edge cases.

2.3 A Deep Learning Approach

2.3.1 Point Cloud Object Detection Networks

The previous section discussed traditional non-deep learning approaches to solving the object detection task in 3D point clouds. This section will identify existing deep learning architectures that are capable of solving the detection task in real time, with a goal of finding an architecture that can be modified to solve the object detection task in 2D point clouds as well.

Before discussing the various networks, it is important to identify how these networks are evaluated. The most popular data set for evaluating point cloud object detection algorithms for autonomous driving applications is the KITTI data set [34]. This data set contains thousands of point clouds of various driving scenarios along with labels identifying all cars, trucks, busses, pedestrians, etc. in the environment as well as the difficulty to detect these objects. These labels can be used to determine the error of a network prediction, which is then used to update the weights of the network via backwards propagation.

From an extensive study in [35], there are two main categories of deep learning methods for object detection in point clouds, region based proposal networks and single shot networks. Region proposal methods propose an initial set of candidate regions that could contain objects and then extract features within these regions to determine specific labels for the objects contained. Single shot networks use a single stage network to predict probabilities of the classes and use regression layers to precisely identify the bounding boxes that are paired with the class probabilities. Due to the single stage design, these networks need minimal post processing and are able to run much quicker than region proposal methods, making them a good choice for autonomous vehicles. Thus, this architecture is chosen as the basis of this work.

There are three subcategories of single shot networks; discretization based methods, point based methods, and bird's eye view (BEV) based methods. The first object detection networks for point clouds was a discretization based method, VeloFCN [36]. Here, a point cloud was converted into a 2D point map, and a 2D FCN was used to predict the bounding box of objects in the point cloud. Being the first of its kind, it only had a run time around 1 fps. A more recent and powerful discretization

architecture is PointPillars [37] which uses PointNet [38] to learn features within vertical columns (pillars) of a 3D point cloud. This method vastly improves on the detection perfomance of VeloFCN and can achieve a 62 fps run time making it one of the fastest point cloud detection networks; however, due to its nature, it is dependent on a 3D input.

The second subcategory, point based methods, are a less popular approach to object detection in point clouds. In these methods, the input to the network is the raw point cloud scan, which means that these methods do not need much prepossessing. A notable point based method is the 3D single shot detector (3DSSD) network [39]. This network removes traditional feature propagation layers that many other networks employ, and uses a fusion sampling strategy for distance and feature predictions, which improves its run time performance. The 3DSSD network then adds a candidate generation layer before the header layer to identify bounds around the representative points in the point cloud. The 3DSSD network out preforms the PointPillars network in terms of accuracy on the KITTI benchmark; however, its runtime performance is not as good at 25 fps.

Unlike the other methods, BEV-based methods can operate on either 3D or 2D inputs by their nature since the input is flattened onto one plane. The PIXOR network [26] discretizes the input point cloud into equally spaced cells that are then passed through an FCN. A prediction of the class of object, its heading angle, center position, width and length are provided for each cell. Using non-max suppression, a final best approximation for an object and it's bounding box are provided. This method achieves good performance on the KITTI benchmark while running at 29 fps. It is because of this good performance and the ability for this architecture to be adapted to solve the object detection in 2D point clouds that makes PIXOR suitable choice for the basis of this research. Before doing an in depth analysis on the PIXOR architecture, the next section will outline how to train a BEV network.

2.3.2 Training a BEV Network

The training environment for a BEV-based network like PIXOR is a supervised learning environment where a point cloud is input to the network, and the output is compared against the known labels for the input point cloud to compute the loss. Using an entry from the KITTI data set as an example, the network is trained as follows. First, a text file for the label is parsed to load all the ground truth information in the environment. This text file includes detailed information about the object, but for

the purpose of training PIXOR, the important information is the object class, the bounding box coordinates in the camera coordinate system, the 3D object dimensions, the 3D object coordinates in the camera coordinate system, and the object yaw in the camera coordinate system.

Next, the ground truth information needs to be transformed from the camera reference frame to the LiDAR reference frame. Figure 9 illustrates the frames of reference used in the KITTI data set.

Figure 9: KITTI Data Set Frames of Reference [8]

Since the BEV bounding box is in the LiDAR x-y plane, the transformation is simple and can be done in two steps. First, a 90 degree anti-clockwise rotation about the camera y-axis is conducted. Now, the carried x-axis overlaps with the LiDAR x-axis, and the carried z-axis overlaps with the LiDAR y-axis. Thus, the carried x coordinate can be used as the LiDAR x-axis, and the carried z-axis can be used as the LiDAR y-axis. Now the ground truth regression target, b_t, and ground truth bounding box corners, C_t, can be defined in the LiDAR coordinate system.

The ground truth regression target, b_t, for an object is a vector that contains information for locating that object in the BEV LiDAR frame. This vector contains the sine and cosine values of the heading angle, θ, the center position of the object in the LiDAR frame, (x_c, y_c), and the length, l, and width, w, of the object.

$$b_t = \begin{bmatrix} \sin\theta \\ \cos\theta \\ x_c \\ y_c \\ w \\ l \end{bmatrix} \tag{7}$$

From this vector, the matrix containing the corners of the bounding box, $C_t \in \mathbb{R}^{4x2}$, can be computed.

$$C_t = \begin{bmatrix} x_c - \frac{1}{2}(l\cos\theta + w\sin\theta) & y_c - \frac{1}{2}(l\sin\theta - w\cos\theta) \\ x_c - \frac{1}{2}(l\cos\theta - w\sin\theta) & y_c - \frac{1}{2}(l\sin\theta + w\cos\theta) \\ x_c + \frac{1}{2}(l\cos\theta + w\sin\theta) & y_c + \frac{1}{2}(l\sin\theta - w\cos\theta) \\ x_c + \frac{1}{2}(l\cos\theta - w\sin\theta) & y_c + \frac{1}{2}(l\sin\theta + w\cos\theta) \end{bmatrix} \tag{8}$$

To evaluate the classification output of a network against the ground truth, the locations of the ground truth objects need to be identified in the prediction space of the network. To do this, C_t must be down sampled by a factor of 4, and then discretized using the same resolution as the input point cloud. The result will be the regions of the output space that are occupied by a ground truth object. The output space can then be iterated and the coordinates of the points contained within the object regions can be saved following the method defined in algorithm 3.

The algorithm works as follows. First, some setup steps are performed. A set containing the down sampled, discretized left-most (l), right-most (r), top-most (t), and bottom-most (b) coordinates of C_t are provided to the algorithm along with the output dimension, D_o. Next, the bottom and top-most coordinates are checked to determine which one is closer to the left most coordinate. Then, the left-most coordinate and right-most coordinate are clamped to the dimensions of the output space. Now the main logic of the algorithm can be performed.

Each vertical column of cells in between the left-most and right-most is iterated. If the x-coordinate of

Algorithm 3: Classification Target Generation

```
Input: {l, r, b, t}, D_o
Output: L_t

if b_x ≤ t_x then
    m_1 ← b
    m_2 ← t
else
    m_1 ← t
    m_2 ← b

x_1 = max{⌈l_x⌉, 0}
x_2 = min{⌊r_x⌋, D_{o,x}}
for x ∈ [x_1, x_2] do
    if x < m_{1,x} then
        y_1 ← SegmentMin(l_x, l_y, b_x, b_y, x)
        y_2 ← SegmentMax(l_x, l_y, t_x, t_y, x)
    else if x < m_{2,x} then
        if m_{1,y} < m_{2,y} then
            y_1 ← SegmentMin(b_x, b_y, r_x, r_y, x)
            y_2 ← SegmentMax(l_x, l_y, t_x, t_y, x)
        else
            y_1 ← SegmentMin(l_x, l_y, b_x, b_y, x)
            y_2 ← SegmentMax(t_x, t_y, r_x, r_y, x)
    else
        y_1 ← SegmentMin(b_x, b_y, r_x, r_y, x)
        y_2 ← SegmentMax(t_x, t_y, r_x, r_y, x)
    y_1 ← max{y_1, max{⌈b_y⌉, 0}}
    y_2 ← min{y_2, min{⌊t_y⌋, D_{o,y}}}
    for y ∈ [y_1, y_2] do
        L_t[x, y] = 1
    return L_t
```

the column is less than the x-coordinate of the first midpoint, then the y-coordinate region of interest is the vertical range of coordinates, $[y_1, y_2]$, between the two line segments from the left-most point to the bottom-most point, and the left-most point to the top-most point at the current x-coordinate. If the x-coordinate of the column is greater than the x-coordinate of the first midpoint, but less than the x-coordinate of the second mid point, then there are two cases. If the first mid point is lower than the second midpoint, then the y-coordinate region of interest is the vertical range of coordinates between the two line segments from the bottom-most point to the right-most point and the left-most point to the top-most point at the current x-coordinate. If the first mid point is higher than the second midpoint, then the y-coordinate region of interest is the vertical range of coordinates between the two line segments from the left-most point to the bottom-most point, and the top-most point to the right-most point at the current x-coordinate. Finally, if the x-coordinate of the column is greater than the x-coordinate of both the x-coordinates, then the y-coordinate region of interest is the vertical range of coordinates between the two line segments from the bottom-most point to the right-most point and the top-most point to the right-most point at the current x-coordinate. The output of the algorithm is then a map of coordinates in the output space that label the ground truth classification target, L_t. The algorithms to find the y-coordinate region of interest for each case are provided in algorithm 4.

Now that the classification target is defined for one object, it is possible to define the ground truth tensor, G_t, for that particular object. The ground truth tensor has the same dimension as the network output. The first two dimensions in this tensor encode spatial information, and the last dimension represents the vector that concatenates the classification target, L_t, with the regression target, b_t.

$$G_t^{ij} = L_t^{ij} \oplus b_t = \begin{bmatrix} L_t^{ij} \\ \sin\theta \\ \cos\theta \\ x_c \\ y_c \\ w \\ l \end{bmatrix} \tag{9}$$

For a set of ground truth targets, T, such as the labels provided by the KITTI data set, the overall ground truth tensor, G, can be defined as the union of all the specific ground truth tensors, G_t.

Algorithm 4: Line Segment Minimum/Maximum y-Coordinate

```
procedure SegmentMin(x_0, y_0, x_1, y_1, x)
if x_0 = x_1 then
    return ⌊y_0⌋
else
    m ← (y_1 − y_0)/(x_1 − x_0)
    if m > 0 then
        return ⌊(y_0 + m(x − x_0))⌋
    else
        return ⌊(y_0 + m(x − x_0 + 1))⌋
end procedure
procedure SegmentMax(x_0, y_0, x_1, y_1, x)
if x_0 = x_1 then
    return ⌈y_1⌉
else
    m ← (y_1 − y_0)/(x_1 − x_0)
    if m > 0 then
        return ⌈(y_0 + m(x − x_0 + 1))⌉
    else
        return ⌈(y_0 + m(x − x_0))⌉
end procedure
```

$$G = \bigcup_{t \in T} G_t \tag{10}$$

To evaluate the network prediction, P, against the ground truth, G for a single input, multiple loss functions are used. PIXOR uses the focal loss function [40] to evaluate the classification output. This function is an extension of cross entropy loss. In the case where there is only one class of objects to detect, the binary cross entropy can be used, which is defined as follows.

$$\mathcal{L}_{BCE,i}(\hat{y}, y) = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})) \tag{11}$$

where $y = G^{ij1}$ is the ground truth classification tensor, and $\hat{y} = P^{ij1}$ is the predicted classification tensor. Since this is the binary classification case, y can only be either 0 or 1. The binary cross entropy cost function for the data set with size n is then defined as follows.

$$J_{BCE}(\hat{Y}, Y) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}_{BCE,i}(\hat{y}_i, y_i) = -\frac{1}{n} \sum_{i=1}^{n} (y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)) \tag{12}$$

Here, $\hat{Y}$ is the set of all predictions made during training, and Y is the set of all ground truths with $\hat{y} \in \hat{Y}$ and $y \in Y$. For the case where multiple classes are being identified, the binary cross entropy loss function can be extended to compute the loss for C classes.

$$\mathcal{L}_{CE,i}(\hat{y}, y) = -\sum_{c=1}^{C} \left(y_c \log\left(\frac{e^{\hat{y}_c}}{\sum_{j=1}^{C} e^{\hat{y}_j}} \right) + (1 - y_c) \log\left(1 - \frac{e^{\hat{y}_c}}{\sum_{j=1}^{C} e^{\hat{y}_j}} \right) \right) \tag{13}$$

For the cross entropy function, y_c is the ground truth class probability distribution, $e^{\hat{y}_c}$ is the predicted probability distribution of one class, c, and $p(\hat{y}) = \frac{e^{\hat{y}_c}}{\sum_{j=1}^{C} e^{\hat{y}_j}}$ is the predicted probability distribution of the model for all classes. The cross entropy cost function takes the same form as the binary cross entropy cost function.

$$J_{CE}(\hat{Y}, Y) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}_{CE,i}(\hat{y}_i, y_i) = -\frac{1}{n} \sum_{c=1}^{C} \sum_{i=1}^{n} \left(y_c \log\left(\frac{e^{\hat{y}_c}}{\sum_{j=1}^{C} e^{\hat{y}_j}} \right) + (1 - y_c) \log\left(1 - \frac{e^{\hat{y}_c}}{\sum_{j=1}^{C} e^{\hat{y}_j}} \right) \right) \tag{14}$$

The focal loss is then an extension of either the balanced cross entropy or binary cross entropy loss function. The balanced cross entropy loss introduces a hyperparameter, α, to balance the importance of positive and negative examples in the training set. However, by itself, this balancing parameter does not differentiate between easier and harder to classify samples in the data set. The focal loss

function adds a tunable modulating parameter to down weight easy samples and focus training on difficult samples in the data set. In a point cloud data set, the easy examples can be considered objects that are close to the LiDAR and are represented by a dense point cloud, while hard examples are further away and sparsely represented. The focal loss for the multiclass classification problem is defined as follows, first express the binary cross entropy loss as a piecewise function depending on the ground truth probability for a single class, y_c.

$$\mathcal{L}_{BCE,i}(p, y_c) = \begin{cases} -y_c \log(p) & y_c = 1 \\ -(1 - y_c) \log(1 - p) & \text{otherwise} \end{cases} \tag{15}$$

For a single class, y_c can only take on values of 0 and 1, so the above simplifies to the binary case.

$$\mathcal{L}_{BCE,i}(p, y_c) = \begin{cases} -\log(p) & y_c = 1 \\ -\log(1 - p) & \text{otherwise} \end{cases} \tag{16}$$

For ease of notation, the focal loss function defines $p_t(p, y_c)$.

$$p_t(p, y_c) = \begin{cases} p & y_c = 1 \\ 1 - p & \text{otherwise} \end{cases} \tag{17}$$

This allows the cross entropy loss to be simplified in notation.

$$\mathcal{L}_{BCE,i}(p, y_c) = \mathcal{L}_{CE,i}(p_t) = -\log(p_t) \tag{18}$$

Now, the binary focal loss function can be defined as an extension of the binary cross entropy loss by adding the balancing hyperparameter and the modulating parameter as follows.

$$\mathcal{L}_{BFL,i} = -\alpha(1 - p_t)^{\gamma} \log(p_t) \tag{19}$$

Finally, to get the focal loss in the multiple class case, the individual focal losses for each class can be summed.

$$\mathcal{L}_{FL,i} = -\alpha \sum_{c=1}^{C} (1 - p_t)^{\gamma} \log(p_t) \tag{20}$$

As p_t approaches 1, the modulating factor goes to 0 and the loss is down weighted for well classified

examples, allowing the network to focus training on the examples that are not well classified. The hyperparameter, γ, can be tuned to vary the affect of the modulation. With $\gamma = 0$, the focal loss becomes the balanced cross entropy loss. The overall cost function for the focal loss is then defined as follows.

$$J_{FL}(\hat{Y}, Y) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}_{FL,i} = -\frac{\alpha}{n} \sum_{i=1}^{n} \sum_{c=1}^{C} (1 - p_t)^{\gamma} \log(p_t) \tag{21}$$

Upon evaluating the classification output of the network, the regression output of the network is evaluated. The regression output is evaluated using the smooth L1 loss function, which was first introduced in fast R-CNN [41]. The smooth L1 loss is similar to the L1 loss function, however it adds a hyperparameter, β. If the L1 component is less than the beta term, the smooth L1 loss function replaces the component with a quadratic term to smooth the loss when it is close to 0. This allows the loss to continue to converge after the network has already learned some features. The smooth L1 loss function for one sample is defined as follows.

$$\mathcal{L}_{SL1,i}(\hat{y}, y) = \begin{cases} \frac{(\hat{y}-y)^2}{2\beta} & |\hat{y} - y| < \beta \\ |\hat{y} - y| - \frac{1}{2} & \text{otherwise} \end{cases} \tag{22}$$

Since this loss function is for the regression output, $y = \begin{bmatrix} G^{ij2} & G^{ij3} & ... & G^{ij7} \end{bmatrix}^T$ is the ground truth regression target tensor and $\hat{y} = \begin{bmatrix} P^{ij2} & P^{ij3} & ... & P^{ij7} \end{bmatrix}^T$ is the predicted regression tensor. The smooth L1 cost function is then defined as follows.

$$J_{SL1}(\hat{Y}, Y) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}_{SL1,i}(\hat{y}_i, y_i) = \frac{1}{n} \sum_{i=1}^{n} \frac{(\hat{y} - y)^2}{2\beta} H(|\hat{y} - y| - \beta) + (|\hat{y} - y| - \frac{1}{2})(1 - H(|\hat{y} - y| - \beta)) \tag{23}$$

Here, the Heaviside function, $H(x)$, is used to sum the loss function for the two possible cases, and the input x is $|\hat{y} - y|$. Now, all the loss functions required to train the network have been defined. The overall cost function for the network can be setup by combining the focal cost function and the smooth l1 cost function.

$$J(P, G) = J_{FL}(P, G) + J_{SL1}(P, G) \tag{24}$$

With the training process for a BEV network like PIXOR well defined, the overall architecture of the PIXOR network can be explored.

2.3.3 PIXOR

The PIXOR network works by taking an input point cloud and representing it as a bird's eye view by discretizing points in different vertical layers into different input channels. This allows for the 3D information to remain, but now, 2D convolution operations can be conducted on each channel which greatly improves performance compared to 3D convolutions. An additional channel is added to the input where each cell in this channel represents the sum of the reflectances of the column of cells below it.

The input of the network is defined as follows by [26]. Assume there is some input point cloud with the physical dimension $l \times w \times h$. The points within the point cloud are then discretized by some chosen resolution, $d_l \times d_w \times d_h$, representing a cell. The result is an occupancy tensor with dimension $\frac{l}{d_l} \times \frac{w}{d_w} \times \frac{h}{d_h}$. The normalized reflectance across all the length and width cells in a column is computed and added as a channel on top of the occupancy tensor, which gives a final input shape, D_i.

$$D_i = \frac{l}{d_l} \times \frac{w}{d_w} \times (\frac{h}{d_h} + 1) \tag{25}$$

The PIXOR architecture uses a resnet [42] backbone before passing through a fully connected header layers. Before the input passes through the backbone, it passes through two convolutional layers with 32 channels, kernel size of 3, and stride of 1. After these layers are the residual blocks. There are 4 residual blocks with 3, 6, 6, and 4 layers in each block, respectively. The first convolutional layer in each residual block uses a stride of 2, which creates a 16 times down sampling factor for the backbone. After passing through the residual blocks, the network up-samples the feature map with two deconvolutional layers, the output of which are added element wise with lateral layers that branch after the last three residual blocks, similar to U-Net [43]. Thus, the output is down sampled 4 times from the input and has dimension.

The header network of PIXOR has two outputs, a classification output and a regression output. The goal of the classification output is to predict the locations and labels of objects in the output space. The goal of the regression output is to predict feature maps that more precisely define the bounding box of the object. The header network consists of 4 convolutional layers, with an output that produces a 1 channel classification prediction and 6 channel regression prediction. The classification layer uses

a sigmoid activation function, and the regression output uses a ReLU activation function. This completes the PIXOR architecture, a diagram of the network is illustrated in Figure 10.

As mentioned previously, the PIXOR network has 7 total output channels, 1 classification and 6 regression with each channel being 4 times down sampled from the input. Following the KITTI data set format, each point cloud has a length of 80 meters, width of 70 meters and height of 3.5 meters. Thus, for a resolution of 0.1 meters in all directions, the input dimension is $800 \times 700 \times 36$ and the output dimension is $200 \times 175 \times 7$.

For the classification channel, each cell represents the probability of an object being located in that area. For the regression tensor, each column along the third dimension represents a vector, b_p^{ij}, for the corresponding cell area that refines the bounding box prediction. The indices i and j are the spatial indices of the cell from which the regression vector can be found. The regression vector at a particular cell tries to predict the values of the regression target, b_t, of the closest object.

$$b_p^{ij} = \begin{bmatrix} \sin\theta \\ \cos\theta \\ x_c^{ij} \\ y_c^{ij} \\ w \\ l \end{bmatrix} \tag{26}$$

The elements of the regression vector for a cell are very similar to the elements of the regression target vector in Equation 7. The only difference is that (x_c^{ij}, y_c^{ij}) do not locate the center of the object in the LiDAR frame, but rather, locate the relative center of the object in the specific cell frame. However, the transformation from the cell frame to the LiDAR frame can be performed for give cell indices i and j. From this predicted regression vector a predicted corner matrix, $C_p^{ij} \in \mathbb{R}^{4\times 2}$, can be defined which predicts the four corners of the bounding box in the cell frame.

$$C_p^{ij} = \begin{bmatrix} x_c^{ij} - \frac{1}{2}(l\cos\theta + w\sin\theta) & y_c^{ij} - \frac{1}{2}(l\sin\theta - w\cos\theta) \\ x_c^{ij} - \frac{1}{2}(l\cos\theta - w\sin\theta) & y_c^{ij} - \frac{1}{2}(l\sin\theta + w\cos\theta) \\ x_c^{ij} + \frac{1}{2}(l\cos\theta + w\sin\theta) & y_c^{ij} + \frac{1}{2}(l\sin\theta - w\cos\theta) \\ x_c^{ij} + \frac{1}{2}(l\cos\theta - w\sin\theta) & y_c^{ij} + \frac{1}{2}(l\sin\theta + w\cos\theta) \end{bmatrix} \tag{27}$$

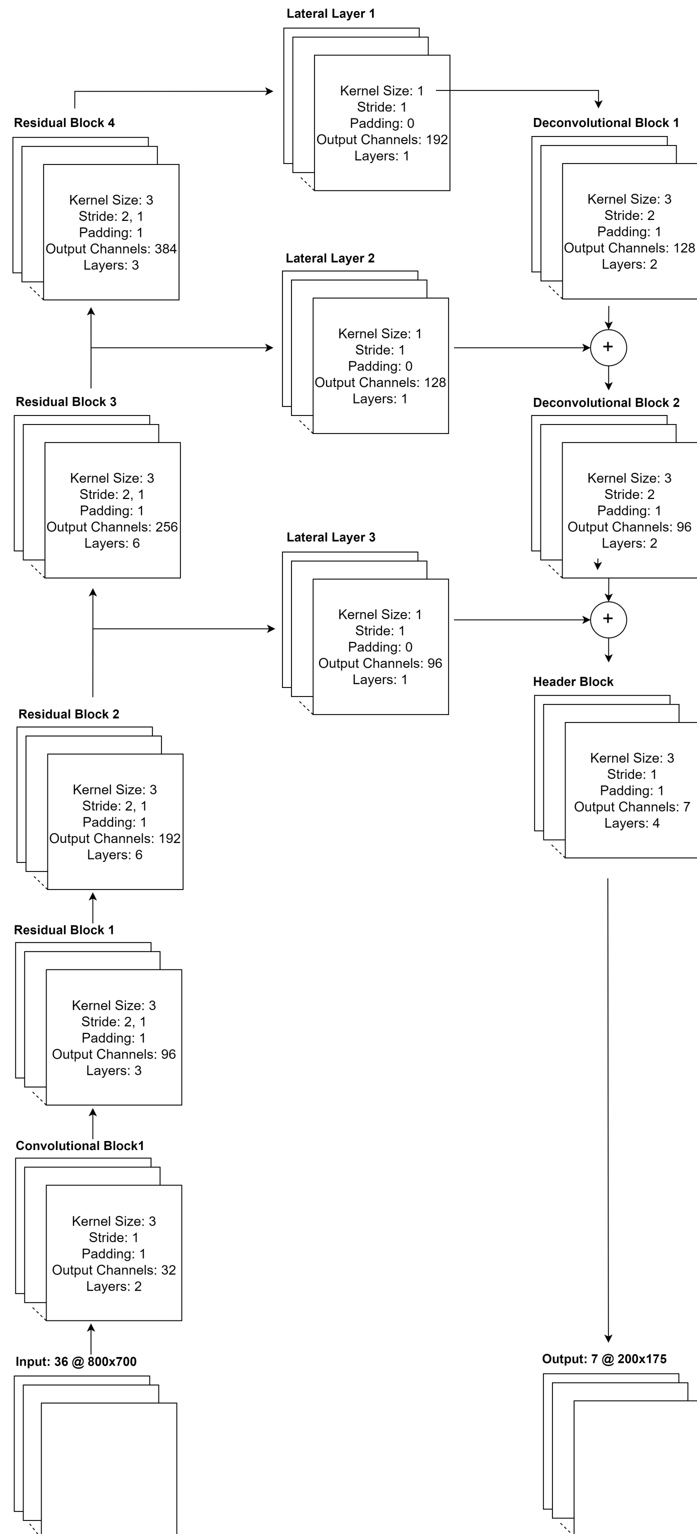


Figure 10: PIXOR Network Architecture

Each row in C_p^{ij} represents a corner of the object, starting at the rear left corner of the object for the first row, proceeding anti-clockwise about the object to the front left corner in the fourth row.

Combining the classification and regression outputs, the overall output can be interpreted as the probability of the existence of a specific bounding box, C, for each vertical column of the output tensor.

Once a prediction has been made by the PIXOR network, the output needs to be further processed to better interpret the result. The first step in post processing is to decode the network output to identify the corners of the bounding boxes in the frame. Recall the regression prediction from Equation 26. The regression output predicts the relative center position, (x_c^{ij}, y_c^{ij}) that each cell is to the center of the bounding box. This encodes more precise positional information on top of the spatial information that is obtained from the indices of the output cell. The goal is to convert the regression output tensor into a tensor containing the entire bounding box information. To do this, two mesh tensors, M^{ij1}, and M^{ij2} with size $\frac{1}{4}(\frac{l}{d_l} \times \frac{w}{d_w})$ are created with the same cell discretization as the output. The mesh tensors correlate their index values to positional information in the LiDAR reference frame. For example, M^{ij1} has increasing values for increasing i, in steps proportional to the output discretization size, and M^{ij2} is the same but along the j index. By performing an element wise addition between the mesh tensors and the relative center position components of the regression tensor, tensors containing the center position of all estimated object centers, X^{ij} and Y^{ij} can be obtained.

$$\begin{aligned} Y^{ij} &= M^{ij1} \oplus b_p^{ij3} \\ X^{ij} &= M^{ij2} \oplus b_p^{ij4} \end{aligned} \tag{28}$$

From these two center position tensors, a corner tensor can be created where each cell in the output predicts the location of a corner for the closest object. To simplify the following notation, $S_\theta^{ij} = b_p^{ij1}$, $C_\theta^{ij} = b_p^{ij2}$, $W^{ij} = b_p^{ij5}$, and $L^{ij} = b_p^{ij6}$. These values represent the sine, cosine, width, and length tensors from the regression output, respectively. Note that in the following context, $\oplus$ and $\otimes$ represent element wise operations.

$$
\begin{aligned}
C^{ij1} &= X^{ij} \oplus \frac{-1}{2}(L^{ij} \otimes C_{\theta}^{ij} \oplus W^{ij} \otimes S_{\theta}^{ij}) \\
C^{ij2} &= Y^{ij} \oplus \frac{-1}{2}(L^{ij} \otimes S_{\theta}^{ij} \oplus (-W^{ij} \otimes C_{\theta}^{ij})) \\
C^{ij3} &= X^{ij} \oplus \frac{-1}{2}(L^{ij} \otimes C_{\theta}^{ij} \oplus W^{ij} \otimes S_{\theta}^{ij}) \\
C^{ij4} &= Y^{ij} \oplus \frac{-1}{2}(L^{ij} \otimes S_{\theta}^{ij} \oplus (-W^{ij} \otimes C_{\theta}^{ij})) \\
C^{ij5} &= X^{ij} \oplus \frac{1}{2}(L^{ij} \otimes C_{\theta}^{ij} \oplus W^{ij} \otimes S_{\theta}^{ij}) \\
C^{ij6} &= Y^{ij} \oplus \frac{1}{2}(L^{ij} \otimes S_{\theta}^{ij} \oplus (-W^{ij} \otimes C_{\theta}^{ij})) \\
C^{ij7} &= X^{ij} \oplus \frac{1}{2}(L^{ij} \otimes C_{\theta}^{ij} \oplus (-W^{ij} \otimes S_{\theta}^{ij})) \\
C^{ij8} &= Y^{ij} \oplus \frac{1}{2}(L^{ij} \otimes S_{\theta}^{ij} \oplus W^{ij} \otimes C_{\theta}^{ij})
\end{aligned} \tag{29}
$$

Note that each operation on the above tensors is element-wise. Also note that the vector along the third dimension in C can build the matrix C_p, meaning that for the output tensor, (C^{ij1}, C^{ij2}) represent rear left corner predictions, (C^{ij3}, C^{ij4}) represent rear right corner predictions, (C^{ij5}, C^{ij6}) represent front right corner predictions, and (C^{ij7}, C^{ij8}) represent front left corner predictions.

Each cell in the output tensor, C, represents a bounding box prediction. Even if cell indices i and j are only taken for regions where the classification prediction is above some minimum threshold, since a single object can be represented by multiple cells, there will be too many predictions. To solve this issue, non maximum suppression is conducted on the predictions to choose the best bounding box for each object. Non maximum suppression is defined in algorithm 5.

Algorithm 5: Non Maximum Suppression

Input: B, S, t
Output: B_{max}

$B_{max} \leftarrow \emptyset$
while $|S| > 0$ **do**
  $i \leftarrow S_1$
  add B_i to B_{max}
  for $j \in B$ **do**
    if $\frac{B_i \cap B_j}{B_i \cup B_j} > t$ **then**
      delete S_j
  delete S_1
return B_{max}

The algorithm works by comparing the intersection over union (IoU) of all the bounding box predictions. The intersection over union is the ratio of the amount two objects overlap to the total area of the two objects. In the bird's eye view prediction, the IoU is a comparison of two areas, A_1 and A_2 of two boxes.

$$IoU = \frac{A_1 \cap A_2}{A_1 \cup A_2} = \frac{A_1 \cap A_2}{A_1 + A_2 - A_1 \cap A_2} \tag{30}$$

Non maximum suppression takes a set of bounding boxes, B, sorted by a corresponding set of scores representing the probability of each bounding box, S, and a minimum overlap threshold, t. Each box is iterated based on the score, and the bounding box with the highest score is added to the output. Then, the intersection over union between every other box is compared against the highest scoring box. If each IoU is above the minimum threshold, then it is deleted since the highest scoring box is already found. Boxes that are lower than the minimum threshold most likely correspond to a different object. The set of boxes is iterated until all the scores have been parsed, resulting in the best matching predictions for every object.

While the PIXOR network is good at detecting objects within point clouds, it is not without its challenges. One issue with the PIXOR network is it was designed for large scale autonomous vehicles equipped with 3D LiDARs. Thus, it was not designed for object detection in 2D point clouds. Furthermore, although it is a very fast network, it was still designed to be operated on powerful equipment and is not suited well for smaller robots with less computational power.

Another issue with the PIXOR network is due to the nature of making predictions in a static time interval. Since there is no temporal knowledge of objects in the environments, some objects are predicted with a 180 degree heading angle error. This error stems from the symmetry of the object, that is, for rectangular shaped bounding boxes, there are two possible heading angles. The 3D PIXOR network can partially mitigate this issue since real world vehicles have other features that can help indicate the heading angle, however, the issue can still persist in certain cases. Also introduced by making predictions in only a single frame is a problem where sequential predictions tend to oscillate due to small errors in the predicted center position. This oscillation oscillation can be removed by filtering techniques, but this means adding another post processing step.

Inspired by the PIXOR network, the next section proposes a new architectures that can solve the

aforementioned challenges by making the network more scalable and by extending the network to take time series data as an input.

2.4 Newly Proposed Deep Learning Approaches

2.4.1 PIXOR2D

As highlighted in the first chapter, the motivation for scaling the PIXOR network to operate in 2 dimensions is so that it can operate on lower cost 2D LiDAR sensors, and on devices with less computational resources. From the architecture shown in Figure 10., it is possible to calculate the memory usage of the 3D PIXOR network. Table 1. summarizes the memory requirements of each block of the 3D PIXOR network.

Table 1: Memory Requirements of 3D PIXOR Network

Block Name	Layers	Output Channels	Output Dimension	Memory Usage (Mb)
Input	1	36	800×700	80.64
Conv. Block 1	2	32	800×700	143.360
Res. Block 1	3	96	400×350	161.280
Res. Block 2	6	192	200×175	161.280
Res. Block 3	6	256	100×87	53.453
Res. Block 4	3	384	50×43	9.907
Lat. Layer 1	1	192	50×43	1.651
Lat. Layer 2	1	128	100×87	4.454
Lat. Layer 3	1	96	200×175	13.44
Deconv. Block 1	2	128	100×87	8.909
Deconv. Block 2	2	96	200×175	26.88
Header Block	4	7	200×175	3.92
Total	-	-	-	669.174

While the total memory requirement may not seem like much, it should be remembered that neural networks require contiguous memory to ensure the fastest forward pass time. This can be a significant requirement for a small robot running on a total of 4Gb of memory, such as a robot using an Nvidia Jetson TX1. Even if a robot is capable of running the full 3D PIXOR network, it will likely need to run many other algorithms in parallel, such as those for localization and mapping, navigation and path planning, data collection, and more. Each of these will also have their own memory requirements, especially if they are also deep learning approaches, so it is important to minimize the memory cost of a deep learning algorithm if possible.

For a 2D implementation of PIXOR, the input channel size is reduced to 1 planar channel, with the

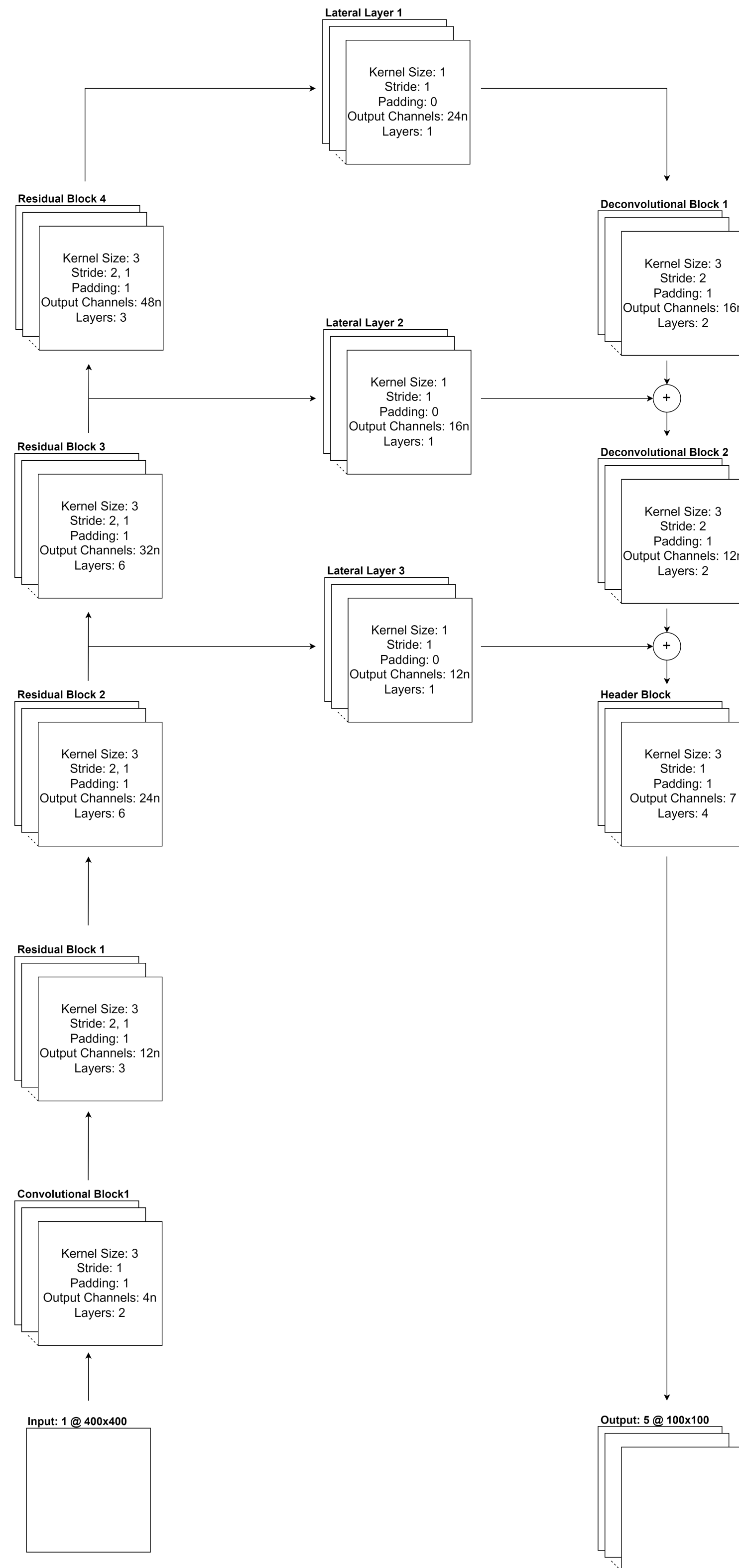


Figure 11: Modified 2D PIXOR Network

option of adding an additional channel if the 2D LiDAR being used returns reflectance or intensity information as well. Since the input to the network is greatly reduced, there is a smaller spatial region in which features can be learned. Due to this, it makes sense to add a scaling parameter, n, that can linearly reduce the quantity of channels in each block between the input and header. This greatly improves the memory cost of the 2D PIXOR network over the 3D PIXOR network. This modified PIXOR2D architecture is illustrated in Figure 11.

The memory requirements of the 2D PIXOR network are provided in Table 2. A scale factor of $n = 1$ is the minimum possible scaling, but it is recommended that a scale factor of $n = 2$ is used minimum. To obtain the same breadth of channels as the 3D PIXOR network, a scale factor of $n = 8$ should be used. From this table, it can be seen that scaling the breadth of the PIXOR network down greatly improves the memory cost of running the network. Reducing the scale factor of the 2D PIXOR network from 8 to 2 has little affect on the performance, as will be demonstrated in Chapter 4.

Table 2: Memory Requirements of 2D PIXOR Network

Block Name	Layers	Output Channels	Output Dimension	Memory Usage (Mb)
Input	1	1	800×700	2.24
Conv. Block 1	2	4n	800×700	17.92n
Res. Block 1	3	12n	400×350	20.16n
Res. Block 2	6	24n	200×175	20.16n
Res. Block 3	6	32n	100×87	6.682n
Res. Block 4	3	48n	50×43	1.238n
Lat. Layer 1	1	24n	50×43	0.206n
Lat. Layer 2	1	16n	100×87	0.557n
Lat. Layer 3	1	12n	200×175	1.68n
Deconv. Block 1	2	16n	100×87	1.114n
Deconv. Block 2	2	12n	200×175	5.04n
Header Block	4	7	200×175	3.92
Total (n = 2)	-	-	-	155.674

2.4.2 Extending PIXOR with Recurrent Layers

While PIXOR is good at identifying bounding boxes in a static frame, predictions in sequential frames tend to oscillate around a central position. Predictions fluctuate due to sensor noise within each LiDAR scan, resulting in small positional error in the bounding box prediction. This error is isotropic, and so, in sequential frames, the bounding box appears to oscillate as can be seen in Figure 12.

This type of motion can be filtered using recursive Bayesian estimation techniques such as Kalman

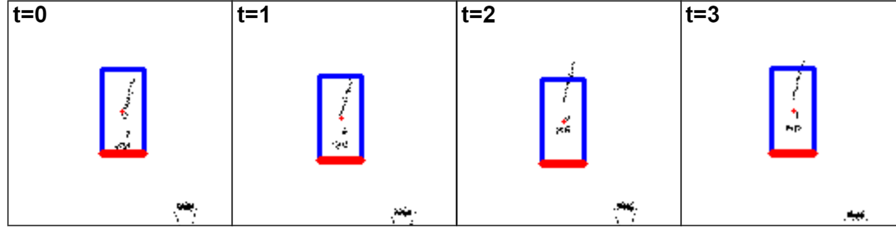


Figure 12: Oscillating Bounding Boxes in Sequential Predictions

filtering, but this would be an additional post processing step. To solve this problem in the same pass that the predictions were made, the PIXOR network was expanded to take a time series of point clouds as an input. This modification also allows the PIXOR network to estimate the velocity of objects since the time between sequential LiDAR frames is close to constant. To make meaningful predictions given the time series input, the PIXOR network was expanded by adding recurrent layers. There are many types of recurrent layers that could be employed such as gated recurrent units (GRUs) [44] and long short term memory (LSTM) [45] layers. However, these architectures require a flattened input. Since the spatial information is the most important for the predictions, the ConvLSTM [46] layer was chosen. The key difference between a ConvLSTM layer and a traditional LSTM layer is that in the ConvLSTM layer, the gates that control the LSTM have their inputs convolved first.

$$
\begin{aligned}
i_t &= \sigma(W_{xi} * x_t + W_{hi} * h_{t-1} + b_i) \\
f_t &= \sigma(W_{xf} * x_t + W_{hf} * h_{t-1} + b_f) \\
c_t &= f_t c_{t-1} + i_t \tanh(W_{xc} * x_t + W_{hc} * h_{t-1} + b_c) \\
o_t &= \sigma(W_{xo} * x_t + W_{ho} * h_{t-1} + b_o) \\
h_t &= o_t \tanh(c_t)
\end{aligned}
\tag{31}
$$

Here, i_t, f_t, o_t are the input, forget and output gates, respectively. c_t and c_{t-1} are the current and previous cell states, and h_t is the current hidden state. W and b represent weight and bias matricies for the different operators, and x_t is the current input. The ConvLSTM layers were introduced after the header network since the goal of the recurrent layers are to smooth out the predictions made once the spatial features have been extracted. The new architecture uses a time series length of 7, however, many other time series lengths are suitable as well. The new output contains 10 channels, 1 classification channel and 4 regression channels for the predictions at time steps $t-1$ and t. The reason for predicting the bounding boxes at $t-1$ is so that the difference can be measured between time steps

to estimate the bounding box velocity. Figure 13. illustrates the newly proposed architecture for the recurrent PIXOR model.

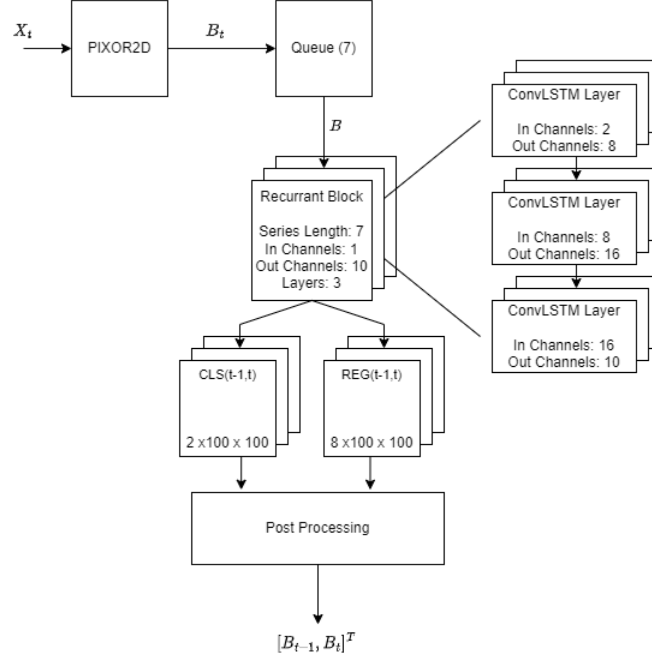


Figure 13: Recurrent PIXOR Architecture

Training the recurrent network is straight forward, as the loss can be computed the same as previously introduced, but aggregating over the amount of predicted frames. Thus, the loss functions remain the same, but the cost functions add another summation term as follows.

$$J_{FL}(\hat{Y}, Y) = -\frac{\alpha}{nT}\sum_{t=1}^{T}\sum_{i=1}^{n}\sum_{c=1}^{C}(1-p_t)^{\gamma}\log(p_t) \tag{32}$$

$$J_{SL1}(\hat{Y}, Y) = \frac{1}{nT}\sum_{t=1}^{T}\sum_{i=1}^{n}\frac{(\hat{y}-y)^2}{2\beta}H(|\hat{y}-y|-\beta) + (|\hat{y}-y|-\frac{1}{2})(1-H(|\hat{y}-y|-\beta)) \tag{33}$$

Here, T is the size of the time series being predicted. From Figure 13., $T = 2$, but this network can be trained to predict any length of time series, however, the accuracy will reduce the further a prediction is made from the last known time step. This new adaption of PIXOR is named PIXFOR for its ability to predict future oriented bounding boxes, and its performance is assessed in Chapter 4.

2.5 Section Conclusions

This chapter introduced point clouds, and how to detect objects contained within them. Traditional approaches to the object detection task were explored before moving on to more novel deep learning approaches. The PIXOR network was selected as deep learning architecture basis for this research due to it's fast run time, and it's ability to be adapted to 2D and 3D point clouds.

There are two key novelties introduced in this section. A new PIXOR network architecture is proposed that can work using the input from a 2D LiDAR, with appropriate scale factors being proposed to reduce the memory requirements so that this 2D architecture is better fit for smaller autonomous systems. Then, this network architecture is extended by adding recurrent layers to smooth predictions and also estimate the velocity of objects. The power of this new recurrent PIXOR model, named PIXFOR, is demonstrated further in the next chapter as its ability to track objects in point clouds is explored.

3 Object Tracking in Point Clouds

3.1 Object Tracking Overview

Object tracking usually consists of two sub processes; data association and track management. Data association is the process of associating groups of data at a current data step with identified data in previous data steps, and track management is the task of managing these associations throughout a time interval. In its most basic form, this means giving each object a unique identifier during the first frame that the object is detected, and re-associating this identifier with the correct object in future frames. In point clouds, tracking can be segmentation based, in which each point in a dynamically changing cluster is tracked, or object based, where the features of the point cluster, such as a bounding box, are tracked. In most cases, object tracking is built to be integrated with, or on top of, object detection to solve the detection and tracking of moving objects (DATMO) problem.

There are many categories of solutions to the DATMO problem, but for the LiDAR based DATMO problem, most approaches fit in two categories; *detect before track* and *track before detect* as identified in [22]. As the name suggests, detect before track approaches perform the object detection task first, and use the result to perform track management. The track before detect approach tracks individual points or regions in the point cloud and then identifies relevant objects from the tracks. In most cases, the detect before track approach is an object-based DATMO approach, and the track before detect approach is a grid-based DATMO approach, as identified in a 2019 review of DATMO methods [47]. While deep learning methods have been used to solve parts of the DATMO problem, such as the object detection task, to the best of the author's knowledge, there is no such deep learning approach to solve the DATMO problem from end to end. The next section of this chapter will explore some of the existing methods and identify their strengths and weaknesses. The following sections will identify two new solutions to the DATMO problem, with the latter being a novel end to end deep learning approach.

3.2 Approaches to the DATMO Problem

3.2.1 Kalman Filtering

Before discussing approaches to the DATMO problem, it is important to introduce the Kalman filter [48] as it is an important step in many DATMO approaches. Kalman filtering is a method where a state vector is estimated using a time series of measurements that are combined with noise and unknown terms. The Kalman filter algorithm is as follows. Given a state vector from a previous time step, x_{k-1}, and control input vector, u_k, the linear state space model of the system can be defined.

$$x_k = \tilde{A}_k x_{k-1} + \tilde{B}_k u_k + w_k \tag{34}$$

$$z_k = \tilde{C}_k x_k + v_k \tag{35}$$

In this state space model, the matricies $\tilde{A}_k, \tilde{B}_k$ and $\tilde{C}_k$ represent the state transition model, control input model and observation model of the system, respectively. The vectors w_k and v_k are the process noise with covariance Q_k and observation noise with covariance R_k respectively.

Once the linear state space model defined, the first step of the Kalman filtering process is to predict the *a priori* state $\hat{x}_{k|k-1}$ and covariance matrix $P_{k|k-1}$. The *a priori* state is the prediction given the state space model, but it is not yet corrected based on the measurements and the *a priori* covariance matrix describes the accuracy of the state prediction.

$$\hat{x}_{k|k-1} = A_k \hat{x}_{k-1|k-1} + B_k u_k \tag{36}$$

$$P_{k|k-1} = A_k P_{k-1|k-1} A_k^T + Q_k \tag{37}$$

Here, $\hat{x}_{k-1|k-1}$ and $P_{k-1|k-1}$ are the state estimate and covariance matrix from the previous time step, updated using measurements up to and including the previous time step. Once this prediction step is complete, the next step is to update the prediction based on the measurement information, by propagating the uncertainty information encoded in the covariance matrix. To do this, the measurement pre-fit residual, y_k, is first determined by finding the difference between the observation and optimal forecast.

$$y_k = z_k - \tilde{C}_k \hat{x}_{k|k-1} \tag{38}$$

This value is also referred to as the innovation. Similarly, the pre-fit residual covariance matrix (or

innovation covariance matrix), $\tilde{S}_k$, is computed from the propagated covariance matrix.

$$\tilde{S}_k = \tilde{C}_k P_{k|k-1} \tilde{C}_k^T + R_k \tag{39}$$

from the pre-fit residual and pre-fit residual covariance matrix, the optimal Kalman gain, K_k can be computed. The Kalman gain provides information on how the predicted state should be updated by minimizing the pre-fit residual error.

$$K_k = P_{k|k-1} \tilde{C}_k^T \tilde{S}_k^{-1} = P_{k|k-1} \tilde{C}_k^T (\tilde{C}_k P_{k|k-1} \tilde{C}_k^T + R_k)^{-1} \tag{40}$$

With the Kalman gain determined, the state estimate and estimate covariance matrix can be updated, yielding an *a posteriori* which are then also used to repeat the process again for the next time step as the *a priori*.

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k y_k \tag{41}$$

$$P_{k|k} = (I - K_k \tilde{C}_k) P_{k|k-1} \tag{42}$$

The final state estimate covariance matrix, $P_{k|k}$ should be smaller than the state prediction covariance matrix, $P_{k|k-1}$. This indicates that there is a higher confidence of the final state estimate, $\hat{x}_{k|k}$, than in the state prediction, $\hat{x}_{k|k-1}$.

The Kalman filter is great at estimating problems that can be described by linear state space models. However, object motion often has nonlinear components. Thus, to solve the tracking problem in most cases, the extended Kalman filter is often required.

3.2.2 Extended Kalman Filter

The extended Kalman filter (EKF) is a nonlinear version of the Kalman filter. Instead of requiring a linear state space model, the EKF only requires the state and observation models to be differentiable. In this nonlinear version of the Kalman filter, the state model and observation model are defined as follows.

$$x_k = f(x_{k-1}, u_k) + w_k \tag{43}$$

$$z_k = g(x_k) + v_k \tag{44}$$

Here, the variables are the same as the Kalman filter, and f and g are non-linear, differentiable func-

tions that describe the system. Following the same process as the linear Kalman filter, the next step after defining the system models is to predict the future state and covariance.

$$\hat{x}_{k|k-1} = f(\hat{x}_{k-1|k-1}, u_k) \tag{45}$$

$$P_{k|k-1} = \tilde{A}_k P_{k-1|k-1} \tilde{A}_k^T + Q_k \tag{46}$$

This time however, the state transition matrix, A_k, is defined as the Jacobian of f.

$$\tilde{A}_k = \left.\frac{\partial f}{\partial x}\right|_{\hat{x}_{k-1|k-1}, u_k} \tag{47}$$

Again, the update step follows the prediction step and continues to have the same form as the linear Kalman filter. This time, the measurement pre-fit residual is determined using the nonlinear function, g, and the innovation covariance is also determined.

$$y_k = z_k - g(\hat{x}_{k|k-1}) \tag{48}$$

$$\tilde{S}_k = \tilde{C}_k P_{k|k-1} \tilde{C}_k^T + R_k \tag{49}$$

Similar to the state transition matrix, for the EKF, the observation matrix, C_k, is defined as the Jacobian of g.

$$\tilde{C}_k = \left.\frac{\partial g}{\partial x}\right|_{\hat{x}_{k|k-1}} \tag{50}$$

Once these initial steps are completed, the rest of the EKF is the same as the Kalman filter. The Kalman gain, K_k can be determined following Equation 40., and then the *a posteriori* state estimate and covariance can be computed following equations defined in Equation 41.

3.2.3 Data Association Techniques

The data association step is employed before the tracking phase and is responsible for deciding which data clusters need to be tracked, and how they are associated with data clusters from previous time steps. A simple method for data association is the global nearest neighbour search algorithm. In this algorithm, a data cluster at time step k is associated to the nearest data cluster at time step $k-1$ based on the Euclidean distance. Based on information acquired from previous measurements of tracked objects in the environment, the result from the nearest neighbour can be used to determine the probability that a measured cluster belongs to a specific target. This probability can be used with methods

like multiple hypothesis tracking (MHT) to generate hypotheses, or the joint probabilistic data association filter (JPDAF).

Multiple hypothesis tracking is a type of probabilistic filtering approach to the tracking problem where multiple association hypotheses between measured data clusters are maintained. This approach fits into the object-based DATMO approach since it is the clustered data which is being tracked. The first approach to multiple hypothesis tracking is designed for the more general task of tracking multiple targets in any type of clustered data set [49]. For a data set k, this method generates a set of hypotheses, Ω^k that associate the set of measurements, Z^k up to and including the current measurement, $Z(k)$.

$$\begin{aligned} Z(k) &= \{z_i(k), i = 1, 2, ..., I_k\} \\ Z^k &= \{Z(1), Z(2), ..., Z(k)\} \\ \Omega^k &= \{\Omega^k_j, j = 1, 2, ...J_k\} \end{aligned} \tag{51}$$

When a new measurement, $Z(k+1)$, is made, a new hypothesis Ω^{k+1} is formed. To do this, the hypothesis $\bar{\Omega}^0 = \Omega^k$ is initialized where $\bar{\Omega}^i$ represents the hypothesis after the ith measurement. Then, a new set of hypotheses, $\bar{\Omega}^i$, are formed for each prior hypothesis $\bar{\Omega}^{i-1}_j$. This new set represents the joint hypothesis that $\bar{\Omega}^{i-1}_j$ is true, and that measurement $Z_i(k+1)$ comes from a particular target with a unique identifier.

The problem with this method is that the quantity of hypotheses grows with every time step due to new possible hypotheses being created as new targets are initiated with each measurement on top of all the previous hypotheses. Thus, this method needs to be combined with hypothesis reduction techniques, otherwise the computational cost is not feasible. The easiest way to reduce the number of hypotheses is to choose the hypothesis with the most likely data association. Once the hypotheses are reduced, Kalman filtering can be used to estimate the state.

Another approach for performing data association and filtering is the Joint Probabilistic Data Association Filter [50]. Unlike MHT, the JPDAF method does not suffer from the set of hypotheses expanding at each time step. In the JPDAF method, the measurement update of the tracks takes into account all possible assignments of data clusters.

$$P(Z_t|S_t^k) = \eta \sum_{j=1}^{J_t} \beta_j^k P(Z_t^j|S_t^k) \tag{52}$$

where $P(Z_t|S_t^k)$ is the probability that data cluster Z_t is from target k, β_j^k is the probability that measurement j belongs to target k, J_t is the set of data clusters at the current time step t, and η is a normalization constant.

The probability β_j^k that measurement j belongs to target k is obtained by summing over all feasible events, χ, where the joint event occurs.

$$\beta_j^k = \sum_{\chi} P(\chi|Y^k)\hat{\omega}_{jk}(\chi) \tag{53}$$

$$\hat{\omega}_{jk}(\chi) = \begin{cases} 1 & \chi_{jk} \ occurs \\ 0 & \text{otherwise} \end{cases} \tag{54}$$

Here, Y^k is the set of all data vectors and $\chi = \cup_{j=1}^{J_t}\chi_{jk}$ is the evaluation of all the events where measurement j originated from target k. Like MHT, once the measurement data is associated by the JPDAF, it needs to be tracked, usually by a Kalman filter.

3.2.4 Occupancy Grid and Obstacle Map Approach

The occupancy grid approach, also sometimes called an occupancy map, represents the environment as a discretized grid of cells which are each tracked. These grid cells are at a resolution smaller than any object in the environment and thus, an object is tracked by aggregating the tracked grid cells that it intersects. One such occupancy grid approach constructs a static obstacle map (SOM) [22] based on the motion of points in the point cloud relative to the ego vehicle. In this approach, the static obstacle map is predicted by comparing the points in the current grid to their location in the previous grid. To calculate the probability of the j-th grid being static at the current time step, the distance to the four nearest grids, $[l_1, l_2, l_3, l_4]$, at the previous time step are used. The predicted probability is the weighted average of the four previous grids which have a midpoint distance less than $\sqrt{2}d_{grid}$, where d_{grid} is the discretization size of the occupancy grid.

$$L = \sum_{i=4}^{4} l_i^{-1} \tag{55}$$

$$P(\bar{x}^j_{static}[k]) = \begin{cases} \sum_{i=1}^{4} \frac{l_i^{-1}}{L} P(\bar{x}^i_{static}[k-1]) & l_i \neq 0 \\ P(\bar{x}^i_{static}[k-1]) & l_i = 0 \end{cases} \tag{56}$$

Once the SOM is predicted, the geometric model free approach (GMFA) presented in [22] constructs correspondences between the non-static points in consecutive scans to update the SOM. Once correspondences are created, clusters of points are matched using ICP, and then the states of these clusters are tracked using an EKF.

3.3 The Detect While Track Approach

3.3.1 Problems with Traditional Approaches

As identified previously, there are two categories of approaches to the DATMO problem, *detect before track* and *track before detect*. Both of these categories use a data association step to identify which data clusters need to be tracked at the current time step, then usually, a Kalman filtering approach is used to estimate the state of the tracked objects. The problem with these two categories is that ordering the detection and tracking tasks creates an implicit delay since one task requires the output of the other to solve the DATMO problem.

This delay can be accommodated for by ensuring the use of efficient algorithms for both detection and tracking such as the approach in [23]. In this method, the data association problem is solved using a nearest neighbour search, and then an efficient l-shape feature extractor [51] is used to predict vehicle bounding boxes. Finally, an unscented Kalman filter (UKF) is used to estimate the state of the vehicles. The nearest neighbour search, L-shape extractor, and UKF are all very efficient algorithms which allow for a fast run time, however, the cost of this efficiency is the accuracy of the method. The drawback of the L-shape extractor is it depends on an L-shape in the environment, but there are scenarios where a point cloud of an object is not represented by an L-shape. Such a case is when the heading angle of the LiDAR is completely parallel or perpendicular to an object. In this case, the shape of the point cloud is an I-shape. Furthermore, the nearest neighbour search algorithm has the potential to merge predictions in cases where two clusters are in close proximity.

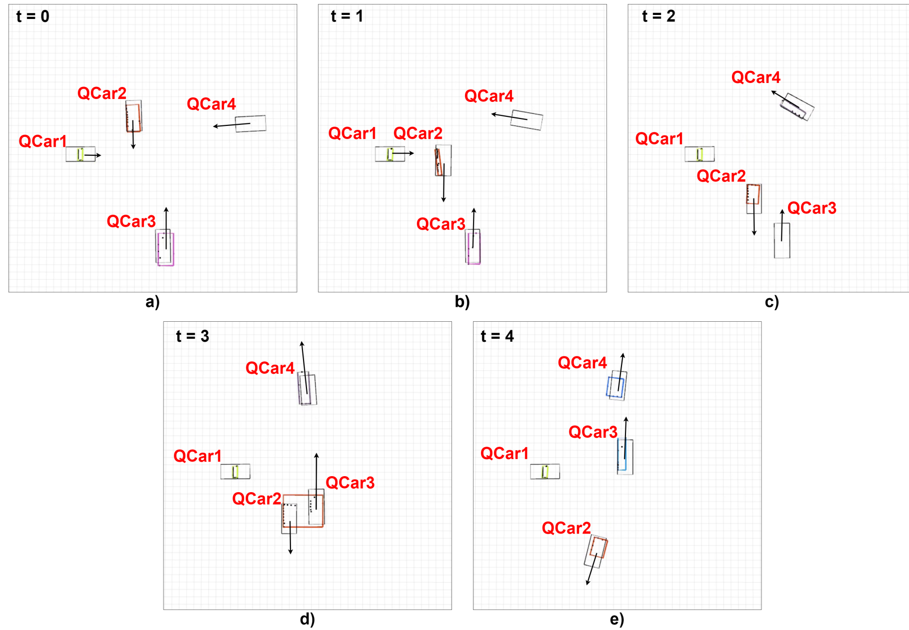


Figure 14: Intersection Scenario Example Depicting Prediction Errors of the Nearest Neighbour Search and L-Shape Feature Extractor

Figure 14. illustrates the issues that can arise when using the nearest neighbour search and L-shape feature extractor to solve the DATMO problem. In the figure, QCar1 is the ego vehicle which made the LiDAR measurements. The black boxes represent the ground truth bounding boxes. The colourful boxes represent the predicted bounding box from the L-Shape tracker. The time series of events are described as follows: **a)** the L-Shape tracker works well. **b)** QCar2 is perpendicular to QCar1, and the L-Shape tracker cannot accurately predict the bounding box of QCar2 because the point cloud of QCar2 forms an I-shape. **c)** QCar2 fully occludes QCar3 **d)** QCar3 re-enters the LiDARs view, but is still partially occluded by QCar2. QCar2 and QCar3 are close enough that the L-Shape tracker predicts that both QCar3 and QCar2 are one vehicle. **e)** QCar3(formerly pink, now blue vehicle) is no longer occluded by the orange vehicle, but it has not been tracked for so long that a nearest neighbour search algorithm believes it must be a new vehicle. It should be noted that the vehicle labels and heading velocity are added in manually for this illustration.

Changing the parameters of the nearest neighbour search to use a smaller Euclidean distance can alleviate the issue at $t = 3$ where the bounding box combines two clusters into one prediction. However, this can have the unintended effect of creating two predictions for one object if the object is suffi-

ciently large. This is why it is better to generate multiple hypothesis or use a JPDAF approach, since these methods can propose better associations for the tracks. The issue of the I-shape feature can be solved by using an object model approach to determine bounding boxes, but due to variations in vehicle shapes in the real world, it is difficult to design an all-encompassing model. A neural network however, can be trained on many real world scenarios and can converge on a best bounding box for a given point cluster.

In a model based tracking (MBT) approach [21], a probabilistic measurement model is used to associate data and then the tracks are maintained using the MHT framework proposed in [52]. This more sophisticated tracking model greatly improves on a nearest neighbour approach. The GMFA approach from [22] then improves on the results from MBT using the static obstacle map, yielding a higher F_1 score. However, due to the nature of the GMFA method, while objects can be tracked, a bounding box is not provided. Furthermore, in [22], some cases are identified where GMFA and MBT cannot detect or track an object due to the nature of the track before detect approach requiring consecutive scans to make a prediction. Thus, a better solution to the DATMO problem should use a probabilistic data association method with the ability to detect objects, even if they only appear in a single frame.

3.3.2 Improving Traditional Approaches with PIXOR

One way to improve on these traditional approaches it to combine them with a convolutional neural network, such as the 2D PIXOR network proposed in Chapter 2., to improve on the object detection component. The benefit of such a method is that the object detection can be run on a GPU while the tracking algorithms can run on a CPU, allowing the detection and tracking tasks to execute simultaneously. This novel method is called a *detect while track* (DWT) approach.

The DWT approach builds on [23] for the tracking approach. As with any tracking solution, the goal is to find a set of tracks, N, where each $n \in N$ represents a tracked vehicle where the track is the following state vector.

$$n = \begin{bmatrix} id \\ x \\ y \\ \theta \\ vx \\ vy \\ \omega \end{bmatrix} \tag{57}$$

Here id is a unique identifier for this track, x and y denote the center position of the track, θ is the orientation of the track, which points in the heading direction, vx and vy are the velocity components of the track, and ω is the rotational rate of the track. Next, a mapping function, f, is defined which maps a track, n, to a tracked measurement, ζ.

$$f : N(K) \mapsto Z(k) \tag{58}$$

where $Z(k)$ is the measurement containing the set of point clusters at some time step, k. For a 2D point cloud input, these clusters are obtained using the adaptive breakpoint detection (ABD) algorithm [53]. The ABD algorithm segments the input point cloud into a set of clusters at a certain time step, Z_k. Next, for a known set of tracks at the previous time step, N_{k-1} and some euclidean threshold, D, the tracks can be recursively updated using a nearest neighbour search algorithm to obtain a set of naive tracks.

$$f(n_k) = \underset{\zeta \in Z_k}{\arg\min} ||\zeta - f(n_{k-1})||, \quad ||\zeta - f(n_{k-1})|| < D \tag{59}$$

A new naive track is created for every cluster, ζ, that is not mapped to a track in the above update function. The reason for calling these naive tracks is because they are not necessarily the final tracks used by DWT, but are rather a starting set of hypotheses. These hypotheses will later be evaluated to determine a true track. Finally, the L-Shape feature extractor proposed by [51] is applied on each $\zeta \in Z_k$ to get a set of L-shapes, L_k. The state of these L-shapes is then tracked by either a EKF, or a UKF following [23].

The approach thus far is congruent to [23]. The difference however, is that the tracks identified thus

far are only hypotheses. At the same time that these tracks are determined, the 2D PIXOR network also process the input point cloud. The output of the network is a set of bounding boxes at the current time step, B_k, which can also be used to generate a set of hypotheses. Thus, the final step of this approach is to evaluate each hypothesis to determine the best fit for the set of tracks at the current time step, N_k.

To reduce the hypotheses, a candidate proposal system is employed as a post processing step. The candidate proposal system begins by assuming there are some number of ground truth objects to be tracked in the current frame. These objects are denoted super candidates, S_k. Suppose there is a set of naive tracks, N_k and a set of bounding boxes, B_k, where each $b_p \in B_k$ is the regression vector defined in Equation 26. at the current time step, k. Next, an empty set of candidates, C_k is initialized for the current time step to represent the hypotheses. The first step of the candidate proposal algorithm is to iterate over $n \in N_k$ and $b_p \in B_k$ to find which clusters are contained within a bounding box. A candidate is created for each of these cases, with the position set to the center of the bounding box. A candidate is also created for each cluster that is not contained within a bounding box, using the center position and velocity vector of the cluster to define the candidate position and orientation. Finally, candidates are also created for each bounding box prediction that does not contain a naive tracked cluster.

For these generated candidates, clusters contained within bounding box predictions have the highest probability of being a true track, bounding boxes with no contained clusters have the next highest probability, and clusters not contained in bounding boxes have the lowest probability. It should be noted that candidates created from bounding box predictions that do not contain clusters have no tracking information at this time. Now, the hypotheses can be reduced. In most cases, there should be as many distinct hypotheses remaining as there were tracks at the previous time step, however, this may not be the case such as when new objects enter the frame, or existing objects leave. Regardless, the first step in reducing the hypotheses is to compute the IOU between all the super candidates at the previous time step, $s_{i,k-1} \in S_{k-1}$ and all the candidates at the current time step, $c_{j,k} \in C_k$.

$$IOU = \frac{s_{i,k-1} \cap c_{j,k}}{s_{i,k-1} \cup c_{j,k}} \tag{60}$$

For a LiDAR with a sufficiently high scan rate, an object should not move much in one time step. Thus, super candidates at the current time step are chosen as the candidates which maximize this IOU.

$$s_{i,k} = \arg\max_{c_{j,k} \in C_k} \frac{s_{i,k-1} \cap c_{j,k}}{s_{i,k-1} \cup c_{j,k}}, \quad s_{i,k-1} \cap c_{j,k} > 0 \tag{61}$$

Algorithm 6: Candidate Matching Algorithm

Input: C_k, C_{k-1}, S_{k-1}, D
Output: S_k
for $s_{i,k-1} \in S_{k-1}$ **do**
    find $c_{j,k} \in C_k$ with largest IOU with $s_{i,k-1}$
    if *largest IOU* > 0 **then**
        $s_{i,k} \leftarrow c_{j,k}$
        mark $s_{i,k-1}$ as tracked
for $s_{i,k-1} \in S_{k-1}$ **do**
    if $s_{i,k-1}$ *is not tracked* **then**
        find $c_{j,k} \in C_k$ that is closest to $s_{i,k-1}$ within threshold D
        if $c_{j,k}$ *is found and has id known by* $s_{i,k-1}$ **then**
            $s_{i,k} \leftarrow c_{j,k}$
        else
            evaluate initial conditions
for $c_{j,k-1} \in C_{k-1}$ **do**
    if $c_{j,k-1}$ *is not tracked* **then**
        find $c_{j,k} \in C_k$ with largest IOU with $c_{j,k-1}$
        if $c_{j,k}$ *is not tracked AND IOU* > 0 **then**
            add $c_{j,k}$ to S_k
return S_k

When a candidate is matched to a super candidate and it has a naive track, then the super candidate adds the *id* associated with the naive track to a set of known *id*s. If not all super candidates from the previous time step are matched, Then a nearest neighbour search is conducted between the unmatched previous super candidates and current candidates to find objects that may have moved too quickly in one frame. If this nearest neighbour search finds a candidate, and the candidate has an *id* that is in the set of known *id*s for the super candidate, then the candidates are matched. If this fails, then it is assumed that the object has left the frame of view. The last step is to check the IOU between all remaining candidates at the current time step with all unmatched candidates from the previous time step. In cases where the IOU is large, it is likely that the object that was proposed in the previous time step is a new object to be tracked, and a new super candidate can be created. Finally, initial conditions can also be passed to propose where super candidates could be on the first time step. This candidate matching algorithm for hypothesis reduction is summarized in algorithm 6. This completes

the methodology of the DWT approach. Figure 15. Illustrates the DWT process as a block diagram.

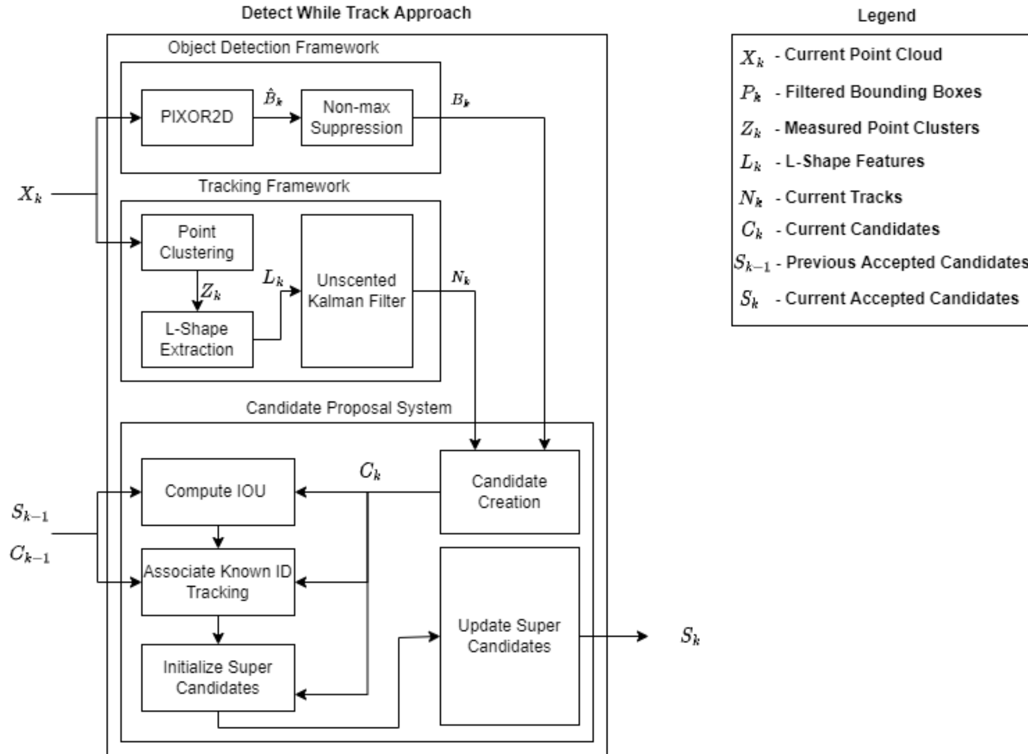


Figure 15: The Detect While Track Block Diagram

An in depth analysis of the performance of the DWT approach is provided in chapter 4. While the DWT approach works well to solve the DATMO problem, it does have some flaws. Since it uses a multiple hypothesis tracking approach, it has the potential to increase the amount of candidates that are proposed in each time step, particularly in noisy environments. The number of proposed candidates can be limited to ensure run time does not increase, however, this could impact the accuracy of the method. Furthermore, since the DWT approach only uses a single frame for the bounding box predictions in the 2D PIXOR component, it suffers from the oscillation issue identified in Chapter 2. which slightly decreases performance. In the next section, a pure neural network approach is proposed to solve the DATMO problem from end to end in a single pass using the PIXFOR architecture proposed in Chapter 2., an approach that can alleviate these afformentioned issues.

3.4 PIXFOR for DATMO

As shown in in the previous chapter using a recurrent model for object detection can improve the detection performance. However, the recurrent model of PIXFOR allows for even more powerful predictions. Since it is trained on time series data, the recurrent model can also make projections about the future states of objects. By slightly changing the architecture to expand the output and changing the training goal, it is possible to have the PIXFOR network solve the DATMO problem in a single forward pass.

Before discussing the modifications to the training regime and architecture, the assumptions of using PIXFOR for the DATMO problem need to be identified. The primary input for PIXFOR to make object detection predictions is a time series of point clouds. Specifically, these point clouds pass through the 2D PIXOR network and then the classification output passes through a recurrent block as identified in Figure 13. The key assumption to solve the tracking problem using this architecture is that a time series of tracks are also passed to the network that identify the objects in the point cloud time series input. This time series of tracks needs to have a one to one correspondence with the time series of point clouds. For most cases, this time series of tracks can be generated from PIXFOR itself, as the current prediction can be propagated to the next time step to represent a previous prediction. However, the issue of initial conditions arise.

An easy solution to this problem is to use the DWT method to generate this initial set of tracks since DWT does not require a time series. Once the 7 time steps required for the PIXFOR network are acquired by DWT, then the overall method can switch to PIXFOR for the remaining tracks. Alternatively, an empty point cloud can be passed for the first 7 time steps, however, this could cause brief prediction errors in the transient between switching from empty to non-empty point clouds.

Another assumption required is that there is a maximum number of objects that can be identified within one frame. This number can be chosen by the user of PIXFOR, however, it cannot be changed after training. The reason of this is due to the way tracking predictions are made. Each track is given a unique *id*, and the PIXFOR network tries to predict the unique track as a multi-class classification problem. The network uses the loss function defined in Equation 32. to train the network, in which the total amount of classes needs to be static. Therefore the track prediction tensor needs to be static

and the PIXFOR network can only predict a set finite amount of tracks. For this work, the maximum value is set to 20, which is more than the possible amount of QCars that could fit within the frame of a single LiDAR scan.

With these assumptions identified, the new network architecture can be defined. Following Figure 13., the first modification is to the amount of input channels to the recurrent block. Since the network needs a time series of the previous tracks as well, another channel is added that represents a 2D map of the track identifications at each time step, $\hat{N}_t$. Since this input is added to the recurrent block, it needs to be down sampled to have the same dimension as the classification prediction, which is the other channel. In this representation, each cell in the 2D track map represents a normalized identification number up to the maximum number of possible tracks in a single frame, n_{max}.

$$\hat{N}_t^{ij} = \frac{n}{n_{max}}, \quad n \in [0, n_{max}] \tag{62}$$

The goal of PIXFOR is then to propagate these ids to the correct objects in the future predictions. Since this version of PIXFOR uses multiple classes to represent the different tracks, there are two ground truth tensors that need to be used to train the network. The first of which is the same ground truth tensor, G, from Equation 9. The second ground truth tensor, $\tilde{G}$, has 3 dimensions and represents the multiple class ground truths. The first two dimensions represent the spatial location, similar to Equation 9. The third dimension represents the specific class index. This tensor is a boolean tensor, and only indicates the presence of a class at a certain location.

$$\tilde{G}_t^{ijk} = \begin{cases} 1 & \hat{N}_t^{ij} = \frac{k}{n_{max}} \\ 0 & \text{otherwise} \end{cases} \tag{63}$$

Due to this change, the output channel dimension of the recurrent block needs to be expanded to account for the multiple class channels at each time step. The overall PIXFOR architecture for DATMO is illustrated in Figure 16.

The classification output has 63 channels, 21 channels for each of the 3 time steps that are predicted. 20 of these channels represent the ids of the tracked object, and 1 channel is used to represent no tracked object at a specific location. The output contains predictions for the current and two future time steps.

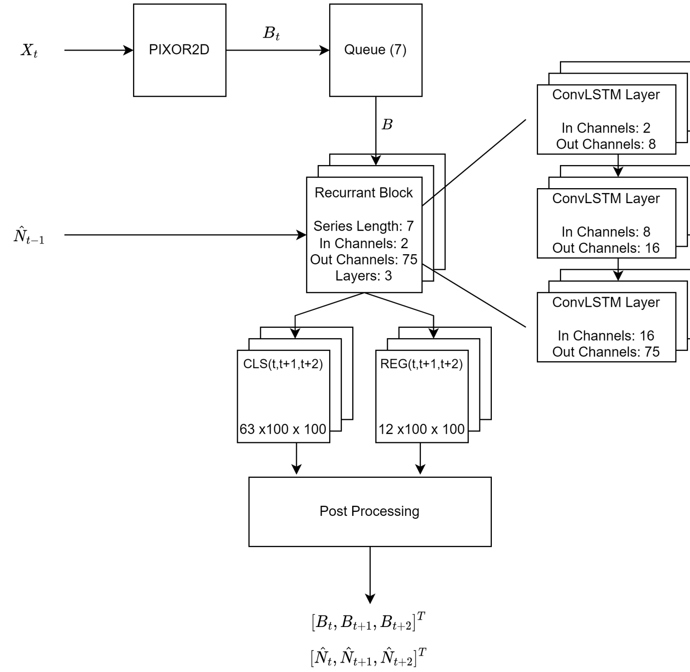


Figure 16: PIXFOR Architecture for DATMO

The training regime for this network is simple. A time series of point clouds with length 10 is used to train the network at each step. The first 7 frames of the time series are shown to the network, and the goal of the network is to predict the last 3 frames. These predictions can then be compared to the real last 3 frames to compute the loss. The loss functions used are the same as Equation 32. and Equation 33. for the classification and regression losses, respectively. This completes the modifications that need to be made to the PIXFOR architecture to allow it to solve the DATMO problem.

3.5 Section Conclusions

This chapter introduced the problem of object tracking and outlined traditional approaches to this problem. After identifying these traditional approaches, two novel methods for solving the DATMO problem were proposed. The first approach combined the 2D PIXOR network identified in Chapter 2. with traditional object tracking approaches to improve on the traditional approaches. The second approach is a pure deep learning approach that is the first to solve the DATMO problem in a single forward pass. In the next chapter, qualitative and quantitative analyses are done on the DWT method and PIXFOR method for DATMO to evaluate their performance.

4 Results

4.1 Data Environments and Metrics

4.1.1 Simulation Environment Introduction

The PIXOR2D, and PIXFOR methods introduced in this thesis were each trained and validated in a simulation environment before being implemented on real hardware. The DWT method, however, was only trained and validated in a hardware environment. The PIXOR2D network, and therefore the DWT method that builds upon it were trained and validated in an intersection scenario containing 4 QCars. This scenario was simulated in ROS using Gazebo using the same environment introduced in [54]. In this simulation, the ego vehicle is equipped with a 2D LiDAR modeled after the Rplidar A2. There are slight discrepancies between the simulated sensor and the real hardware, but they are negligible.

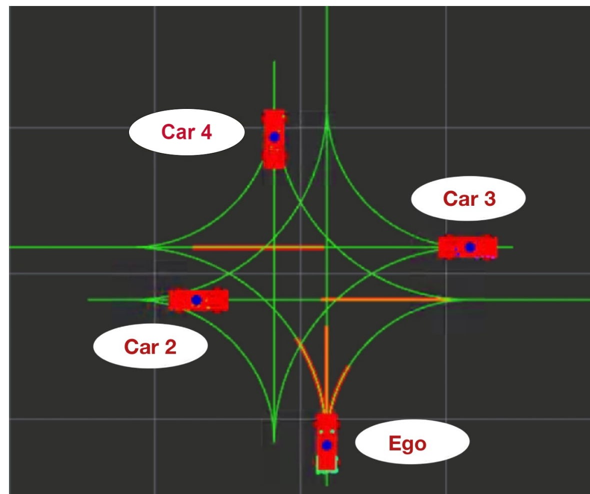


Figure 17: 4 Car Intersection Scenario Simulation

In the simulation, each QCar enters the intersection in a random order, and randomly chooses whether to go straight or make a turn. As each QCar passes through the intersection, the 2D LiDAR collects the point cloud data at a rate of 10 Hz. Once all the QCars have successfully exited the intersection, the simulation resets and another iteration begins. Using ROS, the topic from each LiDAR sensor is time synchronized with the vehicle pose topic from Gazebo for every LiDAR scan. The Gazebo pose information is then used as the ground truth for training the PIXOR2D network. Figure 17. shows a top view of the intersection simulation environment in Gazebo.

Due to the nature of the intersection environment, the vehicles in this scenario would often move slow. To better test the performance of PIXFOR for velocity predictions, a highway scenario was used, again using ROS and Gazebo. In the highway scenario, 6 QCars are initiated in random lanes of a 3 lane highway, each with random velocities. One QCar acts as the ego vehicle which is recording point cloud scans from its LiDAR. Once the ego vehicle reaches the end of the highway, the simulation resets and another iteration begins. The LiDAR data and ground truth pose from Gazebo are saved as described above. Figure 18 shows the highway simulation environment in Gazebo. The white objects are models of the QCars, and the one that has the white bounding box is the ego vehicle from which the data is collected. The blue lines are a visualization of the LiDAR scans from each QCar.

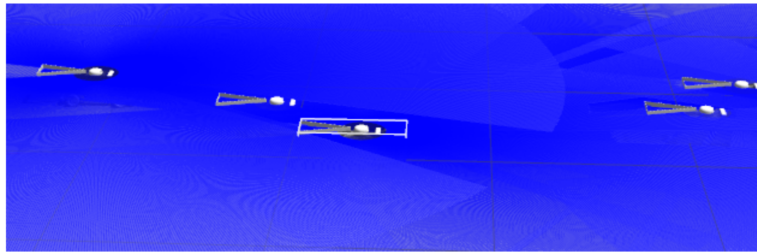


Figure 18: 6 Car Highway Scenario Simulation

4.1.2 Hardware Environment Introduction

As mentioned in Chapter 1., the experimental platform on which data was collected is the Quanser QCar, which is paired with the Optitrack system to obtain ground truth information. By placing visual markers on the QCars, the Optitrack system is able to track a rigid body of the QCar with a positional accuracy of 3×10^{-4} m and a angular accuracy of 0.05°. When running an experiment, the ground truths are obtained from Optitrack, and are time synched to the LiDAR scans from the QCar. This is done through a server which streams the pose obtained, as well as the time the measurement was made, from Optitrack directly to each QCar. A ROS node on the QCar then parses the streamed data and converts it to a timestamped pose ROS topic. This ROS node also listens to each LiDAR scan measurement that the QCar makes, and associates the Optitrack pose and LiDAR scan which share the closest timestamps. The streaming server sends messages at 100 Hz on average, which means that the average temporal error between the Optitrack measurement and the LiDAR measurement is 0.01 ± 0.005 s. The top speed of the QCars in the following experiments is 2 m/s, this means that the largest expected error due to temporal desynchronization is 3×10^{-2} m.

The experimental testing environments aim to emulate the simulated environments closely. A 4-way intersection scenario and a 2-lane highway scenario were created. The intersection scenario is almost identical to the simulation environment, however, the highway scenario has one less lane. Figure 19. shows the intersection environment where 4 QCars, each mounted with the rectangular bounding box, are about to begin navigation. This is the scenario which is used to evaluate the PIXOR2D and DWT methods. Each QCar enters the scenario, and randomly chooses to turn left, turn right, or go straight. LiDAR scans and Optitrack poses were collected for over 40 intersection scenarios to validate the methods in the following sections.

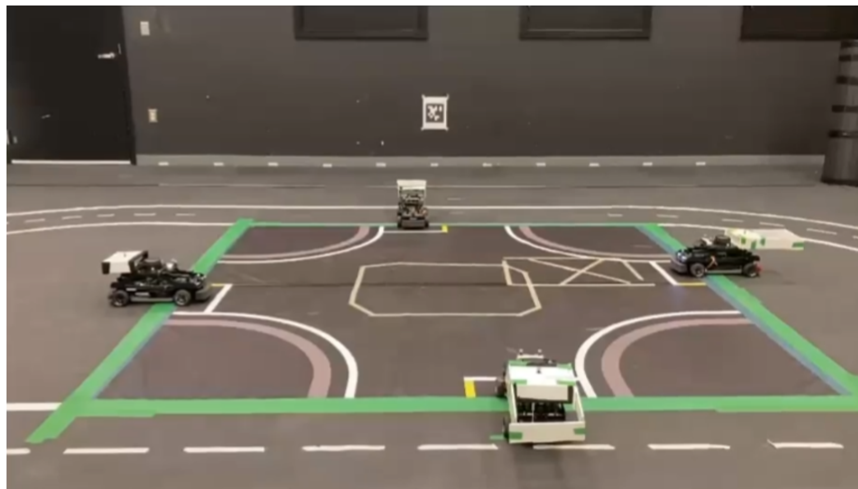


Figure 19: Intersection Experimental Environment with 4 QCars

Figure 20. shows the highway environment with a QCar. Each car goes around the closed highway loop at a random initial velocity, only adjusting velocity to avoid collisions. LiDAR scans and Optitrack poses were collected over hundreds laps of this highway scenario.

The hardware experiments performed in throughout the following section were conducted on two platforms. The first platform is the QCar itself, which is equipped with an Nvidia Jetson TX2. The second platform is a more powerful Nvidia Quadro RTX 4000 GPU with an Intel Xeon Silver 4210 2.20 GHz CPU. The first platform is to show that the methods developed in this thesis are indeed capable of scaling to meet the constraints of hardware with less computational power. The second platform is to show how the methods developed in this thesis can operate on more powerful hardware, and allows the methods to be compared state of the art methods more fairly. The predictions of the

Figure 20: Highway Experimental Environment

methods defined in this thesis are deterministic, and so, they do not need to be compared across different hardware. Instead, the comparison between the different platforms is purely in the run time cost. The final subsection covering experimental results performs a run time analysis of the methods proposed.

4.1.3 Data Quality

The deep learning approaches introduced in this thesis as well as the results outlined in this chapter are trained and evaluated on data sets collected within custom designed simulation and hardware environments. To ensure the PIXOR2D, and PIXFOR networks can make meaningful predictions that can generalize to many scenarios, a high data quality needs to be ensured for both training and validation data sets.

For the static point clouds used as an input to the PIXOR2D network, they key information that need to be captured by the labels within the data sets is the bounding box information. For the simulation environment, this can be easily obtained by using ROS to fuse the pose information of each object in the environment with the LiDAR scans from each QCar, as outlined previously. Since this is done all within one ROS environment, the synchronization between the LiDAR scans and pose information is instantaneous, allowing for perfect bounding box definitions within the simulation environment. For the hardware environment, ROS is also used to combine the LiDAR scans with pose information that

is streamed from the Optitrack system. However, since this is done by a separate system, there is a time delay that adds some imperfection to the ground truth bounding boxes as outlined in the previous section. In most cases, these errors are negligible, however, there are some cases such as when a QCar is moving quickly, or when the QCar wifi connection is weakened where this error could be more significant. Thus, to ensure that quality data is maintained, all collected data is parsed manually. In this process, each point cloud is visualized first without any bounding boxes identified, and manual predictions are made to identify bounding boxes. Then, the ground truth information is highlighted. If the ground truth information is largely skewed from the manual predictions, such as ground truth bounding boxes being plotted over regions of the point cloud where there are no visible points, then the specific data entry is removed from the data set. This is done before any training or validation to ensure there is no bias in how the data is removed.

For the time series of point clouds that are used as an input to the PIXFOR network, the key information is not just the bounding box information of LiDAR scans, but also the time difference between each LiDAR scan. Like with the static point clouds, the simulation environment for the time series of point clouds can capture the bounding box information instantaneously. One issue of the simulation environment that is not representative of a real sensor however, is that each LiDAR scan that is recorded has a uniform time step between the previous scan. This is why it is important to train models both on simulated data sets and hardware data sets. In the hardware environment, the time step between each scan varies, but the average rate is 10Hz. This variance in time step can be large depending on how long the QCar has been running. Thus, to ensure a consistent time step, a data series is only collected if the time step between each point cloud in the series is 0.1 ± 0.05 seconds.

4.1.4 Evaluation Metrics

One way to evaluate the performance of a neural network is to measure the precision and recall. The precision, or the positive predictive value, is the fraction of correct predictions that the network makes over the total number of predictions made by the network. For a set of predictions, $\hat{Y}$, and a set of ground truths, Y, the precision, P can be computed as follows.

$$P = \frac{|\hat{Y} \cap Y|}{|\hat{Y}|} \tag{64}$$

It should be noted that the precision can also be expressed as the ratio of true positives, $|TP|$, to the sum of the true positives and false positives, $|FP|$.

$$P = \frac{|TP|}{|TP| + |FP|} \tag{65}$$

The recall, or sensitivity of the network, is the fraction of correct predictions the networks make over the total number of items in the ground truth set. The recall, R, can then be computed as follows.

$$R = \frac{|\hat{Y} \cap Y|}{|Y|} \tag{66}$$

It should also be noted that the recall of the network can be expressed as the ratio of true positives to the sum of the true positives and false negatives, $|FN|$.

$$R = \frac{|TP|}{|TP| + |FN|} \tag{67}$$

By taking the product of the precision and recall, the average precision (AP) is found, which is a common metric for evaluating the performance of a network. Another way of combining the precision and recall to evaluate the performance of the network is the F_1 score, which is twice the product of the precision and recall over the sum.

$$AP = PR \tag{68}$$

$$F_1 = \frac{2PR}{P + R} \tag{69}$$

The F_1 score is one of the most common methods to evaluate the detection performance of a neural network. It is also a good metric to use to evaluate the tracking performance, as is done in [22]. However, it should be noted that there are some other tracking metrics such as the multiple object tracking accuracy (MOTA) and multiple object tracking precision (MOTP) that are also commonly used. The MOTA is a metric which evaluates the false predictions made on a specific frame, by comparing the quantity of false positives, false negatives, and ID switches, $|IDS|$, to total ground truths, $|GT|$.

$$MOTA = 1 - \frac{|FN| + |FP| + |IDS|}{|GT|} \tag{70}$$

An ID switch is a special case of a false positive prediction where two objects have their IDs swapped, usually when they come within close proximity with each other. The MOTP is a metric which evaluates the localization accuracy of a prediction by determining the average bounding box overlap, d, for a specific frame, i. In this sense, it is very similar to the intersection over union metric.

$$MOTP = \frac{1}{|TP|} \sum_{i=1}^{n} d_i \tag{71}$$

4.2 Simulations

4.2.1 PIXOR2D Results

As mentioned in Chapter 2., the PIXOR2D network has a tuning parameter, n, that can be adjusted before training to set the breadth of the network convolutions layers. The results shown in this section will be for a tuning factor of $n = 8$, called the ”heavy” network, and a tuning factor of $n = 2$, called the ”light” network. Both of these networks were trained on a data set consisting of 6916 2D point clouds and validated on a data set consisting of 4830 2D point clouds of the 4 QCar intersection environment.

Sometimes it is difficult to determine whether a prediction made by a network should be considered a true positive or a false positive. This usually occurs when a prediction partially overlaps the ground truth object, but does not entirely overlap the object. Often, the IOU value of the overlap is used, and if it is above a threshold, then the prediction can be considered a true positive. The KITTI data set [34] suggests that a 70% overlap (corresponding to an IOU value of 0.538) is a good threshold to use when evaluating object detection networks. Table 3. shows the results of the light and heavy 2D PIXOR networks for various IOU thresholds.

Table 3: 2D PIXOR Network Detection Performance

Network	IOU(% Overlap)	Precision	Recall	AP	F_1 Score
Heavy	0.333(50)	0.952	0.902	0.859	0.926
Heavy	0.538(70)	0.877	0.832	0.730	0.854
Heavy	0.666(80)	0.801	0.759	0.608	0.779
Light	0.333(50)	0.966	0.889	0.859	0.926
Light	0.538(70)	0.902	0.830	0.749	0.865
Light	0.666(80)	0.812	0.748	0.607	0.779

It should be noted that the distinction between true positive and false positive detections can affect the error measurement. For stricter IOU thresholds, the average positional error appears better since non-representative predictions are ignored. In real world use, it is not possible to evaluate the IOU of predictions since the ground truth is unknown; however, predictions with low IOU thresholds usually have low classification output values returned by the header network. Thus, it is possible to filter most of the poor predictions by only considering predictions above a certain classification value. The graphs depicted in Figure 21. show the distribution of position errors for the heavy and light networks, Figure 23. and Figure 22. show the yaw errors of the light and heavy networks, respectively. The graphs depict the errors for the three different IOU cases listed in the table above.

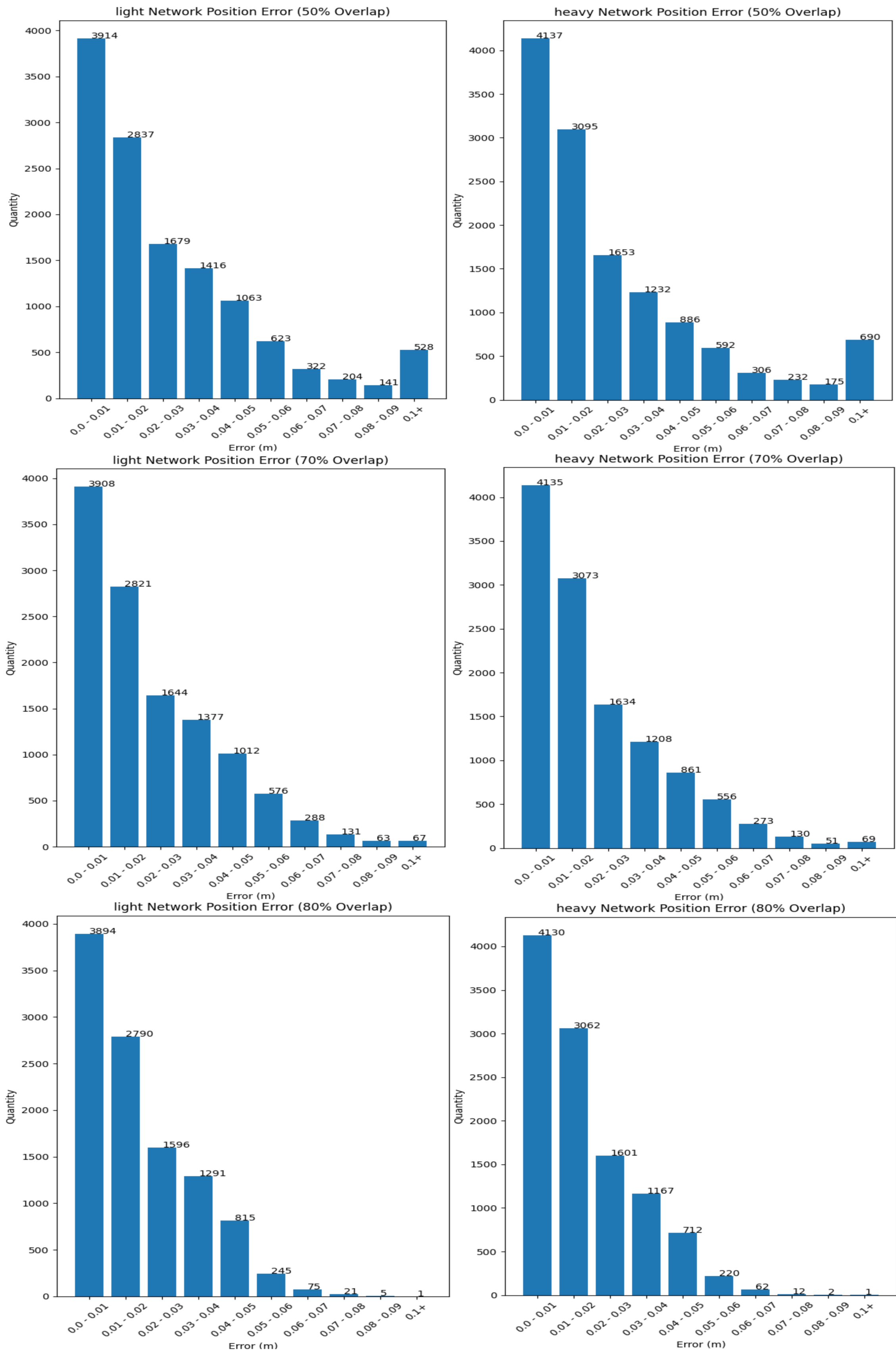
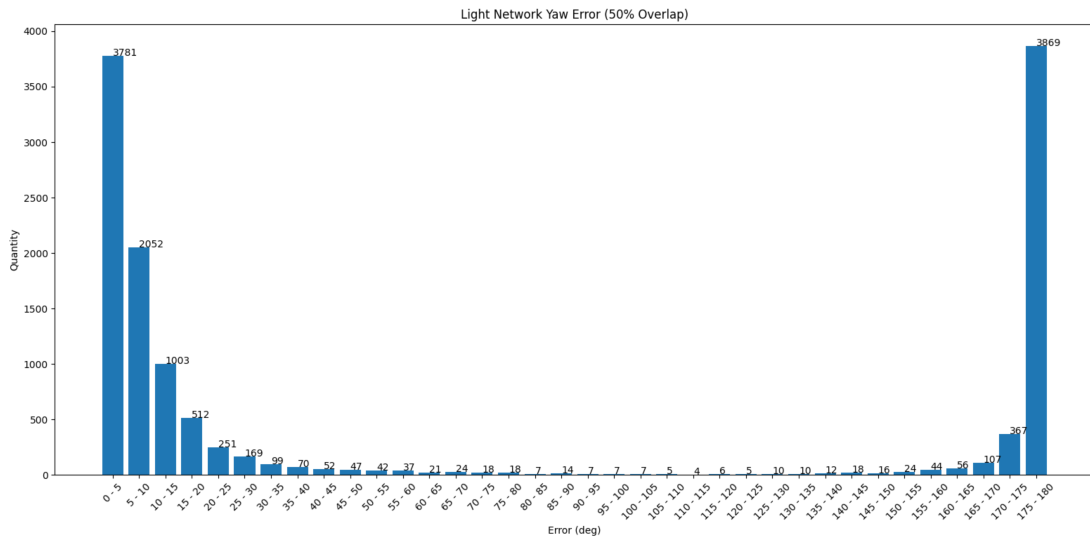
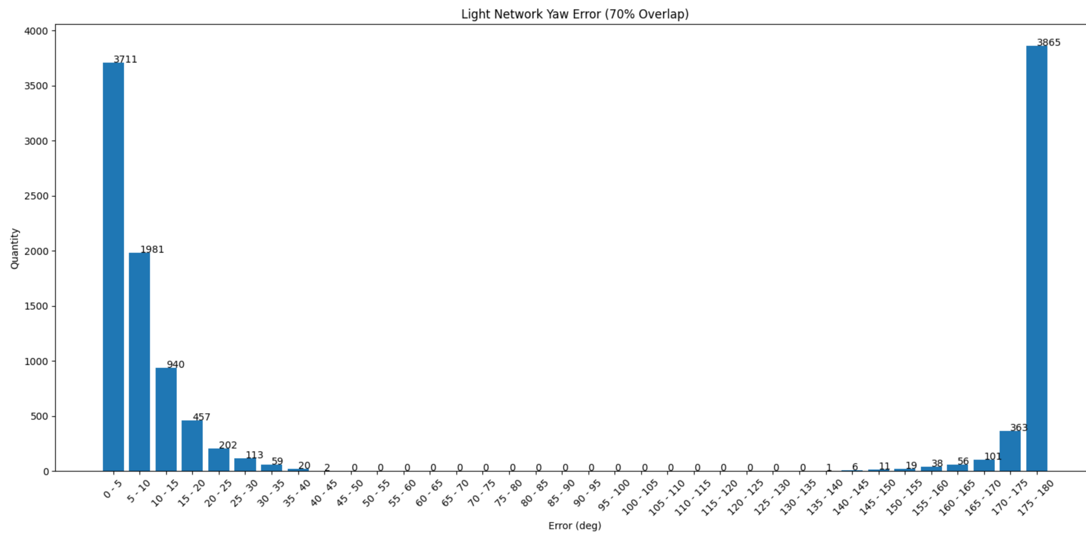


Figure 21: Position Errors for Light and Heavy 2D PIXOR Network on Simulated Data





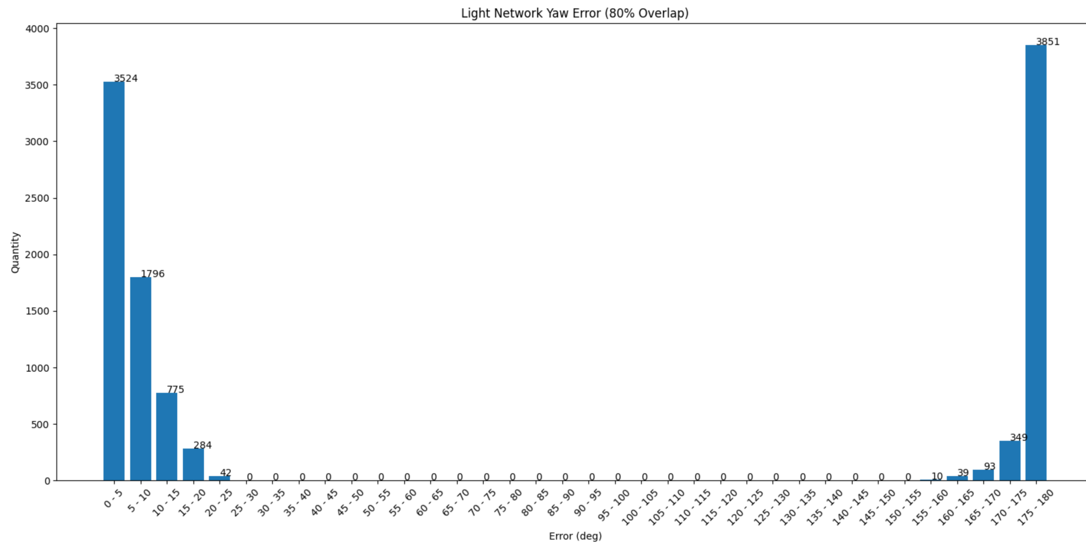
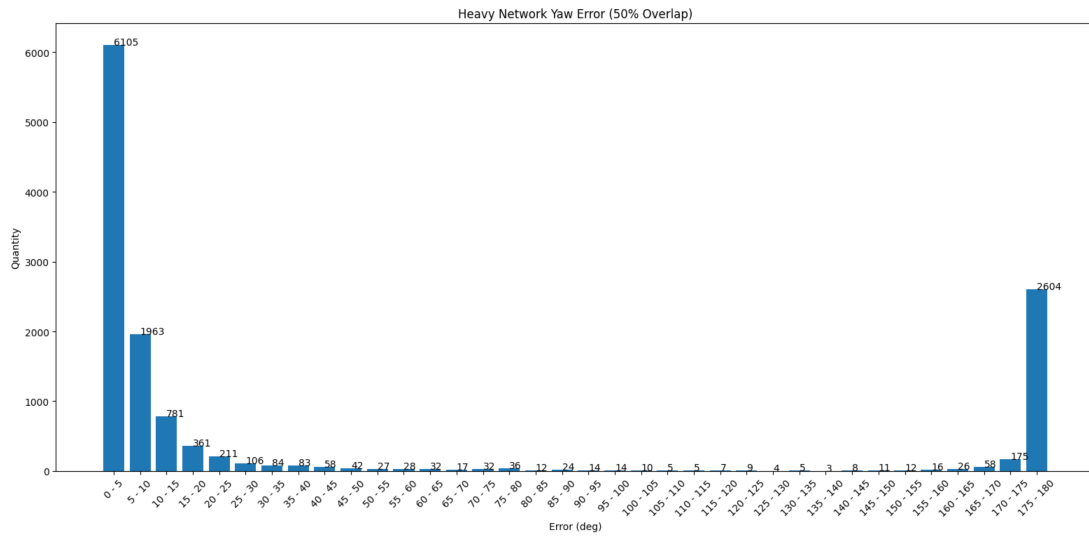
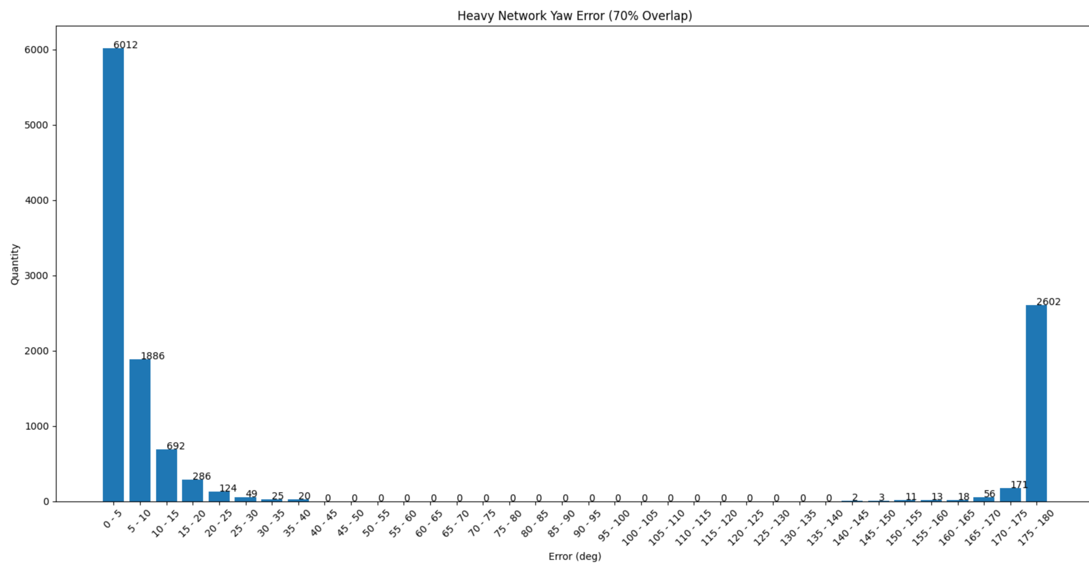


Figure 22: Yaw Error of Light PIXOR2D Network





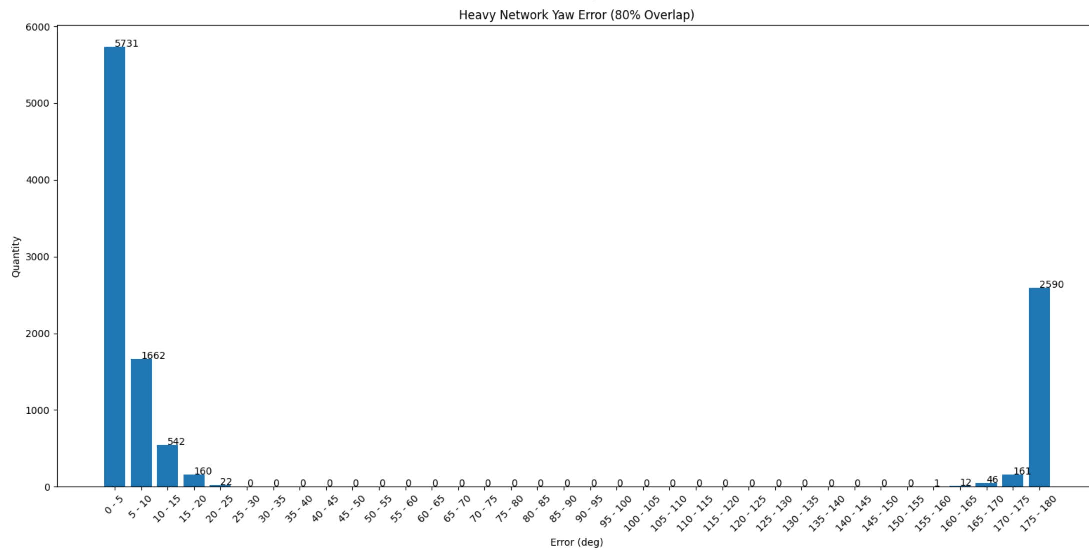


Figure 23: Yaw Error of Heavy PIXOR2D Network

For the heavy network, the average positional error for the 50% IOU condition is 4.54×10^{-2} m. The light network has an average positional error of 2.15×10^{-2} m for the same IOU threshold. The average yaw error for both networks is large, but this is mostly skewed due to a large number of outliers. The outliers creating this large yaw error stem from a flaw in the 2D PIXOR network when predicting shapes with rotational symmetry. Since the bounding boxes are symmetrical, it is sometimes difficult for the network to accurately predict the orientation, since there are two possible configurations. As such, many predicts fit an accurate bounding box, but have a 180 degree yaw angle error since the network predicts the wrong heading direction. This error can be resolved by using a non-rotationally-symmetric bounding box, such as an isosceles triangle shape, or by using time series data to enhance the predictions, as is done in the DWT and PIXFOR methods.

4.2.2 PIXFOR Results

The PIXFOR network was trained separately for each scenario. For the highway scenario, the PIXFOR network was trained on 3001 time series, each containing 10 temporal frames for a total of 30010 point clouds. The PIXFOR network was then evaluated on a separate 2826 time series, also containing 10 temporal frames for a total of 28260 point clouds. For the intersection scenario, the PIXFOR network was trained on 3000 time series, each containing 10 temporal frames for a total of 30000 point clouds. The PIXFOR network was then evaluated on a separate 3000 time series, also containing 10 temporal frames for a total of 30000 point clouds as well. As identified in Chapter 2., the first 7 frames are the input to the network, and the last 3 frames contain the ground truth information which the network is trying to predict.

The focus of the highway simulation is to evaluate the velocity predictions of the PIXFOR network. Since most of the simulations consist of vehicles moving with a constant heading angle, the yaw angle error is not evaluated for this simulation. Figure 24. shows the position error at the current time step, and 2 future time steps of the PIXFOR network for the highway simulation. In this simulation, a single time step has the duration of 0.067 seconds.

For the current time step, the average position error is 1.77×10^{-2} m and for the future time step, the average position error is 2.72×10^{-2} m. Figure 25. shows the velocity error of the PIXFOR network at the current time step. The average velocity error is 9.30×10^{-2} m/s. This error is heavily skewed

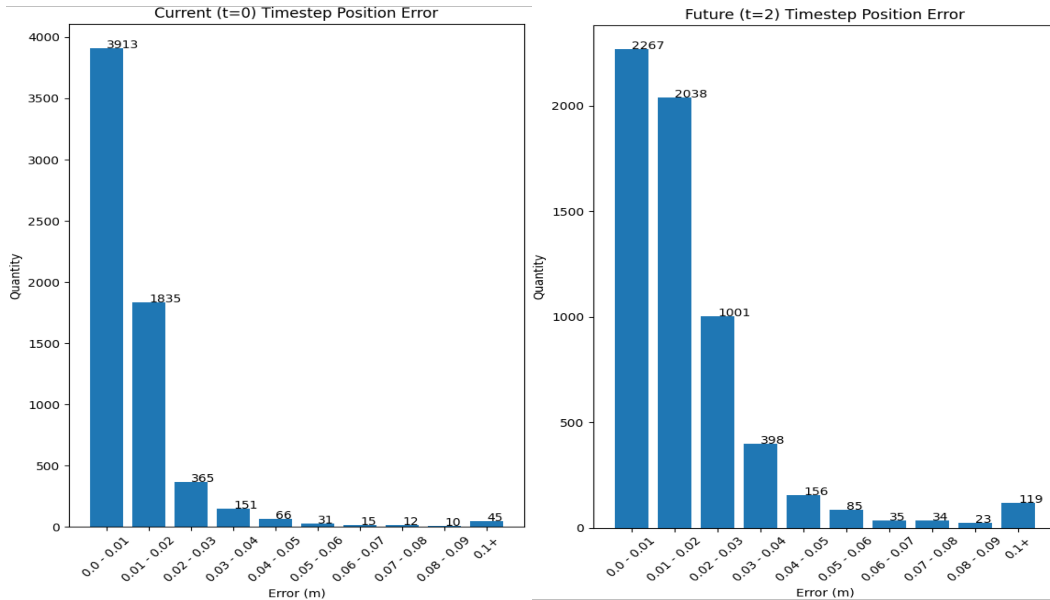


Figure 24: PIXFOR Highway Current and Future Time Step Position Errors

due to the large number of outlier predictions, as seen in the above figure. Many outliers can easily be filtered out when running the network in an inference mode, since many of these velocity predictions suggest the other QCars are moving faster than possible.

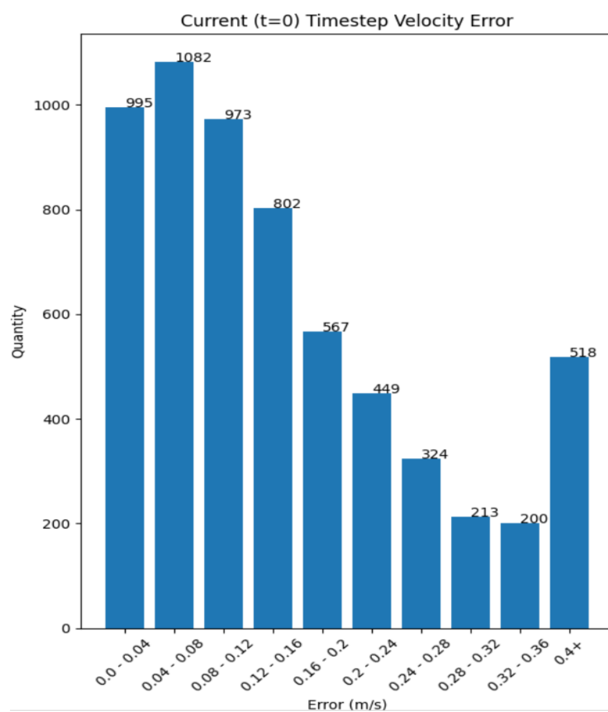


Figure 25: PIXFOR Velocity Error

The focus of the intersection simulation is to provide a more difficult environment to better evaluate the current and future position errors for the PIXFOR network. Figure 26. shows the position error at the current time step as well as 2 future time steps of the PIXFOR network for the intersection simulation. Like the highway simulation, the time step of this simulation is 0.067 seconds.

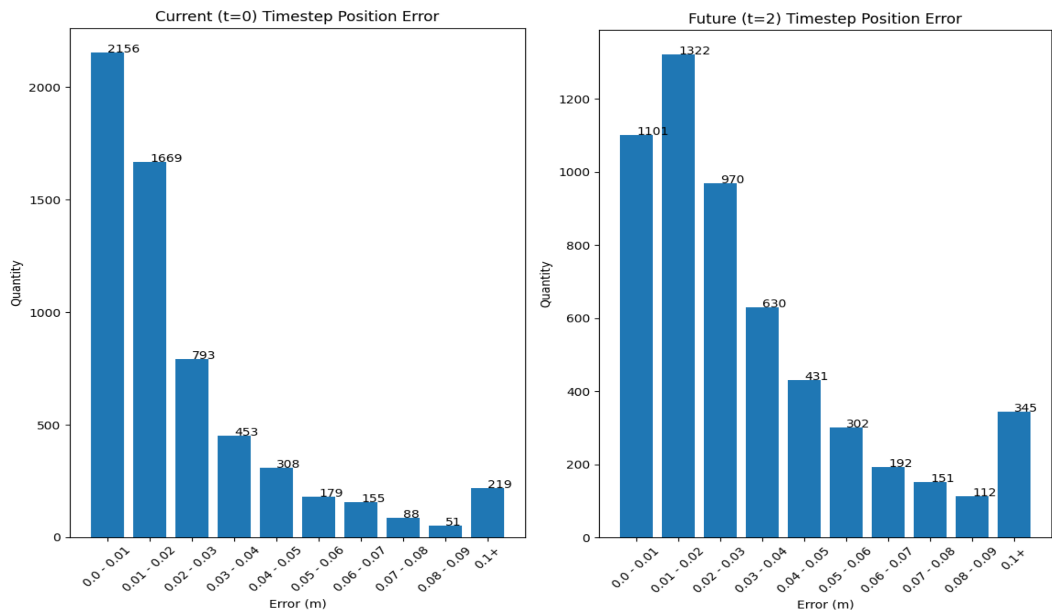


Figure 26: PIXFOR Intersection Current and Future Time Step Position Errors

For the current time step, the average position error is 2.96×10^{-2} m and for the future time step, the average position error is 4.67×10^{-2} m. Figure 27 shows the yaw angle error distribution of the PIXFOR network at the current and future time steps. As seen, there are still a few outliers around a 180 degree error due to the rotational symmetry, however, the quantity is greatly reduced now that a time series of point clouds is considered. Adjusting for these outliers, the average yaw error at the current time step is 8.02 degrees, and the average yaw error at the future time step is 8.71 degrees.

The PIXFOR tracking performance is summarized in Table 4., which shows the current time step performance, and Table 5. which shows the future time step performance. The tracks column represent how many ground truth objects could be tracked at the specific time step. The predictions column represents how many predictions were made by the network at the specific time step. The matched column represents how many of the predictions were correctly tracked at the specific time steps.

From the above tables, the PIXFOR network successfully tracked 94.3% of all objects, and 96.4% of the predictions made were valid at the current time step for the highway simulation. Similarly, the PIXFOR network successfully tracked 94.7% of all objects and 94.2% of the predictions made

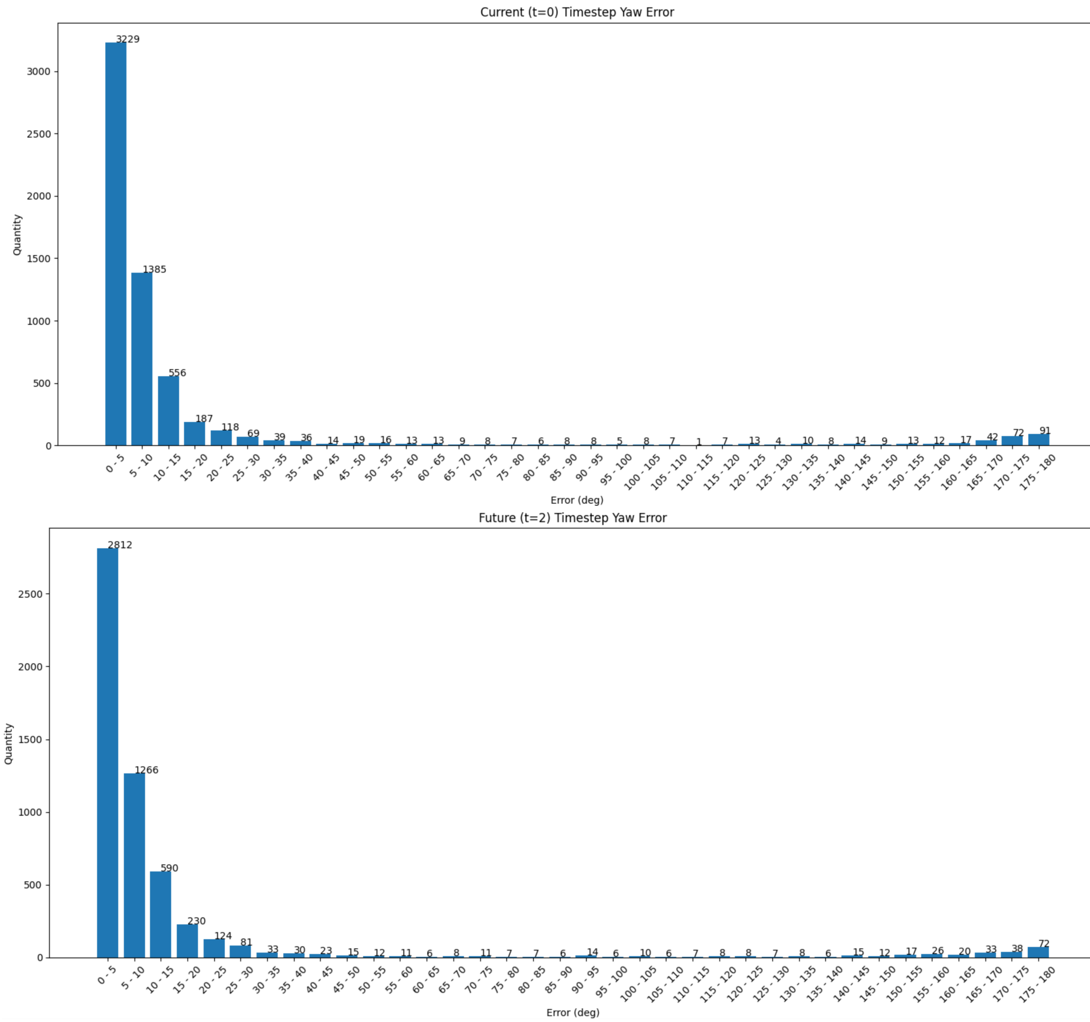


Figure 27: Yaw Error of Current and Future PIXFOR Network Predictions

Table 4: PIXFOR Tracking Performance at $t = 0$

Scenario	Tracks	Predictions	Matched	Precision	Recall	F_1 Score
Highway	6808	6661	6421	0.964	0.943	0.953
Intersection	6732	6762	6372	0.942	0.947	0.944

Table 5: PIXFOR Tracking Performance at $t = 2$

Scenario	Tracks	Predictions	Matched	Precision	Recall	F_1 Score
Highway	6707	6734	6115	0.908	0.912	0.910
Intersection	6656	6794	6025	0.887	0.905	0.896

were valid at the current time step for the intersection simulation. For 2 time steps in the future, the PIXFOR network successfully predicted the future location of 91.2% of the objects, and 90.8% of the predictions made were valid. Similarly, the PIXFOR network successfully predicted the future location of 90.5% of the objects, and 88.7% of the predictions made were valid.

4.3 Experimental Verification

4.3.1 PIXOR2D Results

The results in this section are from a separate PIXOR2D network was trained on hardware data. In this hardware data, the QCars have a large rectangular mounted box, mimicing the shape of a truck. In these tests, a small point cloud sampling size of 760 points per scan is used, which is why the larger box is required. A data set of 5318 point clouds of various intersection scenarios was collected to train the network, and a data set of 1332 point clouds was collected to validate the network. Table 6. shows the results of the light and heavy 2D PIXOR networks for various IOU thresholds on hardware data.

Table 6: 2D PIXOR Network Detection Performance on Hardware

Network	IOU(% Overlap)	Precision	Recall	AP	F_1 Score
Heavy	0.333(50)	0.959	0.962	0.923	0.960
Heavy	0.538(70)	0.922	0.925	0.853	0.923
Heavy	0.666(80)	0.852	0.854	0.728	0.853
Light	0.333(50)	0.962	0.942	0.906	0.952
Light	0.538(70)	0.902	0.883	0.796	0.892
Light	0.666(80)	0.783	0.767	0.601	0.775

Figure 28. shows the positional error of the heavy and light 2D PIXOR networks on the hardware validation data set. Figure 29. and Figure 30. show the yaw errors of the light and heavy 2D PIXOR networks on the hardware validation data set, respectively.

Under the 50% IOU condition, the average positional error for heavy network condition is 2.74cm and the light network has an average positional error of 3.37cm. Like the simulation tests, the average yaw error for both networks is large due to the large number of outliers. As with the simulation results, the yaw error stems from a flaw in the 2D PIXOR network when predicting shapes with rotational symmetry. An interesting thing to note is that the heavy network performs better for the hardware validations, but the light network performs better for the simulation validations. It is difficult to identify the reason for the difference in performance between the simulation and hardware environments since the simulation environment contains more data than the hardware environment.

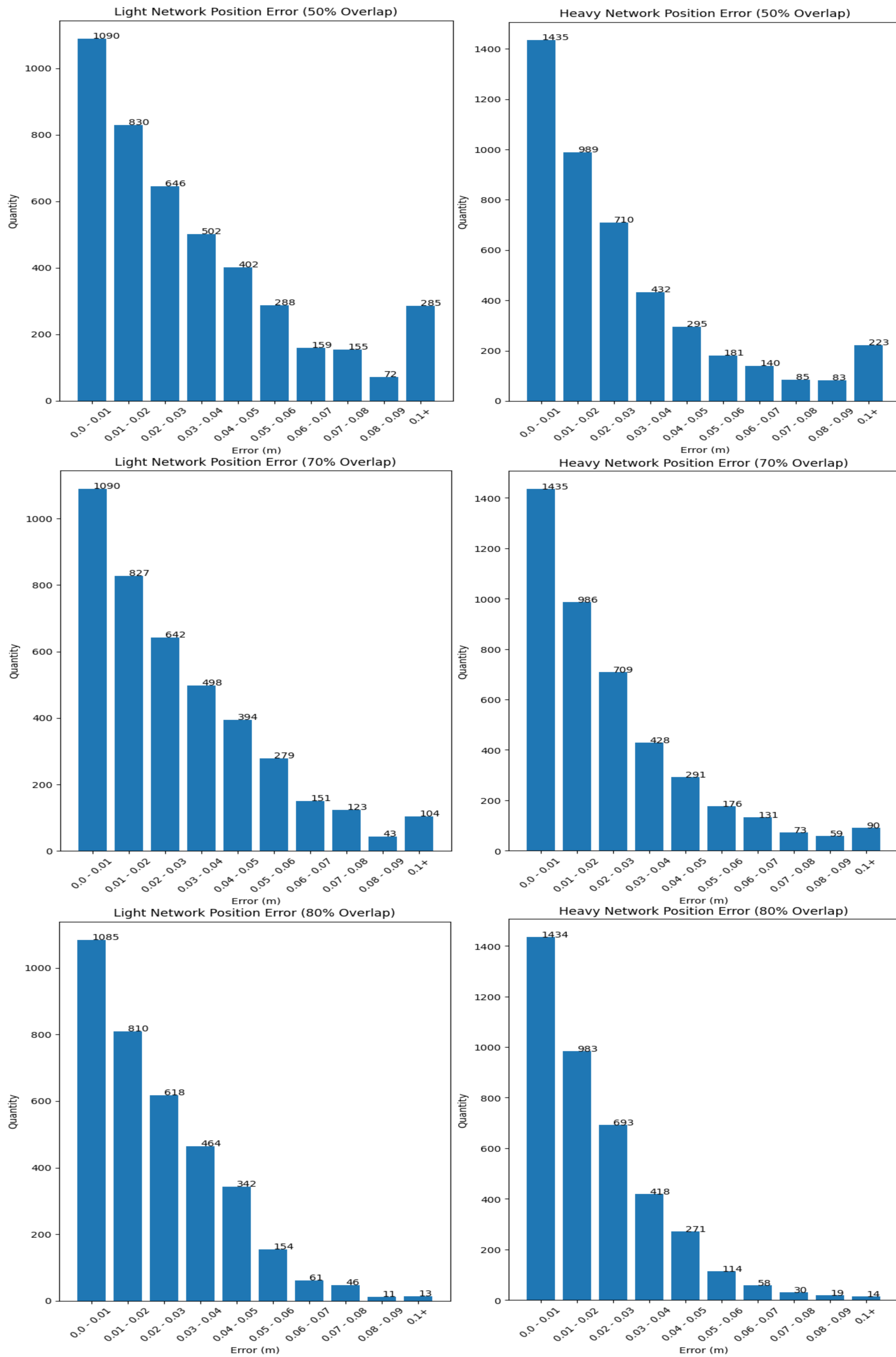
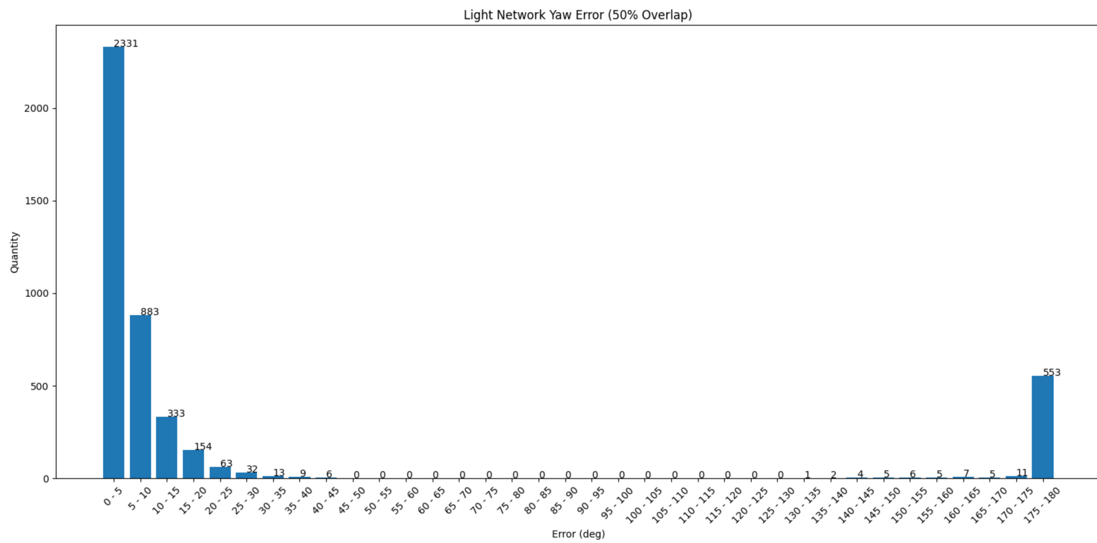
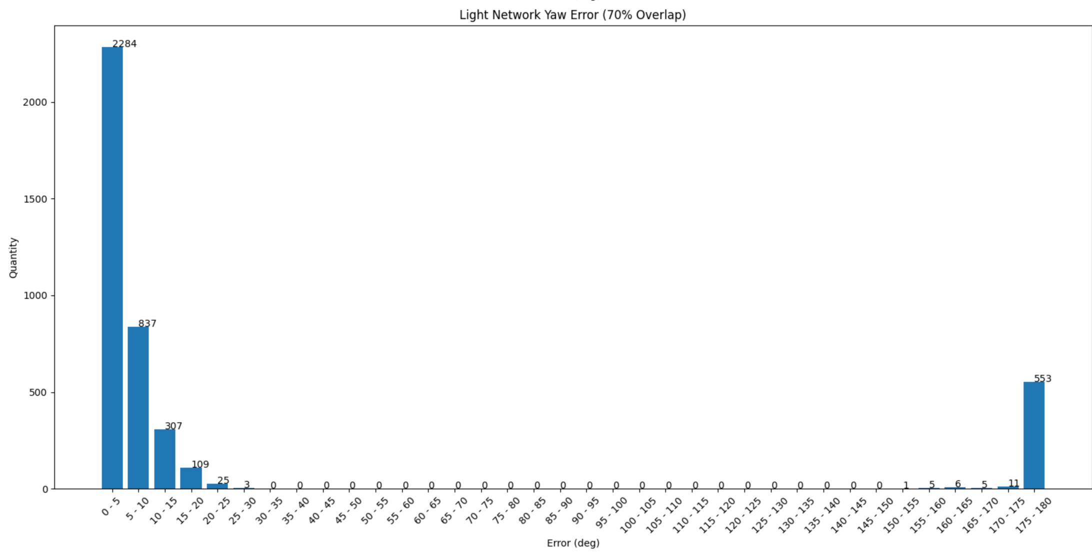


Figure 28: Position Errors for Light and Heavy 2D PIXOR Network on Hardware Data





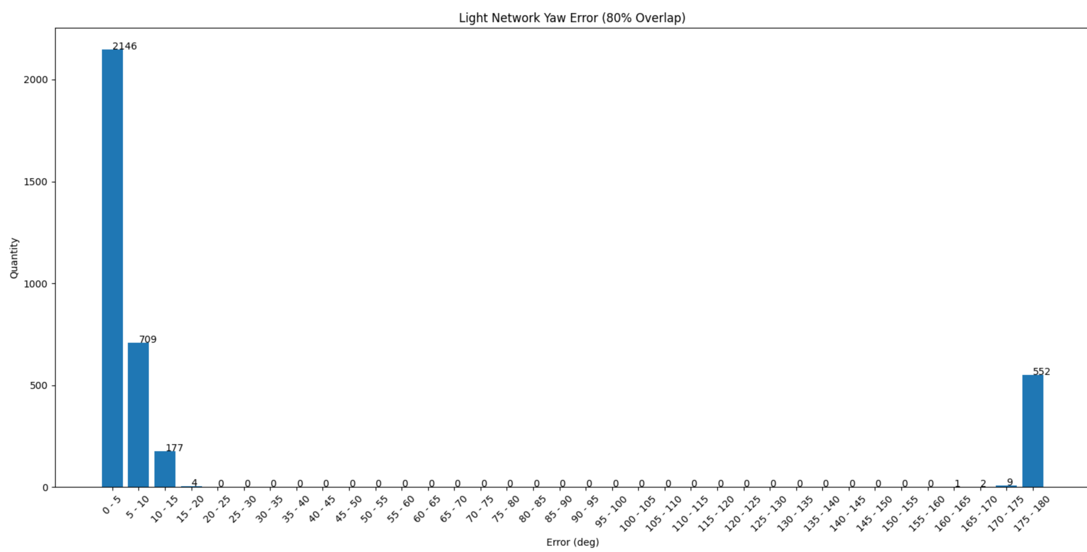
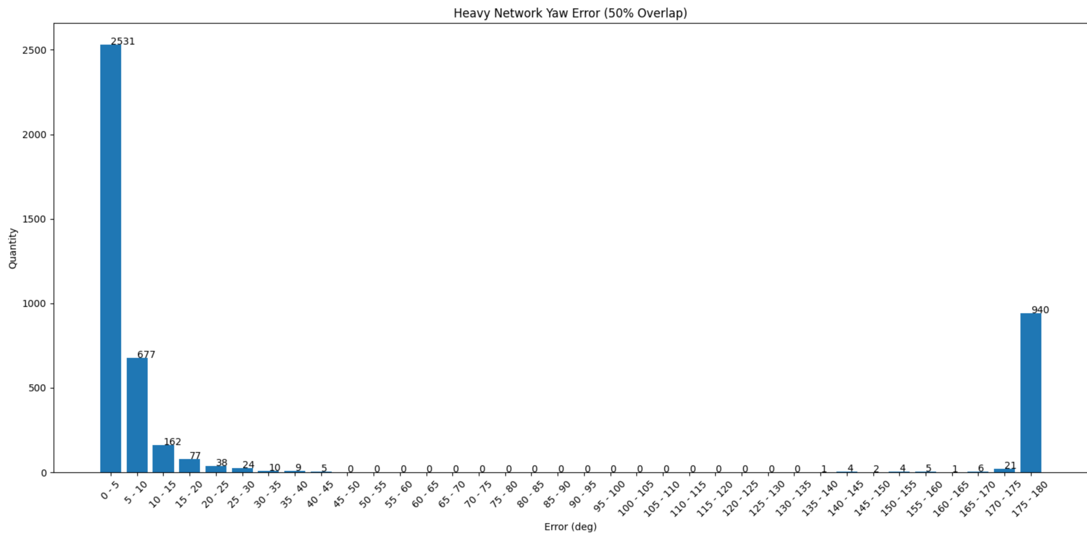
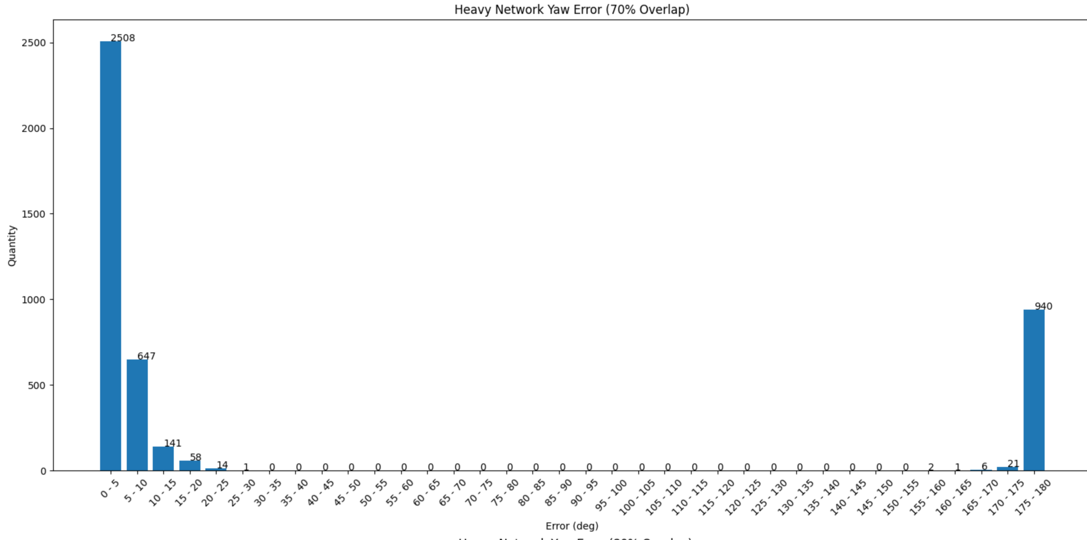


Figure 29: Yaw Error of Light PIXOR2D Network on Hardware Data





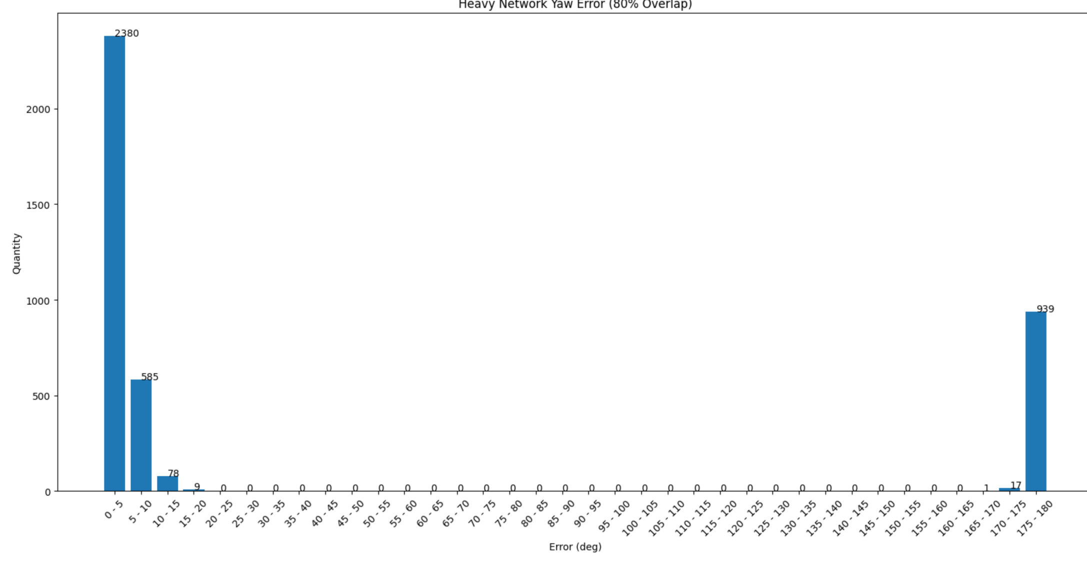


Figure 30: Yaw Error of Heavy PIXOR2D Network on Hardware Data

4.3.2 DWT Results

The DWT results in this section use the same hardware data sets from the previous section, however, instead of being validated on random samples, the DWT method needs to be validated on a sequence of point clouds. The training and validation data sets were taken from over 40 different intersection scenarios. The 1332 point clouds from the validation data set contain 4693 ground truth objects. Table 7. shows the tracking performance of the DWT method at two different IOU thresholds.

Table 7: Tracking Results of DWT

IOU	Ground Truths	Tracked Objects	Correctly Tracked	Precision	Recall	F_1 Score
50%	4693	4281	4235	0.989	0.902	0.943
70%	4693	4281	4144	0.968	0.883	0.924

The tracking performance of DWT is very good, and it does not drop tracks as easily as other methods. For example, Figure 31. shows the same exact intersection scenario from Figure 14., but using the DWT method instead of the tracking method proposed by [23].

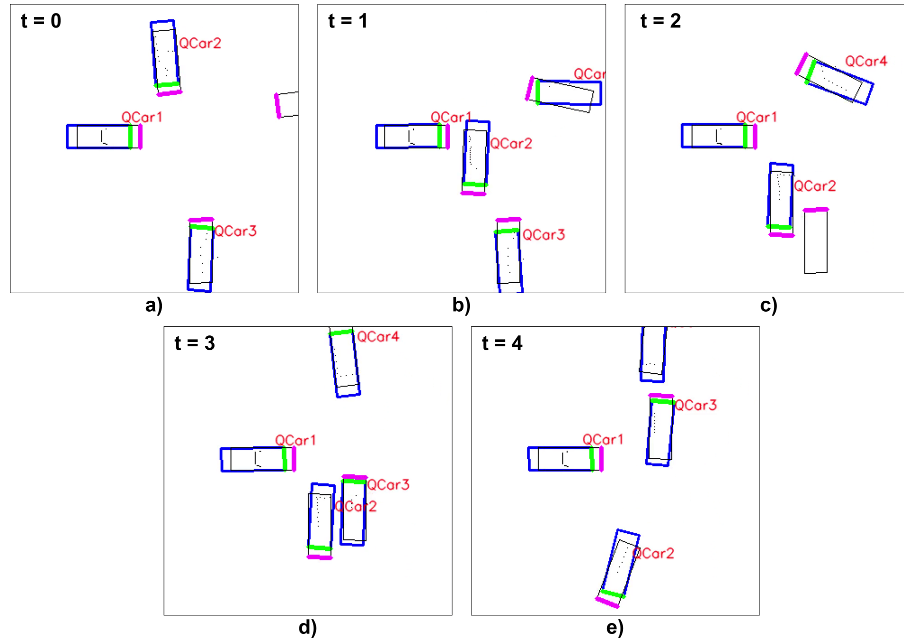


Figure 31: DWT Method Solving Bounding Box Shape and Merging Issues

In Figure 31., the black boxes represent the ground truth, and the magenta line represents the ground truth heading. The blue boxes represent the predicted bounding box, and the green line represents the predicted heading. The provided labels are a result of the tracking algorithm and are not added in

post for illustration. As seen in **b)** and **e)**, the proposed method can predict a suitable bounding box where the L-Shape tracker fails. Further, **d)** illustrates that the DWT algorithm solves the problem of merging multiple objects into one prediction. **c)** still shows no prediction for where QCar3 should be; however, this prediction is impossible since QCar3 is fully occluded by QCar2, and it is not represented by any points in the point cloud.

Figure 32. shows the positional error of the DWT method after validating on three full intersection crossing scenarios. The average positional error of this method is 9.11cm. Since it uses a combination of the 2D PIXOR network and Kalman filtering techniques based on naive tracks, the positional error is slightly worse than just using the 2D PIXOR network. This is because it can continue to predict bounding boxes on frames where the PIXOR network alone may drop a prediction, however, these bounding boxes are not as accurate.

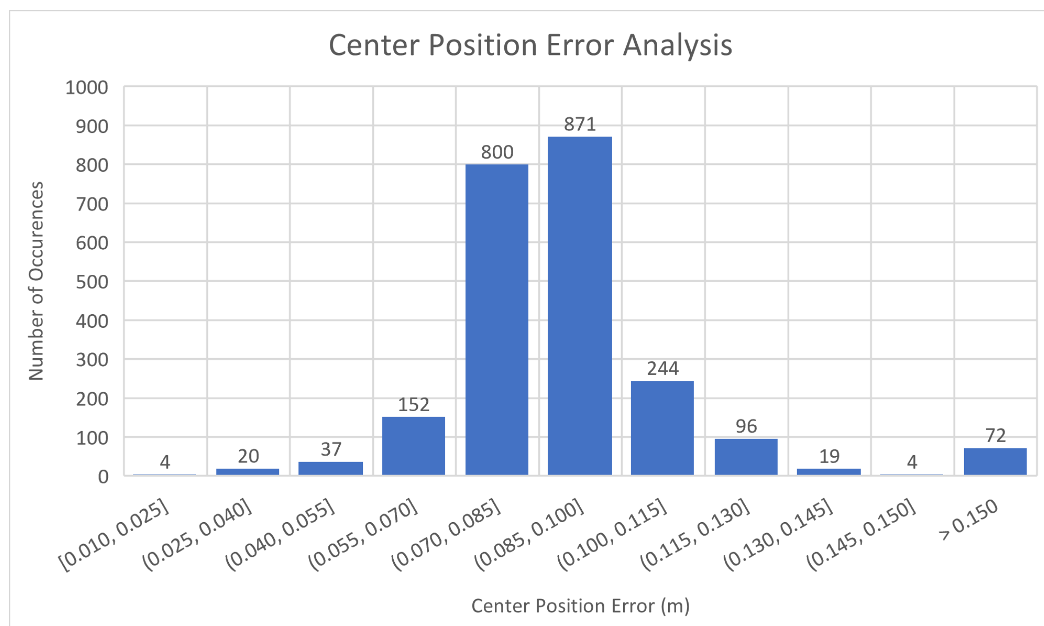


Figure 32: Bounding Box Center Position Error of DWT Method

Despite this slight increase in positional error, the benefit of the DWT method is in it's ability to solve the large yaw errors that arise in the 2D PIXOR network due to the rotational symmetry of the bounding boxes. Figure 33. shows a yaw error analysis of the DWT method on the same three full intersection scenarios, which is greatly improved over the 2D PIXOR network. The average yaw error of the DWT method is 4.23 degrees.

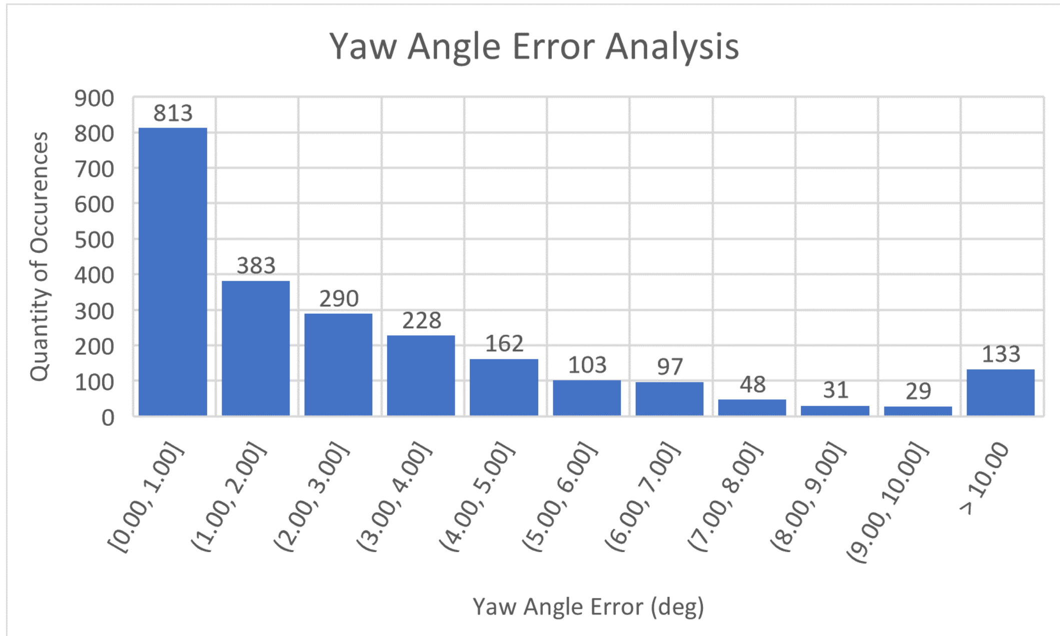


Figure 33: Bounding Box Yaw Error of DWT Method

4.3.3 PIXFOR Results

At this time, the only hardware envrionment on which the PIXFOR network has been evaluated is a highway scenario. In this scenario, the PIXFOR network was trained on 1357 time series, each containing 10 temporal frames for a total of 13570 point clouds. The validation was conducted on 433 time series, also containing 10 temporal frames for a total of 4330 point clouds.

Figure 34. shows the position error at the current time step beside the position error for 2 future time steps of the PIXFOR network. For the Rplidar-A2, the time step between scans is not constant, but is approximately 0.1 seconds on average. For the current time step, the average position error is 2.96cm and for the future time step, the average position error is 4.02cm. The hardware performance of the PIXFOR network is slightly below that of the PIXOR2D network, but it should be noted that the available training data is much more limited compared to the PIXOR2D network. Figure 35. shows the yaw error distribution of the PIXFOR network at the current time step and 2 time steps in the future. Compared to the simulation results, the hardware network has almost no outliers. This is because the hardware environment uses an isosceles triangle shaped bounding box, which does not have the issue of pertaining to rotational symmetry. The average yaw error is 6.44 degrees at the current time step, and 7.21 degrees at the future time step. As more training data can be collected for the

hardware scenario, it is likely that the hardware performance of the PIXFOR network will approach the theoretical performance that is achieved in the simulation results. At this time, the velocity predictions of the PIXFOR network have not been evaluated on hardware data.

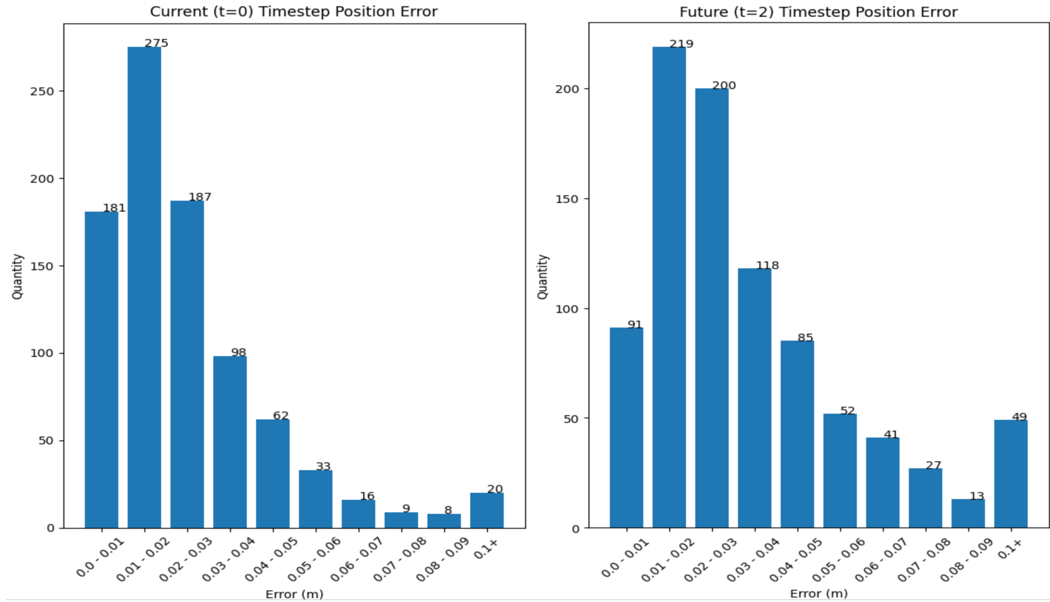


Figure 34: PIXFOR Current and Future Position Errors

The tracking results of the PIXFOR hardware highway tests are summarized in Table 8., which shows the current time step tracking performance, and Table 9. which shows the future time step tracking performance.

Table 8: PIXFOR Tracking Performance at $t = 0$

Scenario	Tracks	Predictions	Matched	Precision	Recall	F_1 Score
Highway	982	912	884	0.969	0.900	0.933

Table 9: PIXFOR Tracking Performance at $t = 2$

Scenario	Tracks	Predictions	Matched	Precision	Recall	F_1 Score
Highway	980	932	874	0.938	0.892	0.914

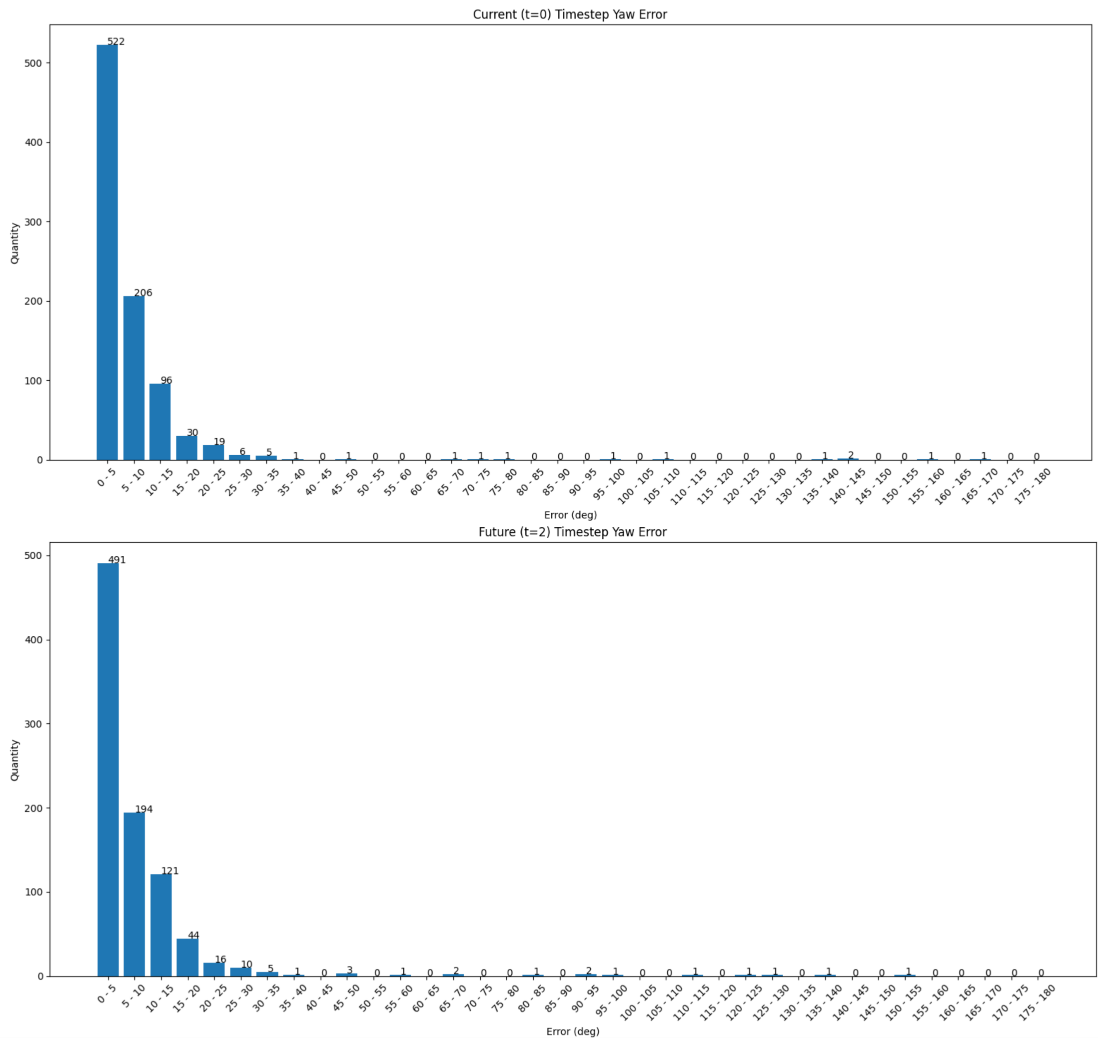


Figure 35: PIXFOR Current and Future Yaw Errors

4.4 Run Time Analysis

A run time analysis was performed on the PIXOR2D and PIXFOR networks to show that the proposed methods are capable of operating in real time, even on hardware with less computational resources. For this test, all networks are trained with a scaling parameter of $n = 2$, and Nvidia TensorRT [55] was used to compile the models running on the Nvidia Jetson TX2 to improve its inference performance. Table 10. shows the run time performance of both networks on both hardware.

Table 10: Run Time Analysis of PIXFOR and PIXOR2D Networks

Hardware	Network	Forward Pass (ms)	Post Processing (ms)	Total (ms)
Quadro RTX 4000	PIXOR2D	17.8	21.7	39.5
Quadro RTX 4000	PIXFOR	35.4	39.2	74.6
Jetson TX2	PIXOR2D	5.4	35.3	40.7
Jetson TX2	PIXFOR	12.6	121.8	134.4

As seen in the above table, compiling a model with Nvidia TensorRT can greatly improve the forward pass time of a network, even if the network is running on a less powerful GPU. However, the CPU of the Jetson TX2 is not as powerful as the Intel Xeon Silver, and so, the post processing time is significantly reduced. As previously shown, the PIXFOR algorithm is capable of accurately predicting 2 time steps in the future. For the Rplidar-A2, the scan rate is 10 Hz (0.1 s per scan). This means that 2 time steps in the future occurs 0.2 seconds in the future. Since the total delay of the PIXFOR network is below this threshold, the network can provide useful information, even on the less powerful Jetson TX2.

4.5 Comparison with State of the Art Methods

When compared to other DATMO methods, such as the model based tracking approach outlined in [21], and the geometric model free approach outlined in [22], it can be seen that the DWT and PIXFOR methods preform well. Table 11. compares the DWT and PIXFOR methods to these state of the art methods. It should be noted that the data sets for each method are not the same, but the environments of the data sets used for DWT and PIXFOR aimed to emulate those in GMFA and MBT closely.

As can be seen, the PIXFOR method out preforms the other methods, however, it has not yet been validated on as large of a data set. However, the DWT method is a close second, and it has been validated on a sufficiently large data set.

Table 11: Object Tracking Results of DWT, PIXFOR and other DATMO Methods

Method	Ground Truths	Tracked Objects	Correctly Tracked	Precision	Recall	F_1 Score
MBT	4455	3646	2918	0.800	0.655	0.720
GMFA	4455	4396	3994	0.909	0.897	0.902
DWT	4693	4281	4144	0.968	0.883	0.924
PIXFOR	982	912	884	**0.969**	**0.900**	**0.933**

5 Conclusions and Future Work

5.1 Conclusions

This thesis proposes two new methods of solving the detection and tracking of moving objects problem that are capable of operating on both small scale and large scale autonomous systems. The first method, detect while track, combines a deep learning convolutional neural network approach with a more classical Kalman filtering and hypothesis tracking approach. The second method, PIXFOR, is a pure deep learning approach based on convolutional neural network and long-short term memory recurrent neural network architectures. The theory behind these two approaches are outlined in Chapters 2. and 3.

Using a small scale autonomous vehicle as a validation platform, both methods, as well as an intermediary method, PIXOR2D, are tested in simulated and real world environments, as described in Chapter 4. While the 2D PIXOR network predicts bounding boxes well, it has issues predicting the yaw angle if the object has rotational symmetry. The detect while track and PIXFOR methods solve this problem by considering a time series of point clouds instead of static point cloud. However, the detect while track method's reliance on a combination of the 2D PIXOR network and classical approaches solves this issue at the cost of an increased positional error. The PIXFOR network on the other hand is not only able to solve this yaw angle issue, but it also improves on the positional error of the bounding boxes while being able to track up to 20 moving vehicles.

The provided simulation and experimental results verify the DWT and PIXFOR methods. The DWT method was experimentally validated on over 40 intersection scenarios involving 4 vehicles. The PIXFOR method was validated in both a simulation and experimental intersection and highway environments. Furthermore, the PIXFOR network and 2D PIXOR network that underlies the DWT method were analyzed on a Nvidia Jetson TX2 to validate the run time performance on a less powerful computer that powers many small robots. The results from the tests demonstrate that the proposed methods have better F_1 scores than some state of the art methods, and are capable of running efficiently on less powerful hardware.

5.2 Future Work

The work completed in this thesis serves as a starting point for the application of deep learning techniques to the problem of detection and tracking of moving objects. While the methods demonstrated solve this problem well, there are still some aspects that could use further development.

The focus of this work was developing solutions to the detection and tracking of moving objects problem that are suitable for 2D point clouds. With a few modifications, the DWT and PIXFOR methods proposed can also be implemented for 3D point clouds. It would be good to evaluate 3D versions of these methods not only on the KITTI data set, but also the NuScenes data set [56] and Waymo data set [57] as well. These data sets contain 3D LiDAR data from real world driving scenarios and thus contain more noise than the data sets collected in this thesis, and so, modifications to the DWT and PIXFOR methods may also need to consider preprocessing techniques to improve signal to noise ratio.

Another future area of development that could enhance this work is to make the deep learning more generalized to all scenarios. However, the convolutional long-short term memory network architecture of the proposed PIXFOR method takes a long time to train for each scenario. Furthermore, training a generalized network that is capable of operating in all scenarios would require much more data, further increasing the training duration. Modern network models such as transformers [58] have greatly improved on LSTMs in making predictions for natural language processing, and are suitable for training on very large data sets. More recently, transformer models have been used for the task of object detection [59] and multi object tracking [60], however, at the current state, transformer models are not as powerful as deep convolutional models when it comes to these tasks. Shifting the PIXFOR approach to a convolutional transformer network could improve on the training duration while also being a powerful object detection and tracking model.

Furthermore, shifting PIXFOR to a transformer model could make it an applicable precursor to other areas of autonomous driving. The ability for the network to make future predictions can assist in solving the decision making problem. This can be seen in recent work where the attention mechanism was applied in tandem with deep reinforcement learning to have a vehicle learn when to make a lane change maneuver [61]. While this work uses image data, it is easy to see how a transformer version

of PIXFOR could serve as a backbone for a deep reinforcement learning network to learn a lane changing policy using point cloud data.

Bibliography

[1] *High Definition Lidar HDL-64e*, Velodyne, 2014. [Online]. Available: https://hypertech.co.il/wp-content/uploads/2015/12/HDL-64E-Data-Sheet.pdf

[2] I. Automotive, "ibeo lux: The trusted 3d sensor." [Online]. Available: https://www.ibeo-as.com/en/products/sensors/IbeoLUX

[3] Quanser, "Quanser qcar - sensor rich autonomous vehicle." [Online]. Available: https://www.quanser.com/products/qcar/

[4] *Flex 13 Data Sheet*, OptiTrack, 2012. [Online]. Available: https://d111srqycjesc9.cloudfront.net/Flex%/2013%20Data%20Sheet.pdf

[5] *RPLIDAR A2 Introduction and Datasheet*, Slamtec, 04 2021. [Online]. Available: https://bucket-download.slamtec.com/20b2e974dd7c381e46c78db374772e31ea74370d/LD208_SLAMTEC_rplidar_datasheet_A2M8_v2.6_en.pdf

[6] B. Fortin, R. Lherbier, and J.-C. Noyer, "Feature extraction in scanning laser range data using invariant parameters: Application to vehicle detection," *IEEE Transactions on Vehicular Technology*, vol. 61, no. 9, pp. 3838–3850, 2012.

[7] Z. Shen, H. Liang, L. Lin, Z. Wang, W. Huang, and J. Yu, "Fast ground segmentation for 3d lidar point cloud based on jump-convolution-process," *Remote Sensing*, vol. 13, no. 16, 2021. [Online]. Available: https://www.mdpi.com/2072-4292/13/16/3239

[8] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *International Journal of Robotics Research (IJRR)*, 2013.

[9] S. Lee, W. Lim, M. Sunwoo, and K. Jo, "Limited visibility aware motion planning for autonomous valet parking using reachable set estimation," *Sensors*, vol. 21, no. 4, 2021. [Online]. Available: https://www.mdpi.com/1424-8220/21/4/1520

[10] Taş and C. Stiller, "Limited visibility and uncertainty aware motion planning for automated driving," in *2018 IEEE Intelligent Vehicles Symposium (IV)*, 2018, pp. 1171–1178.

[11] J. Van Brummelen, M. O'Brien, D. Gruyer, and H. Najjaran, "Autonomous vehicle perception: The technology of today and tomorrow," *Transportation Research*

Part C: Emerging Technologies, vol. 89, pp. 384–406, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0968090X18302134

[12] J. M. Wood, “Nighttime driving: visual, lighting and visibility challenges,” *Ophthalmic and Physiological Optics*, vol. 40, no. 2, pp. 187–201, 2020. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/opo.12659

[13] P. R. Wurman, R. D’Andrea, and M. Mountz, “Coordinating hundreds of cooperative, autonomous vehicles in warehouses,” *AI Magazine*, vol. 29, no. 1, p. 9, Mar. 2008. [Online]. Available: https://ojs.aaai.org/index.php/aimagazine/article/view/2082

[14] M. Menna, M. Gianni, F. Ferri, and F. Pirri, “Real-time autonomous 3d navigation for tracked vehicles in rescue environments,” in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 696–702.

[15] L. Guan, Y. Chen, G. Wang, and X. Lei, “Real-time vehicle detection framework based on the fusion of lidar and camera,” *Electronics*, vol. 9, no. 3, 2020. [Online]. Available: https://www.mdpi.com/2079-9292/9/3/451

[16] P. Li, X. Chen, and S. Shen, “Stereo r-cnn based 3d object detection for autonomous driving,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[17] A. P. Sligar, “Machine learning-based radar perception for autonomous vehicles using full physics simulation,” *IEEE Access*, vol. 8, pp. 51 470–51 476, 2020.

[18] R. Nabati and H. Qi, “Rrpn: Radar region proposal network for object detection in autonomous vehicles,” in *2019 IEEE International Conference on Image Processing (ICIP)*, 2019, pp. 3093–3097.

[19] P. Wei, X. Wang, and Y. Guo, “3d-lidar feature based localization for autonomous vehicles,” in *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*, 2020, pp. 288–293.

[20] Y. Wu, Y. Wang, S. Zhang, and H. Ogai, “Deep 3d object detection networks using lidar data: A review,” *IEEE Sensors Journal*, vol. 21, no. 2, pp. 1152–1171, 2021.

[21] A. Petrovskaya and S. Thrun, "Model based vehicle detection and tracking for autonomous urban driving," *Autonomous Robots*, vol. 26, no. 2, pp. 123–139, 2009.

[22] H. Lee, J. Yoon, Y. Jeong, and K. Yi, "Moving object detection and tracking based on interaction of static obstacle map and geometric model-free approachfor urban autonomous driving," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 6, pp. 3275–3284, 2021.

[23] K. Konstantinidis, M. Alirezaei, and S. Grammatico, "Development of a detection and tracking of moving vehicles system for 2d lidar sensors," M.S. thesis, Dept. of Mech Eng. Delft Univ., Delft, NL, 2020. [Online]. Available: https://repository.tudelft.nl/islandora/object/uuid%3A103fe186-925e-46f7-8275-d746e7c47600

[24] Y. Ye, L. Fu, and B. Li, "Object detection and tracking using multi-layer laser for autonomous urban driving," in *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, 2016, pp. 259–264.

[25] W. Zheng, W. Tang, L. Jiang, and C.-W. Fu, "Se-ssd: Self-ensembling single-stage object detector from point cloud," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2021, pp. 14 494–14 503.

[26] B. Yang, W. Luo, and R. Urtasun, "Pixor: Real-time 3d object detection from point clouds," 06 2018, pp. 7652–7660.

[27] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, pp. 381–395, 1981.

[28] T. Kanungo, D. Mount, N. Netanyahu, C. Piatko, R. Silverman, and A. Wu, "An efficient k-means clustering algorithm: analysis and implementation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 7, pp. 881–892, 2002.

[29] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, vol. 18, no. 9, p. 509–517, sep 1975. [Online]. Available: https://doi.org/10.1145/361002.361007

[30] R. B. Rusu, "Semantic 3d object maps for everyday manipulation in human living environments," *KI - Künstliche Intelligenz*, vol. 24, pp. 345–348, 2010.

[31] F. Tombari and L. Di Stefano, “Object recognition in 3d scenes with occlusions and clutter by hough voting,” 11 2010.

[32] H. Chen and B. Bhanu, “3d free-form object recognition in range images using local surface patches,” *Pattern Recognition Letters*, vol. 28, pp. 1252–1262, 01 2004.

[33] E. Recherche, E. Automatique, S. Antipolis, and Z. Zhang, “Iterative point matching for registration of free-form curves,” *Int. J. Comput. Vision*, vol. 13, 07 1992.

[34] 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.

[35] Y. Guo, H. Wang, Q. Hu, H. Liu, L. Liu, and M. Bennamoun, “Deep learning for 3d point clouds: A survey,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 12, pp. 4338–4364, 2021.

[36] B. Li, T. Zhang, and T. Xia, “Vehicle detection from 3d lidar using fully convolutional network,” 08 2016.

[37] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, “Pointpillars: Fast encoders for object detection from point clouds,” in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 12 689–12 697.

[38] R. Q. Charles, H. Su, M. Kaichun, and L. J. Guibas, “Pointnet: Deep learning on point sets for 3d classification and segmentation,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 77–85.

[39] Z. Yang, Y. Sun, S. Liu, and J. Jia, “3dssd: Point-based 3d single stage object detector,” in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 11 037–11 045.

[40] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, “Focal loss for dense object detection,” in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 2999–3007.

[41] R. Girshick, “Fast r-cnn,” in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1440–1448.

[42] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[43] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *MICCAI*, 2015.

[44] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *CoRR*, vol. abs/1406.1078, 2014. [Online]. Available: http://arxiv.org/abs/1406.1078

[45] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 11 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735

[46] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W. K. Wong, and W.-c. WOO, "Convolutional lstm network: A machine learning approach for precipitation nowcasting," 06 2015.

[47] Llamazares, E. J. Molinos, and M. Ocaña, "Detection and tracking of moving obstacles (datmo): A review," *Robotica*, vol. 38, no. 5, p. 761–774, 2020.

[48] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of the ASME–Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.

[49] D. Reid, "An algorithm for tracking multiple targets," *IEEE Transactions on Automatic Control*, vol. 24, no. 6, pp. 843–854, 1979.

[50] T. Fortmann, Y. Bar-Shalom, and M. Scheffe, "Sonar tracking of multiple targets using joint probabilistic data association," *IEEE Journal of Oceanic Engineering*, vol. 8, no. 3, pp. 173–184, 1983.

[51] X. Zhang, W. Xu, C. Dong, and J. M. Dolan, "Efficient l-shape fitting for vehicle detection using laser scanners," *2017 IEEE Intelligent Vehicles Symposium (IV)*, pp. 54–59, 2017.

[52] H. Cho, Y.-W. Seo, B. V. Kumar, and R. R. Rajkumar, "A multi-sensor fusion system for moving object detection and tracking in urban driving environments," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 1836–1843.

[53] G. Borges and M.-J. Aldon, "Line extraction in 2d range images for mobile robotics," *Journal of Intelligent and Robotic Systems*, vol. 40, pp. 267–297, 07 2004.

[54] M. Yuan, J. Shan, and K. Mi, "Deep reinforcement learning based game-theoretic decision-making for autonomous vehicles," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 818–825, 2022.

[55] H. Vanholder, "Efficient inference with tensorrt," in *GPU Technology Conference*, vol. 1, 2016, p. 2.

[56] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "nuscenes: A multimodal dataset for autonomous driving," in *CVPR*, 2020.

[57] P. Sun, H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. Caine, V. Vasudevan, W. Han, J. Ngiam, H. Zhao, A. Timofeev, S. Ettinger, M. Krivokon, A. Gao, A. Joshi, Y. Zhang, J. Shlens, Z. Chen, and D. Anguelov, "Scalability in perception for autonomous driving: Waymo open dataset," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[58] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[59] H. Vaidwan, N. Seth, A. S. Parihar, and K. Singh, "A study on transformer-based object detection," in *2021 International Conference on Intelligent Technologies (CONIT)*, 2021, pp. 1–6.

[60] F. Ruppel, F. Faion, C. Gläser, and K. Dietmayer, "Transformers for multi-object tracking on point clouds," 2022. [Online]. Available: https://arxiv.org/abs/2205.15730

[61] Y. Chen, C. Dong, P. Palanisamy, P. Mudalige, K. Muelling, and J. M. Dolan, "Attention-based hierarchical deep reinforcement learning for lane change behaviors in autonomous driving," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2019.