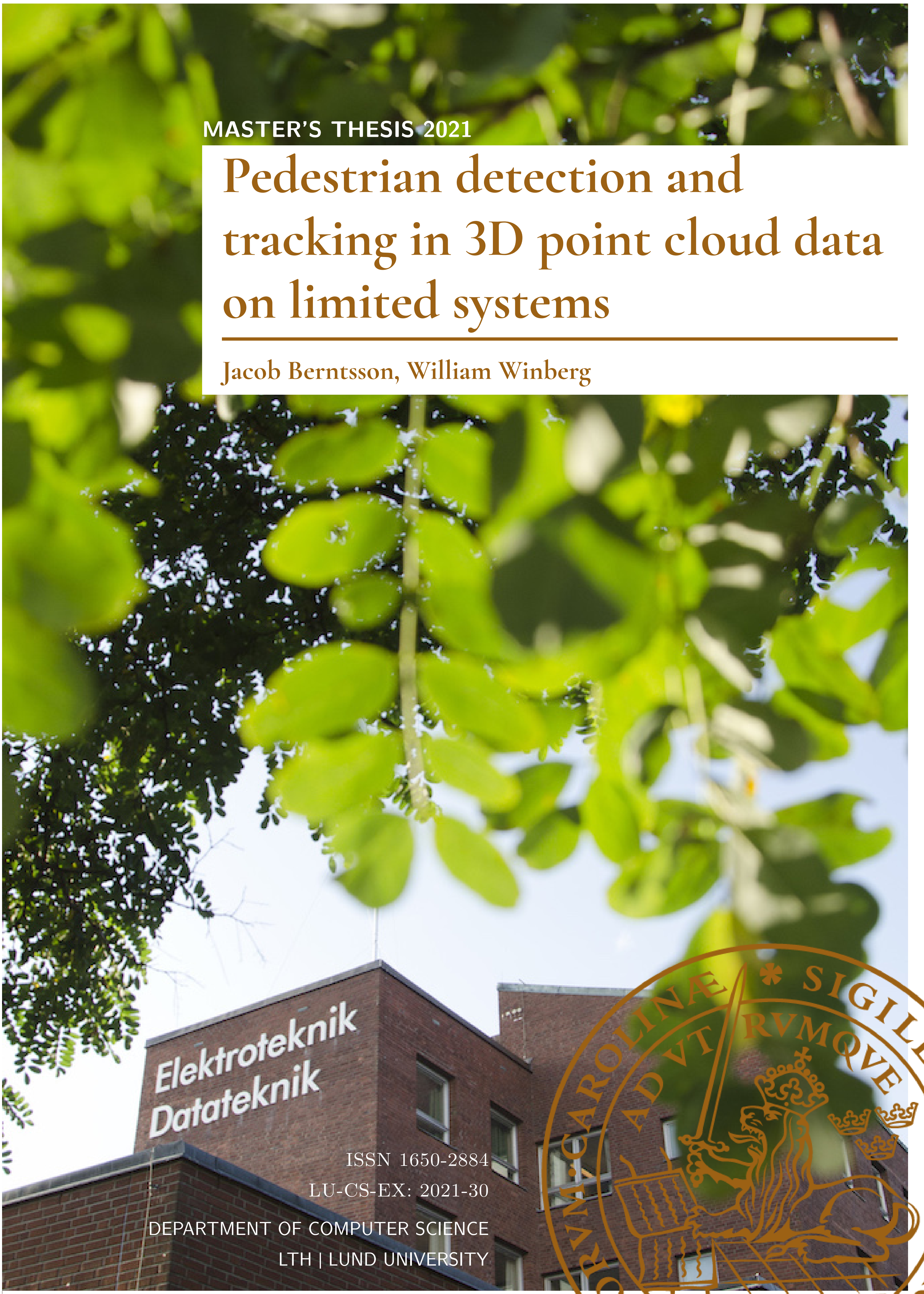


MASTER'S THESIS 2021

Pedestrian detection and tracking in 3D point cloud data on limited systems

Jacob Berntsson, William Winberg

ISSN 1650-2884
LU-CS-EX: 2021-30

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

MASTER'S THESIS
Computer science

LU-CS-EX: 2021-30

Pedestrian detection and tracking in 3D point cloud data on limited systems

Persondetektering och spårning med LiDAR på begränsade system

Jacob Berntsson, William Winberg

Pedestrian detection and tracking in 3D point cloud data on limited systems

Jacob Berntsson
ja8421be-s@student.lu.se

William Winberg
wi2203wi-s@student.lu.se

September 3, 2021

Master's thesis work carried out at
the Department of Computer Science, Lund University.

Supervisors: Niklas Hansson
Carl-Axel Alm
Elin Anna Topp, elin_anna.topp@cs.lth.se

Examiner: Jacek Malec, Jacek.Malec@cs.lth.se

Abstract

The purpose of this Master's Thesis is to detect, track and classify pedestrians in stationary LiDAR point cloud data, and investigate if this is possible to do so on systems with limited hardware. To do this we implement an complete pipeline solving all aforementioned steps, with interchangeable parts so that different methods can be tested and compared. The pedestrian detection was achieved using a combination of background filtration and clustering, with several different background filtration methods implemented and tested. The pedestrian tracking was done by implementing a tracking system matching pedestrians between frames based of proximity, and several state estimation techniques to be able to accurately propagate the pedestrian's position between frames. To accurately classify pedestrians, traditional machine learning methods were used and compared to more advanced neural network classifiers. The classifiers were trained on data extracted from a simulated environment. The complete pipeline was then tested on a small single board computer to see if it was possible to do the pedestrian detection and tracking in real time. By limiting the frames per second, as well as by discarding different ratios of the point cloud we were able to run the entire solution in real time on the limited system with only small changes in the detection, tracking and classification performance.

Keywords: LiDAR, Point cloud, Tracking, Classification, 3D data processing, Limited Systems

Acknowledgements

We would like to thank our supervisors Niklas Hansson and Carl-Axel Alm for always quickly providing the material and information we needed to succeed, and for their kind words and support. We would also like to thank our supervisor Elin-Anna Topp for great advice and pointing us in the right direction, as well as helping us produce an excellent Master's Thesis paper.

Contents

Chapter 1
Introduction

Surveillance systems are widely used in today's society in the form of security solutions able to monitor locations and record the movement of individuals and objects within the monitored scene. These systems are often installed in remote locations, and since high latency can be very costly, i.e., when monitoring critical infrastructure, it is desirable to run the data processing locally on the device.

Detecting and tracking pedestrians has many applications in security and automotive industries. Several different sensors, such as RGB camera, thermal cameras and radar are currently standards within the industry, however, not all of these technologies are applicable to all use cases, and they each come with their own advantages and disadvantages.

In this thesis we explored the use of Light detection and ranging (LiDAR) to try and improve performance on use cases for pedestrian detection and tracking where already established technologies fail to perform.

We implemented a pipeline able to solve this problem using several different already established methods for LiDAR data processing, evaluating their performance and finally suggesting an optimal combination of techniques. The pipeline will contain several well defined steps where for each of these we created several solutions and compare their performance on both simulated data and data collected from an actual LiDAR device.

It is especially in our interest to explore which established techniques perform well on hardware with limited memory and processing power to minimize monetary costs of using LiDAR to solve the aforementioned problem.

With this thesis we hope to increase understanding of the benefits and limits of LiDAR technology, both in computational metrics and investigation on how the LiDAR sensor performs in use cases where other sensors struggle.

This paper will first look into and discuss several different solutions to the pedestrian detection and tracking problem, giving an explanation of the implementation and performance of each of these, as well as discussing eventual problems and briefly suggesting solutions to these. We also aim to give the reader a better understanding of LiDAR technology and 3D data processing.

1.1 Task and purpose

The purpose of this thesis is to explore how to increase the number of possible applications for LiDAR by minimizing the computational cost, and thus the hardware cost, of point cloud processing. We do this in the context of pedestrian tracking and classification. We hope that by investigating this we will enable pedestrian tracking to run in real-time locally on the LiDAR device. We imagine that this would open up for several new applications for LiDAR due to removing need for bulky and expensive processors or extensive cloud computing. LiDAR technology is becoming cheaper to produce as its adoption becomes wider, and with that a number of new possible applications are emerging.

The task were to investigate several different pedestrian tracking and detection techniques which have already been established, and evaluate their performance and relevance for use in limited systems. The task were divided into four distinct parts which are data gathering, dynamic object detection, object tracking and pedestrian classification. We implemented several different solutions to each part, so that we could then compare different combinations of them and conclude which approach/approaches were the most viable. We also explored the trade off between processing speed, tracking and classification accuracy.

The data gathering focused on creating and extracting data from a simulated environment. In this thesis we used the simulator Carla [1], an open source driving simulator, to create several scenes containing labeled LiDAR data. Using a simulator enabled us to automate the data gathering and labeling to create an extensive LiDAR data set, as well as creating several environments to test the performance of our implementation. We also investigated whether it is possible to use the simulated data to create a model able to perform equally well on actual real life LiDAR data.

Due to the curse of dimensionality [2] processing 3D data requires significantly more computationally heavy operations for clustering, tracking and classification than at lower dimensions. LiDAR data is represented as a set of points in 3D space, from now on referred to as a point cloud. To be able to perform these steps in real time on a limited system, we believed that it would be necessary to perform extensive pre-processing on the point cloud data to separate measurements coming from dynamic objects of interest from those resulting from static objects belonging to the background. The dynamic object detection separates these measurements and then clusters them into individual objects of interest.

The object tracking uses different techniques to estimate the position and velocity (state) of tracked objects. It then uses these state estimations to propagate the position of tracked objects within the scene to more accurately map them to new observation in sequential LiDAR frames. This step also includes determining whether any newly observed object should be tracked, and determining when an object should no longer be tracked and considered lost. By estimating the state of the object, we could also approximate its future position which is necessary to improve the tracking and separate objects within close proximity to one another, as well as tracking objects which are partially or entirely obscured for a short amount of time. Our goal was to achieve multi target tracking, allowing us to track several dynamic objects simultaneously.

The pedestrian classification were partly based on simple assumptions about the size, speed and movement pattern of humans. Defining some basic properties and their limits for, e.g., human shape and movement helped us get an initial idea of whether it is likely for an observation to have come from a pedestrian. We then applied traditional machine learning methods on the objects which we deem being human-like to classify objects into two classes, pedestrian and other. We explored traditional machine learning algorithms using manual feature extraction, as well as neural networks with automatic feature extraction, to classify pedestrians.

1.1.1 Testing on a limited system

One of the main focuses of this thesis is to investigate whether it is possible to perform pedestrian detection and tracking on a single-board computer with limited processing power. A single-board computer is a functional computer made on a single circuit board. In order for the circuit board to function as a normal computer it contains microprocessors, memory, input/output devices and other features. The goal is to use one of these single-board computers to get the tracking and detection system to run in real time and to do this we have employed several methods that we imagine will bring down the processing requirements significantly.

1.2 LiDAR

LiDAR has been used for decades in applications, such as satellite tracking, meteorology measuring equipment, military targeting and surface mapping of the moon, as well as the more recent use in self-driving cars and robotics. With the development of more cost effective and more accurate LiDAR devices in the last decade, there has been a lot of progress using LiDAR for object detection and collision avoidance systems in autonomous vehicles. This includes detecting moving objects like pedestrians, animals and other vehicles, as well as stationary objects like signs, lampposts and buildings. The use of of LiDAR in autonomous vehicles is mainly due to its performance in non optimal conditions, such as darkness or when the pedestrian is wearing colors camouflaging them against the background, where other methods to detect pedestrians, e.g., RGB cameras, loose their reliability.

The advances in pedestrian detection raise an interest in LiDAR technology as an improvement in fields where RGB and heat cameras have previously been the primary sensor.

A LiDAR device is able to measure distances by illuminating its surroundings with laser light, and measuring the time it takes for the laser to reflect back into a sensor located on the device. By illuminating a limited field of view (FOV) with many laser beams the device is able to construct a point cloud representing a 3D image of the surrounding area. Since the origin of the light is the LiDAR itself, it has an advantage over normal cameras since it is able to capture its surroundings even when no other light source is present. Using the 3D point cloud from the camera it is also possible to derive other useful information, like the distance between two objects, their speed and acceleration in 3D space, in a more accurate way than some other recording devices.

Due to the main purpose being to explore new potential uses by maximizing computational efficiency, and us imagining the most computationally heavy part to be data handling, we limit ourselves to a stationary LiDAR with a limited FOV. Using a stationary LiDAR will hopefully enable us to efficiently filter out everything but the necessary data, and the limited FOV will greatly reduce the scale of the data.

We expect that the LiDAR device will need some initial calibration upon being mounted.

This calibration will most likely include fine tuning parameters, and/or creating a background model through some pre-processing steps.

While it is possible to detect and track many different things using LiDAR, this report will focus only on pedestrian tracking. We will consider pedestrians moving in scenes with a flat ground plane, as well as curved surfaces.

1.3 Related work

Since this report only considers stationary LiDAR's there will be a static background present in every scene. Different methods of extracting the background from the 3D point cloud have been explored in [3], [4], [5], [6], [7], and [8].

In [3] they implement a 3D density statistic filter (3D-DSF) able to separate static background points from dynamic foreground points.

In [4] they model the background by recording for each horizontal and vertical angle combination the recorded maximum distance over some initial frames.

In the same article they also determine if a point is static or dynamic by taking the points from the current point cloud and counting the number of near lying points in a past point cloud.

In [6] they use an octree structure to represent the 3D space scanned by the LiDAR. Similarly, in [5] they use Dempster Shafer Theory for creating a 3D occupancy grid also represented by an octree structure which describes whether a small subspace in the scanned area is occupied or not.

In [8] they use an Bayesian approach to estimate the probability of a point matching a pre-defined motion model, which in the scope of their report is the movements of a car between frames.

One of the most used methods for clustering points is called Density-based spatial clustering of applications with noise (DBSCAN) [9]. DBSCAN is a data clustering algorithm that given a set of points in some space is able to group points together into separate clusters based on proximity.

Another method for clustering is the Mean shift algorithm [10], which is a centroid based clustering algorithm.

A way of clustering the point clouds generated by a LiDAR in an efficient way has been explored in [11]. In the article, they introduce Lisco, an algorithm for Euclidean-distance based clustering of LiDAR point clouds.

Several methods of tracking a dynamic object as it moves through the scene has been explored in [12], [13] and [14].

In [12] they compare using a Kalman filter and a particle filter to recursively update the position and velocity state of a spherical item moving through the LiDAR point cloud scene.

In [13] they track pedestrians and other dynamic objects with a particle filter by estimating state variables x, y, z positioning and their velocity vx, vy, vz in those directions.

In [14] they describe several uses of the Particle filter, one of them being target tracking. They also use Rao–Blackwellization which is using a particle filter in combination with a Kalman filter to achieve better accuracy in the state estimations.

Accurately classifying clustered 3D point clouds as pedestrians has been explored in [15], [16], [17] and [18]. In [15] they discuss which features are the most important for the classification, and investigate how well SVM, Random forest classifiers and K-means classifiers manage to perform on the task.

In [16] they train two CNN's to detect pedestrians using the 2D depth map and reflectively map resulting from a LiDAR point cloud, and use a form of sensor fusion to combine the classifiers for better accuracy.

In [17] they create a voxel representation of the area using the 3D point cloud, and then use a 3D CNN to create a model able to accurately classify pedestrians.

In [18] they train a deep neural network able to classify objects based only on the point cloud set without having to transform the data into a voxelization or other representation.

1.4 Disposition

The outline of the thesis is as follows: Chapter 2 includes the theory and tools used in the thesis and also the proposed solution to the object detection and tracking methods. The chapter summarizes information regarding already established techniques for data gathering, followed by object detection and lastly it covers object tracking and classification.

In Chapter 3 we present the metrics and methods used to evaluate the dynamic object detection, as well as explain our specific implementation of the techniques presented in chapter 2.

In chapter 4 we will present the performance of our different implementations in a concise and visual way.

In chapter 5 we will discuss the results presented in chapter 4 and also present which techniques seem to perform best in different scenarios. We will present our suggested optimal implementation of the pipeline, as well as discussing in which areas it excels and where it falls short.

1.5 Ethics

Pedestrian tracking using LiDAR data has ethical advantages over other established techniques like RGB cameras, one being that no detailed features like facial structure or hair color that could be used to identify individual people's identities are present in the data. When a pedestrian moves through the 3D point cloud scene the only discernible features are their height, width and stride. While these features are definitely enough to identify people to some degree, especially when trying to find a known individual, they are significantly less distinct features than those captured by regular cameras.

Privacy is a huge concern when recording RGB video, especially with new regulation, such as GDPR, since individuals are often identifiable from the footage alone. Replacing the video with LiDAR data would partially protect the identities of the people being recorded in public settings without their explicit consent.

Recording and tracking movements of pedestrians has a lot of uses in automotive applications, security and robotics, where the LiDAR is used to help avoiding collisions, or enabling interaction with humans. Robust tracking and detecting of pedestrians is a huge requirement for deployment of autonomous vehicles, since they are some of the most vulnerable actors in traffic. Each year roughly 1.35 million people are killed in avoidable traffic accidents, and creating safer and more efficient traffic by removing the human factor would greatly improve world wide health.

Detecting and tracking human movement can also be used with ill intent in, e.g., military applications, such as automatic targeting for ballistics. As we will show in this report, a LiDAR is very capable of determining the relative position of objects moving in the scene, and to further use this information to automatically aim a turret towards the detected object would be trivial. More common sensors like RGB cameras can surely be used in similar ways, and it is unclear if LiDAR technology offers any further improvements over already established techniques in this area. Since they are also anonymous in their detection, it would be difficult to discern enemy combatants from friendly ones, which limits its use in military applications.

The company which hired us to do this thesis are part of an initiative called the UN Global Compact and have pledged to adhere to several principles, two of which are

Principle 1: Businesses should support and respect the protection of internationally proclaimed human rights; and Principle 2: make sure that they are not complicit in human rights abuses.

1.6 Statement of Contribution

The work behind this thesis has been divided equally between the two authors. William was responsible for the tracker and state estimations, while Jacob focused on the simulator and data gathering. The rest of the programming, testing, and writing of the report was done in unison.

Chapter 2
Background

In this chapter we present many of the techniques mentioned in the related work section, as well as the motivation between the selection of each of these. We will also present the different data sets available to us.

The first section in this chapter will be about the acquisition of data. The gathered data will be used to generate data sets which will be used to train the classification models. The three major steps for data processing, which are data acquisition, data labeling and improvement of existing data will be presented. The pros and cons between collected data, simulated data and open-source data sets will also be discussed.

The dynamic object detection section will discuss both background filtration and clustering.

Object tracking will be about both Kalman filters and particle filters and how they can be used to estimate the state of tracked objects.

The last section will be about classification methods and how they can be used in order to determine class belonging of different objects.

2.1 Data Gathering

The most important step when it comes to solve any supervised machine learning task is the data gathering, since the accuracy of a classifier greatly depends on the quality of the data set. So how does one acquire a good data set for training models? According to [19] there are mainly three steps that can be used for data processing, namely data acquisition, data labeling and improvements on existing data and models. Data acquisition is the method to use if there is little or no data available initially. In order to acquire more data one could, for instance, search for pertinent data sets on the Web or create own data sets by using hardware to gather images, readings and so forth.

When the desired amount of data has been acquired it is time to label individual examples from the data. Different techniques, that are explained in [19], can be used depending on how much of the data has already been labeled. For instance, if the data contains enough labels it can enable a self labeling semi-supervised learning technique, which makes prediction by exploiting both the labeled and unlabeled data. However, if not much labeled data is available, one way is to label the data manually with the help of, e.g., active learning which focuses on selecting the most interesting unlabeled examples or crowd-sourcing which focuses on labeling lots of data with less accuracy.

The last step can be used in order to improve already labeled data sets or enable active learning in the model. Techniques for data cleaning, such as HoloClean [20] and ActiveClean [21], could be used in order to remove unwanted noise or biases from the data set. Both HoloClean and ActiveClean are able to detect and correct corrupt or inaccurate data. The difference between the two is that HoloClean uses probabilistic inference and a two-layer neural network model in order to repair data, while ActiveClean uses a progressive and iterative clustering techniques to identify the dirty data. The data set can then be re-labeled in order to improve the quality of the data set. In order to improve the model training transfer learning could be applied to the model. This means that an existing well-trained model is used to gradually train a new improved model.

2.1.1 Cepton LiDAR

The physical LiDAR that we have available for testing is a Cepton Vista-P60 [22]. According to the specifications, the Cepton Vista-P60 is able to measure at a range of 200 meters with a reflectivity of 30%. It has a fixed field of view, 60^o horizontal and 22^o vertical, and an angular resolution of 0.25^o x 0.25^o. It has a scan rate of 10 scans per second. The Cepton Vista-P60 LiDAR has been used to gather real-world data from several different environments, e.g., construction sites, parking lots and train stations. The collected data is raw measurements from the LiDAR, which means that each measurement contains the coordinates of the objects hit by the laser beams in relation to the LiDAR itself, as well as an intensity which refers to the ratio between emitted and reflected light of the object hit. It also contains the horizontal and vertical angle of the points and the distance to the object hit. The generated data is unlabeled which means that it has to be labeled manually.

2.1.2 Simulated Data

Another way of acquiring data is to simulate it with the help of game engines and/or graphics engines. In [23] they used a video game in order to simulate data that was used to train computer vision models that were able to achieve similar results compared to real-world data sets. An advantage of using simulated data is that the annotation of data becomes a much easier task since it is easy to get information of different objects in a scene. The possibilities of adapting the data to one's own liking are also an advantage with simulated data.

A tool for simulating data is described in the subsection below.

Carla simulator

Carla [1] is an open-source autonomous driving simulator created for autonomous driving research. It has been developed from the ground up in order to support development, training and validation of autonomous driving models. Carla has been implemented as an open-source layer over Unreal Engine 4 [24] which provides modern lifelike render quality and realistic physics. An example of a scene from the simulator can be seen in figure 2.1.

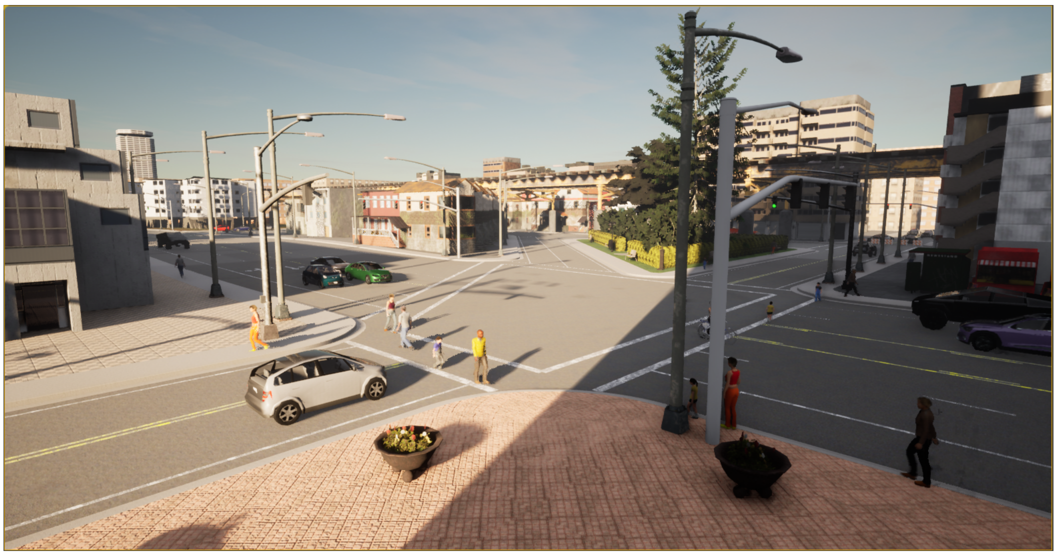


Figure 2.1: A frame from the Carla simulator including vehicles and pedestrians.

Carla is a server-client system, where the server is in charge of the simulation and graphical rendering of the scene. The client is a Python API in charge of the interactions between an agent and the server. Carla contains a multitude of assets and methods for simulating an urban driving environment containing 3D models of both static and dynamic objects, such as pedestrians and buildings. Sensors like RBG cameras or LiDAR devices can be placed within the environment and record images of the 3D environment and the objects moving within it. It is also possible to change the environmental settings of the simulator, settings, such as different kinds of weather and the time of day. There is also a navigation system that can be used to simulate and control the movements of cars and pedestrians moving through the 3D landscape. These are just some examples of what Carla provides, for a full list visit Carla's official website [25]. In order to suite Carla to a specific need, a developer can modify existing assets and methods, as well as add their own assets and scripts in order to better suite a specific area of use.

2.1.3 Open-Source data sets

There is a multitude of articles that use open-source data sets in order to evaluate different models. For instance, Qi et al. (2017) use the ModelNet40 [26] shape classification benchmark in order to evaluate the performance of the model in their article [18]. The ModelNet data set was created in order to evaluate the 3D shapenet [27] model. ModelNet is a large-scale object data set consisting of 3D CAD models generated with modern high quality computer graphics. The ModelNet40 data set contains the 40 most common object categories from the ModelNet data set. The data set is labeled which means no labeling is required. However, the models in the different categories of the ModelNet40 data set are not very lifelike, i.e., the models in the person category may tend to look more like a cartoon than a real human being.

2.2 Dynamic Object detection

Detecting a dynamic object is done by dividing up the points in the cloud into two sets, static and dynamic. Individual dynamic objects can then be found by using distance based clustering on the dynamic set. We mean to explore several different methods that achieve this so that we can compare their performance.

2.2.1 Background filtration

Since we have limited ourselves to stationary LiDAR data we know that there will be a relatively constant static background present in any scene. We wish to explore different methods of creating a representation of that background. The dynamic object detection is based on the assumption that any dynamic object will not be part of the background, and therefore all remaining points in the foreground must be dynamic. By creating a sufficiently accurate model of the background we will be able to label each point in the cloud as dynamic or static.

The background filtering methods all work by creating a 3D space representation of the background from analyzing subsequent scans from the LiDAR, and most of the methods need a pre-processing step to determine the background model. This also means that it will not perform well in some use cases when the background changes regularly, e.g., when monitoring parking spaces or train stations.

We have implemented some of the methods discussed in related work [4] [3]. They are discussed in further detail in this chapter, and these were primarily chosen due to their proven performance and our delimitation of only using stationary LiDAR data.

In [8] the state belonging (static, dynamic) is determined by using an Bayesian approach to estimate each points probability of belonging to a dynamic or static object. The polar space is subdivided into many cells each containing a mixture of Gaussian's able to accurately model the non-gaussian probability of an observation emerging from a dynamic object. We have chosen to not use this method as [4] and [3] performed better in our specific use case.

In [5] and [6] they use a octree, which is a tree data structure able to divide 3D spaces into 8 octants, which becomes the children of the root node. These octants are then recursively divided into 8 more octants, which in turn become their children in the tree. This structure enabled them to more efficiently determine if a octan was occupied, by considering an octan to be occupied if 5 or more of its children nodes were occupied. We opted against this since the methods described in [3] and [4] had similar performance, and due to the background filtration step being only a small step of the entire process pipeline.

Maximum angular distance filter

The data recorded by the LiDAR contains the horizontal and vertical angle, as well as the distance to every measurement. By recording the maximum distance measured for each angle combination it is possible to create a model of the static background present in a scene.

Then when the LiDAR performs subsequent scans it determines if a point belongs to the background or not by comparing the measured distance to the maximum distance for that angle combination. All points that are further than a threshold from the maximum distance can then be classified as belonging to the dynamic set.

Nearest Neighbour filter

By comparing point clouds separated by a short temporal window it is possible to separate the foreground from the background. This works because measurements coming from static objects will be very similar in both point clouds. When a new point cloud is recorded by the LiDAR it is possible to count the number of close neighbours in the past point cloud, and since the static points accumulate they will have a lot of close neighbours while the dynamic points will not, which enables separation of the two sets.

3D density filter

This method works by dividing the scene into many 3D sub-spaces (cubes) and measuring the number of points present in each subspace over several LiDAR frames. Since static objects result in similar measurements the sub-spaces containing background objects will have a lot of measurements recorded within them, and it becomes possible to separate the background from the foreground based on the number of points detected in each subspace. This attribute is generally referred to as the density of the subspace.

Plane segmentation filter

Another way of removing static points from the point cloud is to use plane segmentation. In order to successfully identify the ground plane the well known RANdom SAmple Consensus (RANSAC) algorithm is used [28]. The RANSAC algorithm for ground plane segmentation works by sampling some number of points present in the point cloud and from these constructing a plane in 3D space. It then calculates the number of other points within a threshold distance from this plane, these points are referred to as the inlier set. This process is repeated a pre-defined number of times with different points used to construct the plane at each iteration, and at last returns the plane it calculated to contain the maximum number of other points, also known as the maximum inlier set.

Point cloud distance filter

The point cloud distance filter is based on background subtraction methods commonly used with other sensors, such as RGB cameras. In order to achieve background subtraction an octree [29] can be used to store a background point cloud containing a static environment without moving objects. An example where this has been done can be found in the article [30]. Then the current point cloud is added to the octree and a comparison algorithm is executed on the respective points of the background point cloud and the current point cloud. The resulting octree will contain distance values, where a low value indicates it is a static point and a high value indicates that it is a dynamic point. By determining which parts of the scene remain unchanged over time, i.e., belong to the static background, it is possible to remove these in every scene. To determine if a part of the scene does not change over time, it is possible to use the distance to the closest point when comparing two point clouds.

2.2.2 Clustering

To discover individual dynamic objects from the filtered data we used clustering methods to group together dynamic points into separate individual objects. Since we do not know the numbers of objects present in the scene during this step, we must employ methods able to dynamically calculate the number of clusters in a scene. Clustering points in 3D space is very computationally heavy, and the performance of the methods depends greatly on the number of points present in a scene. It might be necessary to reduce the filtered point clouds even further to achieve real time execution. This can be done by reducing the density of objects in the point cloud by randomly discarding a number of points before clustering.

A k-d tree is a binary search tree where the leaves in the tree are k-dimensional points. [31] It is structured such that every node that is not a leaf node divides a k dimensional space into two halves. This structure takes points in k-dimensional space and organizes them so that points that are close in proximity are also close in the tree. This structure enables efficient searching of close neighbours in the point cloud.

Mean-shift discussed in [10] is not robust against noise, which is prevalent in LiDAR data.

Lisco [11] is a way to continuously cluster point clouds and eliminate need for search-optimized data structures, such as k-d trees.

We chose to only implement and test DBSCAN due to it having similar time complexity to Lisco when the k-d tree structure is used, and due to it being very robust against noise.

DBSCAN

DBSCAN is a density based clustering algorithm. This method first checks if each point seems to belong to a cluster by checking the number of nearby neighbours within a short radius **r** defined by the user, if they have over a certain amount of neighbours they are labeled as core points. The clustering is based on creating edges from core points (nodes) to all points within distance r, and all reachable points are considered to belong to the same cluster. If a point is located in a low density region, and thus not reachable from any other point, it is considered an outlier (noise) and is not included in any cluster. It is very robust against noisy measurements and outliers in the data set. The algorithm struggles to cluster point clouds with high variety in the cluster density, since the radius **r** can not be appropriately chosen.

2.3 Object tracking

When an object has been detected we want to track its movements between different frames from the LiDAR. This step also includes determining which observations should be tracked, and when a tracked object should be considered lost. To accurately match objects between frames it is of outmost importance to be able to accurately determine the position and velocity of an object, so that its future position in following frames can be predicted. This will significantly improve our ability to accurately track objects moving in close proximity to one another.

Measurements from the dynamic object detection are only in the form of noisy position estimates $P^k = (x, y, z)^k$, but the velocity state at time k, $V^k = (v_x, v_y, v_z)^k$, can be naively estimated with two position measurements and the time separating them, Δt, as $\frac{P^k - P^{k-1}}{\Delta t} = V^K$. These estimates would however not be especially accurate and we need to employ some techniques to produce more accurate state estimations.

The techniques we have decided to use are the Kalman filter, as well as the particle filter both presented in [12]. The Kalman filter is known to perform well in position tracking, but due to the Kalman filter being based on some assumptions about the distributions of the noise and measurements that we can not assume are true for our position measurements we also wish to try with a Particle filter to see if better performance is achieved.

2.3.1 State estimation

Kalman Filter

The Kalman filter is an algorithm that receives a series of noisy and inaccurate measurements, and uses these to estimate unknown variables. The Kalman filter estimates a probability distribution of the state variables at each time step that is significantly more accurate than single measurements. The Kalman filter is a recursive estimator, meaning that only the state from the past time step and current measurements are needed to make a new estimation. A more in depth explanation of the process can be found in [32].

To create a Kalman filter for this specific process we need a state transition model F^k able to propagate the state x^k one time step, $x^k = F^k x^{k-1} + w^k$, where w^k is process noise.

We also need an observation model H^k able to reduce our current state estimate to an observation, $z^k = H^k x^k + v_k$, where v_k is observation noise.

The measurement and observation noise is assumed to be a zero mean multivariate normal distribution $p(w^k) \sim N(0, Q^k)$ and $p(v^k) \sim N(0, R^k)$, where Q^k is the covariance of the process noise and R^k is the covariance of the observation noise.

The Kalman filter is designed to work on a linear system with Gaussian noise, and these assumptions do not hold true for 3D position tracking, but it is empirically known to perform sufficiently well even in non-linear cases as shown in [12].

The Kalman Filter algorithm is then performed as shown in algorithm 1 [32].

```
**Input**    : $s^{k-1}, P^{k-1}$
**Output** : $s^k_-, P^k_-$
**Initialize:** $P^0, F, H, Q, R$
$P^k = F * P^{k-1} * F^T$
$s^k = F * s^{k-1}$
$y^k = o^{kT} - H * s^k$
$S = H * P^{k-1} * H^T + R$
$K = P * H * S^{-1}$
$s^k_- = s^k + K * y^k$
$P^k_- = (I - K * H) * P^k$
**return** $s^k_-, P^k_-$
```

Algorithm 1: Kalman Filter

Particle filter

A particle is a hypothesis, a possible state. By generating many particles $\Xi^k = [\xi_1^k, \xi_2^k....\xi_N^k]$ based on current state estimations, and selecting from them the ones to be most likely, we are able to more accurately estimate the state, as shown in [12] and [14]. A more in depth explanation of the process can be found in [33]

We define some posterior probability $P(\xi_n|o)$ able to determine the whether a particle is likely hypothesis of the state.

By sampling with replacement from Ξ^k with probability $P(\xi_n|o)$ we are able to select the most likely particles and our state can be estimated as the mean of these. This step is generally referred to as importance re-sampling.

A particle filter makes no assumptions on linearity of the system or the distribution of the noise, which makes us believe that it might provide more accurate estimates of the state than the Kalman Filter.

The particle filter algorithm is shown in algorithm 2.

```
Input   : $o^k, s^{k-1}, N$
Output : $s^k$
Initialize: $\Xi^k = [\xi_1^k, \xi_2^k....\xi_N^k]$
$\Xi^k$ = propagate_particles($\Xi^{k-1}$)
$\Xi_-^k$ = importance_re_sampling($\Xi^k$, $P(\xi_n^k|o)$)
$s^k$ = mean($\Xi_-^k$)
$\Xi^k = \Xi_-^k$
return $\Xi^k$, $s^k$
```

Algorithm 2: Particle Filter

We have opted for using both a Kalman filter and a Particle filter so that we can compare the computational efficiency and the accuracy of the state estimations.

2.4 Classification

The two classification methods we have decided to use are the Support-Vector Machine (SVM) [15] and 3D-Convolutional Neural Network (3DCNN) [17]. While the methods discussed in [16] also seem interesting, creating the depth and intensity image seem unnecessary and while we imagine that sensor fusion between these could perform well it seems more applicable to use a separate sensor, such as a heat or RGB camera, which we have also decided are outside the scope of this thesis. The method described in [18] is only applicable to point clouds with a consistent number of points, and since this consistency is not present in the detected pedestrian point clouds we have decided not to use this method.

Supervised learning methods work by defining a loss function describing the error between the predicted classes of the model and the actual classes. Since the value of the loss function is low when the model produces accurate predictions, supervised learning works by minimizing the loss function using some form of optimization method. An optimization method works by calculating a descent direction for the function, i.e., a way to modify the parameters within the model to reduce the value of the loss function. This is done over many iterations until the value of the loss function does not change significantly between two iterations, this is referred to as the method converging to the minimum. In this thesis we will use two optimization methods known as Stochastic Gradient Descent (SGD) and ADAM. Stochastic gradient descent is based of regular gradient descent, where the direction of the gradient of the objective function is chosen as the descent direction. SDG estimates the gradient using only a randomly chosen subset of the data, which improves convergence time in high dimensional spaces. Adam is based on SDG, but the descent direction is instead calculated using running averages of the past gradients and their second moments.

2.4.1 SVM - Support-Vector Machine

Support vector machines (SVM) are a supervised learning method able to analyze data for classification. Given a training set $(x_1, y_1), ..., (x_n, y_n)$ where x is a feature vector in some dimension and y the corresponding class label the support vector machine is able to create a model that separates the training examples into the separate classes. The model does this by creating a hyperplane in the same dimension as the feature vector that divides the training set according to the labels, optimized on achieving the maximum margin between the two classes. SVM's are also commonly used to solve outlier detection and regression problems. In [15] they managed to use an SVM to train a classifier able to accurately classify humans in a point cloud with high accuracy.

A hyperplane with normal w and offset from the origin b can be written as the set of feature vectors satisfying

$$w^T x_i - b = 0 \tag{2.1}$$

An example of a hyperplane separating 2d points is shown in figure 2.2.

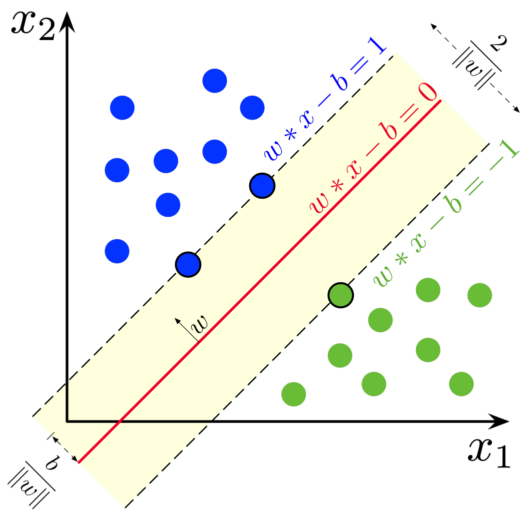


Figure 2.2: A hyperplane that maximizes the margin between two classes of 2d points

In a lot of cases the data is not linearly separable, and it is impossible to find a hyperplane that fully divides the two classes. In this case one can instead use a soft margin SVM. Instead of finding a hyperplane that absolutely separates the two classes its is possible to train the SVM by minimizing the hinge loss.

$$l(w) = \sum_{i=1}^{n} \max\{0, 1 - y_i(w^\top \cdot x_i - b)\} + \lambda ||w||^2 \tag{2.2}$$

The hinge loss function is 0 when the feature vector lies on the correct side of the margin, and when it is on the wrong side the value of the hinge loss scales with the vectors distance from the margin. The parameter λ determines a trade off between the size of the margin and the importance of feature vectors being on the correct side of the margin.

2.4.2 CNN - Convolutional neural network

Convolution neural networks are a class of artificial neural networks which are common to use in image analysis.

An artificial neural networks consists of an input layer that receives external data, an output layer which produces the result and in between those zero or more hidden layers. The first layer takes some input, modifies it, and passes it on to further layers. The final layer then outputs useful information to the task at hand, i.e., the probability of belonging to an object class. The connections between the layers, and how information is passed through them resemble the connections present in the human brain.

In the three dimensional case the network takes as input a tensor with shape $(N) * (width_x) * (width_y) * (height)$ where N is the number of input images.

The tensor is then convoluted with several filters present in convolutional layers, abstracting the tensor into several different feature maps. This process is shown in figure 2.3.

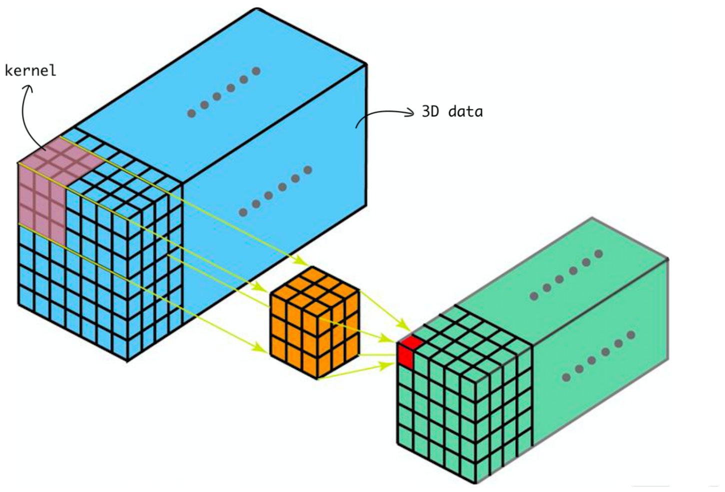


Figure 2.3: A visualisation of the 3D data (blue) convoluted with a filter (yellow) to abstract the data into a feature map (green)

After then convolution the feature map is put into a Max pooling layer, that extracts the most prominent features from the feature map. This process is shown in figure 2.4.

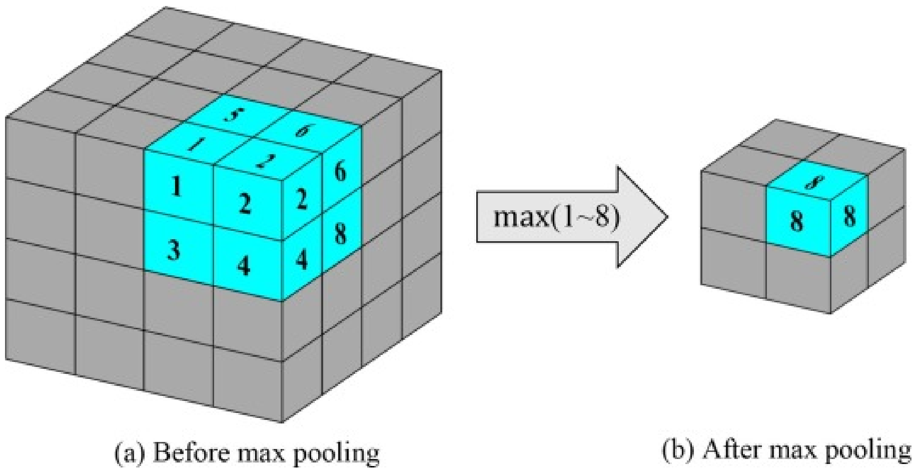


Figure 2.4: A visualisation of the feature map subject to max pooling

As we train the networks, the filter weights are updated to be able to extract the most relevant feature maps for classification.

Each feature map resulting from the convolution is passed through an activation function, a non linear transformation performed on the feature map before it is sent to the next layer. The activation function essentially decides which features maps should be propagated further into the network. In the sense of convolutional neural network, each filter is viewed as a neuron and the activation function decides if the neuron should fire, i.e., if the feature map should be passed to the next layer.

These types of CNN's have been shown to be able to classify 3D voxel images of pedestrians in [17]. The exact implementation and structure of the network is presented in chapter 3.

Chapter 3
Method & implementation

In this chapter we will discuss the different techniques that we have chosen to use in our system and the reason behind these choices. A detailed description on how we choose to implement these techniques will also be included.

First of we will discuss data gathering and why the choice of simulated data is a superior choice when it comes to annotation of data and creating our own data set. We will also explain how the simulator was set up in order to produce results similar to the Cepton Vista-P60 LiDAR. A section on how we collected real-world data for testing the classifier that was trained on a simulated data set is also presented.

Next up is a presentation of the methods used for background filtration and clustering of the point clouds and how they were implemented in our system.

Information regarding the implementation of the Kalman filter and particle filter that we used for object tracking and the pipeline used in our system is presented in the section object tracking.

We will also discuss the implementation of the support vector machine and convolutional neural network that was used in our system and how the data that we fed into these classifiers has to be preprocessed before it can be used by the classifiers.

Last up is the description of the methods we used in order to test our system on a single-board computer, and methods to reduce the computational requirements.

A visualization of the pipeline can be seen in 3.1

3.1 Data gathering

Due to the labeling of data being very costly we have decided to focus mainly on data acquisition from the simulated environment due to our belief that manually annotating enough data would take too much time and resources from the rest of the project. The Cepton Vista-P60 LiDAR is meant to be used mainly as a way to test if our models based on simulated data also

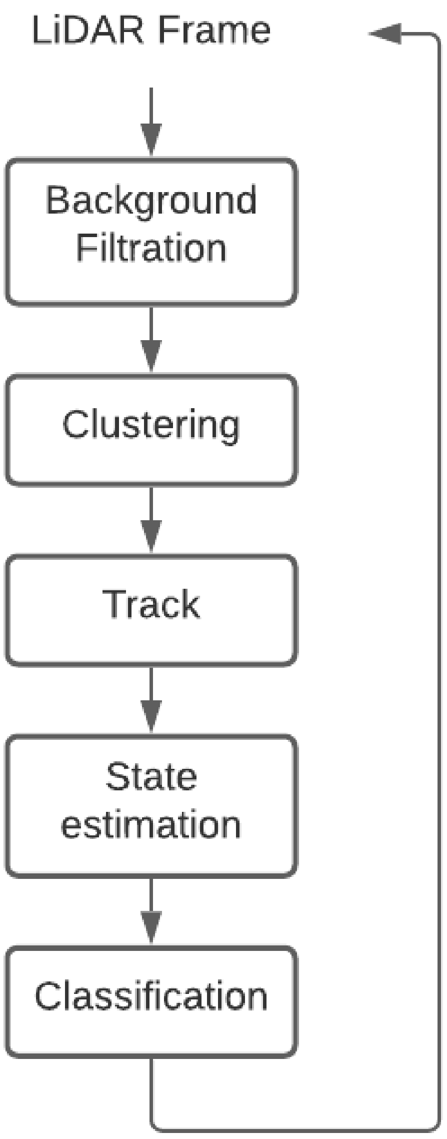


Figure 3.1: A visualisation of the Pedestrian detection and tracking pipeline

work for real world examples. We however plan to acquire a small test set of real life LiDAR data to see how well our classifiers created from simulated data perform in a real world case.

3.1.1 Simulation

Carla can be used in order to gather simulated data with the help of the different assets and methods provided. The first measure should be to create an environment containing various static 3D models in order to simulate, for instance, a traffic junction. In our case we used one of the existing city environments in Carla to act as our urban setting. This provided us with an environment consisting of static 3D models, however, in order for the simulation to look more realistic we also need to include dynamic 3D models. Carla has provided us with example scripts that can be used to spawn actors that will move around in the environment, i.e., simulating pedestrians, vehicles and other actors and their behavior in the environment. We have used both example scripts provided with the Carla simulator, as well as writing our own for the dynamic actors. When the urban environment is established it is time to gather data with the help of a sensor. Carla has a high amount of sensors, such as RGB cameras, IMU sensors and LiDAR sensors [34]. In our case, we used the semantic LiDAR sensor [35] which operates as a spinning 360^o LiDAR. The semantic LiDAR provides, in addition to the position of each point, the label describing the class belonging of that point, i.e., if the point was a result of the LiDAR scanning a pedestrian, car or tree. When using the sensors in Carla, especially the LiDAR sensors, it is of great importance that the sensor and the server are in sync. If a LiDAR sensor and the server become unsynchronized i.e, the tick rate of the server is not the same as the tick rate of the LiDAR, it will result in the LiDAR spinning either too fast or too slow. This will cause the LiDAR to measure incorrectly. This can be prevented by specifying the tick of the LiDAR to be the same as the tick rate of the server resulting in a complete 360^o measurement from the LiDAR.

We choose to use the Carla simulator to generate our simulated data set since the data that we gathered with the Cepton Vista-P60 would not be sufficient to train our classifier and with Carla we can generate close to unlimited amounts of data. Another reason for choosing the Carla simulator is that the data can be simulated to look more like the Cepton Vista-P60 generated LiDAR data, in contrast to online data sets, and be more adapted to our stationary use case.

In order to make the semantic LiDAR sensor in Carla to simulate the Cepton LiDAR some attributes had to be modified. The attributes of the Cepton Vista-P60 LiDAR has been stated in 2.1.1. The attributes of the Carla semantic LiDAR sensor have been modified in the following way.

Attribute	Type	Value	Description
Channels	int	128	Number of lasers.
Range	float	100	Maximum distance to measure in meters.
points_per_second	int	250000	The number of points in each point cloud frame.
rotation_frequency	float	10	LIDAR rotation frequency per second.
upper_fov	float	11	Angle of the highest laser vertically.
lower_fov	float	-11	Angle of the lowest laser vertically.
horizontal_fov	float	60	Angle of the lowest and highest laser horizontally.
sensor_tick	float	0.1	Simulation seconds between sensor captures.

With these attributes the Carla semantic LiDAR simulates the Cepton Vista-P60 LiDAR with great resemblance. Unfortunately, the semantic LiDAR sensor in Carla is not able to add disturbances to the measurements which the normal LiDAR in Carla is able to do. Disturbances, i.e., noise and drop-off, have to manually be added to the data after it has been gathered.

The gathering of data using the Carla simulator was done with the following procedure. An urban environment populated with vehicles and pedestrians was arranged. In this environment the semantic LiDAR sensor was attached to a vehicle that drove around in the urban environment. With this setup the gathered data contained varying events, i.e., pedestrians crossing the road, walking on the sidewalks, walking behind obstacles etc, as well as vehicles driving in front of the LiDAR obstructing large parts of the pedestrians. The dynamic setup enabled us to capture pedestrians from many different angles and distances, which is necessary to fully learn the possible attributes of a pedestrian.

We let the simulation run for a couple of hours, resulting in a data set consisting of around 33000 measurements. This raw data set was then split into training and validation sets consisting of either people or other, where other is everything that is not labeled as a pedestrian. This is possible since the semantic LiDAR sensor in Carla contains both semantic tags and indices. For instance, in order to get a point cloud of a single pedestrian in a measurement consisting of multiple pedestrians, vehicles and obstacles the following was done. For each measurement in the gathered data, all the points belonging to the same semantic tag and index of the object hit were saved as an individual point cloud. The index of the object hit ensures that a point cloud will not contain, for instance, multiple pedestrians. An example of how the data generated by the Carla semantic LiDAR looks like can be seen in figure 3.2

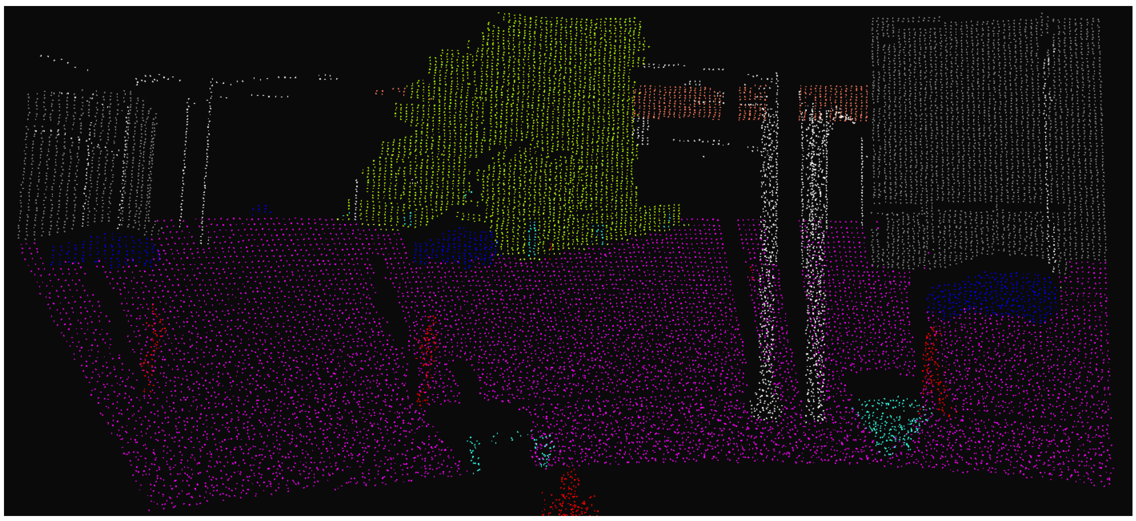


Figure 3.2: A frame from the Carla simulator Semantic LiDAR including vehicles, pedestrians and other objects where pedestrians are red, vehicles blue etc.

Before we save a point cloud we also apply uniform noise to the points, in order to further simulate the Cepton Vista-P60 LiDAR. The final data set consists of 193000 people and

281000 other point clouds, these where then split into training and validation sets such that 90% of the data set is used for training and 10% is used for validation.

3.1.2 Real-World

To collect some real LiDAR data the LiDAR device described in the background chapter was set up in and indoors environment shown in figure 3.3

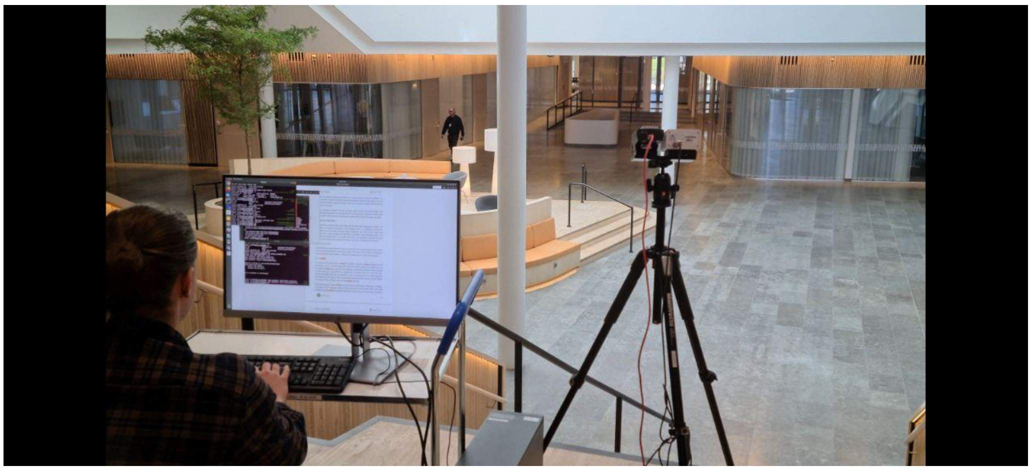


Figure 3.3: A picture of the indoor scene recorded with the LiDAR device (visible on the tripod)

We then had some adult males with slightly different physiques walk around the scene while we recorded them. The persons were instructed to sometimes act unpredictably, e.g., to walk with their hands above their head, pick up and carry a trashcan or having an unnaturally long stride. Due to the only dynamic objects being present in the scene being people and due to the lack of dynamic background points and noise a labeled data set consisting of our volunteers can be created simply by removing the background and clustering the remaining (dynamic) points.

3.2 Dynamic Object detection

In this section we will go more into detail about our implementations of the background filtering techniques discussed in the background chapter. We will also discuss how our chosen clustering is used to perform clustering in both 2 and 3 dimensions .

3.2.1 Background Filtration

Maximum angular distance Filter

We implement the method described in [4]. The maximum distance measured from a laser emitted at a specific horizontal and vertical angle over some initial frames is stored. We use this as our representation of the background. When we use this model to filter new data we compare for each recorded angle how far the current distance is from the maximum distance. If it is further away than some threshold we conclude that it must belong to the foreground, and therefore belong to a dynamic object. The background model is stored as a dictionary $dict$ with keys and values $dict((horizontal\_angle, vertical\_angle)) = max\_distance$

Nearest Neighbour

We also implement the other method described in [7]. We store between 10 and 30 past point clouds representing 1 to 3 seconds in the past. When we receive a new point cloud we compare it to the point cloud furthest in the past, and count the number of points in the old cloud within a threshold radius of the individual new points. Since the position of the static points remain constant over time, the points in the most recent LiDAR frame that are static will have a lot of neighbours in the past point cloud, while dynamic points will have relatively few. Using this we are able to separate the two sets by defining a threshold value for the maximum number of neighbouring points a dynamic object is allowed to have. In this application we chose 5 points within a 0.25m radius as the threshold value.

Density Filter

We also implement the method described in [3]. The scene is divided into cube sub-spaces, and during a pre-processing step we increment for each new point in a given cloud the density of the sub-space cube containing that point. Cubes belonging to the background will have a much higher density than cubes belonging to the foreground. Thus, by defining some threshold value we can separate the high density background from the low density foreground. The threshold value depend greatly on the size of the cubes, and if dynamic objects are present in the scene during the pre-processing step. Optimally, if there are no dynamic objects present in the scene during creating of the background model the threshold can be set to 0.

The background model is stored as a dictionary $dict$ with keys and values $dict((x, y, z)) = density$ where the (x, y, z) have been rounded to some decimal depending on the size of each sub-space.

Plane segment Filter

Open3D [36], the library we used to store representations of our point clouds has a built in RANSAC algorithm for ground plane detection that we decided to use instead of implementing our own. When we receive the first point cloud from the LiDAR we sample 3 points from it and define a plane in our 3D space from these points and calculate the inlier set. This process is repeated 1000 times and the plane resulting in the maximum inlier set is saved and used as a representation of the ground plane. For each future LiDAR frame, since we only consider a stationary LiDAR, this same plane can be used to filter out the ground plane without having to repeat the RANSAC algorithm. Each point within some short distance, in our case 0.1m, of the ground plane is considered as belonging to the background of a scene and therefore removed.

Point cloud distance Filter

Our implementation is an extended version of the compute point cloud distance method from open3D [36]. In open3D the point clouds are stored in a k-d-tree structure enabling nearest neighbour search with a time complexity of $O(logn)$ [31].

The goal of the background subtraction method is to detect the objects that move by comparing the difference between the source point cloud and a background model, where the background model consists of an average of preceding point clouds. Then for each point in the source point cloud it computes the distance to the closest point in the background model, and the mean of all these distances. If a point has a larger distance to its closest neighbour then the mean of all calculated distances, it is considered dynamic. Since static points will be very close to points in the background model, and dynamic will be far, the two sets can be separated.

In order to keep the background model updated the oldest point cloud is removed and replaced by a newer point cloud. This happens every five-hundred frame and keeps the filtering model able to adapt to scenarios where, for instance, a car parks in the middle of the frame for a longer period.

Test environment

To test the performance of the background filtration methods a scene was created in the Carla simulator. An example frame from this simulated scene is shown in figure 3.4.

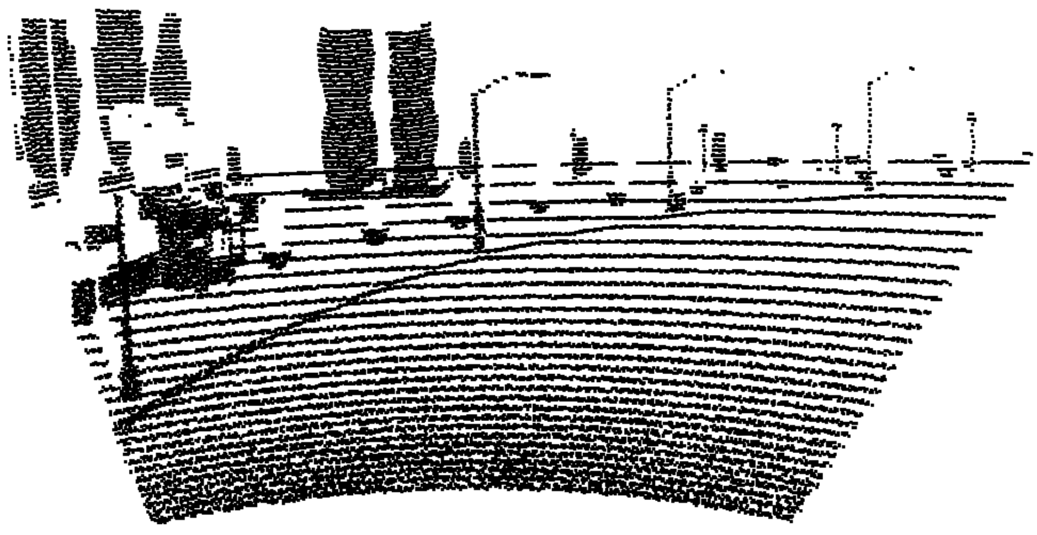


Figure 3.4: A frame from the Carla simulator point cloud used for testing the background filtration methods

The scene consists of roughly 5 minutes of LiDAR data where several pedestrians walk along a sidewalk next to an open square containing several structures, such as trees, plants, sculptures and street lights. The metric we have chosen to determine the performance of the filter is percentage of background points removed versus the number of pedestrian points removed. This is mainly because some of the methods use libraries that take advantage of lower level programming languages with better performance than our own implementations in Python, and therefore measuring and comparing the execution times for filtering a frame would be misleading. Our goal is therefore to maximize the number of background points removed while minimizing the amount of pedestrian points removed, enabling us to perform our clustering on the minimum amount of points while also minimizing the loss of information for the classifiers.

3.2.2 Clustering

Due to the curse of dimensionality [2] we are interested in measuring the performance difference between clustering on 3D data (x, y, z) and reduced 2D data (x, y). The background filtration techniques remove the ground plane, so projecting all points onto the xy plane will not cause any separate pedestrian or background clusters to merge. We imagine that utilizing this might reduce the execution time of the clustering without having significant effects on the accuracy of the clustering.

To visualise to the reader the difference between these two scenarios two examples of the different point sets are shown in figure 3.5.

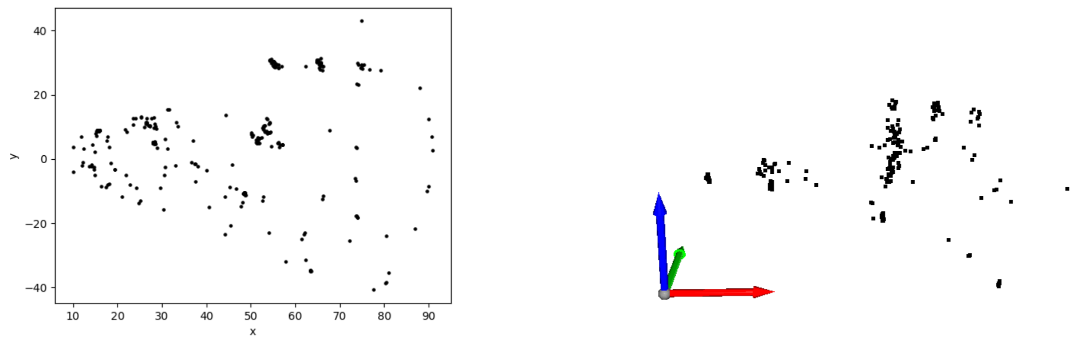


Figure 3.5: The filtered point cloud in 2D and 3D. In the right image the arrows show the orientation of the space in x (red), y (green) and z (blue) coordinates. This scene is the same as the unfiltered one shown in 3.4

3.3 Object Tracking

For the object tracking we have created a scene in Carla consisting of a single pedestrian walking and running around. The pedestrian moves erratically, often changing direction, stopping, accelerating and moving in unpredictable ways. We record the true position of the pedestrian as the mean of all points contained in the pedestrian cluster at each time step. We then compare the true position with the one estimated by our tracker. What is interesting to us here is the execution time of the state estimations in each of the filters, as well as the position tracking accuracy. The recorded data was collected at 10 frames per second, but in the context of minimizing the processing requirements for the system, it is also in our interest to see how the tracking performance deteriorates when the amount of frames per second is reduced.

The tracking process starts with receiving a new frame from the LiDAR. We use the techniques discussed in section Dynamic Object Detection to find objects within the scene, we will refer to the centroid of each object as an observations of the objects position. We refer to a currently tracked object as a track.

When the tracker receives a new observation it creates a new track and assigns the new observation to it. The track is assigned a state vector $s^k = (x, y, z, v_x, v_y, v_z)^k$ containing the position $(x, y, z)^k$ and the velocity in each of those directions $(v_x, v_y, v_z)^k$ at time k.

Each new time step we want to propagate our current tracks so we know where to look for them in the next LiDAR frame. To do this we propagate the position of each of our tracks by adding their velocity to their position. Observations from the new LiDAR frame are then matched to our current propagated tracks based on proximity. If a new observation is not matched to a current track we initialize a new track. If a track is not matched to a new observation we increment the decay of the track, and delete it from the tracker if the track has not been matched with an observation for some number of frames.

Since only observations of the position are sent to the tracker, states $(v_x, v_y, vz)^k$ can be naively estimated by checking the difference in position between frames $\frac{P^k - P^{k-1}}{\Delta t} = V^K$, but this will result in very noisy estimates since our position measurements also contain a lot of noise. To be able to accurately propagate our tracks using the velocity we need to employ tactics to improve the accuracy of our state estimate.

Therefore, if a track is matched to a new observation the observation is passed through a filter to reduce the noise in its state vector.

The tracking pipeline we created is shown in figure 3.6

3.3.1 Kalman Filter

To be able to accurately estimate the state we need to define a state transition model able to propagate the state in time.

Starting from the one dimensional case with state variables position **x**, velocity v_x, acceleration a_x, and time between frames as Δt we can define a motion model as

$$\begin{aligned} x^k &= x^{k-1} + v_x^{k-1}\Delta t + \frac{1}{2}a_x^{k-1}(\Delta t)^2 \\ v_x^k &= v_x^{k-1} + a_x^{k-1}\Delta t \\ a_x^k &= a_x^{k-1} \end{aligned} \tag{3.1}$$

In the three dimensional case the state vector that we are to estimate becomes

$$s^k = (x, y, z, v_x, v_y, v_z, a_x, a_y, a_z)^k \tag{3.2}$$

While the tracker is only really interested in the position and velocity of an object, the acceleration is necessary to include to be able to define a sufficiently accurate motion model.

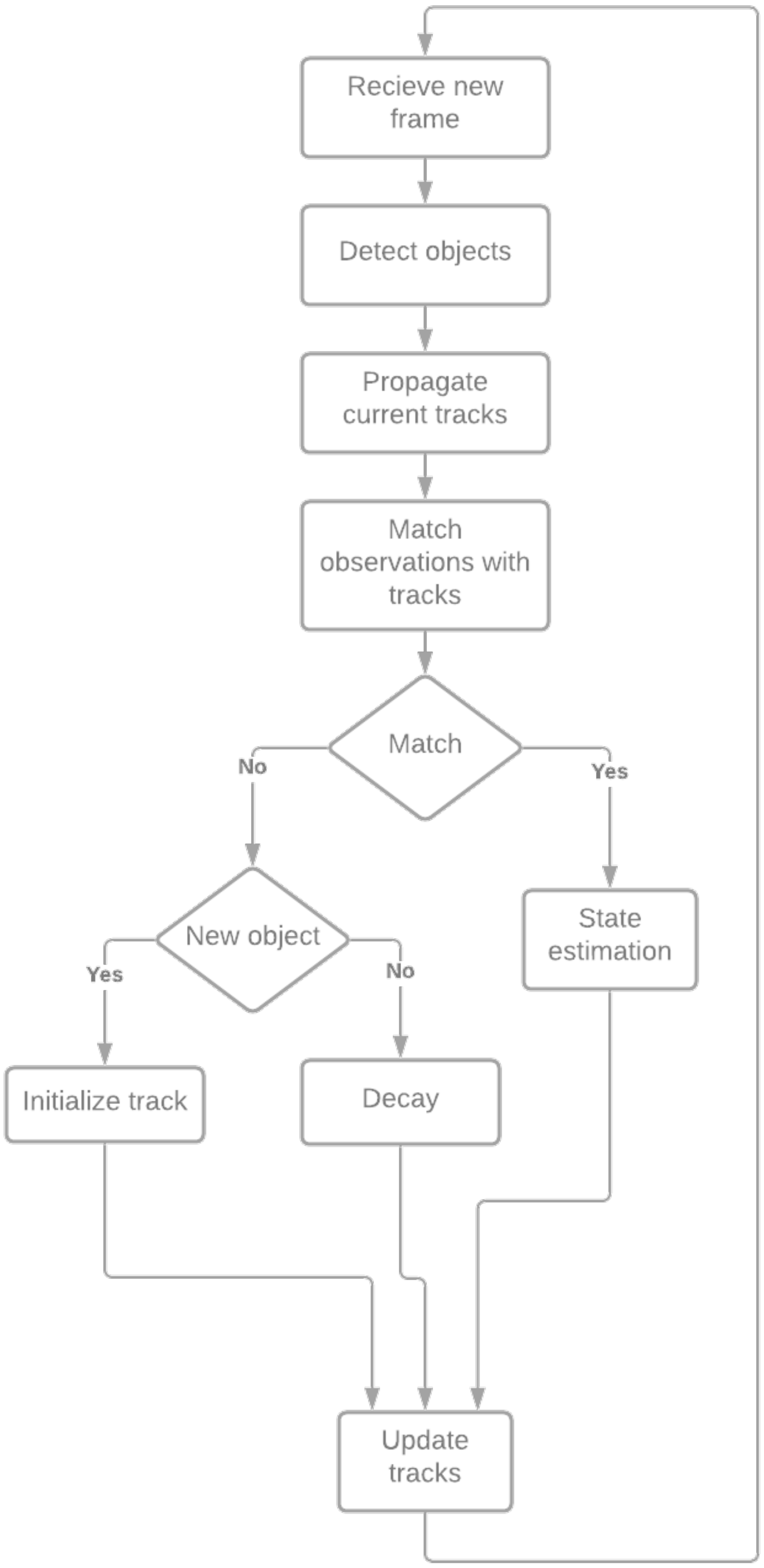


Figure 3.6: This figure is a flowchart describing the tracking process

From this motion model we define the state transiton model **F** to be

$$F = \begin{bmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 & \frac{1}{2}(\Delta t)^2 & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 & 0 & \frac{1}{2}(\Delta t)^2 & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t & 0 & 0 & \frac{1}{2}(\Delta t)^2 \\ 0 & 0 & 0 & 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \Delta t \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

This enables us to express to propagation of the state in time as

$$s^k = Fs^{k-1} + w \tag{3.4}$$

where w is the process noise defined as

$$p(w) \sim N(0, Q) \tag{3.5}$$

with Q being the covariance of the process noise. Here different states are assumed to be uncorrelated, and the values in the diagonal simply describe the expected variance of each state between time steps. We draw our variances from analysis performed in [37]. Here the average movement speed of a walking pedestrian is shown to be 1.4m/s and that it takes roughly 0.5 seconds for a pedestrian to accelerate to full walking speed. Therefore we assume that the variance of the position is $1.4(\Delta t)$ and the variance of the velocity is $\frac{1.4\Delta t}{0.5}$. When a pedestrian goes from standing still to a normal walking speed the pedestrian accelerates very sharply in the first few moments and therefore we have decided on a variance twice as large as the velocity variance.

$$Q = \begin{bmatrix} 1.4(\Delta t) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1.4(\Delta t) & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1.4(\Delta t) & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2.8(\Delta t) & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2.8(\Delta t) & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2.8(\Delta t) & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 5.6(\Delta t) & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 5.6(\Delta t) & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 5.6(\Delta t) \end{bmatrix} \tag{3.6}$$

The state of the Kalman filter is represented by two variables, our state vector s^k, and the matrix $\boldsymbol{P^k}$ describing the posteriori estimate covariance matrix, which can be interpreted as a measurement of belief in our estimates. We want to initialise the covariance to capture the assumed error between the intital state and the actual state. Since the initial position state is based on the position of the observation it can be assumed to be an accurate representation of the actual state, and we therefore set the variance to be 0.1, meaning that we assume that our initial position is within 0.1m in each axis of the actual position. The velocity is initialized as 0 in each axis and with an average pedestrian walking speed of 1.4m/s we set the velocity variance to be $(0-1.4)^2$ and due to the error in the acceleration being harder to define we set it to $(2(0-1.4))^2$ mainly due to observing good results with this choice.

In summary, a high value signifies a low belief in our initial state, while a low value signifies a high belief.

We define our initial $\boldsymbol{P}^0$ as

$$P^0 = \begin{bmatrix} 0.1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1.4^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1.4^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1.4^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2.8^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2.8^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2.8^2 \end{bmatrix} \tag{3.7}$$

For the Kalman Filter to work we need to be able to transform our current state estimate s^k into an observation. The observation o^k passed to the Kalman filter only contains the position of the object (x, y, z), and we need to define an observation model $\boldsymbol{H}$, able to reduce our current state to an observation.

We define $\boldsymbol{H}$ as

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.8}$$

Now we can retrieve an observation z^k from state s^k

$$z^k = H^k * s^k + v \tag{3.9}$$

where

$$p(v) \sim N(0, R) \tag{3.10}$$

and with $\boldsymbol{R}$ being the covariance matrix of the observation noise. Since we assume that our observations of the position are very accurate we give them a low variance.

$$R = \begin{bmatrix} 0.01 & 0 & 0 \\ 0 & 0.01 & 0 \\ 0 & 0 & 0.01 \end{bmatrix} \tag{3.11}$$

The exact co-variances and parameters were selected with a trial and error method and and further analysis would be necessary to actually conclude that our Kalman Filter performs optimally.

3.3.2 Particle filter

Each tracked object is assigned N particles, which are mutations of the current state where we have added some Gaussian noise to each of the individual state variables. The states are initialized in the same way as in the Kalman filter. We define this Gaussian noise as being either motion noise (*mn*), describing possible changes in the velocity, and position noise (*pn*), describing possible changes in position. In our implementation we choose $mn = 0.4$ and $pn = 0.1$ based on anecdotal evidence from experimentation.

When a new measurement is made, each particle is updated. We update each velocity

$$v_x = v_x + N(0, mn) \tag{3.12}$$

we also propagate each position state, and add the position noise

$$x = x + v_x/\Delta t + N(0, pn) \tag{3.13}$$

To determine a probability distribution describing the likelihood of each particle being the correct state, when we receive a new observation **o** we assign each propagated particle p_n a likelihood based on its euclidean distance from the observation

$$P(p_n|o) = (1/distance(p_n, o))/\sum^{N}(1/distance(p_n, o) \tag{3.14}$$

We then sample N particles from our current set of particles with replacement based on this probability distribution. This step is called importance re-sampling, and allows us to select the most likely hypothesis from our particle set.

We create our new state estimation from the mean of the re-sampled set.

Over several iterations, the particles produced by the filter will become more and more accurate, and our state estimation will converge to the true state.

3.4 Classification

To create an efficient decision process for deciding if a dynamic object is a pedestrian or other, e.g., a car, a cat or a flag, we define some initial assumptions about the properties we expect a pedestrian to have in the point cloud.

We define the feature points per distance (ppd) to be the number of points contained in an object divided by the distance from the LiDAR. Initial experiments tell us that for our LiDAR setup the cluster belonging to a grown man has roughly 150 points when standing at 10 meter distance, giving us a ppd of 1500. We establish that one of our conditions for an object to be put up for further consideration is that the ppd is within a range centered around 1500. A second condition is that the detected object must have a velocity above a certain threshold to be further considered.

The two classification methods we have decided to use both are in need of a consistent amount of features to be able to classify objects. Since the number of points in each detected object can vary greatly we need to be able to reduce the points into a consistent feature space. We do this by creating a voxelization of the point cluster. An example of such a voxelization can be seen in figure 3.7

For our two methods, we want to compare the prediction accuracy, as well as the execution time of the feature extraction and the prediction step.

The training data used for creating models consists of 50000 pedestrian voxelizations and 50000 other object voxelizations extracted from the Carla simulator.

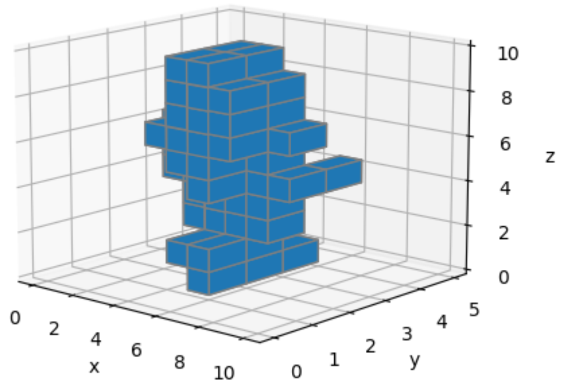


Figure 3.7: Figure showing a 10x5x10 voxelization of a pedestrian point cloud

3.4.1 SVM - Support-Vector Machine

The features we have decided to use consist of a subset of the features presented in [15] which are the width along the x axis and y axis, the height, points per distance and the voxelization. We have specifically selected only the most prominent features from their SVM to improve the execution time of the feature extraction step.

The point cloud is reduced to a voxelization of dimension 10x5x10. A voxel is either filled or empty. The 10x5x10 voxel matrix is then flattened to a 500x1 vector, which represent the first 500 features in our feature vector.

As additional features we also append the detected objects width in x and y dimension, as well as their height. We also finally add our previously defined feature ppd.

In summary, the feature vector x_i consists of 500 voxel values v_i, $width_x$, $width_y$, height, points per distance (ppd)

$$x_i = [v_1, ..., v_{500}, width_x, width_y, height, ppd] \tag{3.15}$$

The loss function used for our SVM is the hinge loss function described in chapter 2. The optimization method used to minimize the loss function is stochastic gradient descent [38].

3.4.2 CNN - Convolutional neural network

For our 3D CNN we are interested in testing the prediction accuracy and execution time of feature extraction and prediction on different resolution images. We do this in hope of being able to get determine a good trade off between execution time, information loss and prediction accuracy. The resolutions we have decided to test are 8x8x8, 16x16x16 and 24x24x24 voxelizations.

For every layer of our neural network we have chosen the Rectified Linear Unit (ReLU) as our activation function, the equation can be seen in 3.16 and figure 3.8 visualizes the function.

$$ReLU(x) = max(0, x) \tag{3.16}$$

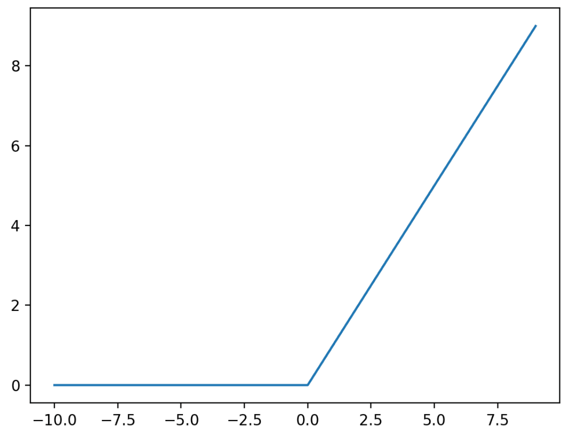


Figure 3.8: Figure showing the ReLU activation function

ReLU converts negative inputs to 0, which results in that only the most prominent feature maps are propagated through the network.

The feature maps output from the convolutional layers are very sensitive to the spatial location of the features in the input. One way to reduce the sensitivity is to use max pooling to down sample the feature map. By max pooling we extract the most relevant features from the feature map, reducing the importance of the spatial location of features within an image.

The final dense layer only has only one neuron and uses a sigmoid activation function.

$$S(x) = \frac{1}{1 + e^x} \tag{3.17}$$

A plot of the sigmoid function can be seen in figure 3.9

Thus, the output from the network will be a probability p between 0 and 1 where the probability of belonging to each class is $P(pedestrian) = p$ and $P(other) = 1 - p$.

The loss function we optimize over is binary crossentropy

$$\text{Loss} \ = -\frac{1}{\text{N}} \sum_{i=1}^{\text{N}} y_i \cdot \log \hat{y}_i + (1 - y_i) \cdot \log (1 - \hat{y}_i) \tag{3.18}$$

where y_i are the actual labels and $\hat{y}_i$ are the labels predicted by the network. We use the binary crossentropy loss function due having having only two classes, pedestrian and other.

The loss function is minimized using ADAM, a stochastic optimization algorithm [39].

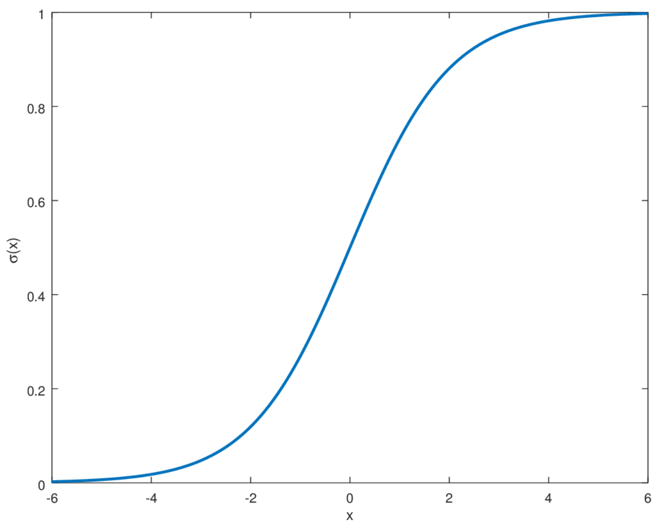


Figure 3.9: Figure showing the sigmoid activation function

We want to test the performance of the classification and feature extraction for different resolution voxelizations images. Some examples of the different resolution images can be seen in figure 3.10 and 3.11.

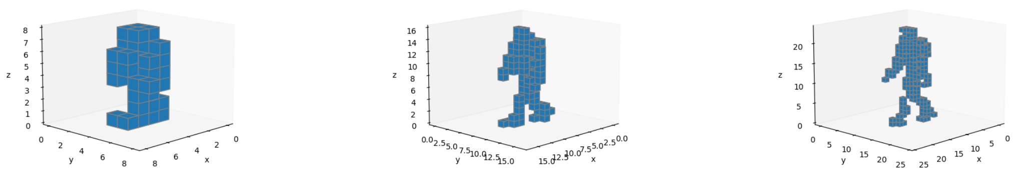


Figure 3.10: Example image of the same pedestrian for 8x8x8 voxels (left) and 16x16x16x voxels (middle) and 24x24x24 voxels (right)

The structure of our 3DCNN can be seen in figure 3.12.

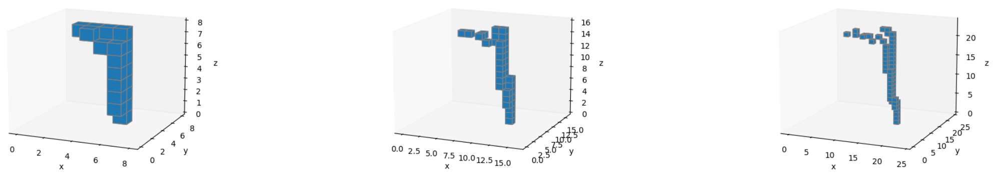


Figure 3.11: Example image of the same street light for 8x8x8 voxels (left) and 16x16x16x voxels (middle) and 24x24x24 voxels (right)

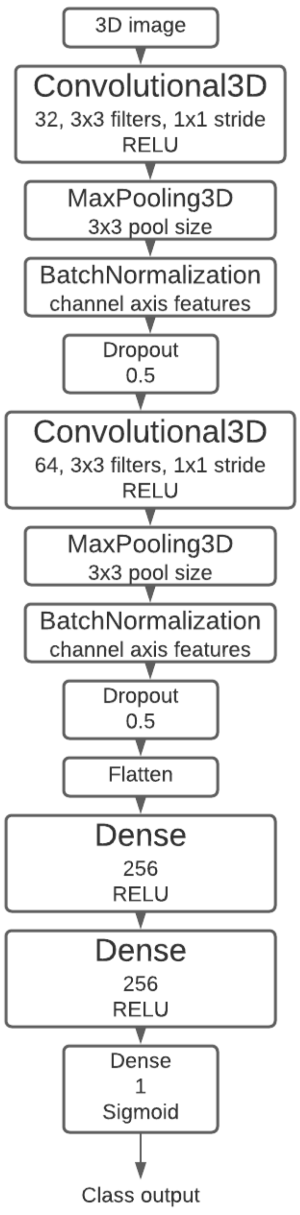


Figure 3.12: The structure of the 3DCNN

3.5 Testing on a limited system

As mentioned in the background chapter and discussed above one of the main focuses of this thesis is to investigate whether it is possible to perform pedestrian detection and tracking on a system with limited processing power. To do this we have acquired a Coral Dev Board [40], which runs the Mendel Linux distribution.

Since the Coral Dev Board has limited processing powers different approaches on lowering the workload of the system can be used. Since all LiDAR data discussed in this report has been recorded at 10fps a very simple method to reduce the computational requirements of the system is to simply reduce the frame rate. Halving the frame rate will halve the total processing cost of the entire system, speeding up the process significantly. As mentioned in the background chapter each frame of the LiDAR data consists of roughly 30 000 points which makes the filtration and the clustering steps very cumbersome, and we plan to investigate whether it is possible to discard many of these points randomly without significant deterioration of the tracking and classification process.

Chapter 4

Results

This chapter covers the results that were obtained for object detection, object tracking and classification.

4.1 Dynamic Object detection

4.1.1 Background Filtration

Maximum distance Filter

Unfortunately we were unable to get the maximum distance filter to work on simulated data as neither the vertical angle nor the distance to the measurement is present in the Carla point cloud data. Therefore the results here had to be extracted from unlabeled point cloud data acquired from the Cepton LiDAR. This means the performance metric displayed will only consider total points removed. The performance of the filter is shown in figure 4.1.

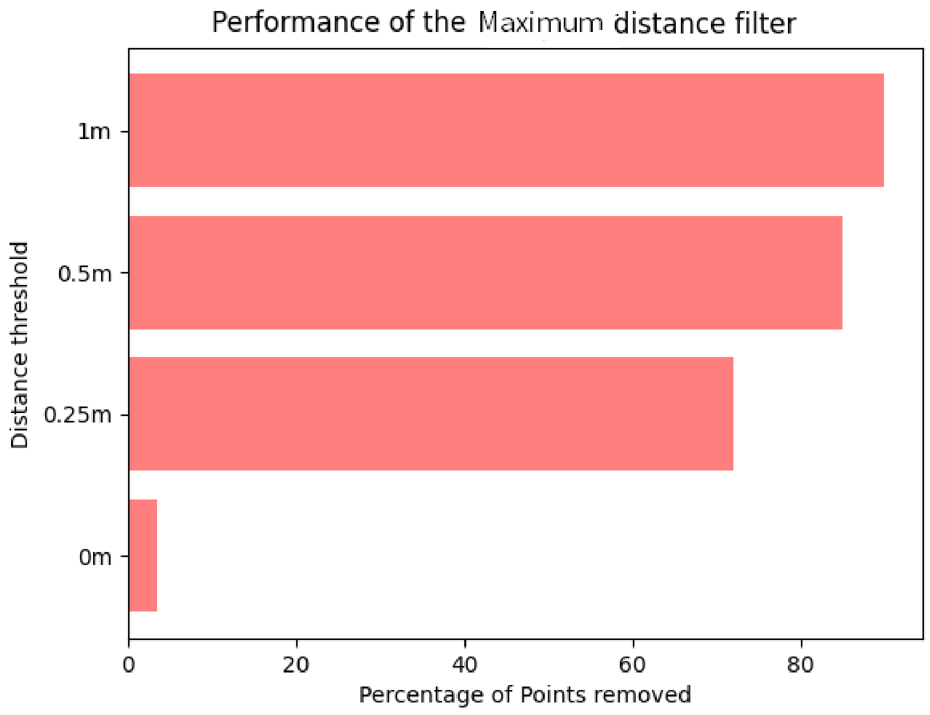


Figure 4.1: This figure shows the percentage of points removed from the data

Nearest Neighbour

The performance of the Nearest Neighbour filter is presented in 4.2. The search radius is the distance points can be separated by and still considered to be near. For each filtration we only looked at the closest 5 points for efficiency, and if all of them where within the search radius distance we labeled the point as belonging to the background.

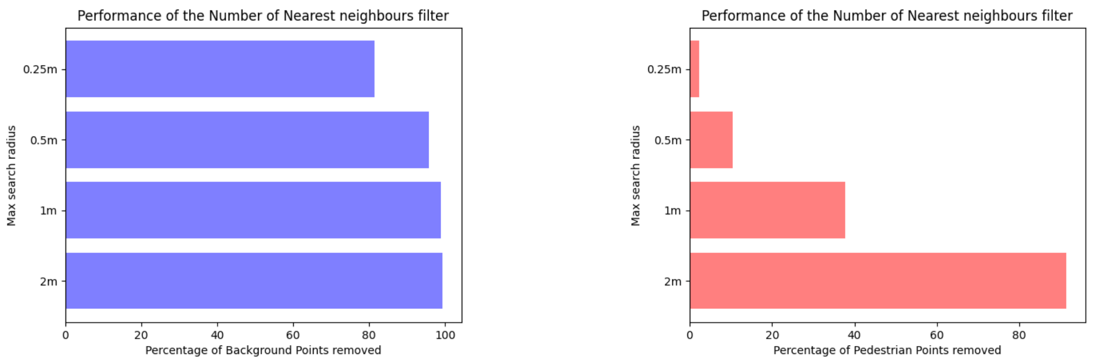


Figure 4.2: The percentage of background points vs pedestrian points removed for different search radius

Density filter

The performance of the Density Filter can be seen in 4.3. The cube size Cs parameter defines the side length of each cube such that their volume is $Cs * Cs * Cs$.

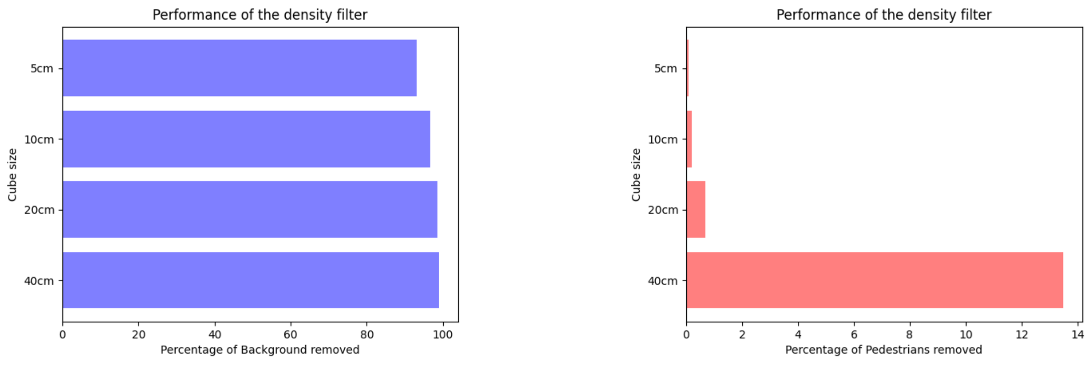


Figure 4.3: The percentage of background points vs pedestrian points removed for different cube side lengths

We can see that while having a larger cube size seems to remove more points overall, the smaller it is the more points we manage to keep on the pedestrians. The amount of background points removed isn't effected as much by the cube side, and good performance is achieved with all different sizes.

Point cloud distance filter

In figure 4.4 the result of the point cloud distance filter with different values of the *number of points* parameters can be seen. The *number of points* parameter refers to how many points that should be taken in to consideration when calculating the mean of the background model.

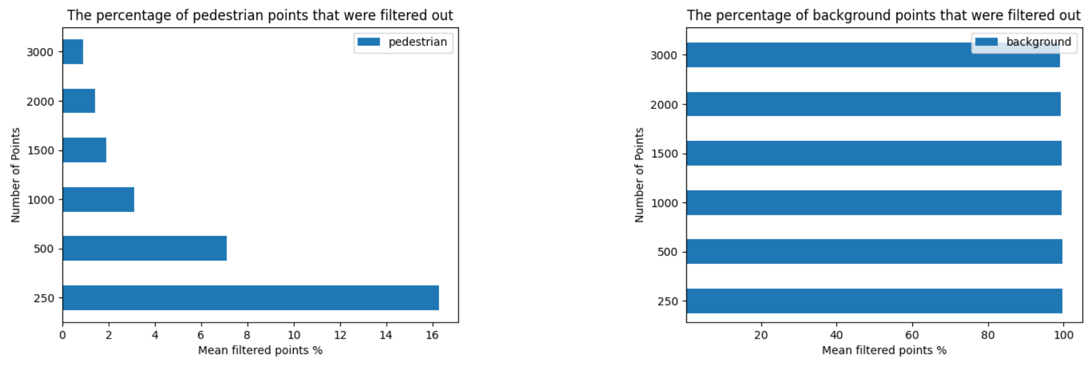


Figure 4.4: The percentage of pedestrian points (left) and the background points (right) that were filtered out with the point cloud distance filter with different values of the *number of points* parameter.

As can be seen in figure 4.4 the percentage of background points that were filtered out is almost identical which implies that the point cloud distance filter is good at detecting static points. When it comes to retain as much information as possible of a pedestrian a higher value of the *number of points* parameter is better. However, when a value of 3000 or higher is used, the percentage of background points that were filtered out starts to decrease rapidly. A decrease of 0.5% were observed between 2000 *number of points* and 3000 *number of points* compared to 0.1% between the other values and an even higher value of *number of points* decreases the percentage of background points that were filtered out even further.

Plane segment filter

In figure 4.5 we show the performance of the plane segment filter with different thresholds. The *threshold* distance parameter is the distance a point can be from the plane and still considered as being contained within it.

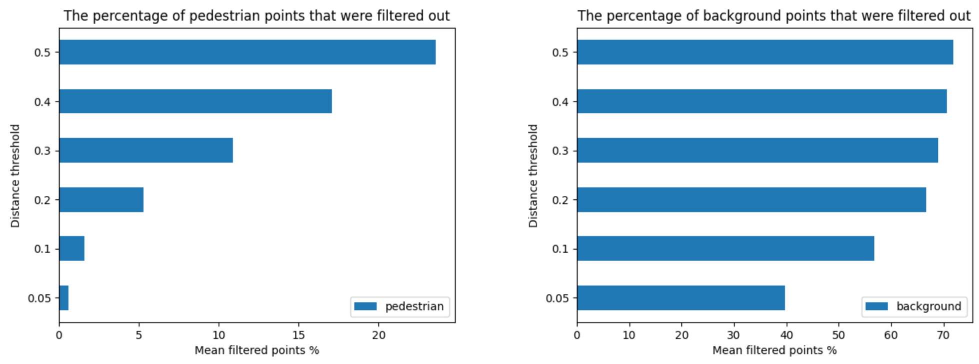


Figure 4.5: The percentage of pedestrian points (left) and the background points (right) that were filtered out with different distance thresholds

Having a large distance threshold removes the lower part of pedestrians walking on the ground plane. The better performance of the background removal with a high distance threshold is mainly due to objects, i.e., trashcans, trees and fences that are standing on the ground plane also have their bottoms removed.

4.1.2 Clustering

Using DBSCAN [9] works very well to cluster the entire point cloud in real time when it has been reduced by some filter, it is also very robust against noise in the data. Some problems we have noticed is that when a very large dynamic object, such as a truck driving close to the LiDAR, enters the scene the number of points in the filtered point cloud rises drastically and significantly slows down the clustering.

The average time for the DBSCAN clustering for the original 3D point cloud and the reduced dimension 2D point cloud are shown in figure 4.6. The individual execution time of the DBSCAN clustering for different sized point cloud is shown in figure 4.7. Not all point cloud sizes were encountered during the measurements, and as a result some of the results are taken from interpolating over the observed measurements.

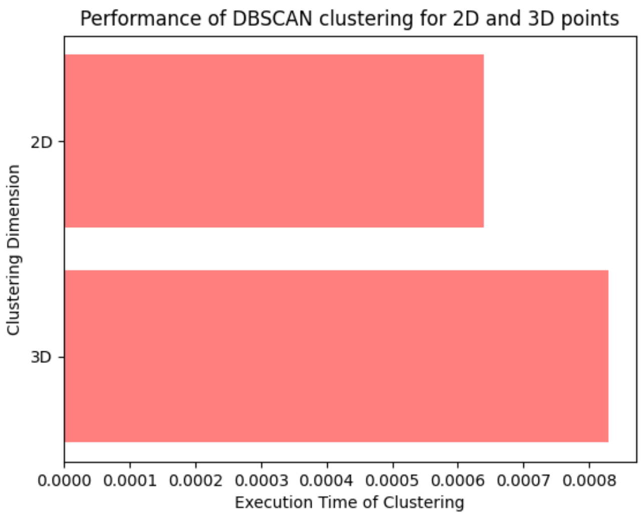


Figure 4.6: This figure shows the execution time of the clustering for 3D points and 2D points

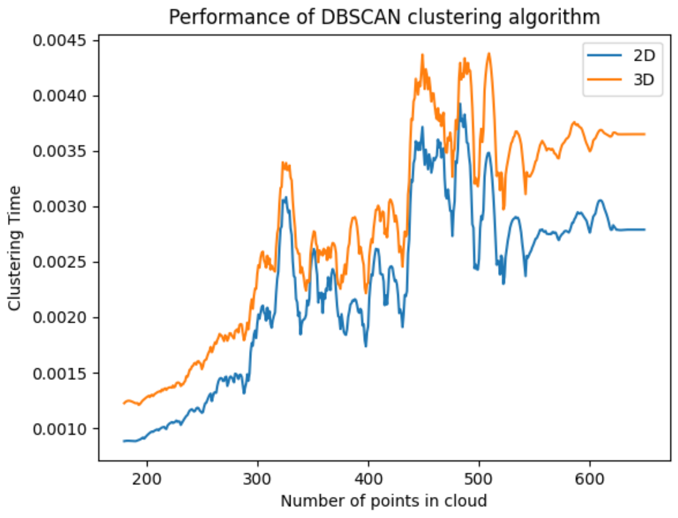


Figure 4.7: This figure shows the execution time of the clustering for 3D points and 2D points for different sized point clouds

The graph in figure 4.7 has been smoothed using a Savitsky-Golay filter [41] to make the performance more readable. It works by fitting a polynomial to a small moving window of the points using least squares with the intention of smoothing the graph. While one might expect that the graph would always be increasing over the x axis the performance of the DBSCAN algorithm is very dependant on the amount of noise and outliers in the data which sometimes causes the execution time to spike even for a low number of points. However, it is still observable that the 2D clustering performs better than the 3D variant, and that this performance difference grows with the number of points clustered upon.

We can see in figure 4.7 and 4.6 that while clustering in 2D is generally faster, it also squeezes sparse points along the z axis together and it more often labels background points from measurements on, e.g., trees as coming from a dynamic object. This phenomenon can be observed in figure 4.8 and 4.9

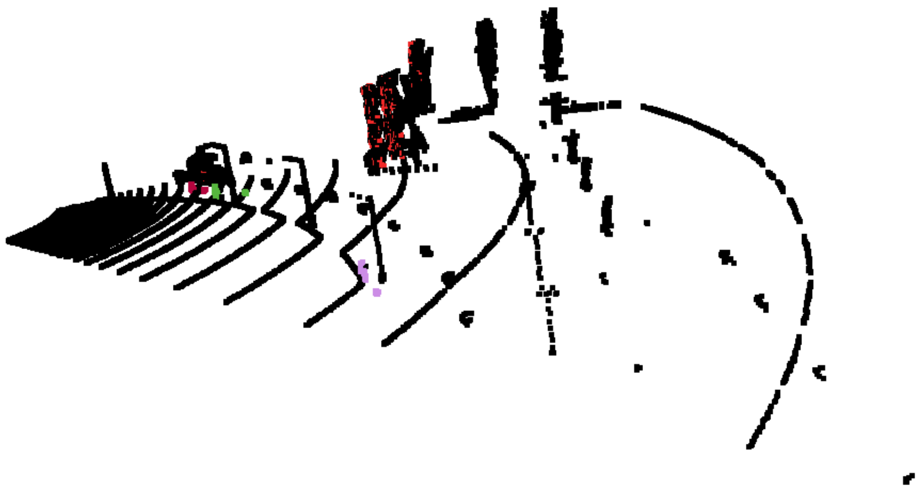


Figure 4.8: Images of the clusters DBSCAN found in the 2D point cloud. Colored points have been determined as belonging to a cluster.

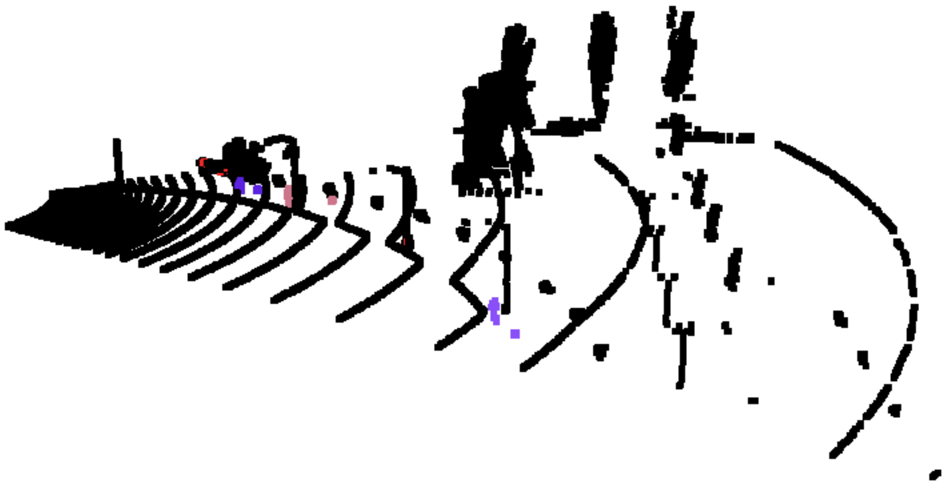


Figure 4.9: Images of the clusters DBSCAN found in the 3D point cloud. Colored points have been determined as belonging to a cluster.

4.2 Object Tracking

In figure 4.10 the execution time of the state estimation for the Kalman filter and the Particle filter with different numbers of particles are shown. In figure 4.11 we see the mean distance difference of the estimated position (x, y, z) by the tracker and true position (x_t, y_t, z_t). In figure 4.11 only the Particle filters with at least 50 particles were used, as the tracker eventually lost sight of the pedestrian when fewer particles were used.

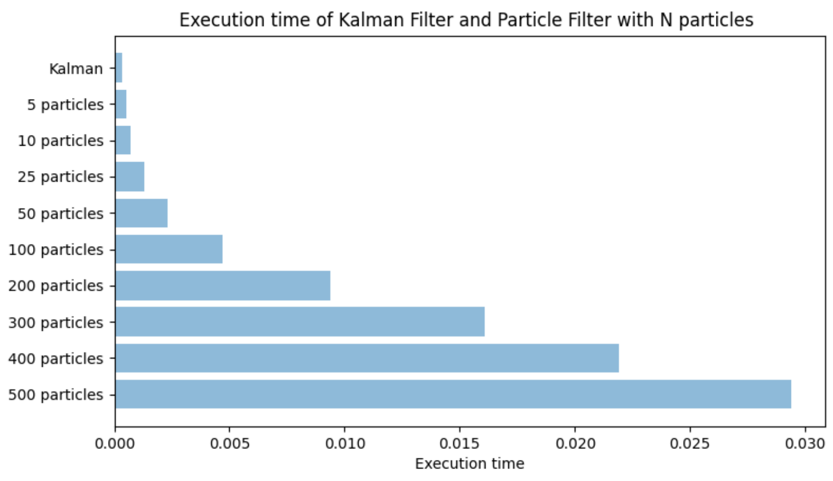


Figure 4.10: This figure shows the execution time of the update step of the Kalman filter and Particle filter with N particles

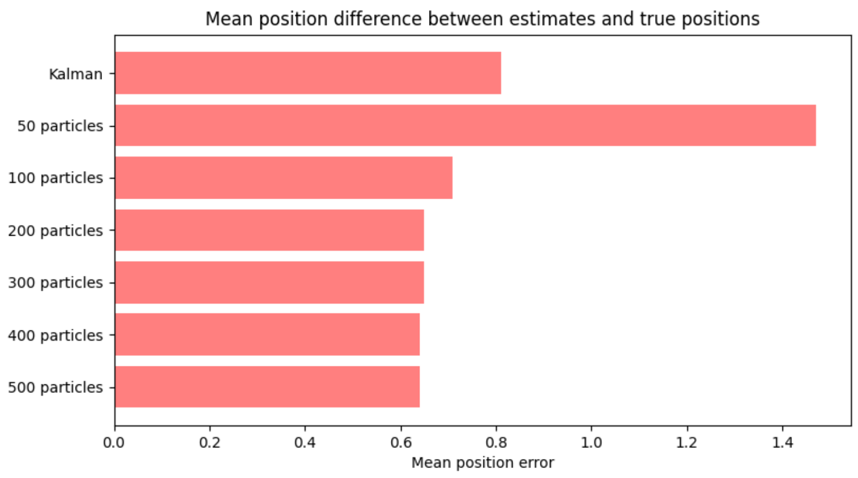


Figure 4.11: This figure show the mean position difference between estimates and true positions of the Kalman filter and Particle filter with N particles

As we can see in figure 4.11 the position estimate using the Particle filter did not improve significantly when more than 100 particles were used, and in our further comparisons we will only consider the Particle filter with 100 particles.

In figure 4.12, 4.13 and 4.14 we display the estimated and true position in (x,y) coordinates. The object first enters the scene at (19,-11) and the long straight line leading up to this is due to the object not being present in the first few frames and us then defining its position as (0,0). The tracker picks up the object some frames after it has entered the scene, which creates a bit of a lag between when the actual position and the estimated position is displayed.

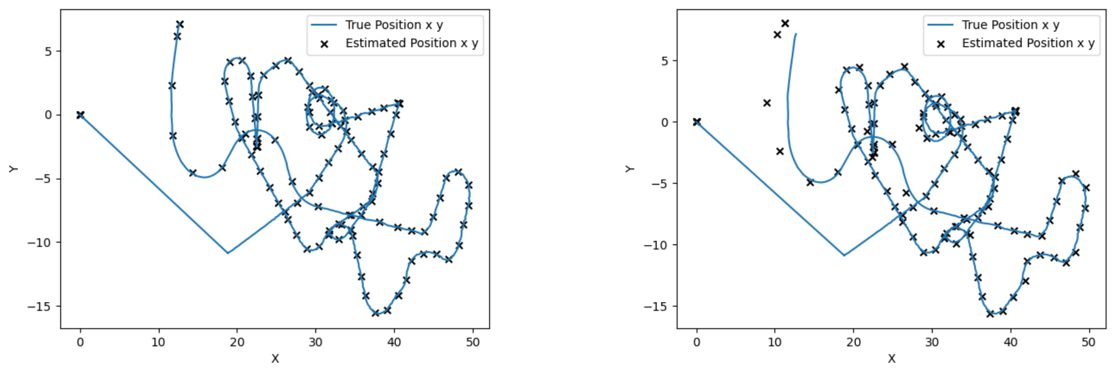


Figure 4.12: The true position and the estimated position using the Kalman filter (left) and the Particle filter with 100 particles (right) at 10fps. The estimated position was sampled once per second

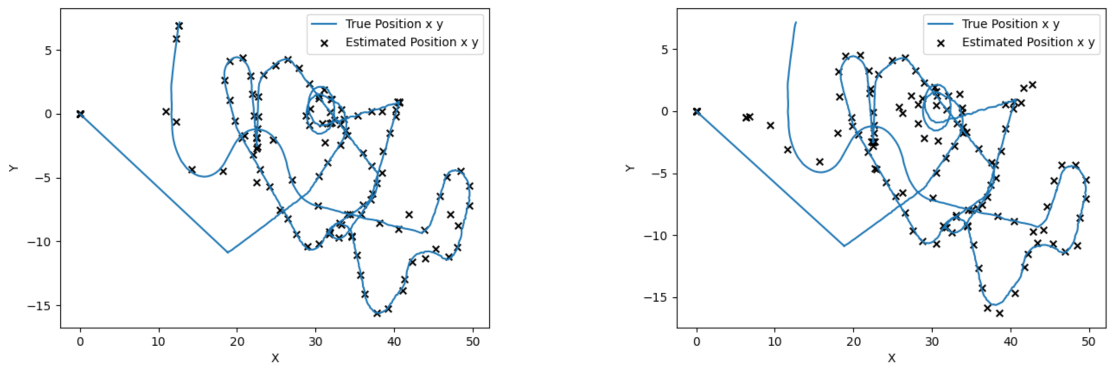


Figure 4.13: The true position and the estimated position using the Kalman filter (left) and the Particle filter with 100 particles (right) at 5fps. The estimated position was sampled once per second

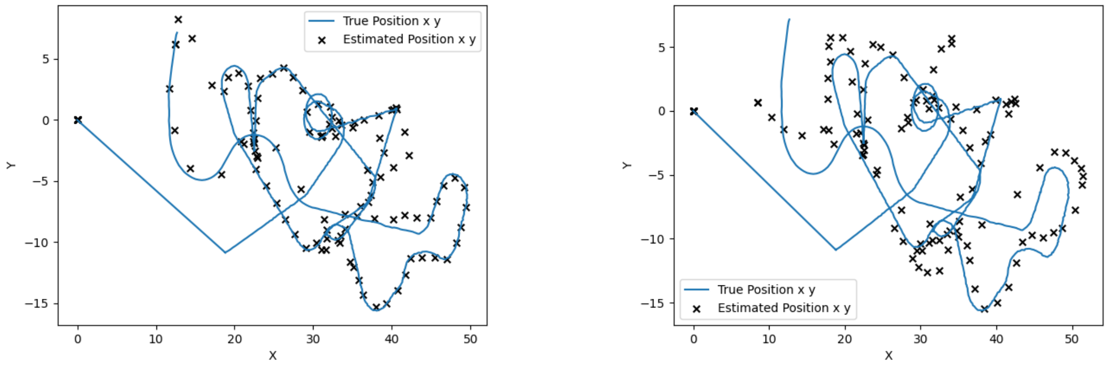


Figure 4.14: The true position and the estimated position using the Kalman filter (left) and the Particle filter with 100 particles (right) at 2fps. The estimated position was sampled once per second

Figure 4.15 shows the mean euclidean distance between the true position and the estimate calculated over all frames in the test scene.

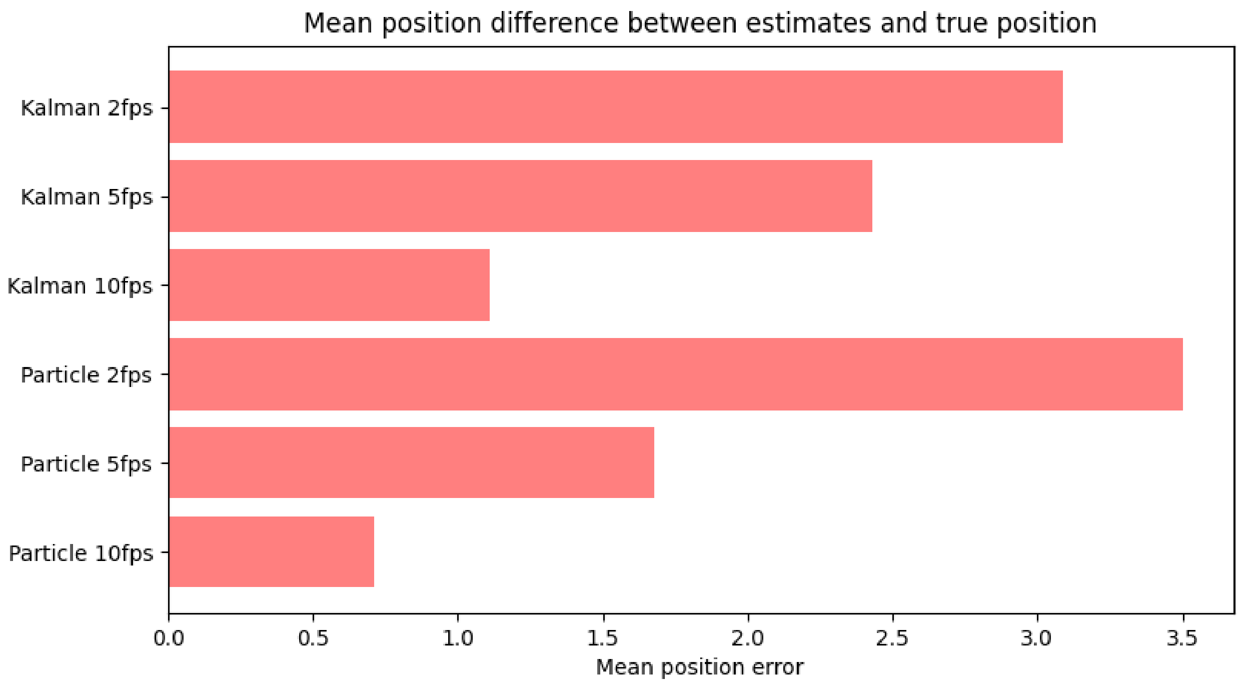


Figure 4.15: This figure show the mean distance difference between our estimate and the true positon for different fps

4.3 Classification

The results presented here will be the properties of the trained model and networks, as well as their classification accuracy and speed. The two classifiers tested were the SVM and 3DCNN described in chapter 3.4. The 3DCNN designed to work on 8x8x8 voxelizations is named cnn_8, and the other resolution 3DCNN's in a similar fashion. In figure 4.16 we show the classification accuracy acquired on the simulated test data set, and in figure 4.17 we show the classification accuracy on the real life data set described in the chapter 3.1. The difference in the classification accuracy between the simulated data and the real-life data are due to the two different models different ability adapt to data that it has not been trained on with

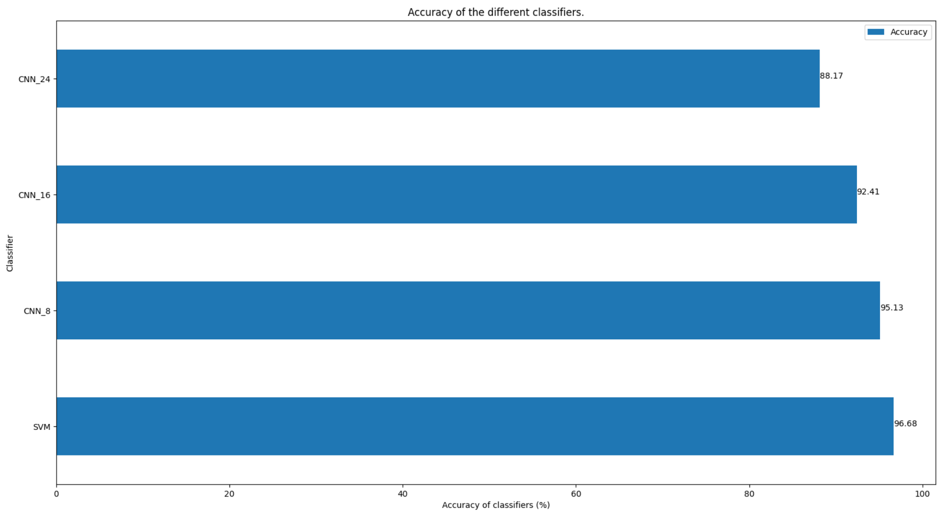


Figure 4.16: This graph shows the accuracy of the different classifiers.

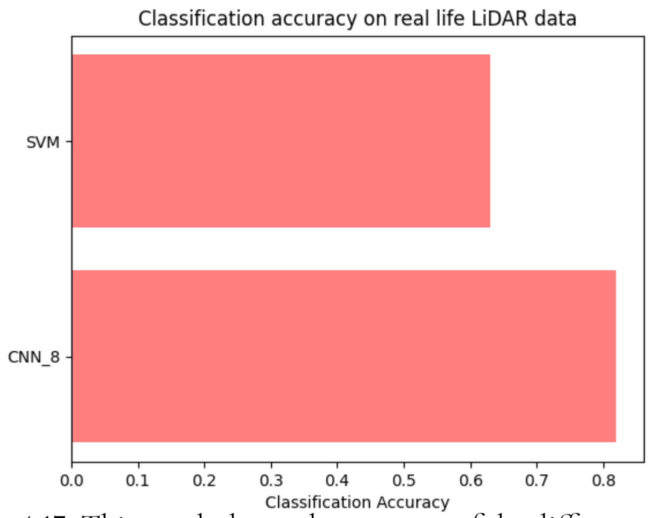


Figure 4.17: This graph shows the accuracy of the different classifiers on real life LiDAR data.

In figure 4.18 we display the time it takes to extract the features from an object used in the different classifier methods.

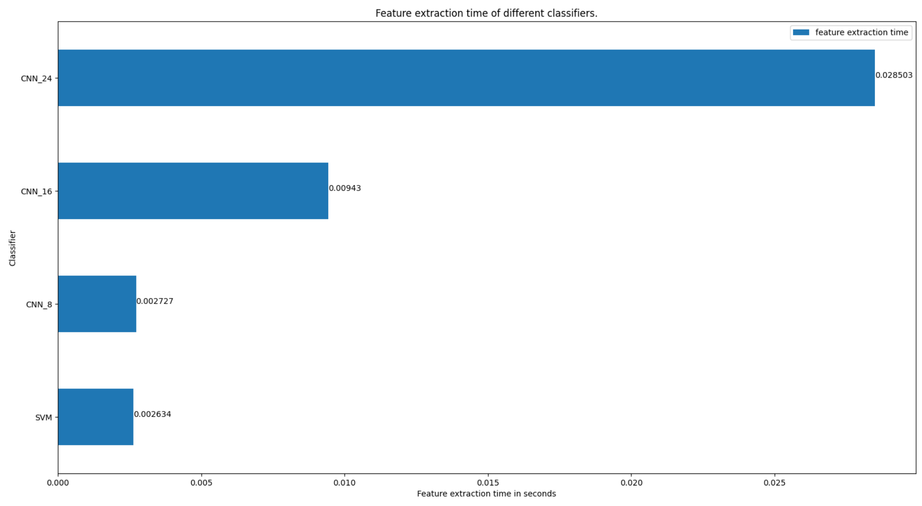


Figure 4.18: This graph shows the time it takes to extract the features from one object.

The features extracted are the height, width etc., as well as the different resolution voxelizations required by the different classifiers. Since the number of features grow rapidly with the increasing resolution, both cnn_24, and cnn_16 have significantly longer execution time than cnn_8 and the SVM.

In figure 4.19 we display the time it takes for our different classifiers to predict the class belonging of an object.

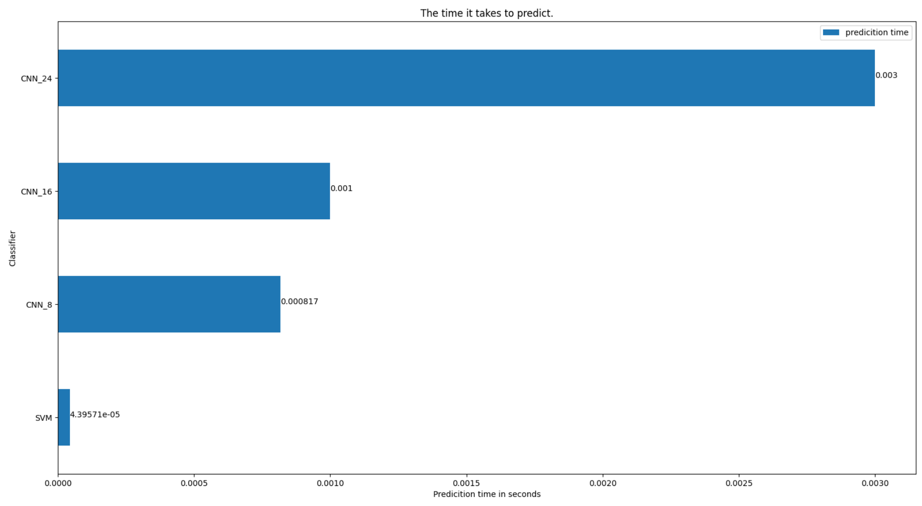


Figure 4.19: This graph shows the time it takes to predict one object with the different classifiers.

We can see that the predict time differs significantly depending on the resolution of the image. Although there is a significant difference between the SVM and the fastest 3DCNN (cnn_8), the predict time is very fast relative to other time costs in the pipeline.

4.3.1 SVM - Support-Vector Machine

The significance of the learned feature weights for our SVM is presented in figure 4.20.

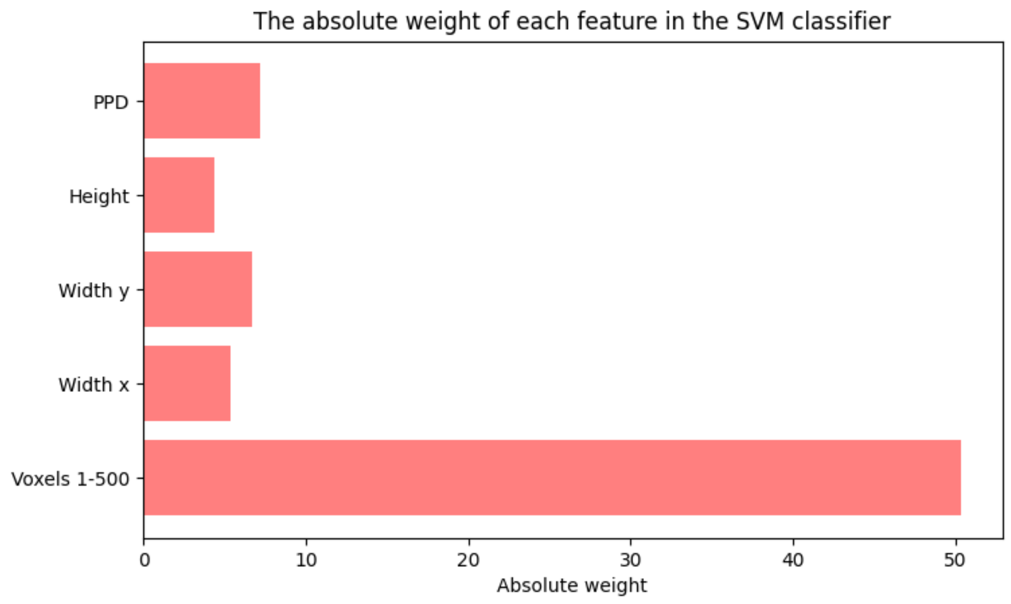


Figure 4.20: This figure show the significance of different features in our SVM implementation

The training and validation accuracy are displayed in figure 4.21.

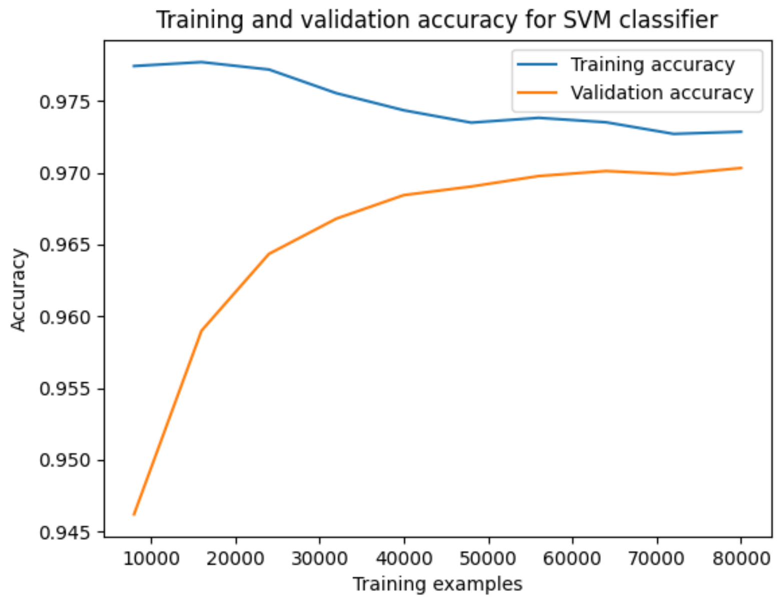


Figure 4.21: This figure show accuracy on the training and validation data set for the SVM classifier

The training accuracy over the training examples deteriorates since larger data sets are harder to fully capture with a simple model. However, the larger training set increases the validation accuracy

4.3.2 CNN - Convolutional neural network

In figure 4.22, 4.23 and 4.24 we show the epoch loss and epoch accuracy for the different resolution 3DCNN.

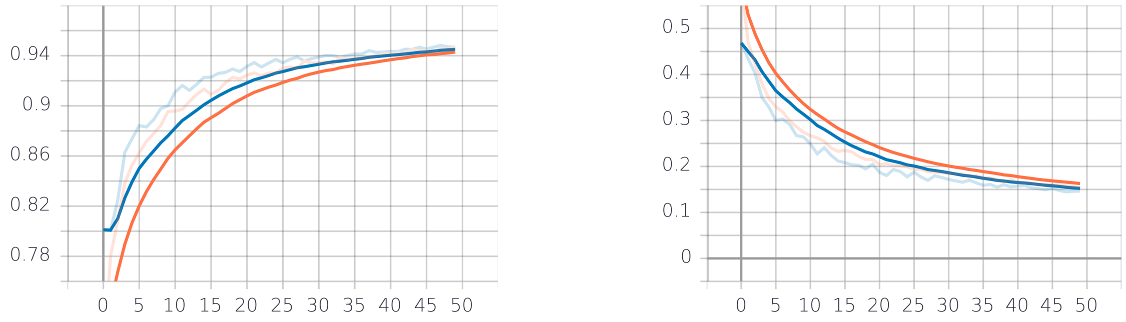


Figure 4.22: The epoch accuracy (left) and epoch loss (right) of the 8-voxel CNN classifier. The orange curve is the training accuracy/loss and the blue curve is the validation accuracy/loss.

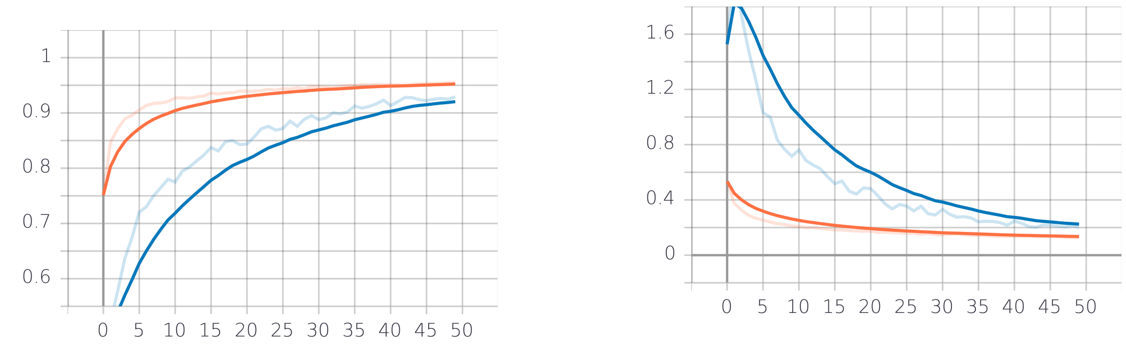


Figure 4.23: The epoch accuracy (left) and epoch loss (right) of the 16-voxel CNN classifier. The orange curve is the training accuracy/loss and the blue curve is the validation accuracy/loss.

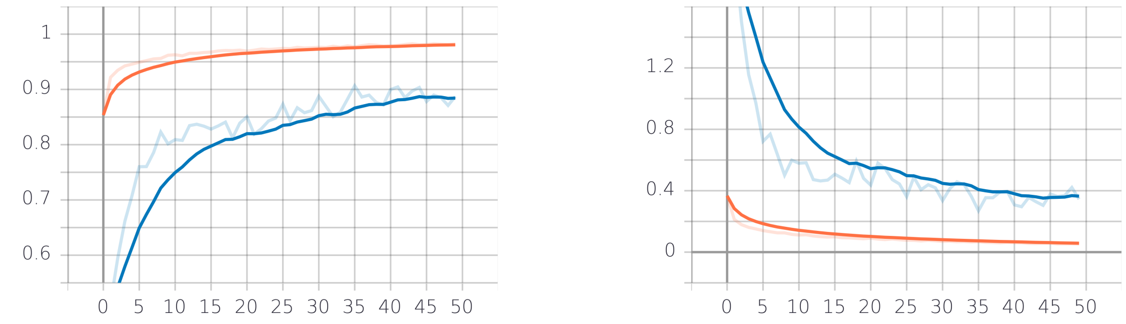


Figure 4.24: The epoch accuracy (left) and epoch loss (right) of the 24-voxel CNN classifier. The orange curve is the training accuracy/loss and the blue curve is the validation accuracy/loss.

As can be seen in figure 4.22, 4.23 and 4.24 the 8-voxel CNN classifier is able to achieve a validation accuracy of around 94% and a validation loss of around 0.15 in 50 epochs and the results of the voxel classifiers get worse in ascending order. The results indicates that in our case the training of a classifier on voxels with higher resolution yields worse results compared to the 8-voxel CNN classifier.

4.3.3 Testing on a limited system

The results shown in 4.1 display the percentage of points that need to be discarded before the point cloud is fed through the pipeline to achieve real time execution. A flat surface means that there is a ground flat plane present in the scene, and curved surface means that there is no flat ground plane present in the scene.

FPS	Points removed	Surface
10	80%	Curved
5	55%	Curved
2	0%	Curved
10	65%	Flat
5	35%	Flat
2	0%	Flat

Table 4.1: The percentage of points needed to be removed to achieve real time execution of the LiDAR tracking pipeline for curved and flat surfaces.

The difference between the results for the curved surfaces and the flat surfaces result from the ability to remove many points efficiently using the plane segment filter if there is a ground plane present in the scene, significantly speeding up the filtration step.

Chapter 5
Discussion

5.1 Data Gathering

While using Carla enabled us to create a huge data set containing point cloud data of our two classes, pedestrian and other, the pedestrian models do not fully capture the range of human shape and motion present in the real world. While Carla has both female and male, adult and child models, it does not provide examples of pedestrians that would be common to see in a real world scenario, e.g., someone carrying a large suitcase, walking with a limp, carrying a small child, using a wheelchair or simply a pedestrian being severely overweight. Also, since each pedestrian was extracted based on an object ID present in the Carla simulator, the data set does not contain any pedestrians that have been clustered together as one object, something that would be common in a crowded real life LiDAR scene. The pedestrians does not display any other behaviour than walking, different behaviors, such as sitting on a bench, reading a book, bending over to tie their shoes or leaning against a wall would more closely represent a real life scenario.

Currently the pedestrians possess a limited amount of animations, such as walking forward, running forward and jumping. These animations are very repetitive and make the simulation of a pedestrian walking around in an urban environment seem unnatural. So a real case scenario where pedestrians interact with one another, avoid collisions and behave more like a human and less like a robot is hard to simulate with the help of Carla. Also, the absence of different dynamic objects, such as animals in the Carla simulator makes it hard to obtain measurements from objects that look similar to humans. Therefore, a more realistic simulation would be desired in order to create a data set that looks more similar to real world data. In order to make the simulations more realistic a plethora of dynamic objects with different shapes, sizes and animations would need to be added to the simulation. This would ease the classification of other dynamic object classes and also make the classifier more robust to objects that look similar to a pedestrian. Since the Carla simulator is built using Unreal engine adding a more diverse range of pedestrians is possible, as well as adding custom animation to these but we have decided that this is outside the scope of this thesis.

The great advantage with using a simulated environment is that the data gathering can be automated, but the automated process also creates some problems. As stated in chapter 3.1 the final data set we managed to extract from the Carla simulator consisted of 193 000 pedestrians and 281 000 other objects, creating a data set containing 474 000 individual objects. The size of the data set makes it very hard to confirm whether there are strange data entries that do not accurately enough represent one of the two classes, e.g., only a single or a few points of an object were observed and labeled as one of the classes.

Our initial detection, tracking and classification pipeline, with the classifier trained on simulated data, is still very able to detect pedestrians that move predictably in real life LiDAR scenes. If an object behaves predictably when it first enters the scene it will be classified as a pedestrian, and since an object is tracked while it is present in the LiDAR scene, even if it starts behaving unpredictably, the objects can have their point clouds extracted to create a more extensive data set containing these movements and orientations of limbs not present in the Carla simulator. We believe that to create the best data set to train the classifiers on, we would use our pipeline to extract pedestrians images from some real world scenes, and when enough data has been acquired the classifiers could be retrained on the real life data set for better performance.

5.2 Dynamic object detection

5.2.1 Background Filtration

The performance of the different background filters can be seen in the results chapter. In this section we will discuss the individual performance of the background filters, highlighting advantages and disadvantages with each of them.

The maximum distance filter managed to perform well when the distance threshold used to separate the background from the foreground was high, filtering roughly 90% of points from the cloud when the threshold was set to 1m. Having a high distance threshold will result in points belonging to dynamic objects that are close to the background, e.g., a person leaning against a wall, being removed. The necessity for a large threshold to achieve comparable performance to the other filters is in part due to the angular noise, caused by vibrations in the device, present in the data, as well as the angles present in the LiDAR being continuous. This causes a need to discretize the angular data with such precision that a limited amount of measured angles are grouped together, as well as making sure that the measured angle pairs during the pre-processing are actually encountered again when performing the filtering. While it may be possible to fine tune the precision to acquire better results than shown in this report, our different attempts always resulted in one of the two problems mentioned above. To create the background model consisting of angular pairs and distances the LiDAR needs to record data from the scene during a pre-processing step. The time it takes to create an accurate enough background model depends on the scene itself and the moving objects present in it, e.g., if the scene consists of a busy road with a sidewalk next to it the background model will take longer to be created with sufficient accuracy than if the scene contains no dynamic objects. This is due to the actual maximum distance for some angle pair will take longer to be encountered since a dynamic object will often block the static background for some time. Another problem is if the background changes, e.g., when monitoring a parking lot some of the vehicles present in the scene drive away, the background model needs to be recomputed to take the changing background into account.

The nearest neighbour filter also manages to perform very well, filtering out the majority of the background while leaving most dynamic objects untouched. A similar problem to the one mentioned above in the maximum distance filter is if a pedestrian moves close to a static object their number of neighbours will increase drastically and due to this being the metric used to separate the background from the foreground they will often be classified as belonging to the background. Since the number of close neighbours in the current point cloud is counted using a point cloud measured some time interval before, and this past point cloud continuously being updated so that the temporal difference between the current point cloud and the past remains the same, the background model is able to handle changing backgrounds, e.g., the parking lot example mentioned above. However, this also results in a dynamic object entering the scene and remaining stationary for some time eventually being considered part of the background and filtered out.

The density filter is one of the best ones we have implemented, and it comes with several advantages over many of the other filters. The main advantage is the ability to adapt to different scenes depending on the background by changing the dimensions of the sub spaces. If the background is mostly consisting of static points, using a small cube size enables us to define the background with a very high precision and to accurately capture small or thin static objects, e.g., poles, cones and fences without including too much of the space surrounding these objects. If a larger dimension subspace is used it instead becomes very robust against noise and dynamic background points, e.g., trees, flags and bushes, by including a larger space around these objects in the background, enabling them to sway in the wind without leaving the defined background space. Very small vibrations and changes in the LiDARs position can create large changes in the position of the points measured far away from the LiDAR, but these effects can also generally be mitigated by using larger sub spaces since it enables the background to shift slightly and still be captured by the background model. The time needed to create the background model is effected by the scene it is supposed to capture, needing more time if the scene contains a lot of moving objects, and less if the scene is empty at the beginning of the computation of the background model. In a similar way as the maximum distance filter it is not currently able to handle changing backgrounds, and if the background changes, the background model needs to be re-computed.

The point cloud distance filter works extremely well, and is able to filter out the majority of the background without a long pre-processing step. This is because the background model consists only of some past point clouds, enabling the background model to be created almost instantaneously from the first few frames acquired from the LiDAR. It is able to capture small background objects with high precision, as well as very successfully handle dynamic background objects present in the scene. Since new point clouds are continuously acquired by the LiDAR it becomes very easy to update the background model when changes occur in the scene.

One drawback of this method is that it becomes computationally heavy when using a large amount of preceding point clouds for the background model. Also, when the threshold parameter is not tuned correctly, the method tends to filter out too much of the foreground points. For instance, the legs of a pedestrian might be considered as background, since the distance between their legs and the ground is smaller than the threshold.

The plane segment filter works well if there is a constant ground plane present in the scene. Since the ground plane does not change significantly between frames of the LiDAR, the first frame acquired by the device can be used to detect the plane with very high precision, and the resulting plane can be used to filter the ground plane in following frames.

The advantages of the first 4 methods mentioned are that they also work well for scenes containing curved surfaces and elevation differences.

If there is a constant ground plane present in the scene we highly recommend first using the plane segment filter, since it is very efficient and the majority of points are usually contained within the ground plane, and then using one of the other filters to remove the points not contained in the ground plane.

Each filtration step after the first become less and less costly since fewer points are considered, and it is possible to use several filters in sequence without significantly increasing the execution time of the filtration step.

5.2.2 Clustering

The results displayed in figure 4.7 show that there is a slight performance increase when reducing the points located in the LiDAR data into two dimensions. While the performance between the two methods are more similar for a smaller amount of points the difference grows with the size of the point cloud.

As also displayed in 4.6 and 4.7 the execution time of the clustering is one of the more computationally heavy aspects of the pipeline, and in cases where a large number of points remain after the filtration there might be a significant advantage to reduce the points to two dimensions. The reduced dimensionality also comes with some slight disadvantages, since measurements separated by a large distance on the Z axis would be pressed closely together in the XY plane. The results of this can mainly be observed by measurements coming from, e.g., trees or pillars being pressed closer together and therefore also being clustered together and considered an observation of a dynamic object when the opposite is true. This problem is generally avoided when clustering in a higher dimension, with the price of increased execution time. It is possible that this problem could be avoided altogether by creating better performing background filters more suited for dealing with dynamic background objects.

A problem with both methods is that pedestrians walking in close proximity to one another will be clustered together as a single object. Since the observation passed to the tracker refer to a position based on the center of the clusters it will be skewed significantly from two objects being clustered together and the tracker receiving only one observation for two separate tracked objects. Since DBSCAN clusters points together based on proximity lowering the threshold for what is considered close could separate objects previously clustered together. A drawback of lowering the threshold is that DBSCAN becomes unable to cluster points coming from dynamic objects located far away from the LiDAR sensor, since the point cloud becomes more sparse the further away an object is from the LiDAR device. One solution to this could be to further consider clusters that have a large amount of points in ratio to their distance from the LiDAR device, clustering again on these with a lower threshold value. It is also important to note that the distances between points shrink significantly when the third dimension is removed, and fine tuning the parameters of the DBSCAN algorithm to take this into consideration is also important.

Rather than declare that one of the methods is better than the other, we would like to suggest two different use cases where we imagine that the different methods would perform better than the other. In a scene where many dynamic objects are present, e.g., a train station or a heavily trafficked street where the majority of background points are from static objects, e.g., buildings, it should be advantageous to use the 2D clustering since background models are more able to fully remove the background points. In another scene where a lot of dynamic background objects, e.g., trees or bushes are present, but we do not expect a large amount of fully dynamic objects to appear it would be advantageous to use the 3D clustering method.

5.3 Object tracking

While we have managed to successfully achieve multi target tracking, the performance of the object tracking scales badly with the number of observations and current tracks in each LiDAR frame. The distance matrix defining the distance between each of the n tracks and m observations is of size n*m, and this matrix needs to be fully searched n times to correctly match each of the tracks to an observation. This gives the tracker a time complexity of $O(n^2 * m)$ in relation to the number of tracks and observations. This could in turn result in severe performance deterioration when the LiDAR is monitoring a scene where many dynamic objects are present, e.g., a train station or a heavily trafficked street.

Due to this performance issue it is very important to have a sound and well defined decision process of when to add tracks and when to discard them, since the performance is very much dependant on the number of tracks currently in the tracker.

In our implementation any observation that is not matched to a current track is initialized as a new one and added to the tracker. A drawback of adding any new observation to the tracker is that static objects not of interest that suddenly move due to strong winds, vibrations, etc., will be added to the tracker. We imagine that adding a more sophisticated decision process of when a track should be added could improve the performance of the tracker, since non-interesting objects would not be considered in the first case. In the pedestrian tracking case this decision process could involve some easily extracted features from observations, such as their dimensions and number of points.

The decay parameter discussed in the method section decides when a track not matched to an observation for some time should be discarded. Having a low decay threshold enables the tracker to quickly discard tracks from dynamic objects that have left the scene, but will also cause the tracker to discard tracks that are still present within the scene but have been entirely obscured for a short amount of time from, e.g., walking behind a car or another background object. This need to be taken in to account when deciding the correct decay threshold and different thresholds might be best for different scenes based on information about the background objects and expected amount of dynamic objects.

As mentioned several times in this thesis, the estimation of the velocity and position of each observation is of out-most importance to accurately map tracks to observations, especially in crowded scenes where several dynamic objects move in close proximity to one another. When two pedestrians walk close to each other we often receive only a single observation from the dynamic object detection as these pedestrians have been clustered together as a single object. Determining the direction of movement for each pedestrian then helps accurately map the two separate tracks to the correct observations when the merged cluster again separates into two separate objects. It is also necessary to update the position of tracks not matched to an observation for several frames when such an object moves behind an obstacle.

As can be observed in the result section in figure 4.10 and 4.11, displaying the execution time and mean position error for the Kalman Filter and Particle filter, the Kalman filter performs significantly better in term of execution time. It also performs significantly better in regard to position estimates for lower fps as can be observed in figure 4.15. While the particle filter manages to perform slightly better when a large number of particles are used, the execution time grows rapidly with an increasing number of particles and the improved position tracking is negated by the significantly increased time for each update step. Therefore we can conclude that the Kalman filter algorithm is the best approach to take in the state estimation of the tracked objects.

It is also important to note that while our state estimators both manage to provide solutions to the given problem, they might not be optimally implemented. We therefore imagine that the performance in sense of execution time and position accuracy could be further improved upon by spending more time analysing the movement.

In our tracking solution the trails are matched with observations only based on proximity even though more information regarding the properties of the tracks could be considered. The LiDAR data contains in addition to the position of each point their intensity which is the ratio between the emitted and reflected light by the LiDAR device. This ratio is decided by the material properties and color of the objects hit, and would differ significantly between pedestrians wearing dark or light clothing. Therefore we imagine that the intensity returned from the LiDAR could also be used to differentiate between different pedestrians based on their clothing since the intensity would be fairly constant between frames and unique to each pedestrian. Other features, such as height and width could also be considered when matching tracks, but we imagine that these would be less unique for individual pedestrians.

5.4 Classification

As mentioned in the method chapter our classification process begins with looking at some easily extracted features, point per distance and the velocity, of tracked objects. Making assumptions about these properties for a pedestrian helps decide which of the dynamic objects detected should be put up for further consideration, and since the aforementioned features are extracted very efficiently from a tracked object compared to the features used in the SVM and 3DCNN it speeds up the overall classification process. Some problems that arise from these initial assumptions are, e.g., that a pedestrian can avoid being detected by moving very slowly through the LiDAR scene. The assumption about points per distance is also based on an assumption about linearity in the number of points over the increasing distances, something we can not be sure is true.

As observed in chapter 4.4 all of our different classifiers manage to reach a classification accuracy of around 90% on the simulated test data set, but due to the increasing feature extraction and prediction time for high resolution 3D images we have decided to only further consider the SVM classifier, as well as the 8x8x8 3DCNN classifier. Both of these methods have a very similar feature extraction time and classification accuracy, but the SVM prediction step is significantly more efficient than the 3DCNN.

Since all of our classifiers have been trained on data extracted from the Carla simulator we also wanted to observe how well they manage to perform on the real life data set described in chapter 3.1. It can be observed that the 3DCNN manages to perform significantly better on the real life data, reaching a classification accuracy of 82%, compared to the SVM classifier, which reaches only 63% accuracy. While the SVM has better classification accuracy on the simulated test data set, we believe this is mostly due to the lack of variation present in the simulator, enabling the SVM to create a very accurate model describing a simulated pedestrian that fails to hold true in the real world case. The 3DCNN doesn't only learn the size and dimensions of a pedestrian, but is able to detect features of pedestrian, e.g., arms, legs and head, and is better able to abstract these features so that it can detect their presence in new 3D images even if the location of these features are not consistent between the training and test data. This is best explained with and example; all pedestrians present in the Carla simulator move in the same way, with their arms swinging down by their sides and taking uniform strides with their legs. Since the SVM works mainly by deciding which voxels in the image are filled, if a pedestrian would suddenly raise their arms above their head, or start walking in a longer stride the SVM would decide that this observed object does not look similar to one of the objects it has been trained on and it would classify it as not being a pedestrian. The 3DCNN instead works by finding features, such as arms, legs and head, and is able to better understand if these features are present even if they are observed in an orientation or position that is not present in the training data. Using the same example as above, if a pedestrian raises their arms above their head the 3DCNN is still able to detect their presence in the image, and therefore will reach a better classification accuracy compared to the SVM when pedestrians start behaving in ways that differ from the simulated environment.

When recording with the LiDAR device it is necessary to define several parameters, such as height above ground plane and the angle of the sensor. We noticed that if these parameters are not defined close enough to the actual angle and height the extracted voxelizations will often be skewed in some ways, e.g., pedestrians are not standing upright but leaning backwards. This in turn creates problems for the classifiers, especially the SVM, since the voxelizations differ more from the simulated data that the classifiers were trained upon.

To increase the accuracy of the classifier it is important to not only consider a single observation of an object, since this would result in a high number of miss-classifications. Thanks to the tracker we are able to follow objects as they move through the scene, and can consider several images of the same object to reduce the number of miss-classifications. In our implementation we decided on looking at 5 consecutive images of the same object, and classifying it as a pedestrian only if at least 4 out of 5 of the observations were predicted as coming from a pedestrian. The risk of miss-classifying a pedestrian is $(1 - 0.82) = 0.18$ for the 3DCNN and $(1 - 0.63) = 0.27$ for the SVM and if each classification is uncorrelated to the last, the risk of miss-classification can be reduced to $0.18^4 = 0.0011$ for the 3DCNN and $0.27^4 = 0.005$ for the SVM.

Since the feature extraction step is the most time consuming part of the classification process it would be of very high interest to develop methods that do not require manual feature extraction, such as the method described in [18]. Being able to classify based only on the points contained within an object could significantly improve the performance of the classifier since such methods have been shown to be able to classify objects with a high accuracy. However, since the number of points contained in the detected dynamic objects vary greatly, and most deep learning methods are in need of consistent input size, each detected point cloud would either need to be up-sampled or down-sampled to create a consistent input.

To down-sample a point cloud one would simply randomly discard points until the desired number is reached, but to up-sample a point cloud one has to first detect a surface area of the object and then uniformly add points distributed across this area until the desired number is reached. When we tried to do this we observed that the up-sampling step was very time consuming, and the resulting point clouds were often simply shapes like triangles and squares with points uniformly distributed over them, which we decided were not able to accurately describe the shape of a pedestrian.

5.5 Testing on a limited system

Based on the results presented in Chapter 4.3.3 we can conclude that real time LiDAR point cloud pedestrian detection and tracking is possible to run on very limited hardware. We managed to achieve this by discarding points present in the point cloud randomly before passing it through our pipeline, and in this section we will mainly discuss how this affects the detection, tracking and classification of pedestrians.

Since our clustering step only considers something as a cluster if it contains more than 3 points separated by less than a meter, discarding many points reduces the distance that we are able to detect pedestrians at. Assuming that the number of points present on an object decreases linearly with the distance from the LiDAR device, discarding 50% of the points would reduce the detection distance by 50%.

The same is true for the pedestrian tracking, since we are only able to track objects found by the dynamic object detection. Another problem that arises in the tracking step is that even though points are randomly discarded there is a slight chance that the points removed are not uniformly distributed across the pedestrian, sometimes resulting in mainly the upper or lower half being removed. Since the tracking only considers the center of the object, removing only the lower or upper half has significant effects on the center of the object, and will be interpreted as a fast movement in the same direction as the change between the centers between two frames. This does not significantly affect the tracking at high frame rate since the state estimators receive measurements at a higher frequency, but at lower frame rate this can cause large inaccuracies in the velocity estimation and result in the tracker loosing track of the pedestrian. The tracking accuracy also deteriorates when the amount of frames per second is reduced, and it is important to find a good trade off between points discarded and reduced number of frames.

The classifiers work with an abstraction of the point cloud in the form of voxels, where a voxel is considered filled if at least one point is present within it. Therefore, if a lot of points are contained within the cluster, i.e., if it is close to the LiDAR device or very large, removing even a large amount of points before the abstraction has no significant effect on the voxel representation of the object. However, if a pedestrian point cloud is far away from the LiDAR device and already very sparse, the classification accuracy will deteriorate. As stated before, discarding points also has an effect on the distance that we are able to classify objects at, but due to the abstraction of the point cloud this effect is not as significant as the effect on the detection and tracking.

Based on observation and experimentation we conclude that we do not recommend removing more than 50% of points and using at least 5fps. Our pipeline was only able to achieve this when a flat ground plane was present in a scene, but we believe this could also be achieved on curved surfaces if the implementation was written in a more computationally efficient language than Python.

5.6 General discussion

In this section we will discuss the performance of the LiDAR and also compare it to some other commonly used sensors. First we will define an example scenario where we have observed problems arise when sensors, such as RGB cameras, heat cameras and radar are used. Imagine a lone tree rooted from a grass patch next to a small road with a sidewalk. The tree is swaying in strong winds, and so is the tall grass located next to the tree. When the RGB camera is used for motion detection the swaying tree and grass will trigger the motion detection even though they are clearly not areas of interest. Due to the unpredictable nature of the wind and the movement of the tall grass and tree it is fairly hard to incorporate these in some sort of background model, and actual objects of interest in close proximity to the false movement might be missed. The same is true for the radar, as it becomes very hard to differentiate the movement caused by the wind and objects standing in close proximity to the moving greenery. Now imagine that the sun sets and there is a car driving past in the dark with full beam headlights, casting light on the tree and surrounding area. The strong light present in the scene causes the entire scene to change brightness, making it difficult for the RGB camera to differentiate this type of change from movement within the scene. The headlight beams are also reflected in the leaves of the tree and the dew covered grass, as well as making the tree cast a large moving shadow over the grassy area, confusing the RGB camera as these reflections and shadows bear a lot of similarity to objects moving through the scene. The car then decides to do a three point turn, pointing its headlights directly at the RGB camera, essentially blinding it entirely until the car has turned completely and driven away. Now the sun rises and clear skies cause strong sunlight to be cast on the entire scene, the dark pavement area on the road and the sidewalk heats up to roughly 37 degrees Celsius. The hot pavement makes it very difficult for the heat camera to detect pedestrians and other warm blooded creatures, essentially camouflaging them. The strong warm sunlight is also reflected against the leaves of the tree, causing problems for both the RGB camera and the heat camera. The beautiful warm day causes a nearby lake to be very crowded, and due to lack of parking spaces near the lake several drivers decide to park their cars on the grassy patch next to the road. The large presence of metallic objects in the scene causes problems for the radar device, making it unable to detect movement in close proximity to the vehicles. Now Peter has heard that his friend Mark is going to the lake that day and is planning a prank on him, he has donned a military ghillie suit that camouflages him against the background greenery and is planning to sneak up on Peter and scare him. The similarity between the material and color of the suit and the background makes it very difficult for the RGB camera to detect him, and the thick suit also camouflages the heat radiating from his body.

Most of the problems discussed above can be avoided by replacing these sensors with a LiDAR device. Since its detection is not based on movement within the scene like radar, it is very robust against a moving background like tall grass swaying in the wind. Problems instead arise if the wind causes a dynamic background object to move enough to escape the area defined as the background, which can happen with tall trees swaying in very strong winds, but is generally avoided with smaller dynamic background objects like bushes and grass that do not move as much even in very windy conditions. It however struggles in the same way as the radar in detecting objects standing close to dynamic background objects, since these objects will have a space surrounding them defined as belonging to the background. Since the LiDAR device emits its own light, and only registers the reflected lasers and not other light phenomena present within the scene, it is not affected by the car's headlights, even when they are pointed directly at the LiDAR device. Since any object hit by the lasers emitted from the LiDAR is registered, there really are not any reliable ways to attempt to camouflage oneself against the background. It would be possible if an object is painted in such dark colors that essentially no light was reflected back to the LiDAR device, but while such colors exist they are not commonly found, especially not in pedestrian clothing. A problem that arises with the LiDAR, especially in our implementation since we have a predefined background model, is that if an object moves in very close proximity to the background, e.g., a person crawling on the ground, it can avoid being detected. Also, if the position and angle of the LiDAR device is changed by vibrations in the ground or someone poking it with a stick, it can cause the scene to shift so much that it escapes the predefined background model, and will cause the background to have to be recomputed before detection can be performed again.

Even though detection using LiDAR has some small disadvantages not present in other sensors we believe that these are outweighed by the advantages, and conclude that a LiDAR is a very robust sensor when it comes to object detection.

Our experiments show that while our implementation has problems classifying objects that are very far from the LiDAR device, if an object is detected close to the LiDAR device it can be followed until it is roughly 100m away from the device. While larger objects can be detected further away from the LiDAR, a pedestrian 100m away from the device consists of only 10 or less points, and while we are able to detect and track such small objects the classification process becomes unreliable.

5.7 Future work

We were able to create a model that is able to classify if something is a pedestrian or not. A reason for this is thanks to the data set that we collected with the help of the Carla Simulator. But how good are data sets generated from simulated data in Carla compared to annotated real-world data collected from a physical LiDAR? A future step would be to use the Carla simulator in order to build a replica of an existing environment and fine tune the settings of the Carla LiDAR to simulate the physical LiDAR as close as possible. Then a comparison between the real-world generated data captured with the physical LiDAR from the real environment and the simulated data generated from the Carla environment could be made. This comparison could tell if it is worth to start using simulated data in order to train classifiers or if there is still work to be done in order to simulate real-world data.

Since the Carla simulator currently contains a limited amount of animations and dynamic objects there is room for improvements of our data set in the future. One way of improving our current data set has already been discussed in the data gathering section in chapter 5. Another way to improve the existing data set would be to let the existing model train on real-world annotated data sets. This would increase the models capabilities to recognize and classify both common and uncommon interactions and behaviours of dynamic objects.

The Cepton Vista-P60 LiDAR that we currently have access to is able to provide LiDAR data which, with some slight modifications, is good enough to be used in our system. However, if a LiDAR with better performance would be introduced to the system an even better result could be achieved. An increase in how far the LiDAR is able to measure, the angular resolution and the scan rate would make it easier for our system to classify different objects more accurately and in a faster manner. However, the increased amount of data that would need to be processed by the clustering method and classifier would require us to make some modifications to the system in order to keep it running efficiently enough.

The classifier has been trained on the whole data set that we generated with the help of the Carla simulator and it performs well at classifying objects in our specific use case. However, the classifier is only as good as the data it has been training on and if we would want a system able to dynamically adapt to new patterns in the data, a method like online machine learning could be used. The new data that we would like to feed into the classifier could either be manually gathered and annotated data sets or data saved by the object tracker within our system. The object tracker could be modified in such a way that it keeps the data of objects until classification of the classifier has been done. If the classifier at a certain point says it is a pedestrian we can save all the point clusters of that object from previous and upcoming frames and feed that data into the classifier. The modification would make the system able to learn new patterns just by being able to classify an object in one frame. Further investigation in the performance and workload of such modification would be necessary.

Some advantages of the LiDAR are that it gives highly accurate depth values, measures at great distances, and is able to generate a point cloud of the environment containing different 3D shapes. A major drawback is that the resolution of the output is not very high. In order to tackle this problem sensor fusion could be used in order to either combine the data from different sensors or the results obtained by the different sensors. The data fusion between a camera, which is a sensor that is able to provide high resolution outputs but does not provide any depth information, and a LiDAR would give us a system capable of providing high resolution output with accurate depth information. Sensor fusion could be beneficial to our system since high resolution images are easier to classify compared to point clouds, we would still retain the depth information of the pixels and the results from two different sensors would increase the confidence in the system. A radar is another sensor that could be used together with the LiDAR. This would make the system also able to penetrate insulators and determine exact position and velocity of a target.

Our approach of an object detection and tracking system is just one of many different approaches. One approach that is widely used is instead of dividing the system into different parts, such as background filtration and classification, feed the whole data set into a convolutional neural network and let the network do the whole process of filtering and classifying. This would require more extensive data sets in order to get the desired performance of the system but could definitely be a future path to improve the system.

References

[1] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[2] David Donoho. High-dimensional data analysis: The curses and blessings of dimensionality. *AMS Math Challenges Lecture*, pages 1–32, 01 2000.

[3] J. Wu, H. Xu, and J. Zheng. Automatic background filtering and lane identification with roadside lidar data. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–6, 2017.

[4] Wen Xiao, Bruno Vallet, Konrad Schindler, and Nicolas Paparoditis. Simultaneous detection and tracking of pedestrian from panoramic laser scanning data. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, III-3:295–302, 06 2016.

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

[6] A. Azim and O. Aycard. Detection, classification and tracking of moving objects in a 3d environment. In *2012 IEEE Intelligent Vehicles Symposium*, pages 802–807, 2012.

[7] Y. Song, H. Zhang, Y. Liu, J. Liu, H. Zhang, and X. Song. Background filtering and object detection with a stationary lidar using a layer-based method. *IEEE Access*, 8:184426–184436, 2020.

[8] A. Dewan, T. Caselitz, G. D. Tipaldi, and W. Burgard. Motion-based detection and tracking in 3d lidar scans. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4508–4513, 2016.

[9] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, page 226–231. AAAI Press, 1996.

[10] D. Comaniciu and P. Meer. Mean shift: a robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):603–619, 2002.

[11] Hannaneh Najdataei, Yiannis Nikolakopoulos, Vincenzo Gulisano, and M. Papatriantafilou. Lisco: A continuous approach in lidar point-cloud clustering. *ArXiv*, abs/1711.01853, 2017.

[12] Lvwen Huang, Siyuan Chen, Jianfeng Zhang, Bang Cheng, and Mingqing Liu. Real-time motion tracking for indoor moving sphere objects with a lidar sensor. *Sensors*, 17(9), 2017.

[13] Y. Du, W. ShangGuan, and L. Chai. Particle filter based object tracking of 3d sparse point clouds for autopilot. In *2018 Chinese Automation Congress (CAC)*, pages 1102–1107, 2018.

[14] F. Gustafsson, F. Gunnarsson, N. Bergman, U. Forssell, J. Jansson, R. Karlsson, and P. . Nordlund. Particle filters for positioning, navigation, and tracking. *IEEE Transactions on Signal Processing*, 50(2):425–437, 2002.

[15] Roy Andersson and Erik Andersson. Lidar pedestrian detector and semi-automatic annotation tool for labeling of 3d data. *Master's Theses in Mathematical Sciences LTH*, 2019. Student Paper.

[16] G. Melotti, A. Asvadi, and C. Premebida. Cnn-lidar pedestrian classification: combining range and reflectance data. In *2018 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*, pages 1–6, 2018.

[17] Y. Tatebe, D. Deguchi, Y. Kawanishi, I. Ide, H. Murase, and U. Sakai. Pedestrian detection from sparse point-cloud using 3dcnn. In *2018 International Workshop on Advanced Image Technology (IWAIT)*, pages 1–4, 2018.

[18] R. Qi Charles, Hao Su, Mo Kaichun, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 77–85, 2017.

[19] Yuji Roh, Geon Heo, and Steven Euijong Whang. A survey on data collection for machine learning: A big data - ai integration perspective. *IEEE Transactions on Knowledge and Data Engineering*, 33(4):1328–1347, 2021.

[20] Theodoros Rekatsinas, Xu Chu, I. Ilyas, and C. Ré. Holoclean: Holistic data repairs with probabilistic inference. *Proc. VLDB Endow.*, 10:1190–1201, 2017.

[21] Sanjay Krishnan, Jiannan Wang, Eugene Wu, Michael J. Franklin, and Ken Goldberg. Activeclean: interactive data cleaning for statistical modeling. *Proceedings of the VLDB Endowment*, 9(12):948 – 959, 2016.

[22] Cepton. Cepton vista-p60. `https://www.cepton.com/products/vista-p`.

[23] Alireza Shafaei, James J. Little, and Mark Schmidt. Play and learn: Using video games to train computer vision models. *ArXiv, abs/1608.01745.*, 2016.

[24] Epic Games. Unreal engine 4. `https://www.unrealengine.com`.

[25] Carla. Open-source simulator for autonomous driving research. `https://carla.org/`.

[26] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. Modelnet40. `https://modelnet.cs.princeton.edu/`.

[27] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1912–1920, 2015.

[28] Wenlong Yue, Junguo Lu, Weihang Zhou, and Yubin Miao. A new plane segmentation method of point cloud based on mean shift and ransac. *2018 Chinese Control And Decision Conference (CCDC), Chinese Control And Decision Conference (CCDC), 2018*, pages 1658 – 1663, 2018.

[29] Rensselaer Polytechnic Institute. Image Processing Laboratory and D.J.R. Meagher. *Octree Encoding: a New Technique for the Representation, Manipulation and Display of Arbitrary 3-D Objects by Computer*. 1980.

[30] Aditya A. V. Sastry. Background modelling using octree color quantization. *ArXiv*, abs/1412.1945, 2014.

[31] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, September 1975.

[32] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[33] Pierre Del Moral. Nonlinear filtering: Interacting particle resolution. *Comptes Rendus de l'Académie des Sciences - Series I - Mathematics*, 325(6):653–658, 1997.

[34] Carla. Carla sensors reference. `https://carla.readthedocs.io/en/latest/ref_sensors/`.

[35] Carla. Carla semantic lidar sensor. `https://carla.readthedocs.io/en/latest/ref_sensors/#semantic-lidar-sensor`.

[36] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018.

[37] J. Zbala, Piotr Ciepka, and Adam Reza. Pedestrian acceleration and speeds. 91:227–234, 01 2012.

[38] J. Kiefer and J. Wolfowitz. Stochastic Estimation of the Maximum of a Regression Function. *The Annals of Mathematical Statistics*, 23(3):462 – 466, 1952.

[39] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.

[40] Coral. Coral dev board. `https://coral.ai/products/dev-board`.

[41] Abraham. Savitzky and M. J. E. Golay. Smoothing and differentiation of data by simplified least squares procedures. *Analytical Chemistry*, 36(8):1627–1639, 1964.

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2021-06-10

EXAMENSARBETE Pedestrian detection and tracking in 3D point cloud data on limited systems

STUDENTER William Winberg, Jacob Berntsson
HANDLEDARE Elin-Anna Topp (LTH)
EXAMINATOR Jacek Malec (LTH)

Detektion och spårning av fotgängare med 3D laserskanning

POPULÄRVETENSKAPLIG SAMMANFATTNING **William Winberg, Jacob Berntsson**

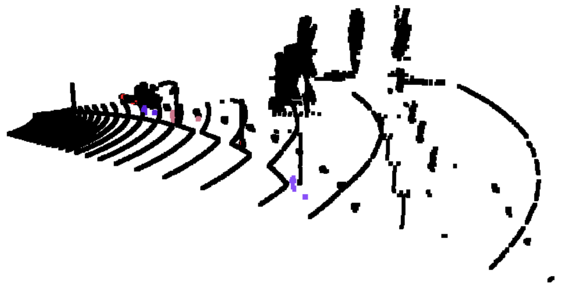
LiDAR (Light detection and ranging) är en teknologi som ständigt förbättras och intresset att använda LiDAR för att detektera och spåra människor ökar. Vi har undersökt vilka metoder som kan åstadkomma detta på system med begränsad hårdvara.

LiDAR-teknologi har förbättrats på senare tid och intresset för att använda dessa sensorer för att detektera och spåra människor som befinner sig på platser de inte ska vara på har ökat. För att utöka användningsområdena för LiDAR-sensorerna krävs låga prestandakrav på datahanteringen och bearbetningen, främst för att minska kostnaderna i implementationen.

Vi har utvecklat en fullständig lösning för detektering och spårning av människor, med ett flertal utbytbara metoder i varje steg av processen för att kunna analysera vilka som fungerar bäst i olika situationer. Vi har definierat tre tydliga steg i processen som vi kallar dynamisk objektdetektering, objektspårning och objektklassificering. Vi har sedan analyserat prestandakraven för de olika lösningarna för att kunna avgöra vilka som fungerar bäst på system med låg prestanda.

Genom att använda en simulator har vi skapat en miljö för att kunna samla in LiDAR-data och skapa ett flertal testfall för vår lösning. Den simulerade miljön möjliggjorde skapandet av väldigt stora annoterade dataset för träning av objektklassificerarna, samt skapandet av varierade testmiljöer för att undersöka för- och nackdelar med LiDAR-sensorer jämfört med andra sensorer. Vi har även analyserat övergången mellan simulerad och riktig data för att undersöka om modeller som skapats för objektklassificering också är applicerbara i verkliga scenarier.

Den fullständiga lösningen, se figur 1, har testats på ett system med begränsad hårdvara för att undersöka om det är möjligt att utföra detektionen och spårningen i realtid och de begränsningar som måste införas för att åstadkomma detta.


Figur 1: Människor som klassificerats och håller på att bli spårade.

Resultaten visar att exekvering i realtid är möjlig om begränsingar på antal skickade bilder per sekund samt mängden data minskar innan objektdetektering och spårning utförs. Dessa begränsningar leder till försämringar i varje steg av processen, men dessa är så pass små så detektering och spårning kan utföras med tillräcklig precision för att uppnå önskat resultat.