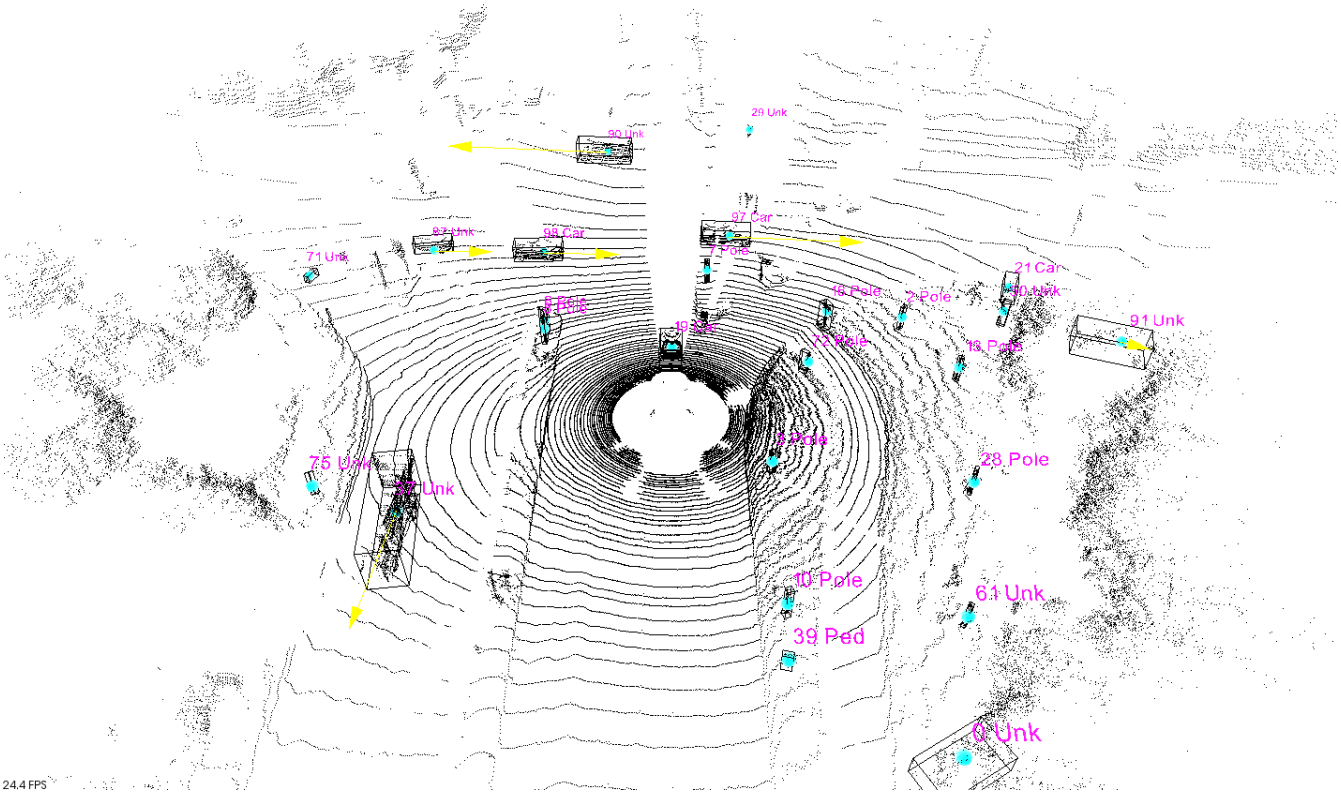
CHALMERS



Robust tracking of dynamic objects in LIDAR point clouds

Master's Thesis in Engineering Mathematics

JOHAN VILLYSSON

Department of Mathematical Sciences
Mathematical Statistics
Chalmers University of Technology
Gothenburg, Sweden 2014
Master's Thesis 2014:1

Abstract

The field of active safety in the automotive industry is a fast-growing area of development today. Safer cars have become one of the top-priorities among car-makers in the recent years. The driver support functions that make cars safer rely on accurate on-board sensor readings of the surroundings of the car. For verification of this accuracy, a LIDAR can be used to get a 360° view of the surroundings of the vehicle that can then be used as ground-truth. Despite high accuracy and resolution combined with wide field of view typical for LIDARs, extraction of objects from the LIDAR data, the so called point clouds, requires advanced post-processing algorithms and is often challenging.
In this thesis, an off-line tracking algorithm that detects and tracks dynamic and stationary objects using LIDAR point clouds is developed and implemented. The tracker is based on Bayesian tracking theory and utilizes the fact that it is run off-line by tracking both forward and backward in time.
The results of the tracker show comparable accuracy to current trackers in the literature. The tracker effectively tracks any sort of objects whether they are stationary or dynamic and gives accurate readings on positions, dimensions, velocities and accelerations.

Acknowledgments

I would like to thank my supervisors at Chalmers, Aila Särkkä and Mats Rudemo, for their support and ideas during the working of the project. A huge thanks also goes to my supervisor at Volvo Car Corporation, Yury Tarakanov, who supported the work with new ideas and immense knowledge.
I would also like to send my gratitude to the people working with the KITTI-dataset for providing such good data for evaluation and the people developing the Point Cloud Library, PCL, which helped a lot in the implementation of the tracker.
Last but not least, a big thanks to my colleague and "partner-in-crime" Jonathan Ahlstedt for his patience with all our discussions and idea-exchanging.

Johan Villysson, Gothenburg 14/01/22

Contents

1

Introduction

The active safety department at Volvo Cars is a fast-growing and leading branch of driver support functions. For example, functions as collision mitigation by braking, lane departure warning and adaptive cruise control are becoming popular additions to new cars. For all these functions to work properly, it is crucial that the vehicle perceives the environment around it, which puts hard requirements on the sensors that are used for the perception. To be able to verify that the sensors of a car such as radars, cameras and lasers perform at their best, an accurate reference sensor is needed. A popular choice is a Laser scanner, LIDAR(a blend of light and radar), that can provide testers with accurate data on the surroundings of the car.
The LIDAR emits laser beams from diodes that rotate at a frequency of 5-15 Hz[1]. The beams are reflected off objects and then gathered in the LIDAR for analysis. The resulting data of the surroundings is a point cloud consisting of three-dimensional point data, see front page for an example. The point cloud describes not only the positions of objects but also the shapes of the objects due to the many laser-beams emitted.
Post-processing of the logged LIDAR-data is needed for sensor verification. Specifically it is crucial to model the dynamics of the surroundings. This is usually done by an object tracker that keeps track of objects around the car to accurately determine velocities and headings of nearby targets such as other cars and pedestrians.
In this thesis a tracking algorithm is developed, implemented and evaluated using ground truth data.
For the tracking algorithm to work, accurate measurements on possible objects to track need to be delivered. These measurements usually come from a segmentation algorithm, that clusters points in the point ploud into possible objects. The segmentation algorithm that was developed and used in this thesis can be found in Appendix A. The tracking-algorithm also relies on that accurate ego-trajectory information, a measurement on the dynamics of the vehicle with the LIDAR, is delivered. An algorithm to calculate the ego-trajectory from the point cloud will not be a part of this thesis, instead it is assumed that such data exist. For example such data can come from a combination of a GPS and

gyroscope sensors.
The tracking is done off-line, i.e. not in real time in the car. The tracking-algorithm has the ability to track stationary objects but the focus and precision is put on dynamic objects such as cars, pedestrians and cyclists as those are the main focus of collision avoidance functions.
The ultimate goal of the project is to develop a sufficiently robust and accurate post-processing chain to be used for on-board sensor verification tasks at Volvo Cars.

The report is partitioned into a theory-part and a hands-on part. The first section in the theory-part gives a basic review on the models and the theory behind tracking. It also contains a small literature review on some of the recent papers published on the subject of dynamic object tracking.
A lot of emphasis is put on the methods and data section which contains information regarding implementation and evaluation of the tracker that has been developed. In this section a small portion is dedicated to the KITTI-dataset[2] which was used to evaluate the tracker and the Point Cloud Library, PCL[3], which aided in the programming implementation.
In the final sections of the report the tracker is evaluated and compared to other tracking results in the literature.
The appendix contains an explanation of the segmentation-routine that was developed and used together with the tracker.

2

Tracking theory

In this chapter, a study of current tracking theory is conducted. The study was done early in the project and contains references to several recent scientific papers on the subject at hand. The focus is on automotive tracking theory, in the context of automotive safety systems and autonomous driving.

2.1 Bayesian tracking theory

The basic model of tracking that is frequently used in the literature on tracking today is based on some sort of Bayesian tracking theory[4, 5, 6, 7, 8, 9, 10, 11, 12]. This will also be the model that is used in this project.
A good background to setting up the model for the state-space of an object can be found in[4] or [6]. In this paper the theory and terminology are based on the theory given in[4], where the theory is presented in an understandable and rigorous way making it easy for the reader to comprehend Bayesian tracking theory.
The main idea is to set up a discrete time model with a state vector for each object at each time step, namely

$$\mathbf{x}_k = [(\mathbf{x}_k^1)^T, (\mathbf{x}_k^2)^T, ..., (\mathbf{x}_k^{n_{obj}(k)})^T]^T, \tag{2.1}$$

where $k \in \mathbb{N} = \{1,2,...\}$ and $n_{obj}(k)$ is the number of objects tracked at time step k. The actual time between the time steps can vary but is mostly constant since observations/measurements come in specific time-intervals. The measurements we get from the sensor equipment at each time step k are denoted by

$$\mathbf{y}_k \in \mathbb{R}^{n_y(k)}, \tag{2.2}$$

where $n_y(k)$ is the number of measurements received at time step k. This number depends on how many objects the measurement device, sensor, can detect at each time

step.
The last definition is the set of measurements up to and including time step k,

$$\mathbf{Y}_{1:k} = [(\mathbf{y}_1)^T, (\mathbf{y}_2)^T, ..., (\mathbf{y}_k)^T]^T. \tag{2.3}$$

Using all measurement data, the goal now is to calculate the posterior density function of the states of all objects, i.e. to find

$$p(\mathbf{x}_k|\mathbf{Y}_{1:k}). \tag{2.4}$$

To put things in perspective the object state vector can include different relevant information about the object, for example

$$\mathbf{x}_k^i = [x_k^i, y_k^i, \dot{x}_k^i, \dot{y}_k^i]^T, \tag{2.5}$$

where $\dot{x}$ and $\dot{y}$ are the velocities in the x- and y-direction respectively. In other words, the state of the object is determined by it's x and y position in the plane and the corresponding velocities in these directions.
When using a range finder, typically a radar, the measurements are simply coordinates in 2D or possibly in 3D,

$$\mathbf{y}_i = [x_i, y_i]^T. \tag{2.6}$$

More elaborate representations of the state space of an object are often used and also more measurement information such as an object's/measurement's size and direction is often available. Most authors have their own unique representation[4, 7, 9, 12].

A tracking model consists of different modules and in this report the modules are based on the modules introduced in [4].
The motivation for these modules is that a tracking system working in real-time should be able to predict the state of an object in the next time step using the previous measurements. This prediction is needed when new measurements are available and one must associate measurements to existing tracked objects or to new objects to track. The list of modules is as follows:

- **Data association**
  In this module, measurements at the current time step are associated with objects that are being tracked.
- **Track management**
  Objects which have new measurements associated with them are validated. Objects that haven't had any associated measurements for a period of time are deleted and new measurements with no object associated to them can be initialized as a new object to track.
- **Filtering**
  The posterior density function $p(\mathbf{x}_k|\mathbf{Y}_{1:k})$ is calculated. The state of the object is updated.

- **Prediction**
  $p(\mathbf{x}_{k+1}|\mathbf{Y}_{1:k})$ is calculated so that new measurements can be associated with the predicted states of the tracked objects.

For the objects that are being tracked we must also decide on a **Motion model** which describes how the state of an object changes between time steps. This motion model is incorporated in the filtering and prediction modules. The goal of a good motion model is to be able to correctly model the motion of an object to create the best predictions of the state of an object at a later time. For the prediction to be good one also incorporates uncertainty measures in the state model.
A standard way of describing the state of an object in Bayesian tracking theory is the following:

$$\mathbf{x}_k = \mathbf{f}_{k-1}(\mathbf{x}_{k-1},\mathbf{v}_{k-1}). \tag{2.7}$$

Here $\mathbf{v}_{k-1}$ is a noise process, often assumed to be zero-mean Gaussian noise[4], which is needed to model the uncertainties in the motion model. $\mathbf{f}_{k-1}$ is simply a function describing the relationship between the former and current state of the object. Using this model one can make the most likely predictions of the state of an object at a later time point.
Similar to how we formulated the motion model for the objects one also formulates a **Measurement model** i.e. a model describing the relationship between measurements and object states, i.e.

$$\mathbf{y}_k = \mathbf{h}_k(\mathbf{x}_k,\mathbf{w}_k). \tag{2.8}$$

As in the motion model, $\mathbf{w}_k$ is a noise process describing the uncertainties in the measurements. Furthermore, $\mathbf{h}_k$ is a function describing the relationship between the measurement and the state of an object.

With the above framework a recursive tracking algorithm can be constructed. The next step is to determine how each of the 4 modules above should be implemented.

2.1.1 Data Association

In this module, the new LIDAR measurements are associated with existing tracklets, objects that are being tracked, using the prediction.
The first step in any association algorithm is Gating[4], which is a way to restrict the number of possible associations that can be made to a certain tracklet by calculating a gate around the tracklet. The possible associations lie inside this gate and the measurements that are too far away to be possible associations lie outside the gate. Both velocity and acceleration can be taken into account when calculating the sort-of threshold to how far away the object can maximally travel during one time-step. When this gate has been calculated you are left with much less possible associations inside the gate. The goal now is to find the most probable association inside the gate to a tracklet.
A simple approach to this problem is the Nearest-Neighbor-approach[4], which in principle associates the closest, in Euclidean distance, measurement to the tracklet. The

benefit of this approach is the simplicity in calculations. The negative aspect is that in the simplest form of the Nearest-Neighbor association, one measurement can be associated with several tracklets. To circumvent this, the so called Global Nearest-Neighbor-approach can be used. A global optimization for the associations means that you can enforce a restriction that a measurement can only be associated with one tracklet.

A downside of any Nearest-Neighbor-algorithm is that when objects are very close to each other, maybe they cross each others paths very close to each other, which could lead to wrong associations. Another downside is the lack of different hypotheses for associations.
To further increase the accuracy of the association one can take a Probabilistic-approach. A commonly used approach is the Joint Probabilistic Data Association, JPDA[13, 14], where certain probabilities are calculated for each possible association. This is done globally so that a measurement can only be associated with one tracklet. A way of calculating the probabilities can be found in[14]. The benefit of this method is that different association hypotheses can be evaluated to find the most probable one.

Multiple hypotheses tracking, MHT[7], is the common name for algorithms that save different association hypotheses in each time step so that the most probable overall association hypothesis can be calculated at a later time. The problem with MHT is that the number of possible hypotheses grows exponentially. This means that it is crucial to prune not-probable hypotheses in each time step. A good introduction to MHT can be found in [7].

2.1.2 Track Management

After new measurements have been collected from the LIDAR and the association-step above has been completed, there are:

- Measurements that have not been associated with existing tracks.
- Current tracklets that get no association with the new measurements.

Therefore, the track-management-module has to:

- Create new tracklets for non-associated measurements and use some measurement information to initialize the state of an object.
- Delete tracklets that haven't had any association during a specific period of time.
- Keep tracklets in memory if no association has been made in the current time step unless above. Use the predicted state as a measurement.
- Validate tracklets, i.e. stable and confident tracklets should be validated and given a higher "rank".

All the above items are done so that only the good and relevant tracklets become the final tracklets at the end of the algorithm.

2.1.3 Prediction and filtering

In the filtering module, the posterior probability density function, $p(\mathbf{x}_k|\mathbf{Y}_{1:k})$, is calculated recursively. Since it is calculated recursively an appropriate a-priori information is needed, $p(\mathbf{x}_0|\mathbf{y}_0)$. In [4], different ways of initializing this a-prior information is discussed. For this report, it is assumed that no a-prior information is available.
The following equations and calculations are based on the theory given in [4].
Now if we assume that from the previous recursion step we have

$$p(\mathbf{x}_{k-1}|\mathbf{Y}_{1:k-1}), \tag{2.9}$$

the prediction, $p(\mathbf{x}_k|\mathbf{Y}_{1:k-1})$ can be calculated using the Chapman-Kolmogorov equation and the motion model (2.7).

$$\begin{aligned} p(\mathbf{x}_k|\mathbf{Y}_{1:k-1}) &= \int p(\mathbf{x}_k|\mathbf{x}_{k-1},\, \mathbf{Y}_{1:k-1})p(\mathbf{x}_{k-1}|\mathbf{Y}_{1:k-1})d\mathbf{x}_{k-1} = \\ &= \int p(\mathbf{x}_k|\mathbf{x}_{k-1})p(\mathbf{x}_{k-1}|\mathbf{Y}_{1:k-1})d\mathbf{x}_{k-1}, \end{aligned} \tag{2.10}$$

since from the motion model (2.7) we have that the state of an object is a Markov process giving $p(\mathbf{x}_k|\mathbf{x}_{k-1},\, \mathbf{Y}_{1:k-1}) = p(\mathbf{x}_k|\mathbf{x}_{k-1})$.
When a new measurement $\mathbf{y}_k$ arrives and is associated with $\mathbf{x}_k$ we can calculate $p(\mathbf{x}_k|\mathbf{Y}_{1:k})$ by using the Bayes rule

$$\begin{aligned} p(\mathbf{x}_k|\mathbf{Y}_{1:k}) = p(\mathbf{x}_k|\mathbf{y}_k,\ \mathbf{Y}_{1:k-1}) &= \frac{p(\mathbf{y}_k|\mathbf{x}_k,\ \mathbf{Y}_{1:k-1})p(\mathbf{x}_k|\mathbf{Y}_{1:k-1})}{p(\mathbf{y}_k|\ \mathbf{Y}_{1:k-1})} = \\ &= \frac{p(\mathbf{y}_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{Y}_{1:k-1})}{p(\mathbf{y}_k|\ \mathbf{Y}_{1:k-1})}, \end{aligned} \tag{2.11}$$

where $p(\mathbf{y}_k|\ \mathbf{Y}_{1:k-1})$ can be calculated as

$$p(\mathbf{y}_k|\mathbf{Y}_{1:k-1}) = \int p(\mathbf{y}_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{Y}_{1:k-1})d\mathbf{x}_k. \tag{2.12}$$

The equations above state the posterior density function $p(\mathbf{x}_k|\mathbf{Y}_{1:k})$ and with this information it is easy to form an optimal estimation of the state of an object $\mathbf{x}_k$ at time step k. Though the above equations are generally not solvable analytically, there are some cases where if we impose certain assumptions they can be solvable. One of the most important cases where an analytical solution can be found is the Kalman filter setting.

Kalman filter

The following assumptions are made in the Kalman filter setting[4]:

- The prior distribution $p(\mathbf{x}_0|\mathbf{y}_0)$ is assumed to be Gaussian.
- Both the measurement noise $\mathbf{w}_k$ and the process noise $\mathbf{v}_k$ are Gaussian and independent for all time steps.

- $\mathbf{f}_{k-1}$ in (2.7) and $\mathbf{h}_k$ in (2.8) are assumed to be linear.

The above assumptions yield the following new motion model and measurement model:

$$\mathbf{x}_k = \mathbf{F}_{k-1}\mathbf{x}_{k-1} + \mathbf{v}_{k-1} \tag{2.13}$$

$$\mathbf{y}_k = \mathbf{H}_k\mathbf{x}_k + \mathbf{w}_k, \tag{2.14}$$

where $\mathbf{F}_{k-1}$ and $\mathbf{H}_k$ are matrices that can depend on the time step k. The noise processes $\mathbf{v}_{k-1}$ and $\mathbf{w}_k$ are zero-mean Gaussian with covariance matrices $\mathbf{Q}_{k-1}$ and $\mathbf{R}_k$ respectively.
From the above assumptions the density functions from (2.10) and (2.11) can be calculated yielding the following results(for more details, see [4]):

$$p(\mathbf{x}_{k-1}|\mathbf{Y}_{1:k-1}) \sim \mathcal{N}(\hat{\mathbf{x}}_{k-1|k-1},\, \mathbf{P}_{k-1|k-1}) \tag{2.15}$$

$$p(\mathbf{x}_k|\mathbf{Y}_{1:k-1}) \sim \mathcal{N}(\hat{\mathbf{x}}_{k|k-1},\, \mathbf{P}_{k|k-1}) \tag{2.16}$$

$$p(\mathbf{x}_k|\mathbf{Y}_{1:k}) \sim \mathcal{N}(\hat{\mathbf{x}}_{k|k},\, \mathbf{P}_{k|k}), \tag{2.17}$$

where $\mathbf{P}$ is a covariance matrix and $\hat{\mathbf{x}}$ is explained below. The key factor is that the densities are Gaussian, hence only the mean and the covariance are needed to describe the densities.
For the prediction that will be used in the association module we get the following expression:

$$\hat{\mathbf{x}}_{k|k-1} = \mathbb{E}[\mathbf{x}_k|\mathbf{Y}_{1:k-1}] = \mathbf{F}_{k-1}\hat{\mathbf{x}}_{k-1|k-1} \tag{2.18}$$

$$\mathbf{P}_{k|k-1} = \mathrm{Cov}[\mathbf{x}_k|\mathbf{Y}_{1:k-1}] = \mathbf{F}_{k-1}\mathbf{P}_{k-1|k-1}\mathbf{F}_{k-1}^T + \mathbf{Q}_{k-1} \tag{2.19}$$

It is now convenient to introduce the innovation, that describes the new measurement information uncorrelated with all previous measurements,

$$\tilde{\mathbf{y}}_k \equiv \mathbf{y}_k - \mathbb{E}[\mathbf{y}_k|\mathbf{Y}_{1:k-1}]. \tag{2.20}$$

Using basic Gaussian theorems it can be proven that the innovation is zero-mean Gaussian with the conditional covariance

$$\mathbf{S}_k = \mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}_k^T + \mathbf{R}_k. \tag{2.21}$$

Now using known theorems on conditional probability density function of multivariate Gaussian random vectors, the motion model (2.13) and the measurement model (2.14) one can derive an expression for the update-step of the Kalman filter. This update-step makes use of the state from the prediction step, $\hat{\mathbf{x}}_{k|k-1}$, and the newly associated measurement, $\mathbf{y}_k$, and forms the optimal update for the state of an object (for more details the reader is referred to [4]).

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k(\mathbf{y}_k - \mathbf{H}_k\mathbf{P}_{k|k-1}), \tag{2.22}$$

$$\mathbf{P}_{k|k} = \mathbf{P}_{k|k-1} - \mathbf{K}_k(\mathbf{S}_k)^{-1}\mathbf{K}_k^T, \tag{2.23}$$

$$\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}_k^T(\mathbf{S}_k)^{-1}, \tag{2.24}$$

where the matrix $\mathbf{K}_k$ is the so-called Kalman gain matrix. It makes sure that if the accuracy in the measurements is high, i.e. the magnitude of $\mathbf{R}_k$ is small, then the state update will be influenced mostly by the measurement. Conversely, if $\mathbf{P}_{k|k-1}$ is small in magnitude compared to $\mathbf{R}_k$ then the object state doesn't change much. Hence, more emphasis is put on the predicted estimate than on the newly associated measure when updating the object state.

Non-linear filtering

In the above theory the assumptions on the motion model and measurement model might seem intrusive. If the motion and measurement models are non-linear, one can use an **Extended Kalman filter**, where the non-linearity among the system models is linearized using the first order Taylor expansion. This means that the same equations can be used for prediction and update as in the ordinary Kalman filter.
Another way of handling non-linear filtering, where the Gaussian assumption is ill-fitting, is by using a **Particle filter**. A very good book on particle filters is the one by Ristic et al. [6].
The idea behind particle filters is to use Monte Carlo approximations and importance sampling to describe the posterior density function $p(\mathbf{x}_k|\mathbf{Y}_{1:k})$. The word "particle" comes from the fact that one uses samples from a distribution to describe it and these samples are referred to as particles.

2.2 Current tracking-literature

The state of tracking theory in context of automotive research is constantly evolving. More elaborate ways of tracking dynamic objects using LIDARs are developed frequently. In the following section, a study and comparison on some of the current scientific papers on the subject is conducted.

Morton et al. [15], conducts a comparative study on different tracking modules using a Velodyne LIDAR. The main focus is laid on how an object is represented when tracking. The baseline is to use the centroid, center of gravity, of an object as a representation. The centroid representation is compared to using the 3D appearance of an object as a representation when tracking. For filtering and prediction the authors use a constant velocity Kalman filter or no filter at all to see the influence of filtering. When associating new measurements to existing tracklets the so-called Hungarian algorithm is used for the centroid representation and ICP, iterative closest point, for the 3D appearance representation.
The authors found that the simplest representation of an object, the centroid, was the best representation for a tracklet. For example when tracking pedestrians, the centroid

representation achieved up to 95% correct data associations compared to 60% for the 3D appearance modeling. And when comparing the use of a Kalman filter for centroids or no filtering at all, the Kalman filter produced the lowest position errors and speed errors. In particular the speed errors were very large when not using any filtering at all.

Zhang et al.[7] implements a multiple hypotheses tracking, MHT, algorithm. The authors make use of data from a Velodyne LIDAR to track vehicle-like objects and use a linear Kalman filter together with a constant velocity motion model for filtering. They put their emphasis on the MHT algorithm for data association. The authors represent the association problem as a linear integer assignment problem, LIAP, which they then solve using an algorithm introduced by Jonker et al. in [16]. To receive different association hypotheses from the assignment problem the authors use Murty's algorithm, given in [17]. Since the number of hypotheses grow exponentially they prune hypotheses by always maintaining the top k best hypotheses, the hypotheses with the lowest cost functions in LIAP.
Their evaluation of their tracking is mainly focused on how well it performs in occlusion, i.e. when a tracked object goes in the shadow behind another object. The goal is to still maintain the tracked object until it emerges from the occlusion again. A bad tracker looses the object in occlusion and starts a new tracklet when it emerges again. The MHT algorithm fared very well in scenarios with lots of occlusion. During 160 frames it only failed one time. The MHT tracker was compared to a tracker that used a simple global nearest neighbor association algorithm that always started a new tracklet when an object went into occlusion.

In [10] the authors develop a tracking module that uses laser range finders as sensors. The module was developed for the DARPA grand urban challenge which is a competition held by the Defense Advanced Research Projects Agency in which teams that have developed autonomous vehicles compete for the first place. Their entry, called Junior, came in second place in the 2007 urban grand challenge.
The authors use a Rao-Blackwellized particle filter for filtering and prediction. They also use a model-approach to their tracking model which they claim handles segmentation and association automatically. Hence, there is no need for separate modules for these processing steps.
A big emphasis is put on what is being tracked as objects. An object is modeled using it's bounding box width and length and an anchor point. This anchor point is different from the center of gravity for an object in the sense that it is revised depending on how the object is seen from the ego-vehicle. The authors claim that this approach mitigates the problem of stationary vehicles that are given a positive velocity even though they are standing still. This problem arises from the fact that the center of gravity of an object changes depending on how the object is seen from the ego-vehicle.
When evaluating the tracking module the results are very good. For example in a scenario of 1577 frames with 5911 vehicles, the module correctly detects and tracks 5676 of these and falsely detects 205 vehicles. This yields a 96% true positive detection rate.

And since the module needs three frames before any vehicle can be detected and started being tracked the theoretical maximum true positive rate is 97.8%. Hence, the module is very close in performance to the theoretical maximum.
The downside of the module is that it only models and tracks vehicles, not pedestrians or bicyclists for example.

Finally, Kalyan et al. [9] develops a detection and tracking module using sensor data from a Velodyne LIDAR. The authors transform the LIDAR scan into a range image around the ego-vehicle in which they then segment out pedestrians that are going to be tracked. They use a probability hypothesis density filter, PHD, that is based on finite set statistics given in [18]. The authors claim that by using finite set statistics they can incorporate data association into the Bayesian filtering framework. The constant velocity linear motion is used as the motion model, and the noise in the system model is assumed to be zero-mean Gaussian.
The authors did not include any measurable performance values in the paper. Instead they evaluated the module visually by checking the trajectories of a number of pedestrians that move in front of a stationary ego-vehicle. Hence it is hard to determine the performance of the tracking module. The authors claim that the tracking module *"effectively tracks pedestrians from noisy Velodyne HDL scans"*[9].

3

Data and methods

For verification of the tracking algorithm implemented, a third-party dataset was used. In the following chapter this dataset is presented along with a programming library that aided in Point Cloud calculations and visualizations. Lastly, the chosen method and details on the implementation of the tracking-module are presented.

3.1 KITTI Vision Benchmark Suite

The KITTI Vision benchmark suite is a collaboration between Karlsruhe Institute of Technology and Toyota Institute at Chicago[2]. It is a project that is based on autonomous driving and computer vision. The responsible team working with KITTI has previously competed in the DARPA Grand Challenge for autonomous cars with their submission "AnnieWAY". The Darpa Grand Challenge which was held year 2007 was a competition organized by the Defense Advanced Research Projects Agency in the United States. Several teams consisting of different universities and companies competed in creating the best and safest autonomous vehicles.
The KITTI Vision Benchmark suite is based on team AnnieWAY's submission and contains roughly 100 different real-world scenarios in traffic. These scenarios were recorded using state-of-the-art equipment such as a Velodyne HDL-64E LIDAR[1] and Oxts RTRange GPS/Gyroscope[19] combination for ground-truth data. They also recorded the video streams from two high-resolution color and gray scale cameras.
The goal of KITTI is to provide a real-world benchmark suite for developers to test their methods and algorithms on. They provide all raw data from their equipped vehicle and manually extract ground-truth data for verification of different point cloud processing techniques such as object detection, object tracking and road extraction.
For this project the object tracking part of the benchmark suite was used along with raw point cloud data and ego-trajectory data from the GPS/Gyroscope-combination.

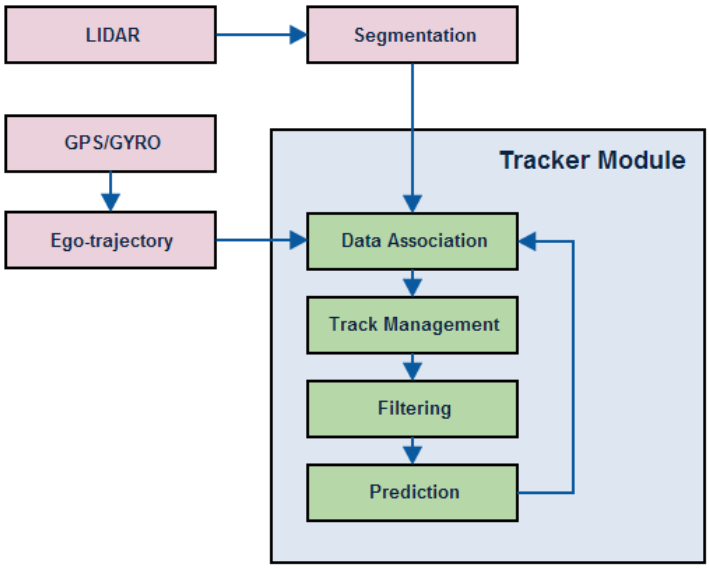
3.2 PCL Point Cloud Library

For the manipulation of raw point cloud data and visualization, Point Cloud Library, PCL, was used. This is a standalone open project library in which developers can create and add processing routines for point cloud data. It consists of hundreds of contributions from developers around the world working with computer vision and point cloud processing.
The main parts of PCL that were used in this project include the visualizer for point clouds, euclidean segmentation routine, octree-structure for voxel-representation in segmentation, and standard input/output-routines for point clouds. For more information regarding PCL the reader is referred to [3].

3.3 Tracking module

As mentioned in the beginning of this report a standard Bayesian tracking framework is used in this project for the tracking module. This means that it consists of four parts: filtering, prediction, data association and track management. Since this project is meant to run off-line i.e. not in real-time, a strategy to further increase the accuracy of the tracking module is to simply track both forward and backwards and then to merge these two results in a best outcome. This strategy was implemented and is explained in the end of this section.
Figure 3.1 gives an overview of the standalone tracking module.

Figure 3.1: Flowchart of the implemented tracking module.



Two important sources of data come from the segmentation-algorithm and the ego-trajectory data.
The segmentation yields measurements at each time step in the form of centroids (center of gravity) of extracted objects, but it can also deliver a lot more data on segmented objects such as the size of an object and the points making up the object. For more info on the segmentation see Appendix A.
The data about the ego-trajectory of the vehicle is crucial when tracking objects on the move. The tracker has to know how the ego-vehicle moved between time steps to accurately track dynamic objects. For this project the ego-trajectory comes from a very accurate GPS and Gyroscope, RT3000 from Oxford Technical Solutions [19]. The GPS/Gyroscope-combination delivers several measurements that are important to calculate the ego-trajectory such as positional data, velocity, acceleration, heading, pitch, roll etc. This data is then used to calculate rotational matrices and translation vectors that accurately describe ego-vehicle motion between time steps.
In the tracking module all measurements given by the segmentation algorithm are transformed to an absolute coordinate system using rotations and translations. This means that the tracking occurs in a fixed absolute coordinate system in which the ego-vehicle moves.

3.3.1 Filtering and Prediction

A linear Kalman Filter was chosen as the Bayesian filter for the tracking-module.
A tracked object's state vector was chosen as:

$$\mathbf{x}_k^i = [x_k^i, y_k^i, z_k^i, \dot{x}_k^i, \dot{y}_k^i, \dot{y}_k^i, \ddot{x}_k^i, \ddot{y}_k^i, \ddot{z}_k^i]_k^T.$$

Where $\dot{x}$ and $\ddot{x}$ are the representation for velocity and acceleration. This way a lot of information of an object's trajectory can be extracted.
For the motion model the choice was *Constant Acceleration*. This means that we assume the acceleration to be a process with independent zero-mean Gaussian increments with constant variance.
The above setup yields the following model given the Bayesian tracking framework pre-

sented in the theory chapter. Equation (2.13) becomes:

$$\mathbf{x}_k = \begin{pmatrix} 1 & 0 & 0 & T_s & 0 & 0 & T_s^2/2 & 0 & 0 \\ 0 & 1 & 0 & 0 & T_s & 0 & 0 & T_s^2/2 & 0 \\ 0 & 0 & 1 & 0 & 0 & T_s & 0 & 0 & T_s^2/2 \\ 0 & 0 & 0 & 1 & 0 & 0 & T_s & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & T_s & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & T_s \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \mathbf{x}_{k-1} + \begin{pmatrix} T_s^2/2 & 0 & 0 \\ 0 & T_s^2/2 & 0 \\ 0 & 0 & T_s^2/2 \\ T_s & 0 & 0 \\ 0 & T_s & 0 \\ 0 & 0 & T_s \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \mathbf{v}_{k-1}, \tag{3.1}$$

where T_s is the sampling period of the sensor. For a Velodyne LIDAR with 10 Hz update rate this is equates to 0.1 seconds[1].
The measurements that are acquired from the segmentation routine are position vectors, the centroids (center of gravity) of extracted objects. In fact, the segmentation delivers every point that makes up the object and also an oriented bounding box for the segmented object. For tracking, only the centroids are used.
Equation (2.14) then becomes:

$$\mathbf{y}_k = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \mathbf{x}_k + \mathbf{w}_k \tag{3.2}$$

The noise terms $\mathbf{v}_{k-1}$ and $\mathbf{w}_k$ are assumed to be zero-mean Gaussian. Different covariance matrices were tried out and visually evaluated for the noise terms. The final matrices were:

$$\mathbf{Q}_{k-1} = \begin{pmatrix} \sigma^2 T_s^4/4 & 0 & 0 & \sigma^2 T_s^3/2 & 0 & 0 & \sigma^2 T_s^2/2 & 0 & 0 \\ 0 & \sigma^2 T_s^4/4 & 0 & 0 & \sigma^2 T_s^3/2 & 0 & 0 & \sigma^2 T_s^2/2 & 0 \\ 0 & 0 & \sigma^2 T_s^4/4 & 0 & 0 & \sigma^2 T_s^3/2 & 0 & 0 & \sigma^2 T_s^2/2 \\ \sigma^2 T_s^3/2 & 0 & 0 & \sigma^2 T_s^2 & 0 & 0 & \sigma^2 T_s & 0 & 0 \\ 0 & \sigma^2 T_s^3/2 & 0 & 0 & \sigma^2 T_s^2 & 0 & 0 & \sigma^2 T_s & 0 \\ 0 & 0 & \sigma^2 T_s^3/2 & 0 & 0 & \sigma^2 T_s^2 & 0 & 0 & \sigma^2 T_s \\ \sigma^2 T_s^2/2 & 0 & 0 & \sigma^2 T_s & 0 & 0 & \sigma^2 & 0 & 0 \\ 0 & \sigma^2 T_s^2/2 & 0 & 0 & \sigma^2 T_s & 0 & 0 & \sigma^2 & 0 \\ 0 & 0 & \sigma^2 T_s^2/2 & 0 & 0 & \sigma^2 T_s & 0 & 0 & \sigma^2 \end{pmatrix} \tag{3.3}$$

where σ^2 was set to be 0.1 by investigating the positional error different values of σ^2 gave, and

$$\mathbf{R}_{k-1} = \begin{pmatrix} 0.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 0.5 \end{pmatrix} \tag{3.4}$$

The initial $\mathbf{P}_0$ matrix was also found by testing and evaluating:

$$\mathbf{P}_0 = \begin{pmatrix} \sigma_0^2 T_s^4/4 & 0 & 0 & \sigma_0^2 T_s^3/2 & 0 & 0 & \sigma_0^2 T_s^2/2 & 0 & 0 \\ 0 & \sigma_0^2 T_s^4/4 & 0 & 0 & \sigma_0^2 T_s^3/2 & 0 & 0 & \sigma_0^2 T_s^2/2 & 0 \\ 0 & 0 & \sigma_0^2 T_s^4/4 & 0 & 0 & \sigma_0^2 T_s^3/2 & 0 & 0 & \sigma_0^2 T_s^2/2 \\ \sigma_0^2 T_s^3/2 & 0 & 0 & \sigma_0^2 T_s^2 & 0 & 0 & \sigma_0^2 T_s & 0 & 0 \\ 0 & \sigma_0^2 T_s^3/2 & 0 & 0 & \sigma_0^2 T_s^2 & 0 & 0 & \sigma_0^2 T_s & 0 \\ 0 & 0 & \sigma_0^2 T_s^3/2 & 0 & 0 & \sigma_0^2 T_s^2 & 0 & 0 & \sigma_0^2 T_s \\ \sigma_0^2 T_s^2/2 & 0 & 0 & \sigma_0^2 T_s & 0 & 0 & \sigma_0^2 & 0 & 0 \\ 0 & \sigma_0^2 T_s^2/2 & 0 & 0 & \sigma_0^2 T_s & 0 & 0 & \sigma_0^2 & 0 \\ 0 & 0 & \sigma_0^2 T_s^2/2 & 0 & 0 & \sigma_0^2 T_s & 0 & 0 & \sigma_0^2 \end{pmatrix} \tag{3.5}$$

where σ_0^2 was set to be 0.1 by investigating which σ_0^2 gave the most stable and accurate positional estimations.
The above setup determines the Kalman Filter and the **Prediction** and **State update** steps are performed according to eq. (2.18) and (2.11), respectively.

3.3.2 Data Association

When associating new measurements to existing tracklets, a simple global nearest neighbor technique was chosen. This kept computation time down and gave satisfying results. Below is a mock-up of the association algorithm:

Data: Current tracklets, New measurements
Result: Associated measurements, non-associated measurements, non-associated tracklets
for *All tracklets predicted positions* **do**
  Calculate euclidean distance to all measurements;
  if *dist(tracklet -> measurement)<gate(individual)* **then**
    add tracklet association to sorted Queue(ascending wrt distance);
  end
end
while *queue not empty* **do**
  Save association for first entry in Queue(tracklet T1 and measurement M1);
  Add to associated measurements/tracklets;
  for *Queue* **do**
    if *Element in Queue contains T1 OR M1* **then**
      Remove from Queue
    end
  end
  Sort(Queue);
end
Save all non-associated measures;
Save all non-associated tracklets;

Algorithm 1: Data Association

This is a simplified outline of the implemented association-routine.
One important aspect of the method is that one can choose an individual gate for each tracklet which can depend on current speed, uncertainty and other important factors. The gating then eliminates a lot of the computational burden of associating correctly between tracklets and measurements.
An assumption that each tracklet can only give rise to one measurement is made. Hence, when a tracklet has been associated with it's best match, both the tracklet and the measurement can't be associated with something else(they are removed from the queue).

3.3.3 Track Management

Following the data association routine, the track manager has to update it's records of current tracklets and initiate/delete tracklets.
There are two levels of tracklets:

- Possible tracklets
- Validated tracklets

Possible tracklets are those objects that the tracker has just started to track. These tracklets are uncertain and they require two consecutive measurement associations before they even get a speed and acceleration. Hence possible tracklets get a larger gate in the beginning due to the uncertainty.
If a possible tracklet acquires M consecutive measurement associations it will be upgraded to a validated tracklet. M was chosen by trial to be 6 by investigating which value would give the most stable tracklets and at the same time ignoring irrelevant objects. The best value of M could vary between different scenarios but 6 seemed to give good overall results. A validated tracklet is a confirmed object that is being tracked and will end up in the final list of tracked objects when a scenario is complete. A validated tracklet has a higher certainty which means that a smaller gate can be imposed since the object trajectory is more accurate.
A mock-up of the track manager can be found below in Alg. 2:

Data: Possible tracklets, Validated tracklets
;
Predict;
for *All tracklets(validated+possible)* **do**
  Predict state using Kalman Filter;
end
;

Association;
Associate(Validated tracklets, New measurements);
Associate(Possible tracklets, Non-associated measurements from above);
;

Validation;
for *All possible tracklets* **do**
  if *Consecutive assoc>=6* **then**
    Put possible tracklet in validated tracklets;
  end
end
;

```
**Deletion**;
for All validated tracklets do
    if Consecutive non-assoc>=12 then
        Put validated tracklet in final tracklets;
    end
end
for All possible tracklets do
    if Consecutive non-assoc>=1 then
        Delete possible tracklet;
    end
end
;

**Update state**;
for All validated tracklets do
    if Tracklet associated then
        Update State using Kalman Filter;
    else
        Use predicted position as new state;
    end
end
for All possible tracklets do
    if Tracklet non-associated then
        Use predicted position as new state;
    else if Tracklet associated AND associated meas. == 2 then
        Calculate velocity;
        Update state using measurement and calc. velocity;
    else
        Update state using Kalman Filter;

end
;

**Initialize new tracklets**;
for All non-associated measurements do
    Initialize new tracklet with zero velocity and acceleration;
end
```

Algorithm 2: Track management

One important aspect of the track manager is how a tracklet is initiated. When a measurement doesn't get associated with any existing tracklets, the track manager

creates a new tracklet for it. However, at this first step no info on velocity or acceleration is given by the measurement alone, and they will be initiated as zero. This is a problem when it is time to predict the state of this new tracklet. The predicted state will be the same as the previous one since it has zero velocity and acceleration. This makes it hard to associate this new tracklet with another measurement in the next step. As stated above this means that the initial gate for such a tracklet has to be considerably larger for the association-routine to find the tracklet's correct association. When a good association has been found inside this large gate, the velocity between the two first associated measurements/positions are calculated. This velocity will be the initial velocity estimate for this tracked object. After this step a smaller gate can be applied and the estimated trajectory will be more accurate.
In the deletion step a validated tracklet is allowed to have 12 consecutive non-associations before it is terminated and a possible tracklet is removed as soon as there is no associated measurement. These numbers were set so that a validated tracklet could still be tracked in occlusion but a possible tracklet could not. There is a trade-off between minimizing possible mismatches and being able to track in occlusion when deciding on these numbers. Different numbers of consecutive non-associations were investigated but the number 12 gave the best overall results.

3.3.4 Forward- and Backtracking

Since the tracking-module developed in this project was meant to run off-line, i.e not in real-time, the possibility of exploiting the off-line running became evident. A natural extension of an off-line tracker is to track both forward and backwards in time. Both tracking results can then be merged to create the best estimations of objects trajectories. Tracking backwards with the above tracking module only meant that we reverse the order of loops of the scenario. The velocities and accelerations were then simply adjusted by multiplying with -1 to go back to forward-time.
During the merging, tracklets are merged if they have the same associated measurements since the measurements are exactly the same for both forward- and backtracking.
A mock-up of the merging algorithm follows:

```
Data: Forward tracklets(n = # tracklets), Backward tracklets(m = # tracklets)
Result: Final tracklets
Initialize zero-matrix A with n rows and m cols.;
for All time-steps do
    for All forw. tracklets i do
        for All backw. tracklets j do
            if tracklet i and j has same measurement then
                A(i,j)+=1;
            end
        end
    end
end
while Max(A)>0 do
    Find largest element in A -> row i column k;
    if tracklet k hasn't been merged AND largest element > 0 then
        Merge tracklet i and k;
        Set row i to be all zeros in A;
        Set column k to be all zeros in A;
    end
end
```

Algorithm 3: Merging

For the merging, the states were simply added together and then divided by two after the sign of the velocities and accelerations had been multiplied by -1 from the back-tracking.

4

Results

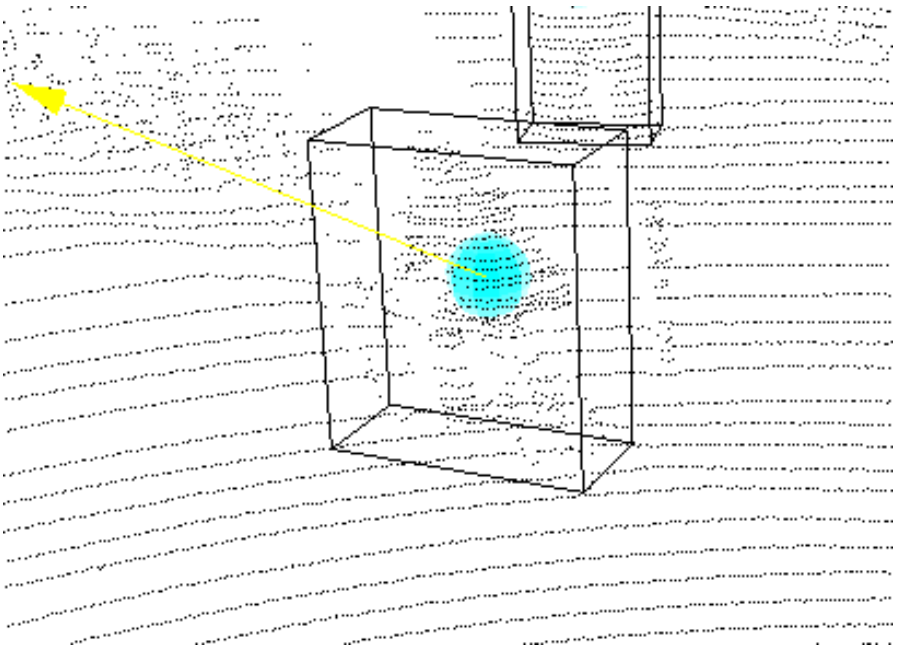
For the evaluation of the tracking module, some ground-truth is needed. The KITTI-data contains scenarios with ground-truth data, manually labeled tracklets, though not every scenario contains those. And since the goal of the KITTI-dataset is to benchmark processing techniques using only the stereo cameras, the labeled tracklets are only labeled if they are visible to the forward-facing stereo camera. Hence it was not possible to fully evaluate the tracking module in an objective manner.
There were some other drawbacks with the ground-truth data. For example, the labeled tracklets only contained position, not velocity, which meant that the error in velocity could not be evaluated. The position was always the manually found center of the object, not the centroid(center of gravity) of the points making up the object. This issue made the evaluation of positional error a bit tricky. For the evaluation purposes the tracking module was modified so that it tracked objects using the center of the bounding box that the segmentation algorithm had calculated instead of the centroid. The bounding box from the segmentation algorithm was not perfect but gave a more fair evaluation against the ground-truth data obtained from KITTI. Note, however, that tracking on the bounding box center was only done when checking the positional error of the tracker. For the other evaluation procedures the original tracker was used, since the tracking of the centroid gave more stable tracking-results.

The last two sections in this chapter contain info on the robustness of the tracker. For example an evaluation of false positives and negatives was done. However, since the KITTI-data only contained manually labeled tracklets that are visible to the camera, this evaluation is also flawed. The tracking-module developed for this project effectively tracks all objects around the vehicle, hence a lot of information is lost for this evaluation.

4.1 Position error

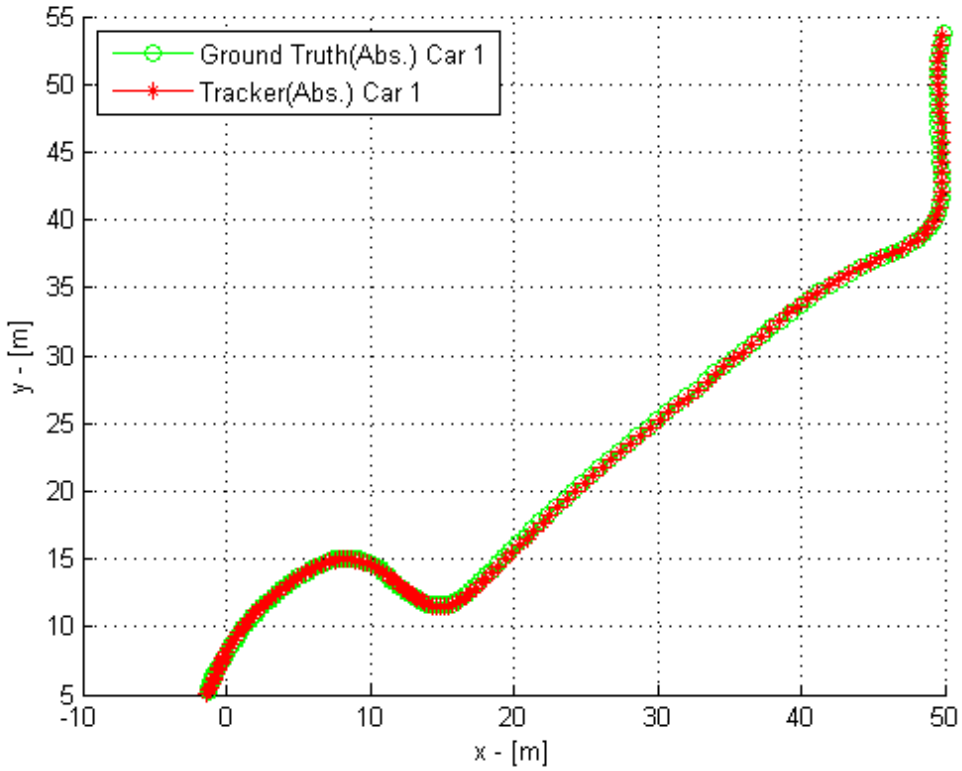
Evaluation of the positional error means that the tracker had to be modified to track using the center of the bounding box instead of the centroid, center of gravity of the points. The center of the bounding box is a more fair comparison compared to tracking on the centroid when evaluating against the labeled ground-truth data.
The scenario that was chosen from the KITTI-dataset contained several objects visible to the camera, though only one object was visible during the entire scenario. This object was a cyclist traveling in front of the ego-vehicle, shown in Figure 4.1.

Figure 4.1: Cyclist traveling in front of ego-vehicle. The cyclist is in the bounding box in the center of the figure. The sphere is the estimated center of the object.



In Figure 4.2, the position of the cyclist has been plotted in the absolute coordinate system that he is being tracked in against the ground truth.

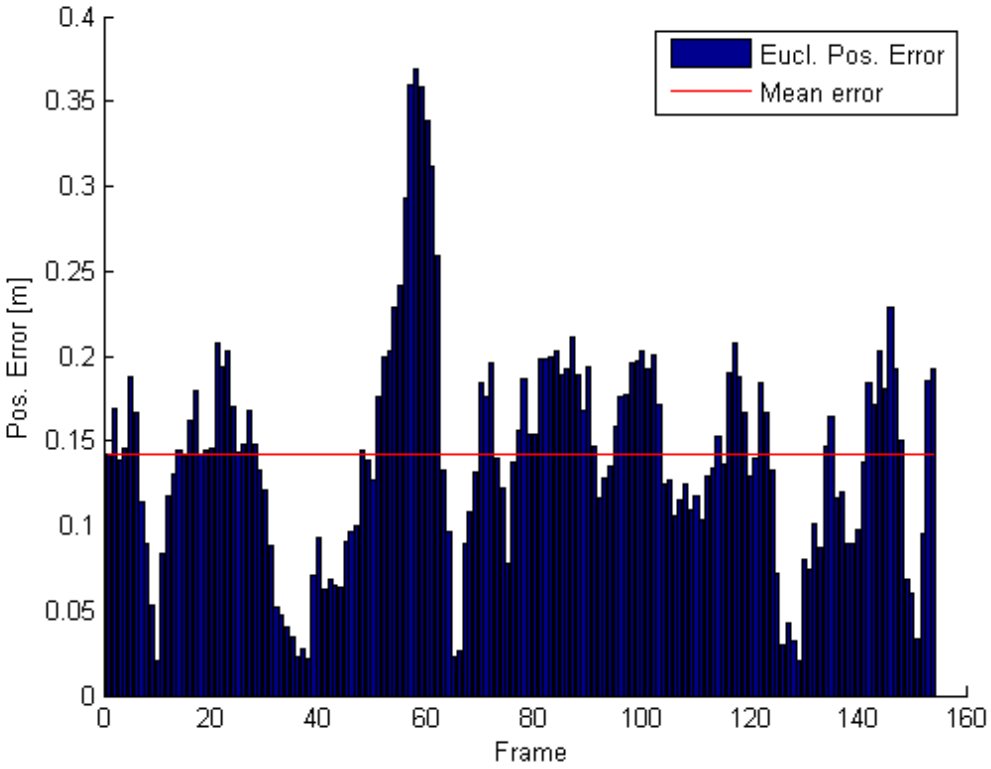
Figure 4.2: Position of cyclist in absolute coordinate system.



We can see from Figure 4.2 that the tracker's estimated trajectory and the ground truth coincide rather well. It is important to note that in this scenario both the cyclist and ego-vehicle are dynamic and move in a non-linear fashion, hence the tracker is stretched to it's limits.

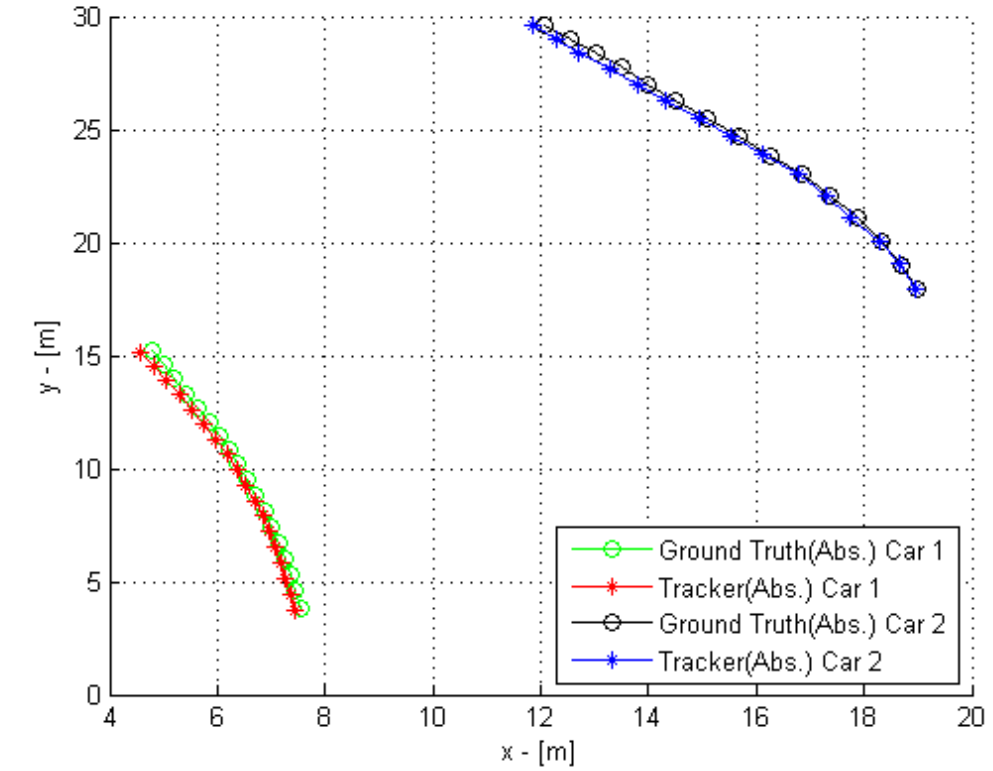
The euclidean positional error was then calculated at each time step and plotted together with the mean error, see Figure 4.3.

Figure 4.3: Positional error for the cyclist, i.e. the difference in Eulidean distance from the estimated position and the ground truth.

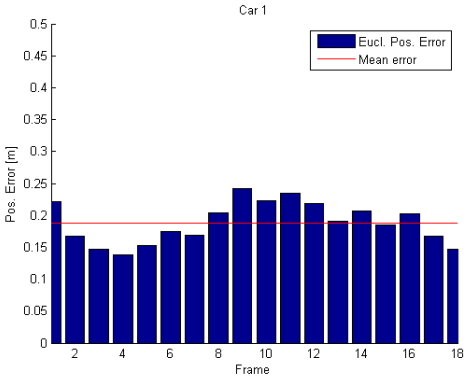


It is important to note that the tracked center of the bounding box is not always the correct center of the object. Hence the above plot is largely dependent on the segmentation result and the GPS/Gyroscope data.

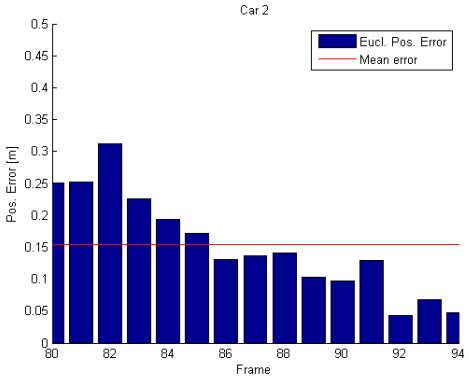
The positional error was also checked for two cars in a scenario from the KITTI-dataset. In Figure 4.4, one can see that the systematic difference in the center of the object between the tracker and the ground-truth was even more apparent because of the sizes of the cars. The tracker had been set to track on the bounding box center to be as close to the ground-truth center as possible. Though these different centers are almost never the same points. There will be a systematic error that is based on the segmentation, GPS/Gyroscope and most of all on that the ground truth doesn't "track" the same point as the tracker.



(a) Positions of vehicles in absolute coordinate system.



(b) Positional error Car 1



(c) Positional error Car 2

Figure 4.4: Positional error for two vehicles from the KITTI-dataset.

4.2 Ghosting and loss of objects

Ghosting is a name for objects that are being tracked but don't exist. Such objects are known as false positives. Conversely there are objects that exist but don't get tracked, called false negatives.
The tracking module was evaluated on 4 different scenarios, both dynamic and static. The scenarios contained multiple objects such as cars, pedestrians and cyclists. In total the 4 different scenarios spanned over roughly 60 seconds.
Since the ground truth is the same as for the error-evaluation above, only objects that are visible to the forward-facing stereo cameras are present. This meant that only a small part of the tracking module could be evaluated. Nonetheless it seemed most objective to use the ground truth data instead of visually evaluating if an object exists or not.
Another drawback is the fact that the ground truth data only contains objects such as cars, trucks, pedestrians and two-wheeled vehicles. The tracking module can track any sort of objects. Therefore an object is only counted as a miss or a ghost if it is in one of the classes that the ground truth contains. The evaluation was also restricted to objects on the road or sidewalks. Forests and vegetation give rise to a lot of possible objects that could be passed as ghost objects.
The false positives rate, FP, is calculated as the ratio between tracklets that don't exist according to the ground truth, $\#gh$, and the total number of objects in the ground truth, $\#gt$:

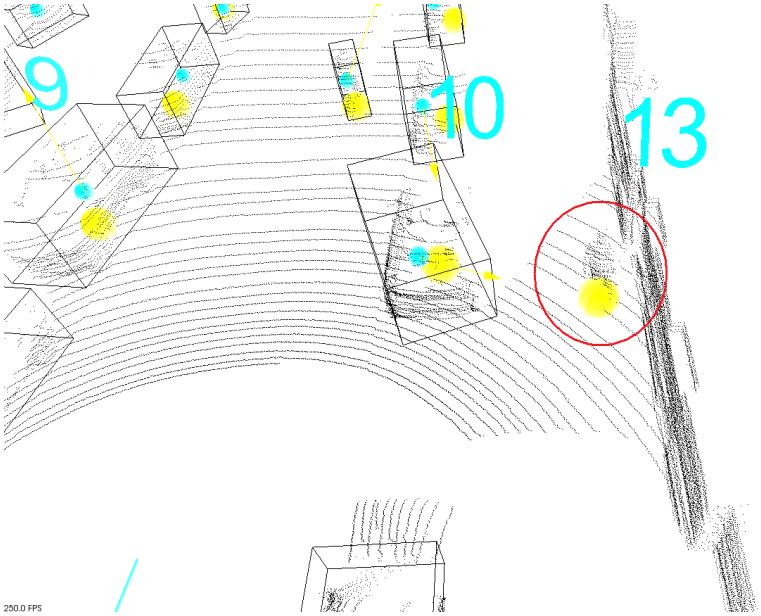
$$FP = \frac{\#gh}{\#gt}. \tag{4.1}$$

Similarly the false negative rate, FN, is calculated as the ratio between missed objects according to ground truth, $\#mo$, and the total number of objects in the ground truth, $\#gt$:

$$FN = \frac{\#mo}{\#gt}, \tag{4.2}$$

In our case the false positive rate was 0%, and the false negative rate was 2.5% for objects that had a minimum distance to the ego-vehicle of less than 40m and 7.5% for objects that had an arbitrary minimum distance to the ego-vehicle. The reason behind the two different rates is the fact that it is interesting to see the performance of the tracker at close range and at long range. However, since the ground truth data was a bit flawed it is important to mention that there were only 40 ground truth objects in these 4 different scenarios that could be evaluated. So the 0% false positive rate doesn't mean that there will never be any ghost objects, and the false negative rate will probably be reduced when averaging over more objects and with a more LIDAR-oriented ground truth instead of the camera-oriented ground truth provided by KITTI.
The objects that the tracker missed which became false negatives were objects that didn't get segmented properly. For example, the object that gave rise to the 2.5% false negative rate was a pedestrian that walked on the sidewalk too close to a wall and did not get segmented properly, see Figure 4.5.

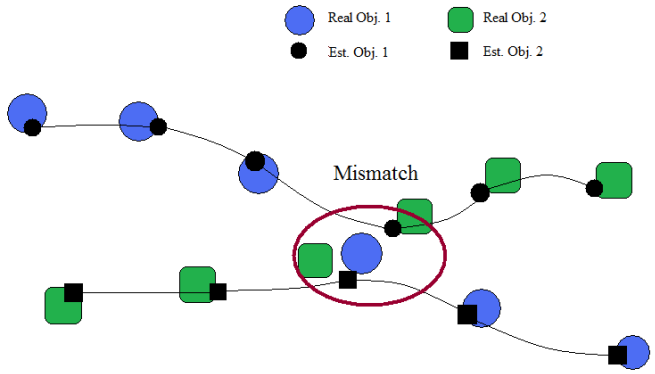
Figure 4.5: Example of false negative. The pedestrian, object 13, is too close to the wall and does not get segmented but exists in the ground truth.



4.3 Fragmentation and mismatch

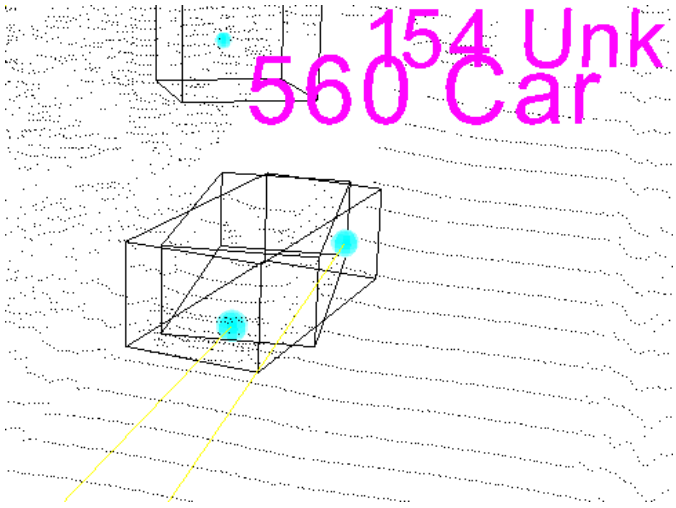
The final part of the evaluation is a measure on how continuous the tracking module is. The goal of the tracker is that trajectory of an object should be a continuous trajectory. It should not change tracking ID in the middle of the tracking procedure and track fragments of the real trajectory. Ideally, an object trajectory should only contain one fragment.
Another scenario in which a tracker can fail is when it performs a mismatch. A mismatch means that two tracklets get mixed up when being close to each other or crossing each others paths.
The illustration in Figure 4.6 explains the problem of mismatching:

Figure 4.6: Mismatching between two objects that are being tracked. Objects cross eachothers paths and the tracker makes wrong associations.



As before, both fragmentation and mismatching were evaluated using the 4 scenarios from the previous section. The result was that only one out of 40 ground-truth tracklets was fragmented. Though this was mainly due to poor segmentation, the object was oversegmented into two parts and the tracker started a new tracker for each part. When the object eventually was segmented correctly one of the two tracklets had to vanish which created the fragmentation. A visualization of the fragmented object can be seen in Figure 4.7.

Figure 4.7: Fragmentation of tracklet. The two boxes in the center of the figure are representations of the same vehicle. The spheres are the tracked position of the object for each tracklet.



The mismatching-evaluation gave a perfect result, no mismatches among the 40 objects were made.

4.4 Visual results

Below, in Figures 4.8-4.11, is a series of images of a tracked scenario with several pedestrians and other objects. There are four images which were taken one second apart, in other words every 10:th frame. The ego-vehicle is stationary at a pedestrian crossing. On the left side of the plots, several groups of pedestrians are seen both walking and standing still. The images are best seen in color.

Figure 4.8: Frame 1 of stationary pedestrian crossing.

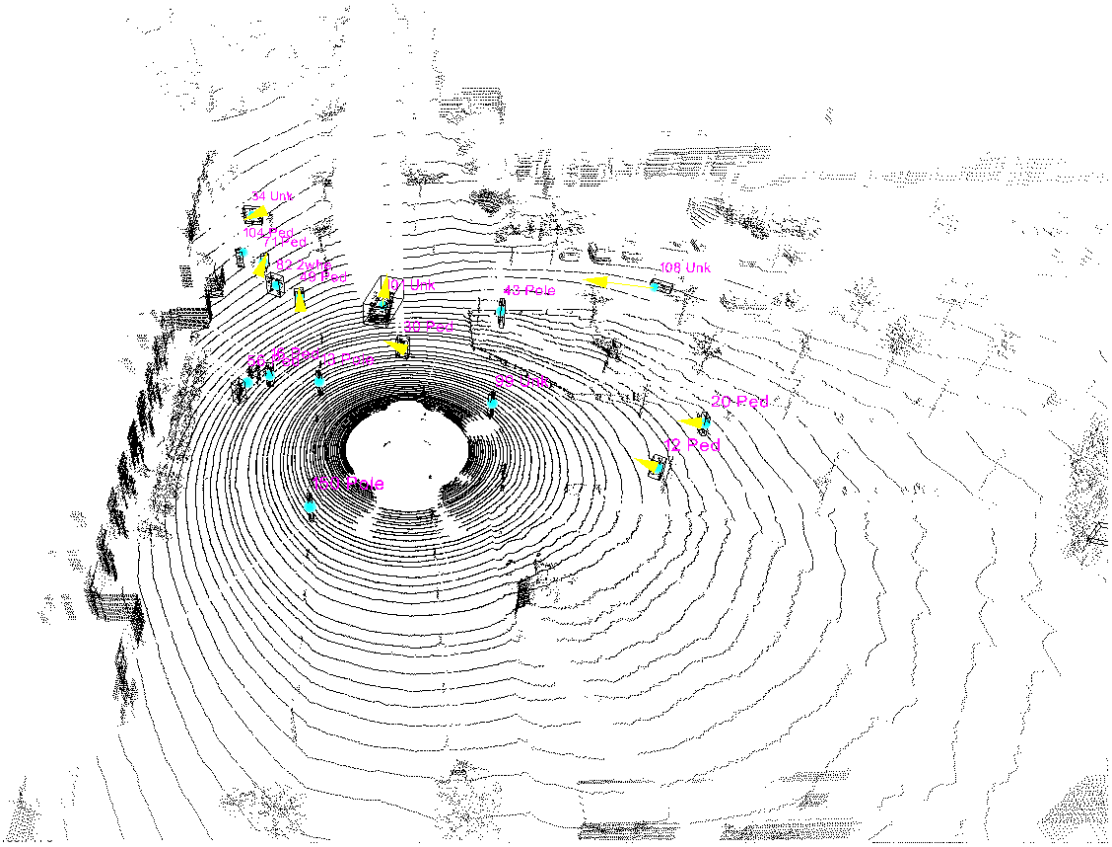


Figure 4.9: Frame 10 of stationary pedestrian crossing.

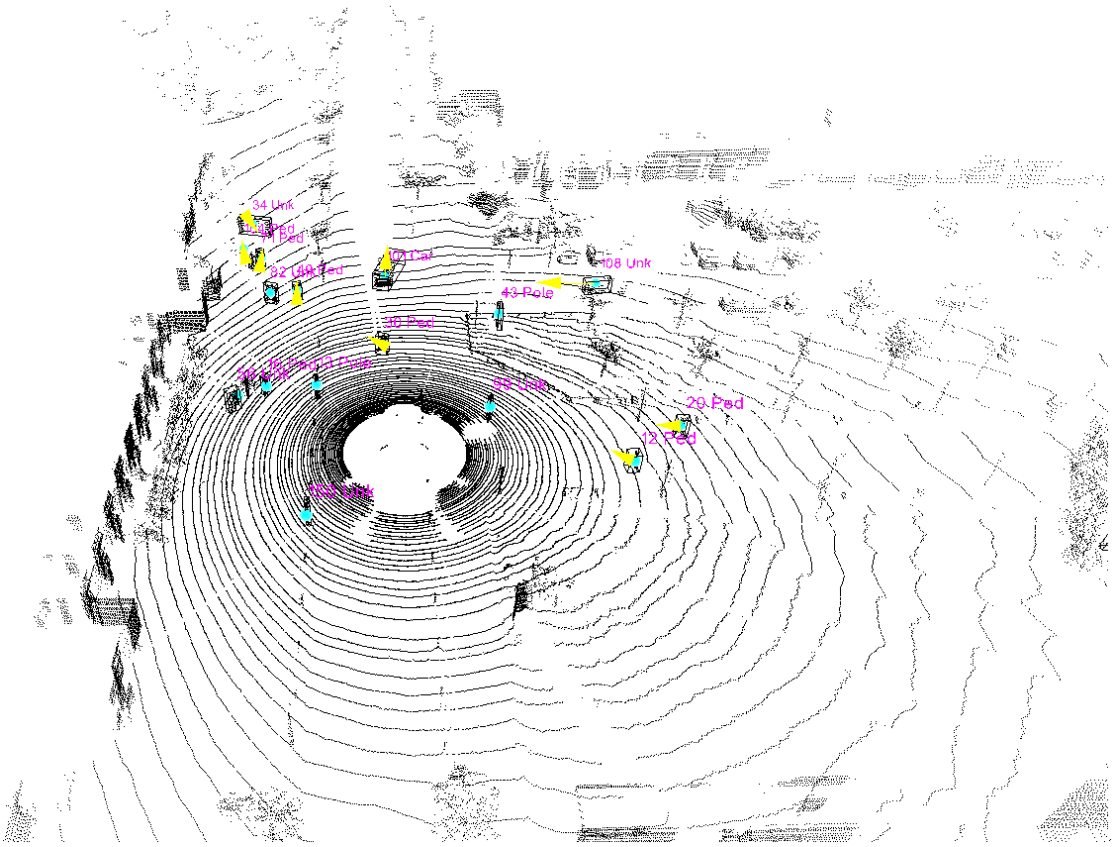


Figure 4.10: Frame 20 of stationary pedestrian crossing..

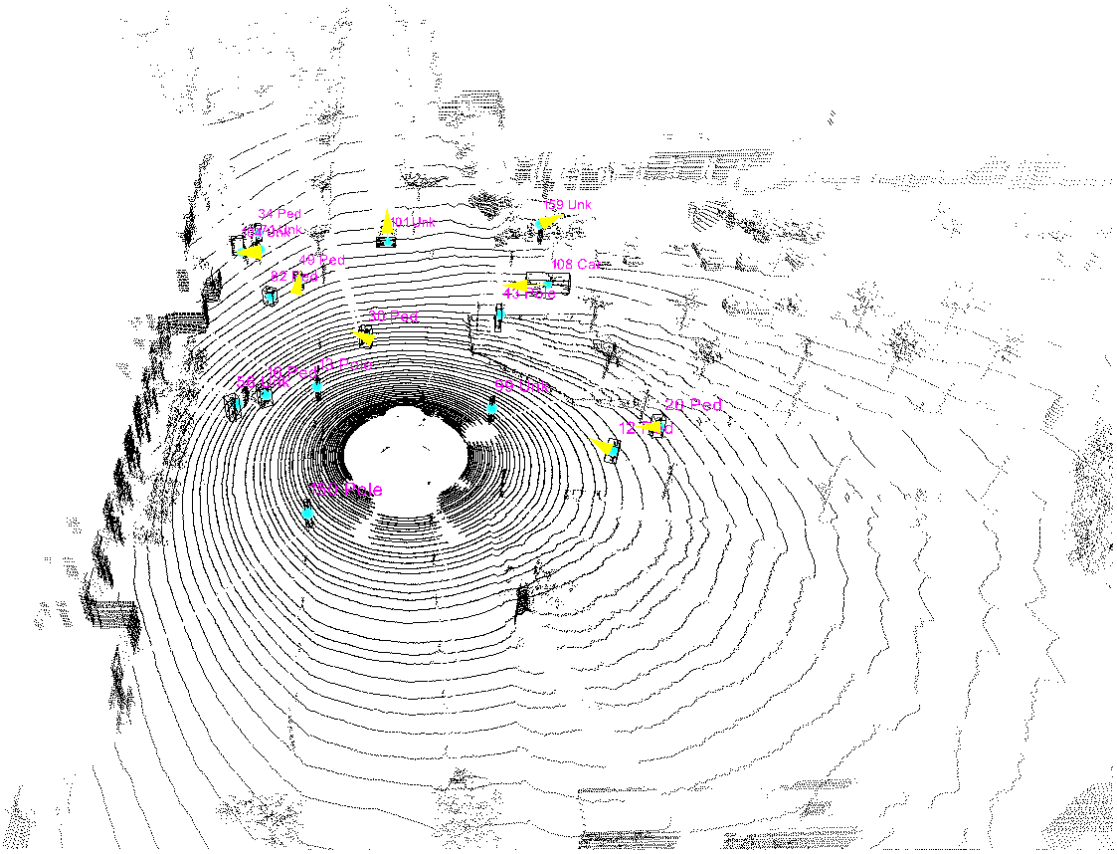
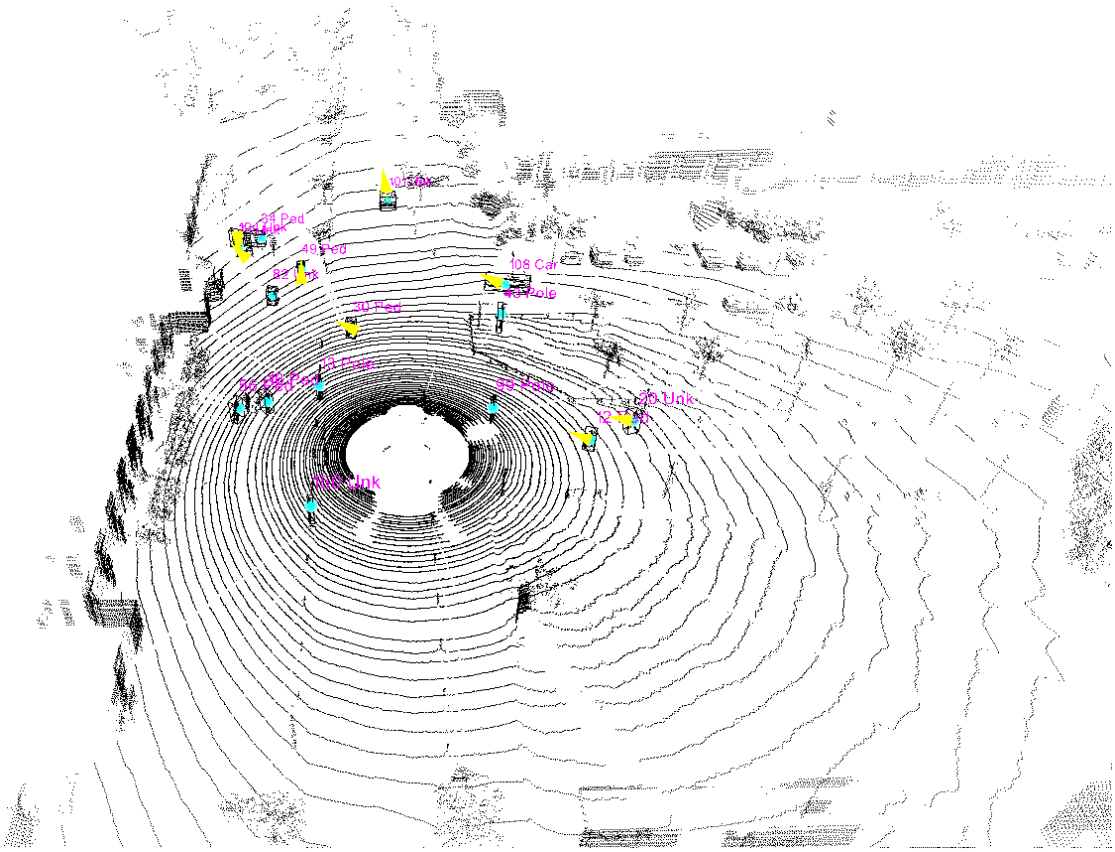


Figure 4.11: Frame 30 of stationary pedestrian crossing..



5

Discussion

The results from the evaluation were reasonably good considering the ground truth data used was flawed.
It is hard to compare the results to other literature considering that everyone uses different ground truth data. However, there exists a benchmark-metrics that can be calculated from the evaluation and then compared even though the ground truth is different. It does not offer a perfect comparison but gives a hint on how the tracker performs. The metrics was introduced in [20] and is called the *CLEAR MOT*-metrics. It involves two different metrics MOTP, multiple object tracking precision, and MOTA, multiple object tracking accuracy. MOTP is a measure on how precise the tracker is. Essentially it is the average positional error over all matches made. Since the ground truth had different centers that it tracked compared to the tracking module, a full evaluation of the MOTP-value is not done. However, for comparison purposes one can look at the mean errors calculated for the cyclist and the two cars in the results section in Figure 4.2, 4.4(b) and 4.4(c). These mean errors are calculated exactly as the MOTP-value but involve only one object each. A more realistic MOTP-value would be found by averaging over several objects.
The MOTA-value, however, can be calculated using the results from the ghosting, loss of objects and mismatches. It is defined as

$$MOTA = 1 - \frac{\#gh + \#mo + \#mm}{\#gt}, \tag{5.1}$$

where $\#gh$ is the number of ghost objects, $\#mo$ the number of missed objects and $\#mm$ the number of mismatches.
For the off-line tracker developed in this project the results gave two different MOTA scores: one for objects that had a minimum distance to the ego-vehicle less than 40m and one for objects that had an arbitrary minimum distance to the ego-vehicle.

$$MOTA(0 < \text{dist} < 40) = 1 - \frac{1 + 0 + 0}{40} = 0.975 \Rightarrow 97.5\%. \tag{5.2}$$

$$MOTA(\text{dist} > 0) = 1 - \frac{3 + 0 + 0}{40} = 0.925 \Rightarrow 92.5\%. \tag{5.3}$$

To compare this value we turn to the paper [11] where a table of both the author's tracker and trackers developed in [21] are presented. Table 5.1 includes the results from the tracker developed in this project, the off-line tracker, and the results from the trackers presented in [11, 21].

Tracker	MOTP	MOTA	FN	FP	MM
[21] BU	<0.16m	23.1%	18.7%	57.7%	—
[21] BUTD	<0.16m	89.1%	2.6%	7.6%	—
[11]	<0.14m	77.7%	8.5%	10.1%	3.6%
Off-line tracker	—	97.5%(0<dist< 40) 92.5%(0<dist)	2.5%(0<dist< 40) 7.5%(0<dist)	0%	0%

Table 5.1: Table of results for comparison.

From Table 5.1 we see that the tracking module presented in this paper is equally as good or better than the other trackers. It is, however, very important to note that the values of the other trackers in the table come from a different ground truth evaluation scenario that only involves *pedestrians*. The LIDAR was also stationary in [11, 21] which could lead to better results. A more thorough evaluation of the off-line tracker is needed, however, this requires better and more extensive ground truth data. But as mentioned earlier the values can still be compared to give an indication on how well the tracker performs.

When visually evaluating the tracker both drawbacks and benefits are spotted. One important part is how the tracker performs when objects are occluded, in shadow. By checking the different KITTI-scenarios it looks very good, this is mostly due to the fact that the track manager has a built-in routine which uses the predicted position if no association can be made. The track manager can do this for an arbitrary number of times but a cap of 12 frames was set to avoid mismatches later on. In Figure 5.1 the reader can see an example of how an object is tracked in occlusion in the KITTI-dataset.

One drawback of the tracker when objects are in occlusion, is that the velocities are not very stable. As soon as an object exists from occlusion the velocity is often changed more vigorously than otherwise. This is expected since the motion model *Constant acceleration* is not a perfect model for human driving.
Another drawback is when objects are far away and the segmentation-routine isn't able to segment the entire object. Often, the segmentation-routine can only cluster a few LIDAR-points of distant objects which leads to unstable estimates of velocity and position for the object. See Figure 5.2.

Figure 5.1: Example of object in occlusion. Box 23 creates a shadow so that no points on object 206 are visible, though object 206 is still tracked. Object 206 is a car that the tracker found before it went into occlusion.

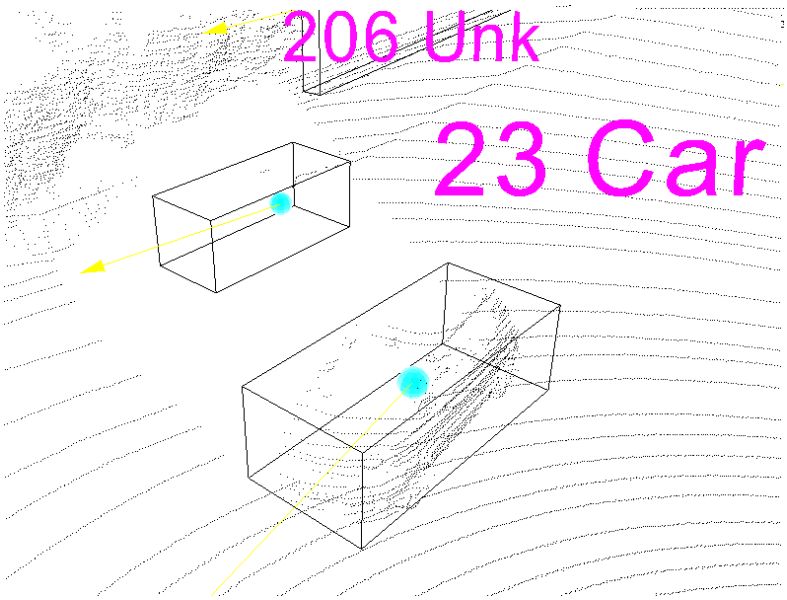
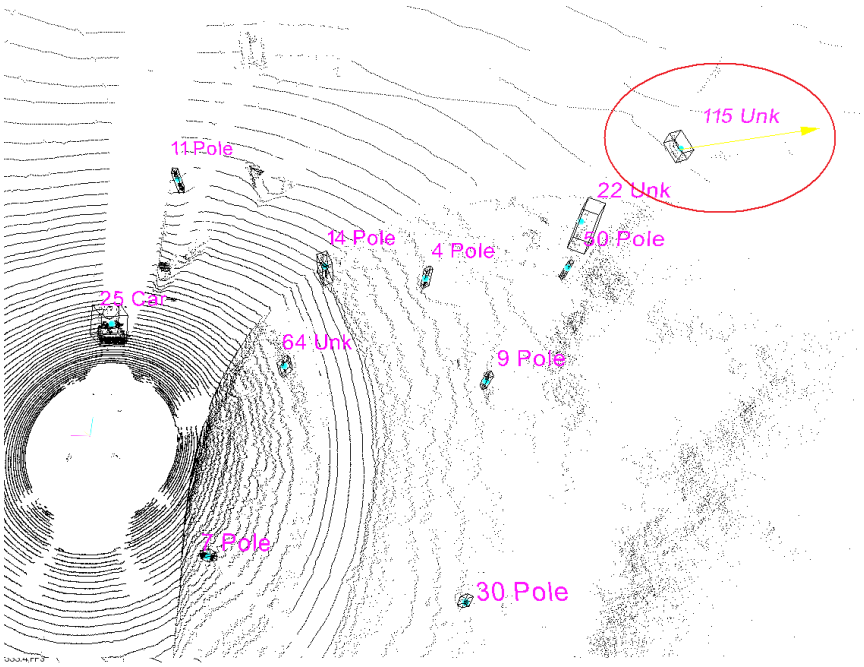
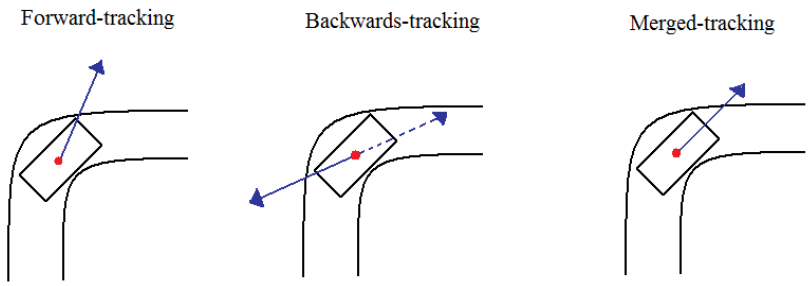


Figure 5.2: Example of an object far away. Object 115 is a car exiting the field-of-view of the LIDAR. The only thing that is visible to the LIDAR of object 115 is the back of the car.



When evaluating the impact that the forward- and backtracking had on the results, one interesting phenomenon was noted. The velocity directions of objects doing sharp turns were greatly improved when merging both forward and backward results. Since the Kalman Filter is not perfect at estimating the direction of a dynamic object, some tendencies to over- or underestimation in the direction of the object were seen. But when tracking both forward and backwards in time these over- or underestimations were canceled out, see Figure 5.3 for an illustration.

Figure 5.3: An illustration of the effect of tracking both forward and backward in time.



Overall the tracker performs well. Most of the errors and flaws come from other parts in the processing of the point clouds from the LIDAR. Often a combination of poor segmentation and flawed ego-trajectory gives rise to problems in the tracker. Considering the positional error, it doesn't matter that the tracker does not track the absolute center of each object. For verification purposes, all data can be extracted from both the tracker and the segmentation. From the segmentation, all points making up the car can be extracted. So essentially one can compare any sort of point on the car, for example the closest point on the car to the ego-vehicle can always be extracted even though that point is not being tracked.
In that sense it is more important what MOTA score the tracker performs. The stability and robustness of the tracker is more important than the precision of the tracked center of an object.

6

Conclusions and continuation

A robust off-line object-tracker was created. It was based on Bayesian tracking theory and tuned to work with LIDAR point clouds. The off-line advantage was used to track both forward and backwards in time and then merge these two results. The outcome was considerably better compared to tracking only forward in time.
The accuracy and precision of the tracker were evaluated using ground truth data from the KITTI-dataset. This ground truth was not perfect in the sense of evaluating a tracker working on LIDAR point clouds. It was more suited for camera-trackers. This made the evaluation procedure a bit flawed. Roughly 70-80% of the LIDAR field-of-view was not possible to evaluate since the ground truth only contained camera-objects. The ground truth also lacked velocity readings on objects which meant that the velocity-precision could not be evaluated. Instead this was visually evaluated. The most important information from the velocity readings are the direction an object is heading in. And from the different scenarios in the KITTI-dataset the direction of objects seemed very good. In the case of cars the direction of the velocity was consistently close to the direction of the body of the car. The direction of objects is very important for active safety systems to be able to induce safety measures to avoid collisions. It was noted that the very good result in velocity-direction was partly due to the tracking both forward and backwards in time.

When compared to other trackers in the literature both the precision and the accuracy of the tracker were equally as good or better. This was most apparent when looking at the MOTA-measure that contains information on missed object, ghosting and mismatches. The precision in position of the tracker was hard to evaluate considering the differences in what point the tracker used and what point the ground-truth used for tracking. A systematic error in position was found depending on how the object was seen by the LIDAR. Even so, the mean error in position can still be compared to for example [11] and is found to be similar in size for both the cyclist case and the two cars case evaluated in this report.

During the work on the project some problems with the original plan of the work became apparent. For starters the segmentation-routine played a more vital role than anything else for the tracker. Bad segmentation gave very bad tracking results. Therefore, a lot of attention was shifted to creating a good segmentation routine in the beginning of the project. It turned out to be one of the most important parts since it meant a simpler and more efficient tracker could be implemented.
Another problem that arose was how the ego-trajectory data would be delivered. From the start it was thought in the project group at Volvo Cars that it was possible to determine the ego-trajectory using only the point clouds. As it turned out this field of interest has not reached to the level of accuracy that is required by a robust object tracker. The KITTI-group used a GPS and Gyroscope to estimate ego-trajectory, therefore it was natural that this would be the case for the tracker in this project also. And as it turned out to be sufficiently accurate for the object tracker, a GPS and Gyroscope became the choice of ego-vehicle positioning and trajectory.

During the work on the project there were some things that could have been done differently or be researched more for the tracker. Below are some of these ideas and improvements that could be done in the future for the tracker.

- Evaluate if a more complex particle filter could outperform the Kalman Filter and the Gaussian-assumption. A particle filter could perhaps capture some of the more unpredictable manner of the trajectory of an object.
- For the motion model, a multiple model approach could be taken where for instance pedestrians and cars have different motion models. Often a pedestrian moves in a more unpredictable manner than a car and can change course in an instance. Cars are more bound to the road and traffic-rules.
- Implement a multiple hypotheses association routine. This should strengthen the performance of the tracker when objects are occluded or very close to each other.
- Investigate if there are more stable and accurate segmentation routines that have less over- and under-segmentation. Big trucks and lorries are a problem for the segmentation, if they're not seen from a good angle they are often over-segmented. Perhaps a post-segmentation after the tracker can be done to further improve the results.
- Ideally, the ego-trajectory should be calculated from the raw point clouds using stationary objects as reference. The problem is that if you need to know whether an object is stationary or not you need a tracking-result which relies on ego-trajectory data. A chicken and the egg problem arises.
- The off-line advantage could probably be exploited even more than just forward- and backtracking. One idea could be to use a sort-of global optimization routine over all frames and objects in the scenario to find the most probable associations

and trajectories. This was discussed during the project but no literature on the subject could be found and hence it would probably require more development time which was not possible during the working of the project.

- Investigate if Smoothing in the Kalman filter would lead to a better tracker. Smoothing is a technique in which future data is used to improve the present state. Since the tracker works off-line this would be possible at the end of a tracking scenario.

The above points are just some of the ideas that sparked during the working of the project but a lack of time meant they were not researched. A natural continuation would be to further investigate the above points and try to improve the original tracker. It would also be good if other sources of LIDAR point clouds could be found with more extensive ground truth data. It is very important to be able to evaluate all changes to see if they are positive or unnecessary.

Processing LIDAR point clouds is a field that has been researched and developed extensively in the recent years. LIDARs are very accurate sensors that could be used in many automotive development fields. Not only active safety in cars benefit from LIDAR technology but autonomous driving is a field that would benefit from LIDAR point cloud data. The tracker developed in this project is a very versatile tracker that could be used with any sort of sensor that produces positional measurements. The implemented standalone tracker could also be used in a real-time environment because of the efficient algorithms.

Bibliography

[1] Velodyne, Hdl-64e product description, `http://velodynelidar.com/lidar/hdlproducts/hdl64e.aspx`, acquired September 2013 (2012).

[2] A. Geiger, P. Lenz, R. Urtasun, Are we ready for autonomous driving? The KITTI vision benchmark suite, in: 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2012, pp. 3354–3361.

[3] R. B. Rusu, S. Cousins, 3D is here: Point Cloud Library (PCL), in: IEEE International Conference on Robotics and Automation (ICRA), Shanghai, China, 2011.

[4] L. Danielsson, Tracking and radar sensor modelling for automotive safety systems, Doktorsavhandlingar vid Chalmers tekniska högskola. Ny serie, no: 3064, Chalmers University of Technology, 2010.

[5] J. Gunnarsson, Models and Algorithms-with applications to vehicle tracking and frequency estimation, Doktorsavhandlingar vid Chalmers tekniska högskola. Ny serie, no: 2628, Chalmers University of Technology, 2007.

[6] B. Ristic, S. Arulampalm, N. J. Gordon, Beyond the Kalman filter: Particle filters for tracking applications, Artech House Publishers, 2004.

[7] L. Zhang, Q. Li, M. Li, Q. Mao, A. Nüchter, Multiple vehicle-like target tracking based on the velodyne lidar, in: 8th IFAC Symposium on Intelligent Autonomous Vehicles, Vol. 8 of Intelligent Autonomous Vehicles, 2013, pp. 126–131.

[8] F. Moosmann, Interlacing self-localization, moving object tracking and mapping for 3d range sensors, Ph.D. thesis, Zugl.: Karlsruhe, Karlsruher Inst. für Technologie, 2012 (2013).

[9] B. Kalyan, K. Lee, S. Wijesoma, D. Moratuwage, N. M. Patrikalakis, A random finite set based detection and tracking using 3d lidar in dynamic environments, in: Systems Man and Cybernetics (SMC), 2010 IEEE International Conference on, IEEE, 2010, pp. 2288–2292.

[10] A. Petrovskaya, S. Thrun, Model based vehicle tracking in urban environments, in: IEEE International Conference on Robotics and Automation, Workshop on Safe Navigation, Vol. 1, 2009, pp. 1–8.

[11] P. Yves, Dynamic objects tracker in 3d, Master's thesis, Swiss Federal Institute of Technology Zurich (2011).

[12] J. Shackleton, B. VanVoorst, J. Hesch, Tracking people with a 360-degree lidar, in: Advanced Video and Signal Based Surveillance (AVSS), 2010 Seventh IEEE International Conference on, IEEE, 2010, pp. 420–426.

[13] B. Habtemariam, R. Tharmarasa, T. Thayaparan, M. Mallick, T. Kirubarajan, A multiple-detection joint probabilistic data association filter, IEEE Journal of Selected Topics in Signal Processing 7 (2013) 461–471.

[14] T. Fortmann, Y. Bar-Shalom, M. Scheffe, Sonar tracking of multiple targets using joint probabilistic data association, Oceanic Engineering, IEEE Journal of 8 (3) (1983) 173–184.

[15] P. Morton, B. Douillard, J. Underwood, An evaluation of dynamic object tracking with 3d lidar, in: Proc. of the Australasian Conference on Robotics & Automation (ACRA), 2011.

[16] R. Jonker, A. Volgenant, A shortest augmenting path algorithm for dense and sparse linear assignment problems, Computing 38 (4) (1987) 325–340.

[17] I. J. Cox, S. L. Hingorani, An efficient implementation of reid's multiple hypothesis tracking algorithm and its evaluation for the purpose of visual tracking, IEEE Transactions on Pattern Analysis and Machine Intelligence 18 (2) (1996) 138–150.

[18] R. P. Mahler, Multitarget bayes filtering via first-order multitarget moments, IEEE Transactions on Aerospace and Electronic Systems 39 (4) (2003) 1152–1178.

[19] Oxford Technical Solutions RT Range, `http://www.oxts.com/products/rt-range/`, accessed: 2014-01-09.

[20] B. Keni, S. Rainer, Evaluating multiple object tracking performance: the clear mot metrics, EURASIP Journal on Image and Video Processing 2008.

[21] L. Spinello, M. Luber, K. O. Arras, Tracking people in 3d using a bottom-up top-down detector, in: Robotics and Automation (ICRA), 2011 IEEE International Conference on, IEEE, 2011, pp. 1304–1310.

A

Segmentation

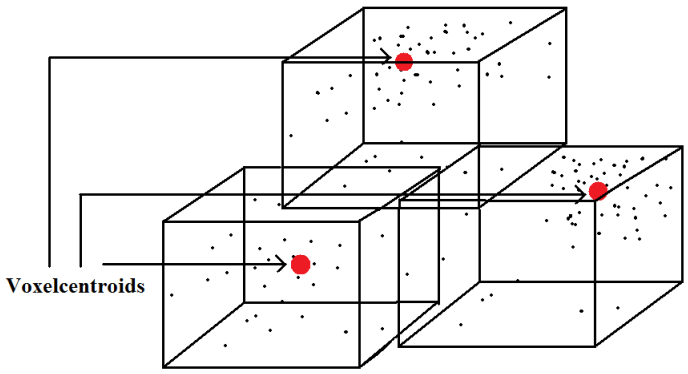
Initially the segmentation of point clouds was not a part of this master's thesis. When it became apparent that the segmentation played such a vital role in the performance of the tracker, some of the focus of the tracker was shifted to developing a good segmentation routine.
Since time was a factor, the segmentation routine had to be based on existing segmentation routines in the Point Cloud Library, PCL[3]. There exist many different implemented segmentation algorithms for point clouds in PCL. Some different ones were tried to evaluate both the quality of the outcome and the computational time. The final choice was to use one of the simplest Euclidean clustering algorithms together with a ground-plane extractor.

The segmentation routine is built up by three parts.

- Remove ground plane.
- Create a voxel-grid and cluster points using euclidean clustering.
- Cluster smaller objects using finer thresholds.

First of all, a voxel is a representation of a rectangular box containing a number of points in a point cloud. The voxel itself is the center of gravity of the points inside the rectangle. With a fixed sized voxel one can create these rectangular boxes all over the raw point cloud and create a downscaled version of the point cloud. Voxelization is a very effective way of downscaling the data while still maintaining the overall look of surfaces and objects. An example of voxelization can be seen in Figure A.1.

Figure A.1: Example of voxelization.



One key thing to remember during the voxelization is that the mapping from a point in the original point cloud into these voxels is always stored. So, if the downscaled voxel representation has been processed one can always go back to the original points in the point cloud and see what happens if they get the same treatment as the voxel they where situated in.
This means in essence that the actual segmentation can take place on the downscaled data to save computational time and then be mapped back to the original full-scale point cloud. An implementation in PCL was used for the voxel representation.

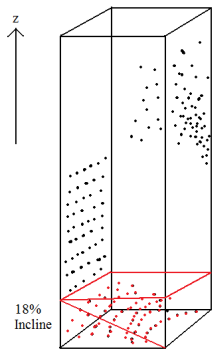
Ground-plane segmentation
Since a simple Euclidean segmentation routine is used for segmenting objects it is necessary to remove the ground plane so that the ground points don't get clustered together with objects. Not only does it help in object segmentation, it is also a way of downscaling the data. The majority of points received in the LIDAR are in fact ground plane points depending on the surroundings of the car. If the ground plane points can be removed this will also cut down the computational time for the Euclidean clustering.
The ground plane segmentation works by creating a grid over the $xy-$plane, the horizontal plane. The number of grids can be chosen arbitrarily, a good value that is used in this project is a grid of 200 by 200 "boxes". In each grid a portion of the lowest points in the $z-$axis is then removed. A mock-up of the algorithm follows:

Data: Raw point cloud data
Result: Point cloud without ground-plane
N=number of grids in each direction;
(min x, min y)=Calculate Minimum Values for Point Cloud;
(max x, max y)=Calculate Maximum Values for Point Cloud;
(Gridsize x, Gridsize y)=Calculate size of each grid using min/max of point cloud ... and N;
Create Grid Over x- and y-plane with N*N grids;
for *All points i in point cloud* **do**
  Map point i to relevant grid;
end
for *All grids j* **do**
  Calculate min z for grid j;
  for *All points i in grid j* **do**
    if *Point i.z < min z + max(Gridsize x, Gridsize y)*0.18* **then**
      Remove Point i from point cloud;
    end
  end
end

Algorithm 4: Ground plane segmentation

When removing points from each grid, the lowest $z-$valued point is used as a base. Then it is assumed that a road can have a maximum incline or decline of 18%. Figure A.2 illustrates how points are removed using this incline/decline-assumption.

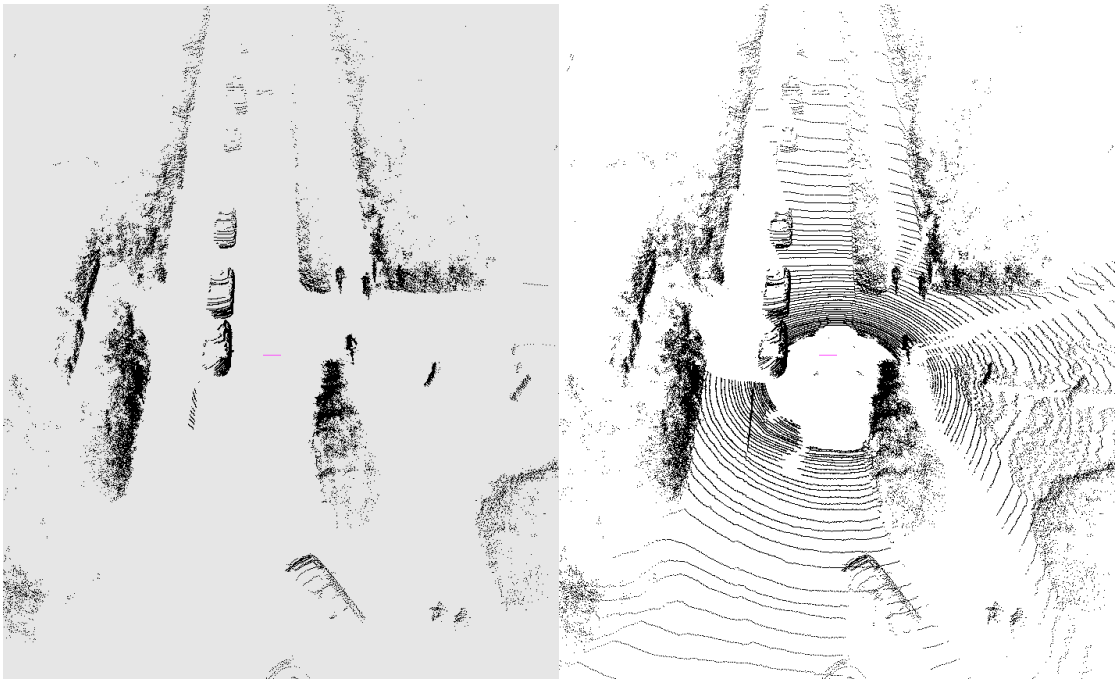
Figure A.2: Road-removal in a single grid. The points in the bottom part of the "grid-box" are removed using the incline/decline assumption.



For a large number of grids the ground-plane segmentation is very effective in removing

the road and ground plane. In some cases small parts of tires and feet get removed but this is no problem since they are such small portions that get removed. It is important to note that for smaller numbers of grids, the percentage in incline/decline of a road should be adjusted down to avoid point-losses in objects.
Figure A.3 illustrates the ground-extraction for a LIDAR point cloud.

Figure A.3: Example of road-segmentation. The right image is the original point cloud with the ground plane present. The left image is the same point cloud after the ground plane removal has been done.



Euclidean voxel-clustering
After the ground-plane has been removed, points can now be clustered into objects. As mentioned above, the first step is to voxelize the data into small rectangular boxes and downscale the data to the centers of gravity of these boxes. A routine in PCL was used for this purpose, hence the reader is referred to [3] for more information on how this is done effectively. Instead the focus will be laid on the Euclidean clustering algorithm which was also a routine that was already implemented in PCL. The goal of the algorithm is to cluster points which are very close to each other. To do this a distance threshold is imposed, d_{th}. This threshold determines at which distance points can be from one another and still be a cluster. Since the point cloud is voxelized a larger threshold can be imposed since the data is downscaled. In this project a threshold of $0.67m$ seemed to produce the best results.
A mock-up of the algorithm follows:

Data: Voxelized point cloud data
Result: Segmented clusters of original point cloud data
dth=distance threshold;
Create a KdTree-repr. to easily find nearest neighbor points of Point cloud P;
Initialize a list of empty clusters CL;
Create an empty queue for points to check;
for *All points i in P* **do**
  Add the point i to the queue Q;
  for *All points j in queue* **do**
    Extract all neighboring points Pj to point j with distance<dth;
    for *All points k in Pj* **do**
      if *Point k hasn't been processed earlier* **then**
        Add point k to queue Q;
      end
    end
  end
  All points in Q has been processed -> add points in Q to a new cluster in C;
  Set Q to be empty;
end
Map the clusters back to non-voxelized data;

Algorithm 5: Euclidean segmentation segmentation

The KdTree-representation mentioned above is just an efficient way of organizing the data structure for the points so that nearest neighbor searches can be handled efficiently. As an outcome of the segmentation we obtain clusters that include all the original points from the point cloud, i.e. the clusters have been mapped back from voxels to the original points.
There was also the possibility of imposing a maximum and minimum size of points/voxels in a cluster. The maximum and minimum sizes which were found to be working best was 5000 and 3, respectively. It is important to note that this means 5000 or 3 voxels, not points.

Finer segmentation To further improve the segmentation result, a finer clustering was done on objects that were roughly of the size of a car or smaller. The dimensions of all clusters were calculated to see which ones were smaller than a car. These objects were then put through the Euclidean segmentation again but with a smaller distance threshold and without the voxel-downscaling. The reason for this extra segmentation is that for example pedestrians that are very close to each other are very hard to segment out. To increase the chances of extracting pedestrians out of a group of people, it was necessary to use the non-voxelized data to get the best quality for the clustering. One could ask why this wasn't done in the first place, why all the hassle with downscaling?. There are two reasons for this: First the computational time is greatly reduced when

downscaling the data and the quality loss is so small that the benefits outweigh the disadvantages. The second reason is that large objects were segmented better when using a voxelized representation, and the problem of over-segmenting large objects was reduced with this downscaling.

The full segmentation routine is now running at roughly 0.3 seconds per LIDAR-frame. This is just three times the real time when the LIDAR is rotating at frequency of 10 Hz. With some extra effort put in optimizing the code it would be possible to reduce this time so that segmentation could be done in real-time.
Figure A.4 illustrates the complete segmentation routine on a LIDAR point cloud.

Figure A.4: Example of complete segmentation on a LIDAR point cloud. The image to the right is the original point cloud. The image to the left is the segmented point cloud with clusters represented by boxes and different colors.