DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Master's Thesis in Robotics, Cognition, Intelligence

Multitarget Multisensor Motion Tracking of Vehicles with Vehicle Based Multilayer 2D Laser Range Finders

Robert Lösch

DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Master's Thesis in Robotics, Cognition, Intelligence

Multitarget Multisensor Motion Tracking of Vehicles with Vehicle Based Multilayer 2D Laser Range Finders

Multi-Target-Multi-Sensor-Motion-Tracking von Fahrzeugen mit fahrzeugbasierten, mehrschichtigen 2D-Laserentfernungsmessern

Author:	Robert Lösch
Supervisor:	John Dolan
Advisor:	Prof. Matthias Althoff
Submission Date:	15.01.2017

I confirm that this master's thesis is my own work and I have documented all sources and material used.

[illegible]

Munich, 15.01.2017 Robert Lösch

Acknowledgments

This thesis summarizes my research during the last six month at the Robotics Institute (RI) of the Carnegie Mellon University. This thesis would not have been possible without my supervisor John M. Dolan who made my stay at Carnegie Mellon possible. I am grateful for his proof-reading, his constructive criticism, and advice. I would also like to thank my advisor Prof. Matthias Althoff for introducing me to Mr. Dolan and for his support.

I am also thankful for Chiyu Dong for always having an answer to all my questions and, including Wenhao Luo, for the additional test drive. I am also grateful for the other students and Ph.D. candidates for the great atmosphere in the office.

Finally, I would like to thank my parents for their enduring support which brought me into the position of writing this thesis and I would like to thank Lisa for her sympathy and understanding when I had hard times.

Abstract

For tracking, autonomous vehicles often use 3D laser range finders (LRFs), which are expensive. In order to make autonomous cars affordable as a mass product, we use multiple, more affordable 2D LRFs for tracking and implement an algorithm with the goal of achieving a similar performance. Our tracking algorithm comprises the following steps: Data preprocessing, segmentation, classification, feature extraction, data association, and track management. To compensate the information reduction, we re-use track information and focus on reducing over-segmentation and effects of the shape change problem. We test the algorithm on own labeled data and training data sets of "The KITTI Vision Benchmark Suite" with the CLEARMOT and MT/PT/ML metrics. In addition, we test specific properties with specific scenarios. Our algorithm obtains a MOT accuracy, which reflects the amount of correctly detected objects, of 0.79 on our own and negative values on the KITTI data sets. Obtained MOT precision, which is an averaged detection precision, is around 60 percent on all data sets. Out of all tracks, 47.83 percent are mostly tracked on our own data and zero percent on KITTI data sets. Our algorithm for tracking multiple objects with multiple moving 2D LRFs does not reach 3D performance. Improvements of feature extraction and especially classification to distinguish between vehicle and non-vehicle objects would boost its performance.

Contents

1 Introduction

In this thesis, we attempt to implement an algorithm to track multiple objects based on data from multiple, moving two-dimensional (2D) laser range finders (LRFs) which ideally achieves a similar performance as a three-dimensional (3D) tracking algorithm. In the following sections, we want to address why it is important to contribute to the research of 2D Light Detection and Ranging (LIDAR) tracking, what the objective of this thesis is, what its limitations are, and how it is structured.

1.1 Motivation and Problem Statement

Car manufacturers, universities, and third party service providers for autonomous driving are conducting research on how to let a vehicle drive autonomously through traffic. In recent years back to the 2007 Urban Challenge organized by the Defense Advanced Research Projects Agency (DARPA), autonomous vehicles often used 3D LIDAR combined with other sensors such as cameras (e.g. [1]) or radar (e.g. [2]). In the coming years, autonomous vehicles are supposed to be produced for the mass market as a product or a service. Unfortunately, 3D LRFs are expensive and can easily cost \$70,000 [3]. In 2014, Power and Others [4] conducted a study in which 24 percent of the respondents said that they are willing to pay up to \$3,000 for an autonomous driving mode in their next vehicle. That compares with 21 percent in 2013 and 20 percent in 2012. Therefore, the current technology cost and the market demand do not fit. The technology that is used to enable autonomous driving needs to become less expensive or be replaced with a more affordable solution. 3D LIDAR sensors may become significantly cheaper in the future [5] but estimates of when autonomous cars will become available on the market start at 2018 and fluctuate around 2020 [6] already. Until 3D LRF are cheap enough, one solution could be using multiple 2D LIDARs instead. Of course, 2D sensors provide significantly less information compared to 3D sensors. That is why the challenge is to achieve a similar performance with 2D sensors to enable autonomous driving and keep it affordable for customers. In addition to the cost factor, commonly occurring problems of tracking with (moving) LRFs, like over-segmentation or the shape change problem, are not yet solved to a satisfying degree.

1.2 Objective

Therefore, the purpose of this thesis is to implement a tracking algorithm based on the information gathered by multiple 2D LRFs. The algorithm shall be capable of tracking multiple objects, namely cars and trucks, around the vehicle and it is only allowed to use 2D LIDAR information from multiple sensors and the Global Positioning System (GPS) signal of the ego-vehicle. The vehicle and therefore the sensors are moving, and all data points which are processed are real-world data with outliers and noise. Since it is desirable that the algorithm performs similarly to a 3D LIDAR tracking algorithm, the effect of common problems like over-segmentation should be reduced. The algorithm shall be implemented in MATLAB [7].

1.3 Limitations

Since the algorithm is implemented in MATLAB, it is not necessary to prove real-time capabilities and the algorithm does not need to be implemented in C++ or run on the research platform the LIDAR points were obtained from. However, in order to get an idea of feasibility, we estimate real-time speed based on the MATLAB performance.

1.4 Structure

After compactly mentioning related work and deficiencies of current methods in chapter 2, we give an overview of the autonomous driving research platform and the used LRFs in chapter 3. Chapter 4 explains the implemented algorithm in detail. We continue with the explanation of performance testing and present the results in chapter 5. In chapter 6, we discuss the results and conclude the thesis in chapter 7.

2 Related Work

In recent years, a lot of research has focused on detection and tracking of moving objects (DATMO), especially with 3D laser measurement systems or laser measurement sensors (LMS). Before that, 2D sensors were used for tracking and detection. DATMO consists of several parts: Segmenting a scan to get objects, optionally classifying them into different classes, and extracting features of every object to finally track their movement. While some papers are about the whole DATMO algorithm, some only focus on a specific part like LIDAR clustering or feature extraction. We subsequently present related work about the whole DATMO algorithm, segmentation, classification, feature extraction, and finally summarize the mentioned deficiencies and what we want to improve.

2.1 DATMO Research

Prassler et al. [8] used only 2D LIDAR information to detect and track numerous pedestrians in crowded environments in order to navigate a mobile vehicle autonomously through them. The system detects movements due to changes in successive sequences of temporal lattice maps. In order to distinguish whether a change was caused by motion of a dynamic object or due to ego-motion, the system performs sensor motion estimation parallel to motion detection and tracking.

In [9], simultaneous localization and mapping (SLAM) creates the map which is used to detect moving objects. Wang et al. performed SLAM with DATMO to perform detection and tracking at high speeds. They segment a new scan by a simple distance threshold (D_{thd}) and transform the surrounding map first into the laser scanner's coordinate frame and second into polar coordinates. In this way, moving points are easily detectable. The system uses the interacting multiple model (IMM) algorithm to model and predict an object's motion and it uses multiple hypothesis tracking (MHT) to improve detection and data association.

The algorithm of Mendes et al. [10] implements a collision avoidance system which uses the computed time to collision (TTC) of every moving object. The DATMO part consists of three parts: First, scan segmentation, which is based on the distance between two consecutive points of the scan. To reduce over-segmentation, small and close segments are merged assuming they represent two legs of one person. Also, large lines are merged provided they might belong to the same object. Second, object classification, which uses a voting scheme of several object properties which results in a classification confidence level. Third, object tracking using a Kalman filter (KF) that allows for the object type and assumes constant velocity and white noise acceleration. In the case of ambiguous segment-track-assignments, distance, dimension, orientation, occluded time, and lifetime are used to solve the discrepancy.

MacLachlan and Mertz [11] built a system that focuses on accurate motion estimates and was used as a collision warning system. After segmenting the scan, they extracted stable features, namely right-angle corners. The subsequent data association detects spatial overlap between the predicted track and segmented rectangle. The tracker estimates the position, velocity, acceleration, and turn rate of the objects using a Kalman filter while compensating for changes due to ego-motion. When using multiple sensors, their scans are processed separately and, if necessary, merged during data association. After an extensive real-life test, the authors conclude acceptable performance for collision warning.

Premebida et al. [12] used not only 2D LIDAR but also a camera to track pedestrians and vehicles. The information from both sensors was fused to facilitate segmentation and object detection. The system classifies objects into cars or pedestrians by classifying objects separately in laser and vision space and combining them afterwards. For tracking, they used a "multi-independent Kalman filter strategy" which is performed in Cartesian space.

Multiple sensors, including LIDAR and GPS, were used by Gao and Coifman [13] to detect and track cars, pedestrians, road boundaries, or other objects in traffic. Their system groups LIDAR measurements into categories ("V", "H", "L", "O") and classifies those as vehicles or fixed objects. It tracks vehicular objects using independent one-dimensional two-state Kalman filters for vertical and horizontal movements. The classification into vehicles or fixed objects (road boundaries) is based on constructing a density image. All points are transformed into world coordinates and averaged with older scans, resulting in areas with high density caused by fixed objects. In the end, the authors suggest combining tracking and clustering to overcome the over-segmentation problem and they encountered classification problems due to oddly shaped clusters and GPS errors.

Darms et al. [2] realized that LIDAR alone does not provide enough information to track all objects around their vehicle. Therefore, they describe the obstacle detection and tracking algorithm used by "Boss", Carnegie Mellon University's winning entry in the 2007 DARPA Urban Challenge. They fused Radio Detection and Ranging (RADAR) and LIDAR to achieve both long-range detection and short-range shape estimation. However, there is no single model that can be used properly across all sensors because of different return and noise characteristics. That is why they used multiple models and proposed a novel adaptive model-switching mechanism which chooses the most informative tracking model depending on the quality of the sensors. As future work, the authors suggest focusing on improving low-level sensor details such as feature-extraction algorithms for LIDAR.

Table 2.1 shows a summary of basic characteristics of the aforementioned papers. An overview with similar structure containing more papers can be found in [14].

Table 2.1: Summary of basic criteria of DATMO papers.

Reference	Out-door	Object types			Dimension	Sensor
		Cars	Pedes-trians	Other Objects		
Prassler et al. [8]			✓		2	a
Wang et al. [9]	✓	✓			2	1 c 2 b
Mendes et al. [10]	✓	✓	✓	✓	2	a
MacLachlan et al. [11]	✓	✓	✓		2	a
Premebida et al. [12]	✓	✓	✓		2	a
Gao and Coifman [13]	✓	✓	✓	✓	2	f
Darms et al. [2]	✓	✓			2 2 3	b d e

Legend: a = SICK LMS200; b = SICK LMS291; c = SICK LMS221;
d = IBEO AlascaXT; e = Velodyne HDL-64E; f = LIDAR

2.2 Segmentation

The very first thing to handle in DATMO with LIDAR is a point cloud. In order to detect vehicles, segmentation or clustering groups points which may belong to the same object into segments or clusters such that points within one cluster are closer to each other than to points of another cluster. Based on that, vehicles can be detected and processed further by for example a classification algorithm. The following papers mostly or entirely focus on the segmentation of LIDAR points.

Borges and Aldon [15] developed a fuzzy clustering algorithm in a split-and-merge framework which they use for line and feature extraction. After comparison with the classical line tracking (LT) and iterative end-point fit (IEPF) in a simulated environment, the algorithm shows a good performance.

In [16], the same authors test their algorithm in a real environment. Their algorithm outperforms LT and IEPF again, but they conclude that robustness can only be expected to a certain level of outliers. As line extractors they developed two "breakpoint detectors": an adaptive and a KF-based detector.

Sparbert et al. [17] performed lane detection and street-type classification just using LIDAR data. They use the same segmentation approach as [18], which is based on a simple distance threshold (D_{thd}). This threshold is based on the minimum Euclidean distance between two consecutive points (see Table 2.2) and therefore takes the divergence of laser beams into account.

Adams [19] implemented an on-line algorithm which performs segmentation and feature extraction at once. It detects surface discontinuities and is based on changes of the spatial gradient considering the last three points.

The report of Premebida and Nunes [22] mentions all the authors above. Premebida and Nunes summarize and compare many segmentation and feature (primitives) extraction algorithms for 2D LIDAR points. They proposed two groups of segmentation algorithms: point-distance-based segmentation (PDBS) and Kalman-filter-based segmentation (KFBS). PDBS algorithms separate points when the (Euclidean) distance between them exceeds a threshold: **If** $D(r_i, r_{i+1}) > D_{thd}$ **then** segments are separated **else** segments are not separated, where D is distance and r is radius, i.e., the distance between the LRF and the point. Table 2.2 shows how the aforementioned papers calculate their distance threshold, with $\Delta\alpha$ being the angular resolution of the LRF and ϕ being the angle of the object. KFBS algorithms use KFs or extended Kalman

Table 2.2: PDBS Methods

Author	Remarks
Dietmayer et al. [18]	$D_{thd} = C_0 + C_1 min\{r_i, r_{i+1}\}$; $C_1 = \sqrt{2(1-\cos\Delta\alpha)} = D(r_i, r_{i+1})/r_i$; C_0 constant parameter used for noise reduction
Santos et al. [20]	$D_{thd} = C_0 + \frac{C_1 min\{r_i, r_{i+1}\}}{\cot(\phi)[\cos(\Delta\alpha/2) - \sin(\Delta\alpha/2)]}$ ϕ aims to reduce the dependency of the segmentation with respect to r; $C_{0/1}$ same as in [18]
Lee [21]	$D_{thd} = \left\lvert \frac{r_i - r_{i+1}}{r_i + r_{i+1}} \right\rvert$
Borges and Aldon [16]	$D_{thd} = r_i \frac{\sin\Delta\alpha}{\sin(\lambda - \Delta\alpha)} + \sigma_r$ λ is an auxiliary parameter; σ_r is a residual variance

filters (EKFs) to predict the next LIDAR point. Based on that prediction, the segment is separated or not according to a threshold or something similar. Table 2.3 shows the KFBS algorithms mentioned in [22].

The next two papers describe segmentation algorithms for 3D LIDAR points. Klasing et al. [23] enhance the classical k-nearest neighbor (k-NN) algorithm into a radially bounded nearest neighbor (RBNN) graph, which performs better than k-NN. Instead of connecting k neighbors, the algorithm connects all neighbors within a certain radius.

Held et al. [24] combine spatial, temporal, and semantic information and were able to achieve a more robust system and significantly reduce under-segmentations and over-segmentations while running in real-time.

General segmentation algorithms not specific to LIDAR, comparisons, and further information about clustering can be found in [25] and [26].

Table 2.3: KFBS Methods

Author	Remarks
Borges and Aldon [16]	$r_{k+1} = r_k + \Delta\alpha \frac{dr_k}{d\alpha}$ $\frac{dr_{k+1}}{d\alpha} = \frac{dr_k}{d\alpha}$
Roumeliotis and Bekey [27]	extended Kalman filter (EKF), EKF's Validation Gate
Adams [19]	EKF, based on spatial gradient $r_{k+2} = \frac{r_k r_{k+1}}{(2r_k \cos\Delta\alpha) - r_{k+1}}$

2.3 Classification

After creating a list of segments, they can be classified into different classes to simplify further processing or sort segments out. As some papers focus on segmentation, others focus on classifying clusters of LIDAR points. For example, Sparbert et al. [17], already mentioned in section 2.2, classify objects like cars, trucks, and bicycles. They accomplish this by comparing the static and dynamic states of each object with stored a priori knowledge, which reveals the object's type with a certain probability.

Nashashibi and Bargeton [28] use mainly geometrical information but also occlusion reasoning, sensor specifications, and tracking information in order to classify an object. After classifying it, the confidence level is estimated using the classification, geometrical configuration, and tracking duration.

2.4 Feature Extraction

After segmenting and classifying, feature extraction usually takes place. As already mentioned in section 2.2, Borges and Aldon [15] perform feature extraction and segmentation simultaneously. They basically perform line extraction on the whole range image. By splitting lines at characteristic points, they get their feature information and separate segments. In [16], the authors introduce two new methods for detecting line break points (see Table 2.2 and 2.3) and test their algorithm on real data.

Vandorpe et al. [29] build a dynamic map using geometrical primitives, namely lines and circles. A point is added to a line when the distance to the last added point, the distance to the line, and the change in angle between the last and the current point lie within a threshold. In addition, at least three points are necessary to form a line. If one of these conditions does not hold for at least two consecutive points, the algorithm checks whether these points form a cluster and can be represented by a circle. To add a new point to a circle cluster, its distance to the circle center has to lie within a threshold. The parameters describing the primitives are also provided with uncertainties.

2.5 Deficiencies and Suggestions for Future Research

Some authors of the aforementioned work mentioned their problems and what they want to continue researching on. Those who did are mentioned in Table 2.4

Table 2.4: Deficiencies and suggestions for future research of related work

Author	Part	Deficiency
Prassler et al. [8]	Tracking	Tracking method may lose moving object due to occlusion effects.
Prassler et al. [8]	Future work	Integrating fast matching methods to disambiguate moving objects based on their shape; incorporating a motion prediction algorithm to cope with the problems caused by occlusion or by crossover tracks; facilitating motion planning in time-varying environments
MacLachlan and Mertz [11]	Motion detection (Moving Sensor)	Object shape appears to change as different aspects of the object come into view, which can easily be misinterpreted as motion (shape change problem).
MacLachlan and Mertz [11]	Motion detection	Maintaining an occupancy grid is computationally expensive.
Premebida et al. [12]	Stability and Consistency	"Voting schemes and methods based on 'heuristic' rules have the disadvantage of not having a self-consistent mathematical framework in order to support its stability and consistency."
Premebida et al. [12]	Stability and Consistency	SICK LMS200 has poor capacity to detect cars in some circumstances in outdoor scenarios far from 15 meters.
Gao and Coifman [13]	Segmentation	Over-segmentation problem; Coping with irregular-shaped vehicles that occasionally appear.

Gao and Coifman [13]	Classification	Difficulty to accurately differentiate between vehicles and non-vehicle objects in the classification process. (partial occlusion problems, detection errors, and noise)
Darms et al. [2]	Information Acquisition	“No sensor could provide all the information necessary to track the vehicles.”
Darms et al. [2]	Future work, Feature Extraction	Improving the low-level sensor details. “More intelligent feature-extraction algorithms could help to more robustly reject spurious returns caused, e.g., by bushes.”
Borges and Aldon [16]	Future work, Robustness	Improving reliability in local map building using segmentation with complementary reasoning at a higher level.
Premebida and Nunes [22]	General Segmentation Performance	PDBS methods do not adapt dynamically.
Premebida and Nunes [22]	General Performance	Real-time capability (embedded systems)
Klasing et al. [23]	Performance, Feature Extraction	Balancing between more powerful algorithms and real-time capability; extraction of geometry

2.6 Implication

Since we attempt to achieve approximately 3D performance, we will pay particular attention to three problems. First, our algorithm shall be able to continue an interrupted track by estimating movements during scans in which the track is occluded. Second, it is often misinterpreted as motion when moving sensors see different aspects of a standing object. Third, noise, occlusion, or badly parameterized segmentation algorithms can all cause overly segmented objects. By merging segments that belong to the same object, we want to further improve tracking performance.

3 Autonomous Driving Research Platform

The LIDAR data used in this master's thesis were obtained from an autonomous driving research platform described in [30]. The subsequent sections briefly describe this platform, what kind of LIDAR sensors are used, and what kind of raw data pre-processing takes place already on the car.

3.1 Autonomous Driving Research Vehicle

The research platform is based on a 2011 Cadillac SRX. The vehicle was electronically and mechanically modified and equipped with sensors of several types to enable autonomous driving features. A more detailed description of these modifications is provided in [30]. The vehicle is, among other things, equipped with six LRFs and two GPS sensors. The vehicle and the sensor distribution are depicted in Figure 3.1. For implementing our tracking algorithm, we only use information obtained from all LIDAR sensors and the *GPS2* sensor which is an *Applanix POS LV* [31]. The six LRFs are distributed in such a way that the area all around the vehicle is covered, giving it a 360-degree coverage. The area in driving direction is covered more densely than other parts, as depicted in Figure 3.2a. Since the horizontal field of view (FoV) of the sensors is smaller than 180°, there are blind spots around the car in its immediate vicinity (Figure 3.2b). They are small enough to avoid overlooking a car but still affect tracking negatively.

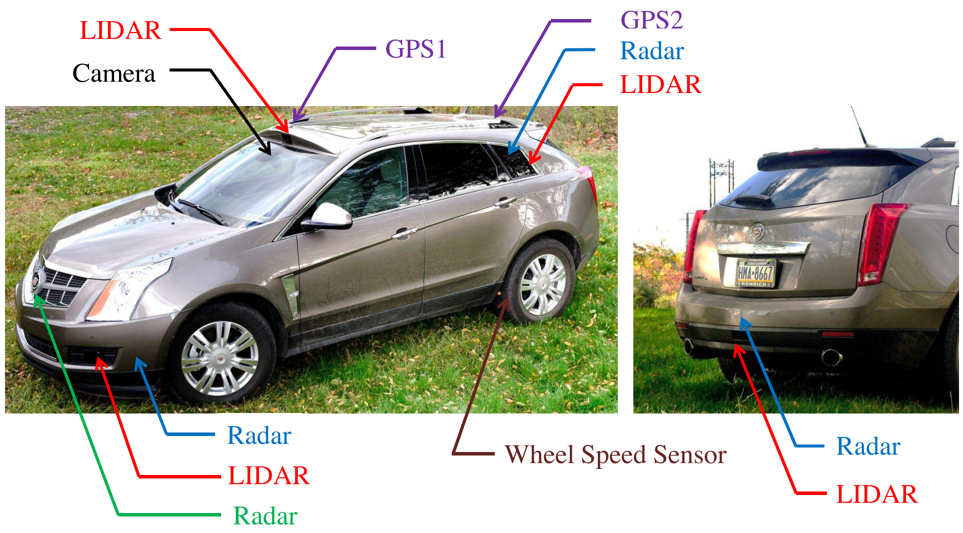
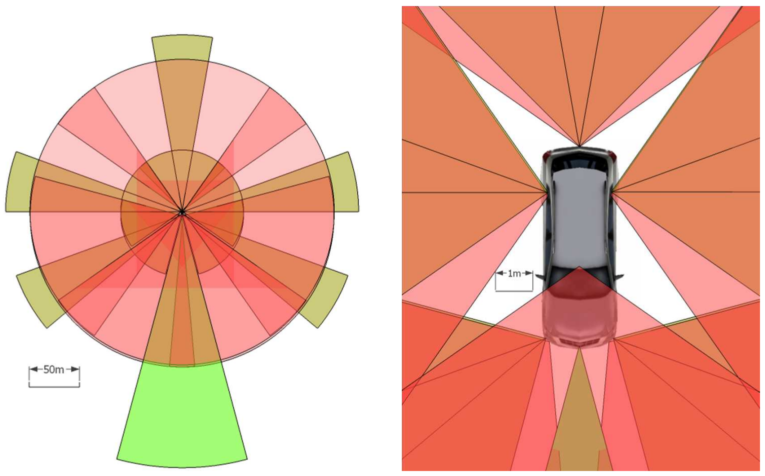


Figure 3.1: Sensor distribution (From: [30]).



(a) Maximum ranges.

(b) Sensor FoV and coverage close to car.

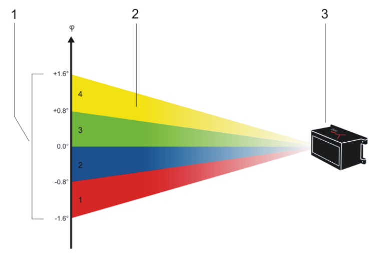
Figure 3.2: Perception system coverage (green = RADAR, red = LIDAR) (From: [30]).

3.2 LIDAR Sensors

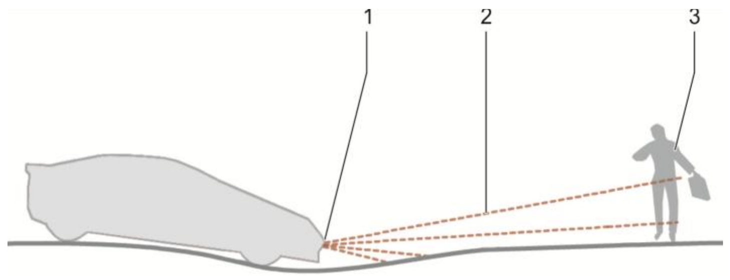
A LIDAR sensor basically consists of a LASER, which stands for light amplification by stimulated emission of radiation, and a rotating mirror. The mirror is inclined and changes the direction of the LASER towards the environment. By rotating, the mirror beams the LASER towards other objects. The targeted objects reflect the light which is received by the sensor. Based on the time of flight of the LASER beam, the distance can be calculated. In the end, the sensor provides an angle and a distance (polar coordinates) which can easily be transformed into, for example, Cartesian coordinates.

Table 3.1: Excerpt of ibeo LUX 2010 Fact Sheet [32]

Laser		
Range		200 m
Measurement		
Horizontal FoV:	2 layers:	110° (50° to −60°)
	4 layers:	85° (35° to −50°)
Vertical FoV:		3.2°
Number of parallel layers:		4
Data update rate:		50 Hz
Accuracy (distance independent):		10 cm
Angular resolution:	Horizontal:	up to 0.125°
	Vertical:	0.8°
Distance Resolution:		4 cm
Software		
Raw data pre-processing:	All measurements will be classified and tagged as valid / ground / dirt / transparent / clutter.	



(a) Four layers of ibeo LUX 2010 (From: [33]).



(b) Multiple layers compensating change of inclination (From: [33]).

Figure 3.3: Multiple layers and their purpose. (a) The four layers of ibeo LUX 2010 and vertical angular resolution, (b) the layers' purpose of compensating a change of inclination of the road.

The six LIDAR sensors which were mounted on the research vehicle are 2D LRFs of the model *ibeo LUX 2010* [33]. They consist of four LASERs and mirrors and therefore comprise four layers, as shown in Figure 3.3a. Theoretically, such a LRF delivers 3D information already. However, the four layers rather serve the purpose of compensating pitch of the vehicle or change of inclination of the road than enhance the information from 2D to 3D (Figure 3.3b). Despite the four layers, the sensors are referred to as 2D or sometimes “2D+”. The two upper layers are shifted with respect to one another about the vertical axis by half of the angular resolution. This doubles the horizotnal resolution, as depicted in Figure 3.4. In fact, the horizontal

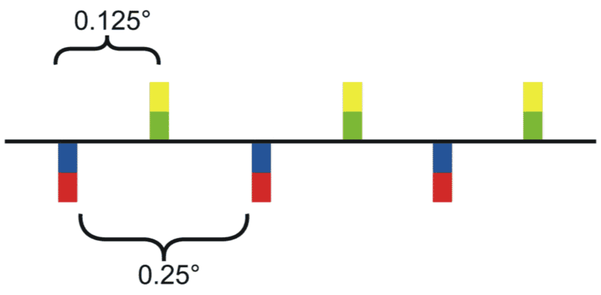
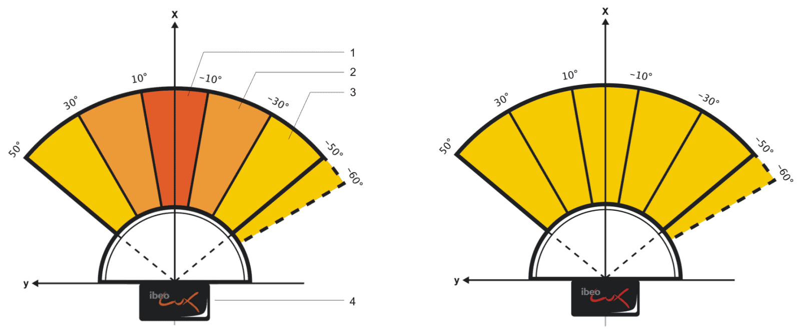


Figure 3.4: Displacement of the four layers (From: [33]).

angular resolution as well as the frequency can be adapted. Table 3.1 shows relevant technical information about the ibeo LUX 2010. The horizontal angular resolution can be adapted in certain sectors of the horizontal FoV as depicted in Figure 3.5a.



(a) Horizontal angular resolution differing by sector.

(b) Homogeneous horizontal angular resolution.

Figure 3.5: Angle resolution (From: [33]).

The resolution can be changed as depicted only with a frequency of 12.5 Hz and as shown in Table 3.3. Without different sectors, the horizontal resolution can also be changed but depends on the frequency (Figure 3.5b, Table 3.4).

Our scans are created with a frequency of 12.5 Hertz and an angular resolution like the one in Figure 3.5a.

Table 3.3: Horizontal angular resolution for different sectors for 12.5 Hz frequency (Adapted from: [33]).

Angle resolution	Central range	Medium range	Lateral range
Scan frequency 12.5 Hz	0.125°	0.25°	0.5°
Sector of Figure 3.5a	red	orange	yellow

Table 3.4: Horizontal angular resolution for different frequencies (Adapted from: [33]).

Scan frequency	12.5 Hz	25 Hz	50 Hz
Angle resolution	0.25°	0.25°	0.5°

3.3 Pre-Processing Data Points

Before the data are processed by our algorithm, they have already been pre-processed on the research platform. For one thing, all points are transformed from their sensor coordinate frame to the vehicle coordinate frame and, for another thing, the ground points are removed.

3.3.1 Sensor Fusion

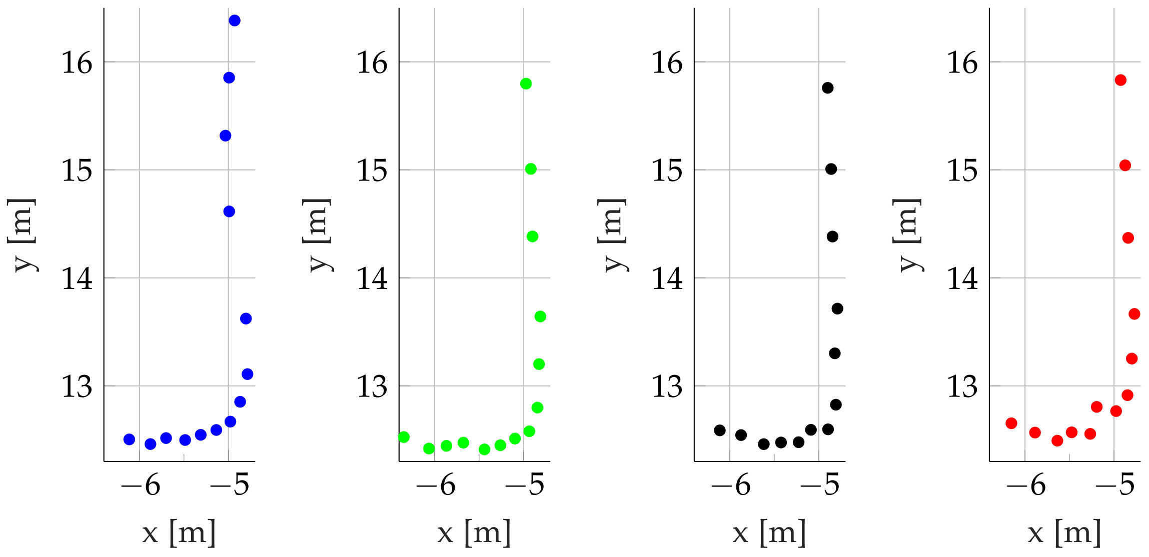
Every sensor has one coordinate system of its own. Due to the working principle of a LRF, it uses a polar coordinate system. The six sensors are mounted on the vehicle, so their position is known within the vehicle's Cartesian coordinate system. After scanning the environment, all points are transformed into this coordinate system with the ibeo's internal software. Via Ethernet, the data points can be retrieved in polar coordinates from the sensor itself or in Cartesian coordinates from the sensor's application program interface (API) [33].

3.3.2 Deleting Ground Points

As shown in Table 3.1, ground points are tagged as such already. These points are then deleted either before the tracking algorithm is performed or as one of the first steps within the algorithm.

3.4 Differences Compared to One One-Layered LRF

In order to clarify the differences between multiple or multi-layered devices with one one-layered device, we want to consider one L-shaped cluster. Figure 3.6a, b, c, and d depict layer 1, 2, 3, and 4 from one ibeo LUX 2010 of the same cluster. Due to the



(a) Layer 1. (b) Layer 2. (c) Layer 3. (d) Layer 4.

Figure 3.6: Points of one LRF reflecting the same object separated by layer.

rather thin line and clear outline, one could already guess the order of the points without looking into the metadata. With the order information, it is no problem to find the first and last point of the cluster. A third point of the cluster with the largest perpendicular distance to the connection line of the first and last point would be the corner. This corresponds to one iteration of the IEPF algorithm. Another use of the order information would be the possibility to determine the gradient from one to the next point. The largest change in direction would mark the corner (similar to [19]). However, the outlier in the bend of layer 4 would probably cause problems. If all layers are merged (Figure 3.7), the outline of the L is still clear but the line is thicker and the order of the points becomes unrecoverable. For example, starting counting at the point with the largest y-coordinate, the second point of layer 1 and the first point of all other layers are basically at the exact same spot. The order of the points becomes useless. The same problem occurs with multiple one-layered devices.

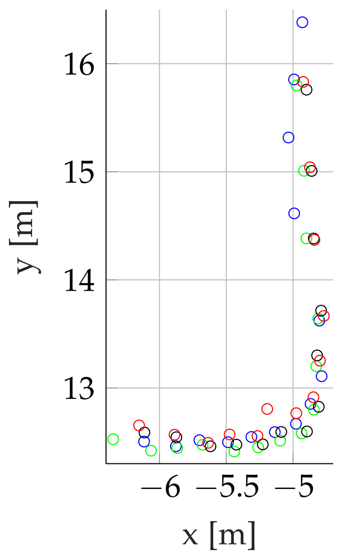


Figure 3.7: The points of all four layers of one LRF merged. Blue = Layer 1, Green = Layer 2, Black = Layer 3, Red = Layer 4.

There are two ways to work with these clusters:

- Handle each layer/device separately and merge the results afterwards (like for example [11]).
- Merge the points of all layers/devices, handle all points as one set, and relinquish order information, which is our approach.

Section 4.4.2 describes in detail how we process these points.

4 Tracking Algorithm

In order to track other vehicles, the LIDAR data generated by the LRF are simply preprocessed by a coordinate transformation and by cutting off points. Then clusters are segmented and classified using geometry and track information. After post-processing, features are extracted and post-processed using geometry and track information as well. In the end, the new scan is associated with the old scan and already known tracks. If there are new objects or some old ones are missing, track management creates new or deletes old tracks. This algorithm is depicted in Figure 4.1 and is explained in more detail in the following sections.

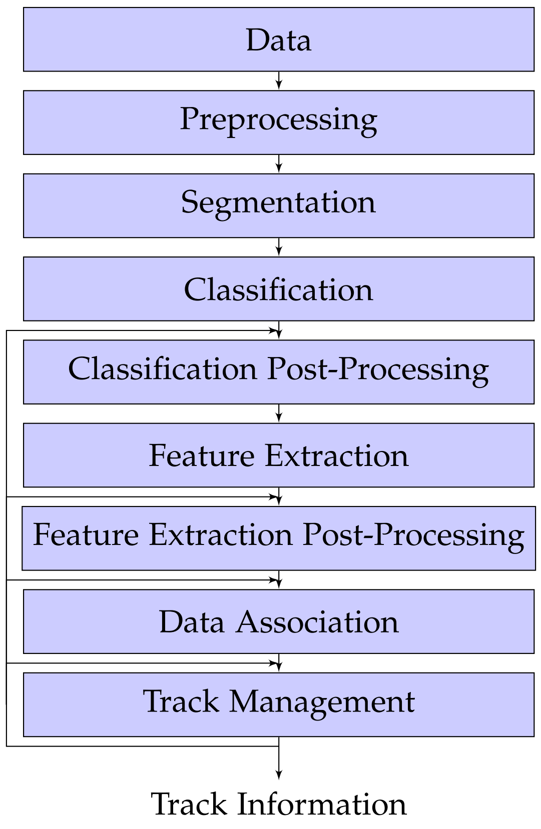
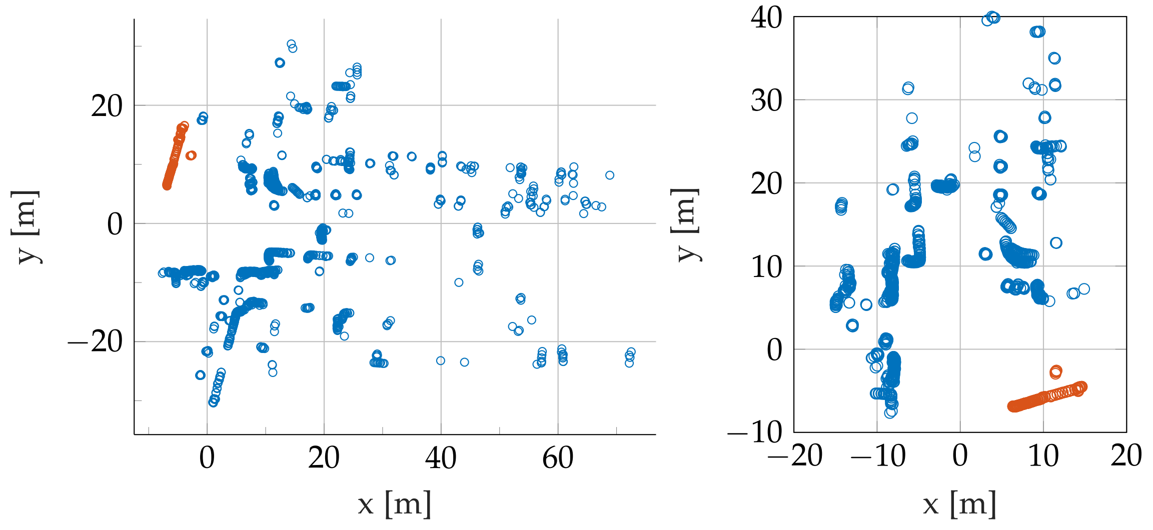


Figure 4.1: Tracking Algorithm.

4.1 Preprocessing

After the LIDAR points are received from the LRF and before the tracking algorithm starts, they can be preprocessed for many reasons. One could fix or adapt something, delete some points, or delete or add information from or to each point. We only perform a coordinate transformation and cut off points which are too far away to be considered important or a vehicle. Figure 4.2a shows all points of a scan without any processing. Figure 4.2b shows the remaining points after the coordinate transformation and the cutting off of points at the borders.



(a) Untouched points.

(b) Points after coordinate transformation and cropping.

Figure 4.2: LIDAR points (a) before and (b) after preprocessing. First, x- and y-coordinates are swapped and second, points in the left, right, and front are deleted. For better comparison, we marked a cluster that appears in both figures.

4.2 Segmentation

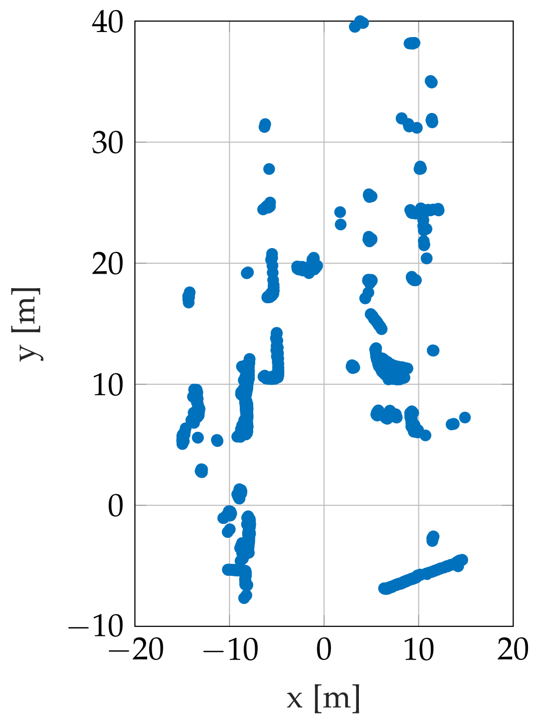
Other cars, pedestrians, or in general other objects surround the ego-vehicle. In a perfect scan, every object would cause exactly one cluster or segment of points. Therefore, the segmentation algorithm groups points which may belong to the same object into clusters or segments such that points within one cluster are closer to each other than to points of another cluster. Various distance measures could be used, but we choose Euclidean distance. Segmentation provides a list of segments which ideally represent one object each. Due to non-optimal segmentation parameters, partial occlusion, and other factors, it often happens that a segment belongs to one or more objects or that an object causes one or more segments. Of course, an object could also fail to cause a segment because it is fully occluded.

There are various segmentation algorithms. Hierarchical clustering, for example, groups points according to their distance to each other and by varying the distance threshold, other clusters are formed. Those algorithms provide a hierarchy in which the clusters are merged or split according to the chosen distance threshold. The hierarchy is often depicted using dendograms. Centroid-based clustering, like *k*-means clustering, groups points around a centroid which does not need to be a point itself. Distribution-based clustering, like Gaussian mixture models using the expectation-maximization algorithm, assume that points are caused due to a specific distribution and group points which most likely belong to the same distribution. Density-based clustering (like DBSCAN) groups points with a similar (usually high) density. Points in sparse areas are often considered noise.

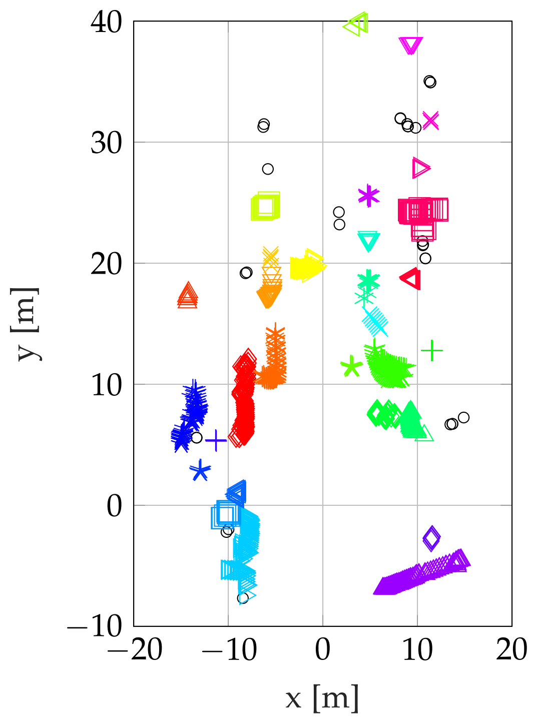
Since points close to a LRF are more dense than points with a larger distance, we considered using a density-based algorithm. Due to its speed, we use density-based spatial clustering of applications with noise (DBSCAN) [34]. The algorithm requires two parameters: a *minimum-point-* and a *distance*-threshold (D_{thd}). DBSCAN counts all neighboring points within a D_{thd}-neighborhood. If that number is smaller than the minimum-point-threshold, it is considered noise. Otherwise, it is added to the current cluster or a new cluster will be created. Figure 4.3 shows the points from Figure 4.2b after segmentation.

4.3 Classification Step

After segmenting all data points, we categorize every cluster into one of several classes according to its geometry. Further processing is easier if the shape or other characteristics are known because the clusters of different classes can be handled differently. All but one class are based on the different perspectives the LRF is



(a) LIDAR points before segmentation.



(b) LIDAR points after segmentation.

Figure 4.3: LIDAR points (a) before and (b) after segmentation. Every cluster with another color is one segment. Black points are considered noise.

perceiving from another vehicle. If a cluster is oddly shaped, it is assigned the class "O", which stands for *other*. The clusters are classified by assuming they belong to a specific class and extracting the respective features with a similar pre-fit algorithm used in feature extraction, which is described in detail in subsection 4.4.2. If the features fit and the errors are small, the cluster is assigned to the respective class. The difference from the actual feature extraction is that the classification pre-fit algorithms use linear regression in contrast to orthogonal regression. Doing this, this classification is solely based on geometry of the clusters. After classification, we take the track history into account by predicting the future position of a track and switching the class of oddly shaped clusters if they appear to occupy the anticipated space. In addition, we reduce over-segmentation by merging close clusters of appropriate classes. In the following subsections, we briefly explain linear regression and go into detail on how we perform class assignment and how the post-processing works.

4.3.1 Linear Regression

Assume we have n data points $\{(x_i, y_i), i = 1 \dots n\}$ and that y_i relates to x_i as follows:

$$y_i = \alpha + \beta x_i + \epsilon_i, \tag{4.1}$$

with α being the y-intercept, β being the slope of the regression line, and with ϵ being the residual, i.e., the difference between the y-coordinate of the measured point and the regression line (compare Figure 4.4a). The goal of linear regression [35, 36] is to find the function of the regression line and therefore α and β:

$$y = \alpha + \beta x \tag{4.2}$$

such that the sum of squared residuals (SSR) is minimized. The ordinary least squares (OLS) [37] estimation of the regression line is calculated as follows:

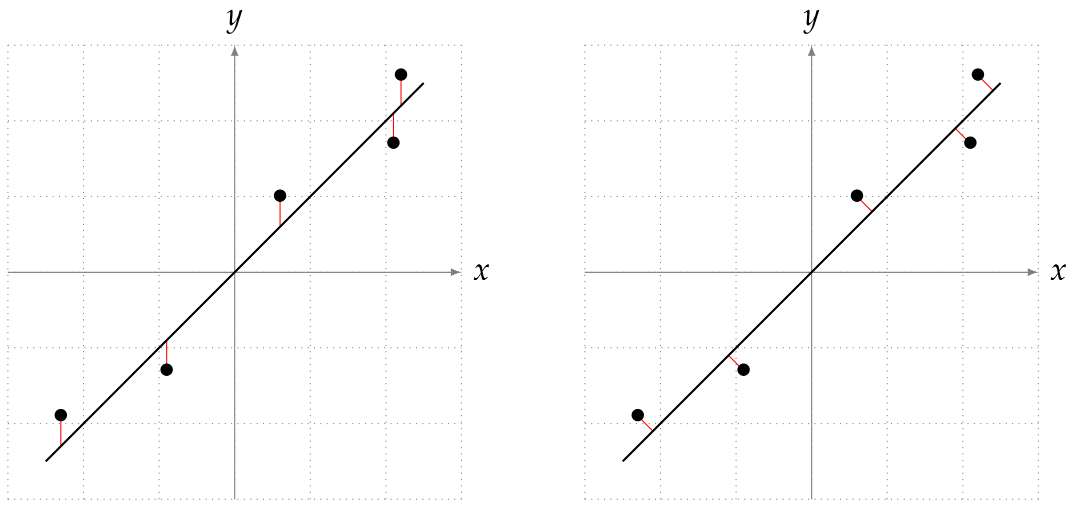
$$\underset{\alpha,\beta}{\operatorname{argmin}}\, SSR(\alpha, \beta), \tag{4.3}$$

with

$$SSR(\alpha, \beta) = \sum_{i=1}^{n} \epsilon_i^2 = \sum_{i=1}^{n} (y_i - \alpha - \beta x_i)^2. \tag{4.4}$$

4.3.2 Classification

The clusters seen in a scan reflect objects in the real world. One LRF perceives only one or two sides of a vehicle at the same time. If a cluster consists of just one side of a vehicle, the cluster looks like a line which can be horizontal, vertical, or something in between. These lines are classified as “H” or “V” depending on the angle (see Figure 4.5a and 4.5b for examples). If two sides of a car are visible, the cluster looks like an “L” which is also the assigned class (Figure 4.5c). Since the autonomous driving research platform is equipped with several LRFs attached around the vehicle (see section 3.2), it might happen that three sides of an object are visible. Therefore, we introduced a “U” class (Figure 4.5d). All other clusters which cannot be assigned to one of these classes are classified as “O” (for example Figure 4.5e and 4.5f). When a cluster is classified as “O”, it implies that it most certainly is not a road vehicle. The basic idea of classifying clusters in this way is inspired by [13].



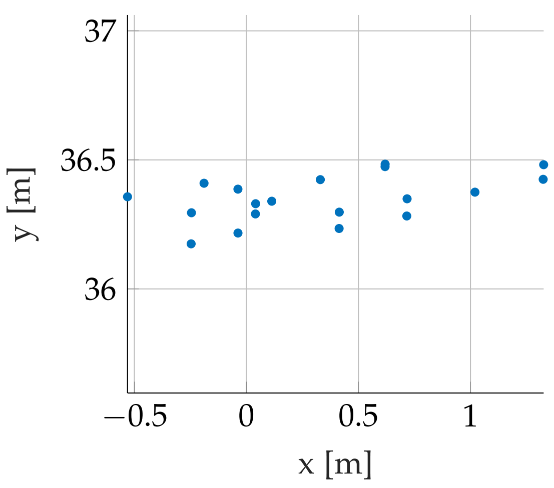
(a) Linear regression. (b) Orthogonal regression.

Figure 4.4: Linear and orthogonal regression. (a) Linear regression minimizes the distance between points and regression line parallel to the y-axis. (b) Orthogonal regression minimizes the perpendicular distance between points and regression line.

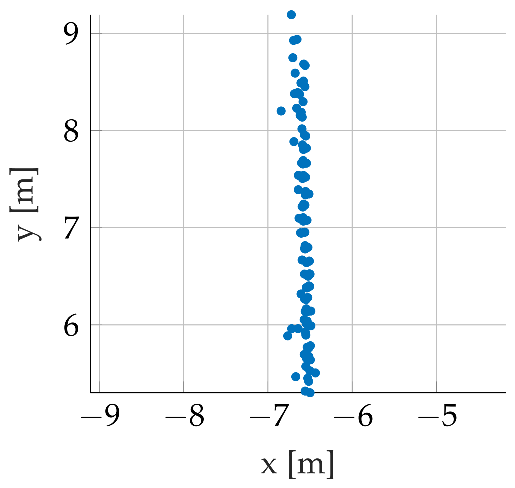
When classifying a cluster, we assume that it is an "H", "V", "L", or "U" cluster. We then perform the pre-fit algorithm used in feature extraction (section 4.4) with linear instead of orthogonal regression. Basically, the algorithm fits one ("H", "V"), two ("L"), or three ("U") lines along the cluster and returns the parameters describing the lines as well as the angles between them. If the lines have a good fit with small error and the angles are approximately 90°, the cluster is assigned to the respective class. In addition, the ratio of the area of the alpha shape [38] to the area of the minimum bounding box (BB) is taken into account to guarantee that the two sides of U-shaped clusters are long enough (Figure 4.6). If all checks fail, the cluster is assigned to the "other" class ("O").

4.3.3 Post-Processing

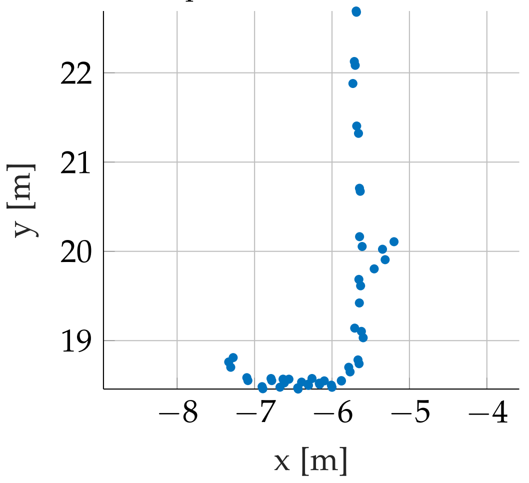
Since the previous classification step considers geometrical information only, a post-processing step is supposed to change the class of wrongly classified clusters. The post-processing comprises one part in which track information is used and another part in which the geometry of cluster pairs is used to reduce over-segmentation.



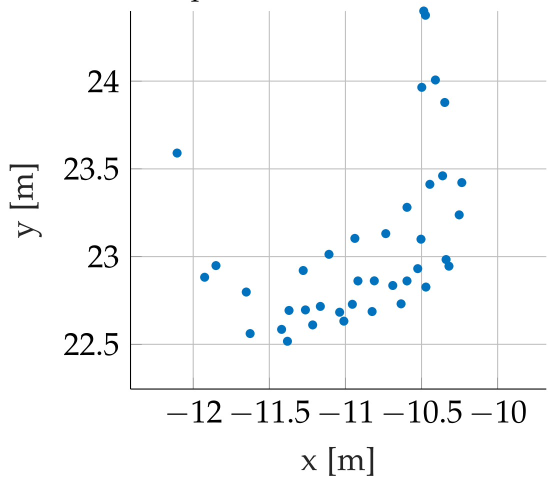
(a) Example cluster of class “H”.



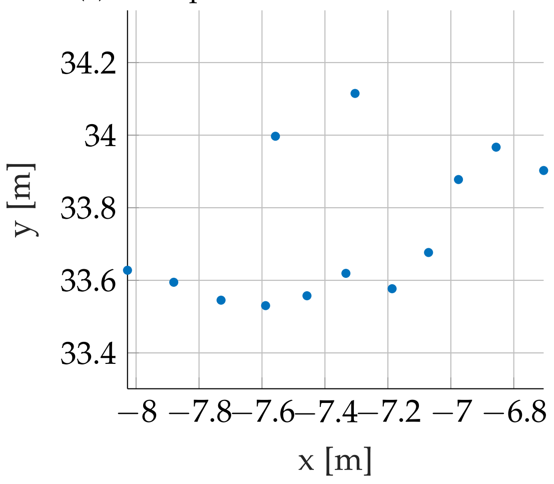
(b) Example cluster of class “V”.



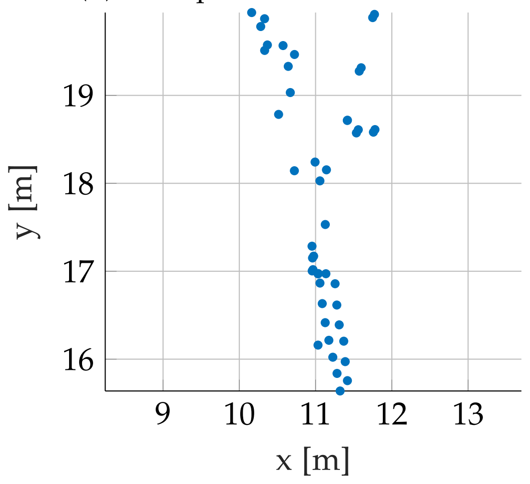
(c) Example cluster of class “L”.



(d) Example cluster of class “U”.

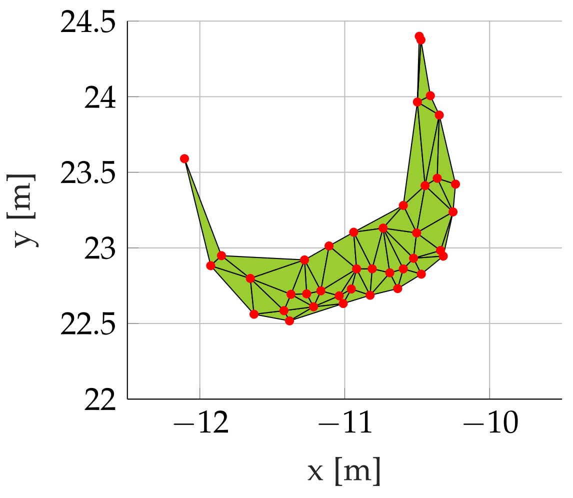


(e) Example cluster of class “O”.

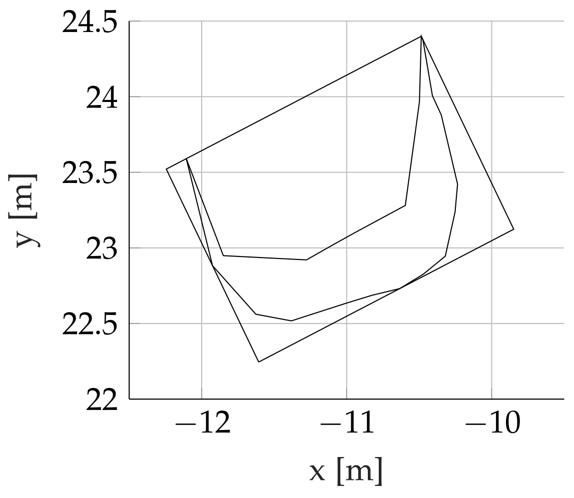


(f) Example cluster of class “O”.

Figure 4.5: Examples of different classes: (a) H, (b) V, (c) L, (d) U, (e) and (f) O



(a) Alpha shape of U-shaped cluster.



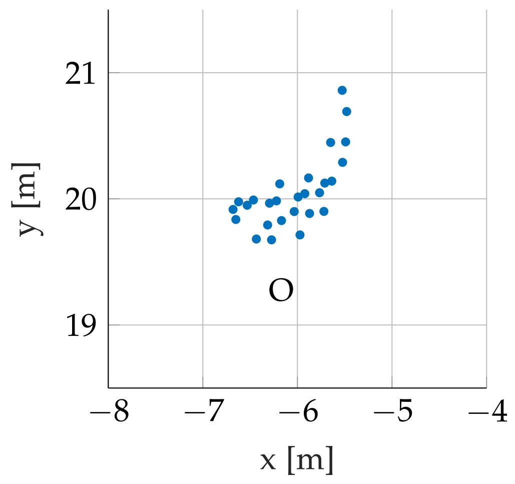
(b) Boundary of alpha shape and minimum bounding box of U-shaped cluster.

Figure 4.6: U-shaped cluster which is depicted in Figure 4.5d with (a) Alpha shape and (b) boundary of alpha shape and minimum bounding box.

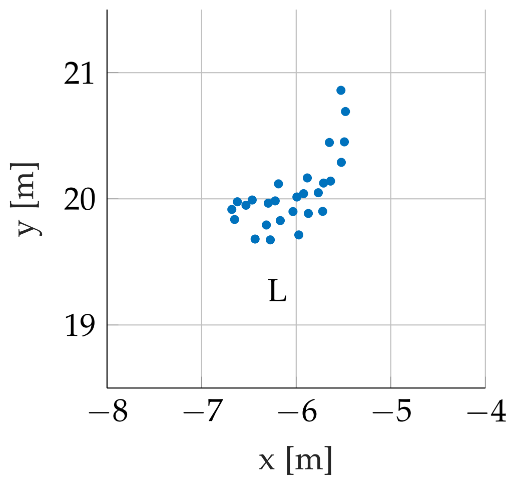
Using Track Information

Sometimes, clusters are oddly shaped even if they should represent a simple cluster shape. As a result, the classification algorithm fails to detect the correct class and assigns the *other* class "O". However, if the motion history suggests that one of those clusters might be a road vehicle and should be assigned otherwise, we predict the movement of tracks and check whether an "O"-cluster occupies the space of one of these predictions. If that is the case, we classify the clusters again but with fewer or more relaxed constraints (Figure 4.7). If again no classification is possible, the cluster will be assigned class "C"[1]. This way, the cluster will be considered in the next steps of the tracking algorithm and might be merged with another cluster if caused by over-segmentation.

[1]The "C" might stand for a crossed class (class does not reflect the actual appearance) and the C-shape looks similar to those of an "O".



(a) Cluster of vehicle classified as "O".



(b) Same cluster classified as "L" during post-processing.

Figure 4.7: Same cluster (a) after classification and (b) after post-processing. Classification as "L" with relaxed constraints is successful during post-processing.

Reducing Over-Segmentation

A perfectly segmented point cloud would consist of one cluster per object in the physical world. Over-segmentation means that more than one cluster belongs to a physical object. The frequency of this happening mostly depends on the distance-threshold of the segmentation algorithm and also occurs due to occlusion, noise, missing points, or in general wider gaps between points within a cluster than expected. Figure 4.8a and 4.8d depict two examples of over-segmented clusters.

Whether two V-, two H-, L- and H-, L- and V-, or L- and C-clusters are merged, ultimately, two straight segments are processed and eventually merged. In order to check whether these segments qualify for being merged, the regression line and minimum bounding box of each segment as well as the minimum bounding box of all points are taken into account, as depicted in Figure 4.8b. Before two clusters can be merged, they have to fulfill certain requirements (compare Figure 4.8b):

- Clusters need to be neighbors (no third cluster in between).
- Distance between clusters needs to be within a threshold.
- Difference of angles between regression lines needs to be within a threshold.
- Difference of widths of small BBs needs to be within a threshold.

- Difference of angles between small BBs to overall BB needs to be within a threshold.
- Length-to-width ratio of overall BB has to be bigger than a threshold.

When all requirements are fulfilled, the two clusters are merged and no longer over-segmented (for example the clusters in Figure 4.8e).

4.4 Feature Extraction Step

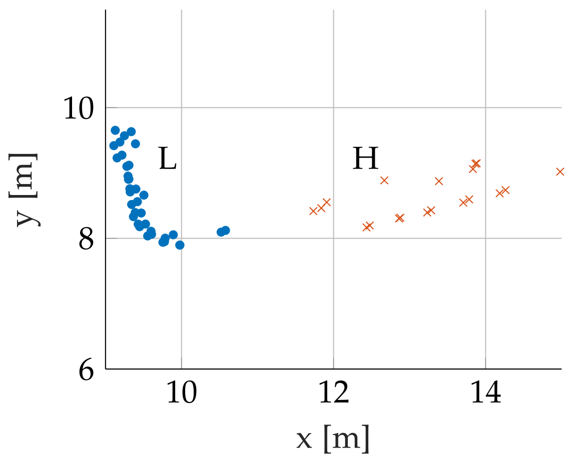
In order to properly track an object and accurately determine values like velocity or acceleration, it is reasonable to extract a stable feature of a vehicle. Due to a car's rectangle-like shape, the feature chosen is a corner. According to the class of a cluster, a different approach extracts the corners of a vehicle. To achieve that, we perform a pre-fit, which is also used for classification, and adapt parameters afterwards. As described in section 3.2, the used LRFs are not considered 3D but still have some 3D information due to their four layers. We want to use this extra information by using orthogonal instead of linear regression with the pre-fit. Usually the extracted corners, especially of L- and U-shaped clusters, are not right corners. That is the reason why we force the angle to be 90° afterwards and adapt other parameters accordingly. In a post-processing step, we calculate the area of the vehicle if the cluster shape allows it. If not, we use the length of, for example, H- or V-clusters to determine whether the cluster is too small to be considered a vehicle. In the following subsections, we briefly explain orthogonal regression, go into detail on how we perform feature extraction, and how the post-processing works.

4.4.1 Orthogonal Regression

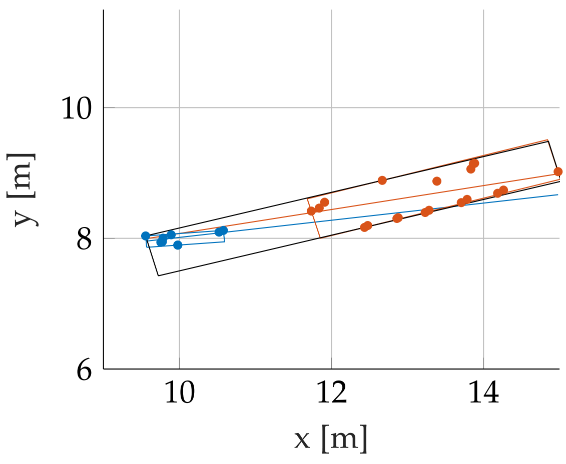
Assume we have n data points $\{(x_i, y_i), i = 1 \dots n\}$ and that x_i and y_i relate to their respective values on the regression line (x_i^*, y_i^*) as follows:

$$y_i = y_i^* + \epsilon_i \tag{4.5}$$

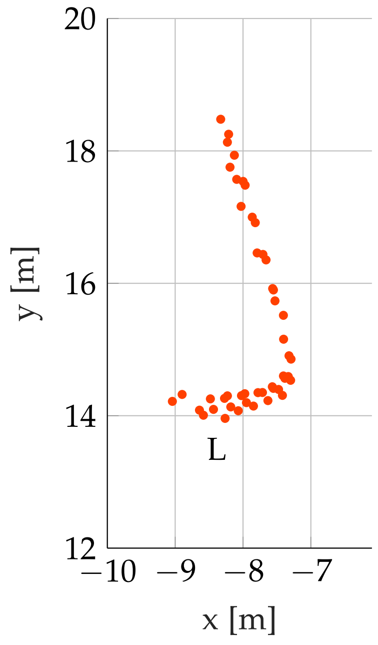
$$x_i = x_i^* + \eta_i \tag{4.6}$$



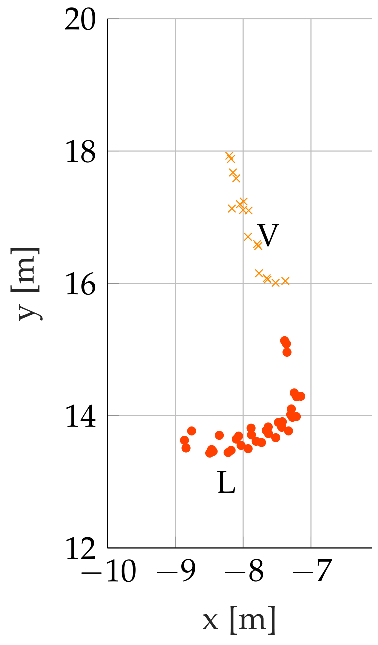
(a) An over-segmented L-shaped cluster resulting in an L- and H-cluster.



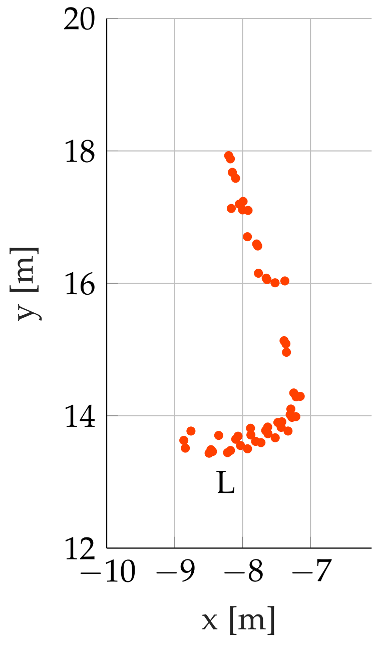
(b) Regression lines, minimum BBs, and overall minimum BB used during check of whether clusters can be merged.



(c) L-shaped cluster normally segmented.



(d) Clusters belonging to the same object of (c) are over-segmented due to a wider gap in the long side of the L.



(e) L- and V- are merged after post-processing.

Figure 4.8: Merging over-segmented clusters.

with residuals ϵ and η in y- and x-direction respectively (compare Figure 4.4b). The residuals are independent and the ratio of their variances is assumed to be known:

$$\delta = \frac{\sigma_\epsilon^2}{\sigma_\eta^2}. \tag{4.7}$$

Since we do not know the variances of ϵ and η, we assume them to be equal and therefore $\delta = 1$, which is called *orthogonal regression* [39]. Like simple linear regression, orthogonal regression determines the parameters of the regression line:

$$y^* = \alpha + \beta x^* \tag{4.8}$$

by minimizing the SSR as well:

$$\underset{\alpha,\beta}{\operatorname{argmin}}\, SSR(\alpha, \beta), \tag{4.9}$$

with [40, p. 37]

$$SSR(\alpha, \beta) = \sum_{i=1}^{n} \left(\frac{\epsilon_i^2}{\sigma_\epsilon^2} + \frac{\eta_i^2}{\sigma_\eta^2} \right) = \frac{1}{\sigma_\epsilon^2} \sum_{i=1}^{n} \Big((y_i - \alpha - \beta x_i^*)^2 + \delta (x_i - x_i^*)^2 \Big). \tag{4.10}$$

4.4.2 Feature Extraction

Roughly, two basic approaches shape the algorithms used to extract features: a plane fitting and a bounding box approach. Features, i.e. corners, of L- and U-shaped clusters are extracted by fitting a plane through the 3D points of each side of the L or U. Parameters of those planes are then adapted to turn the angle into a right one. H- and V-clusters consist of points of just one side of a vehicle. Here we use a combination of the bounding box and linear regression to determine the corner of the vehicle.

Pre-fit L

The idea of the pre-fit algorithm for L-shaped clusters is based on the IEPF algorithm [41] and comprises the following steps:

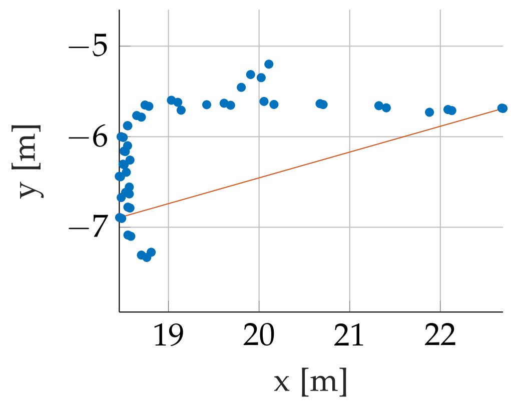
1. Determine the two points with the largest distance between each other (Figure 4.9a).
2. Split line at point with the largest perpendicular distance to the line connecting the two points of step 1 (Figure 4.9b).
3. Determine the points on each of the two sides with the largest distance to that split point (Figure 4.9c)
4. Determine four points with the largest perpendicular distance to the line connecting the two points of step 3 and split the line at the one which results in an angle closest to 90° (Figure 4.9d).
5. For both lines do:
   a) Determine perpendicular distance of all points to line.
   b) Take best fitting 80 percent.
   c) Perform orthogonal regression (Figure 4.9e) if there are at least four points left. Perform linear regression otherwise.

The result in 2D is depicted in Figure 4.9f. The same algorithm with only linear regression is used for classification (subsection 4.3.2). With orthogonal regression, the hyperplane fitting is less prone to outliers which still affect the angle of the corner.

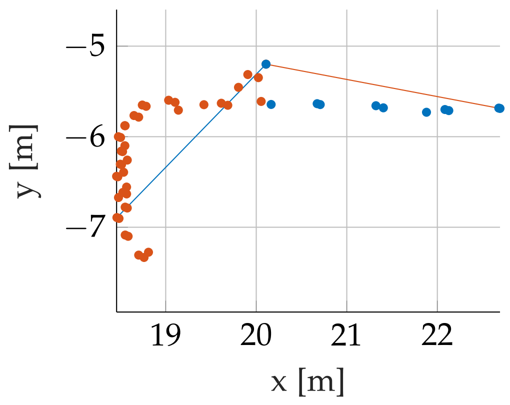
L

The angle provided by the pre-fitting rarely is 90°. Since we know that the corner of a car has a 90° angle, we want to change that. Therefore, we do the following:

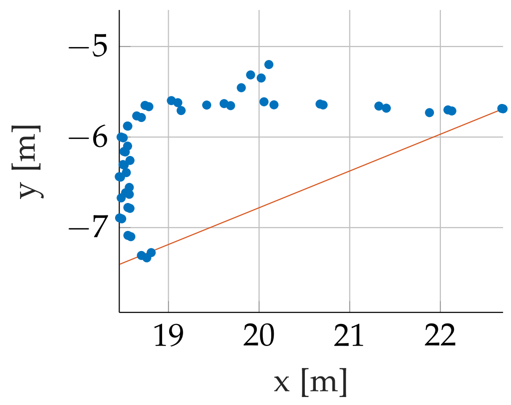
1. Iteratively change slopes such that the angle becomes 90° and adapt y-intersect accordingly (Figure 4.10a).
2. Turn small side around corner by 90° such that it is in line with the long side (Figure 4.10b).
3. Pre-fit a plane through all points.
4. Fit a plane through 80 percent of best-fitting points.
5. Choose slope of small side such that the angle equals 90° (Figure 4.10b).
6. Determine all four corners (Figure 4.10b).



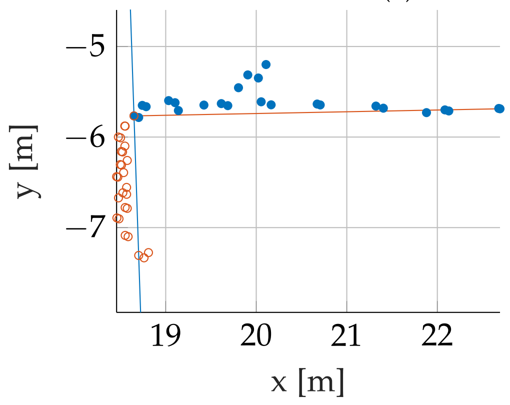
(a) Farthest points connected with a line.



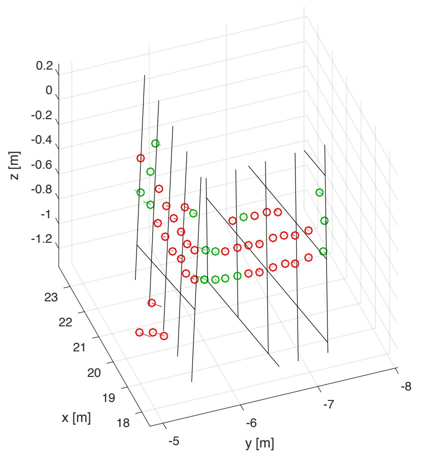
(b) First split at point with largest perpendicular distance to line of (a).



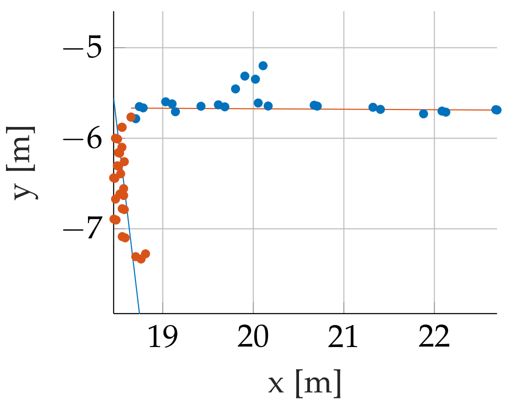
(c) Farthest points from split point connected with a line.



(d) Final split at one of four points with largest perpendicular distance to line of (c) and closest angle to 90°.

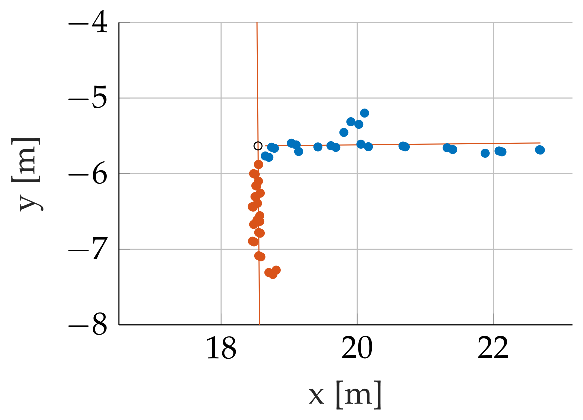


(e) Orthogonal regression.

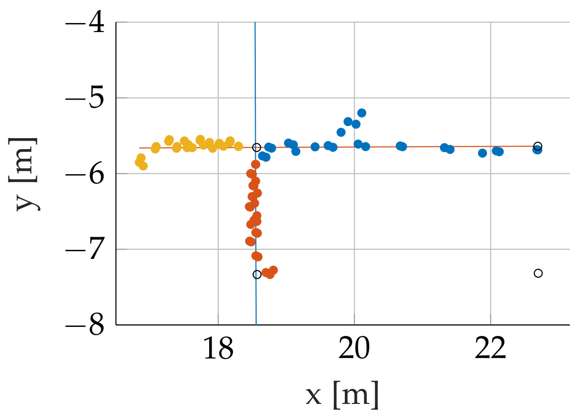


(f) Result after (orthogonal) regression.

Figure 4.9: Pre-fit algorithm for L-shaped clusters.



(a) Angle forced to be 90°.



(b) Small side turned around, plane fitted through 80 percent of the points, and all four corners.

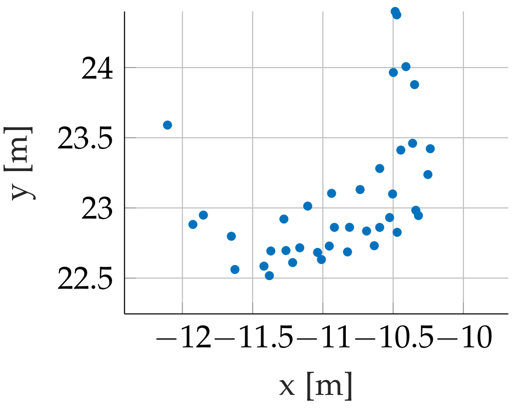
Figure 4.10: Final feature extraction of L-shaped clusters.

Pre-fit U

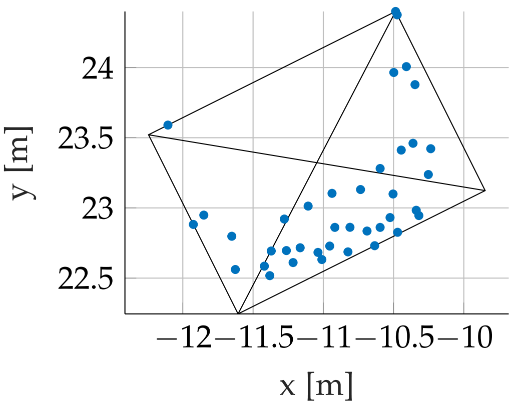
Since the same or a similar algorithm for pre-fitting L-shaped clusters does not work with U-shaped ones, we use another approach:

1. Determine minimum bounding box and diagonals (Figure 4.11b).
2. Group each point with one of the four sides based on the quadrants established by the diagonals (Figure 4.11c).
3. If necessary: Merge points such that actual empty side becomes empty (Figure 4.11d).
4. For each line do:
   a) Fit line through points with largest distance to each other.
   b) Take 80 percent of best fitting points.
   c) Perform orthogonal regression (Figure 4.11e) if there are at least four points left. Perform linear regression otherwise.

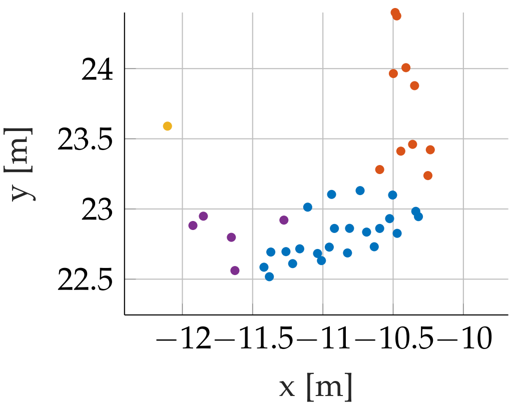
The result in 2D is depicted in Figure 4.11f. The same algorithm with linear regression only is used for classification (subsection 4.3.2). Orthogonal regression reduces the effect of outliers but they still affect the corners and angles.



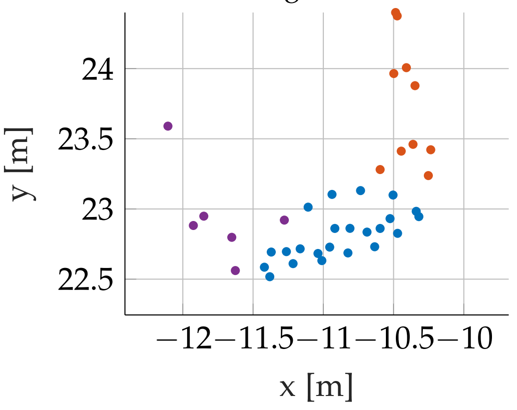
(a) Points of U-shaped cluster.



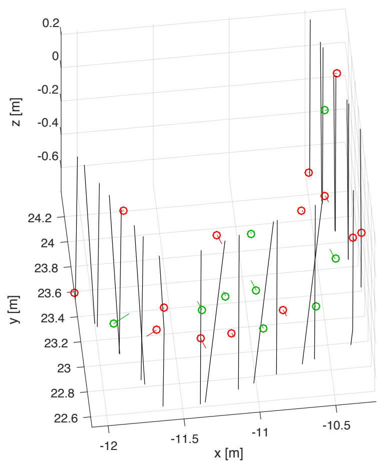
(b) Quadrants established by diagonals of minimum bounding box.



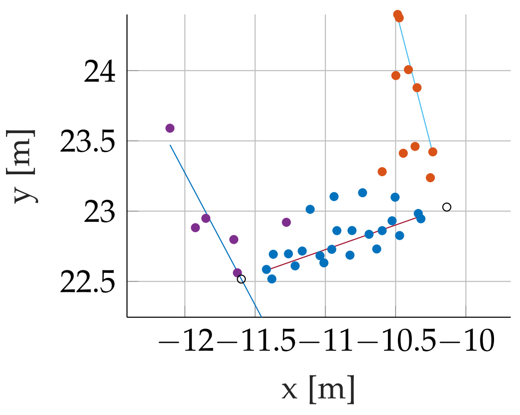
(c) Points grouped based on quadrants.



(d) Three remaining groups after merging.



(e) Orthogonal Regression.



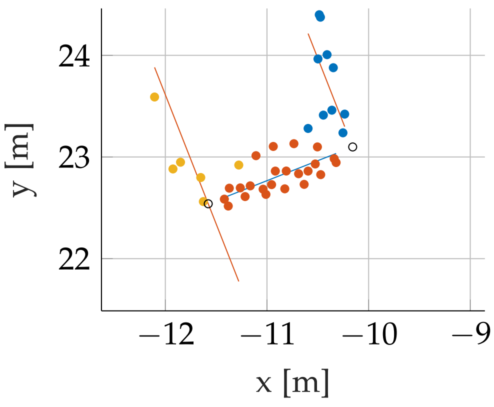
(f) Results after (orthogonal) regression.

Figure 4.11: Pre-fit algorithm for U-shaped clusters.

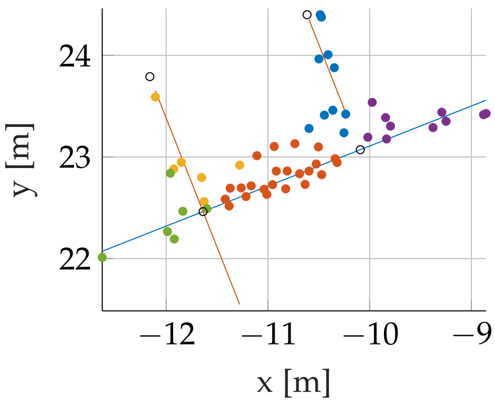
U

Due to outliers and rounded corners, the angles provided by the pre-fitting rarely is 90°. To guarantee a right angle, we perform a similar algorithm to the one for L-shaped clusters:

1. Iteratively change slopes such that the angles become 90° and adapt y-intersects accordingly (Figure 4.12a).
2. Turn outer sides around corners by 90° such that they are in line with middle part (Figure 4.12b).
3. Pre-fit a plane through all points.
4. Fit a plane through 80 percent of best-fitting points.
5. Choose slope of outer sides such that the angles equals 90° (Figure 4.12b).
6. Determine all four corners (Figure 4.12b).



(a) Angles forced to be 90°.



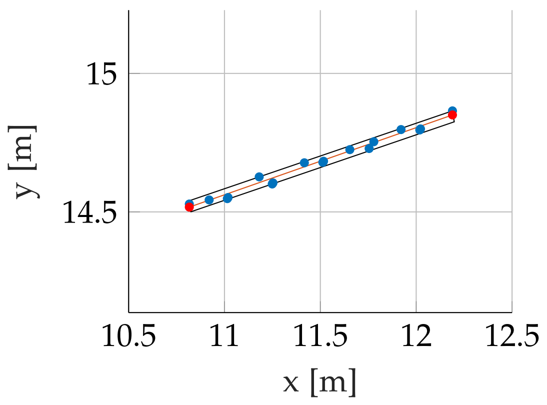
(b) Outer sides turned around, plane fitted through 80 percent of the points, and all four corners.

Figure 4.12: Final feature extraction of U-shaped clusters.

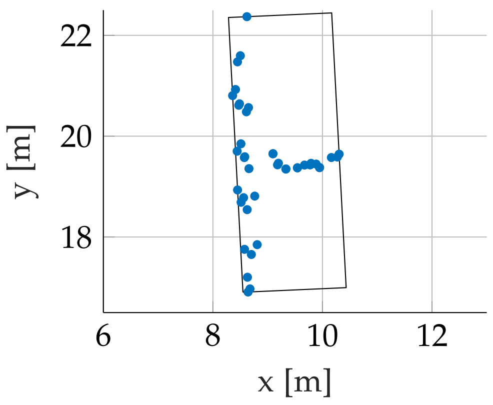
H/V

H- or V-clusters represent just one side of a car and may contain one, both, or no corners of the actual car. As features of H- or V-clusters, we choose both ends of these clusters. To determine a good corner estimate, we do the following:

1. Determine minimal bounding box.
2. Pre-fit a line through all points.
3. Fit a line through 90 percent of best-fitting points.
4. Determine intersection of regression line and bounding box (Figure 4.13a).



(a) Minimum bounding box, regression line, and extracted features of an H-cluster.



(b) C-cluster and its minimum bounding box whose corners are used as features for C-clusters.

Figure 4.13: Feature extraction of H-, V-, and C-clusters. (a) The features of an H- or V-cluster are depicted as red points which are the intersection of the regression line and the minimum bounding box. (b) The features of C-clusters are corners of the minimum bounding box.

C

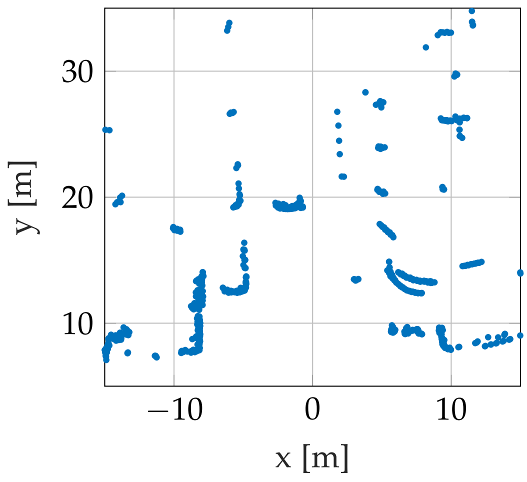
Since a C-cluster can have any shape (it is just assigned "C" due to its motion history), we use the corners of the minimum bounding box as features (Figure 4.13b).

4.4.3 Post-Processing

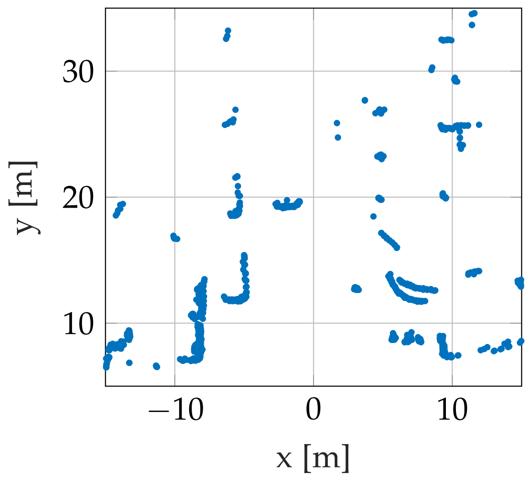
After extracting all features, it is possible to calculate the 2D area an L- or U-shaped cluster covers. The area of the object represented by an H- or V-cluster cannot be calculated. However, for post-processing, their length provides enough information. To cancel out too small or too large objects for the next steps in the tracking chain, we define some empirically determined thresholds for minimum length, maximum length, minimum area, and maximum area. If a track's prediction overlaps with a cluster, we use less restrictive thresholds. If a cluster does not lie within the defined range, we change its class to "O".

4.5 Data Association

After determining classes and extracting features, we have to identify the same object in a sequence of scans to compute the motion history. During a drive, the LRF provides several new scans per second and as an example, Figure 4.14a and 4.14b depict two consecutive scans. The so called correspondence problem is about finding the cluster in scan at time $t + 1$ which belongs to the same object in the real world represented by a certain cluster in the scan at time t. To solve this problem we use the motion history and predict the movement of the object. If the future position of the object and the cluster match, we assign the cluster to the object.



(a) Scan at time t.



(b) Scan at time $t + 1$.

Figure 4.14: Two consecutive scans.

Since we use the motion history, the correspondence problem is only solved for objects which are tracked during that scan. Subsequently, we explain the prediction and how the matching works.

4.5.1 Kalman Filter

We predict the movements of objects with Kalman filters (KFs) [42]. The values we are interested in are: position (x- and y-coordinate), velocity (in x- and y-direction), and the yaw angle ϕ of the object and its angular velocity ω. Therefore, the state vector is:

$$\textit{State vector } \mathbf{x} = \begin{pmatrix} x & v_x & y & v_y & \phi & \omega \end{pmatrix}^T \tag{4.11}$$

State $x(n)$ transitions into next state $x(n{+}1)$ with:

$$\textit{State transition } \mathbf{A} = \begin{pmatrix} 1 & \Delta t & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & \Delta t & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.12}$$

This way, the position gets updated with the old position and the velocity. With two consecutive velocities $v(t)$ and $v(t-1)$, it is possible to calculate an average acceleration over that time range: $a = \frac{v(t)-v(t-1)}{\Delta t}$. We do that and add the translational and rotatory acceleration as the control vector:

$$\textit{Control vector } \mathbf{u} = \begin{pmatrix} a_x & a_y & \alpha \end{pmatrix}^T \tag{4.13}$$

These values are added to the state prediction with:

$$\textit{Control matrix } \mathbf{B} = \begin{pmatrix} 0.5 * \Delta t^2 & 0 & 0 \\ \Delta t & 0 & 0 \\ 0 & 0.5 * \Delta t^2 & 0 \\ 0 & \Delta t & 0 \\ 0 & 0 & 0.5 * \Delta t^2 \\ 0 & 0 & \Delta t \end{pmatrix} \tag{4.14}$$

With the state transition and control vector, the next state is predicted as follows:

$$x(n+1) = x(n) \quad + v_x(n)\Delta t \quad + a_x(n) * 0.5\Delta t^2 \tag{4.15}$$

$$v_x(n+1) = \quad + v_x(n) \quad + a_x(n)\Delta t \tag{4.16}$$

$$y(n+1) = y(n) \quad + v_y(n)\Delta t \quad + a_y(n) * 0.5\Delta t^2 \tag{4.17}$$

$$v_y(n+1) = \quad + v_y(n) \quad + a_y(n)\Delta t \tag{4.18}$$

$$\phi(n+1) = \phi(n) \quad + \omega(n)\Delta t \quad + \alpha(n) * 0.5\Delta t^2 \tag{4.19}$$

$$\omega(n+1) = \quad + \omega(n) \quad + \alpha(n)\Delta t \tag{4.20}$$

For the measurement covariance matrix **R**, we use a standard deviation (STD) of 1 cm for the x- and y- position, an empirically determined STD of $5\frac{m}{s}$ for the velocities, an empirically determined STD of $1°$ for the angle and $5\frac{deg}{s}$ for angular velocity:

$$\textit{Measurement covariance } \mathbf{R} = \begin{pmatrix} 0.01^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 5^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.01^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 5^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 5^2 \end{pmatrix}. \tag{4.21}$$

To complete the list of matrices needed for a Kalman filter, the remaining matrices are:

$$\textit{Observation matrix } \mathbf{H} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.22}$$

$$\textit{Initial probability}\ \mathbf{P} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.23}$$

$$\textit{Process covariance}\ \mathbf{Q} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}. \tag{4.24}$$

4.5.2 Data Association

As seen in Figure 4.14a and 4.14b, two consecutive scans look very similar. However, due to moving objects and moving LRFs, the clusters themselves will most likely look differently. Due to slightly different cluster shapes and wider or more narrow gaps, the segmentation algorithm might perform differently, which again affects all following steps in the tracking chain. After extracting features, the correspondence problem has to be solved in order to properly track an object.

Figure 4.15 depicts all information needed to do so. The black rectangle is the currently known position of the tracked object. The blue points are the features of a cluster extracted in the step before. Now, we predict the future position of the object with a Kalman filter described in subsection 4.5.1. The predicted position is depicted as the red dashed rectangle in Figure 4.15. When the predicted rectangle and the rectangle of the detected object overlap, we assign the cluster as a measurement to the track. However, the cluster-track-combination has four permutations. There might be a track but no cluster, a cluster but no track, a track and multiple clusters, or multiple tracks and just one cluster. Section 4.6 describes how these variations are handled. If the assignment is unique, like the one in Figure 4.15, the track or rather its Kalman filter is updated with the measurement. The value returned by the Kalman filter determines the new position of the tracked object, which is depicted as a blue dashed rectangle for our example in Figure 4.15.

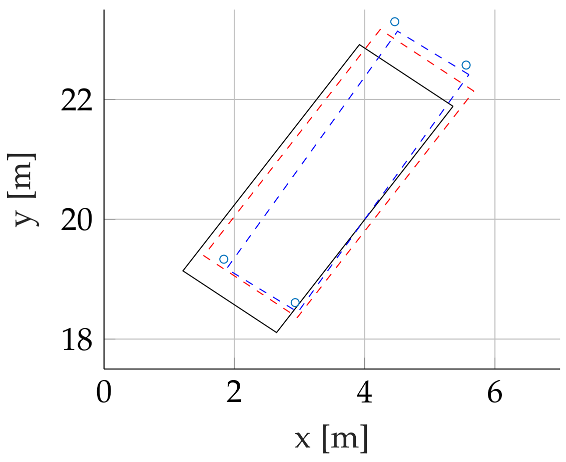


Figure 4.15: Correspondence problem. The black rectangle is the current position of the track. The red dashed rectangle is the prediction of the track provided by the Kalman filter. The blue points are the extracted features of a detected object. The blue dashed rectangle is the new position of the track after assigning the measured cluster to the track and updating the Kalman filter with it.

4.5.3 Reference Point and Handling of View Change

During the lifetime of a track, the observed object will most certainly move, which again leads to different views the LRFs will perceive. That is the reason the feature extraction algorithm (section 4.4) extracts corners, because one sensor sees at most two sides of an object. If a sensor detects two sides, the corner in the middle is a stable feature. If the sensor perceives just one side, it is impossible to determine whether the ends of the line are the corners of an object without further information.

When the object is moving and changing its angle relative to the sensor, one of the sides or line segments will disappear and another side or line segment will appear eventually. This change of viewpoint leads to the loss of the reference point and problems during tracking. Figure 4.16 shows an example of this process.

We handle this issue by memorizing the position of the reference point relative to the object in the longitudinal and lateral direction. Figure 4.17 depicts the same example in a schematic and simplified version. For the sake of this example, we assume that Figure 4.17a is the first time the object is detected. The reference point is the extracted corner in the lower left. Along with other information, we store the longitudinal and lateral vector along the vehicle originating from the reference point.

The vehicle will steer to the right and the left side will disappear. Therefore, the left side is missing in Figure 4.17b. In H- or V-clusters, the closest endpoint of the line is equivalent to the reference point. After turning a bit more, the other endpoint becomes closer to the sensor and therefore becomes the reference point. As seen in Figure 4.17c, the lateral vector reverses direction. During the process of associating the detected objects and properly updating the motion history of tracks, the stored reference vector is compared with the current one. If the two vectors point in opposite directions, the reference point will be changed to the other corner. If the turning continues, the right side of the object will appear, as depicted in Figure 4.17d. Now we just compare longitudinal and lateral vectors separately and change the reference point accordingly.

4.6 Tracking

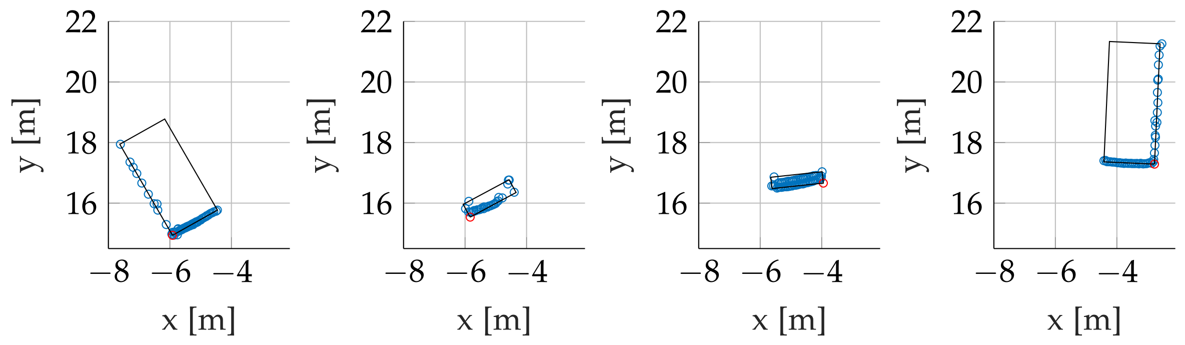
As briefly mentioned in subsection 4.5.2, there are several variations of cluster-track assignments: a cluster without a track, a track and exactly one cluster, a track and multiple clusters, multiple tracks and just one cluster, or a track without a cluster. How the algorithm handles all these cases is explained in the following subsections.

4.6.1 Track Creation

If there is a cluster which qualifies as a vehicle and is therefore desirable to track, a new track should be created. Our algorithm starts observing every cluster classified as an "L" and every H-cluster in front of us. This "observing", however, serves as a pre-tracking in order to detect objects which not only look like but also behave like a road vehicle. After three scans we start checking properties of these objects including speed, orientation, and extracted features. If they pass the test, the observed objects are transformed into tracks. To avoid conflicts, all non-track objects are still tracked in the background.

4.6.2 Track Update

Once a vehicle is tracked and it is assigned exactly one cluster after a new scan, its history is updated with this one cluster.



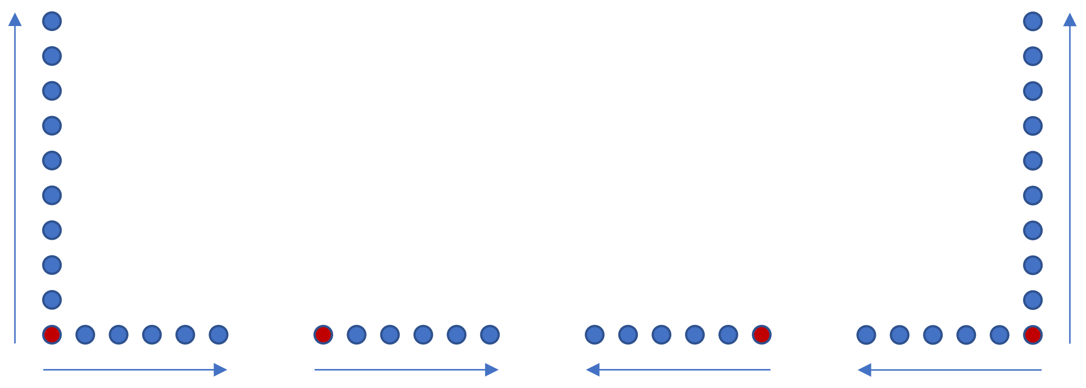
(a) L-shaped cluster at first point in time steering to the right. The reference point is the lower left corner.

(b) Same object a few scans later. Due to the change of the angle only the rear of the car is visible. The reference point is still the left point.

(c) The same object a few scans later. Due to the change of orientation the reference point shifted to the right point of the H-cluster.

(d) After a few scans, the right side of the vehicle is visible and the reference point is again a corner, but this time the lower right one.

Figure 4.16: Change of perspective leads to change of reference points.



(a) L-shaped cluster with reference point and the respective vectors.

(b) H-cluster with reference point and the respective vector which is the same as in (a).

(c) H-cluster with changed reference point and reversed reference vector.

(d) Mirrored L-shaped cluster with reference point at another corner and the respective vectors.

Figure 4.17: Schematic depiction of Figure 4.16. The arrows show the vectors starting at the reference point and pointing towards the sides of the object for H- and L-clusters.

4.6.3 Ambiguity Handling

In the case of a track and multiple clusters or multiple clusters assigned to one track, we update a track with the cluster which is closest to its predicted position and adapt the other cluster-track assignments accordingly. If there is an ambiguity, we determine all clusters which are assigned to the currently considered track or all tracks which are assigned to the currently examined cluster. We then predict the future position of the tracks and compare those positions with the cluster positions. The combination with the smallest distance will be preserved. That means that if a track was assigned multiple clusters, it will be assigned to the closest one. When multiple tracks are assigned to the same cluster, the cluster will be assigned to the track with the closest prediction and it will be removed from the list of assigned clusters of the other tracks. All tracks must go through a new track management iteration. One track will be updated and the others might be updated or handled differently depending on the number of remaining assigned clusters.

4.6.4 Track Deletion

It may also happen that a LRF does not recognize an object where the prediction expects one. This can occur for several reasons. In the case that no cluster is assigned to a certain track, the Kalman filter takes over and predicts its position for several frames if the track fulfills two requirements: The STDs of the orientation and heading of the last three scans have to be within a certain threshold. If it is impossible to recover the track for a certain amount of time, the track will be deleted. This time range amounts to one second for tracks and 0.2 seconds for non-track objects. If a track moves in the opposite direction of the ego-vehicle and already passed it, it won't be predicted any further to reduce the number of false negatives (FNs).

5 Evaluation

In order to evaluate to what extent the implemented algorithm performs successfully, we test it in different ways. Before showing the results, we want to describe how the testing works and what specific quality metrics we are using.

5.1 Methodology

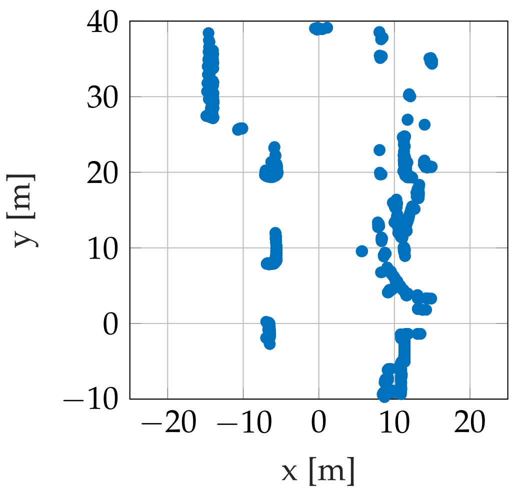
We want to test tracking-related properties and other qualities we focused on. For tracking-related quality evaluation, we use labeled data as ground truth (GT) and for the other measures, we use special scenarios.

5.1.1 Labeled Data as Ground Truth

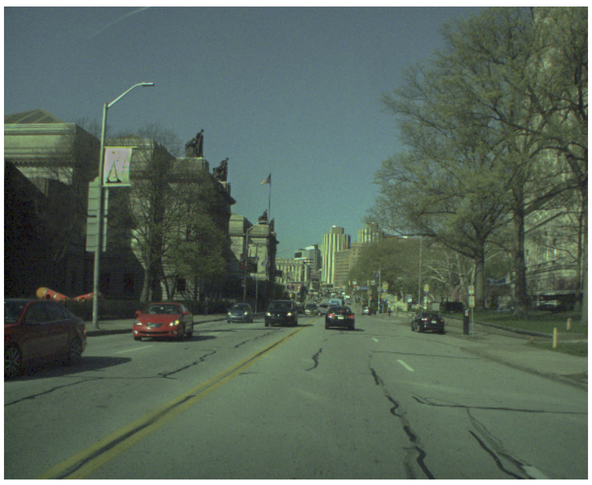
Obtaining real ground truth for LIDAR data which are generated by a moving platform with moving obstacles is practically impossible. That is the reason why measured data is annotated with the type and position of objects the sensor, whether it is a camera, a LRF, or another sensor, should detect. We labeled some scans of our obtained data by ourselves and used the KITTI Vision Benchmark Suite [43] as our ground truth. However, at this point, we want to mention that the used annotation software and the annotator influence the annotation quality, which again has an influence of the evaluation result [44].

Own Labeled Data

We took 30 scans of our data and labeled every cluster by deciding whether it is a vehicle or not and we gave every vehicle a unique id to enable track-based quality evaluation. During these scans, the ego-vehicle is driving on a straight road and the LRFs are able to detect several vehicles of the oncoming traffic (Figure 5.1).



(a) Labeled LIDAR data.



(b) Camera picture of labeled scene.

Figure 5.1: Own labeled data. (a) LIDAR points of one of the labeled cycles and (b) camera picture of the same cycle.

The KITTI Vision Benchmark Suite

Karlsruhe Institute of Technology and Toyota Technological Institute at Chicago teamed up to create the KITTI Vision Benchmark Suite [43]. It is a comprehensive collection of several kinds of data and annotations including different kinds of vehicles and pedestrians. For benchmarking purposes they provide annotated training data sets and test sets without labels. Despite the project being vision-based, it also provides 3D LIDAR data. Since we need 2D data, we need to adapt the data set. To simulate 2D data points, we determined a coordinate on the vertical axis which is approximately at the same level where road vehicles are (the used LRF is mounted on the top of the car). We then took all points which are a bit above and below this coordinate and removed the rest. This reduced data set serves us as a 2D LIDAR data set.

5.1.2 Special Scenarios

The second part of the evaluation is about three properties of the tracking algorithm: First, tracking velocity, second, handling of changing view point, and third, occlusion handling.

Scenario 1 - Velocity

Usually, the ego-vehicle does not know the velocity of other traffic participants. A Kalman filter, described in subsection 4.5.1, tracks position and velocity. In order to test this velocity estimation, the ego-vehicle drives towards another, still, vehicle. If we fix the position of the ego-vehicle afterwards, it seems like the other vehicle is moving and we can compare the estimated velocity from the Kalman filter with the known velocity of the ego-vehicle. Since the Kalman filter is initialized with a velocity of zero, it will take a short moment until the estimated velocity will match the real one.

Scenario 2 - Changing Viewing Angle

The second scenario tests the handling of a changing viewpoint described in subsection 4.5.3. For that, the ego-vehicle drives around another still vehicle. This way, all four sites and corners come into view.

Scenario 3 - Occlusion

One important property of a tracking algorithm is to overcome short sequences without measurements, for instance caused by occlusion. To test the prediction capability of the Kalman filter, the ego-vehicle stands still and points towards another still vehicle a few meters away. A third vehicle drives laterally to the ego-vehicle beyond the second one. Shortly after it reaches the second car, it will be occluded and the LRFs should not be able to detect it. During that time, the Kalman filter should predict the movement such that the prediction and the re-appearing car will match a few scans later.

5.2 Quality Metrics

In order to evaluate the quality of the tracking algorithm with labeled data, we use two kinds of metrics: frame-based and track-based metrics. The frame-based metrics used are the CLEARMOT [45] and the track-based are the mostly tracked (MT)/partially tracked (PT)/mostly lost (ML) metrics [46].

5.2.1 Frame-Based Quality Metrics

The multiple object tracking (MOT) metrics are MOT accuracy (MOTA) and MOT precision (MOTP). MOTA takes all errors, i.e., false positives (FPs), false negatives (FNs), and id switches (IDs), into account. Since the metric subtracts the number of errors relative to the number of GT objects, the MOTA value might be negative. It is calculated as follows:

$$MOTA = 1 - \frac{\sum_t (\mathrm{FP}(t) + \mathrm{FN}(t) + \mathrm{ID}(t))}{\sum_t N_{\mathrm{GT}}(t)}, \tag{5.1}$$

with t being an index indicating a cycle. MOTP measures how well a tracker localizes other objects. It averages the distance between GT and hypotheses (Hs) over all matched pairs. In 2D, it represents the average overlap of matched bounding boxes:

$$MOTP = \frac{\sum_{t,i} \bar{d}(\mathrm{GT}_i^t, \mathrm{H}_{g(i)}^t)}{\sum_t m_t}, \tag{5.2}$$

with $\bar{d}$ being the distance function, m being the number of matches, and i being an index indicating a cluster.

5.2.2 Track-Based Quality Metrics

As opposed to the CLEARMOT metrics, the MT-, PT-, and ML-metrics take the whole track into account. They classify a track into three categories, like their names ("mostly tracked", "partially tracked", "mostly lost") suggest, depending on the amount of the calculated track which is consistent with the annotated track (Table 5.1). In addition to

Table 5.1: MT/PT/ML metrics and their definition

	ML	PT	MT
Percentage of track compliant with GT	$< 20\,\%$	$\geq 20\,\%$ and $< 80\,\%$	$\geq 80\,\%$

these three values, how many times a track was lost, which is called fragments (FM), is often added.

5.3 Results

Subsequently, we present the results of the aforementioned tests.

5.3.1 Labeled Data as Ground Truth

First, we show the results of our own labeled data in three tables showing MOTA, MOTP, and the MT/PT/ML-metrics respectively. Second, we summarize all relevant information of all 21 KITTI training data sets in one table.

Own Labeled Data

Table 5.2 contains all relevant data regarding the MOTA metric. For comparison with other methods, only the first row is of interest. For better analysis, we split the metrics by track/non-track and type of classes depicted in the rows below.

Table 5.2: MOTA of own labeled data.

	MOTA	FP	FN	ID	TP	TN	Num of objects
All	0.8741	0	92	0	53	586	731
Track	0.3655	0	92	0	53	0	145
No track	1.0000	0	0	0	0	586	586
L	0.5686	0	44	0	11	47	102
H	0.8790	0	15	0	24	85	124
V	0.9553	0	11	0	16	219	246
C	0.8750	0	2	0	2	12	16
U	0.5556	0	4	0	0	5	9
O	0.9316	0	16	0	0	218	234

Table 5.3 summarizes all data regarding the MOTP. Since MOTP is used as a relative value, the mean of the second row of the table can be used for comparison. We distinguished track from no-track objects and all class types as well and added absolute values in meter and the maximum and minimum occurring values for better analysis. The relative values range between zero and 100 percent.

Table 5.4 shows the MT/PT/ML metrics. We listed every single track and its percentage. The last row depicts the actual MT/PT/ML metrics for all objects and all cycles. For FM, the last but one row shows the final result.

Table 5.3: MOTP of own labeled data.

			Mean	Max	Min
All	absolute	[m]	0.4766	14.7479	0.0001
	relative	[%]	56.28	99.96	0.00
Track	absolute	[m]	0.1904	1.1320	0.0004
	relative	[%]	63.85	99.65	2.43
No track	absolute	[m]	0.5650	14.7479	0.0001
	relative	[%]	53.94	99.96	0.00
L	absolute	[m]	0.2595	2.9949	0.0054
	relative	[%]	86.33	99.69	0.00
H	absolute	[m]	0.2500	1.3296	0.0004
	relative	[%]	37.95	99.89	0.00
V	absolute	[m]	0.8057	14.7479	0.0009
	relative	[%]	43.27	99.72	0.00
C	absolute	[m]	0.0827	0.2327	0.0014
	relative	[%]	87.89	99.56	63.67
U	absolute	[m]	0.5522	1.3427	0.0877
	relative	[%]	74.27	93.54	59.15
O	absolute	[m]	0.1023	2.3281	0.0001
	relative	[%]	82.23	99.96	18.42

The KITTI Vision Benchmark Suite

Table 5.5 contains all relevant information of both CLEARMOT and MT/PT/ML metrics. In addition, the table contains the *recall*. This value describes the relative amount of true positives (TPs) of all objects that should be TPs. In other words:

$$Recall = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{5.3}$$

Table 5.4: MT, PT, ML, and FM of own labeled data.

	MT [%]	PT [%]	ML [%]	FM
1	96.67	-	-	1
2	100.00	-	-	0
3	100.00	-	-	0
4	89.29	-	-	1
5	100.00	-	-	0
6	100.00	-	-	0
7	-	66.67	-	1
8	100.00	-	-	0
9	100.00	-	-	0
10	-	71.43	-	1
11	-	63.64	-	1
12	-	69.57	-	1
13	100.00	-	-	0
14	100.00	-	-	0
15	-	60.00	-	1
16	-	50.00	-	1
17	-	33.33	-	1
18	-	33.33	-	1
19	-	50.00	-	1
20	-	31.25	-	1
21	100.00	-	-	0
22	-	71.43	-	1
23	-	33.33	-	1
Σ	11	12	0	14
%	47.83	52.17	0.00	-

Table 5.5: CLEARMOT and MT/PT/ML metrics of all KITTI training data sets.

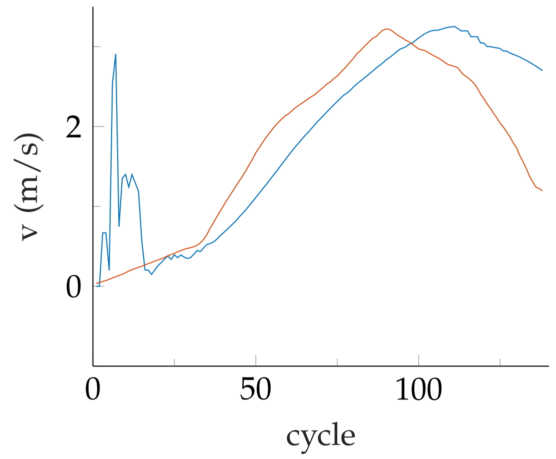
Training data set	MOTA	MOTP	Recall	MT	PT	ML	TP	FP	FN	ID	FM
0000	-1.3070	0.6687	0.3042	0.0000	0.3333	0.6667	73	329	167	0	2
0001	-0.1451	0.6153	0.0322	0.0000	0.0460	0.9540	75	407	2253	0	2
0002	-0.0800	0.6592	0.0228	0.0000	0.0667	0.9333	23	96	984	0	0
0003	-0.3533	0.5432	0.0689	0.0000	0.0000	1.0000	23	141	311	0	5
0004	-0.3914	0.5762	0.0091	0.0000	0.0000	1.0000	7	308	762	0	0
0005	-0.1139	0.6387	0.0033	0.0000	0.0000	1.0000	4	142	1208	0	0
0006	-0.6986	0.6029	0.1336	0.0000	0.3636	0.6364	68	408	441	2	7
0007	-0.2524	0.6418	0.0855	0.0000	0.0943	0.9057	169	658	1808	0	4
0008	-0.1052	0.5361	0.0020	0.0000	0.0000	1.0000	2	109	1015	0	0
0009	-0.1709	0.6146	0.0205	0.0000	0.0380	0.9620	51	471	2434	0	1
0010	-0.3046	0.5190	0.0017	0.0000	0.0000	1.0000	1	178	580	0	0
0011	-0.1626	0.6082	0.0481	0.0000	0.0392	0.9608	138	605	2734	0	9
0012	-0.8671	0.6087	0.0490	0.0000	0.0000	1.0000	7	131	136	0	1
0013	-23.2400	0.5240	0.2188	0.0000	0.0000	1.0000	7	581	25	0	0
0014	-0.2421	0.6071	0.0363	0.0000	0.1429	0.8571	15	115	398	0	2
0015	-1.1552	0.5882	0.0229	0.0000	0.0000	1.0000	13	668	554	0	0
0016	-0.4390	Inf	0.0000	0.0000	0.0000	1.0000	0	367	836	0	0
0017	-Inf	Inf	0.0000	0.0000	0.0000	0.0000	0	460	0	0	0
0018	-0.4131	0.5987	0.1453	0.0000	0.1111	0.8889	178	684	1047	0	8
0019	-3.5096	0.5891	0.1330	0.0000	0.5000	0.5000	112	3031	730	0	7
0020	-0.1015	0.6420	0.0957	0.0093	0.0935	0.8972	473	969	4470	0	12

5.3.2 Special Scenarios

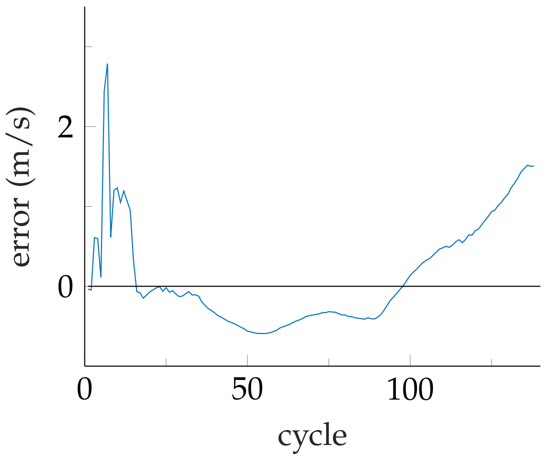
In the following subsections, we present the results of scenario 1 as graphs and the results of scenario 2 and 3 as snapshots of specific and interesting moments.

Scenario 1 - Velocity

Figure 5.2 depicts two graphs. The first one shows the velocity of the ego-vehicle and the velocity estimation of the Kalman filter. The second one shows the difference between the two of them.



(a) GT velocity (red) and velocity estimated by Kalman filter (blue).

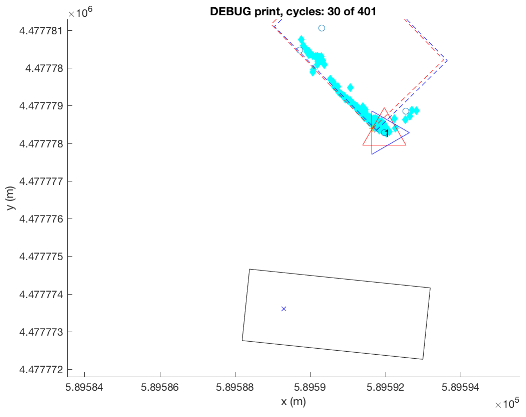


(b) Difference of GT and estimated velocity.

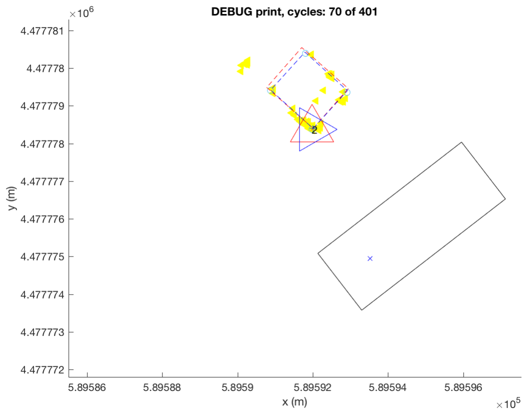
Figure 5.2: Results of Scenario 1: (a) GT velocity, estimated velocity, and (b) the difference between them.

Scenario 2 - Changing Viewing Angle

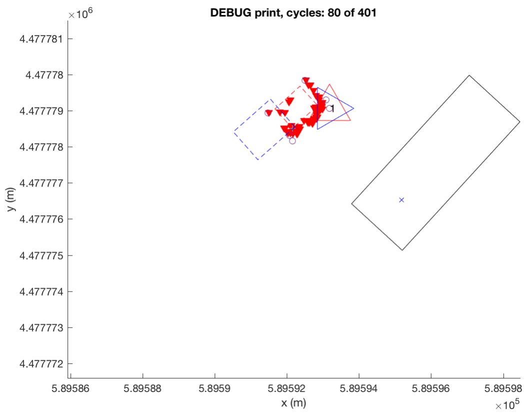
Figure 5.3 and Figure 5.4 show some key moments as the ego-vehicle drives away from the other car and returns to it. The figures show how the other car is detected and where the detected and adapted reference points are. The red triangle in the figures marks the reference point as it is detected. The blue triangle marks the reference point after a possible adaptation.



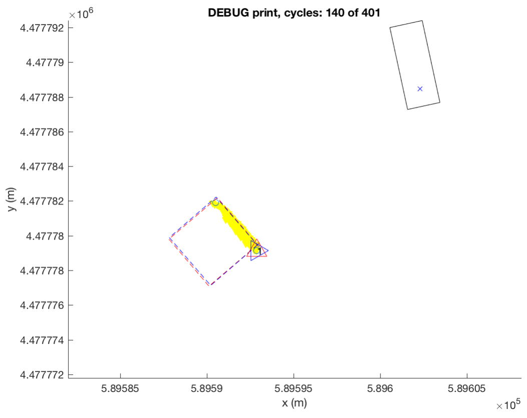
(a) Cycle 30. Detecting the vehicle.



(b) Cycle 70. Detecting the vehicle from another view.



(c) Cycle 80. Changing the reference point.

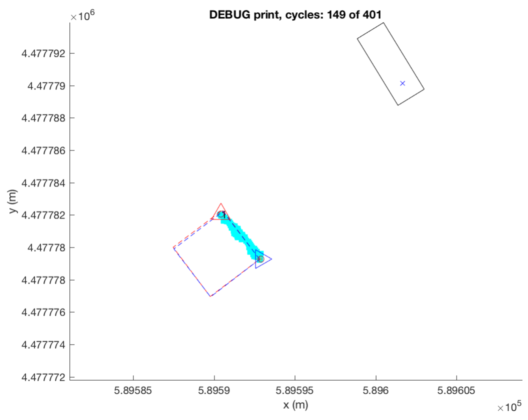


(d) Cycle 140. The other vehicle appears as a line.

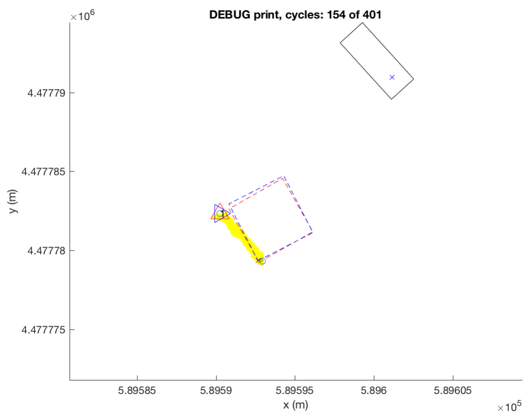
Figure 5.3: Results of Scenario 2: Interesting moments during the first part of the scene. Detecting the vehicle and adapting the reference point. The red triangle marks the reference point as it is detected. The blue triangle marks the reference point after a possible change.

Scenario 3 - Occlusion

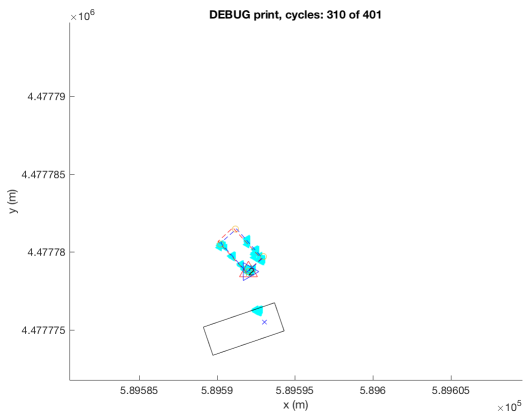
Figure 5.5 shows the beginning of the scene, the moment the car is fully occluded, the moment it reappears, and the end of the scene.



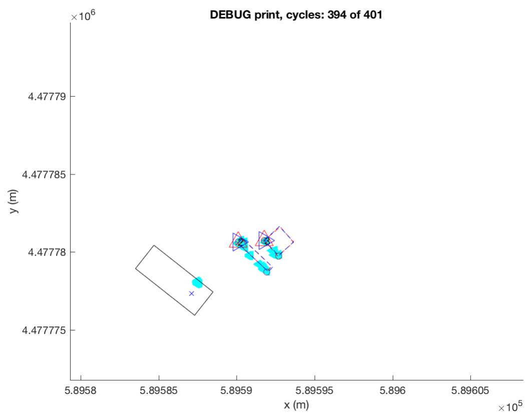
(a) Cycle 149. Adapting the reference point.



(b) Cycle 154. Changing the reference point and orientation of the detected vehicle.



(c) Cycle 310. Detection of an oddly shaped cluster as vehicle.

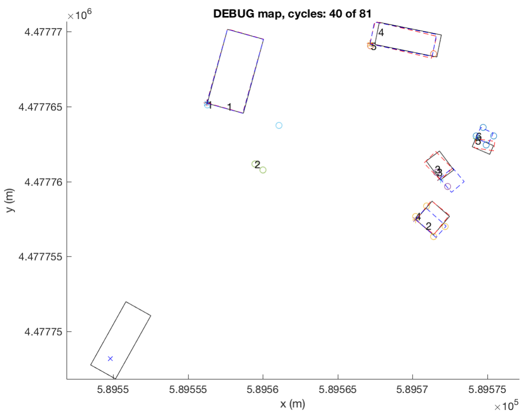


(d) Cycle 394. Detection of two clusters of the same object.

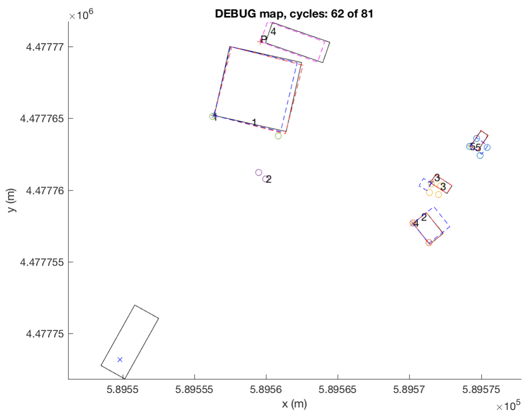
Figure 5.4: Results of Scenario 2: Interesting moments during the second part of the scene: Adapting and changing the reference point, detecting clusters. The red triangle marks the reference point as it is detected. The blue triangle marks the reference point after a possible change.

5.3.3 Time Consumption

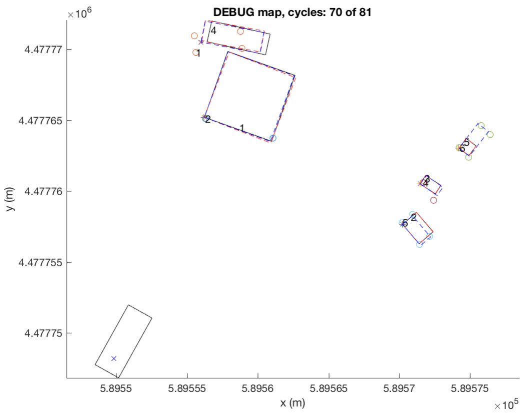
Table 5.6 shows the averaged time consumption of the algorithm on an Intel Core i7-4702MQ CPU at 2.20 GHz with 7.7 GiB memory for different data sets. The first two rows show the amount of points before and after cropping and the duration of the algorithm per 1,000 points. The last rows show the absolute and relative duration of each step of the algorithm. The last row shows the value for the whole algorithm.



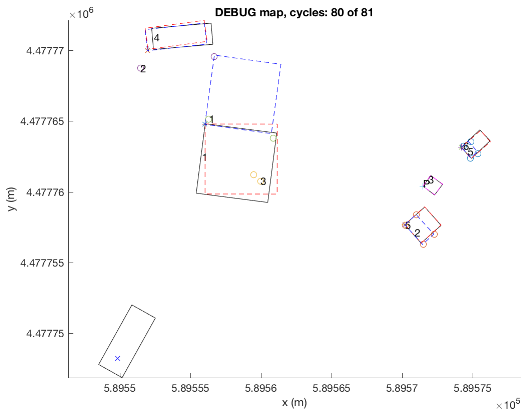
(a) Cycle 40. Beginning of the scene.



(b) Cycle 60. Vehicle is fully occluded.



(c) Cycle 70. Vehicle reappears.



(d) Cycle 80. End of the scene.

Figure 5.5: Results of Scenario 3: Key moments of this scenario. The vehicle drives behind an obstacle, is fully occluded while driving behind it, and reappears later.

Table 5.6: Time consumption of algorithm with different data sets.

	KITTI 0000 (151 cycles)		KITTI 0007 (800 cycles)		IBEO (151 cycles)	
	Amount	Duration	Amount	Duration	Amount	Duration
Average Points	4694.5629	0.7077 $\frac{s}{k\ points}$	3352.8300	0.6280 $\frac{s}{k\ points}$	1349.6623	3.5823 $\frac{s}{kpoints}$
Cropped Points	3853.1788	0.8623 $\frac{s}{k\ points}$	2929.5763	0.7188 $\frac{s}{k\ points}$	955.5762	5.0596 $\frac{s}{kpoints}$
	Absolute	Relative	Absolute	Relative	Absolute	Relative
Coord. Transform	0.0003 s	0.0079 %	0.0003 s	0.0120 %	0.0002 s	0.0047 %
Bounding Points	0.0005 s	0.0143 %	0.0004 s	0.0196 %	0.0004 s	0.0075 %
Segmentation	0.8920 s	26.8467 %	0.4068 s	19.3183 %	0.0694 s	1.4350 %
Classification	0.6367 s	19.1637 %	0.4159 s	19.7501 %	0.2672 s	5.5271 %
Postprocessing Classification	0.9721 s	29.2586 %	0.6487 s	30.8083 %	0.9161 s	18.9486 %
Feature Extraction	0.1367 s	4.1137 %	0.1018 s	4.8353 %	0.2341 s	4.8424 %
Postprocessing Feature Extraction	0.2358 s	7.0975 %	0.1710 s	8.1189 %	1.4289 s	29.5532 %
Association/Tracking	0.4484 s	13.4975 %	0.3609 s	17.1375 %	1.9185 s	39.6815 %
All	3.3224 s	100 %	2.1057 s	100 %	4.8349 s	100 %

6 Discussion

In this section, we want to discuss the test results by analyzing them and their meaning for the performance of the tracking algorithm. Afterwards, we will mention possible sources of errors and weaknesses of the algorithm and discuss time consumption.

6.1 Discussing the Results

The goal of this thesis was to develop a 2D LIDAR tracking algorithm that ideally performs as good as a 3D LIDAR tracking algorithm. Now, we discuss the evaluation parts separately and analyze how the algorithm performed.

6.1.1 Own Labeled Data

Table 5.2 shows MOTA and related values of the evaluation with own labeled data. The overall MOTA is 0.8741, which is a good value. We split the accuracy by classes and by objects the algorithms is supposed to track (TP, “Track”) and objects the algorithms is not supposed to track (true negative (TN), “No track”) for better analysis. The MOTA for “tracks” is 0.3655 and therefore only around 40 percent of the overall MOTA. This value could be better and out of all 145 objects, that should be tracked, only 53 were detected. In contrast to that bad ratio, the algorithm detected zero FPs. Almost two thirds missing targets and no FP might be indicators for too strict track creation criteria. We introduced those criteria to reduce the number of FPs. In order to demonstrate the effect of those criteria, we performed the evaluation without them. Table 6.1 shows the results. Without track creation criteria, the algorithm causes 94 FPs but only 30 missing targets and therefore a “track” MOTA of 0.7931 (with an overall MOTA of 0.8235). It seems like the algorithm is capable of detecting a large amount of objects to be tracked but causes a significant amount of FPs without track creation criteria.

Table 6.1: MOTA of own labeled data without track creation criteria.

	MOTA	FP	FN	ID	TP	TN	Num of objects
All	0.8235	94	30	5	115	492	731
Track	0.7931	0	30	0	115	0	145
No track	0.8311	94	0	5	0	492	586
L	0.5980	41	0	0	55	6	102
H	0.7581	21	7	2	32	64	124
V	0.8862	21	5	2	22	198	246
C	0.3125	9	1	1	3	3	16
U	0.6667	2	1	0	3	3	9
O	0.9316	0	16	0	0	218	234

Table 5.3 shows MOTP and related values. For better analysis, we split up all objects by "track"/"no track" objects and by classes as well. The overall MOTP is 56.28 percent, "track" MOTP is 63.85 percent, and the MOTP of L-shaped, C-, U-shaped, and O-clusters range between 74.27 and 87.89 percent. H- and V-clusters obtain the lowest values with 37.95 and 43.27 percent. Except for H- and V-clusters, the algorithm performs with a good MOTP. However, the absolute MOTP of L-shaped clusters is on average 25.95 centimeters and the absolute MOTP of H-clusters is 25.00 centimeters. The reason for low MOTP for H- and V- clusters might be that the area of the bounding box is significantly smaller compared to that of L-shaped, C-, or U-shaped clusters. The maximum of 14 meters occurred at a V- and "no track"-cluster. Such a large value is an outlier and it might be caused by an exterior wall of a building.

Table 5.4 shows MT/PT/ML metrics and the number of FMs. Our algorithm tracks 47.83 percent of all objects to be tracked more than 80 percent of the time, which represents a good but still improvable performance. That no object was tracked less than 20 percent is a good sign. However, it is noticeable that every partially tracked object has exactly one fragment. A small amount of FMs is good but it could also mean that the Kalman filter is not (or rarely) capable of resuming a track with prediction once a track is lost.

6.1.2 The KITTI Vision Benchmark Suite

In contrast to the more or less good performance of the own labeled data, all MOTA values of the KITTI data set are negative, which means that there are more errors than ground truth objects. This is mostly caused by a large amount of FPs. The recall is often around ten percent, once 30.42, once 21.88, but mostly less than five percent. As already mentioned, all training data sets show numerous FPs. Since the KITTI data set shows similar recall and FP values as the own labeled data, we tested all KITTI training data sets without track creation criteria as well. Table 6.2 shows these results. The MOTA is worse, the recall increases to a maximum of 46.86 percent and to some values around 20 percent but it is still too bad to be considered good. The KITTI training data sets show that the track creation criteria not only affect the amount of FPs but also the amount of TPs and it shows that it is harder to detect all objects compared to our own labeled data.

The MOTP is around 55 to 60 percent and, therefore, similar to the MOTP of our own labeled data.

The MT/PT/ML metrics are also different to those of the own labeled data. Almost every time, 0.00 percent of all objects were tracked most of the time. PT values fluctuate between zero, some around five, some around ten, and very few even around 30, 40, and 50 percent. However, most of all data sets show 80 and more percent of all tracks as mostly lost (ML).

It is noticeable, that the performance of the algorithm on the KITTI data sets is significantly worse than the performance on the own labeled data regarding MOTA and MT/PT/ML. Possible reasons are discussed in section 6.2. It shows a similar performance in MOTP. Nonetheless, the KITTI data sets again show that it is hard to distinguish between FPs and TPs.

Table 6.2: CLEARMOT and MT/PT/ML metrics of all KITTI training data sets without track creation criteria.

Training data set	MOTA	MOTP	Recall	MT	PT	ML	TP	FP	FN	ID	FM
0000	-4.2698	0.6535	0.4686	0.0000	0.4444	0.5556	127	989	144	0	6
0001	-0.9750	0.6258	0.0932	0.0000	0.1149	0.8851	219	2457	2131	0	14
0002	-0.7330	0.6438	0.0668	0.0000	0.1333	0.8667	68	782	950	1	6
0003	-1.1048	0.5445	0.1138	0.0000	0.2500	0.7500	38	407	296	0	5
0004	-1.0728	0.6379	0.0583	0.0000	0.1154	0.8846	45	865	727	2	6
0005	-1.0264	0.5940	0.0479	0.0000	0.0303	0.9697	58	1301	1154	1	8
0006	-1.1577	0.6034	0.1984	0.0000	0.4545	0.5455	101	671	408	2	9
0007	-1.5190	0.6188	0.2138	0.0000	0.4906	0.5094	425	3393	1563	4	36
0008	-0.8466	0.6120	0.0166	0.0000	0.0000	1.0000	17	873	1005	0	1
0009	-0.7759	0.6078	0.0818	0.0000	0.1013	0.8987	204	2115	2289	2	18
0010	-1.2806	0.6747	0.0551	0.0000	0.0000	1.0000	32	776	549	0	4
0011	-0.4889	0.6086	0.1059	0.0000	0.1176	0.8824	305	1691	2574	11	32
0012	-1.4056	0.6087	0.0490	0.0000	0.0000	1.0000	7	208	136	0	1
0013	-88.9200	0.5370	0.2857	0.0000	0.0000	1.0000	10	2223	25	0	0
0014	-1.1090	0.5802	0.0760	0.0000	0.1429	0.8571	32	482	389	0	2
0015	-2.3298	0.5679	0.0370	0.0000	0.1111	0.8889	21	1342	546	0	1
0016	-0.9043	Inf	0.0000	0.0000	0.0000	1.0000	0	756	836	0	0
0017	-Inf	Inf	0.0000	0.0000	0.0000	0.0000	0	1435	0	0	0
0018	-3.0351	0.5909	0.1878	0.0000	0.1667	0.8333	231	3940	999	4	16
0019	-13.7470	0.5879	0.1866	0.1667	0.3333	0.5000	159	11606	693	0	14
0020	-0.4563	0.6311	0.1312	0.0093	0.1963	0.7944	649	2894	4296	1	32

Table 6.3 shows all methods which ran the KITTI test data sets and were submitted online. A lot of them are anonymous submissions and most methods are vision-based. As expected, MOTA, MT, and ML values are better than ours. Our ID and FM are significantly smaller, which is per se good. Given the fact that our other quality metrics show a bad performance and the amount of ML tracks is large, the two metrics do not indicate a good performance either. The only metric which is good and comparable to other methods is MOTP although it is worse than all other methods.

6.1.3 Scenario 1 - Velocity

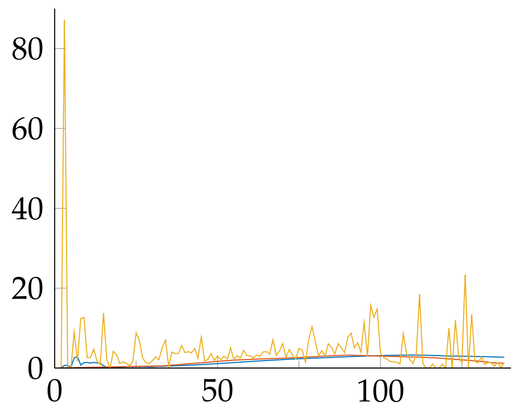
Figure 5.2 shows the velocity of ground truth and Kalman filter and the difference between them. One of the first values seems to be totally off because the estimated velocity is significantly larger than the ground truth. Figure 6.1a shows these two velocities and the calculated average velocity between two scans, Figure 6.1b shows the same data but better scaled. Figure 6.1a confirms the hypothesis because one of the first values is around $80\frac{m}{s}$ while the others being mostly less than $20\frac{m}{s}$. However, a Kalman filter naturally needs some time to adapt to the real value. After 20 cycles, the Kalman filter shows a good estimation for the ground truth velocity. In general, the estimated velocity seems to be shifted to the right or down. After a large deceleration at approximately cycle 85, the Kalman filter shows a slow response. Due to the adjustment in the beginning and the slow response in the end, the difference between ground truth and estimated velocity is at its highest values in the beginning and the end.

6.1.4 Scenario 2 - Changing Viewing Angle

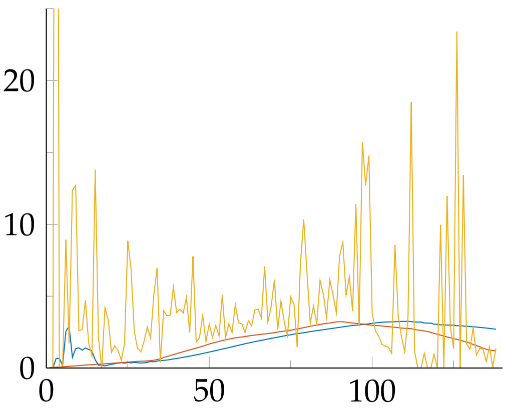
Figure 6.2a shows the first cycle of scenario 2. We would have expected an L-shaped cluster but the scan shows one thick and one thin line. After some cycles, the algorithm is capable of detecting the vehicle and recognizing the rear left corner as reference point (Figure 5.3a). At cycle 80, the perspective of the ego-vehicle and the shape of the standing car are changing. Here, also the reference point changes to the rear right corner. However, since the perceived orientation of the car also changes, the reference point relative to the received car is still in the rear left corner (Figure 5.3c). Some cycles later, the shape of the car looks like one line (for example in Figure 5.3d).

Table 6.3: Methods and results published on the KITTI website [47]

Method	MOTA	MOTP	MT	ML	ID	FM
IMMDP	75.37 %	82.74 %	60.37 %	10.82 %	178	382
TuSimple	73.20 %	83.97 %	71.65 %	7.01 %	300	515
wan	72.99 %	82.83 %	50.61 %	12.20 %	24	248
MCMOT-CPD	72.11 %	82.13 %	52.13 %	11.43 %	233	547
RBPF	71.44 %	82.25 %	63.87 %	5.49 %	284	673
DuEye	70.15 %	83.52 %	60.98 %	5.49 %	402	1043
NOMT*	69.73 %	79.46 %	56.25 %	12.96 %	36	225
CCF-MOT	69.46 %	78.36 %	52.29 %	13.11 %	72	408
MDP	69.35 %	82.10 %	51.37 %	13.11 %	135	401
DSM	68.66 %	84.39 %	41.46 %	15.40 %	29	418
NOMT-HM*	67.92 %	80.02 %	49.24 %	13.11 %	109	371
SLP*	67.36 %	78.79 %	53.81 %	9.45 %	65	574
SCEA*	67.11 %	79.39 %	52.13 %	10.98 %	106	466
LP-SSVM*	66.35 %	77.80 %	55.95 %	8.23 %	63	558
mbodSSP*	62.64 %	78.75 %	48.02 %	8.69 %	116	884
SSP*	60.84 %	78.55 %	53.81 %	7.93 %	191	966
NOMT	55.87 %	78.17 %	39.94 %	25.46 %	13	154
DCO-X*	55.49 %	78.85 %	36.74 %	14.02 %	323	984
ODAMOT	54.87 %	75.45 %	26.37 %	15.09 %	403	1298
NOMT-HM	53.03 %	78.65 %	33.23 %	27.13 %	28	250
RMOT*	53.03 %	75.42 %	39.48 %	10.06 %	215	742
LP-SSVM	51.80 %	76.93 %	35.06 %	21.49 %	16	430
SCEA	51.30 %	78.84 %	26.22 %	26.22 %	17	468
SSP	50.42 %	77.64 %	28.66 %	24.09 %	7	714
DBHM*	49.63 %	77.72 %	52.13 %	7.47 %	1632	2144
TBD	49.52 %	78.35 %	20.27 %	32.16 %	31	535
TDCS	49.25 %	75.20 %	23.02 %	21.34 %	126	991
mbodSSP	48.00 %	77.52 %	22.10 %	27.44 %	0	704
RMOT	46.63 %	75.18 %	20.43 %	31.40 %	51	382
CEM	44.31 %	77.11 %	19.51 %	31.40 %	125	398
MCF	43.17 %	78.25 %	14.33 %	37.04 %	23	589
HM	41.47 %	78.34 %	11.59 %	39.33 %	12	576
FMMOVT V2	37.46 %	80.05 %	20.27 %	30.79 %	588	1132
DP-MCF	35.72 %	78.41 %	16.92 %	35.67 %	2738	3239
FMMOVT	29.11 %	77.68 %	21.19 %	34.60 %	514	940
DCO	28.72 %	74.36 %	15.24 %	30.79 %	223	622



(a) GT velocity (red), velocity estimated by KF (blue), and the calculated average velocity between two cycles (yellow).



(b) Same data as in (6.1a) but scaled differently.

Figure 6.1: Scenario 1: GT, estimated, and average velocity between two cycles in two different scales.

Figure 5.4a shows the desired behavior when the reference point changes. The received reference point (red triangle) does not comply with the memorized reference point and is adjusted accordingly (blue triangle). Just a few cycles later, at cycle 154 (Figure 5.4b), the perceived orientation (and location) of the car changes again and the global reference point is now the front right corner. Relative to the car's orientation, it is still the rear left corner. During the next cycles, cycle 222, 232, and 237 are depicted in Figure 6.2, detection of the other car completely fails due to blind spots in the ego-vehicles vicinity (see also Figure 3.2b) and, as a result, oddly shaped clusters. During these cycles, testing is barely possible. For example the other car is, contrary to expectations, represented as two line-shaped clusters or as a P-shaped cluster as depicted in Figure 6.2e. In the end of the scenario, in cycle 310 and depicted in Figure 5.4c, the detection adaptation of the reference point works correctly. However, in the last cycles, cycle 394 is depicted in Figure 5.4d, the detection is completely wrong and two lines of the other car are detected as two separate objects.

In order to sum up, the cycles were not perfect for testing the method for changing the reference point. However, during the sequences which were qualified for testing, we conclude that the algorithm is capable to adapt the reference point if the orientation of the car is not changing. As soon as the perceived orientation changes, the reference point changes as well and all future assignments are wrong.

6.1.5 Scenario 3 - Occlusion

Figure 5.5 shows key moments of scenario 3. In the beginning, the detection of all vehicles is successful (Figure 5.5a). Once the moving vehicle is fully occluded, the prediction of the Kalman filter is used to complete motion history (Figure 5.5b). The Kalman filter predicts x- and y-position only with x- and y-velocity. Hence, the motion is predicted straight and does not work in a curve with the current implementation. The occluded moving vehicle steers and drives faster than the prediction which can be seen when the prediction and real object match again in Figure 5.5c. In the following cycles, the Kalman filter shows a slow response after "catching" the object again (Figure 5.5d).

6.2 Possible Sources of Errors

With our own labeled data, the algorithm achieves either a good MOTA or a good recall. It has difficulty distinguishing between FP and TP objects, i.e. non-vehicle L-shaped clusters are recognized as vehicles. At the current implementation state, either considering all objects as tracks or choosing certain objects to be tracks causes one of those two quality metrics to represent a bad performance.

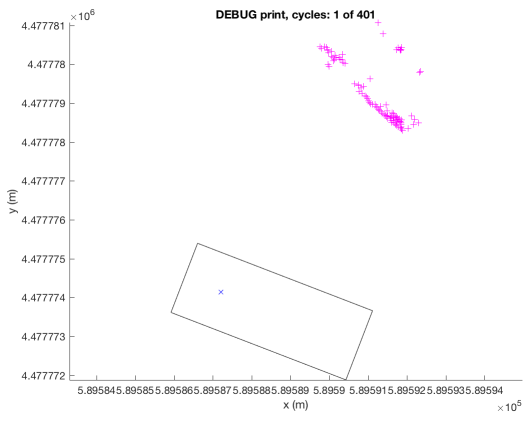
Maybe the most obvious insight after testing with the KITTI dataset is that the algorithm performed significantly worse compared to own labeled data. It should be mentioned again that the database is called "The KITTI **Vision** Benchmark Suite". Admittedly, they provide LIDAR data for download but the benchmark is based on the camera image. The CLEARMOT metrics are also vision-based. Reason for a bad recall value might be that there are a lot of cars in the KITTI dataset which are far away (for example the van in Figure 6.3a), that there are a lot of parked and therefore occluded cars (for example in Figure 6.3c and d), and that there are annotated vehicles which are not on the street (for example in Figure 6.3d). In addition, in order to simulate 2D LIDAR data, we cut out a z-slide from all LIDAR data. This does not represent the same scene scanned with multiple 2D LRFs but is the best estimation.

As clearly seen in scenario 3, the Kalman filter itself is not capable of predicting a curvature. With x, y, v_x, v_y, ϕ, and α as state vector, it is possible to predict a straight movement and the heading but not the movement in the direction of heading.

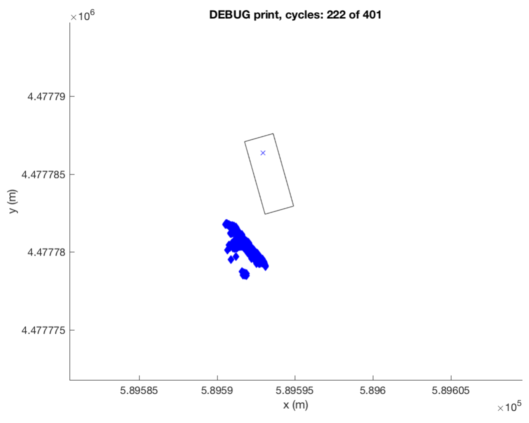
Further calculations like adapting the movement in heading direction or determining acceleration out of motion history would enable a prediction of a curvy movement but are not part of the current implementation.

6.3 Time Consumption

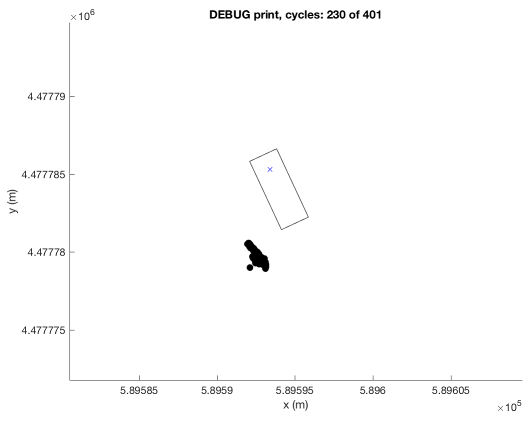
Table 5.6 shows that the average duration of one cycle depends on the data set. Data set “KITTI 0000” needs 3.3 seconds for 4,695 LIDAR points, “KITTI 0007” needs 2.1 seconds for 3,353 points, and data set “IBEO” 1.9 seconds for 1,349 points. The more points a scan has the more time the algorithm needs. Dataset “KITTI 0000” has a similar time per 1,000 points as “KITTI 0007” but on average more points and therefore needs more time. However, the first 151 cycles of our data set has the least amount of points in the comparison but takes the longest time. The reason is that these scans involve more cars and therefore more points the algorithm needs to process after the classification stage. Every cluster classified as “O” is not processed after classification. However, independent of the exact composition of the scans, the algorithm takes too much time. A rule-of-thumb estimation says that a C++ implementation needs a tenth of the time a MATLAB implementation needs, which would be 0.19 seconds. With a frequency of 12.5 Hz the algorithm needs to run faster than $\frac{1}{12.5\,Hz} = 0.08\,s$ when excecuted online.



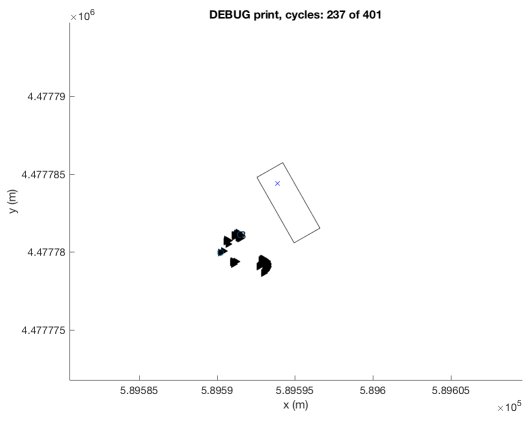
(a) Cycle 1. The very first cycle of the scene shows some oddly shaped cluster.



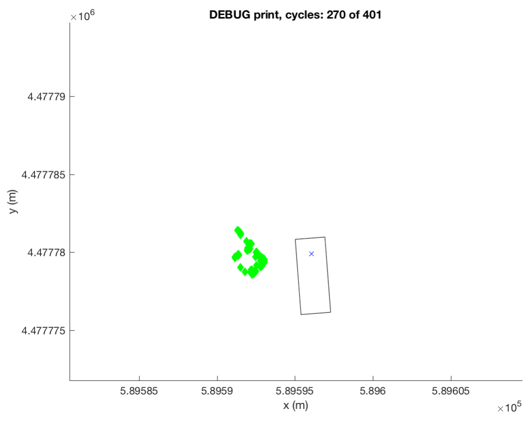
(b) Cycle 222. Oddly shaped cluster before blind spot interferes.



(c) Cycle 232. Blind spot causes closes part of object to be unseen.

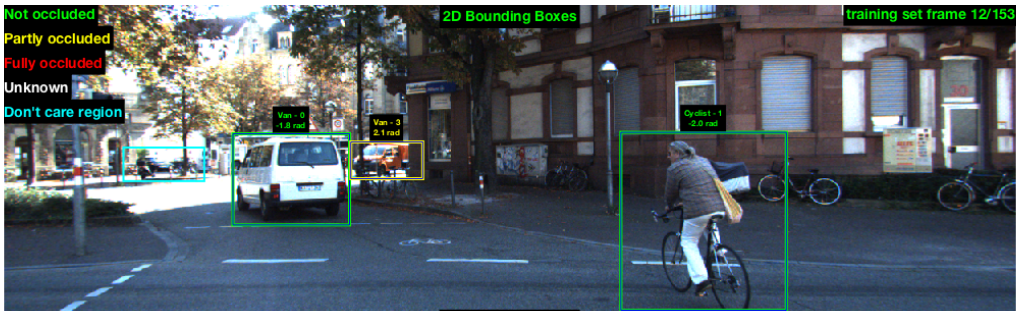


(d) Cycle 237. Blind spot splits object in half.

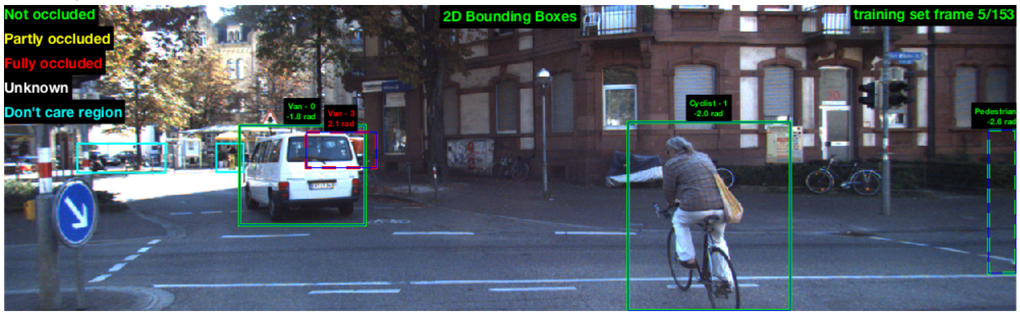


(e) Cycle 270. P-shaped cluster.

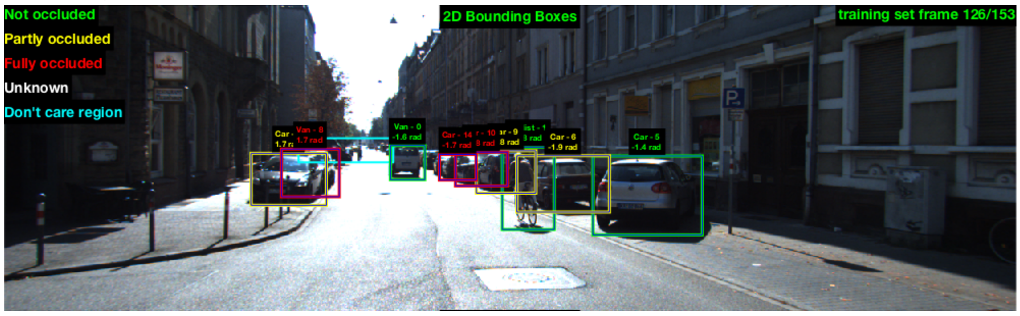
Figure 6.2: Scenario 2: Cycles that caused the algorithm to fail. (a) shows the first cycle and the unexpected cluster shape. (b)-(d) show how blind spots interfere with clusters. (e) shows an unexpected P-shaped cluster.



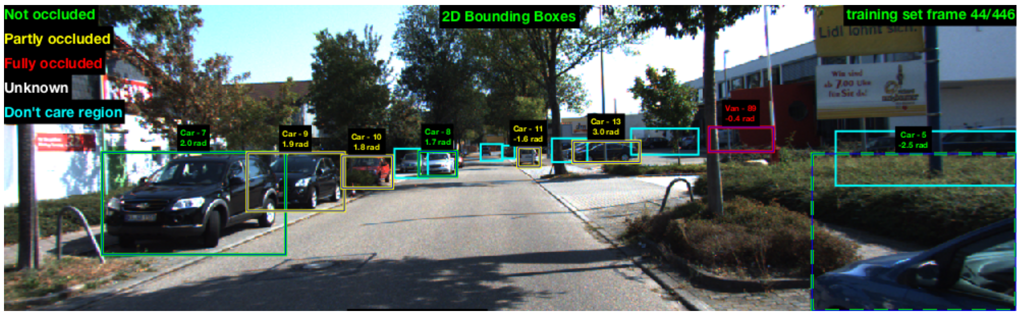
(a) Cycle 12 of first KITTI training data set. The van is too far away for the algorithm to detect it.



(b) Cycle 5 of first KITTI training data set. The same van's first annotation as “fully occluded”.



(c) Cycle 126 of first KITTI training data set. Many parked and therefore occluded cars.



(d) Cycle 44 of second KITTI training data set. Many parked and occluded cars, some of them are not on the street.

Figure 6.3: Example camera pictures of urban scenes from the first and second KITTI training data set.

7 Conclusion

Finally, we summarize the implemented algorithm, how we tested its performance, and the test results. We suggest future steps and research and conclude the thesis.

Current technology used to equip vehicles to drive autonomously often includes 3D LRFs which are too expensive to mass-market. That is the reason why this thesis describes the attempt to implement a tracking algorithm based on data from multiple, more affordable, 2D LRFs which ideally achieve a similar performance as a 3D tracking algorithm. The task has included tracking multiple objects, moving sensors, real-world data, and coping with related problems. Further, we focused on reducing over-segmentation and the so-called shape change problem. Our autonomous driving research platform is equipped, among others, with six LRFs and two GPS sensors of which we used one. After receiving data points, we perform a coordinate transformation and crop distant points. To identify objects in the scan, we perform a segmentation. The resulting clusters are classified into one of five classes according to their geometry. If necessary, we change some classes with help of track information and, using current geometry information, merge clusters which seem to belong together. After that and when ideally every cluster represents one object, we extract features to get stable and meaningful information about each relevant object. With this information, we can determine spatial magnitudes and rule out objects which are too small or too large to be a vehicle. All these steps are performed for every single scan. In order to determine the motion history of an object, we associate a cluster of a scan with the track respectively the cluster of the previous scan. With the information from motion history and the new cluster, we try to prevent the shape change problem by changing the location of our perceived reference point relative to the object. In addition to all preliminary steps, we implemented a track management where we decide what objects to track over time, what tracks are updated with what object in the case of ambiguity, and when to delete tracks whose objects are lost.

In order to test the performance of our algorithm, we used two groups of quality metrics: CLEARMOT and MT/PT/ML metrics. We applied these metrics on own labeled data and on training data sets of the KITTI Vision Benchmark Suite. Velocity estimation, changing the reference point, and occlusion handling were tested with specific scenarios. On our own labeled data, the algorithm performed well with a MOTA of 0.79 without track creation criteria, a MOTP of approximately 60 percent, and approximately 50 percent mostly tracked tracks. In contrast, the algorithm performed bad on KITTI data sets: It achieved a negative MOTA, a MOTP of approximately 60 percent, and zero percent mostly tracked tracks. The first scenario, testing the velocity estimation, shows that the Kalman filter is able to provide a more or less good estimation of the velocity of other vehicles but responses slowly after large velocity changes. Scenario 2, which tests the change of reference point, shows that our concept works locally but not globally. The last scenario, the occlusion case, shows that the Kalman filter is able to predict the movement of a fully occluded car but the current implementation is fragile and does not provide a curvy prediction.

One of the next steps to improve the algorithm could be to implement the algorithm in for example C++ in order to test the algorithm in real-time. To neglect points which are not on the road, we cut the scan off at the lateral borders. A better approach would be to cut the scan off exactly at the road boundary. We attempted to use OpenStreetMap but those maps do not deliver enough information. One could use another map or GPS-based solution but since the actual task was to use only LIDAR, a future endeavor could be to extract road boundaries using only LIDAR like [13], [17], or [48] did. One great weakness of the algorithm is that it has a hard time detecting off-road vehicles which was one reason for bad recall values on the KITTI data sets. Whether a tracking algorithm should be able to detect off-road vehicles is a matter of its requirements. Our algorithm classifies all those objects based on geometric features. With enough labeled data, a trained classifier would probably provide a better performance since all objects are easily distinguishable from each other by their geometry. However, not all tracking algorithms comprise a classification step. Because of that and since H- and V-clusters are treated the exact same way, it might be a good idea to refrain from classification and save that time. As already mentioned in section 6.2, the KITTI Vision Benchmark Suite is not a LIDAR benchmark. In order to adequately test and compare LIDAR tracking algorithms, a corresponding benchmark is needed. One of the very first objectives of this thesis was to only use

LIDAR information. We then added GPS to the requirements in order to transform points into world coordinates. To achieve this objective, one solution would be a compensation of ego-motion to get a "global" representation instead of using the GPS position of the ego-vehicle. Another idea for future research would be reduction of under-segmentation. Usually, tracking algorithms try to reduce over-segmentation. Both alterations affect subsequent steps (mostly classification and later ambiguities at data association). However, one idea is to adapt the segmentation parameters and the attempt to improve segmentation results with reduction of over- and under-segmentation. Using multiple sensors comes with its own peculiarities. Some objects are seen by multiple sensors from different angles. There are two options how to process the data: Combining the data and processing all information (like we did) or processing sensor data separately and combining the information afterwards (like [11] did, the same applies for one sensor with multiple layers). The authors are not aware of a comparison or what the advantages and disadvantages of the particular methods are. Whether one of these or a combination of both methods yields the best result could be part of future research. However, a LIDAR of exactly one laser beam shows typical LIDAR characteristics: i.e. the larger the distance to the sensor the larger the distance between two rays. This fact should also taken into account when processing scans separately. Another characteristic based on the working principle of a LRF which is not considered during segmentation is that every point in a scan has its own timestamp. This affects the scan if an object is moving while it is scanned. However, noise has probably a larger effect than the object's movement during one scan cycle. Since 2D LIDAR scans are rather sparse compared to 3D scans, one could also overlap multiple scans and perform segmentation and classification on that result. However, movements during the scans need to be compensated. Our algorithm's biggest problem was classifying non-vehicle objects as vehicles and not detecting vehicle objects that could be clutter, buildings, or off-road objects. To improve the performance of our algorithm, we suggest not just using spatial but also temporal, and semantic information (like [24]). Our last suggestion aims at improving ambiguity handling. Our first two approaches were based on reachable sets and multiple hypothesis tracking (MHT). Reachable sets include multiple future outcomes we based our assignments on. MHT uses the measurements as future outcomes and uses all assignment permutations. We realized that our implementation of the reachable sets was prone to noise and outliers and that MHT is not well suited for tracking inert and easily predictable objects that cars are. Our current solution compares one prediction (the one given by the Kalman filter) with measurements. We think that multiple predictions especially in an occlusion case could yield better results.

We conclude that our algorithm did not perform as good as we hoped and worse than every submitted method tested on the KITTI data sets. The MOTA was either good or bad depending what labeled data was used for ground truth. If it includes parked cars and off-road vehicles, MOTA is bad. On the other side, MOTP was stable 60 percent independent of the used data set. This shows that the distinction between vehicle and other rectangular objects is one of the biggest problems. However, solely tracking one object including occlusion and velocity estimation works. Our algorithm for tracking multiple objects with multiple moving 2D LRFs does not reach 3D performance but has still a lot of potential. The incentive of this thesis was to see how well a 2D LIDAR tracking algorithm can perform and what deficiencies need to be remedied. If 2D LRFs will be used in autonomous vehicles, the result of sensor fusion will be better if much information can already obtained by 2D LIDAR but eventually its shortcomings will probably be compensated by other types of sensors.

Bibliography

[1] David Held, Jesse Levinson, and Sebastian Thrun. Precision Tracking with Sparse 3D and Dense Color 2D Data. In *2013 IEEE International Conference on Robotics and Automation*, pages 1138–1145. IEEE, may 2013. doi: 10.1109/ICRA.2013.6630715.

[2] Michael Darms, Paul Rybski, Christopher R Baker, and Christopher Urmson. Obstacle Detection and Tracking for the Urban Challenge. *IEEE Transactions on Intelligent Transportation Systems*, 10(3):475–485, 2009.

[3] Gary Silberg, Richard Wallace, G Matuszak, J Plessers, C Brower, and D Subramanian. Self-driving cars: The Next Revolution. *White paper, KPMG LLP & Center of Automotive Research*, page 36, 2012.

[4] J.D. Power and Others. US Automotive Emerging Technologies Study. *J.D. Power and Associates*, 2014.

[5] Evan Ackerman. Lidar that will make Self-Driving Cars Affordable [News]. *IEEE Spectrum*, 53(10):14–14, 2016.

[6] Alexander Hars. Driverless car market watch | Gearing up to save lives, reduce costs, resource consumption, 2017. URL `http://www.driverless-future.com/?page_id=384`. [Online; accessed 15 Jan 2017].

[7] MATLAB. *version 9.0.0.341360 (R2016a)*. The MathWorks Inc., Natick, Massachusetts, United States, 2016.

[8] Erwin Prassler, Jens Scholz, and Alberto Elfes. Tracking Multiple Moving Objects for Real-Time Robot Navigation. *Autonomous Robots*, 8(2):105–116, 2000. doi: 10.1023/A:1008997110534.

[9] Chieh-Chih Wang, Charles Thorpe, and Arne Suppe. Ladar-based Detection and Tracking of Moving Objects from a Ground Vehicle at High Speeds. In *Intelligent Vehicles Symposium, 2003. Proceedings. IEEE*, pages 416–421, 2003.

[10] Abel Mendes, Luis Conde Bento, and Urbano Nunes. Multi-Target Detection and Tracking with a Laser Scanner. In *Intelligent Vehicles Symposium, 2004 IEEE*, pages 796–801, 2004.

[11] Robert MacLachlan and Christoph Mertz. Tracking of Moving Objects from a Moving Vehicle Using a Scanning Laser Rangefinder. In *Intelligent Transportation Systems 2006*, volume 2006, pages 301–306. IEEE, 2006.

[12] Cristiano Premebida, Goncalo Monteiro, Urbano Nunes, and Paulo Peixoto. A Lidar and Vision-based Approach for Pedestrian and Vehicle Detection and Tracking. In *2007 IEEE Intelligent Transportation Systems Conference*, pages 1044–1049. IEEE, 2007. doi: 10.1109/ITSC.2007.4357637.

[13] Bin Gao and Benjamin Coifman. A Vehicle Detection and Tracking Approach Using Probe Vehicle LIDAR Data. In *Traffic and Granular Flow'05*, pages 675–685. Springer, 2007.

[14] Christoph Mertz, Luis Ernesto Navarro-Serment, Robert MacLachlan, Paul Rybski, Aaron Steinfeld, Arne Suppé, Christopher Urmson, Nicolas Vandapel, Martial Hebert, Chuck Thorpe, David Duggins, and Jay Gowdy. Moving Object Detection With Laser Scanners. *Journal of Field Robotics*, 30(1):17–43, 2013.

[15] Geovany Araujo Borges and Marie-José Aldon. A Split-And-Merge Segmentation Algorithm for Line Extraction in 2D Range Images. In *Proceedings 15th International Conference on Pattern Recognition. ICPR-2000*, volume 1, pages 441–444, Barcelona, 2000. IEEE Comput. Soc. doi: 10.1109/ICPR.2000.905371.

[16] Geovany Araujo Borges and Marie-José. Aldon. Line Extraction in 2D Range Images for Mobile Robotics. *Journal of intelligent and Robotic Systems*, 40(3): 267–297, 2004. doi: 10.1023/B:JINT.0000038945.55712.65.

[17] Jan Sparbert, Klaus Dietmayer, and Daniel Streller. Lane Detection and Street Type Classification Using Laser Range Images. In *Intelligent Transportation Systems, 2001. Proceedings. 2001 IEEE*, pages 454–459, 2001.

[18] Klaus Dietmayer, Jan Sparbert, and Daniel Streller. Model Based Object Classification and Tracking in Traffic Scenes from Range Images. In *Proceedings of IV IEEE Intelligent Vehicles Symposium*, pages 1–6, 2001.

[19] Martin D Adams. On-line Gradient Based Surface Discontinuity Detection for Outdoor Scanning Range Sensors. In *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, pages 1726–1731, 2001.

[20] Sérgio Santos, José Eduardo Faria, Fernando Soares, Rui Araújo, and Urbano Nunes. Tracking of Multi-Obstacles With Laser Range Data for Autonomous Vehicles. In *Proc. 3rd National Festival of Robotics Scientific Meeting (ROBOTICA)*, pages 59–65, 2003.

[21] Kenneth Jay Lee. Reactive Navigation for an Outdoor Autonomous Vehicle. *Master's Thesis, University of Sydney, Sydney, Australia*, 2001.

[22] Cristiano Premebida and Urbano Nunes. Segmentation and Geometric Primitives Extraction From 2d Laser Range Data for Mobile Robot Applications. Technical report, Instituto de Sistemas de Robótica – Pólo de Coimbra, 2005.

[23] Klaas Klasing, Dirk Wollherr, and Martin Buss. A Clustering Method for Efficient Segmentation of 3D Laser Data. In *ICRA*, pages 4043–4048, 2008.

[24] David Held, Devin Guillory, Brice Rebsamen, Sebastian Thrun, and Silvio Savarese. A Probabilistic Framework for Real-time 3D Segmentation using Spatial, Temporal, and Semantic Cues. *Proceedings of Robotics: Science and Systems*, 2016.

[25] A. K. Jain, M. N. Murty, and P. J. Flynn. Data Clustering: A Review. *ACM Computing Surveys*, 31(3):264–323, 1999. doi: 10.1145/331499.331504.

[26] Rui Xu and D. WunschII. Survey of Clustering Algorithms. *IEEE Transactions on Neural Networks*, 16(3):645–678, may 2005. doi: 10.1109/TNN.2005.845141.

[27] Stergios I. Roumeliotis and George A Bekey. Segments: A Layered, Dual-Kalman Filter Algorithm for Indoor Feature Extraction. In *Intelligent Robots and Systems, 2000.(IROS 2000). Proceedings. 2000 IEEE/RSJ International Conference on*, pages 454–461, 2000.

[28] Fawzi Nashashibi and Alexandre Bargeton. Laser-Based Vehicles Tracking and Classification Using Occlusion Reasoning and Confidence Estimation. In *2008 IEEE Intelligent Vehicles Symposium*, pages 847–852. IEEE, 2008. doi: 10.1109/IVS. 2008.4621244.

[29] J. Vandorpe, H. Van Brussel, and H. Xu. Exact Dynamic Map Building for A Mobile Robot Using Geometrical Primitives Produced By A 2D Range Finder. In *Proceedings of IEEE International Conference on Robotics and Automation*, volume 1, pages 901–908. IEEE, 1996. doi: 10.1109/ROBOT.1996.503887.

[30] Junqing Wei, Jarrod M Snider, Junsung Kim, John M Dolan, Raj Rajkumar, and Bakhtiar Litkouhi. Towards a Viable Autonomous Driving Research Platform. In *Proceedings of the 2013 IEEE Intelligent Vehicles Symposium*, pages 763–770. IEEE, 2013.

[31] Applanix Corporation Fact Sheet. *POSLV Specifications*. Applanix, 2015.

[32] Ibeo Fact Sheet. *ibeo LUX (model 2010) - Technical facts...* Ibeo Automotive Systems GmbH, Hamburg, 2010.

[33] Ibeo Manual. *Operating Manual ibeo LUX 2010 Laserscanner*. Ibeo Automotive Systems GmbH, Hamburg, 2010.

[34] Martin Ester, Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases With Noise. *Kdd*, 96(34):226–231, 1996.

[35] Adrien Marie Legendre. *Nouvelles méthodes pour la détermination des orbites des comètes*, volume 1. F. Didot, 1805.

[36] Carl Friedrich Gauss. *Theoria motus corporum coelestium in sectionibus conicis solem ambientium auctore Carolo Friderico Gauss*. sumtibus Frid. Perthes et IH Besser, 1809.

[37] Francis Galton. Regression Towards Mediocrity in Hereditary Stature. *The Journal of the Anthropological Institute of Great Britain and Ireland*, 15:246–263, 1886.

[38] Herbert Edelsbrunner, David Kirkpatrick, and Raimund Seidel. On The Shape of A Set of Points In The Plane. *IEEE Transactions on information theory*, 29(4): 551–559, 1983.

[39] R. J. Adcock. A Problem in Least Squares. *The Analyst*, 5(2):53–54, 1878.

[40] Wayne A. Fuller. *Measurement Error Models*. John Wiley & Sons, Inc., 1987.

[41] Urs Ramer. An Iterative Procedure for The Polygonal Approximation of Plane Curves. *Computer Graphics and Image Processing*, 1(3):244–256, 1972. doi: 10.1016/S0146-664X(72)80017-0.

[42] R. E. Kalman. A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering*, 82(1):35, 1960. doi: 10.1115/1.3662552.

[43] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are We Ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[44] Anton Milan, Konrad Schindler, and Stefan Roth. Challenges of Ground Truth Evaluation of Multi-Target Tracking. In *2013 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 735–742. IEEE, 2013. doi: 10.1109/CVPRW.2013.111.

[45] Keni Bernardin and Rainer Stiefelhagen. Evaluating Multiple Object Tracking Performance : The CLEAR MOT Metrics. *EURASIP Journal on Image and Video Processing*, 2008(1):1–10, 2008. doi: 10.1155/2008/246309.

[46] Yuan Li, Chang Huang, and Ram Nevatia. Learning to Associate: Hybridboosted Multi-Target Tracker for Crowded Scene. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2009*, pages 2953–2960, 2009. doi: 10.1109/CVPRW.2009.5206735.

[47] Andreas Geiger. The KITTI Vision Benchmark Suite, 2017. URL `http://www.cvlibs.net/datasets/kitti/eval_tracking.php`. [Online; accessed 10 Jan 2017].

[48] W.S. Wijesoma, K.R.S. Kodagoda, and A.P. Balasuriya. Road-Boundary Detection and Tracking Using Ladar Sensing. *IEEE Transactions on Robotics and Automation*, 20(3):456–464, 2004. doi: 10.1109/TRA.2004.825269.

List of Figures

List of Tables

List of Abbreviations

2D	Two-dimensional
3D	Three-dimensional
API	Application program interface
BB	Bounding box
CLEAR	Classification of events, activities, and relationships
DARPA	Defense Advanced Research Projects Agency
DATMO	Detection and tracking of moving objects
DBSCAN	Density-based spatial clustering of applications with noise
EKF	Extended Kalman filter
FM	Fragments
FN	False negative
FoV	Field of view
FP	False positive
GPS	Global Positioning System
GT	Ground truth
H	Hypothesis
ID	Id switch
IEPF	Iterative end-point fit
IMM	Interacting multiple model
***k*-NN**	K-nearest neighbor
KF	Kalman filter

KFBS	Kalman-filter-based segmentation
LASER	Light amplification by stimulated emission of radiation
LIDAR	Light Detection and Ranging
LMS	Laser measurement system or laser measurement sensor
LRF	Laser range finder
LT	Line tracking
MHT	Multiple hypothesis tracking
ML	Mostly lost
MOT	Multiple object tracking
MOTA	MOT accuracy
MOTP	MOT precision
MT	Mostly tracked
OLS	Ordinary least squares
PDBS	Point-distance-based segmentation
PT	Partially tracked
RADAR	Radio Detection and Ranging
RBNN	Radially bounded nearest neighbor
SLAM	Simultaneous localization and mapping
SSR	Sum of squared residuals
STD	Standard deviation
TN	True negative
TP	True positive
TTC	Time to collision

List of Symbols

Lowercase Symbols

a	Acceleration
$\bar{d}$	Distance function
m	Number of matches
n	Number of data points, number of current scan
r	Radius
t	Time
$\mathbf{u}$	Control vector of Kalman filter
v	Velocity
x	X-coordinate
$\mathbf{x}$	State vector of Kalman filter
y	Y-coordinate

Uppercase Symbols

$\mathbf{A}$	State transition matrix of Kalman filter
$\mathbf{B}$	Control matrix of Kalman filter
D	Distance
D_{thd}	Distance threshold
$\mathbf{H}$	Observation matrix of Kalman filter
$\mathbf{P}$	Probability matrix of Kalman filter
$\mathbf{Q}$	Process covariance matrix of Kalman filter
$\mathbf{R}$	Measurement covariance matrix of Kalman filter

Greek Symbols

α	Y-intercept of regression line, Angular acceleration
β	Slope of regression line
δ	Ratio of variances
ϵ	Residual of y-coordinate
η	Residual of x-coordinate
ω	Angular velocity
ϕ	Angle
σ	Variance
Δ	Difference / Change
Σ	Sum