8-10-2018

Object Detection Using Feature Extraction and Deep Learning for Advanced Driver Assistance Systems

Tasmia Reza

Object detection using feature extraction and deep learning for advanced driver assistance systems

By

Tasmia Reza

A Thesis
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Master of Science
in Electrical Engineering
in the Department of Electrical and Computer Engineering

Mississippi State, Mississippi

August 2018

Object detection using feature extraction and

deep learning for advanced driver

assistance systems

By

Tasmia Reza

Approved:

John E. Ball
(Major Professor)

Derek T. Anderson
(Committee Member)

Bo Tang
(Committee Member)

Jim Fowler
(Graduate Coordinator)

Jason M. Keith
Dean
Bagley College of Engineering

Name: Tasmia Reza

Date of Degree: August 10, 2018

Institution: Mississippi State University

Major Field: Electrical Engineering

Major Professor: John E. Ball

Title of Study: Object detection using feature extraction and deep learning for advanced driver assistance systems

Pages of Study: 66

Candidate for Degree of Master of Science

A comparison of performance between tradition support vector machine (SVM), single kernel, multiple kernel learning (MKL), and modern deep learning (DL) classifiers are observed in this thesis. The goal is to implement different machine-learning classification system for object detection of three-dimensional (3D) Light Detection and Ranging (LiDAR) data. The linear SVM, non linear single kernel, and MKL requires hand crafted features for training and testing their algorithm. The DL approach learns the features itself and trains the algorithm. At the end of these studies, an assessment of all the different classification methods are shown.

Key words: Advanced Driver Assistance Systems, Support Vector Machine, LiDAR, Convolutional Neural Networks

DEDICATION

To my professor Dr. John E. Ball.

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my gratitude to the Almighty Allah for giving me the opportunity, strength, and patience to complete my masters' studies. Then I would like to thank my parents for raising me as a strong and confident girl. I can't express in words how fortunate I am to have both of them in my life. They didn't think twice before sending me off alone to pursue my dream of higher studies. I would like to give special thanks to my mother for considering my education the highest priority than anything else even today. Now I would like to thank my elder sister for being such an inspiration for me. Those words still ring in my ears when you instructed me to focus on getting funding for my higher studies and you will take care of the rest. You have kept your promise throughout my graduate journey. I won't be exaggerating when I say that I have come this far mostly because of you. I also thank my brother-in-law for helping me in every way possible. I would like to thank my friends for all the motivations, love, and support.

I want to thank my major professor Dr. John E. Ball for welcoming me in his research group. You have been the best mentor anyone can imagine. I never felt hesitant to communicate with you regarding my studies as well as other problems while my stay at Mississippi State University. You guided and walked me through all the hard times while I was doing my master's. I really admire your patience and multi tasking skill which I lack in myself. I wish someday I could be a better human being like you. I would also like to express my

gratitude towards my committee members Dr. Derek T. Anderson and Dr. Bo Tang. Thank you for supervising my research work. I appreciate your ideas, insights, and guidance on this thesis. Last but not the least, I would like to thank my fellow graduate students Pan Wei and Lucas Cagle for all the help and support. I have learned a lot from you and my best wishes for your future endeavors.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

1.1 Motivation

Autonomous driving research efforts have been significant for quite some time. The autonomous driving system has many levels where the vehicle can be either partially or fully automated. An automated vehicle can have multiple abilities. It can identify and localize objects as well as can do path planning and free space mapping. Some other important applications are collision avoidance, automated emergency braking, blind spot warning, adaptive cruise control, lane departure warning, rear cross traffic warning, etc., to name a few. All of these applications are developed to avoid accidents, minimize damage to the car if an accident can't be avoided, and protect against loss of life and valuable property. These systems must to be highly reliable, operate in real–time, and be robust. To implement these systems, various sensors such as cameras, Light Detection and Ranging sensors (LiDARs), radars, ultrasonic sensors, etc. have been used. Among all these sensors, the LiDAR uses time–of–flight measurements from short laser pulses to accurately estimate ranges and angles to points in the environment. Since the LiDAR is an active device, it can perform well under low lighting, different temperatures and fog, rain, etc. weather scenarios. It is used in various autonomous driving system development due to its unique advantages.

1.2 Overview of the problem

This thesis work has been done for a research project funded by a tier one industrial company where the goal was to establish an autonomous driver assistance system for their industrial vehicle. The system has been developed to use an eight–beam Quanergy M8 LiDAR on an industrial vehicle to quarantine certain areas with valuable assets. I have developed a machine–learning classification system for object detection based on three-dimensional (3D) LiDAR. The proposed real–time system operates a LiDAR on an industrial vehicle. I have developed a set of 3D hand–crafted features which will be used in classification. Those features were used with a linear Support Vector Machine (SVM), a non linear kernel and multiple kernel learning (MKL) classifier. Finally the raw LiDAR data were used with a 2D CNN deep learning (DL) performance for this classification scenario. At the end results from multiple data collections are analyzed and presented. Moreover, the feature effectiveness and the pros and cons of the approach are examined.

1.3 Contributions

In this research work, multiple classifiers have been implemented for the 3D LiDAR object detection problem. At first, the linear Support Vector Machine (SVM) has been utilized for the hand–crafted features. The SVM performed very well in real–time for a limited number of features. Examination of the optimal linear features reveals that these features are not truly linearly separable. The next classifiers examined were two nonlinear SVM machines using multiple kernel learning (MKL). The multiple kernel approach has the advantage of combining more than one kernel for improved results, which is a difficult

problem, because choosing a specific kernel for a specific problem is not trivial. Both the single kernel and multiple kernel approaches performed better than the SVM as the data is nonlinear. As deep learning (DL) is a novel research topic for machine learning and shows phenomenal performance and results for classification, it was also implemented. All the state–of–the–art LiDAR point cloud data are classified using a convolutional neural network (CNN). Most of these methods are applied to a thirty–two or sixty–four beam LiDARs. As an eight–beam Quanergy M8 LiDAR was used for this research, the point cloud data were not as dense as the other LiDAR datasets thus, making it difficult to use a 3D CNN. A very basic 2D CNN was used for classification with the sparse LiDAR data. The 2D CNN implementation was successful but it did not outperform the SVM and MKL. There might be various solutions for this issue among which choosing the right parameters and tweaking the CNN can be one of the options. This thesis provides a comparison of the linear SVM, two nonlinear MKL, and CNN methods for the stated problem. In conclusion, my contributions are as follows:

1. An efficient method to extract features from 3D lidar data using a very sparse eight–beam LiDAR.
2. A system implemented in real–time that uses a linear SVM for discrimination of beacons in an industrial setting.
3. Examination of two MKL methods that provide higher accuracy results using a smaller number of features than the linear SVM.
4. An efficient method of transforming 3D LiDAR point clouds into 2D multi–channel imagery processed by a CNN.

CHAPTER II

BACKGROUND

2.1 Object Detection and Classification

One of the greatest (and still unsolved) challenges of computer vision is object detection. The objective of object detection is to identify and localize an object within an image. There are various methods and approaches which attempt to solve the object detection problem. Conservative model adaptation method in [6] identifies the worst case of adaptation process for object detection by calculating the maximum cross-entropy error. The conservative model adaptation then can be transformed into the classic min-max optimization problem. From there it is easier to find the adaptation parameters which minimize the maximum of the cross–entropy errors of the cover. A binary classification can be done by that adapted object detector. Different researchers have worked on detection for different objects. Pedestrian detection research [10] offers the opportunity to study diverse approaches of object detection. All these approaches use a sliding window paradigm. These methods then go through the steps of feature extraction, binary classification and dense multi–scale scanning of detection windows followed by a non maximum suppression. There are various detectors that can be applied in this scenario such as Viola and Jones (VJ), SHAPENET, Pose-Invariant (POSEINV), Histograms of Oriented Gradients (HOG), Multi-feature (MULTIFIR) and many more [10]. These examples show that

various detection problems can be approached by a variety of detectors. The appropriate detector then can be determined by comparing the performances of these detectors. Classification is closely related to detection. It is generally the labelling of an object which needs to be detected. There are many data classification methods described in [27]. The popular traditional methods are the decision tree, Bayesian networks (BN), Support Vector Machine (SVM), boosting, bagging, k-nearest neighbor (KNN) classifier etc. [15]. More modern methods use deep learning, neural networks (NNs), etc., which learn the features directly from the data instead of developing hand-crafted features. A few of the established classification methods are explored in the next sections.

2.1.1 Decision Tree Classifier

Decision tree classifiers have been applied in many diverse areas such as radar signal classification, character recognition, remote sensing, medical diagnosis, expert systems, speech recognition etc. [36]. It breaks up a complex decision into a combination of several more simple decisions. The final solution obtained by this method would resemble the intended and desired solution. One of the main features of decision tree classifier is the flexibility of using different feature subsets and decision rules at different stages of classification. It also has the capability of trade offs between classification accuracy and time/space efficiency. In a tree classifier, a data sample is tested against only certain subsets of classes. It eliminates unnecessary computations of each data sample testing against all classes compared to other classifiers. Decision trees have several shortcomings despite of performing well on some classification problems. Decision trees can have overlapping

in case of large number of classes. Thus the number of terminals will be larger than the number of actual classes. It will eventually increase the search time and memory space requirements.

2.1.2 Bayesian Network Classifier

In a Naive Bayesian Classifier (NBC) the main assumption indicates that every attribute i.e., every leaf in the network is independent from the rest of the attributes where the state of the class is variable [14]. It calculates the correlation between the dependent target and other independent variables [15]. This classifier is a probabilistic model and it tries to find the probability of a certain class in multiple disjoint events. The NBC is applied to learning tasks where each instance x is described by a conjunction of attribute values. The target function $f(x)$ can take on any value from some finite set V. A set of training examples of the target function is provided. The learner is asked to predict the target value for this new instance. It is a representation of the joint distribution over all the variables represented by nodes in the graph. The Bayesian Network has the possibility of taking into account prior information about a given problem. An important advantage of Naive Bayes is that the simple structure lends itself to comprehensible visualizations. It can also handle incomplete data sets. This classifier is used when there is large amounts of data and the attributes are independent of each other. It can provide more efficient output.

2.1.3 Support Vector Machine

The support vector machine (SVM) is a machine learning technique utilized for multiple group classification problems [8]. The general idea of a SVM classifier is to input

vectors that are non-linearly mapped to a very high dimensional feature space. In this feature space a linear decision surface is constructed which ensuring high generalization ability. The support-vector network was initially been implemented for the restricted cases where the training data can be separated without errors. Later it has been extended to non-separable training data. The SVM can be applied for highly complex decision surfaces in a high (even infinite) dimensional space. This algorithm is often compared to other techniques such as linear classifiers, KNN classifiers and NNs.

2.1.3.1 Linear Support Vector Machine (SVM)

In this section, a simple example of linearly separable two-class data is given to explain the linear SVM [42]. Figure 2.1 shows two linearly separable sets with the separating hyperplane shown. Let us consider the binary classification case with a set of training patterns x_i, i = 1,....,n. They are assigned to either of the two classes ω_1 and ω_2, with corresponding labels y_i = ± 1. The discriminant function of the SVM is used for classification, and is denoted by

$$g(x) = \omega^T x + \omega_0 \tag{2.1}$$

where if $g(x) < 0$ then the object is decided as class $+1$ or ω_1, otherwise, it is decides as class -1 or ω_2.

A separable set of points can be classified with many (potentially infinite) separating hyperplanes. The margin is the sum of the distances from the separating hyperplane to the closest example from class ω_1 and the closest example from class ω_2. The maximal margin

classifier determines the hyperplane for which maximizes the margin. The assumption is that the larger the margin, the better the generalization error of the linear classifier defined by the separating hyperplane. A variant of the perceptron rule introduces a margin $b > 0$ to seek a solution so that

$$y_i(\omega^T x_i + \omega_0) \geq b \tag{2.2}$$

Assuming linearly separable data, then the perceptron algorithm provides a solution for which all points, x_i are at a distance greater than or equal to the separating plane. Therefore, without loss of generality, a value $b = 1$ may be taken which will define the canonical hyperplanes, H_1: $\omega^T x + \omega_0$= +1 and H_2: $\omega^{T_x} + \omega_0 = -1$. From here it can be written

$$\begin{aligned} \omega^T x_i + \omega_0 \geq +1 \quad for \quad y_i = +1 \\ \omega^T x_i + \omega_0 \leq -1 \quad for \quad y_i = -1 \end{aligned} \tag{2.3}$$

Maximizing the margin implies that we seek a solution that minimizes ω subject to the constraints

$$C1 : y_i(\omega^T x_i + \omega_0) \geq 1 \quad i = 1, ..., n \tag{2.4}$$

A standard approach to optimization problems with equality and inequality constraints is the Lagrange formalism which leads to the primal form of the objective function, L_p given by

$$L_p = \frac{1}{2}\omega^T\omega - \sum_{i=1}^{n} \alpha_i(y_i(\omega^T x_i + \omega_0) - 1) \tag{2.5}$$

To solve the equation, we differentiate L_p with respect to ω_0 and ω and equate to zero. It yields

$$\sum_{i=1}^{n} \alpha_i y_i = 0 \tag{2.6}$$

$$\omega = \sum_{i=1}^{n} \alpha_i y_i x_i \tag{2.7}$$

The Kuhn–Tucker conditions provides sufficient and necessary conditions for minimizing an objective function subject to inequality and equality constraints. In the primal form of the objective function these conditions are

$$\frac{\partial L_p}{\partial \omega} = \omega - \sum_{i=1}^{n} \alpha_i y_i x_i = 0 \tag{2.8}$$

$$\frac{\partial L_p}{\partial \omega_0} = -\sum_{i=1}^{n} \alpha_i y_i = 0 \tag{2.9}$$

$$\frac{\partial L_p}{\partial \omega_0} = -\sum_{i=1}^{n} \alpha_i y_i = 0 \tag{2.10}$$

$$y_i(x_i^T\omega + \omega_0) - 1 \geq 0 \tag{2.11}$$

$$\alpha_i \geq 0 \tag{2.12}$$

$$\alpha_i(y_i(x_i^T\omega + \omega_0) - 1) = 0 \tag{2.13}$$

The condition 2.13 is known as the Karush-Kuhn-Tucker condition. It implies that for active constraints $\alpha_i \geq 0$; otherwise for inactive constraints $\alpha_i = 0$.

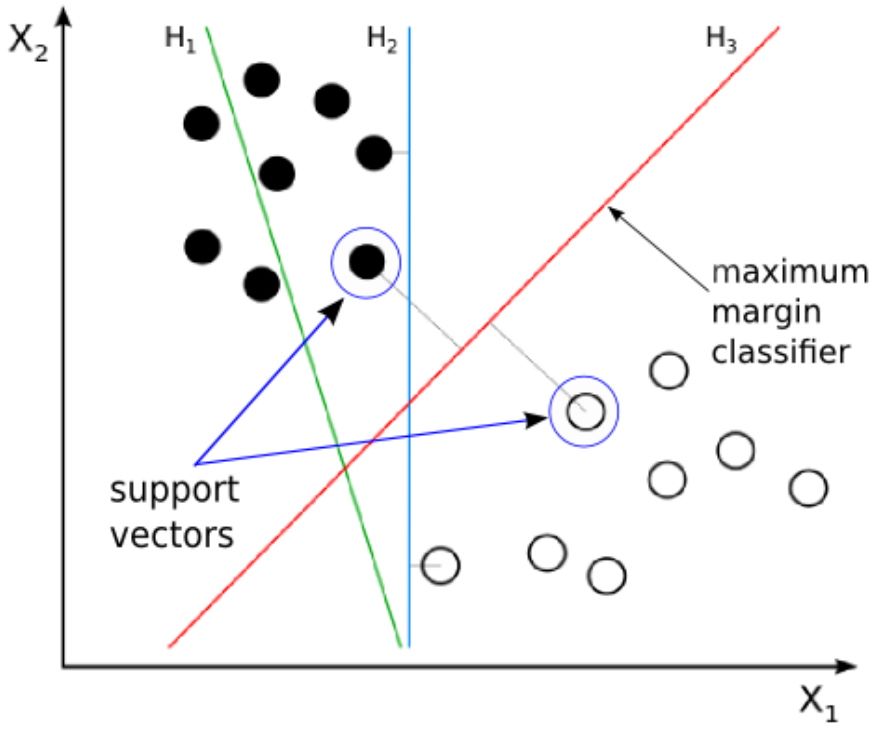


Figure 2.1 Linearly separable two-class SVM example.

2.1.3.2 Kernel (Non–Linear) Support Vector Machine (SVM)

The SVM can also be applied to non–linear classification problem. The simplest representation is the previously mentioned binary classification problem [42]. Here the support vector algorithm can be applied in a transformed feature space, $\phi(x)$, for some nonlinear function ϕ. The discriminant function can be written as

$$g(x) = \omega^T \phi(x) + \omega_0 \tag{2.14}$$

where the decision rule is,

$$\begin{aligned} \omega^T \phi(x) + \omega_0 > 0 \quad for \quad y_i = +1 \\ \omega^T \phi(x) + \omega_0 < 0 \quad for \quad y_i = -1 \end{aligned} \tag{2.15}$$

The dual form of the Lagrangian equation for this case becomes

$$L_D = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{i=1}^{n} \alpha_i \alpha_j y_i y_j \phi^T(x_i) \phi(x_j) \tag{2.16}$$

Here, y_i = ± 1, i = 1,..., n are class indicator values and α_i, i = 1,...., n are Lagrange multipliers satisfying

$$\sum_{i=1}^{n} \alpha_i y_i = 0, \quad 0 \leq \alpha_i \leq C \tag{2.17}$$

for the regularization parameter, C. Maximizing 2.16 subject to the constraints 2.17 leads to support vectors identified by nonzero values of α_i. The solution for ω is,

$$\omega = \sum_{i \in S\nu} \alpha_i y_i \phi(x_i) \tag{2.18}$$

Optimization of L_D and the subsequent classification of a sample relies only on scalar products between transformed feature vectors, which can be replaced by a kernel function

$$K(x, y) = \phi^T(x) \phi(y) \tag{2.19}$$

This is called the kernel trick where, because one can avoid computing the transformation $\phi(x)$ explicitly and replace the scalar product with $K(x,y)$ instead. The discriminant function becomes

$$g(x) = \sum_{i \in S\nu} \alpha_i y_i K(x_i, x) + \omega_0 \tag{2.20}$$

Some common types of kernels that may be used in an SVM are given in the table 2.1.

Table 2.1 Support vector machine kernels.

Nonlinearity	**Mathematical form**
Simple polynomial	$(1 + x^T y)^p$
Polynomial	$(r + \gamma x^T y)^p, \quad \gamma > 0$
Gaussian	$\exp\left(\frac{-\lvert x-y\rvert^2}{\sigma^2}\right)$
Sigmoid	$tanh(kx^T y - \delta)$

Here p, r, γ, σ, δ and k are variable terms that are used for the mathematical expressions.

2.1.3.3 Multiple Kernel Learning (MKL)

Another classification technique is the kernel method which is used for pattern recognition problems in machine learning. There is generally no way to predict the best kernel for a particular problem and thus *multiple kernel learning* (MKL) is used to learn the aggregation of a set of valid kernels into a single (ideally) superior kernel [31]. Recently it has been studied that using multiple kernels instead of a single one can improve the overall performance [18] depending on the dataset. There has been a significant amount of research on multiple kernel methods. Many researchers have worked toward developing

an algorithm that can automatically determine which kernels are useful and then combine them. An efficient and general MKL algorithm has been proposed in this paper [39]. This approach makes the MKL applicable for large scale problems by iteratively using existing support vector machine algorithm. Its downside is that this iterative algorithm needs several iterations before converging towards a reasonable solution.

One MKL approach implements an adaptive 2–norm regularization formulation [34]. Weights on each kernel matrix are included in the standard SVM empirical risk minimization problem with a $l1$ constraint which encourages sparsity. An algorithm is proposed for solving this problem. It provides an new insight on MKL algorithms based on block 1–norm regularization. The resulting algorithm converges rapidly and its efficiency is good compared to other MKL algorithms. It is shown that the proposed algorithm [34] gives equivalent accuracy performance results while selecting more kernels than the available semi–infinite linear problem (SILP) MKL algorithm. Two MKL formulations are explored in this paper [31] which focus on aggregation using the Choquet fuzzy integral (FI) with respect to a fuzzy measure (FM). The p-norm GA MKL (GAMKLp) approach learns an MKL classifier using a genetic algorithm (GA) and it then generalizes p-norm weight domain. These algorithms perform a feature-level aggregation of the kernel matrices and produces a new feature representation.

A decision-level MKL called DeFIMKL is also proposed here [31]. It learns an fuzzy measure (FM) with respect to the Choquet FI to fuse decisions from individual kernel classifiers. Two additional decision-level methods are explored here [31] based on a least-squares formulation. In the decision level least-squares MKL (DeLSMKL) they computed

the weights for decision values from an ensemble of classifiers using a closed form expression. This method expanded using a nonlinear cost function and a GA to compute the weights in decision-level GA MKL (DeGAMKL). A downside of MKL methods is the necessity to store multiple kernel matrices. The size of kernels is directly related to the number of feature vectors in the dataset. Thus, approximating the kernel matrices that reduce the required number of values to store allows MKL methods to be used for the large datasets. A leading machine learning MKL method is called MKL group lasso (MKLGL) [45] which is applied on several benchmark datasets.

To understand this algorithm consider some nonlinear mapping function $\phi : x_i \rightarrow \phi(x_i) \in R_K^D$, where D_K is the dimensionality of the transformed feature vector $\phi(x_i)$ [31]. Following ref. [31], for brevity, $\phi(x_i)$ will be denoted as ϕ_i. With kernel algorithms, one does not need to explicitly transform x_i, one simply needs to represent the dot product $\phi_(x_i) \cdot \phi_(x_j) = k(x_i, x_j)$. The kernel function k can take many forms, with the polynomial $k(x_i, x_j) = (x_i^T x_j + 1)^p$ and *radial-basis-function* (RBF) $k(x_i, x_j) = exp(\sigma|x_i - x_j|^2)$ being two of the most well known. Given a set of n feature vectors X, one can thus construct an $n \times n$ kernel matrix $K = [K_{ij} = k(x_i, x_j)]_{n \times n}$. This kernel matrix K represents all pairwise dot products of the feature vectors in the transformed high-dimensional space H_K - called the *Reproducing Kernel Hilbert Space*. We will assume that the kernel K is composed by a weighted combination of precomputed kernel matrices, i.e.,

$$K = \sum_{k=1}^{m} \sigma_k K_k \tag{2.21}$$

where there are m kernels and σ_k is the weight applied to the k-th kernel.

2.1.4 Deep Learning

In recent years, deep learning (DL) has been widely used in research fields such as computer vision (CV), speech recognition, natural language processing, radar signal processing, etc [4]. The NN is an established section of artificial intelligence that has been revived due to algorithmic advancements, high performance computing, big data, etc. In other classifiers the algorithm needs hand-crafted or human-made features where in the DL method the algorithm learns the features itself and often produce equivalent or better results. Convolutional Neural Networks (CNNs) have a known grid-like topology [19]. Convolutional networks have been tremendously successful and widely implemented in practical applications. There are many DL approaches that have been used for various problems. The main four of them are CNN, autoencoder (AE), deep belief networks (DBNs) and recurrent NN (RNN) [4]. The CNN is the most popular, established and published DL method among them. The following subsections discuss each of these architectures at a high level.

2.1.4.1 Convolutional Neural Network (CNN)

A neural network is very loosely inspired by the functionality and structure of a brain and the visual system. This definition of a neural network is adapted from the book [1].
"A neural network is a massively parallel distributed processor made up of simple processing units that has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects:

1. Knowledge is acquired by the network from its environment through a learning process.
2. Interneuron connection strengths, known as synaptic weights, are used to store the acquired knowledge."

A CNN is constructed of different layers to get the classification output. A very basic structure of CNN classifier is to take the input image, pass it through a series of convolutional, nonlinear, pooling and fully connected layers and typically a softmax classifier layer at the end [9]. The output can be a single class, multiples classes, and it might also contain the probability of classes that best describes the image. Designing and tuning different parameters of those layers is the tricky part and it impacts the output. A classic CNN architecture would look as shown in Figure 2.2.

Figure 2.2 A classic CNN architecture.

The convolution layers learn patterns from the data. The pooling layer keeps only the maximum convolution responses. It then provides inputs to the following convolutional layers at a different scale which allows the CNN to learn a hierarchy of filters. A rectified linear unit (ReLU) zeros inputs below zero and thus allows only positive inputs through. It is a simple and efficient function to evaluate and is therefore very popular in DL architectures. The fully connected layers let the CNN learn complicated patterns from their inputs. The softmax layer is a form of probabilistic classifier.

2.1.4.2 AutoEncoder (AE)

An AutoEncoder (AE) is primarily used for unsupervised learning of efficient codings from unlabeled data [4]. It can also be used as a denoising mechanism. Its advantage is that the codings often reveal useful features from unsupervised data. In an AE, the size of the adjacent layers is reduced which forces the AE to learn a compact representation of the data. The AE maps the input through an encoder function f to generate an internal (latent) representation, or code, h. The AE also has a decoder function, g that maps h to the output x. There are many applications of auto encoders. In [3], a formal connection between features learned by regularized auto–encoders and sparse representation is establishes. They combined existing AEs and activation functions by bringing them under a unified framework but also uncovered more general forms of regularizations and fundamental properties that encourage sparsity in hidden representation. It also yields new insights into AEs and provides novel tools for analyzing regularization/activation functions that help predicting whether the resulting AE learns sparse representations.

2.1.4.3 Deep Belief Network (DBN)

A deep belief network (DBN) is a probabilistic graphical model (PGM) which can also be called a deep neural network. It is a deep or large directed acyclic graph (DAG) [4] which combines the probability and graph theory. In this network the data can flow both ways and training can be either bottom-up or top-down.

2.1.4.4 Recurrent Neural Network (RNN)

Connections between units create a directed cycle in recurrent neural network. It is normally used on speech and time-series analysis. The RNN has persistence due to its lack of memory which makes it different than the AE and CNN. An important milestone for RNNs is the utilization of long short–term memory (LSTM) unit [37] which allows information to be written in a cell, output from the cell and also store in the cell. It allows the flow of information which ultimately counteracts the vanishing/exploding gradient problems in very deep networks.

CHAPTER III

LIDAR OBJECT CLASSIFICATION USING SUPPORT VECTOR MACHINE (SVM)

3.1 Abstract

The target of this research is to develop a machine-learning classification system for object detection based on three-dimensional (3D) Light Detection and Ranging (LiDAR). The proposed real-time system operates a LiDAR on an industrial vehicle and provides a 3D point cloud for spatial sensing. I have developed 3D features which allow a linear Support Vector Machine (SVM) as well as Multiple Kernel Learning (MKL) to determine if objects in the LiDARs field of view are beacons (a passive, highly–reflective object used to mark boundaries in an industrial application) or other objects (e.g. people, buildings, equipment, etc.). Results from multiple data collections are analyzed and presented. Moreover, the feature effectiveness and the pros and cons of the approach are examined.

3.2 Introduction

There has been a significant increase in intelligent vehicle research in the automobile industry over the last century. Intelligent vehicle research has expanded to the point where human factors are merging with intelligent–vehicle technology to create a new generation of driver assistance systems that go beyond automated control systems by attempting to work in harmony with a human operator [29]. ADAS (Advanced Driver Assistance Sys-

tems) is the set of systems and subsystems on the way to a fully automated highway system [5]. ADAS is very similar to the automatic pilot in airplanes. ADAS covers many areas, including blind spot detectors, Adaptive Cruise Control, Autonomous Intelligent Cruise Control, platoon driving, etc., to name a few. Many of these technologies are either invented and available on the market or ready to be marketed. Some systems are developed but still under test as a prototype. With the increased popularity of ADAS [16], many researchers are diverting their interest toward object detection and classification using sensors, e.g., cameras, radars, LiDARs, etc. There are some existing methods of driver assistance systems based on computer vision [20]. Exclusively vision–based systems are not capable enough to perform all driving assistance relevant tasks, especially in bad weather. Most researchers are combining different types of sensors like ultrasonic, radar and LiDAR to extend the range of sensor information available for building a reliable system.

In an industrial environment it is often required to keep certain areas off limits to vehicles for protection of people and valuable assets. Herein, a system is developed to use an eight–beam Quanergy M8 LiDAR on an industrial vehicle to quarantine certain areas via beacons that delineate a noentry area. The beacons are made of standard orange traffic cones with a highly reflective vertical pole attached at top. This unique structure of the beacon makes it easier to classify and detect it among other objects present in the environment. The LiDAR performs well regardless of lightening condition, temperature or bad weather such as fog, rain, etc. It can readily detect these beacons but might suffer from false positives due to other reflective surfaces such as worker safety vests or vehicles. All the classification algorithms have been designed and implemented considering the above

mentioned restrictions. The detection system is applied in real-time scenario on a LiDAR attached to an industrial vehicle. Next in this chapter different established algorithms that use linear and non-linear (kernel) classifiers are discussed. The proposed method is described in detail to explain the construction of the algorithms. The environment for object detection for this case is unique comparing other regular outdoor environment. The data that has been used for training and testing the system were collected by some graduate students of Electrical and Computer Engineering department at Mississippi State University. In the final section, all the plots and outputs from training and testing the algorithm are discussed.

3.3 Related work

LiDAR detection offers some advantages as well as some difficulties in classification. The main obstacle is the sparse 3D point cloud which makes it tough to classify and detect objects reliably. There are several research articles where the LiDAR 3D point cloud data is processed for classification. One approach [26] introduced an improved eigen–feature analysis of weighted covariance matrices with a Support Vector Machine (SVM) classifier. They used airborne LiDAR point cloud data for classification in urban areas. The target is to generate reliable eigen–features from a point cloud to improve classification accuracy. Each point in the local point set is assigned a weight to represent the spatial contribution of the point. Point density has computational efficiency and this property has been leveraged here by introducing point density in the weighting function. It ultimately approximates the area of surface occupied by a point. Moreover, the geometric median is used instead of

the sample mean in calculating the covariance matrix. The geometric median and point weights have been obtained by an iterative solver. The experimental results are based on the simulated point cloud. The eigen values obtained from their proposed weighted covariance matrix improves classification accuracy compared to the eigen values calculated from the standard covariance matrix. In another approach, the feature vectors are classified for each candidate object with respect to a training set of manually labeled object locations [17]. The authors of this paper experimented with several classifiers including a k-nearest neighbors (NN) classifier with $k = 1$ and $k = 5$, random forests, and SVM with penalty $C = 2.5$ and 5th order polynomial kernels. One technique [43] uses an environmental change detection pipeline to perform in real-time on distorted 3D point clouds with slow acquisition rate in cluttered environments. Points are classified with a learning-based algorithm which achieves robustness to noisy point clouds and under-sampled reference maps. This method achieves high classification performance despite the low amount of labeled training data. Most of the available methods on LiDAR data classification are implemented on a high–density (high beam count) LiDAR. We are using eight–beam Quanergy M8 LiDAR which collects comparatively scarce point cloud data. So, these methods could not be successfully implemented in our system with an eight–beam LiDAR.

3.4 Proposed Method

Quanergy's M8 LiDAR is a compact and rugged sensor which is designed to meet the demands of the most challenging real–world applications. It has multiple laser beams and Time–of–Flight (TOF) range measurement to provide 3D point clouds for spatial sensing.

The sensor functions same under any lighting, temperature and weather condition including rain, snow and dust. The basic specifications of the Quanergy M8 LiDAR are given in the following table.

Table 3.1 Quanergy M8 LiDAR key specifications.

Item	**Specification**
Frame Rate (Update Frequency)	5-20 Hz
Range	1 to 150 m
Azimuth Angular Resolution	0.03 - 0.2 dependent on frame rate
Field of View (FOV)	Azimuth: 360° Vertical: 20° ($+3^{\circ}$ / -17°)
Data Outputs	Angle, Distance, Intensity, Beam Number, Synchronized Time Stamps)
Environmental Protection	IP69K - rating for ingress protection against dust and water)
Operating Wavelength	905 nm
Laser Class	Class I (Eye Safety IEC 60825-1)
Nominal Power Consumption	18W @ 24VDC
Startup Sequence	45 seconds
Operating Temperature	-20C to +60C (-4F to +140F)
Certifications and Compliance	FDA, FCC, CE, RoHS, WEEE, IEC -60079-15, ASTM G154)

3.4.1 Ground Point Removal

The major issue with the ground points is that it creates false positives from the reflections of highly reflective paint or other reflective objects on the ground. In addition, it can interfere with feature extraction. There are many examples of ground point removal from areas with varying ground conditions. Wang et al. [41] addresses a novel ground filtering method called SLSGF which uses scan line information in LiDAR data for segmentation. The bright side is that its parameters are insensitive and robust to noise. Separating

point clouds into ground and non-ground points is a necessary step to generate digital terrain model (DTM) from LiDAR dataset. Another paper [35] proposes a new method for ground filtering of LiDAR data based on multi–scale analysis of height difference threshold. It utilizes three windows with different sizes in small, average and large to cover the entire LiDAR point clouds. A height difference threshold then separates point clouds to ground and non–ground in each local window. The best threshold values for size of windows are determined based on physical characteristics of the ground surface and size of objects. Another approach [32] relies on Demspter–Shafer Theory to model the occupancy induced by the LiDAR data in a point cloud and then detect whether the point is dynamic or static. Since this industrial application has a smooth, flat area for the ground, we employ a simple vertical threshold to remove ground points. For this particular application complex ground removal algorithm was not necessary, since it is an industrial application and the ground is very flat. In this case, a simple vertical threshold was utilized to remove ground points. Other algorithms, similar to those mentioned above, could be used if the environment was more complex.

3.4.2 Clustering

In this approach, after ground point removal, the bright (e.g. high-intensity) points are clustered. The idea is that most surfaces return very low–intensity points, so the beacon, which has a tall and thin, highly–reflective pole, returns many bright points. In this manner, the clutter is significantly reduced except for certain other objects such as people wearing safety vests with retro-reflective markers.

Algorithm 1: LiDAR bright pixel clustering.

Input: LiDAR point cloud $P = \{x_j, y_j, z_j, i_j, r_j\}$ with NP points.

Input: High–intensity threshold: T_H.

Input: Cluster distance threshold: ϵ (meters).

Input: Ground Z threshold: T_G (meters).

Remove all non–return points (NaN's).

Remove all points with intensity $< T_H$.

Remove all ground points (Z–values below T_G).

Cluster bright points:

for *each point P_j in the modified point cloud* **do**

  if *The point does not belong to a cluster* **then**

    Add point to cluster

    Increment the number of clusters: $cl \leftarrow cl + 1$.

    Assign the point to cluster cl.

    Set the centroid of cluster cl to the point.

    Scan through all remanining points and recluster if necessary.

    if *Distance from point to centroid of cluster $cl < \epsilon$* **then**

      Add P_m to cluster cl and recalculate centroid. Recalculate centroid of cluster cl.

    end

  end

end

Partitioning and hierarchical algorithms are two basic types of clustering [22]. From the paper [11] , partitioning algorithm constructs a partition of a database D of n objects into a set of k clusters. CLARANS (Clustering Large Applications based on RANdomized Search) algorithm is introduced in paper [30] which is an improved k-medoid method. CLARANS is more effective and more efficient than previous k-medoid algorithms. There are some more clustering algorithms which are developed over time to perform accurately and efficiently. In the paper [11], a clustering algorithm called DBSCAN is introduced. It relies on a density–based notion of clusters. It requires only one input parameter and with that it supports the user in determining an appropriate value for it. For our data processing we have used a modified DBSCAN clustering algorithm [11] which clusters based on point cloud density as well as intensity to cluster the bright points, as shown in algorithm 1. The cluster parameter ϵ was experimentally determined to be 0.5m based on the structure of the beacon. It was chosen as 0.5m because values larger than that could group two nearby beacons (or a beacon and another nearby reflective object) together into one cluster which is not desired.

In Equation 1, the distances are estimated using Euclidean distances with only the x (front-to-back) and y (left–to–right) coordinates. This algorithm clusters bright LiDAR points by projecting them down onto the $x - y$ plane. This approach is more computationally efficient than using all three coordinates in the clustering algorithm.

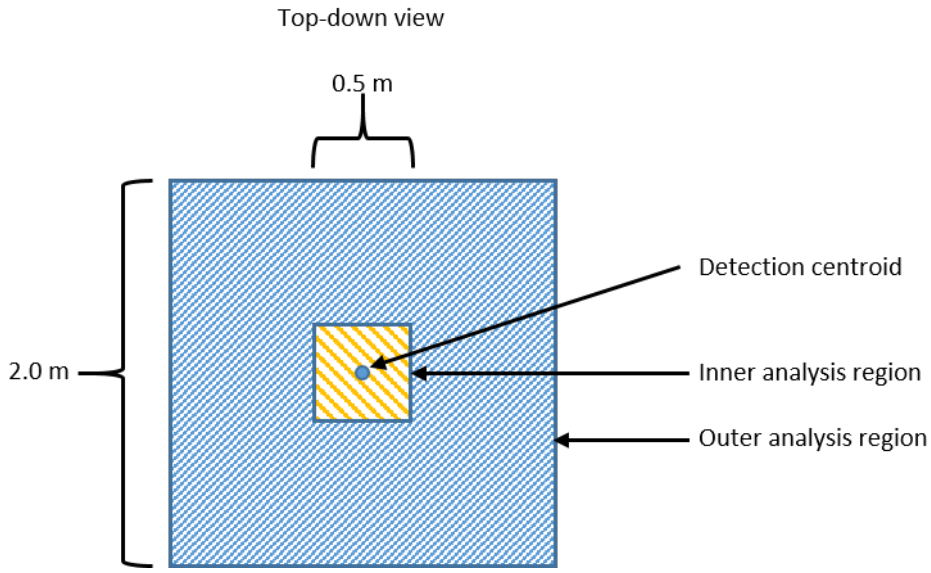
3.4.3 Feature Extraction

Ground points have been removed before extracting the features. The point intensities are then compared to an empirically determined threshold. The beacon is designed so that it provides brighter returns to the LiDAR comparing other traffic cones and random objects. As noted above, there are also objects in the scene that can have high returns, such as other industrial vehicles with retro–reflective markings, shiny surfaces of the vehicles or workers wearing safety vests with retro–reflective stripes. In order to classify objects as beacons and non-beacons, hand–crafted features are utilized. Ground removal leaves us with bright points coming from objects rather than the ground. The points closer in the bright point cloud are clustered. After clustering the beacons appear as tall, thin objects, whereas all other objects are either not as tall, or wider. People with reflective vest have closer structure to a beacon, but a beacon is much taller and does not have the horizontal extent that people do. Features are extracted around the cluster center in a small rectangular volume which is centered at each object's centroid. Another feature extraction is done using a larger rectangular volume which also centers around the object's centroid. The idea is that the beacon mainly has points in the inner rectangular area, while other objects will have points extending in the areas outside of the inner area.

These features include the number of bright points in each region, determining the x, y, and z extents (e.g. max value minus min value) of the points in each region, etc. Beacons mainly have larger values in the smaller region, while other objects also have values in the larger regions. Figure 3.1 shows the detection regions from a top–down view. The centroid

of the bright regions is shown as the dot in the figure. The dimensions of the region were determined by experimental analysis of returns from beacons and non–beacons.

Figure 3.1 LiDAR detection regions (inner and outer) visualized from a top–down view.



Algorithm 2: LiDAR high–level feature extraction preprocessing.

Input: LiDAR point cloud $P = \{x_j, y_j, z_j, i_j, r_j\}$.

Input: Low and high–intensity thresholds: T_L and T_H.

Input: Ground Z threshold: T_G (meters).

Output: Feature vector **f**.

Remove all non–return points (NaN's).

Remove ground points: Remove points with $Z < T_G$.

Create threshold point clouds:

Set $P_{HT} = \varnothing$ and $P_{LT} = \varnothing$.

for *each point* P_j *in the modified point cloud* **do**

    if *Point* P_j *has intensity* $\geq T_H$ **then**

        Add P_j to P_{HT}.

    if *Point* P_j *has intensity* $\geq T_L$ **then**

        Add P_j to P_{LT}.

Extract features **f** using Algorithm 3.

Algorithm 3: LiDAR feature extraction.

Input: LiDAR high–intensity point cloud $P_{HT} = \{x_j, y_j, z_j, i_j, r_j\}$.

Input: LiDAR low–intensity point cloud $P_{LT} = \{x_j, y_j, z_j, i_j, r_j\}$.

Input: Inner region x and y extents: Δx_I and Δy_I (meters).

Input: Outer region x and y extents: Δx_O and Δy_O (meters).

Input: LiDAR height above ground: $Z_L = 1.4$ (meters).

Output: Feature vector **f**.

Cluster the high–intensity point cloud:

Calculate features:

for *each cluster center point* $c = (x_C, y_C, z_C)$ *in the point cloud* **do**

- Determine points in P_{HT} in inner region using Eq. 3.1 and calc. feature 1,13 and 17 from Table 3.2.
- Determine points in P_{HT} in the outer region using Eq. 3.2 and calc. feature 4 from Table 3.2.
- Determine points in P_{LT} in the inner region using Eq. 3.1 and calc. features 6,7,9,10,11,14,16 and 18 from Table 3.2.
- Determine points in P_{LT} in the outer region using Eq. 3.2 and calc. features 2,3,5,8,12,15,19 and 20 from Table 3.2.

Return $\mathbf{f} = [f_1, f_2, f_3, \cdots, f_{20}]$.

The two regions are shown with bounding boxes in Figure 3.1. The idea of using an inner and an outer analysis region is that a beacon will mostly have bright points located in the inner analysis region, while other objects, such as humans, other industrial vehicles, etc., will extend into the outer regions. Equations 3.1 and 3.2 define whether a LiDAR point p_j with coordinates (x_j, y_j, z_j) is in the inner region or outer region, respectively, where the object's centroid has coordinates (x_C, y_C, z_C). Reference Figure 3.1 for a top–down illustration of the inner and outer regions.

Figure3.1 shows an example beacon return with the analysis windows superimposed. Both the inner and outer analysis regions have x and y coordinates centered at the centroid location. The inner analysis region has depth (x coordinate) of 0.5 meters, width (y coordinate) of 0.5 meters, and the height includes all points with z coordinate values of -1.18 meters and above. The outer region extends 2.0 meters in both x and y directions and has the same height restriction as the inner region. These values were determined based on the dimensions of the beacon and based on the LiDAR height. The parameters Δx_I, Δy_I and z_{MIN} define the inner region relative to the centroid coordinates. Similarly, the parameters Δx_O, Δy_O and z_{MIN} define the outer region relative to the centroid coordinates.

A point is in the inner region if

$$\left(x_C - \frac{\Delta x_I}{2}\right) \leq x_j \leq \left(x_C + \frac{\Delta x_I}{2}\right) and$$
$$\left(y_C - \frac{\Delta y_I}{2}\right) \leq y_j \leq \left(y_C + \frac{\Delta y_I}{2}\right) and \tag{3.1}$$
$$z_{MIN} \geq z_j,$$

and a point is in the outer region if

$$\left(x_C - \frac{\Delta x_O}{2}\right) \leq x_j \leq \left(x_C + \frac{\Delta x_O}{2}\right) and$$
$$\left(y_C - \frac{\Delta y_O}{2}\right) \leq y_j \leq \left(y_C + \frac{\Delta y_O}{2}\right) and \tag{3.2}$$
$$z_{MIN} \geq z_j.$$

Table 3.2 describes the extracted features. Herein, extent means the maximum value minus the minimum value, e.g. Z extent is $max\{Z\}$ - $min\{Z\}$. It is noted that many features were examined and they each had different abilities to discriminate the beacons from non–beacons. For the LiDAR, beam 7 is the upper–most beam, beam 6 would be horizontal (assuming the LiDAR is level with the ground), and beam 0 the closest beam to the LiDAR. The beam spacing is approximately 3 degrees per beam.

Table 3.2: Feature Descriptions. HT/LT = High/Low threshold data.

Feat. #	**Data Subset**	**Analysis Region**	**Description**
1	HT	Inner	Extent of Z in cluster.
2	LT	Outer	Max of X and Y extents in cluster, beam 7.
3	LT	Outer	Max of X and Y extents in cluster, beam 5.
4	HT	Outer	Max$\{Z$ in cluster - LiDAR height$\}$.
5	LT	Outer	Extent of Z in cluster.
6	LT	Inner	Number of valid points in cluster, beam 7.
7	LT	Inner	Max of X and Y extents in cluster, beam 6.
8	LT	Outer	Number of points in cluster, beam 5.
9	LT	Inner	Extent of X in cluster.
10	LT	Inner	Number of points in cluster, beam 4.
11	LT	Inner	Number of points in cluster, beam 5.
12	LT	Outer	Number of points in cluster, beam 6.
13	HT	Inner	Number of points in cluster, beam 6.
14	LT	Inner	Max of X and Y extents, beam 5.
15	LT	Outer	Pts. in cluster divided by cluster radius, beam 5.
16	LT	Inner	Extent of X in cluster.

Continued on next page

Table 3.2 – *Continued from previous page*

Feat. #	**Data Subset**	**Analysis Region**	**Description**
17	HT	Inner	Number of points in cluster, beam 7.
18	LT	Inner	Number of points in cluster.
19	LT	Outer	Number of points in cluster.
20	LT	Outer	Extent of Z in cluster.

To determine the best features, a simple (but sub–optimal) feature selection process was performed. Each feature was evaluated on the training set for its ability to distinguish beacons from non–beacons using a measure of classifier performance, which is a score from 0 to 1,000, where higher numbers indicate better performance, as shown in eq. 3.3.

$$score = 500\frac{TP}{TP + FN} + 500\frac{TN}{TN + FP} \tag{3.3}$$

where TP, FP, TN and FN refer to the number of true positives, false positives, true negatives and false negatives, respectively [25]. The score increases as TP and TN increase, and decreases with bad decisions, which cause FP or FN to increase.

After the features are scored, they are sorted by ascending order of score, and concatenated one at a time until the level of performance desired is achieved.

3.4.4 Support Vector Machine (SVM)

The goal always has been to train a linear SVM which can distinguish beacons from other objects (e.g., humans, cars, buildings etc.). To do this the extracted features from the LiDAR detections are stored in a vector f. The SVM then computes discriminant function, which is the dot product of the weight vector and the training features. It then adds a bias term to allow the hyperplane to move and not have to be anchored to the origin. The SVM weights are learned by the SVM training, which uses labeled samples and an optimization routine to find the weight set that maximizes the SVM margin. If the discriminant value, d, is less than zero then SVM results indicate that the LiDAR sensor detected a beacon. Else the SVM results indicate that the LiDAR did not detect a beacon.

The feature vector is given by

$$f = [f_1, f_2, ..., f_M]^T \tag{3.4}$$

where f_k is the k-thfeature, T denotes a matrix transpose, and M is the number of features ($M = 20$). The discriminant is calculated as

$$d = \omega^T f + b \tag{3.5}$$

where

$$\omega = [\omega_1, \omega_2, ..., \omega_M]^T \tag{3.6}$$

is the vector of optimal SVM weights and b is the bias term. The weights ω_x and bias term b are derived from the training data. The SVM discriminant function is used to differentiate beacons from non-beacons. The Matlab toolbox liblinear [12] is used for this processing. The liblinear toolbox provides the linear SVM optimization code that computes the weight and bias coefficients from the training data.

3.4.5 Multiple Kernel Learning (MKL)

There are various algorithms that use kernels to transform the input data to an appropriate and useful space. From the paper [31] it can be assumed that the kernel K is composed by a weighted combination of pre computed kernel matrices, i.e.,

$$K = \sum_{k=1}^{m} \sigma_k K_k \tag{3.7}$$

where there are m kernels and σ_k is the weight applied to the k-th kernel.

In [45] the authors have presented an efficient algorithm for multiple kernel learning by showing the combination of MKL and group-lasso regularizer. They calculated the kernel weights by a closed-form formulation, which therefore leverages the dependency of previous algorithms on employing complicated or commercial optimization software. Moreover, in order to improve the accuracy of traditional MKL methods, they generalized MKL to L_p-MKL that constrains the p-norm ($p \geq 1$) to the kernel weights. This algorithm can be applied to the whole family of L_p-MKL models for all $p \geq 1$.

The second part of the experiments were designed to determine if using a non-linear SVM (kernel SVM) would produce better results and is there possibility of producing them with a smaller number of features. Herein, MKL was utilized as the non-linear supervised classifier. It means that we are using the same features and data processing techniques with the MKL to classify beacon and non-beacon. MKL has various forms, and in this work, single-kernel MKL. In single-kernel MKL, Radial Basis Function (RBF) kernels were utilized and the system optimized the kernel parameter, the RBF standard deviation. In MKLGL, and MKLGL (MKL with Group Lasso) were utilized, where the weights and classification surface are simultaneously solved by iteratively optimizing a min-max optimization until convergence. The MKL performs better than the linear SVM because the features are not quite linearly separable. I used the fuzzy library [2] to get the basic MKL implementation example which we modified and applied to this problem.

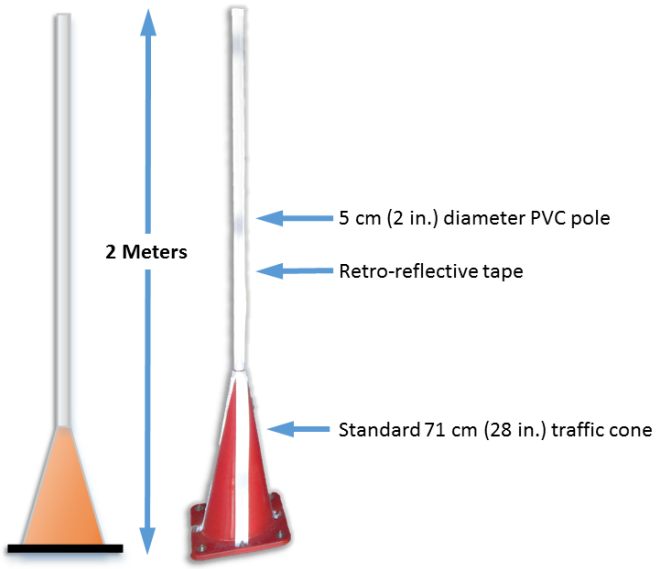
3.5 Experiments

The dataset utilized herein was collected using a Quanergy M8 LiDAR over the span of seven months. Training and tuning data was collected at Raspet Research Building II site and the Mississippi State University Center for Advanced Vehicular Networks (CAVS) parking lot in Starkville, MS. All the data collection included collecting LiDAR detections for humans, beacons, buildings, and parking lot objects such as trailers, vehicles, and gas tanks.

3.5.1 Beacon

The beacon that has been used for our experiment and data collection is constructed of a regular orange traffic cone where a long pole is attached at its top. The traffic cone is orange coloured 28" high with a vertical reflective tape affixed. It has a 2" diameter highly–reflective pole extending the beacon to two meters vertically in total. The beacon is shown in Figure 3.2. The beacon is presented as a series of high intensity values in a LiDAR scan. It provides a high signal level compared to most background objects. It delineates an area in the industrial complex that is off limits to the vehicle to protect people and high valued assets.

Figure 3.2 Beacon.



The LiDAR data were collected to train and test our algorithms. These data were collected over the span of around seven months from July 2017 to January 2018. Most of

them were collected in a scenario where one or multiple beacons were present. The beacon data was collected for both known and random locations. Other than only beacon data, there were some data collection with only people where they were wearing the reflective vest for some cases and were not wearing them in other cases. The target of the data collection was to collect enough data for both training and testing so that the algorithms can be implemented and tested.

3.6 Results and Discussion

The confusion matrix for the SVM, single MKL and MKLGL algorithms are given in tables below to compare the accuracy of these techniques. All of them perform well and their accuracy values are close to each other. It clearly indicates that the algorithms perform well in classifying the beacon. For the SVM algorithm, the training and testing confusion matrices are shown in tables 3.3 and 3.4, respectively, where the reference data are in columns, and the classified data are in rows.

Table 3.3 Training Confusion Matrix for Linear SVM.

	Class 1 (Beacon)	**Class 2 (Non-beacon)**
Class 1 (Beacon)	13,158	32
Class 2 (Non-Beacon)	610	14,599

For beacons, the user accuracy is 99.76% and the producer accuracy is 95.57%. For non–beacons, the user accuracy is 95.99% and the producer accuracy = 99.78%. The overall accuracy is 97.74 %.

Table 3.4 Testing Confusion Matrix for Linear SVM.

	Class 1 (Beacon)	**Class 2 (Non-beacon)**
Class 1 (Beacon)	5,653	13
Class 2 (Non-Beacon)	302	11,782

For beacons, the user accuracy is 99.77% and the producer accuracy is 94.93%. For non–beacons, the user accuracy is 97.50% and the producer accuracy is 99.89%. This shows that the SVM didnt significantly overtrain. The overall accuracy is 98.23%. For the SVM algorithm, the training and testing confusion matrices are shown below, where reference data are columns, and classified data are in rows. For the single MKL algorithm, the training confusion matrix is shown below in Table 3.5, where the reference data are in columns, and the classified data are in rows. The overall accuracy is 98.34% for the single MKL method.

Table 3.5 Training Confusion Matrix MKL Single-kernel SVM.

	Class 1 (Beacon)	**Class 2 (Non-beacon)**
Class 1 (Beacon)	3,721	19
Class 2 (Non-Beacon)	134	5,355

For the MKLGL algorithm, the training confusion matrix is shown in Table 3.6, where the reference data are in columns, and the classified data are in rows.

Table 3.6 Training Confusion Matrix for MKLGL.

	Class 1 (Beacon)	**Class 2 (Non-beacon)**
Class 1 (Beacon)	3,721	19
Class 2 (Non-Beacon)	8	5,481

The MKLGL overall accuracy is 99.71%. Here, Figure 3.3 shows that for both training and testing, the overall accuracy (OA) increases after a certain number of features and then remains approximately the same. The operating point of the system using reduced features is shown by the circle. This operating points was selected as a trade off in the complexity in calculating features and in obtaining good performance in real-time. For both the cases, it reaches an acceptable level of overall accuracy at around twenty features.

In Figure 3.4, the score value reaches optimum value for both training and testing around twenty features. At first, the OA was utilized to optimize which features were selected. This was problematic due to class imbalances and the SVM ended up favoring one class almost completely. The class score is given by

$$SCORE = 500\frac{TP}{TP+FN} + 500\frac{TN}{TN+FP} \tag{3.8}$$

where TP, TN, FP and FN are the number of true positives, true negatives, false positive and false negatives, respectively [25]. The score ranges from 0 to 1,000, where higher numbers are better. It also penalizes for any false positives or false negatives.

Figures 3.5 and 3.6 show the plots for Probability of detection (Pd) and Probability of false alarm (Pfa) for both training and testing cases. Comparing these two plots with

the plots from figures 3.5 and 3.6, we can say that all these plots are showing that for our particular problem the SVM performs very well with around twenty features. Figure 3.7 shows the probability density functions (PDFs) for the SVM discriminant function for beacons and non-beacons. The data has good results, but is not linearly separable.

From figures 3.8 and 3.9, we can see the score plots with respect to the number of features and the overall accuracies (OA) for the different algorithms, respectively. Here in both the cases, the single Kernel and MKLGL performs much better than the linear SVM. Even we can achieve the same efficiency with the MKLGL by using only seven features where in the linear SVM we are using twenty features. It reduces the computational time in MKL.

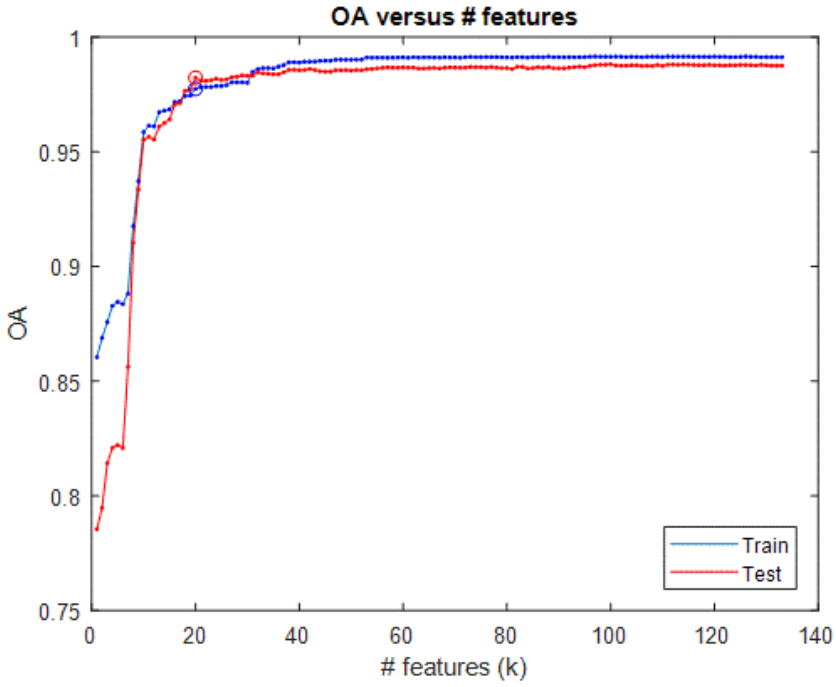


Figure 3.3 Overall accuracy (OA) with respect to number of features.

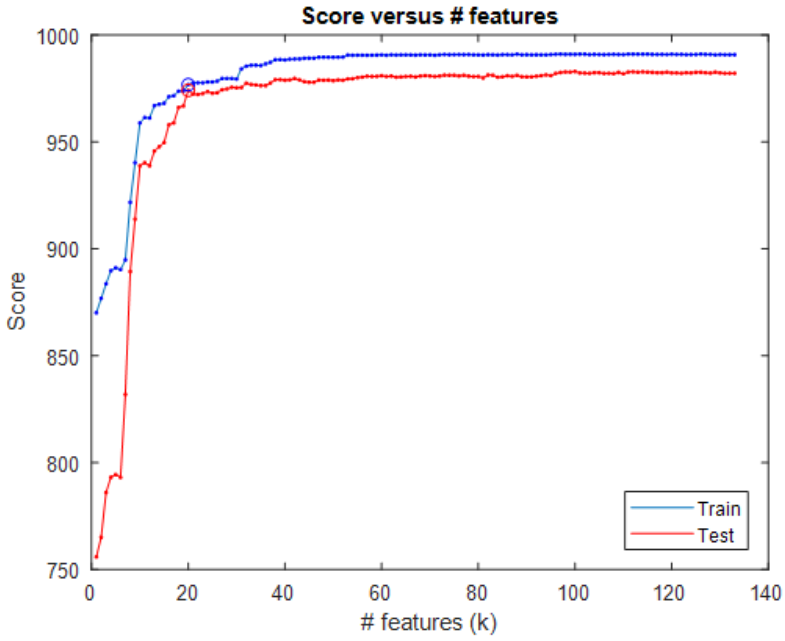


Figure 3.4 Score with respect to number of features.

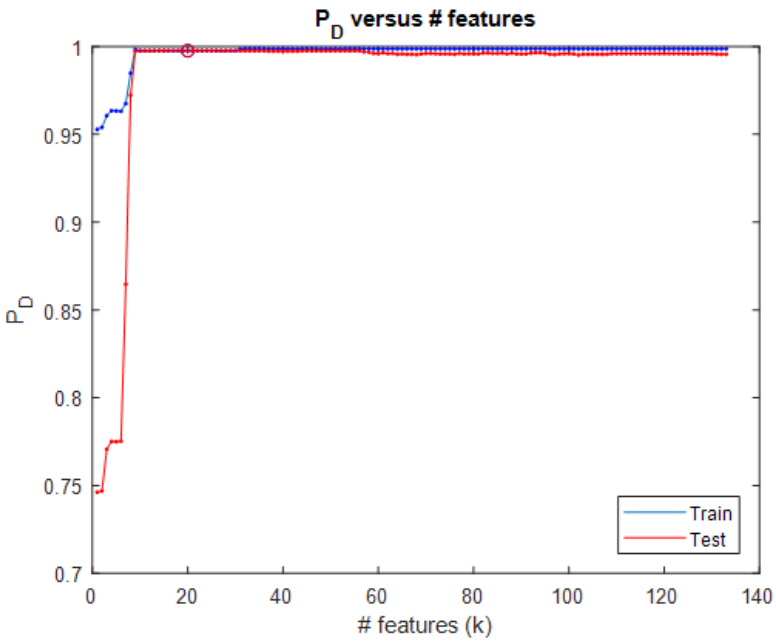


Figure 3.5 Probability of Detection (P_d) with respect to number of features.

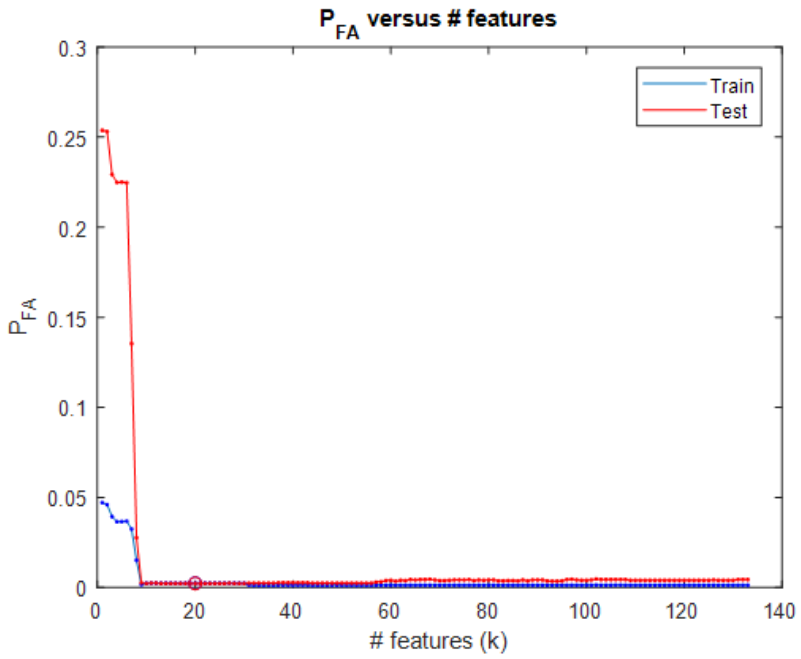


Figure 3.6 Probability of False Alarm (P_{fa}) with respect to number of features.

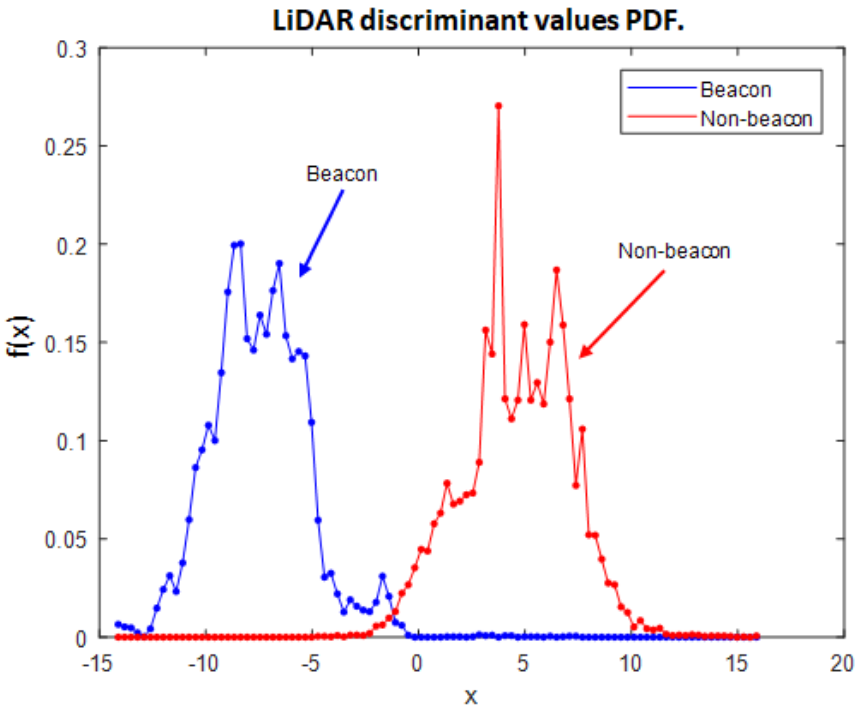


Figure 3.7 PDF values for beacon and non-beacon for optimal SVM discriminant function with ten features.

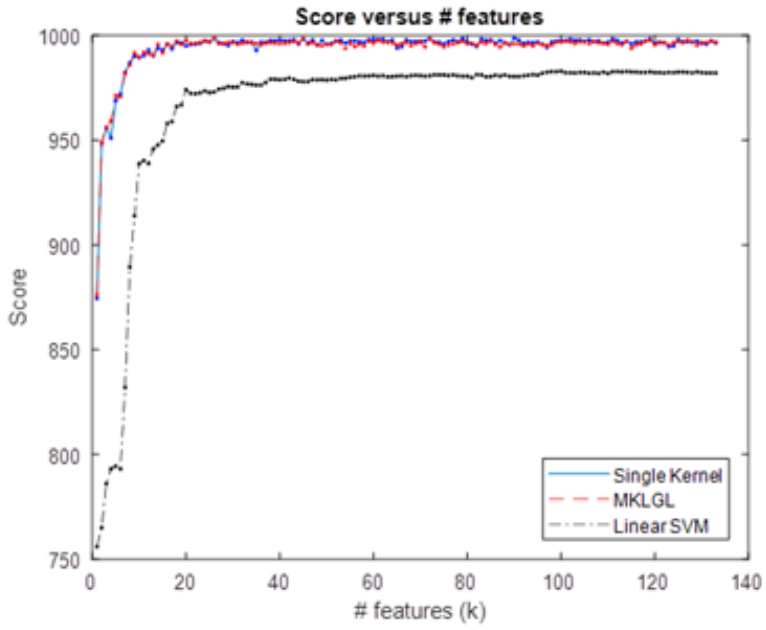


Figure 3.8 Score with respect to number of features for linear SVM, single Kernel and MKLGL.

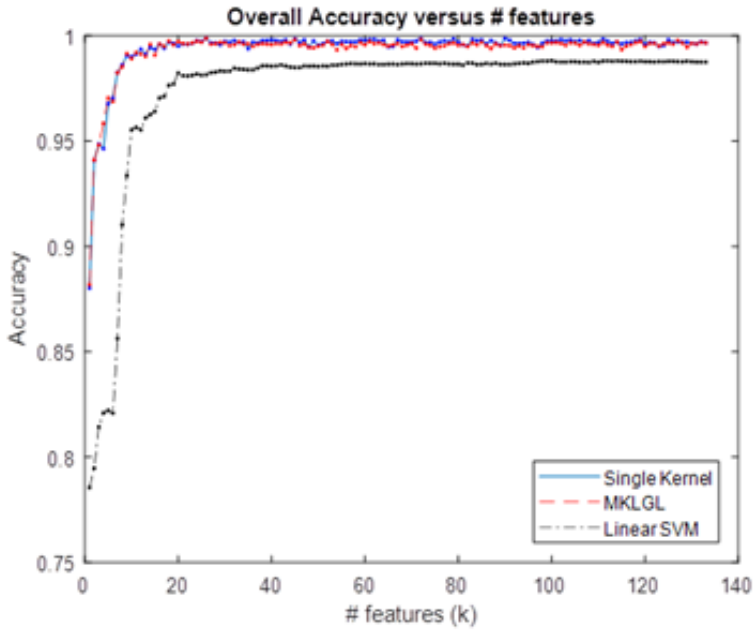


Figure 3.9 Overall Accuracy with respect to number of features for linear SVM, single Kernel and MKLGL

CHAPTER IV

LIDAR OBJECT CLASSIFICATION USING DEEP LEARNING

4.1 Abstract

Deep learning is the up and coming classification algorithm that is outperforming other traditional classifiers. The linear SVM as well as non linear single kernel and MKL performs well on three-dimensional (3D) Light Detection and Ranging (LiDAR) data classification. These methods requires hand crafted features to train the algorithms. In this research work a 2D CNN is implemented to see if it can perform object detection. The dataset used for this method is similar to the SVM and MKL dataset which gave an excellent opportunity to compare the SVM, MKL and 2D CNN performance. Moreover the advantages and disadvantages of CNN has been discussed briefly.

4.2 Introduction

In this chapter a deep learning algorithm is applied for the LiDAR point cloud data. The specifications and advantages of the LiDAR data has been described in the previous chapter. For this chapter the dataset are same as the SVM and MKL methods so that later the performance of all these classifiers can be compared. Next in this chapter different established algorithms that use deep learning based classifiers are discussed. The proposed method is described in detail to explain the construction of the algorithms. The environ-

ment for object detection for this case is exactly same as the previous chapter. In the final section, all the plots and outputs from training and testing the algorithm are discussed.

4.3 Related work

Deep learning has recently been studied intensely due to significant performance gains and the ability to learn hierarchical features from the data, without the need for hand-crafted features [4]. Deep learning discovers the complex structure in large data sets by using a neural network learning method such as backpropagation, stochastic gradient descent, etc., to indicate how a machine should change its internal parameters that are used to compute the representation in each layer from the representation in the previous layer [23].The four mainstream deep learning architectures applied by researchers are the auto encoder (AE), convolutional neural network (CNN), deep belief networks(DBNs)and recurrent NN (RNN).The CNN is the most popular and most published branch of deep learning. A 3D deep learning can be applied for efficient and robust landmark detection in volumetric data [46]. Separable filter decomposition and network sparsification are two technologies which have been proposed to speed up the detection using neural networks. Deeply learned image features trained on amulti-resolution image pyramid are used to improve detection robustness. Finally the boosting technology incorporates deeply learned hierarchical features as well as Haar wavelet features to further improve the detection accuracy.

There has been significant work on shape descriptors for voxel representations and point cloud representation recently. Other shape descriptors include the light field descriptor, heat kernel signatures, and spherical harmonic representation [21]. Most deep learning

based 3D object classification problem involves two steps namely, deciding a data representation to be used for the 3D object and training a CNN on that representation of the object. VoxNet is a 3D CNN architecture for efficient and accurate object detection from LiDAR and RGBD (Red, Green and Blue plus Distance) point clouds. The effect of various design choices on its performance is studied [28]. Most up to date methods either use voxel representation or a set of multiple 2D projections of the polygon mesh from several camera positions. A good example of deep learning for volumetric shapes is the Princeton ModelNet dataset which has proposed a volumetric representation of the 3D model and a 3D Volumetric CNN for classification [44]. A typical 3D representation of a map can be presented by a voxel space in Cartesian coordinates with resolution specified according to context and to memory requirements [13]. In the paper an illustration of the voxel space in spherical coordinates is introduced and then the resolution is derived by the properties of the LiDAR. Deep Sliding Shapes is a 3D ConvNet formulation that takes a 3D volumetric scene from a RGB-D image as input and outputs 3D object bounding boxes [38]. The first 3D Region Proposal Network (RPN) learns objectness from geometric shapes and then first joint Object Recognition Network (ORN) extracts geometric features in 3D and color features in 2D. A unique way of demonstration is recognizing 3D shapes from a collection of their rendered views on 2D images is utilized in [40]. In this technique, a novel CNN architecture combines information from the multiple views of a 3D shape into a single and compact shape descriptor offering better recognition performance. High quality diverse set of 3D object proposals can be generated in the context of autonomous driving [7]. These proposals are 3D and represented as cuboids due to the importance of 3D reasoning. A

stereo image pair is considered as input and the depth is computed by the state-of-the-art depth system. Further, this depth is used to conduct all reasoning. All these methods have been implemented and have shown promising results. Despite their better performances, they can not be implemented for our particular problem. The primary reason is that in most of the cases they are using very dense point cloud datasets which often have dense points similar to pixels in images. This property of their data makes it easier to implement complex 3D CNNs and get good results. Also they often have various number of classes to be defined and for that a complex CNN architecture is a good option. For our sparse point cloud dataset collected from an eight-beam LiDAR, a basic 2D CNN architecture seems feasible consider we only have two classes (beacon and non–beacon) to identify.

4.4 Proposed Method

This sections discusses the proposed method, including data processing, ground point removal, and the 2D CNN architecture.

4.4.1 LiDAR Data description

The Quanergy M8 LiDAR collects three dimensional point cloud data with five different information. The point cloud data consists of x, y, z coordinates; intensity and lastly the beam number or ring number. Intensity is a value ranging from 0 to 255, but it typically varies from 0 to about 20–30. The lower intensity values are collected from objects that has low reflection. Similarly, the higher intensity values are from highly reflective objects and in our case these are from the beacon, people wearing the reflective vest and vehicles with shiny surface. The LiDAR is mounted level to the ground on the industrial vehicle. The

fifth data is the beam number or ring number. It indicates which beam has hit the object. The figure 4.1 shows the eight beam projection of the LiDAR on the beacon.

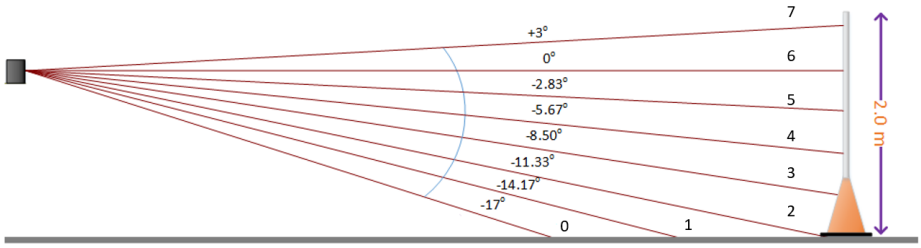


Figure 4.1 Quanergy M8 LiDAR beam spacing visualization and beam numbering. The LiDAR is the small box on the left and the beacon is placed on the right in front of the LiDAR.

4.4.2 Data Processing

In this paper [24], Velodyne 64E LiDAR range scans have been projected as 2D maps similar to the depthmap of RGBD data. The projection is done using the following equations given in [24]

$$\begin{aligned} \theta &= atan2(y, x) \\ \phi &= arcsin(\frac{z}{\sqrt{(x^2 + y^2 + z^2)}}) \\ r &= [\theta / \Delta\theta] \\ c &= [\phi / \Delta\phi] \end{aligned} \tag{4.1}$$

Here $P = (x, y, z)^T$ is a 3D point and (r, c) is the 2D map position of its projection. θ and ϕ indicates the azimuth and elevation angle when observing the point. $\Delta\theta$ and $\Delta\phi$ are the average horizontal and vertical angle resolution between consecutive beam emitters,

respectively. This method is very similar to our LiDAR data processing for each dimension of the point cloud data. Each point cloud data has five different information stored in it. Based on this data, a five-dimensional image was created. The image is eight pixels tall (based on the LiDAR having eight beams), and 127 pixels wide. The image channels are depth, intensity, x, y, and z. Again, ground points and removed and a density-based clustering was performed and the center of the image corresponds to the points closest to the centroid of each cluster.

4.4.3 Ground Point Removal and Clustering

The ground point removal and clustering has been done for the processed point cloud data too. Both the ground point removal and clustering was done using the same method described in the previous chapter.

4.4.4 2D CNN architecture

The DL architecture is given in Table 4.1. In that table, CO = Convolution layer, PL = Max Pooling, BN = Batch Normalization, FC = Fully Connected, DR = Dropout, and SM = Softmax. It is noted that several architectures were examined, and this one produced the best results.

The DL network contained the following layers, as shown in Table 4.1. The training parameters are as follows: Batch size $1,024$; learning rate 0.002; momentum 0.90. The Stochastic Gradient Descent method was used for training.

4.5 Experiments

We have colleced a set of data for training and testing our algorithm. The same dataset as the SVM and MKL have been used in case which gave the opportunity to compared the performance of all the classifiers for one particular detection problem. All the data collection description has been briefly discussed in the previous chapter.

4.6 Results and discussion

The confusion matrices for the 2D CNN algorithm are given in tables 4.2 and 4.3 to show the accuracy of this technique. Here the reference data are in columns and the classified data are in rows.

The overall accuracy for DL CNN training is 99.76% and testing is 94.44%. It indicates that the algorithm perform well in classifying the beacon. The CNN architecture was designed keeping in mind the dataset type and detection problem. There had been some testing with the algorithm to find the suitable architecture that can perform well. It was observed that when the DL was trained longer, the testing OA decreased, indicating over-training. This is one major factor that should be consider while implementing the deep learning methods. One of the major characteristics of DL is that its performance can not be predicted before implementing it. This is one of the reasons that the DL did not out-perform the linear SVM and non–linear kernels. Also, Matlab 2017 version was used to run the experiment which allows only three channels for input image data. Fusing more channels in the data might improve the results. There is another idea which might help to improve the classification results. Also, a hybrid system can be designed which will

have a combination of some extracted features and the DL technique. This idea is worth exploring for future cases. Finally, perhaps the DL system could be modified by adding more non–linearities into the system (e.g. radial basis functions, sigmoids, etc.), providing a more robust and more non–linear decision boundary, since the problem appears to be non-linearly separable.

Table 4.1 Deep Learning CNN Setup.

Layer	**Contents**	**Parameters**
Input	Image input	$[8 \times 127 \times 3]$
1	CO	20 $[1 \times 1]$ Stride $[1 \times 1]$
2	PL	$[2 \times 2]$ stride: $[1 \times 1]$
3	CO	5 $[1 \times 5]$ Stride $[1 \times 1]$
4	PL	$[2 \times 2]$ Stride $[1 \times 1]$
5	CO	5 $[3 \times 3]$ Stride $[1 \times 1]$
6	PL	$[2 \times 2]$ Stride $[1 \times 1]$
7	BN	-
8	FC	20 neurons
9	FC	100 neurons
10	DR	50%
11	FC	200 neurons
12	FC	100 neurons
13	FC	2 neurons
14	BN	-
15	SM	2 outputs
Output	Classifier	.

Table 4.2 Confusion Matrix for training data using DL CNN.

	Class 1 (Beacon)	**Class 2 (Non-beacon)**
Class 1 (Beacon)	7,814	15
Class 2 (Non-beacon)	15	4,858

Table 4.3 Confusion Matrix for testing data using DL CNN.

	Class 1 (Beacon)	**Class 2 (Non-beacon)**
Class 1 (Beacon)	2,005	206
Class 2 (Non-beacon)	152	4,070

CHAPTER V

CONCLUSION

5.1 Contribution

The target of this research work was to explore the possibilities of various linear and non-linear classifier implementation for our beacon detection case. Some promising results and a good comparison of different methods were achieved for the beacon detection case. Quanergy M8 LiDAR collects very sparse point cloud data as it has only eight beams compared to other higher beam counts LiDAR. Another challenge was implementing our system in real-time scenario. Though considering high number of hand crafted features performs better with linear and non-linear classifiers, it increases computational complexity and slight latency in real-time beacon detection. The SVM weights of individual features were calculated and the overall accuracy curve was plotted with respect to feature numbers. The optimal twenty features for SVM was chosen as it showed almost the similar accuracy as all the features combined. Despite of these drawbacks, all of the methods performed well and showed high efficiency. The linear SVM showed overall accuracy of 97.74% and 98.23% for training and testing data respectively. The MKL technique reached similar high efficiency even though with only seven hand crafted features. The DL implementation was the trickiest part of this research due to the complex structure and higher computation time. Most of the state-of-the-art DL methods for LiDAR point cloud data

are 3D CNN. Among them VoxNet [28], 3D shapenets [44], and Pointnet [33] methods were studied to implement on our problem. The data processing and initial algorithms were implemented with our collected data. The implementations were not successful as the neural network structures of these methods were too much complicated to apply for the sparse dataset for this problem. Also, these methods need really dense point cloud data for object detection. In most of the examples, their point cloud data was really dense that the object shape was clearly evident. They are suitable for multiple class detection cases where beacon detection is technically a binary classification problem with only beacon and non beacon outputs. Finally, their high computation time is unfeasible for a real time scenario. In the end, a basic 2D CNN was applied for this case. As the CNN has really simple structure, it was suitable for the sparse data and classification problem. Some signal processing was done on the point cloud data before inputting them to the CNN. At the end, it showed training overall accuracy 99.76% and testing overall accuracy 94.44%. There is scope of overtraining which ultimately reduces the overall accuracy.

5.2 Future work

There are many possible methods and scope of improvement for this object detection problem. Many of the possibilities were explored and their possible outcomes were noted while doing this research. There are many scope of improvement in this research that I would like to mention in this concluding note. The beacon detection system can be made more robust and efficient for real time implementation. This research can be expanded to multiple object i.e., people, vehicles etc. detection in one environment. The result will be

a good example to study the competence of different classifiers. Due to the data structure and time constraints, 3D CNN methods could not be implemented. There is scope of investigation whether the 3D CNNs will perform better if they are modified for the eight beam LiDAR data. The modification might be complicated and require much time and tuning to achieve the suitable neural network architecture. With the increasing demand and popularity of Advanced Driver Assistance System (ADAS), it can be predicted that these research possibilities will be explored in near future. As the 2D CNN did not out peform the SVM and MKL, then there might be a possibility of a hybrid algorithm implementation. In that system a few of the hand crafted features and the 2D CNN can be combined which might ultimately improve the overall accuracy.

REFERENCES

[1] I. Aleksander and H. Morton, *An introduction to neural computing*, vol. 3, Chapman & Hall London, 1990.

[2] D. Anderson, J. Keller, and C. S. Chan, "Fuzzy Integral and Computer Vision Library,", 2017.

[3] D. Arpit, Y. Zhou, H. Ngo, and V. Govindaraju, "Why regularized auto-encoders learn sparse representation?," *arXiv preprint arXiv:1505.05561*, 2015.

[4] J. E. Ball, D. T. Anderson, and C. S. Chan, "Comprehensive survey of deep learning in remote sensing: theories, tools, and challenges for the community," *Journal of Applied Remote Sensing*, vol. 11, no. 4, 2017, p. 042609.

[5] K. A. Brookhuis, D. De Waard, and W. H. Janssen, "Behavioural impacts of advanced driver assistance systems–an overview," *European Journal of Transport and Infrastructure Research*, vol. 1, no. 3, 2001, pp. 245–253.

[6] G. Chen, T. X. Han, and S. Lao, "Adapting an object detector by considering the worst case: A conservative approach," *CVPR 2011*, June 2011, pp. 1369–1376.

[7] X. Chen, K. Kundu, Y. Zhu, A. G. Berneshawi, H. Ma, S. Fidler, and R. Urtasun, "3d object proposals for accurate object class detection," *Advances in Neural Information Processing Systems*, 2015, pp. 424–432.

[8] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, 1995, pp. 273–297.

[9] A. Deshpande, "A beginners guide to understanding convolutional neural networks," *A Beginner's Guide to Understanding Convolutional Neural Networks–Adit Deshpande–CS Undergrad at UCLA ('19). Np*, vol. 20, 2016.

[10] P. Dollar, C. Wojek, B. Schiele, and P. Perona, "Pedestrian Detection: An Evaluation of the State of the Art," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 4, April 2012, pp. 743–761.

[11] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al., "A density-based algorithm for discovering clusters in large spatial databases with noise.," *Kdd*, 1996, vol. 96, pp. 226–231.

[12] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, "LIBLINEAR: A library for large linear classification," *Journal of machine learning research*, vol. 9, no. Aug, 2008, pp. 1871–1874.

[13] F. Ferri, M. Gianni, M. Menna, and F. Pirri, "Dynamic obstacles detection and 3d map updating," *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*. IEEE, 2015, pp. 5694–5699.

[14] N. Friedman, D. Geiger, and M. Goldszmidt, "Bayesian network classifiers," *Machine learning*, vol. 29, no. 2-3, 1997, pp. 131–163.

[15] G. Geetika, "A Survey of Classification Methods and its Applications," *International Journal of Computer Applications*, vol. 53, no. 17, 2012, pp. 14–16.

[16] D. Geronimo, A. M. Lopez, A. D. Sappa, and T. Graf, "Survey of pedestrian detection for advanced driver assistance systems," *IEEE transactions on pattern analysis and machine intelligence*, vol. 32, no. 7, 2010, pp. 1239–1258.

[17] A. Golovinskiy, V. G. Kim, and T. Funkhouser, "Shape-based recognition of 3D point clouds in urban environments," *Computer Vision, 2009 IEEE 12th International Conference on*. IEEE, 2009, pp. 2154–2161.

[18] M. Gönen and E. Alpaydın, "Multiple kernel learning algorithms," *Journal of machine learning research*, vol. 12, no. Jul, 2011, pp. 2211–2268.

[19] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*, vol. 1, MIT press Cambridge, 2016.

[20] U. Handmann, T. Kalinke, C. Tzomakas, M. Werner, and W. Seelen, "An image processing system for driver assistance," *Image and Vision Computing*, vol. 18, no. 5, 2000, pp. 367–376.

[21] V. Hegde and R. Zadeh, "Fusionnet: 3d object classification using multiple data representations," *arXiv preprint arXiv:1607.05695*, 2016.

[22] L. Kaufman and P. J. Rousseeuw, *Finding groups in data: an introduction to cluster analysis*, vol. 344, John Wiley & Sons, 2009.

[23] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, 2015, p. 436.

[24] B. Li, T. Zhang, and T. Xia, "Vehicle detection from 3d lidar using fully convolutional network," *arXiv preprint arXiv:1608.07916*, 2016.

[25] K. Lillywhite, D.-J. Lee, B. Tippetts, and J. Archibald, "A feature construction method for general object recognition," *Pattern Recognition*, vol. 46, no. 12, 2013, pp. 3300–3314.

[26] C.-H. Lin, J.-Y. Chen, P.-L. Su, and C.-H. Chen, "Eigen-feature analysis of weighted covariance matrices for LiDAR point cloud classification," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 94, 2014, pp. 70–79.

[27] J. Luo, *Quadratic Surface Support Vector Machines with Applications*, doctoral dissertation, North Carolina State University, 2014.

[28] D. Maturana and S. Scherer, "Voxnet: A 3d convolutional neural network for real-time object recognition," *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*. IEEE, 2015, pp. 922–928.

[29] J. C. McCall and M. M. Trivedi, "Video-based lane estimation and tracking for driver assistance: survey, system, and evaluation," *IEEE Transactions on Intelligent Transportation Systems*, vol. 7, no. 1, March 2006, pp. 20–37.

[30] R. T. Ng and J. Han, "E cient and E ective Clustering Methods for Spatial Data Mining," *Proceedings of VLDB*. Citeseer, 1994, pp. 144–155.

[31] A. J. Pinar, J. Rice, L. Hu, D. T. Anderson, and T. C. Havens, "Efficient multiple kernel classification using feature and decision level fusion," *IEEE Transactions on Fuzzy Systems*, vol. 25, no. 6, 2017, pp. 1403–1416.

[32] G. Postica, A. Romanoni, and M. Matteucci, "Robust moving objects detection in lidar data exploiting visual cues," *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*. IEEE, 2016, pp. 1093–1098.

[33] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, vol. 1, no. 2, 2017, p. 4.

[34] A. Rakotomamonjy, F. Bach, S. Canu, and Y. Grandvalet, "More efficiency in multiple kernel learning," *Proceedings of the 24th international conference on Machine learning*. ACM, 2007, pp. 775–782.

[35] P. Rashidi and H. Rastiveis, "Ground Filtering LiDAR Data Based on Multi-Scale Analysis of Height Difference Threshold," *ISPRS-International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 2017, pp. 225–229.

[36] S. R. Safavian and D. Landgrebe, "A survey of decision tree classifier methodology," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 21, no. 3, May 1991, pp. 660–674.

[37] J. Schmidhuber and S. Hochreiter, "Long short-term memory," *Neural Comput*, vol. 9, no. 8, 1997, pp. 1735–1780.

[38] S. Song and J. Xiao, "Deep sliding shapes for amodal 3d object detection in rgb-d images," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 808–816.

[39] S. Sonnenburg, G. Rätsch, C. Schäfer, and B. Schölkopf, "Large scale multiple kernel learning," *Journal of Machine Learning Research*, vol. 7, no. Jul, 2006, pp. 1531–1565.

[40] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller, "Multi-view convolutional neural networks for 3d shape recognition," *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 945–953.

[41] L. Wang and Y. Zhang, "LiDAR Ground Filtering Algorithm for Urban Areas Using Scan Line Based Segmentation," *arXiv preprint arXiv:1603.00912*, 2016.

[42] A. R. Webb, *Statistical pattern recognition*, John Wiley & Sons, 2003.

[43] L. Wellhausen, R. Dubé, A. Gawel, R. Siegwart, and C. Cadena, "Reliable Real-time Change Detection and Mapping for 3D LiDARs," *2017 IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*. IEEE, 2017, pp. 81–87.

[44] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3d shapenets: A deep representation for volumetric shapes," *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1912–1920.

[45] Z. Xu, R. Jin, H. Yang, I. King, and M. R. Lyu, "Simple and efficient multiple kernel learning by group lasso," *Proceedings of the 27th international conference on machine learning (ICML-10)*. Citeseer, 2010, pp. 1175–1182.

[46] Y. Zheng, D. Liu, B. Georgescu, H. Nguyen, and D. Comaniciu, "3D deep learning for efficient and robust landmark detection in volumetric data," *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, 2015, pp. 565–572.