Development of a Detection and Tracking of Moving Vehicles system for 2D LIDAR sensors

Konstantinos Konstantinidis

Master of Science Thesis

TU Delft Delft University of Technology

Delft Center for Systems and Control

Development of a Detection and Tracking of Moving Vehicles system for 2D LIDAR sensors

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

Konstantinos Konstantinidis

February 7, 2020

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Abstract

The main objective of this thesis was the development and evaluation of a Detection and Tracking of Moving Objects (DATMO) system, that is capable of reliably tracking nearby vehicles from a moving car. The developed system takes in raw 2D LIght Detection And Ranging (LIDAR) measurements as input and detects objects of interest by clustering them with the Adaptive Breakpoint Detector algorithm. The resulting clusters are fitted with oriented bounding boxes, by incorporating the Search-Based Rectangle Fitting algorithm.

The tracking part of the system receives, extracted from the rectangles, L-shapes and associates them with already tracked vehicles, using the Global Nearest Neighbor algorithm. However, since LIDAR measures only the distance to surfaces that face the sensor, vehicle appearances change over time. In order to counteract tracking errors that originate from these changes, an L-shape switching algorithm is implemented. The kinematic poses of the tracked vehicles are estimated with two different tracking filters, a Kalman Filter (KF), with a constant velocity model and an Unscented Kalman Filter (UKF), with a Coordinated Turn kinematic model. The dimensions of the detected vehicles are estimated with a constant shape Kalman Filter.

The proposed system was evaluated using both simulation and real-world experiments. The real-world experiments were made on the Delft Scaled Vehicle (DSV), an experimental car platform in the form of a 1/10 scale radio controlled car, and the ground truth data were provided by a Motion Capture (MoCap) system. The simulation experiments were made in a highway environment, which was created specifically for the development and testing of this system. Evaluating the experiment results reveals that the developed system can reliably estimate the position, speed, heading angle and dimensions of surrounding vehicles and therefore it can be used in similar research platforms to expand their environment perception capabilities.

In addition, during the course of my thesis it became important to develop a localization system for the DSV aimed at demonstration purposes. The resulting localization system is presented in the first appendix.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, the general software architecture of a self-driving vehicle is presented, followed by a quick introduction to the specific task that the work of this thesis seeks to fulfill. Next, the three main approaches towards it are summarized and a choice is made. Lastly, a brief overview of the whole MSc thesis is given.

In recent years, a lot of researchers have focused their efforts on providing solutions to the various problems that need to be addressed before self-driving vehicles become a reality. Generally, three different components need to work in cooperation for a vehicle to be self-driving (Figure 1-1). The first component, is environment perception, which takes as input the various sensor measurements and makes a representation of the surrounding environment. This representation of the surrounding environment is used by the Reasoning and Decision system to produce a set of actions that best correspond to the given situation, which are then given to the Action component, that produces instructions for the vehicle actuators. This MSc thesis will focus on the development of a sub-component of environment perception.

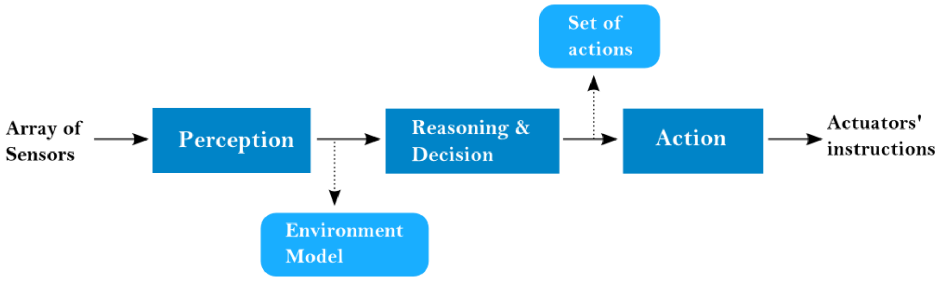


Figure 1-1: General architecture of an autonomous driving system. Source [1].

Environment perception can additionally be subdivided in three distinct but codependent tasks [10]. The first task is localization, in which the vehicle localizes itself in the environment by establishing the spatial relationships between itself and stationary objects. The second one is mapping, in which a map of the environment is built by establishing the spatial relationships

between surrounding static objects. The last task and the one that this MSc thesis will focus on, is Detection and Tracking of Moving Objects (DATMO), which establishes the spatial and temporal relationships between the vehicle and moving objects.

1-1 Problem Statement: Detection and Tracking of Moving Objects (DATMO)

Given a set of sensor measurements (LIDAR, Radar, Camera), a DATMO system is responsible for detecting the various moving objects in the vehicle's environment and estimating their position, speed and orientation. While, some DATMO systems are capable of additionally estimating the object's dimensions and classifying them in predetermined classes (pedestrian, bicycle, car).

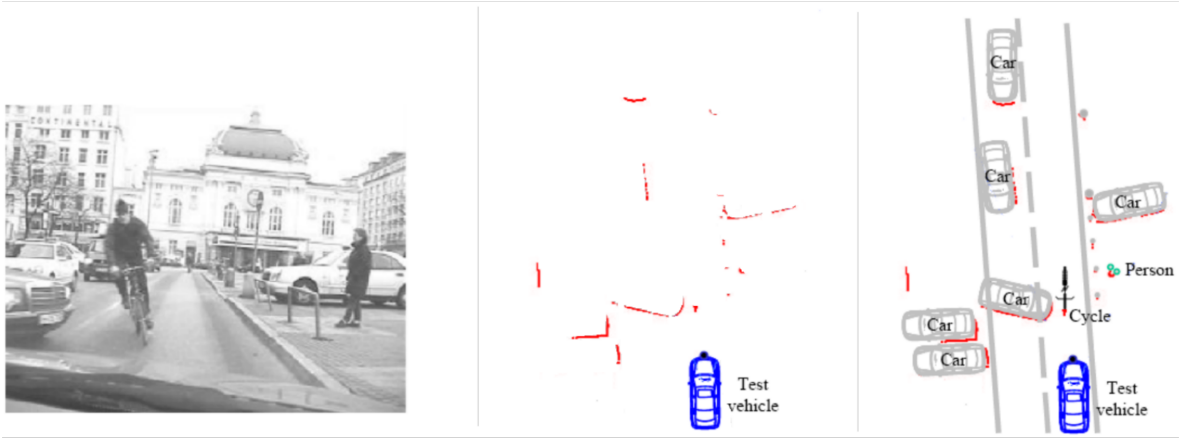


Figure 1-2: Visualization of the input and output of an environment perception system; (a) image of the actual scene; (b) raw measurements from a LIDAR sensor mounted on the test vehicle; (c) output of a DATMO system, overlayed on the LIDAR measusements. Source [1].

In Figure 1-2 the input and output of a DATMO system are visualized. On the left there is an image from a camera sensor mounted on the vehicle and in the middle is the measurements from the 2D LIDAR sensor that is also mounted on the vehicle. On the right, the output of a DATMO system is visualized, in which the LIDAR measuremnts are divided in groups, which correspond to objects from the actual scene and are classified in classes by fusing information from the camera.

In the developed system, the 2D LIDAR measurements will be used to estimate the position $(x,\ y)$, velocity $(v_x,\ v_y)$, orientation (ψ), turn rate (ω) and dimensions (Length, Width) of every detected vehicle. The resulting state vectors $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ that needs to be estimated are the following:

$$\mathbf{x}_1 = \begin{bmatrix} x_{center} & y_{center} & v_x & v_y & \psi & \omega & Length & Width \end{bmatrix}^T$$
$$\mathbf{x}_2 = \begin{bmatrix} x_{center} & y_{center} & v_x & v_y & \psi & \omega & Length & Width \end{bmatrix}^T$$
$$\vdots$$
$$\mathbf{x}_n = \begin{bmatrix} x_{center} & y_{center} & v_x & v_y & \psi & \omega & Length & Width \end{bmatrix}^T$$

,where n is the number of detected vehicles, which usually changes frequently between time instances. In Figure 1-3 the eight states that need be estimated are visualized.

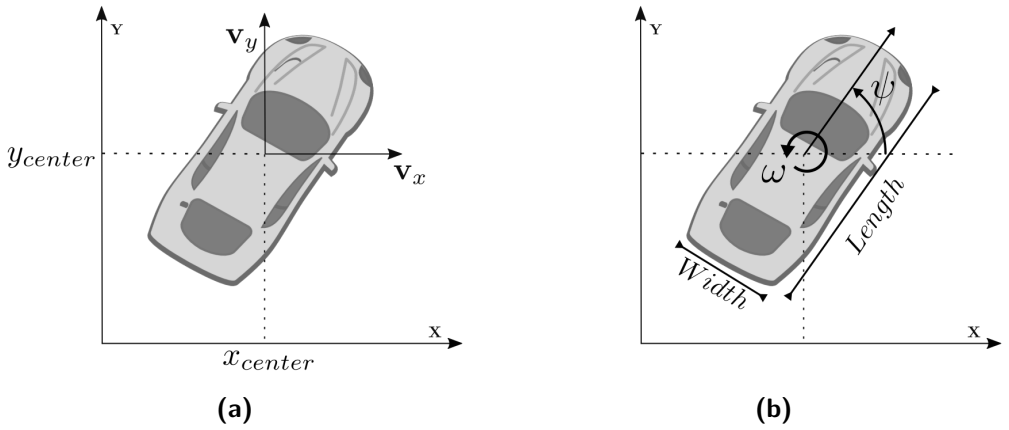


Figure 1-3: The eight states that are estimated for every detected vehicle; (a) the states x_{center},y_{center}, v_x, v_y; (b) the states ψ, ω, Length and Width.

1-2 Categories of Approaches to the DATMO problem

In this section, the different approaches for designing a DATMO system will be divided into three categories, the traditional one, the model based and the grid based one (Figure 1-4). All three approaches have as input sensor data and in case that multiple sensors are used a virtual sensor (blue line in Figure 1-4) is constructed, which contains all measurements in a common frame.

The first method is the traditional approach, in which the sensor data is initially divided into clusters and then associated with already tracked objects from previous time instances. In more advanced systems, the clusters are fitted with geometric shapes whose center is then tracked with a parametric Bayesian filter [11], otherwise the geometric mean of each cluster is tracked [12].

The model based approach fits the sensor data directly onto geometric shape models by utilizing particle filters, which also handle data association [13],[14],[15].

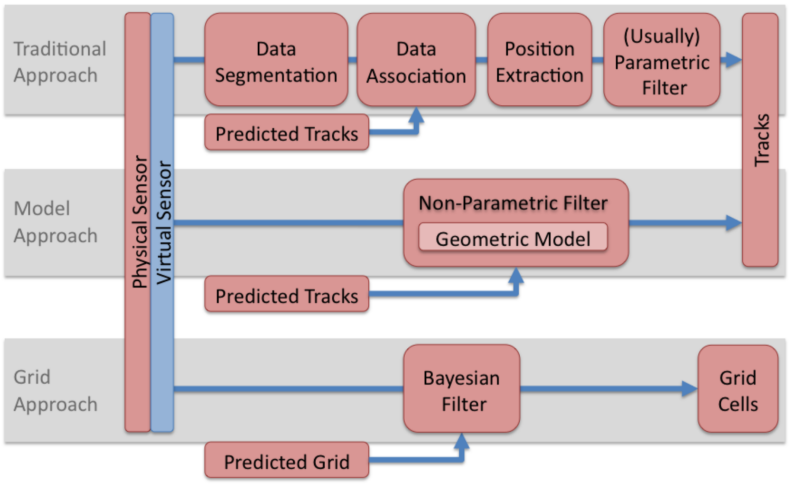


Figure 1-4: The three general classes of approaches to the DATMO problem. Source [2].

Lastly, the grid based approach [16] is based around the construction and use of an occupancy grid, which models the space around the vehicle. The grid cells are then tracked using a Bayesian filter and in some systems, additional object level representations are fitted on top of the occupied grid cells[17].

1-2-1 Traditional Approach

This approach corresponds to the top row in Figure 1-4 and is usually used in combination with LIDAR sensors. It's main characteristic is the data pre-processing that is required before the application of an object tracking filter. The first pre-processing step is to segment the sensor data into clusters that correspond to objects of the real world. This step is followed by the data association step, which pairs clusters with previously tracked objects. Those two steps are highly coupled and erroneous output of the clustering step can negatively impact the performance of data association. After data association, either the geometric mean of the cluster or the center of an extracted geometric shape are tracked by a parametric filter.

1-2-2 Model Approach

Model based methods are represented by the middle row of Figure 1-4. These methods are characterized by the use of geometric models in conjunction with particle filters. Model based methods, do not have separate clustering and data association steps, as was the case in the traditional approach, because those steps are handled by the filter itself. Figure 1-5 explains the advantages of model based methods over the traditional ones. Although, at first it seems that the problem is simplified, a new challenge arises. Making the particle filter computations fast enough so they can meet the high update frequency required by autonomous driving applications.

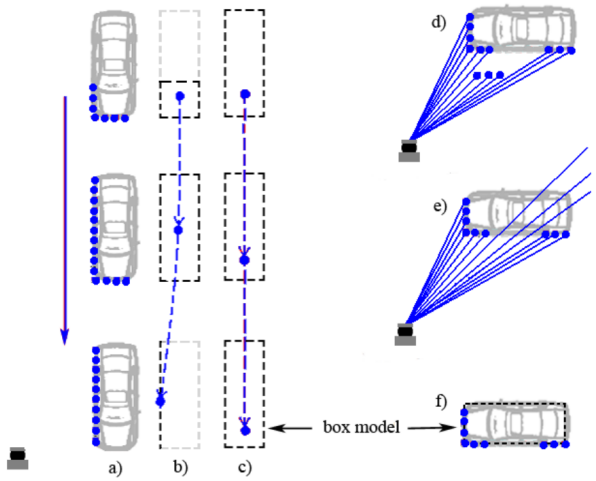


Figure 1-5: Problems in the traditional approach that get mitigated by model based methods. (a) LIDAR measurements of an approaching car. (b) Phantom motion due to geometric mean shift of cluster model. (c) Enhanced motion estimation using box model. (d) (e) Problems due to occlusions and sensor limitations. (f) Mitigation of those problems with box models. Source [3].

1-2-3 Grid Approach

Grid based methods (bottom row in Figure 1-4) use occupancy grids to represent the surrounding area of a vehicle. Occupancy grids [18] partition a planar surface in a grid of cells, that usually are uniform in size. Each of these cells $c = 1, \ldots, n$ is assigned a probability of occupancy $P(O_c)$. $P(O_c) = 0$ represents the case where the cell is surely unoccupied and $P(O_c) = 1$ the case in which the cell is surely occupied. When there is no information about the occupancy status of a cell, it is given the value $P(O_c) = 0.5$. The grid size is defined by two parameters, the number of cells (n) and the size of those cells. Those two values in combination, define both the size of the area around the car that is mapped and also the resolution of this mapping. Theoretically, covering a large area with high resolution is the optimal goal, however this requires a lot of computational resources for calculating and updating the grid.

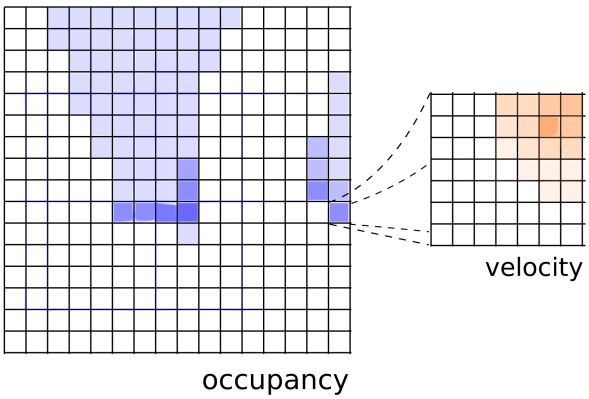
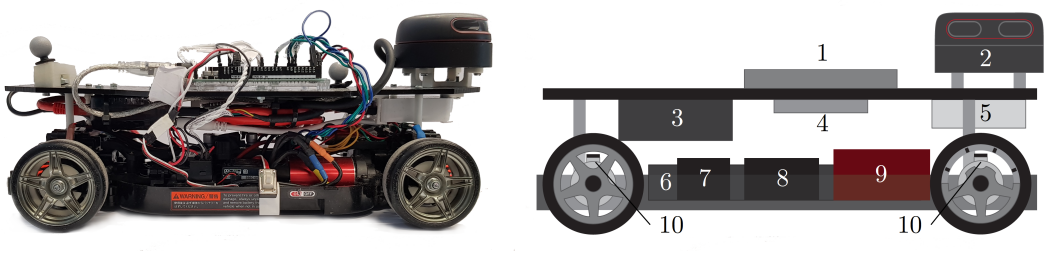


Figure 1-6: Output of the BOF algorithm, the probability of occupancy is represented with shades of blue and the velocity estimation is represented by a histogram assigned to each individual cell. Source [4].

This occupancy grid map can be immediately used for low-level navigation purposes (obstacle avoidance), since it maps the areas that are free of obstacles and the occupied ones. Better estimates of occupancy, alongside velocity information (Figure 1-6), can be inferred by using a Bayesian Occupancy Filter (BOF) [19], which is an adaptation of Bayesian filtering for occupancy grids. In case that, object level representations are required, the Fast Clustering and Tracking Algorithm (FCTA) [20] can be used to track and detect moving objects.

1-3 Experimental Platform

The developed system was implemented and tested on the Delft Scaled Vehicle (DSV), which is shown in Figure 1-7. The DSV is based on the Berkeley Autonomous Race Car (BARC)[1] project, which is a open source project for the construction of scaled testing platforms for autonomous driving.



(a) The Delft Scaled Vehicle (DSV) **(b)** Schematic of the DSV

Figure 1-7: The DSV with it's sensors: 1: Arduino Mega, 2: LIDAR Rplidar-A2, 3: Odroid-xu4 computer, 4: IMU, 5: Router, 6: Battery, 7: Servo, 8: Motor Controller, 9: Motor, 10: Wheel Encoders.

Table 1-1: Overview of sensor output frequencies and actuator input frequencies.

Device	Frequency	Latency
Steering Input	30 Hz	<0.05s
Throttle input	30 Hz	<0.05s
Wheel Encoders	30 Hz	0.05s
IMU	120 Hz	<0.05s
LIDAR	12 Hz	0.15s
MoCap	30 - 240 Hz	<0.05s

In Table 1-1 the sensor frequencies of all the sensors on the DSV are listed. The first four sensors are not used by the developed system, but are used by the localization system, which will be presented in the first Appendix. Since the LIDAR sensor has a frequency of 12 Hz, a goal is set for the developed system to operate always at a faster speed, so that it can incorporate all LIDAR measurements in its predictions.

[1] http://www.barc-project.com/

1-4 Research Question

In recent years a lot of experimental autonomous platforms have been developed for research in autonomous ground vehicles that use a 2D LIDAR sensor for environment perception. These platforms are mainly used to facilitate the testing and development of required systems for autonomous vehicles, such as navigation and path planning, platooning, collision avoidance, etc. For those systems to be tested on the experimental platforms, there is a need for a robust and lightweight DATMO system that will provide a description of the dynamic environment the tested system should react to. To address this need, the main objective of my thesis will be to achieve the following:

Main Research Objective

Design, implement and evaluate a detection and tracking of moving objects system, tailored for experimental autonomous platforms equipped with a 2D LIDAR sensor; the developed system will take raw, 2D LIDAR measurements as input and provide the kinematic state and dimensions of the detected objects as end output in a robust, causal, and real time manner.

This general objective can be broken down to several sub-objectives, whose fulfillment will reflect the contributions of my thesis:

1. Identify the requirements of object detection and tracking in terms of target motion uncertainties and 2D LIDAR sensor limitations.
2. Select and use algorithms which fulfill the identified requirements to accurately achieve both the tasks of detection and tracking.
3. Design a modular system architecture, so the proposed system can be easily expanded in the future to incorporate new algorithms and potentially measurements from other sensors.
4. Perform real world experiments using the Delft Scaled Vehicle to evaluate the performance of the proposed system.
5. Design and develop a simulation platform in which the capabilities and robustness of the proposed system could be accurately tested in various scenarios.
6. Build the proposed system on top of the Robot Operating System (ROS) framework, which facilitates software and hardware exchangeability.
7. Keep the computational requirements at a minimum, since experimental platforms are generally equipped with single board computers that are required to run many applications concurrently and in real time.

1-5 Report Outline

This thesis report is structured in the following way: in this chapter, an introductory overview of the general problem was given, as well as an outline of the approach to its solution.

In Chapter 2 the detection part of the proposed system is presented.

In Chapter 3 the tracking part of the system in presented.

In Chapter 4 the experimental methods used for evaluating the system are described and the results are presented and analyzed.

In Chapter 5 the general results of the thesis are summarized and conclusions are drawn. In addition, proposals for future work in areas that are not sufficiently covered by this thesis are given.

In Chapter A the development of a localization system for the DSV is presented.

In Chapter B a part of this MSc thesis is presented in a IEEE double column format paper.

Chapter 2

Detection of Moving Objects

In this chapter, the detection stage of the developed DATMO system will be presented and analyzed. The main goal of the detection stage is to differentiate moving objects from the measurements provided by the LIght Detection And Ranging (LIDAR) sensor. In Figure 2-1 a flowchart of the general steps of the implemented detection stage is given, which also corresponds to the organization of this chapter.

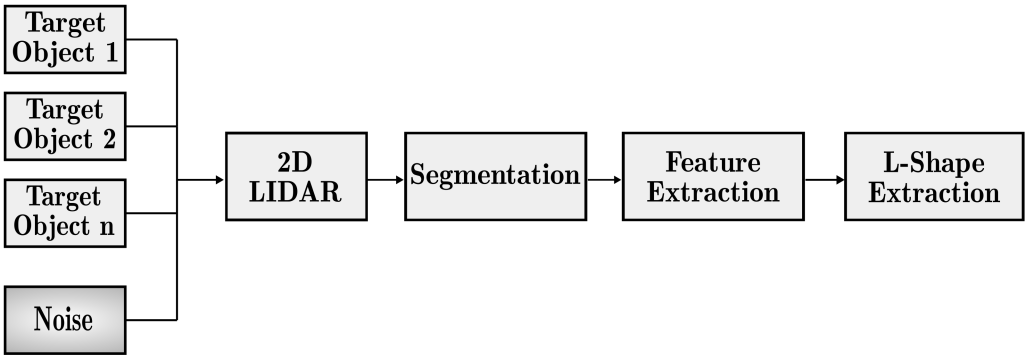


Figure 2-1: Flowchart of the vehicle detection stage of the developed system.

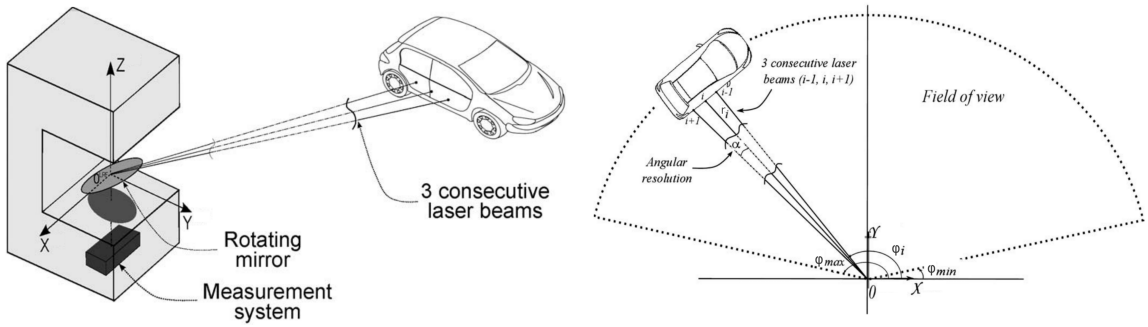
The first step of the detection stage is a segmentation algorithm, which extracts clusters of LIDAR points from the raw LIDAR data.

These clusters are then passed to a feature extraction algorithm, which extracts geometric shapes from the clusters. Common extracted geometric shapes are lines/rectangles for vehicles, circles for pedestrians and ellipses for bicycles/bikes [21]. Since the focus of the developed system is vehicle tracking, rectangles are extracted from the clusters.

Lastly, L-shapes are extracted from the closest corner of every rectangle and those are passed to the tracking stage of the system, which will be presented in the next chapter.

2-1 Data Acquisition by a 2D LIDAR sensor

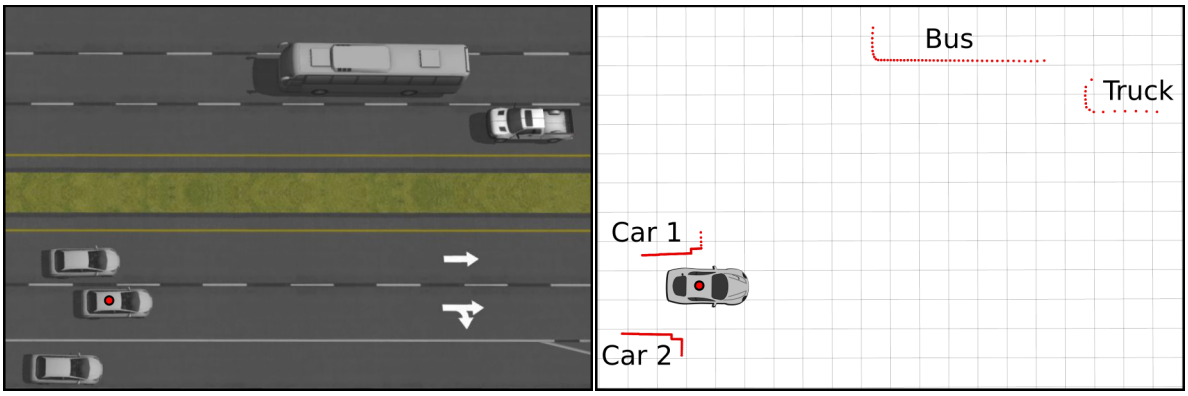
In their most basic form, LIDAR sensors calculate distance to objects by emitting a laser beam, capturing its reflection and calculating the distance to the object that caused the reflection by measuring the time of flight. Typically, it is desirable to cover an area as wide as possible with a single sensor and therefore the laser emitter is placed behind a rotating mirror or on top of a rotating base (Figure 2-2a).



(a) Workings of a LIDAR sensor. **(b)** Reference variables of a LIDAR sensor.

Figure 2-2: Visualization of the principles of operation of a LIDAR sensor. Adapted from [5].

In Figure 2-2b there is a graphical representation of the most important features of a LIDAR sensor. Given that the emitted laser beam moves in a circular motion, the produced measurements are naturally derived in polar coordinates (r, ϕ). Their field of view $(\phi_{\min}, \phi_{\max})$, is usually bigger than 180°, and it is common for LIDARs to cover 360°. Typical values for their angular resolution (α) are between $0.25° - 1°$, while their measurement frequency is usually between 10 and 20 Hz for a complete scan [2]. Their maximum range $(r_{\max})$, ranges from about 8 meters for small sensors, to 80 meters for top of the line sensors.



(a) Image of the simulation environment. **(b)** LIDAR raw measurements.

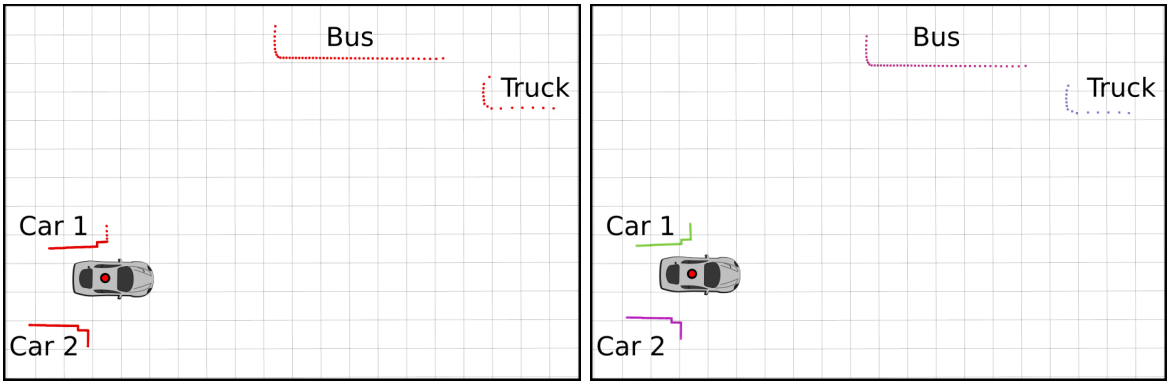
Figure 2-3: Example of LIDAR data acquisition. On (a) there is an image of the developed simulation environment, in which a LIDAR sensor is mounted on the middle of the three cars at the bottom left. On (b) there is visual depiction of the LIDAR data acquired by the sensor at the same time instance.

The measurements of a LIDAR sensor can be better understood by examining Figure 2-3, in which there is both a screenshot of the simulated environment and the resulting LIDAR measurements from the same time instance. The ego vehicle and the one that the LIDAR sensor is mounted on top, is the middle one of the three cars at the left (red dot). In Figure 2-3b the ego vehicle is represented by the car with the red dot and the LIDAR measurements are depicted as red points. This specific time instance will be used throughout this chapter to explain and visualize the operation of the detection stage.

The Delft Scaled Vehicle (DSV) is equipped with a LIDAR sensor (RPLIDAR-A2) that has a field of view of 360°, angular resolution of 1° and range of 8 meters. The virtual sensor used in the simulation environment has similar characteristics, with the only difference being the maximum range. Since the vehicles in the simulation environment are bigger than the scaled vehicles, the 8 meters range of the real sensor is quite limiting and therefore the range of the simulated sensor was chosen equal to 50 meters.

2-2 Segmentation

The segmentation process (Figure 2-1) is responsible for separating the raw LIDAR measurements (Figure 2-4a) in groups that correspond to moving objects of the real world that need to be tracked. In Figure 2-4b the LIDAR points after the application of this algorithm are visualized and are drawn with a different color for every segmented cluster. We can observe that the system created four different clusters which correspond accurately to the four surrounding vehicles of the simulation.



(a) LIDAR raw measurements. **(b)** Output of the Adaptive Breakpoint Detector.

Figure 2-4: Output of the Adaptive Breakpoint Detector algorithm.

In this section there will be two algorithms that accomplish this task presented, however it should be kept in mind that there are multiple algorithms for data clustering (k-means, k-d tree). The methods presented here and implemented in the proposed system, are especially designed for 2D LIDAR sensors and therefore have some advantages over generic algorithms. One such advantage originates from the knowledge that LIDAR measurements are captured in a sequential manner, this information is exploited in the operation of the algorithm and points are checked if they belong to the same cluster sequentially. In this way, the computational requirements are considerably reduced.

2-2-1 Breakpoint Detector Algorithm

The Breakpoint Detector Algorithm [22] clusters the 2D LIDAR point cloud of n points, $X \in R^{n\times 2}$, based on the euclidean distance between consecutive points.

Consecutive points p_n and p_{n-1} are clustered together if their euclidean distance is lower than a predefined threshold distance $D_{\max}$. In case that the breakpoint Equation 2-1 holds, a new cluster is started whose first point is p_n.

$$\|p_n - p_{n-1}\| > D_{\max} \tag{2-1}$$

The operation principle of this algorithm is visualized in Figure 2-5. The points in the yellow eclipse are the first cluster and the points in the green eclipse are the points of the second cluster. This cluster separation is based on the detection of a breakpoint between points p_n and p_{n-1}.

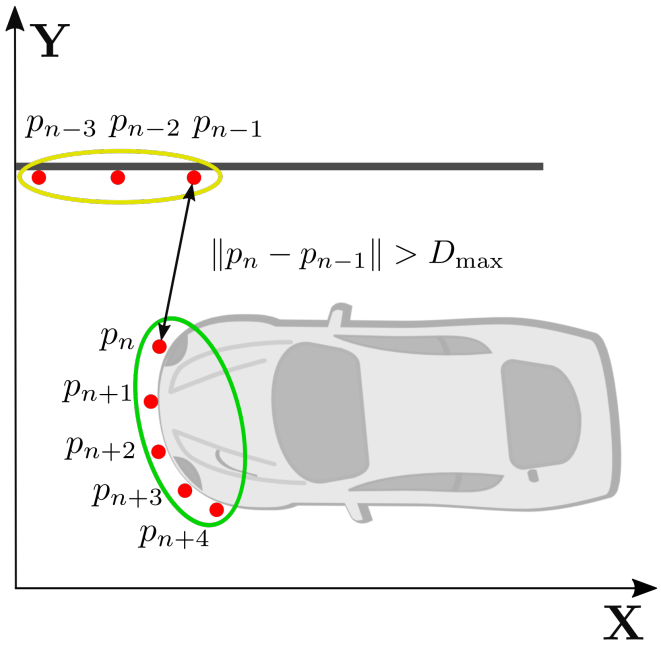


Figure 2-5: Visualization of the Breakpoint Detector Algorithm.

The fixed threshold distance $D_{\max}$ that this algorithm operates on, does not account for the fact that the point cloud that the LIDAR sensor produces becomes sparser as the distance from the sensor increases. This can be observed in Figure 2-4a, where the LIDAR measurements from the truck, which is the furthest away from the sensor are considerably sparser than the point clouds from the two neighboring vehicles. This has as a result, that objects that are far away from the sensor can have measurements that are distant enough that are separated in different clusters. A way to overcome this limitation is by adapting the threshold distance ($D_{\max}$) depending on the range of the measurement.

2-2-2 Adaptive Breakpoint Detector Algorithm

A way to adapt the threshold breakpoint distance $D_{\max}$, according to the range distance r_n of the examined point, was first presented in [23] and will be explained below.

First, a line is drawn through the range point p_{n-1}, which represents the worst case for an incidence angle of a real world object that can be detected by the sensor. This line creates an angle λ with respect to the scanning angle ϕ_{n-1}. The maximum range distance r_n^h, for p_{n-1}, is calculated in the following way:

$$r_{n-1} \cdot \sin(\lambda) = r_n^h \cdot \sin(\lambda - \Delta\phi) \tag{2-2}$$

By reworking the equation above, $\left\|p_n^h - p_{n-1}\right\|$ is calculated, which can be used as a threshold distance ($D_{\max}$) in the breakpoint detection algorithm.

$$\left\|p_n^h - p_{n-1}\right\| = r_{n-1} \cdot \frac{\sin(\Delta\phi)}{\sin(\lambda - \Delta\phi)} \tag{2-3}$$

However, because the sensor noise is not taken into account, problems can arise when the range distance is small. Therefore, the sensor error variance σ_r is added to the max distance .

$$D_{\max} = \left\|p_n^h - p_{n-1}\right\| + \sigma_r \tag{2-4}$$

In Figure 2-6, we can see the threshold circle, which gets drawn around p_{n-1}, with a radius that equals to $D_{\max}$. In case that, the next point p_n is within the circle the two points are clustered together, otherwise a breakpoint is detected and a new cluster gets initialized, in exactly the same way as in the Adaptive Breakpoint Detector algorithm.

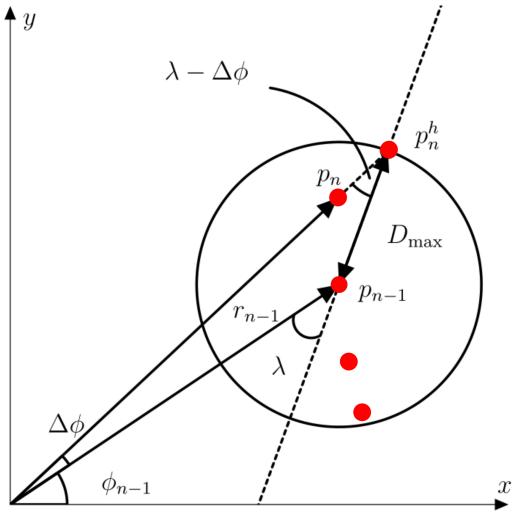


Figure 2-6: Visualization of the Adaptive Breakpoint Detector Algorithm. Adapted from [6].

Algorithm 1 is the pseudocode implementation of the Adaptive Breakpoint Detector. The general structure of the algorithm is the same for the Breakpoint Detector, with the only difference being the calculation of the $D_{\max}$ value, which in that case is a predefined constant value.

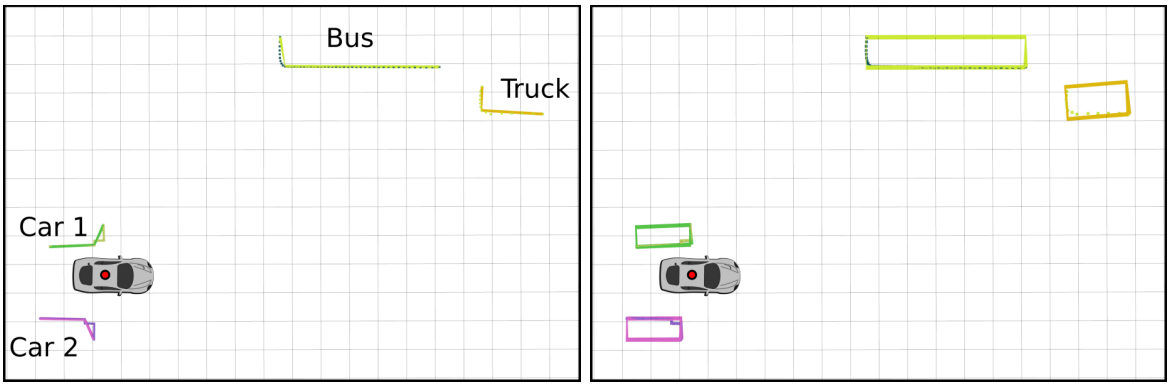
Algorithm 1 Adaptive Breakpoint Detector Segmentation Algorithm

Input: range data points $X \in R^{n\times 2}$
Output: set of point clusters S

1: $n = 1$
2: **while** n < length(X) **do**
3: $D_{\max} \leftarrow r_{n-1} \cdot \frac{\sin(\Delta\phi)}{\sin(\lambda-\Delta\phi)} + \sigma_r$
4: **if** $\|p_n - p_{n-1}\| < D_{\max}$ **then**
5: C push p_n
6: $i = i + 1$
7: **else**
8: S push C
9: $i = i + 1$
10: **end if**
11: **end while**

2-3 Feature Extraction

The purpose of the Feature Extraction process (Figure 2-1) is to extract geometric shapes from the clustered points. Since the main goal of the developed system is vehicle tracking the chosen geometric shapes to be extracted from the clusters were lines (Figures 2-7a) and rectangles (Figures 2-7b).



(a) Line extraction. **(b)** Rectangle extraction.

Figure 2-7: Output of the two implemented Feature Extraction algorithms.

Initially, line extraction was chosen since it was simpler to implement and computationally less intensive. Even though, it produces good results when the measurements follow precisely the shape of the detected vehicle, it fails to correctly estimate the vehicle's shape when the measurements are less precise. This can be observed in Figure 2-7a, where the line extraction algorithm draws the lines representing the sides of the vehicles in a non optimal way. For the case of the pickup truck at the right of the image, both algorithms fail to extract correctly its orientation since the measurements are sparse at that distance.

2-3-1 Line Extraction

The algorithm that was chosen and implemented for line extraction is the Iterative End-Point Fit algorithm, otherwise known as the Ramer-Douglas-Peucker algorithm. The main function of this algorithm is to simplify a curve composed of multiple line segments to a similar one with fewer segments. It should be noted, that the simplified line consists of a subset of the points that defined the original curve and no new points are created.

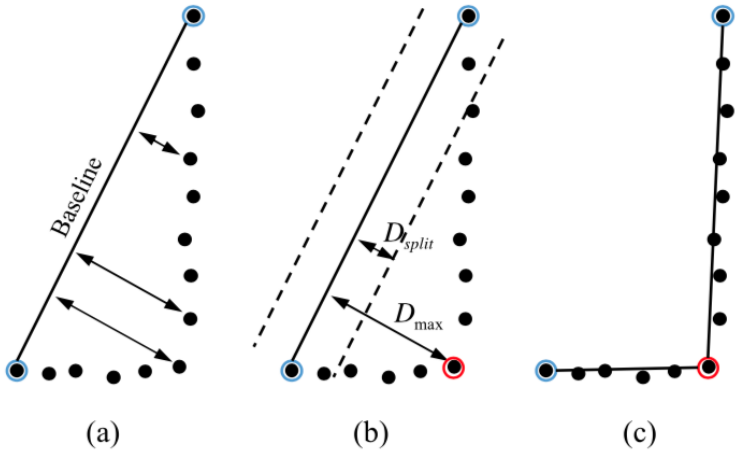


Figure 2-8: Visualization of the operation of the Iterative End-Point Fit Algorithm, on LIDAR points reflected from a vehicle. Source [7]

Algorithm

The input to the algorithm is the ordered set of LIDAR points of a cluster $X \in R^{n\times 2}$ and the predefined constant D_{split}, which defines the accepted dissimilarity between the original and resulting lines.

At the beginning, the algorithm marks the first and last points of the initial point set to be kept (blue circles in Figure 2-8). The next step of the algorithm, is to find the point that is farthest from the line segment ($D_{\max}$) with the first and last points as end points (Figure 2-8a). In the case that, $D_{\max} > D_{split}$, then the algorithm recursively calls itself with the first point and the farthest point and then with the farthest point and the last point (Figure 2-8b). In the other case that, $D_{\max} \leqslant D_{split}$, then any points not currently marked to be kept can be discarded without the simplified curve being worse than D_{split}. When the recursion is completed a new output curve is created consisting of all and only those points that have been marked as kept. In Figure 2-8c the kept points are the ones marked in blue and red circles.

Although this algorithm is computationally more efficient than the rectangle fitting one that was used in the final version of the system, it has some disadvantages that significantly reduce the robustness of the system as a whole. The most evident problem being the algorithms dependence on the correct tuning of the D_{split} parameter, since a wrong value for this parameter can lead to problems similar to the ones depicted in Figure 2-7a. In addition, the number of line segments to be extracted can not be predetermined and therefore there are cases in which, more than two line segments are extracted from a given cluster.

2-3-2 Rectangle Extraction

For each segmented cluster of LIDAR points, a rectangle needs to be found that best fits around them. The algorithm that was chosen and implemented for rectangle fitting is the Search-Based Rectangle Fitting algorithm [24], whose basic idea is to iterate through all the possible directions and at each iteration find a rectangle that contains all the LIDAR scan points. Afterwards, a performance score is calculated for each rectangle and the one with the highest score is chosen.

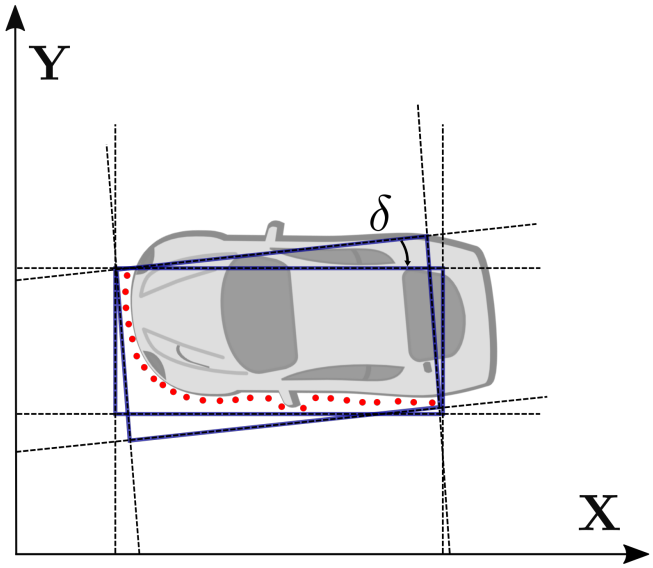


Figure 2-9: Principle of operation of the Search-Based Rectangle Fitting Algorithm.

In Figure 2-9 the search-based operation of the algorithm is visualized. It can be seen that the algorithm fits, as an example, two rectangles on the LIDAR range points that differ between them by an angle δ. Although, both rectangles contain all the measurement points, one rectangle is better than the other at representing the vehicle that the points originated from. The rectangle selection is done at a later stage, through the calculation of some criterion, which represents the prior belief about the characteristics of the best fitting rectangle.

The implementation of the Search-Based Rectangle Fitting algorithm that was used in the system is presented in pseudocode as Algorithm 2. The input of the algorithm is the n points of the examined cluster, $X \in R^{n\times 2}$. The output of the algorithm are the line representations of the four edges of the fitted rectangle. The search space for θ ranges from 0° to 90°, because the two sides of a rectangle are orthogonal, and therefore only one edge needs to be calculated, since the other is $\theta + \pi/2$. The criterion that is used in Algorithm 2 line 7, is the performance score that represents how closely the rectangle fits over the given LIDAR points. This criterion can be defined in three different ways, each one with distinct advantages and disadvantages.

Algorithm 2 Search-Based Rectangle Fitting

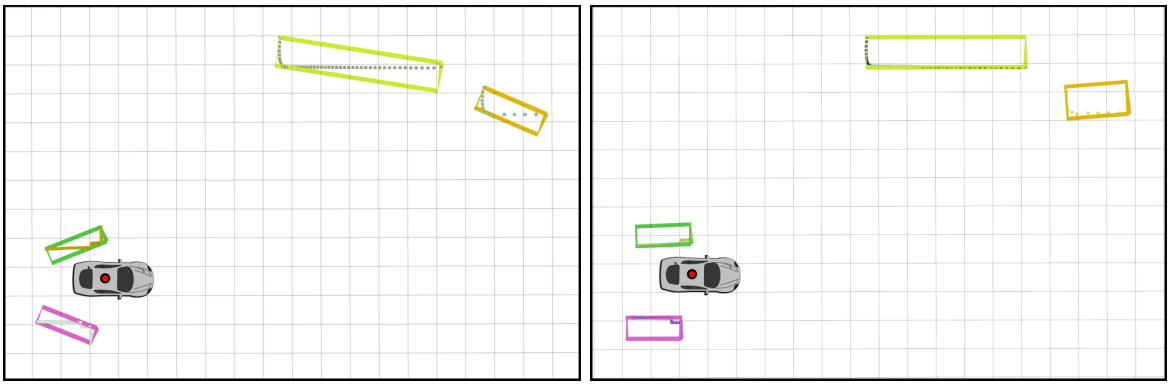
Input: range data points $X \in R^{n\times 2}$
Output: rectangle edges $\{a_i x + b_i x = c_i | i = 1, 2, 3, 4\}$

$Q \leftarrow \emptyset$
for $\theta = 0$ **to** $\pi/2 - \delta$ **step** δ **do**
  $\hat{e}_1 \leftarrow (\cos\theta, \sin\theta)$ ▷ rectangle edge direction vector
  $\hat{e}_2 \leftarrow (-\sin\theta, \cos\theta)$
  $C_1 \leftarrow X \cdot \hat{e}_1^T$ ▷ projection on to the edge
  $C_2 \leftarrow X \cdot \hat{e}_2^T$
  $q \leftarrow \text{CalculateCriterion}(C_1, C_2)$
  insert q into Q with key (θ)
end for
select key (θ^*) from Q with (θ)
$C_1^* \leftarrow X \cdot (\cos\theta^*, \sin\theta^*)^T, C_2^* \leftarrow X \cdot (-\sin\theta^*, \cos\theta^*)^T$
$a_1 \leftarrow \cos\theta^*, \quad b_1 \leftarrow \sin\theta^*, c_1 \leftarrow \min\{C_1^*\}$
$a_2 \leftarrow -\sin\theta^*, \quad b_2 \leftarrow \cos\theta^*, c_2 \leftarrow \min\{C_2^*\}$
$a_3 \leftarrow \cos\theta^*, \quad b_3 \leftarrow \sin\theta^*, c_3 \leftarrow \max\{C_1^*\}$
$a_4 \leftarrow -\sin\theta^*, \quad b_4 \leftarrow \cos\theta^*, c_4 \leftarrow \max\{C_2^*\}$

Selection Criteria

In [24], they are presenting three different selection criteria, namely: rectangle area minimization (Algorithm. 3), point-to-edges closeness maximization (Algorithm. 4), and points-to-edges squared error minimization. And each of one of the three can be used to substitute the CalculateCriterion(C_1, C_2) function.

In the developed system only the first two criteria were implemented, since the second one provided satisfactory results and according to the original authors the third criterion, more than doubles the required computational time from 4.00 (ms) to 8.37 (ms), while only decreasing the Absolute Error by 0.92°. Because the application at hand is real time and computational efficiency is of high importance the third algorithm was not implemented.



(a) Area minimization criterion. **(b)** Point-to-edges closeness maximization criterion.

Figure 2-10: Visualization of rectangle fitting with the two implemented selection criteria.

The area minimization criterion (Algorithm 3), gives the highest score to the rectangle that covers the smallest area. The output of the rectangle fitting algorithm, when this criterion is selected can be seen in Figure 2-10a. It is evident that the chosen rectangles do not correspond accurately to the vehicles orientations and therefore the area criterion was not used in the final system.

Algorithm 3 Area Criterion

function CALCULATEAREA(C_1, C_2)
  $c_1^{\max} \leftarrow \max\{C_1\}, \quad c_1^{\min} \leftarrow \min\{C_1\}$
  $c_2^{\max} \leftarrow \max\{C_2\}, \quad c_2^{\min} \leftarrow \min\{C_2\}$
  $\alpha \leftarrow -\left(c_1^{\max} - c_1^{\min}\right) \cdot \left(c_2^{\max} - c_2^{\min}\right)$
  return α
end function

The points-to-edges closeness maximization criterion, shown in Algorithm 4, gives the highest score to the rectangle that has its edges closer to the LIDAR points. The output of the rectangle fitting algorithm when this criterion is selected, can be seen in Figure 2-10b and it is evident that the selected rectangles are more accurate than the ones selected by the area criterion.

Algorithm 4 Closeness Criterion

Parameter: d_0

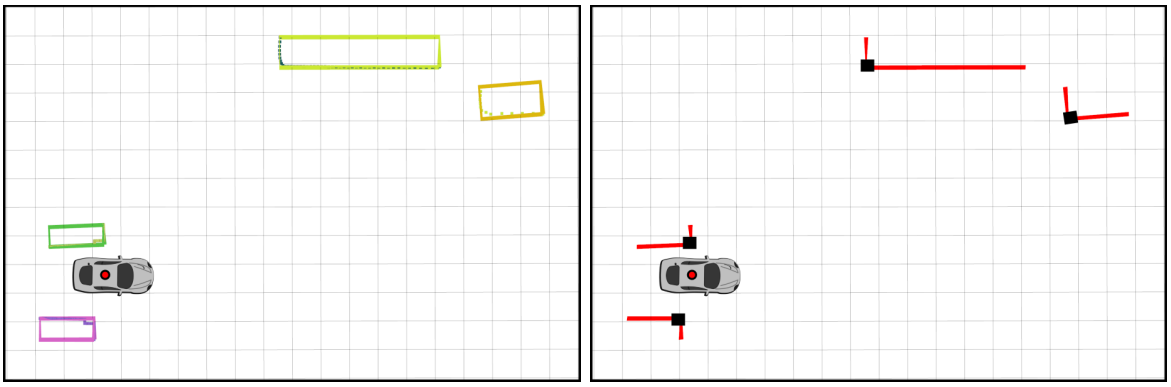
function CALCULATECLOSENESS(C_1, C_2)
  $c_1^{\max} \leftarrow \max\{C_1\}, \quad c_1^{\min} \leftarrow \min\{C_1\}$
  $c_2^{\max} \leftarrow \max\{C_2\}, \quad c_2^{\min} \leftarrow \min\{C_2\}$
  $D_1 \leftarrow \arg\min_{v \in \{c_1^{\max} - C_1, C_1 - c_1^{min}\}} ||v||_{l_2}$
  $D_2 \leftarrow \arg\min_{v \in \{c_2^{\max} - C_2, C_2 - c_2^{min}\}} ||v||_{l_2}$
  $\beta \leftarrow 0$
  for $i = 1$ **to** $\pi/2 - \delta$ **step** 1 **do**
    $d \leftarrow \max\left\{\min\left\{D_{1(i)}, D_{2(i)}\right\}, d_0\right\}$
    $\beta \leftarrow \beta + 1/d$
  end for
  return β
end function

On the projected 2-D plane, c_1^{max} and c_2^{min} represent the boundaries for all the points on axis $\hat{e}_1$. The vectors $c_1^{max} - C_1$ and $C_1 - c_1^{min}$ contain the distances of all the points to the two boundaries. From the two boundaries, the one that is closer to the range points is chosen and denoted as distance vector 1 (D_1). Distance vector D_2, is chosen the same way for projection axis $\hat{e}_2$. Finally, the closeness score is calculated as, $\sum_{i=1,\ldots,m} 1/d_i$, where d_i is the i-th point's distance to the closest edge. This way the score is increased both by reducing the distance and increasing the number of points.

In addition, a minimum distance threshold is introduced, to counteract possible divisions by zero, in case that the LIDAR points are close to the boundary, and also to reduce the score influence of points that are very close to the edges.

2-4 L-shape Extraction

After every cluster of LIDAR points is fitted with a bounding rectangle, an L-shape feature is extracted from every rectangle (Figure 2-11), mainly for two reasons. First, the information about the closest corner of surrounding vehicle is highly valuable for collision avoidance systems and secondly by extracting L-shapes of the closest sides of surrounding vehicles, their appearance changes can be mitigated in later stages of the developed system.



(a) Rectangle Fitting. **(b)** L-shapes (red) - Closest corners (black).

Figure 2-11: L-shape extraction.

L-shapes are extracted from the bounding rectangles by choosing as L-shape corner point, the corner point that is closest to the sensor. The two bounding box edges that connect to the corner point are named L_1 and L_2, by following a clockwise assignment convention, shown in Figure 2-12. The orientation angle (θ) of the L-shape is always defined as the orientation of L_1.

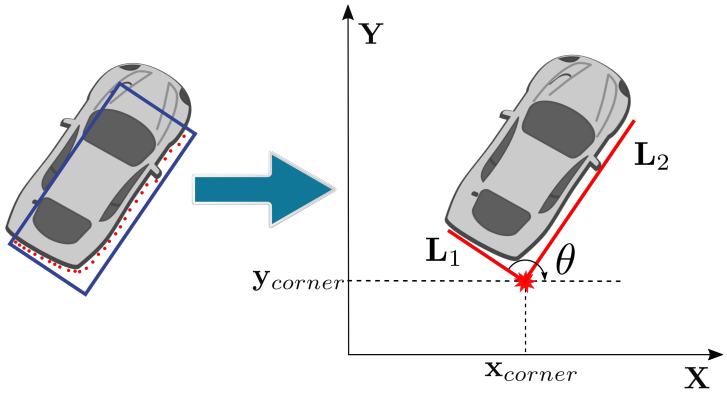


Figure 2-12: Conversion of a bounding rectangle to an L-shape feature.

Summarizing the above, the L-shape feature contains five values that are extracted from the bounding box and which will be used in later stages for vehicle tracking. The position of the corner point (x_{corner}, y_{corner}), the lengths L_1, L_2 and the orientation angle θ.

Chapter 3

Tracking of Moving Vehicles

In this chapter, the tracking stage of the developed DATMO system will be presented and analyzed. The objective of the vehicle tracking stage is to estimate as accurately as possible the position, speed and dimensions of all detected vehicles. A flowchart of this stage is given in Figure 3-1 and a brief explanation of its operation will be given below.

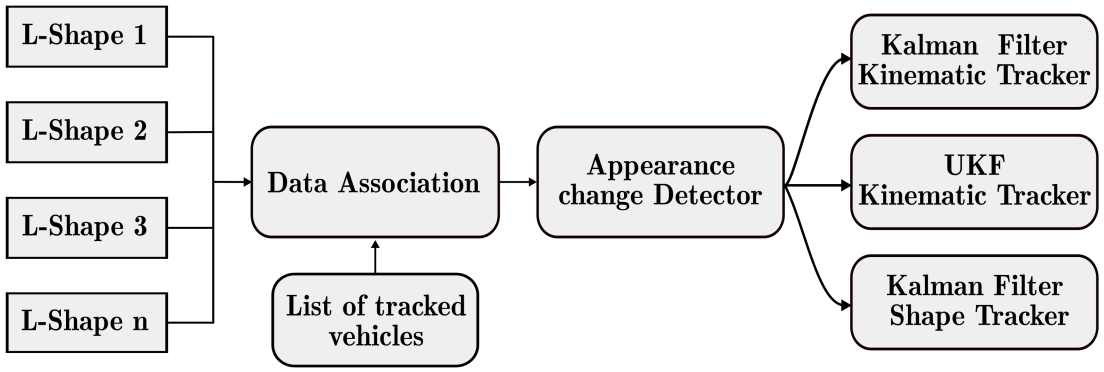


Figure 3-1: Flowchart of the vehicle tracking stage of the developed system.

At the left side of the flowchart is the input into the tracking stage, which are the extracted by the detection subsystem L-shapes. The first component of the tracking stage is the Data Association, in which the newly received L-shapes are associated with tracked vehicles from previous time instances. After the L-shapes are associated with vehicles, it is investigated if the observed corner of the vehicles changed and with it the direction of the associated L-shape. If this is true, the three trackers are updated to reflect the change. Lastly, the position of the L-shape is used to update the two L-shape kinematic trackers and its dimensions and orientation are used for updating the shape tracker. The Kalman Filter (KF) kinematic tracker uses a linear vehicle motion model and its aimed at systems with low computational capabilities, while the Unscented Kalman Filter (UKF) tracker uses a nonlinear motion model and it is geared towards system with higher capabilities.

3-1 Data Association and Track Management

Data association is the process of associating detection results with already tracked objects by working out which observations were generated by which targets.

The data association problem in multiple vehicle tracking is complicated because of the inherent uncertainty of sensor measurements and the fact that the number of observations does not necessarily correspond to the number of surrounding objects. Additionally, the true number of objects is difficult to estimate since one object might be temporarily occluded or unobserved and because objects can enter or go out of range of the vehicle's sensors.

Track management for multiple object tracking consists of deducing the number of true objects and identifying if each observation corresponds to an already known object being tracked, to a false measurement or to a new object in the scene that needs to be tracked. The complexity of track management and data association grows in correspondence with the number of objects that need to be tracked.

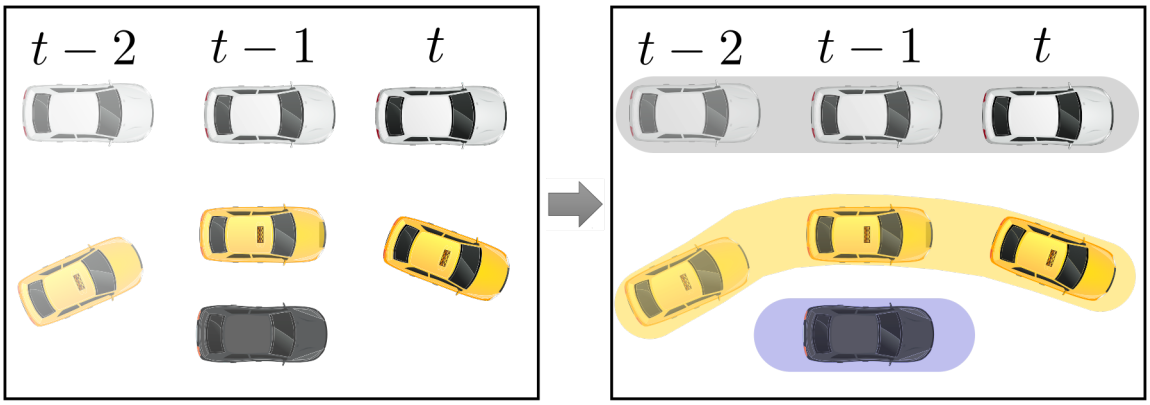


Figure 3-2: Visualization of the data association process. Left: A set of observations, where every circle is a detected object and the numbers are the timesteps. Right: Output of a data association module composed of three object tracks (gray, yellow, blue).

Figure 3-2 shows an example of successful data association and track management given observations over three time steps ($t-2$, $t-1$, t). At $t-2$ two tracks are created (the gray and yellow ones) and at $t-1$, t they are correctly associated with the respective vehicle observations. At $t-1$, the blue object track is created to track the new vehicle that entered the scene, but it is deleted at the next time step, since there is no new observation.

3-1-1 Data Association

There are two main classes of data association filters: deterministic filters and probabilistic filters. The most commonly used deterministic filters are the Nearest Neighbor (NN) and the Global Nearest Neighbor (GNN). Probabilistic filters are used in case that there are ambiguities and clutter in the observations, and deterministic assignments of measurements to targets are hard to make. Two of the most common probabilistic filters are the Multiple Hypothesis Tracking (MHT) filter and the Joint Probabilistic Data Association Filter (JPDAF).

Nearest Neighbor - Global Nearest Neighbor

The simplest method of data association is the Nearest Neighbor (NN), which associates objects with tracks based on eucliden distance. This method can only be applied successfully when the sensor update rate is sufficiently higher than the velocities of the moving objects and as a consequence objects remain in the same area between consecutive measurements.
A more sophisticated method and the one used in the proposed system is the Global Nearest Neighbor (GNN), which additionally ensures that each piece of data is assigned to at most one object.

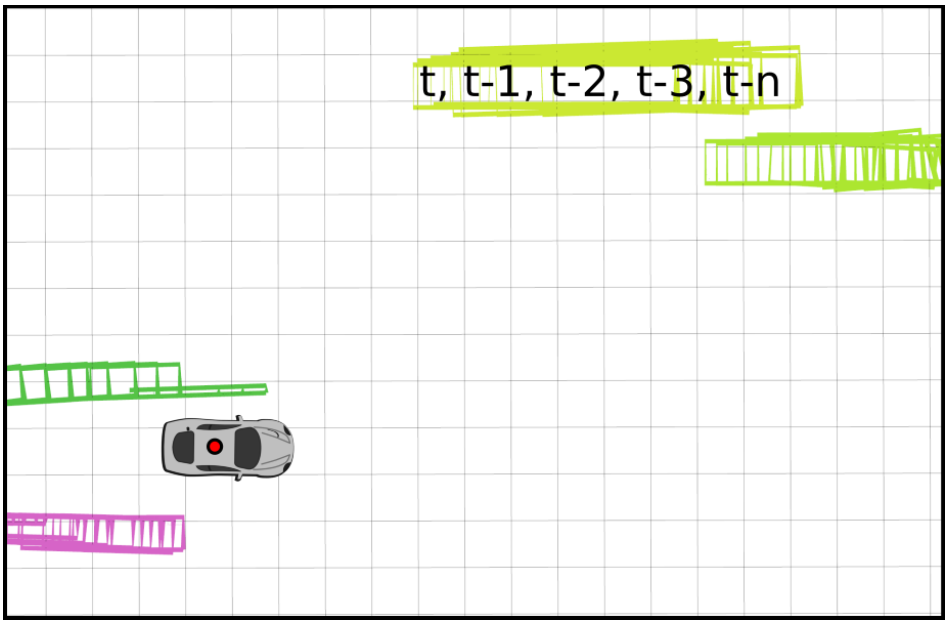


Figure 3-3: Data association in the developed system.

Fig. 3-3 visualizes several consecutive time instances ($t,\ t-1, \ldots,\ t-n$) from the simulation and the correct association of new measurements to already tracked vehicles can be observed. The operation of data association and track management is visualized through assigning a unique color to every object at its creation. Therefore, in case that there was an error, not all four object would have retained the same color throughout the time window, but there would be a color change at the time of the association error.

Multiple Hypothesis Tracking (MHT)

Multiple Hypothesis Tracking (MHT) [25] maintains multiple association hypotheses, where each hypothesis corresponds to a specific probable assignment of detected objects with tracks. For each of the hypotheses, a hypothesis score is computed by summing the track existence scores of all the targets within it. The probability of each hypothesis can then be computed from it's hypothesis score. In conflict situations, instead of taking an immediate decision (NN, GNN) or combining hypotheses (JPDAF), hypotheses are propagated into the future in anticipation that the association ambiguities will be resolved [16].

Although, the MHT framework can handle situations where the measurements arise from a varying number of targets or from background clutter, in practical applications, several issues arise. The most serious one is the combinatorial increase in the number of generated tracks and hypotheses, which results in a high computation load [2].

Joint Probabilistic Data Association Filter (JPDAF)

The Joint Probabilistic Data Association Filter (JPDAF) was originally proposed by Fortmann et al. in 1983 [26]. Unlike NN and GNN, JPDAF does not make a hard assignment of measurements to targets. Instead, it makes a soft assignment by considering the probability of each measurement being assigned to each target. The JPDAF method permits to assign several detected objects to one track by a weighted probabilistic sum, but it differs from MHT, in the regard that a single association hypothesis is maintained.

Unlike the MHT algorithm, JPDAF does not suffer from the combinatorial increase of generated tracks and hypotheses.

3-1-2 Track Management

Tracks is the name given to objects that are tracked by a Detection and Tracking of Moving Objects (DATMO) system and track management is the process of managing the list of tracks. The main goal of track management is to reduce the amount of tracked objects, both for reducing the amount of computations performed at each timestep but also for preventing false data associations. Generally, there are no dominant designs for track management and the implementation of track management differs between applications, because it depends on the rest of the components of the DATMO system, especially data association.

In this thesis, the track management that is used is relatively simple in its design and function, mainly originating from the choice of GNN as the data association method. The implemented track management system operates in the following way. After every measurement update and clustering step, the clusters not associated with any already tracked object are used to initiate new tracks. The tracks that are associated with newly detected clusters are unaffected, while the not associated tracks are immediately deleted. Once the track management phase is completed, all the newly created and associated tracks are propagated to the next time step.

3-2 State Estimation

In each time step, for every tracked vehicle, we want to estimate its longitudinal and lateral position, heading angle, velocity and dimensions (width and length). For estimating those values from the available measurements, the use of the Kalman Filter (KF), Extended Kalman Filter (EKF) and Unscented Kalman Filter (UKF) were considered. The Kalman Filter is the optimal estimator for linear systems, while the EKF and the UKF are extensions of the Kalman filter so that nonlinear functions can be used as the dynamic and measurement functions.

Although the Kalman Filter and its extensions can be written in a single step, they are usually divided in two distinct steps: Prediction and Correction. The prediction step uses the state estimate from the previous timestep to produce an estimate of the state at the current timestep. This predicted state estimate is also known as the *a priori* state estimate because, although it is an estimate of the state at the current timestep, it does not yet include measurements from the current timestep. In the correction step, the current *a priori* prediction is combined with current measurements to calculate the *a posteriori* state estimate.

Typically, the two steps alternate, with the prediction step advancing the state until the next measurement becomes available, and the update step incorporates it. However, this is not necessary; if the measurement is unavailable at some time step, the update may be skipped and multiple prediction steps can be performed in a row. In a DATMO system this can occur if a tracked vehicle is unobserved due to occlusion, or getting out of range of the vehicles sensors. Likewise, if multiple independent observations are available at the same time, numerous update steps can be performed before the next prediction step (typically with different measurement models). In a DATMO system this is useful in cases that measurements from multiple sensors need to be fused.

3-2-1 Kalman Filter

In the standard Kalman filter formulation, both the dynamics and measurement models are represented by linear functions with added Gaussian noise. Under these assumptions, the tracking procedure can be modeled via the following equations:

$$\begin{aligned} x_k &= A_k x_{k-1} + w_k \\ z_k &= H_k x_k + v_k \end{aligned} \tag{3-1}$$

where A_k is the dynamics model, H_k is the measurement model, w_k is the zero mean Gaussian distributed process noise and v_k is the zero mean Gaussian distributed measurement noise. Thus, the Kalman filter produces a belief about the true state, which is fully described by a vector of mean values and a covariance matrix.

Prediction Step

Given a state vector estimate $\hat{x}_{k-1}$, at previous time step $k-1$, the *a priori* state estimate $\hat{x}_k^-$ is calculated by using the dynamics model matrix A

$$\hat{x}_k^- = A_k \hat{x}_{k-1} \tag{3-2}$$

the *a priori* estimate of the error covariance matrix is calculated by

$$P_k^- = A_k P_{k-1} A_k^T + Q_k, \tag{3-3}$$

where Q_k is the process noise covariance and P_{k-1} is the *a posteriori* estimate of the error covariance.

Correction Step

In the correction step, the *a posteriori* state estimate is calculated using

$$\hat{x}_k = \hat{x}_k^- + K_k \left(z_k - H_k \hat{x}_k^- \right), \tag{3-4}$$

where K_k is the Kalman gain, computed by

$$K_k = P_k^- H_k^T \left(H_k P_k^- H_k^T + R_k \right)^{-1}, \tag{3-5}$$

where R_k is the measurement noise covariance.

Finally, the *a posteriori* estimate of the error covariance is given by

$$P_k = \left(I - K_k H_k \right) P_k^-. \tag{3-6}$$

Because of the assumptions that it is being built upon, the Kalman filter is the most computationally efficient of all the filters that are presented here. However, object motion can be nonlinear and therefore many researchers use the EKF and UKF filters which are capable of handling nonlinear dynamics and measurement models.

3-2-2 Extended Kalman Filter (EKF)

The Extended Kalman Filter (EKF) [27], extends the Kalman filter so that nonlinear functions can be used as the system's dynamic and measurement models (Equation 3-7). The system is then modeled as:

$$\begin{aligned} x_k &= f\left(x_{k-1}\right) + w_k \\ z_k &= h\left(x_k\right) + v_k \end{aligned} \tag{3-7}$$

where f is the nonlinear dynamics model, h is the nonlinear measurement model, w_k is the Gaussian distributed process noise and v_k is the Gaussian distributed measurement noise. The basic principle of the EKF is based around linearizing the models with a first order Taylor series expansion and then applying the regular Kalman filter equations.

In cases that the functions f and h are relatively linear and the posterior distribution does not have local maximum values (is unimodal), then the EKF can produce good approximations of the true belief. However, if the assumption of local linearity is violated, it can result in highly unstable filters.

The EKF, like the Kalman Filter, uses a predictor-corrector mechanism and therefore its operation can be broken down into prediction and correction steps.

Prediction Step

Given a state vector estimate $\hat{x}_{k-1}$, calculated at time step $k-1$, the *a priori* state estimate $\hat{x}_k^-$ is calculated by using the dynamics model f

$$\hat{x}_k^- = f\left(\hat{x}_{k-1}\right) \tag{3-8}$$

and the *a priori* estimate of the error covariance matrix is calculated by

$$P_k^- = F_k P_{k-1} F_k^T + Q_k \tag{3-9}$$

where Q_k is the process noise covariance, P_{k-1} is the *a posteriori* estimate of the error covariance and F_k is Jacobian matrix which linearizes the dynamics function f

$$F_{k,[i,j]} = \frac{\partial f_{(i)}}{\partial x_{(j)}}\left(\hat{x}_{k-1}\right) \tag{3-10}$$

Correction Step

In the correction step, first the *a posteriori* state estimate is calculated using

$$\hat{x}_k = \hat{x}_k^- + K_k\left(z_k - H_k\hat{x}_k^-\right), \tag{3-11}$$

where K_k is the Kalman gain and H_k is a Jacobian matrix which linearizes the measurement function h used to combine the measurement vector z_k. H_k is calculated in the following way:

$$H_{k,[i,j]} = \frac{\partial h_{(i)}}{\partial x_{(j)}}\left(\hat{x}_k^-\right), \tag{3-12}$$

while the Kalman gain is computed by

$$K_k = P_k^- H_k^T\left(H_k P_k^- H_k^T + R_k\right)^{-1} \tag{3-13}$$

where R_k is the measurement noise covariance.

Finally, the *a posteriori* estimate of the error covariance is given by

$$P_k = \left(I - K_k H_k\right) P_k^- \tag{3-14}$$

3-2-3 Unscented Kalman Filter (UKF)

The UKF [28] uses another method to linearize the transformation of a Gaussian, when it is passed through a nonlinear function. Specifically, it performs a stochastic linearization through the use of a weighted statistical linear regression process [29], called the unscented transform. This transform deterministically picks a sample set of points, referred to as sigma points, and passes them through the nonlinear function. Usually, the chosen points are picked equal to the mean and at symmetric locations of the covariance.

Using the principle that a set of discretely sampled points can be used to parameterise mean and covariance, this estimator yields performance equivalent to the KF for linear systems yet

generalizes effectively to non linear systems without the linearization steps required by the EKF. The asymptotic complexity of the UKF algorithm is the same as for the EKF and for linear systems it can be shown that the estimates generated by the UKF are identical to those generated by the Kalman filter, as is the case also for the EKF.

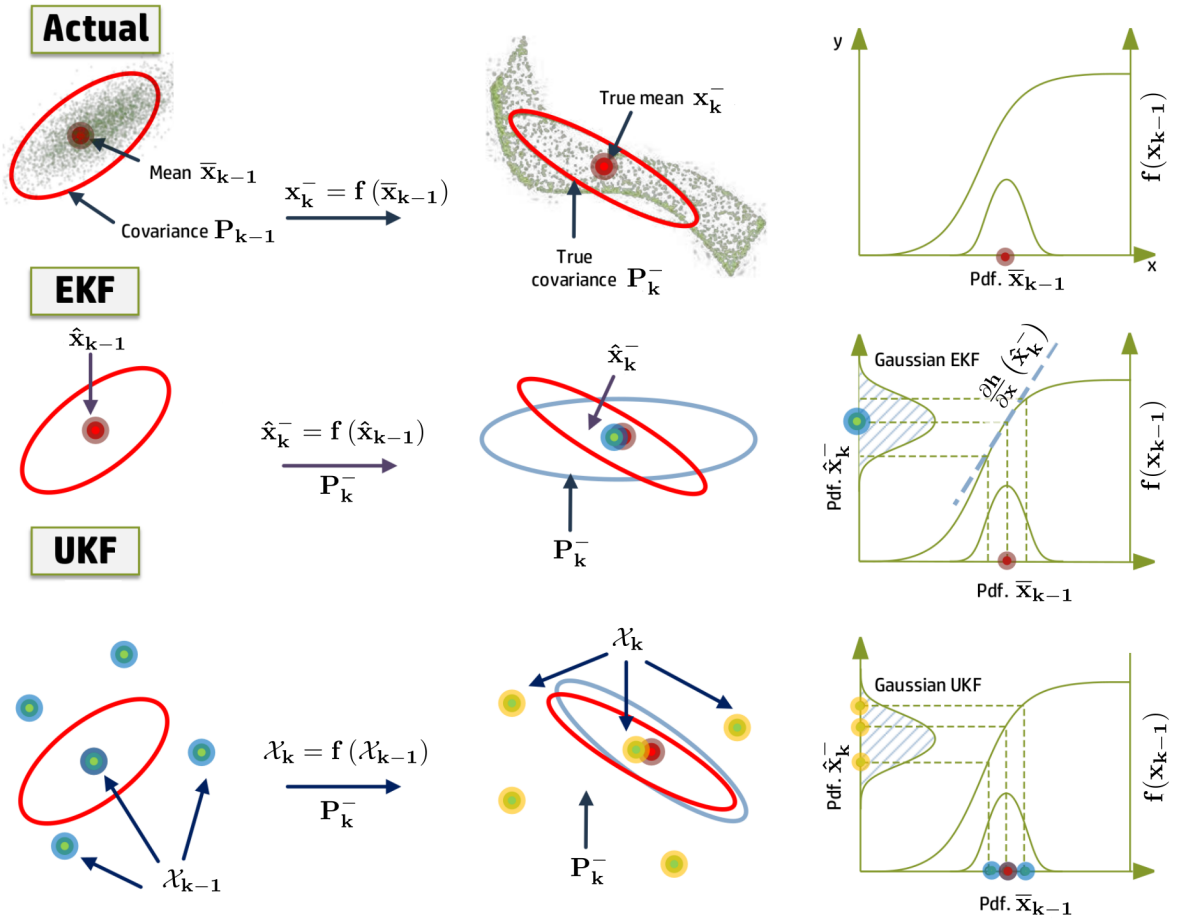


Figure 3-4: Comparison of the EKF and UKF in predicting the mean and covariance of a non linear system. Adapted from [8][9].

However, for nonlinear systems the UKF produces similar or better results than the EKF, where the improvement over the EKF depends on the amount of nonlinearities and the spread of the prior state uncertainty. Another advantage of the UKF over the EKF is the fact, that it is not necessary to derive the Jacobian matrices, which results in easier implementation. The implementation of the UKF in the proposed system, which is presented below, is based on the presentation on [30].

Prediction step

Given a state vector estimate $\hat{x}_{k-1}$, at time step $k-1$, a collection of sigma points is calculated which is stored in the columns of a $n \times (2n+1)$ sigma point matrix $\mathcal{X}_{k-1}$, where n is the state

vector dimension. The columns of $\mathcal{X}_{k-1}$ are computed by

$$\begin{aligned}(\mathcal{X}_{k-1})_0 &= \hat{x}_{k-1} \\ (\mathcal{X}_{k-1})_i &= \hat{x}_{k-1} + \left(\sqrt{(n+\lambda)P_{k-1}}\right)_i, \qquad i = 1 \dots n \\ (\mathcal{X}_{k-1})_i &= \hat{x}_{k-1} - \left(\sqrt{(n+\lambda)P_{k-1}}\right)_{i-n}, \quad i = n+1 \dots 2n,\end{aligned} \tag{3-15}$$

where $\left(\sqrt{(n+\lambda)P_{k-1}}\right)_i$ is the ith column of the matrix square root and λ is defined by

$$\lambda = \alpha^2(n+\kappa) - n, \tag{3-16}$$

where α is a scaling parameter which determines the spread of the sigma points and κ is a secondary scaling parameter. Note that it is assumed that $\left(\sqrt{(n+\lambda)P_{k-1}}\right)_i$ is symmetric and positive definite, which enables the calculation of the square root using a Cholesky decomposition. After the calcualation of $\mathcal{X}_{k-1}$, the prediction step is performed by propagating each column of $\mathcal{X}_{k-1}$ through time by Δt using

$$(\mathcal{X}_k)_i = f\left((\mathcal{X}_{k-1})_i\right), i = 0 \dots 2n \tag{3-17}$$

where f is the differential transition function of the system model. The *a priori* state estimate is then calculated

$$\hat{x}_k^- = \sum_{i=0}^{2L} W_i^{(m)} (\mathcal{X}_k)_i \tag{3-18}$$

where $W_i^{(m)}$ are weights defined by

$$\begin{aligned}W_0^{(m)} &= \frac{\lambda}{(n+\lambda)} \\ W_i^{(m)} &= \frac{1}{2(n+\lambda)}, i = 1 \dots 2n.\end{aligned} \tag{3-19}$$

As the last part of the prediction step, the *a priori* error covariance is calculated

$$P_k^- = \sum_{i=0}^{2n} W_i^{(c)} \left[(\mathcal{X}_k)_i - \hat{x}_k^-\right] \left[(\mathcal{X}_k)_i - \hat{x}_k^-\right]^T + Q_k \tag{3-20}$$

where $Q_k i$ is the process error covariance matrix, whose weights are defined by

$$\begin{aligned}W_0^{(c)} &= \frac{\lambda}{(n+\lambda)} + \left(1 - \alpha^2 + \beta\right) \\ W_i^{(c)} &= \frac{1}{2(n+\lambda)}, i = 1 \dots 2n\end{aligned} \tag{3-21}$$

Note that β is a parameter used to incorporate any prior knowledge about the distribution of x.

Correction step

During the correction step, the columns of $\mathcal{X}_{k-1}$ are first transformed through the measurement function. Therefore, let

$$(\mathcal{Z}_k)_i = h\left((\mathcal{X}_k)_i\right), i = 0 \dots 2n \tag{3-22}$$

$$\hat{z}_k^- = \sum_{i=0}^{2n} W_i^{(m)} (\mathcal{Y}_k)_i \tag{3-23}$$

where h represents the measurement model of the system. With the transformed state vector $\hat{z}_k^-$, the *a posteriori* state estimated is calculated using

$$\hat{x}_k = \hat{x}_k^- + K_k \left(z_k - \hat{z}_k^-\right), \tag{3-24}$$

where K_k is the Kalman gain, which in the UKF formulation is defined by

$$K_k = P_{\hat{x}_k \hat{z}_k} P_{\hat{z}_k \hat{z}_k}^{-1} \tag{3-25}$$

where

$$P_{\hat{z}_k \hat{z}_k} = \sum_{i=0}^{2n} W_i^{(c)} \left[(\mathcal{Z}_k)_i - \hat{z}_k^-\right] \left[(\mathcal{Z}_k)_i - \hat{z}_k^-\right]^T + R_k \tag{3-26}$$

$$P_{\hat{x}_k \hat{z}_k} = \sum_{i=0}^{2n} W_i^{(c)} \left[(\mathcal{X}_k)_i - \hat{x}_k^-\right] \left[(\mathcal{Z}_k)_i - \hat{z}_k^-\right]^T \tag{3-27}$$

where R_k is the measurement noise covariance matrix. The last part of the correction step is to compute the *a posteriori* estimate of the error covariance defined by

$$P_k = P_k^- - K_k P_{\hat{z}_k \hat{z}_k} K_k^T. \tag{3-28}$$

3-3 L-shape Tracker

In this section, the different ways that the system tracks vehicle motion and dimensions will be presented. In the previous chapter, we have explained that in order to simplify the LIDAR measurements L-shapes are extracted from every cluster at every time step. Those L-shapes are used as representations of the vehicles and therefore their motion and dimensions are tracked. However, the L-shapes represent only two sides of a given vehicle and therefore at a later stage a box model for every vehicle is calculated.

Summarizing the previous chapter, every L-shape feature contains five values that are extracted from the bounding boxes. Those values are, the position of the corner point (x_{corner}, y_{corner}), the lengths L_1, L_2 and the orientation angle (θ) of L_1 (Figure 3-5). Those five measurements are divided in two groups, the x_{corner} and the y_{corner}, are used as measurements for updating the filter that tracks the motion of the L-shape and L_1, L_2, θ are used for updating the filter that tracks its shape.

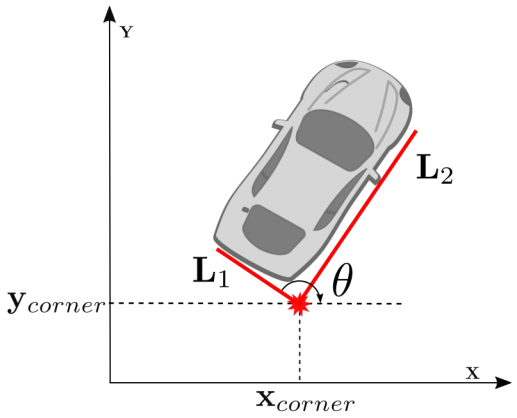


Figure 3-5: Every L-shape consists of the x_{corner},y_{corner}, L_1, L_2, θ values. The corner of the bounding box that is closest to the LIDAR sensor is selected as the L-shape corner.

In order to track the position of the L-shape corner, the proposed system implements two solutions. The first one uses a Kalman Filter for tracking the corner of the L-shape, while the second one uses a UKF. The first approach is based on the work presented in [7], while the second one is novel.

The system implements two solutions for two main reasons: the first being for comparing the accuracy of the Kalman Filter and the UKF in this particular application. And the second one is, providing the users of the system with options, so they can make a choice depending on the accuracy demanded by their application and the available computational resources of their platform.

The dimensions (L_1, L_2), orientation (θ) and turn rate (ω) of the L-shape are tracked by a separate Kalman Filter.

3-3-1 Kalman Filter Kinematic Tracker

The Kalman Filter used for tracking the motion of the corner point uses a Constant Velocity (CV) model which estimates position and velocities (visualized in Figure 3-6), with the following state vector $\mathbf{x}_{CV}$.

$$x_{CV} = \begin{bmatrix} x_{corner} & y_{corner} & v_x & v_y \end{bmatrix}^T \tag{3-29}$$

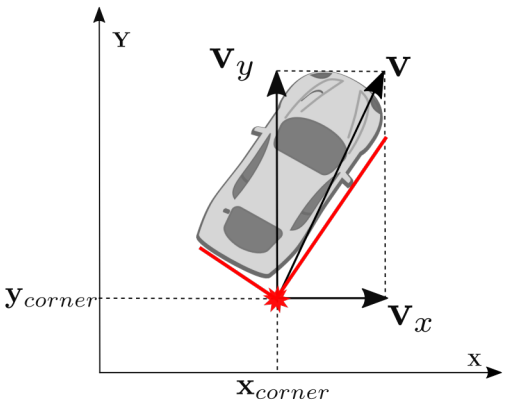


Figure 3-6: The kinematic model models the motion of the corner point.

The kinematic model A_{CV} that is used to track the position (x_{corner}, y_{corner}) and velocities v_x, v_y of the corner point, as show in Fig. 3-6, is the following:

$$A_{CV} = \begin{bmatrix} 1 & 0 & T & 0 \\ 0 & 1 & 0 & T \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3-30}$$

where T is the sampling time. Given that only the position of the corner point is measured, the measurement vector and model are the following:

$$\begin{aligned} z_{CV} &= \begin{bmatrix} x_{corner} & y_{corner} \end{bmatrix}^T \\ H_{CV} &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \end{aligned} \tag{3-31}$$

The process noise covariance matrix Q_{CV} was configured initially based on [31], in the following way:

$$Q_{CV} = \begin{bmatrix} \sigma_x^2 T^4/4 & \sigma_x^2 T^3/2 & 0 & 0 \\ \sigma_x^2 T^3/2 & \sigma_x^2 T^2 & 0 & 0 \\ 0 & 0 & \sigma_y^2 T^4 & \sigma_y^2 T^3/2 \\ 0 & 0 & \sigma_y^2 T^3/2 & \sigma_y^2 T^2 \end{bmatrix} \tag{3-32}$$

where $\sigma_x^2 = \sigma_y^2 = 150$, which according to [31] is for medium sensor noise scenarios.

However, the above tuning of the Q_{CV} matrix resulted in sub optimal estimations, compared to the final tuning of the Q_{CV} and R_{CV} matrices presented below:

$$Q_{CV} = \begin{bmatrix} T & 0 & 0 & 0 \\ 0 & T & 0 & 0 \\ 0 & 0 & 10T & 0 \\ 0 & 0 & 0 & 10T \end{bmatrix} \qquad R_{CV} = \begin{bmatrix} \sigma_x & 0 \\ 0 & \sigma_y \end{bmatrix} \tag{3-33}$$

where σ_x, σ_y where tuned depending on the scale of the experiment.

3-3-2 UKF Kinematic Tracker with Coordinated Turn Model

Higher tracking accuracy of vehicles during turns can be achieved by utilizing a Coordinated Turn Model with Cartesian Velocity [32], which cannot be used with a standard Kalman filter since it is nonlinear. This model in addition to the four states that are tracked by the Constant Velocity model, tracks also the turn rate of the vehicle (ω). Therefore, the state vector $\mathbf{x}_{CTM}$ is:

$$\begin{bmatrix} x & y & v_x & v_y & \omega \end{bmatrix}^T \tag{3-34}$$

And its kinematic function f_{CTM} is the following:

$$f_{CTM}(x) = \begin{bmatrix} x + \frac{v_x}{\omega}\sin(\omega T) - \frac{v_y}{\omega}(1 - \cos(\omega T)) \\ y + \frac{v_x}{\omega}(1 - \cos(\omega T)) + \frac{v_y}{\omega}\sin(\omega T) \\ v_x\cos(\omega T) - v_y\sin(\omega T) \\ v_x\sin(\omega T) + v_y\cos(\omega T) \\ \omega \end{bmatrix} \tag{3-35}$$

The measurement vector and model are similar to the ones used in the Kalman Filter, since the available measurements are the same.

$$\begin{aligned} z_{CTM} &= \begin{bmatrix} x_{corner} & y_{corner} \end{bmatrix}^T \\ H_{CTM} &= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}. \end{aligned} \tag{3-36}$$

The process and measurement noise covariance matrices that were used for this model are the following:

$$Q_{CTM} = \begin{bmatrix} T & 0 & 0 & 0 & 0 \\ 0 & T & 0 & 0 & 0 \\ 0 & 0 & 10T & 0 & 0 \\ 0 & 0 & 0 & 10T & 0 \\ 0 & 0 & 0 & 0 & 0.5T \end{bmatrix} \qquad R_{CTM} = \begin{bmatrix} 0.1\sigma_x & 0 \\ 0 & 0.1\sigma_y \end{bmatrix} \tag{3-37}$$

The α paremeter of the UKF was chosen equal to 0.0025, β was chosen equal to 2 and κ was chosen as 0.

3-3-3 Alternative kinematic models

In addition to the kinematic model presented above, two other kinematic models were implemented and tested. Those are the Omnidirectional kinematic model [33] and the Constant Turn-Rate Velocity (CTRV) model [31]. However, their estimations are not presented in the evaluation chapter, because they are considerably less reliable than the ones from the CTM model.

Omnidirectional Model

This model uses as kinematic function f an omnidirectional 2D discrete time kinematic model [33], which in addition to the position and speed tracks also, the yaw (ψ) and turn rate (ω). Therefore, the state vector $\mathbf{x}_{OMNI}$ is:

$$x_{OMNI} = \begin{bmatrix} x & y & \psi & v_x & v_y & \omega \end{bmatrix}^T \tag{3-38}$$

and the kinematic function is the following:

$$f_{OMNI} = \begin{bmatrix} x + (v_x \cos\psi - v_y \sin\psi)\, T \\ y + (v_x \sin\psi + v_y \cos\psi)\, T \\ \psi + \omega T \\ v_x \\ v_y \\ \omega \end{bmatrix}. \tag{3-39}$$

Constant Turn-Rate Velocity (CRTV) Model

This model makes an assumption that the velocity and the turn rate of the tracked vehicle are constant, while it is also tracking the position (x, y) and the orientation (ψ).

Therefore, the state vector $\mathbf{x}_{CTRV}$ is:

$$x_{CTRV} = \begin{bmatrix} x & y & \psi & v & \omega \end{bmatrix}^T \tag{3-40}$$

and the kinematic function used is the following:

$$f_{CTRV} = \begin{bmatrix} x + \frac{v}{\omega}\left(-\sin(\psi) + \sin(T\psi + \psi)\right) \\ y + \frac{v}{\omega}\left(\cos(\psi) - \cos(T\omega + \psi)\right) \\ T\psi + \psi \\ v \\ \omega \end{bmatrix}. \tag{3-41}$$

3-3-4 Shape Tracker

The shape of the target vehicle is tracked using a Kalman Filter and a state vector composed of line lengths (L_1, L_2), the orientation of $L_1(\theta)$ and the yaw rate (ω). Those states are visualized in Figure 3-7 and are contained in vector $\mathbf{x}_s$.

$$\mathbf{x}_S = \begin{bmatrix} L_1 & L_2 & \theta & \omega \end{bmatrix}^T \tag{3-42}$$

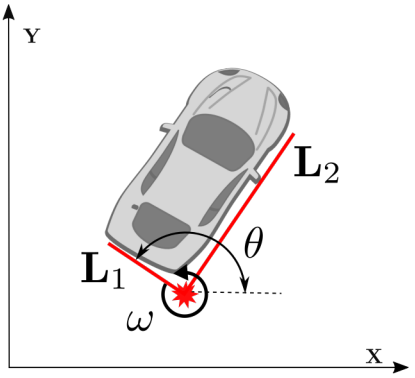


Figure 3-7: The shape filter estimates the yaw, yaw rate and size of the L-shape.

For estimating the vehicle's shape, a static model is applied to the line lengths (L_1, L_2) based on the assumption that the vehicle size does not change over time. For estimating the L-shape's yaw and, since the yaw rate does not change particularly fast, a constant turn rate model is chosen. The two above models are combined in a single process matrix:

$$A_S = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & T \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{3-43}$$

where T the sampling time and A_S is the process matrix containing the static model for the line lengths and the constant turn rate model for the orientation and the yaw rate.

Among the states of the shape model, only the yaw rate is not contained in the L-shape and therefore the measurement vector and model are the following:

$$\begin{aligned} z_S &= \begin{bmatrix} L_1 & L_2 & \theta \end{bmatrix}^T \\ H_S &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}. \end{aligned} \tag{3-44}$$

The process and measurement noise covariance matrices that were used for this model are

the following:

$$Q_S = \begin{bmatrix} T & 0 & 0 & 0 \\ 0 & T & 0 & 0 \\ 0 & 0 & T & \frac{T^2}{2} \\ 0 & 0 & 0 & T \end{bmatrix} \tag{3-45}$$

$$R_S = \begin{bmatrix} \frac{1}{10L1} & 0 \\ 0 & \frac{1}{10L2} \end{bmatrix}, \tag{3-46}$$

where the measurement noise of the length states is inversely proportional to their measurement to represent the uncertainty of measurement when a side of the vehicle is occluded.

At this point it should be mentioned that the shape model could be augmented with two additional states that track the length and width difference between time steps [11]. Those additional states can be used to detect occlusion events and reduce the change applied to the estimated line lengths, during those periods.

3-4 Corner Point Switching

The designed system is tracking L-shapes, which represent the closest corner of observed vehicles. However, while the ego vehicle and the observed ones are moving, it is expected that the closest corners of the other vehicles will be periodically changing and therefore the extracted L-shapes should also be changing.

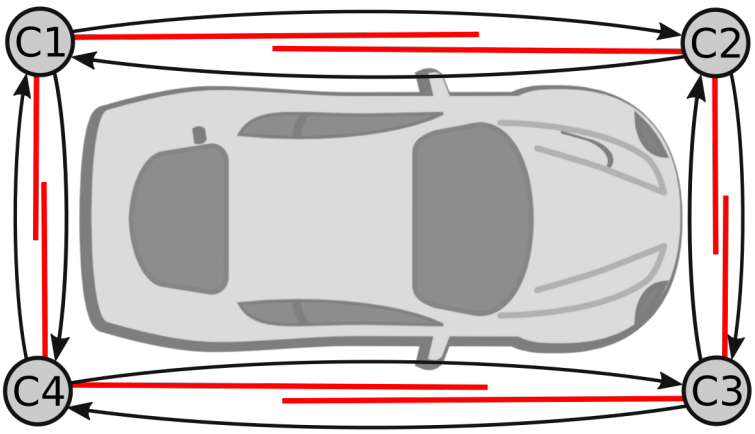
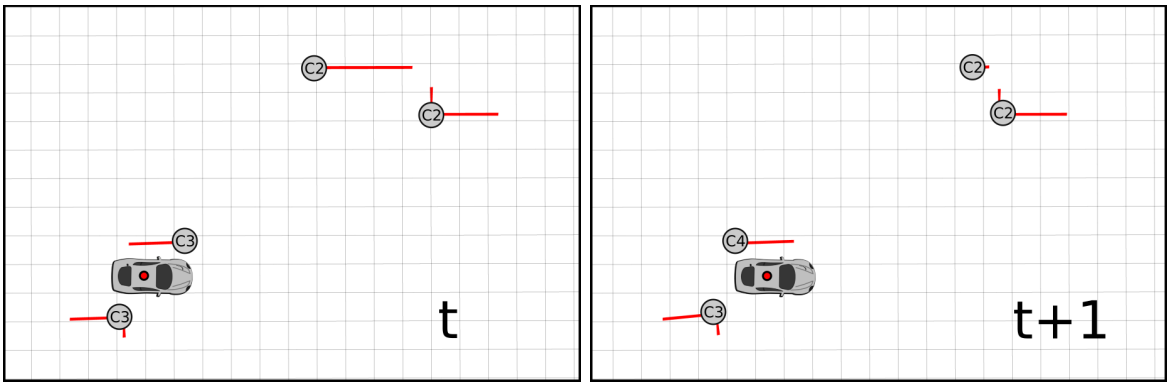


Figure 3-8: Visualization of all the corner points of a vehicle (C1-C4) and of the clockwise and counter clockwise changes between them (black arrows).

All the possible extracted L-shapes from a vehicle are drawn in Figure 3-8 and are marked by values C1 to C4.

An example of a corner change occurrence can be observed in Figure 3-9, in which two instances from the simulation are given side by side. In the first time instance (t), the overtaking vehicle at the left of the ego-vehicle is tracked by its front-right corner (C3). As it moves ahead in the second time instance ($t+1$), the closest corner changes to the one at the vehicle's rear-right corner (C4).



(a) First time instance. **(b)** Second time instance.

Figure 3-9: Clockwise change of closest corner point, from corner C3 to corner C4.

In the proposed system, corner point changes were detected based on the Mahalanobis distances of the new measurement to the already tracked corner point and the two neighboring ones. If for example, the already tracked corner point is C3, with state vector ($\mathbf{c_3}$), then the state vectors of corner points C2 ($\mathbf{c_2}$) and C4 ($\mathbf{c_4}$) are calculated and their values are compared with the state vector of the new L-shape measurement ($\mathbf{m}$), via the Mahalanobis distance. The Mahalanobis distance (D_M), which is a multivariate distance metric, is calculated based on the following formula:

$$\begin{aligned} D_M^2 &= (\mathbf{c}-\mathbf{m})^T \cdot P^{-1} \cdot (\mathbf{c}-\mathbf{m}) \\ \mathbf{c} &= [x_{corner},\ y_{corner},\ \theta,\ L_1,\ L_2,]^T \\ \mathbf{m} &= [x_{corner}, y_{corner}, \theta, L_1, L_2,]^T \end{aligned} \tag{3-47}$$

where P is the covariance matrix of the already tracked corner point.

If the shortest Mahalanobis distance is between the measurement vector $\mathbf{m}$ and the previously tracked corner vector, for example $\mathbf{c_3}$, then no corner switch took place. However, if the shortest distance is with $\mathbf{c_4}$, a clockwise switch took place and if it is with $\mathbf{c_2}$, a counter clockwise switch took place. In the example of Figure 3-9, we can observe that a clockwise corner switch between corners C3 and C4 occurred. In such cases, the filters that track the L-shape cannot be immediately updated because the new L-shape measurement corresponds to another corner than the one that the filter states are tracking. Therefore, the filter states should be altered, so that the represent the same corner with the new measurement.

In case that a clockwise switch is detected, the filter states are changed in the following way:

$$\begin{aligned} x_{\text{corner}}^{Cj} &= x_{\text{corner}}^{Ci} + L_1^{Ci}\cos\theta^{Ci} \\ y_{\text{corner}}^{Cj} &= y_{\text{corner}}^{Ci} + L_1^{Ci}\sin\theta^{Ci} \\ v_x^{Cj} &= v_x^{Ci} + L_1^{Ci}\omega\sin\theta^{Ci} \\ v_y^{Cj} &= v_y^{Ci} + L_1^{Ci}\omega\cos\theta^{Ci} \\ \theta^{Cj} &= \theta^{Ci} - \pi/2 \\ L_1^{Cj} &= L_2^{Ci} \\ L_2^{Cj} &= L_1^{Ci} \\ &\text{where} \begin{cases} j = i+1, & i < 4 \\ j = 1, & i = 4 \end{cases} \end{aligned} \tag{3-48}$$

where the superscripts Ci and Cj represents the corner number before and after the model change.

In case that a counter clockwise switch is detected the kinematic and shape states are changed in the following way:

$$\begin{aligned} x_{\text{corner}}^{Cj} &= x_{\text{corner}}^{Ci} + L_2^{Ci}\sin\theta^{Ci} \\ y_{\text{corner}}^{Cj} &= y_{\text{corner}}^{Ci} + L_2^{Ci}\cos\theta^{Ci} \\ v_x^{Cj} &= v_x^{Ci} + L_2^{Ci}\omega\cos\theta^{Ci} \\ v_y^{Cj} &= v_y^{Ci} + L_2^{Ci}\omega\sin\theta^{Ci} \\ \theta^{Cj} &= \theta^{Ci} + \pi/2 \\ L_1^{Cj} &= L_2^{Ci} \\ L_2^{Cj} &= L_1^{Ci} \\ &\text{where} \begin{cases} j = i-1, & i > 1 \\ j = 4, & i = 1 \end{cases} \end{aligned} \tag{3-49}$$

An enhancement of this corner model switching scheme is presented in [34], in which the transitions between the corner points are used for characterizing the motion of the observed vehicles (overtaking, lane keeping, U-turn).

3-5 L-shape to Box Model Conversion

In the L-shape to box model conversion stage the kinematic state is changed from corner point motion to vehicle motion. The position of the center of the box model can be calculated by using the geometric information of the shape model

$$\begin{aligned} x_{center} &= x_{\text{corner}} + \varepsilon_x, \quad \text{where } \varepsilon_x = \left(L_1\cos\theta + L_2\sin\theta\right)/2 \\ y_{center} &= y_{\text{corner}} + \varepsilon_y, \quad \text{where } \varepsilon_y = \left(L_1\sin\theta - L_2\cos\theta\right)/2. \end{aligned} \tag{3-50}$$

Since the velocity of the corner point is the sum of the target velocity and the tangential velocity, the rotational motion of the corner point includes both translational velocity and

rotational velocity. The rotational motion is assumed as a uniform circular motion, since a constant turn rate model for the shape model has already been assumed. Therefore, an equation for tangential velocity can be derived by using the distance from the center as the radius (r) of the circular motion.

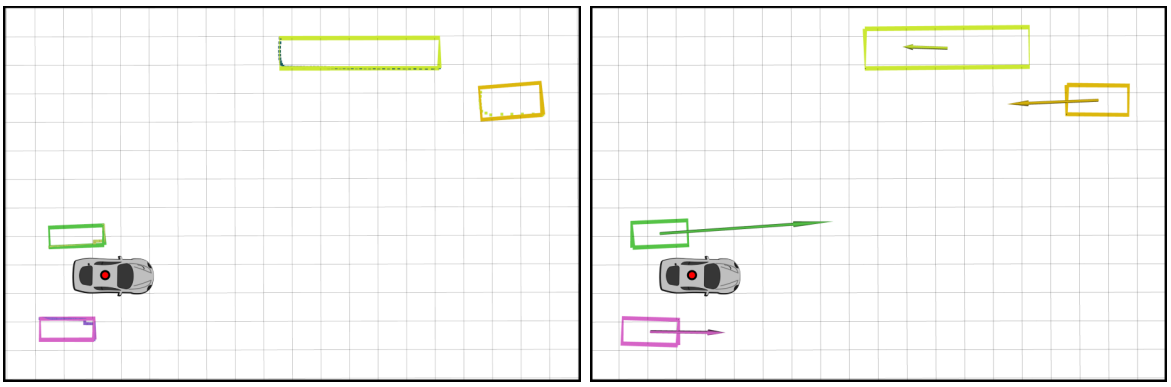
$$\begin{aligned} v_{x,\text{center}} &= v_{x,\text{corner}} - r\omega\cos\alpha \\ v_{y,\text{center}} &= v_{y,\text{corner}} - r\omega\sin\alpha \\ &\text{where } \alpha = \tan^{-1}(\varepsilon_y/\varepsilon_x) - \pi/2. \end{aligned} \tag{3-51}$$

A vehicle's yaw (ψ) is typically difficult to estimate from an L-shape because of the ambiguities, since it is hard to judge which is the front part and which is the rear part of the vehicle. Therefore, it would be reasonable to keep the following four hypotheses:

$$\psi_i = \theta + i\frac{\pi}{2}, i \in \{0, 1, 2, 3\} \tag{3-52}$$

where ψ_i represents the yaw angle of the vehicle [35]. Next, for calculating the actual yaw of the vehicle, these four angles are compared with the angle of the vehicle's speed and the one with the least absolute difference from the speed's orientation is chosen as the vehicle yaw. The turn rate of the box model is taken equal to the turn rate (ω) of the tracked L-shape.

In Figure 3-10 a comparison is made between the output of the rectangle fitting algorithm and the output of the box model conversion stage. In Figure 3-10b arrows are used to represent the estimated velocities of the surrounding vehicles, while the starting point of the arrows are the estimated center of the surrounding vehicles.



(a) Output of rectangle extraction. **(b)** Output after L-shape to box model conversion.

Figure 3-10: Comparison between the output of the rectangle fitting algorithm and the boxes calculated from the L-shapes.

It can additionally be observed, that the dimensions of the box models are bigger than the extracted rectangles, since they are based on the estimations of the shape filter and that the estimated orientation of the truck at the top right corner is closer to the true value, than the one estimated by the rectangle fitting algorithm (Fig. 2-7b).

Chapter 4

Experimental Evaluation

In this chapter, the experiments used to test and evaluate the proposed system will be presented and the evaluation results will be explained. The developed system was tested with two different methods, the first method was in a simulation environment, and the second one were experiments with robotic cars. The simulation environment provided a reliable and accurate way to test the proposed system at every step of its development. While the experiments with the robotic cars, proved the usefulness of the proposed system as a tool for other researchers, with similar robotic platforms.

4-1 Simulation Experiments

The simulation environment[1] was developed with the goal of providing a fast, repeatable and accurate way to test the developed system in multiple scenarios. It was created in the Gazebo simulator and is based around the complete mechanical model of a Toyota Prius car, which is used both as the ego vehicle and the two tracked ones. For all experiments, the three Prius cars were controlled by joystick inputs, since the added randomness of human input was perceived as an advantage over predetermined path following.

During this simulation experiment the output of the system and the reference measurements are both recorded and they are later analyzed to assess the accuracy of the proposed system. The reference measurements provided by the simulation environment are very accurate and with sufficiently higher frequency (100 Hz) than the estimations of the system (12 Hz) and therefore are used as ground truth data. For evaluating the tracking performance, the system's estimates for the position (x, y), velocities (v_x, v_y), orientation (ψ), turn rate (ω) and dimensions (Length, Width)of the box models are plotted against the ground truth data, which also refer to the center of the vehicles.

Although, at this point only LIDAR data were used, the simulated Prius cars are additionally equipped with camera and radar sensors, which can be used for sensor fusion experimentation in the future.

[1]The highway simulator can be found at: https://github.com/kostaskonkk/highway_driving_simulator

4-1-1 Overtakes Experiment

The overtakes experiment[2] was one of the most challenging simulation experiments for the developed system. In Figure 4-1 which is a snapshot of the simulation environment during this experiment, we can observe that there a total of five vehicles, from which the three at the bottom lanes are controlled via joystick input, while the bus and truck at the top lanes are following straight paths. The ego vehicle is the Prius in the middle lane and during this experiment it overtakes the car that drives on the emergency lane. At the same time, the third car which also starts in the middle lane, changes to the left lane, overtakes the ego-vehicle and then merges back in the middle lane in front of it.

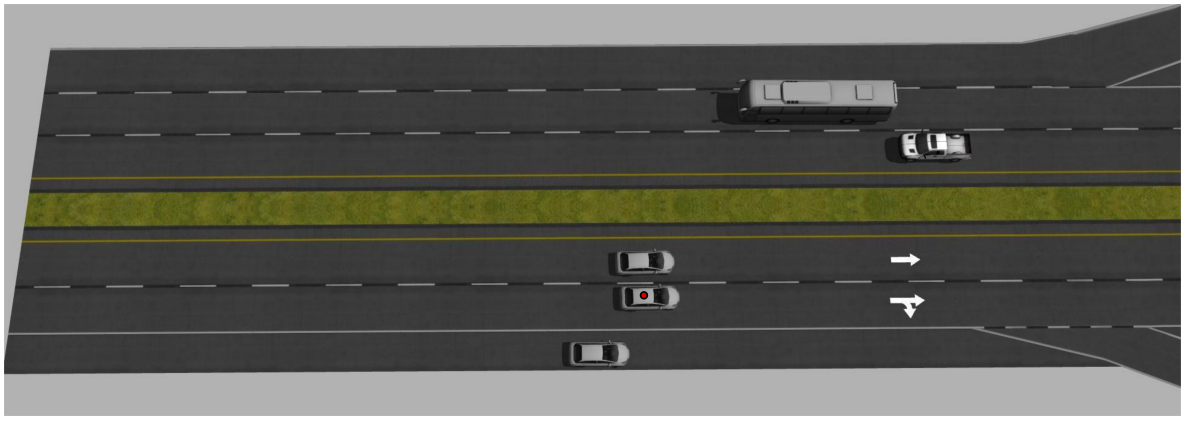


Figure 4-1: Image of the developed simulation environment, during the overtakes experiment.

The estimations of the system are plotted against the ground truth data on Fig. 4-2 for the car at the emergency lane and at Fig. 4-4 for the car at the left lane, while the statistics of the estimation errors are plotted in Figure 4-3 and 4-5, respectively.

In Fig. 4-2, we can observe that both filters produce identical estimates for the position (x, y) of the car in the emergency lane. However, the velocities estimated by the two filters are different, with the KF overshooting the reference and the UKF undershooting it. This can be explained, by the different process models that the filters employ. The orientation of the vehicle is estimated with high accuracy by the shape filter, after the vehicle starts moving. The turn rate however is not estimated accurately since its abrupt changes were not tracked correctly by the filter. Lastly, it is evident that the accuracy of length and width estimation, depends on which side the incoming measurements represent. Therefore, when the length is measured the length estimation is accurate and when the width is measured its estimation is more accurate. The two spikes at the beginning of those diagrams can be explained by the fact that, the orientation of the vehicle can not be estimated when the vehicles are stationary.

The results for the overtaking car are drawn in Fig. 4-4, where it can be observed that the system has generally similar performance. However, in the (v_y) graph, a spike can be seen, which is attributed to two rapid corner switches that occur while the vehicles are almost stationary. In both cases, it is evident that the UKF offers no significant benefits to the KF, and that is probably due to the motion of the two vehicles, which is generally straight, while the model of the UKF estimates the kinematics of constant turns.

[2] A video of the experiment can be found at: https://youtu.be/JrbJjmIv730

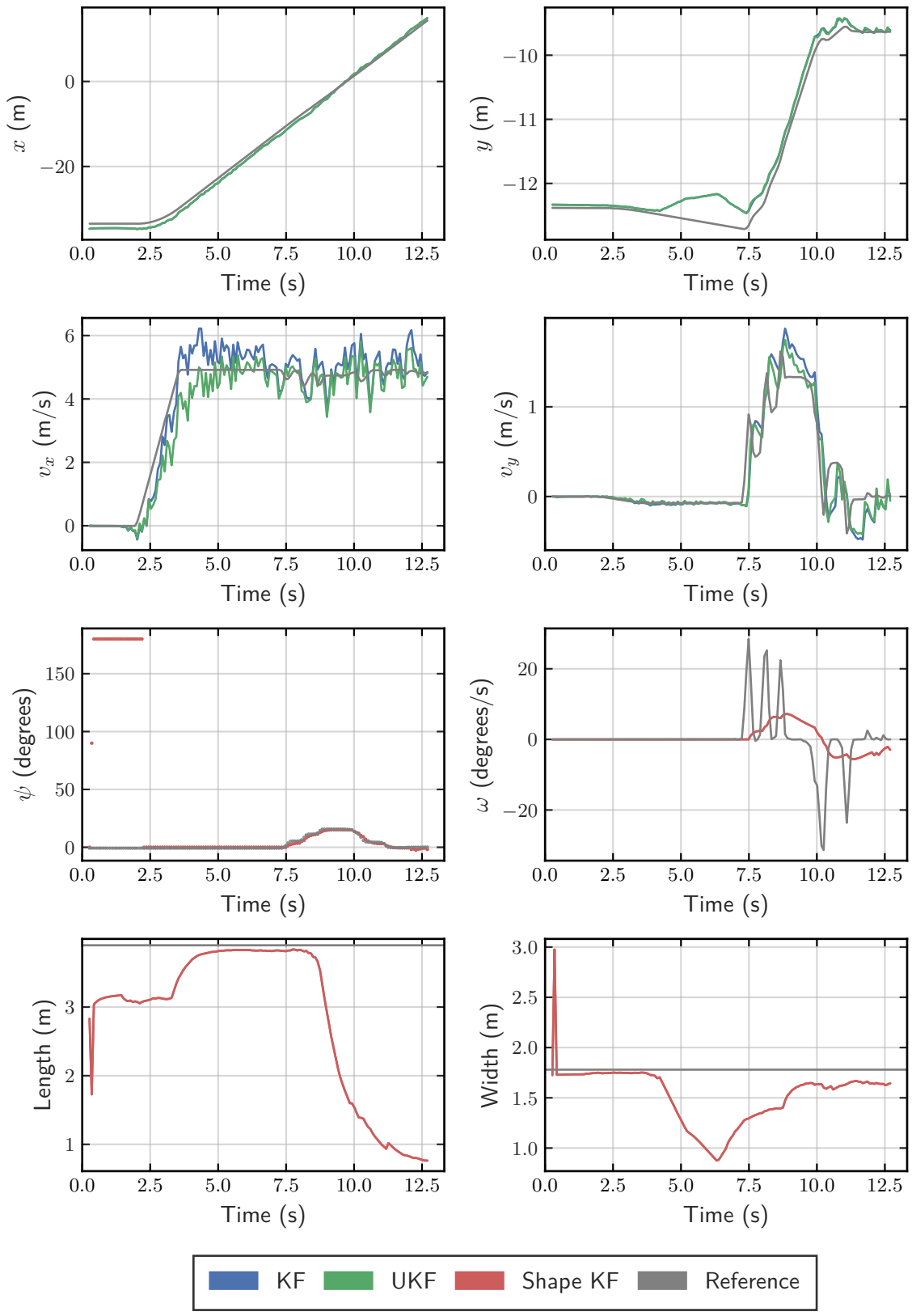


Figure 4-2: Estimated states by the developed system against the simulation ground truth, for the car at the right of the ego-vehicle.

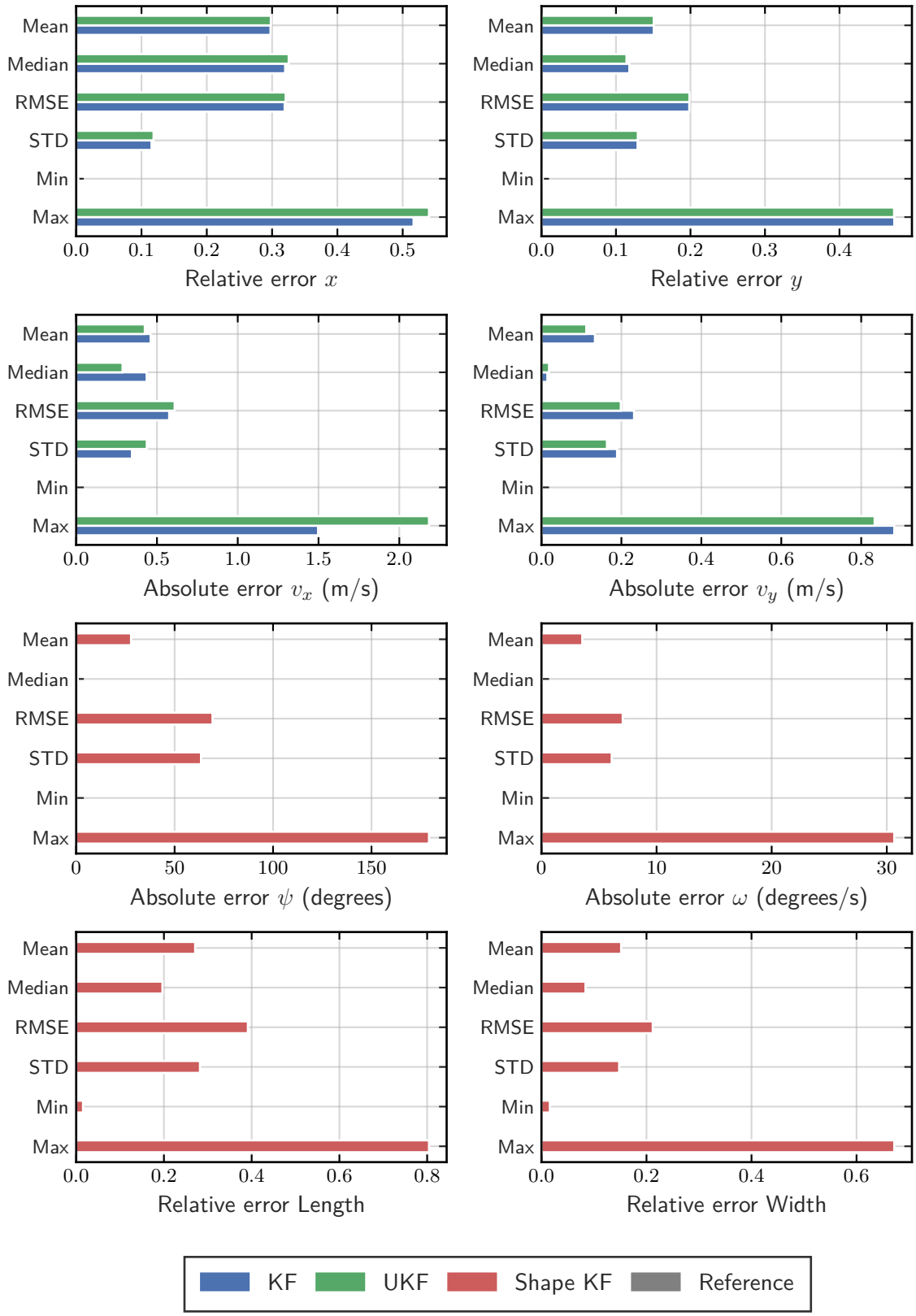


Figure 4-3: Statistics of the errors of the estimated states by the developed system against the simulation ground truth, for the car at the right of the ego-vehicle.

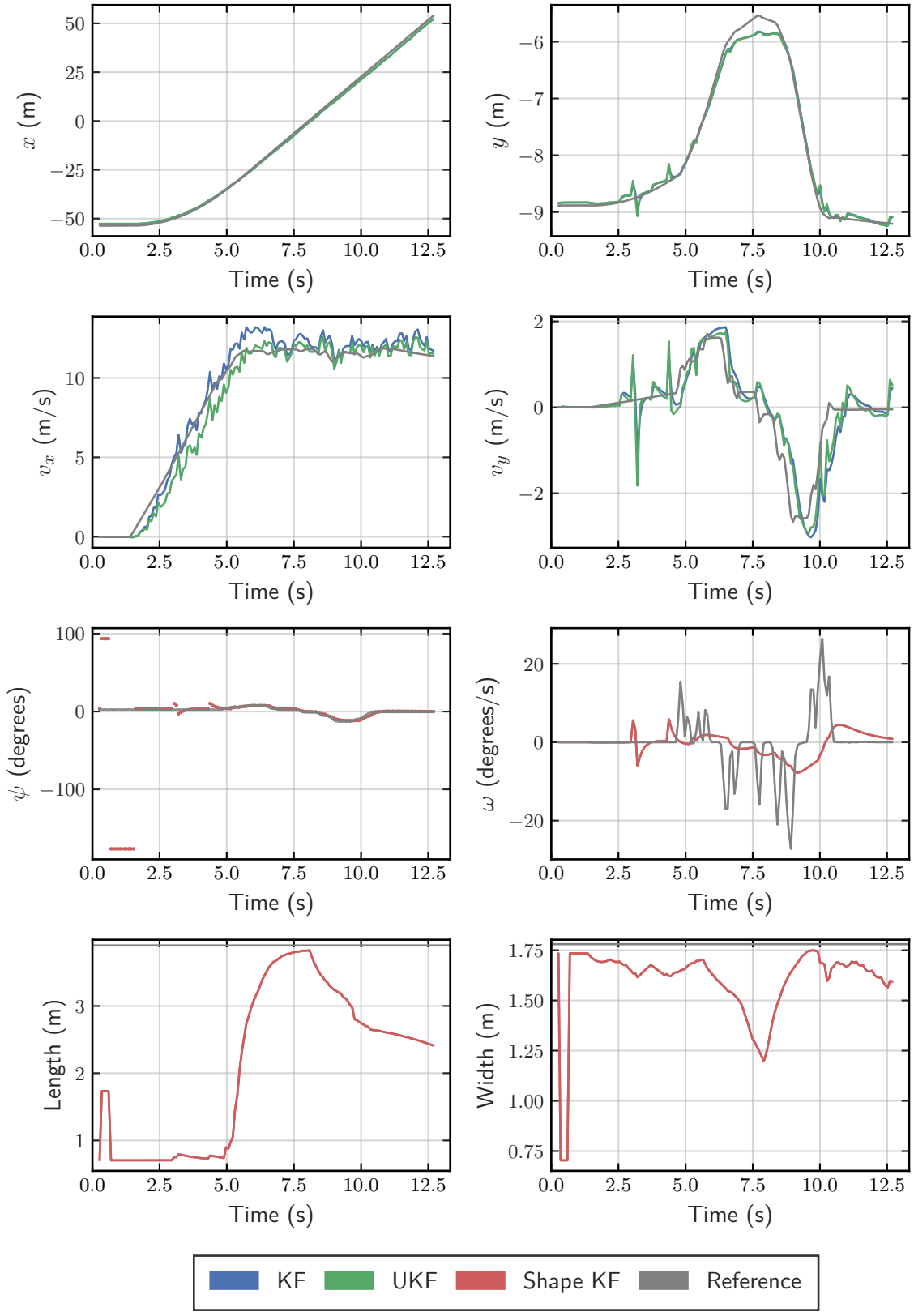


Figure 4-4: Estimated states by the developed system against the simulation ground truth, for the over passing car at the left of the ego-vehicle.

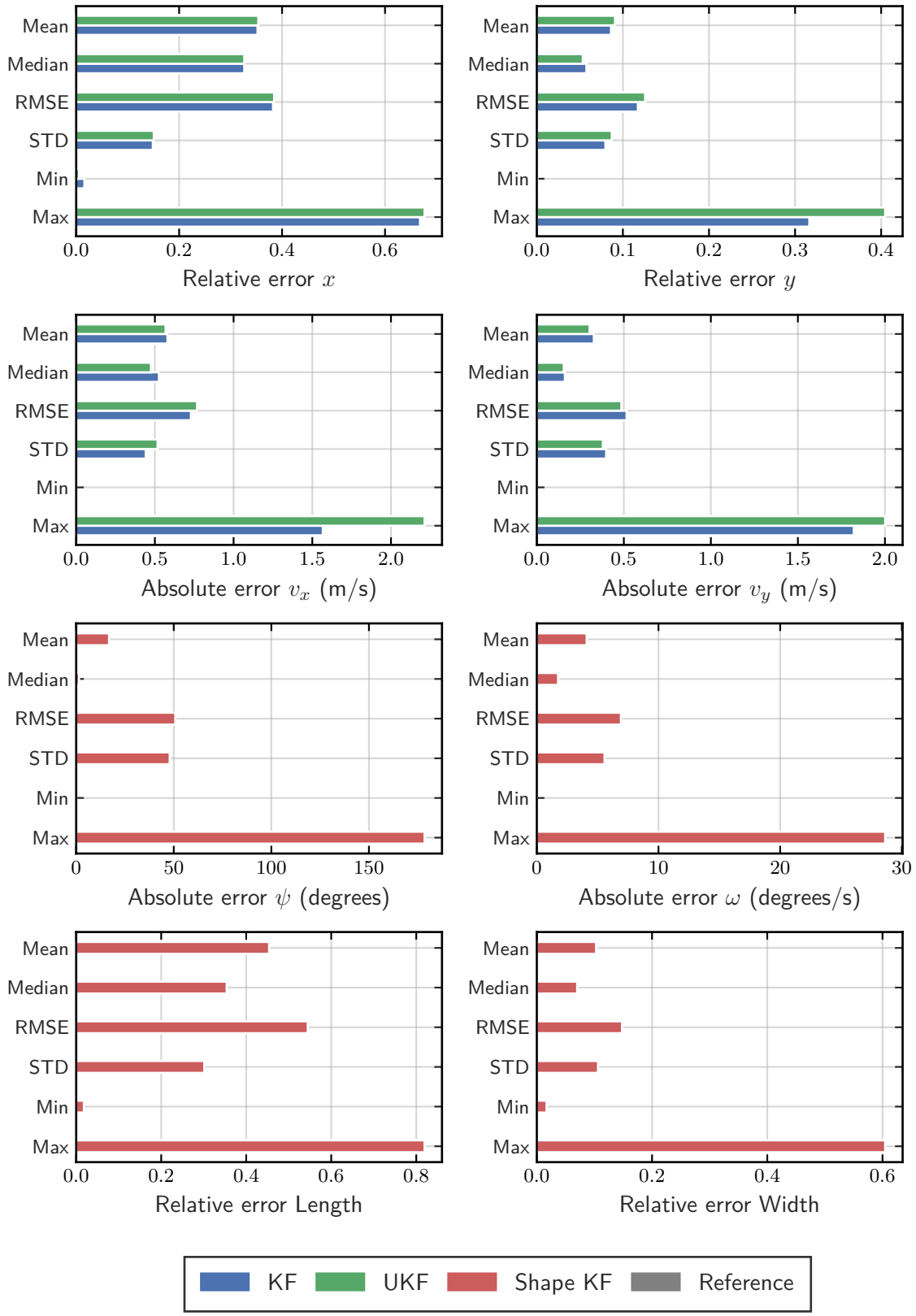


Figure 4-5: Statistics of the errors of the estimated states by the developed system against the simulation ground truth, for the over passing car at the right of the ego-vehicle.

4-1-2 Opposite Lane Tracking

The opposite lane tracking experiment[3] presented a different challenge to the developed system, since it measured the error in the case that the target vehicle moves in the opposite direction of the ego vehicle.

In Figure 4-6 which is a snapshot of the simulation environment taken during this experiment, we can observe that there a total of three Prius vehicles, with the one in the middle being the ego vehicle and the one at its opposite lane being the one whose tracking will be evaluated. Although, the car at the right of the ego vehicle is also tracked, the acquired results are not presented, since it follows a similar motion to the previous experiment.

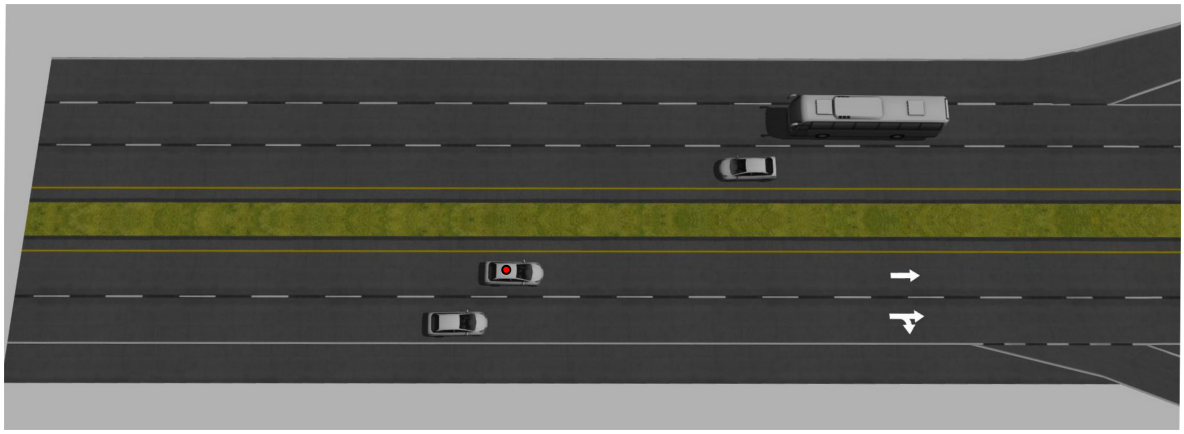


Figure 4-6: Image of the developed simulation environment, during the overtakes experiment.

The estimations of the system and the reference measurements given by the simulation environment are plotted on Figure 4-7, while the statistics of the errors are plotted on Figure 4-8.

In Figure 4-7, we can observe that in this experiment, the tracked car is not visible from the beginning since it is far away from the ego vehicle and therefore it starts being tracked after the third second of the experiment. The x position of the car in the opposite lane is tracked accurately by both filters, and even though it seems that the y position estimates are not very accurate, this is due to fact that the car moves less than a meter in this direction during the whole experiment. This can also be seen in Figure 4-8 where the relative error for y has half the RMSE than that of x.

The velocity estimation is comparable for the two filters, however the UKF converges faster to the true value of v_x than the KF, but it is also more sensitive to noise and therefore produces worse estimates for the v_y orientation. The estimated orientation is accurate when the vehicle comes in close range of the LIDAR sensor, however the turn rate estimation makes a big error because of an undetected corner point switch. Lastly, the estimation of the vehicle's dimensions is accurate, when the vehicles are close and measurements from both sides of the tracked vehicle are available.

[3] A video of the experiment can be found at: https://youtu.be/AkO3fpvwXCg

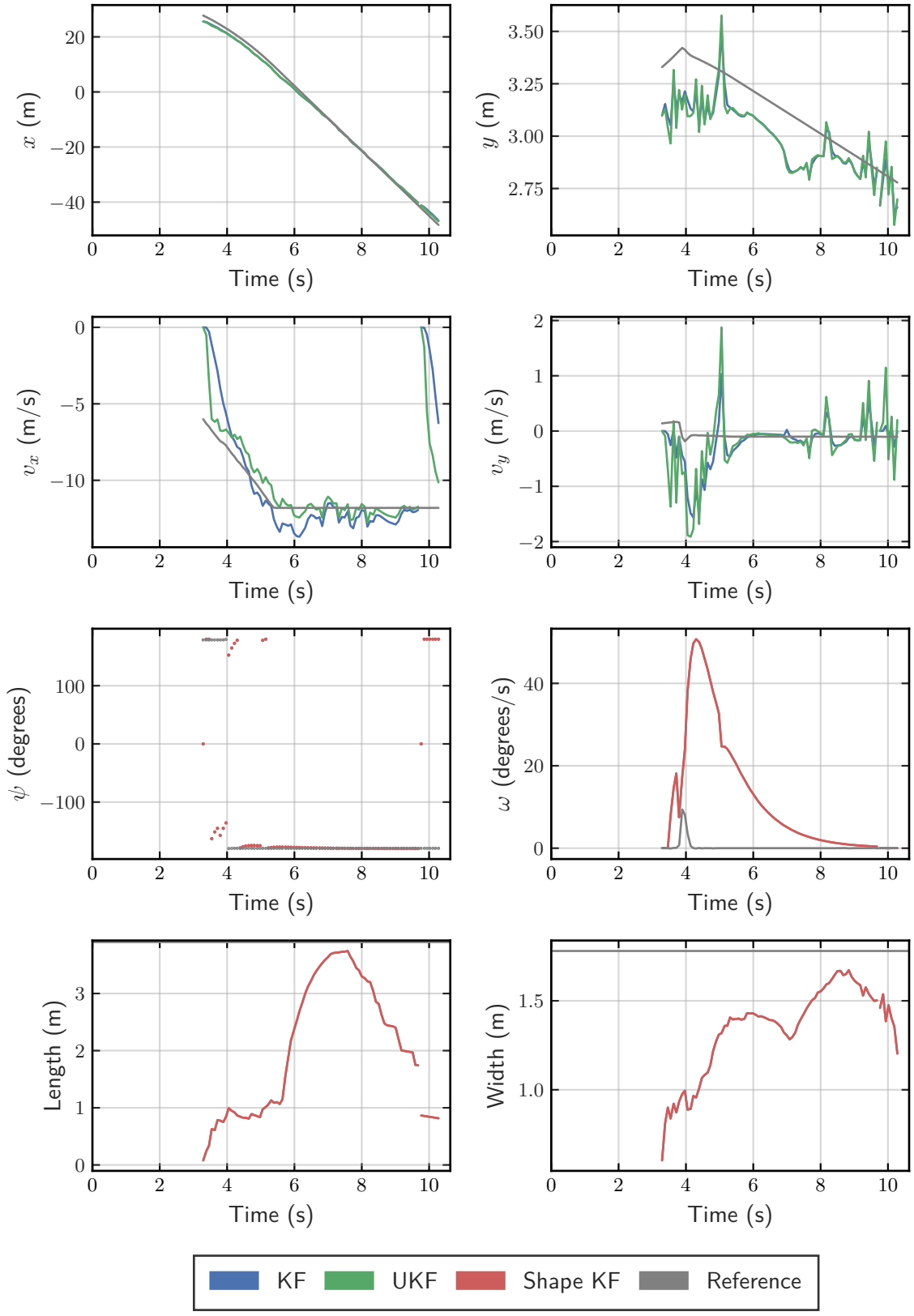


Figure 4-7: Estimated states by the developed system against the simulation ground truth, for the car at the opposite lane.

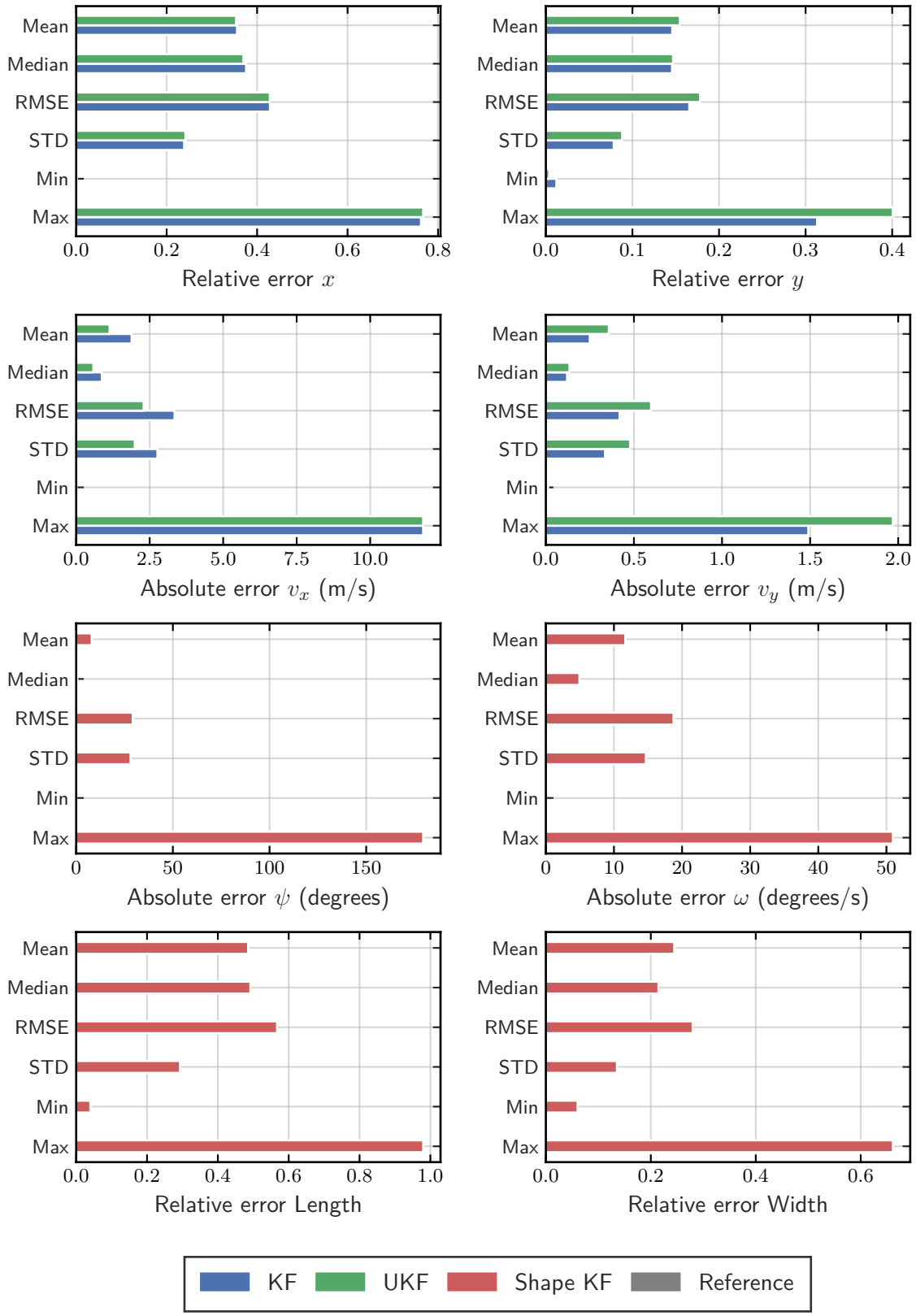


Figure 4-8: Statistics of the errors of the estimated states by the developed system against the simulation ground truth, for the car at the opposite lane.

4-2 Scaled Cars Experiments

The proposed system was additionally tested and evaluated in a experiments with scaled RC driven cars. The reference measurements provided by the Motion Capture (MoCap) system have high accuracy (2mm) and sufficiently higher frequency (120 Hz) than the estimations of the system (12 Hz). It should also be mentioned that the system produces estimates in the coordinate frame of the MoCap system, which is possible because localization measurements are provided to the ego-vehicle by the MoCap system.

4-2-1 Racing experiment

The racing experiment[4] provides a plethora of driving maneuvers in a short time frame and therefore can be used to evaluate the overall performance of the system. During this experiment the cars drove around the four cones two times, while overtaking each other. In Figure 4-12, we can observe the ego-vehicle (black car), while it is overtakes the red car.

Figure 4-9: Photograph of the racing experiment during an overtaking maneuver by the DSV.

In Figure 4-10 and 4-11, we can observe that both filters produce almost indistinguishable estimates for the position of the car center. In velocities estimation the two filters have different performance, and by looking at the statistics it can be seen that the estimates of the UKF are more accurate. Orientation estimation is accurate for the most part, except at the beginning, the middle and the end of the experiment, in which time points the tracked vehicle has zero velocity. In this experiment, the yaw rates are significant since the vehicles follow an elliptical trajectory and therefore it is the first experiment in which the turn rate tracking capabilities of the system are displayed.

Lastly, it can be observed that all state estimates are abruptly stopped before the 15 second mark and this results from the red vehicle coming closer than the minimum range of the LIDAR sensor. Since no new observations of the red car are made, it is deleted by the system and no estimates about its motion are given, until it is detected again.

[4] A video of the experiment can be found at: https://youtu.be/BDvzPGi3hjg

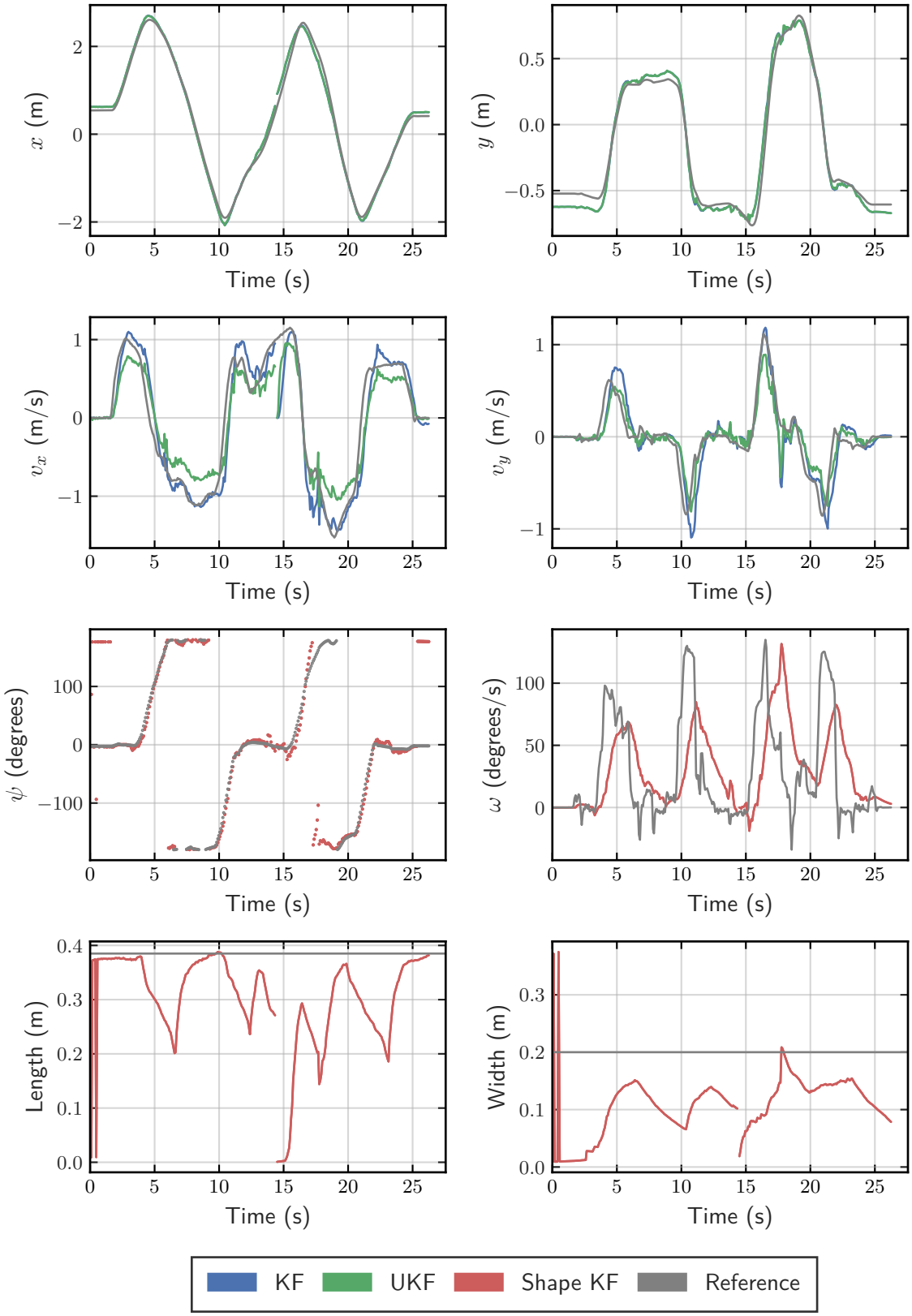


Figure 4-10: Estimated states by the developed system against the MoCap ground truth.

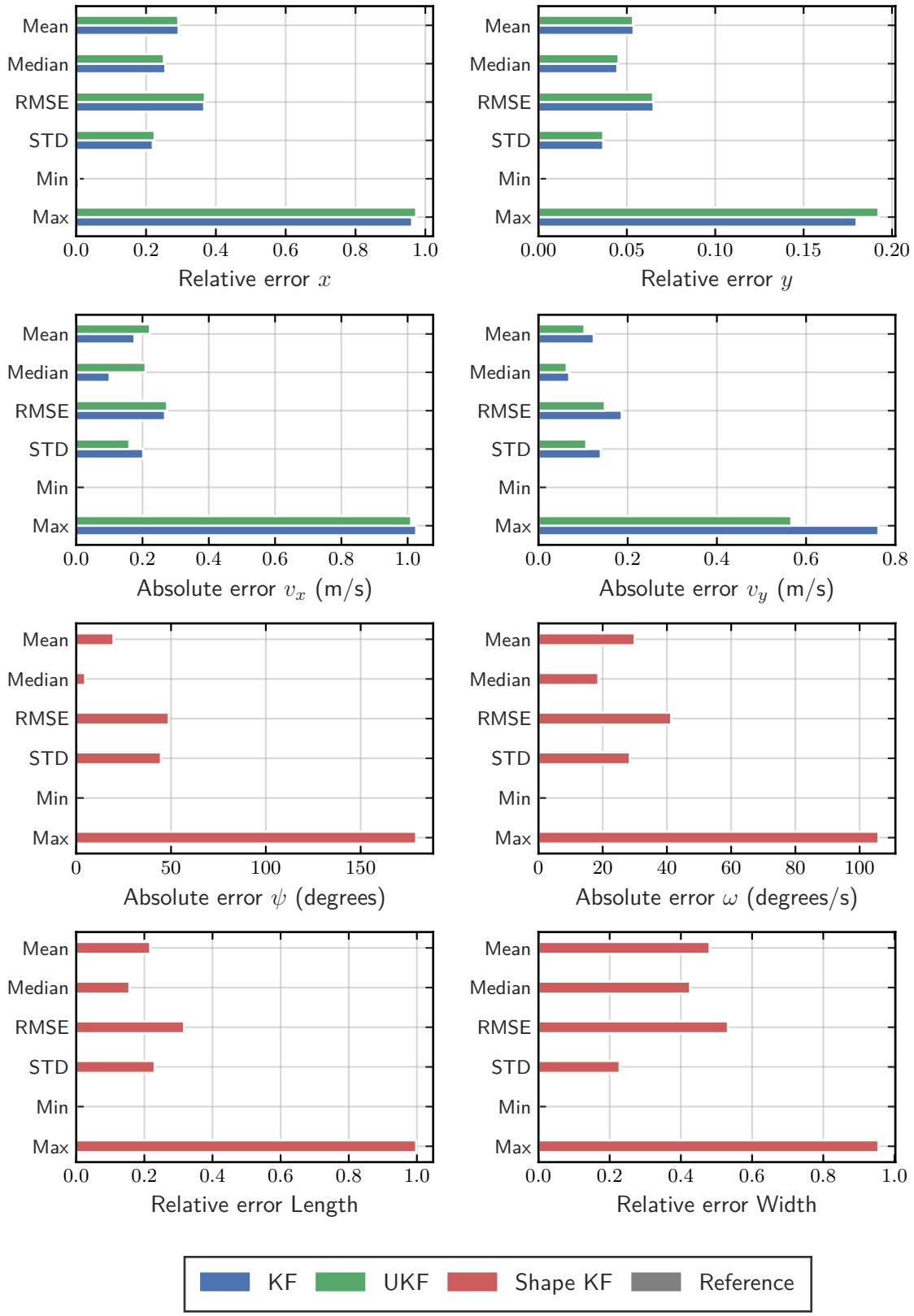


Figure 4-11: Statistical analysis of the absolute error of the two methods for each state.

4-2-2 Road Intersection

The intersection experiment[5] was the most challenging experiment for the developed system, since the target vehicle moves in a perpendicular direction and at a high speed. This experiment tries to simulate the scenario in which another car runs through a red traffic light and comes toward the ego vehicle at a high speed. In Figure 4-12, which is a photo of this experiment, we can observe the ego-vehicle moving slowly forward, while the red car moves with a high speed.

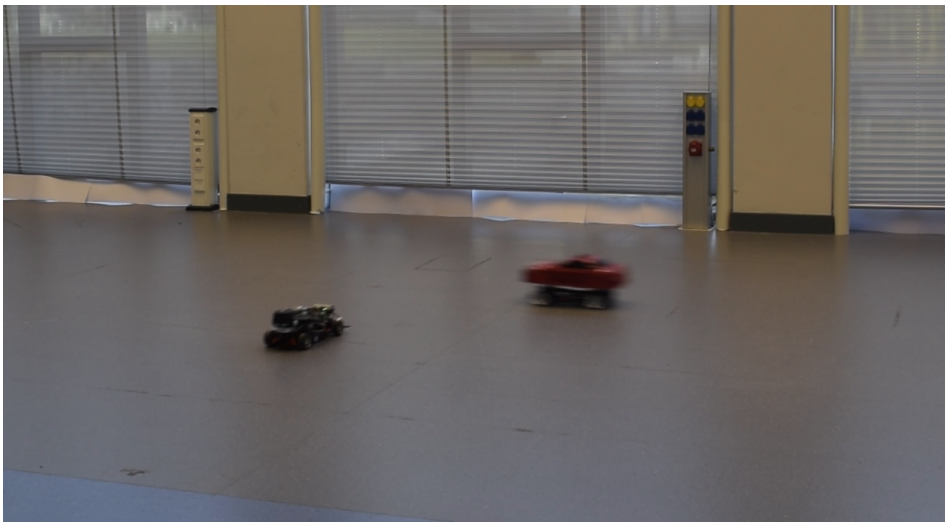


Figure 4-12: Photograph of the racing experiment during an overtaking maneuver by the DSV.

The motion of the red car is parallel to the x axis and therefore we are most interested in the estimations of x and v_x. While both filters produce the same estimates for x (Figure 4-13), the estimate of v_x by the UKF converges faster to the reference value which is very important in such scenarios. However, it is expected that this performance can be replicated in the KF by tuning the R matrix. In the orientation (ψ) and turn rate (ω) plots we can observe that the system makes a false corner point switching decision around the 3 second mark, which causes the turn rate estimate to severely diverge. The false corner point switch can also be observed in the width estimate which has a peak at the same time instance.

In summary, this is an experiment that brings the developed system to its limit and therefore it is not advised to rely on this system for similar scenarios.

[5] A video of the experiment can be found at: https://youtu.be/Xh4UryjuOTk

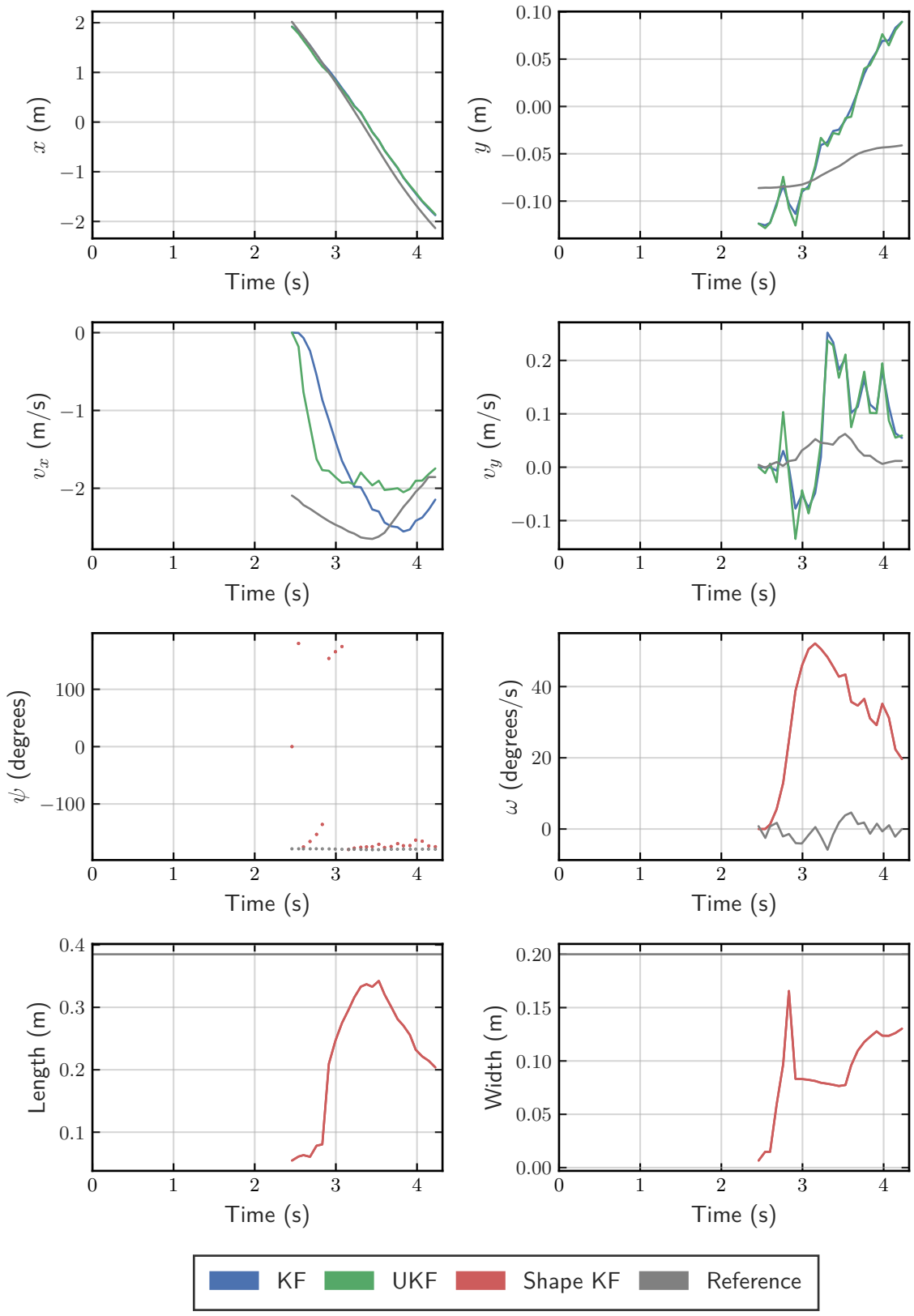


Figure 4-13: Estimated states by the developed system against the MoCap ground truth.

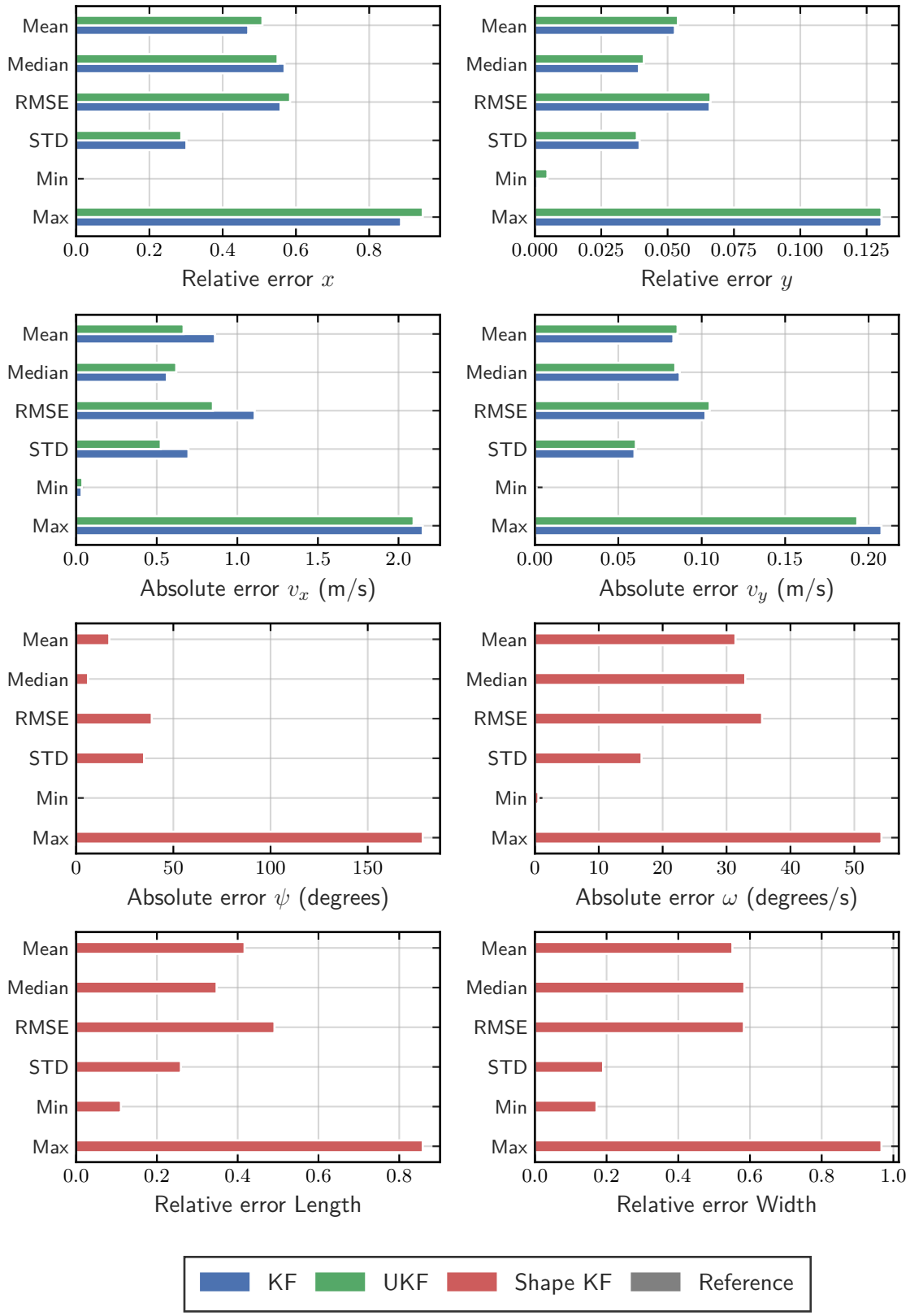


Figure 4-14: Statistical analysis of the absolute error of the two methods for each state.

4-3 Evaluation of system speed

The operating frequency of the system should be higher than the sampling frequency of the LIDAR sensor, which in our case is 12 Hz both in the simulation and the experiment. For this reason, the proposed system should be able to perform all necessary calculations with a higher frequency than 12 Hz and in a period smaller than 83 milliseconds.

Since, there are time restrictions, it was necessary for the system to be computationally efficient and therefore it was decided to develop it in C++, which has fewer computational overheads than other programming languages. However, while the computational requirements of the system grew, and especially after the inclusion of the UKF, there were instances where the system exceeded the 83 millisecond limit. A solution to this problem was given by computing all the filter calculations in parallel. This proved especially useful for the microcomputer on the DSV, which has 8 CPU cores and therefore achieved a major boost in performance.

Figure 4-15 is a graph of the execution time of the program, during the racing experiment. It can be observed that the execution time stayed below the 83 millisecond limit throughout the duration of the experiment. It should be further mentioned, that this use case is the hardest one, because both the Kalman Filter and the UKF were used.

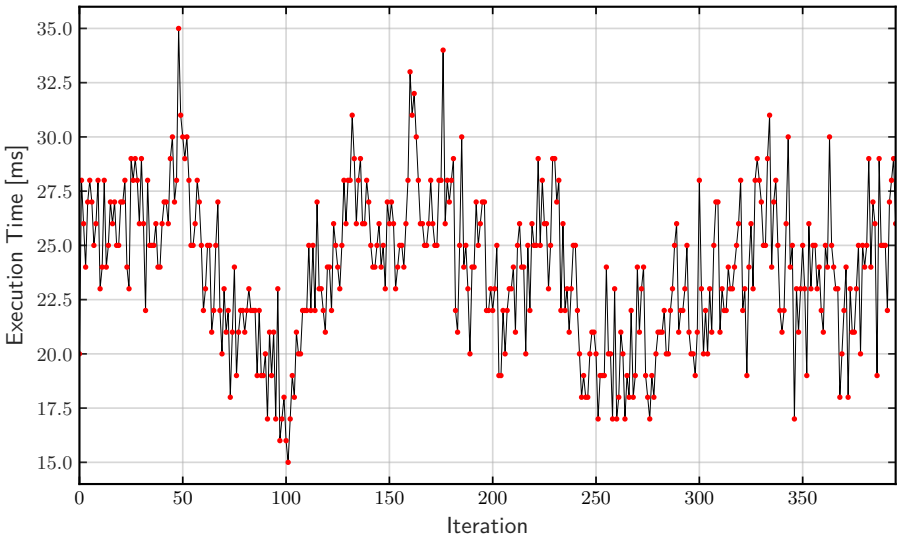


Figure 4-15: Execution time of the proposed system in the DSV during the racing experiment.

Lastly, it is worth noting that this system is developed as part of the Robot Operating System (ROS) ecosystem[6]. Therefore, it accepts LIDAR measurements with the standardized format of ROS measurements and provides estimates of the target locations also in standardized format. This communication standardization enables the system's easy exchangeability with similar ROS programs who perform the same function.

[6]The complete system code can be found at:https://github.com/kostaskonkk/datmo

Chapter 5

Conclusion and Future Work

In this chapter, the steps of the proposed solution and the algorithms used are summarized alongside the contributions of this thesis. Additionally, in the future work section, some ideas about potential additions and improvements to the system are provided.

5-1 Conclusion

The increasing effort of developing robust systems for autonomous vehicles has created a need for low cost test and experimental platforms, in which new and prototype algorithms can be fast and safely evaluated and tested. Algorithms, for platooning, collision avoidance and path planning, are crucially dependent on an accurate representation of the surrounding environment and especially the moving vehicles inside it. Therefore, an efficient and modular software system was developed, which can be used in experimental platforms with 2D LIDAR sensors for detection and tracking of multiple moving vehicles.

The developed system encompasses the complete process of multi-object tracking; it receives 2D LIDAR measurements as input and estimates the kinematic state and dimensions of surrounding vehicles. The detection part of the system differentiates objects in the LIDAR measurements using the Adaptive Breakpoint Detector algorithm and calculates rectangles around the detected objects with the Search Based Rectangle Fitting algorithm. The tracking part of the system receives L-shapes and associates them with already tracked vehicles, using the Global Nearest Neighbour algorithm. It estimates kinematic poses with two different tracking filters, a Kalman filter, with a constant velocity model and a UKF, with a nonlinear constant turn kinematic model, while the dimensions of neighboring vehicles are estimated using a constant shape Kalman filter.

The proposed system was evaluated using both simulation and real-world experiments. Analyzing the experiment results reveals that the system can reliably estimate the position, speed, orientation and dimensions of surrounding vehicles in normal driving conditions. Since it is able to run in a real-time manner in a single board computer, it is expected to be applicable for a variety of algorithm development projects using low-cost platforms.

5-2 Future Work

There are several improvements and additions to the proposed system that can be implemented to increase its performance and robustness. The ones whose application is expected to be the most significant are presented below:

1. The system can be expanded to incorporate measurements from camera and radar sensors. Camera images can be used in conjuction with object recognition algorithms, so that an estimation about the class of every detected object can be made. Furthermore, radar sensors can be used to improve the kinematic pose estimation, since they provide accurate velocity measurements, which can be fused with the LIDAR measurements.

2. The data association capabilities of the system can be expanded by implementing the JPDAF algorithm which is proven to perform better in highly cluttered environments [26].

3. The computational requirements of the existing system can be reduced by implementing a potentially more computationally efficient L-shape fitting algorithm, such as the one presented in [35].

4. The L-shape model used, assumes that the vehicle shape is a rectangle, however actual vehicle corners are typically smoothly curved. Therefore, it can be researched if a U-shape extraction algorithm [36] and consequently U-shape model achieve higher accuracy tracking results.

5. In addition to vehicle detection via rectangle extraction, the system can be expanded to detect pedestrians and bikes with the inclusion of a circle extraction algorithm.

6. The accuracy of the existing system can be potentially improved by calculating the propagation of covariance matrices during corner point switching and box model estimation.

7. The state estimation error can be potentially reduced by employing an Interactive Multiple Model (IMM) approach. In this way, multiple filters with different models can be used to estimate the motion of the target at the same time.

8. Another possible expansion of the system would be the inclusion of an Autocovariance Least-Squares (ALS) implementation to estimate the noise covariance matrixes Q and R, in case that reliable ground truth data are available.

9. Lastly, the system can be expanded to use a map of the surrounding environment, generated for example by a SLAM algorithm, to reduce the search area for new moving objects.

Appendix A

Localization

During my thesis work there was a possibility that the students working on the DSV would showcase their work in a research institution. This work of the other students included parking assist, planning of lane change maneuvers and vehicle platooning. The last two of which were depended on the work of my thesis, for estimating the position and speed of vehicles in the other lanes and in the platoon respectively. The only obstacle was that everybody had developed their systems to depend on accurate localization information. In lab experiments, localization was provided by the Motion Capture (MoCap) system which is not transportable and therefore, a need arose for the development of a localization system for the Delft Scaled Vehicle (DSV), that is not dependent on external sensors.

The problem of vehicle localization, is the problem of estimating vehicle position and orientation relative to the environment. A common method of obtaining localization information is through the use of GPS sensors. However, in this case it would not be a fitting solution, since the DSV is mainly used indoors, where there is no GPS signal and in addition, the accuracy of GPS, which has a reported variance of more than one meter, is inadequate for the size of our vehicle, which is about 30 centimeters long. Since, GPS was out of the picture a different method had to be found.

Available information that can be used for localization, are the sensor measurements (wheel encoders, IMU, LIDAR) and the control inputs given to the vehicle. It was chosen to only use sensor information and not the available control inputs, so that the developed localization system could be used easily by all the students without having to modify the localization system to interpret their control signals. Since, all sensors are imperfect, and their measurements are prone to errors, better overall position estimation was obtained by fusing the measurements from all the available sensors.

Finally, the developed localization system was evaluated, during an experiment, in which the DSV and two other vehicles were moving inside the area surveyed by the MoCap. In this way, the location estimation produced by the proposed localization system could be compared with the high precision localization information of the MoCap system and subsequently the accuracy of the system could be evaluated.

A-1 Localization by Fusion of Odometry and IMU

It was previously stated that for localizing a vehicle its position and orientation has to be estimated. In state vector this is translated to the estimation of the following states.

$$x_{localization} = \begin{bmatrix} x & y & \psi & v_x & v_y & \omega \end{bmatrix}^T, \tag{A-1}$$

where x and y represent the position of the vehicle in the x and y axis respectively and ψ is the orientation/yaw of the vehicle. The other three states are the derivatives of the first three states, so v_x, v_y are the velocities on the x and y axis and ω is the angular velocity of the vehicle.

The wheel encoders installed on all four wheels of the DSV count the rotations of the wheels and produce measurements of the wheels angular velocities. The wheel angular velocities can be in turn used to calculate the speed of the vehicle (v) and if the angle of the steering wheel (δ) is also known then the velocities in the x and y direction can be calculated.

The IMU sensor measures the orientation of the DSV (ψ) using a magnetometer and estimates the angular velocity (ω) by integrating the rotational acceleration.

Therefore, an estimate of the state vector can be calculated by fusing the above discussed sensor measurements and integrating the velocities.

A-1-1 Velocities calculation via Odometry measurements

The velocities (v_x, v_y) of the vehicle are estimated by using the wheel speed measurements and the steering angle (δ).

The speed of the DSV was calculated via the angular velocity (ω) of the front left (ω_{fl}) and right wheel (ω_{fr}). The angular velocities of the back wheels were discarded, because the DSV is real wheel driven and thus the rear wheels are prone to slippage.

$$v = R_e \frac{\omega_{fl} + \omega_{fr}}{2}, \tag{A-2}$$

$$\tag{A-3}$$

where R_e is the effective wheel radius, which was measured equal to 0.0313m.

The longitudinal (v_x) and lateral (v_y) velocities were calculated in the following way:

$$v_x = v \cos \delta \tag{A-4}$$

$$v_y = v \sin \delta \tag{A-5}$$

A-1-2 Fusion of Odometry and IMU measurements

The measurements produced by the odometry sensors and the measurements of the IMU, were fused using an EKF with an omnidirectional motion model to produce an estimate of the localization state vector.

The Inertial Measurement Unit (IMU) measurements have a data frequency of 100 Hz and the wheel encoders 30Hz. The Extended Kalman Filter (EKF) formulation and algorithm were presented in Section 3-2-2 and therefore now only the kinematic and measurement function will be presented. For our application, the kinematic function f is an omnidirectional 2D kinematic model derived from Newtonian mechanics.

$$f(x_{localization}) = \begin{bmatrix} x + (v_x \cos\psi - v_y \sin\psi)\, d_t \\ y + (v_x \sin\psi + v_y \cos\psi)\, d_t \\ \psi + \omega d_t \\ v_x \\ v_y \\ \omega \end{bmatrix} \tag{A-6}$$

The standard EKF formulation specifies that H should be the Jacobian matrix of the measurement model function h. Since different sensors are simultaneously used, the developed localization system should be capable of partial updates of the state vector. Specifically, when measuring only m variables, Z becomes an m by 6 matrix of rank m, with its only nonzero values existing in the columns of the measured variables. This is important for using sensor measurements that does not always measure the same states of the state vector. The EKF presented in [33], which provides the above functionality, was used for fusing the data. Additionally, given the assumption that each sensor produces direct measurements of the state variables, the nonzero values in H are ones.

Consequently, the odometry sensors produce measurements of the form:

$$z_{odometry} = \begin{bmatrix} v_x & v_y \end{bmatrix}^T \tag{A-7}$$

and the IMU produces measurements of the form:

$$z_{IMU} = \begin{bmatrix} \psi & \omega \end{bmatrix}^T \tag{A-8}$$

The results of fusing the odometry (30 Hz) and IMU (100 Hz) measurements using the previously described model in an EKF, are plotted in Figure A-1, with a red line. In the same Figure, the measurement of the MoCap system are plotted with a gray line, so a direct comparison between the two can be made.

The experiment that was used to test the accuracy of the localization system, and the one that the plotted data comes from, was a scenario in which the DSV and two other vehicles were moving in circles under the MoCap system.

It should be noted, that the data could not be directly plotted, since they are in different coordinate frames. The procedure used for calculating the translation and rotation between the two coordinate frames, will be explained in Section A-3-1.

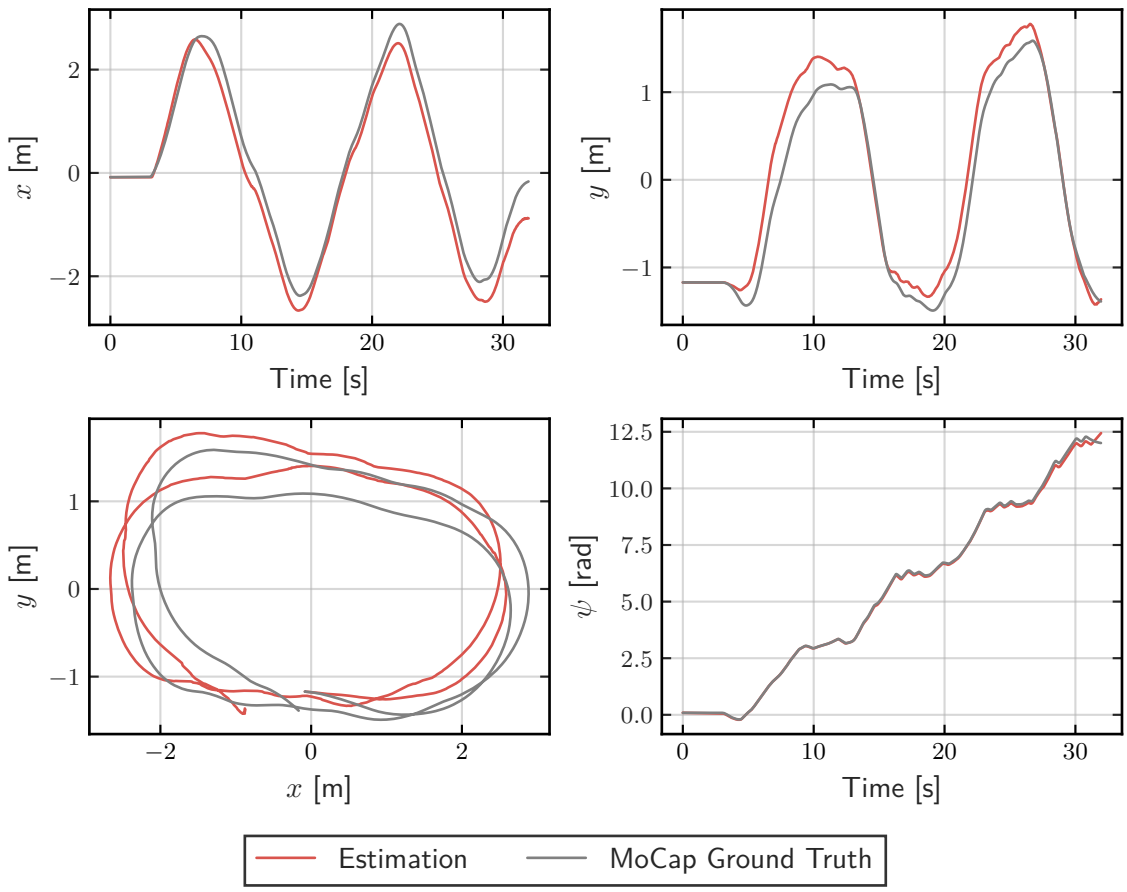


Figure A-1: Localization results by fusing wheel encoder and IMU data.

Although the results are adequately accurate for short experiments, like the 30 second one presented, the dead reckoning process that is used, is subject to cumulative errors because of the velocity integration that is used to calculate the position states. This problem can only be solved by fusing measurements of the x and y states.

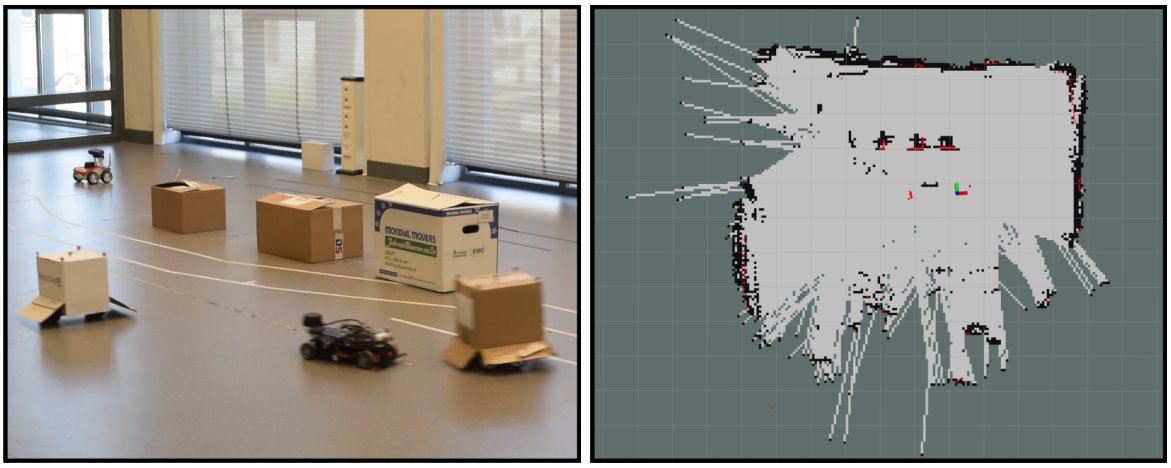
A-2 Self Localization and Mapping (SLAM)

One method for acquiring measurements of the vehicle pose (x, y, ψ) is through the use of a SLAM algorithm for 2D LIDAR sensors. Since SLAM is a deeply researched subject in the robotics community, there are multiple approaches to the problem, with many of them implemented as ROS packages. An up to date summary of the available methods can be found in [37] and through this summary a few of them were selected to be tested on the DSV.

The main criterion considered in the selection process, was the ability to perform frequent loop closures, which is required because of the restricted spaces that the DSV is usually used in. Additionally, the computational resources used by all the tested systems were monitored and the ones with low computational requirements were heavily favored. After the testing of three different packages (Cartographer, GMapping, Hector SLAM), it was found that

Hector SLAM [38] was the best suited one for the application at hand. Although GMapping has the lowest computational requirements, it does not take advantage of any prior location information and therefore it is very susceptible to errors when few LIDAR measurements are available. Cartographer, was the most computationally expensive of the three and in addition it was incapable of detecting the very frequent loop closures resulting from the DSV moving in the same area. As the name suggests, it is especially designed for building maps.

Although, the Hector SLAM algorithm can work without previous location estimations, higher accuracy can be achieved when an estimation of the movement of vehicle between two LIDAR measurement instances is also provided. In the developed system, the SLAM algorithm was given the localization estimation of the previously presented EKF as additional input. The output of the SLAM algorithm is a map of the environment and an estimation of the position and orientation states of the vehicle.



(a) Image of the experiment. **(b)** Map built by the SLAM algorithm.

Figure A-2: Image of the localization experiment and map of the DCSC lab built by the SLAM algorithm.

In Figure A-2a, we can see the DSV and the two other vehicles, which are covered with carton boxes to facilitate their detection by the LIDAR sensor. All three vehicles, were moving around the three carton boxes in the center, while keeping their in between distances and order unchanged.

In Figure A-2b, the map built by the SLAM algorithm after a complete rotation around the carton boxes is visualized. In this picture, the DSV by the red-green-blue axes, and the LIDAR data at the instance when the picture is taken are represented as red dots. It is interesting to observe that there are cases in which the two vehicles covered by cartboxes became erroneously part of the map and therefore there are black dots in unoccoupied areas. This error is the reason, that there moving objects in the localization experiment, to test for the robustness of the system in moving objects. Additionally, it is interesting to observe that since the laser rays emitted by the LIDAR are not reflected by the glass on the left side of the lab, the map is not properly built for that area.

In addition to the map of the environment, the SLAM algorithm produces a localization estimations for the position and orientation of the ego-vehicle.

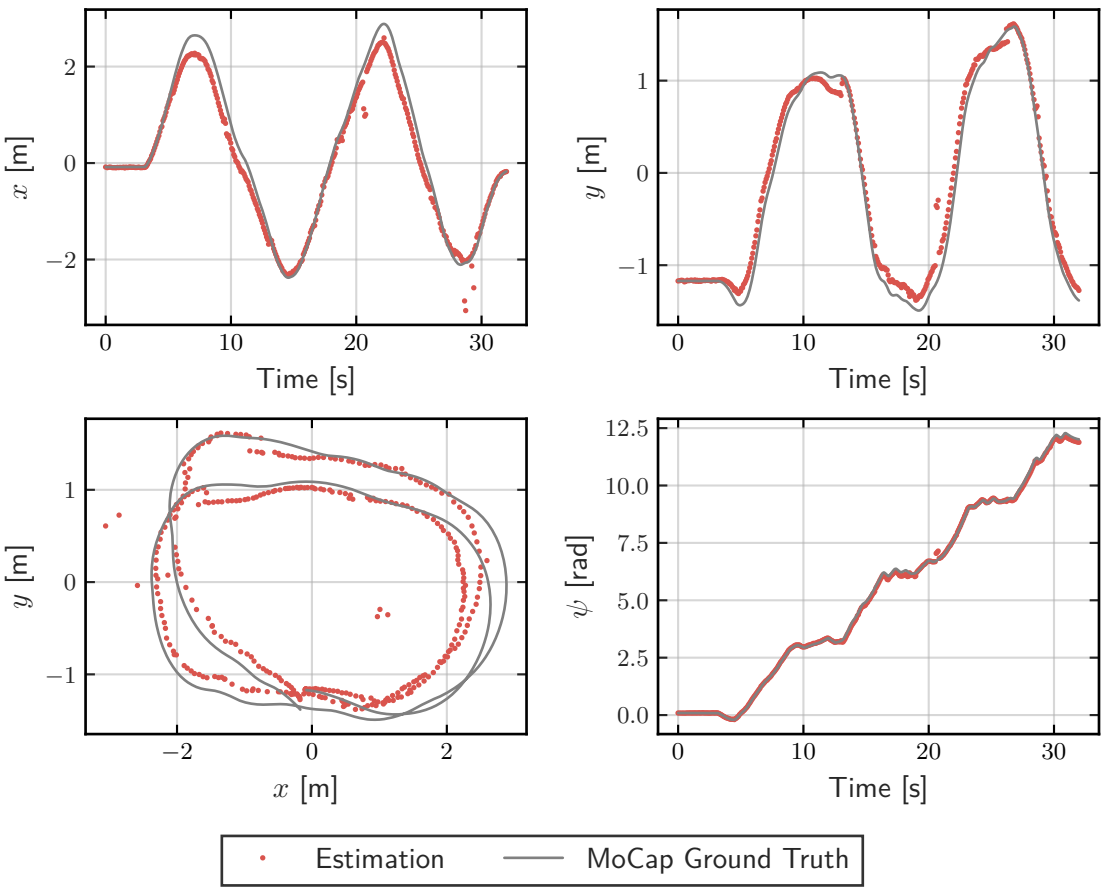


Figure A-3: Localization output of the SLAM algorithm.

In Figure A-3 the localization output of the SLAM algorithm is compared with the reference measurements of the MoCap system. The output of the algorithm is visualized with points to emphasize the fact that the SLAM algorithm provides localization estimations with a frequency equal to the frequency that data is provided by the LIght Detection And Ranging (LIDAR) sensor ($\simeq$ 12 Hz).

Observing the top left plot in Figure A-3 we can find two instances, one after the 20 second mark and another before the 30 second mark, in which the SLAM algorithm fails to make an accurate estimation of the position of the vehicle. Therefore, it was concluded that the SLAM algorithm alone, is not capable of providing the required localization accuracy, since estimation jumps like these ones can negative influence the operation of the algorithms that depend on it. However, it should be noted, that the SLAM algorithm not only provides the measurements of the states but also a covariance matrix that describes the accuracy of its output.

The solution that was given to the SLAM algorithm failures will be explored in the next Section.

A-3 Fusion of Odometry, IMU and SLAM

A solution to the problem of SLAM algorithm failures, was given by fusing the output of the SLAM algorithm with the odometry and IMU measurements. Since the SLAM package provides not only positions but also uncertainty, by fusing the output of the SLAM algorithm with the odometry and IMU measurements, it is possible to achieve higher localization accuracy.

In addition to the map of the environment the SLAM algorithm produces localization estimations for the pose and orientation of the ego-vehicle. These estimations are given as measurements to another EKF as measurements of the form:

$$\begin{aligned} z_{Odometry} &= \begin{bmatrix} v_x & v_y \end{bmatrix}^T \\ z_{IMU} &= \begin{bmatrix} \psi & \omega \end{bmatrix}^T \\ z_{SLAM} &= \begin{bmatrix} x & y & \psi \end{bmatrix}^T \end{aligned} \tag{A-9}$$

In Figure A-4 the output of the EKF (30 Hz) is compared with the reference values provided by the MoCap.

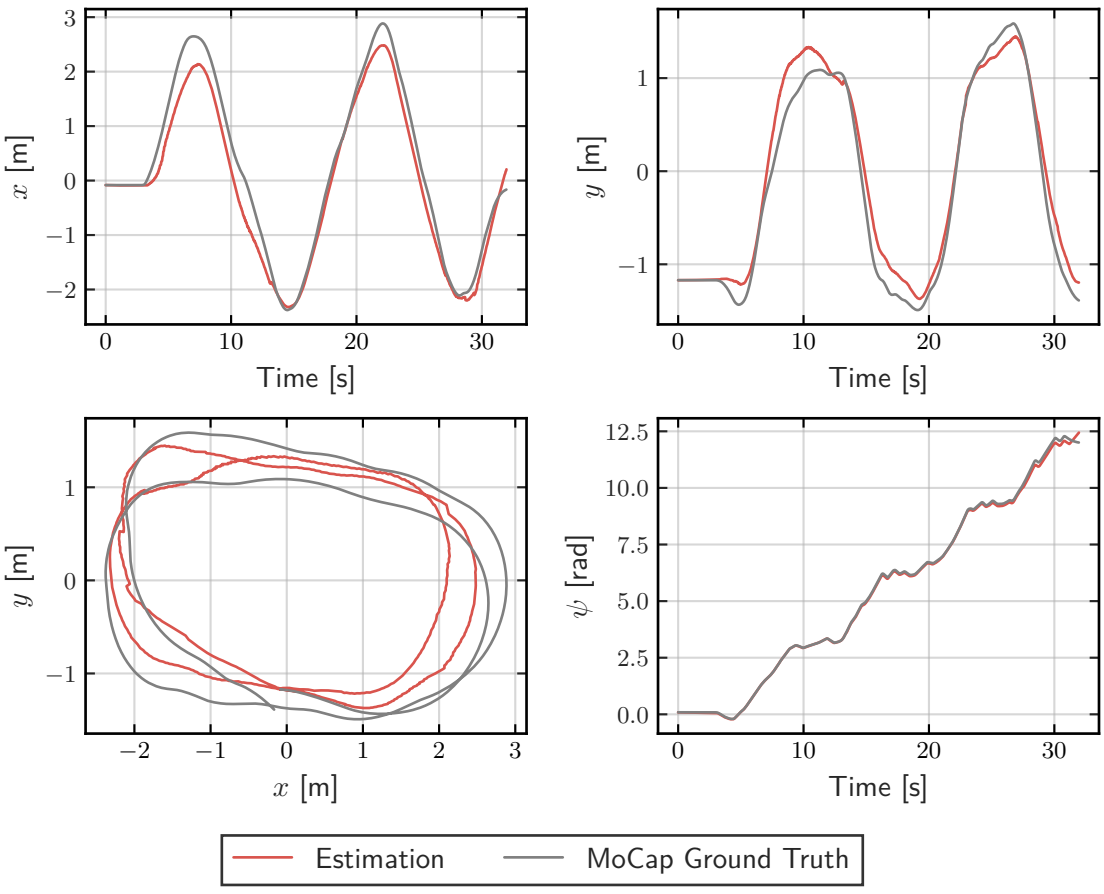


Figure A-4: Localization results by fusing odometry and SLAM.

Method	RMSE	Mean	Median	STD	Min	Max	SSE
Odometry+IMU	0.435	0.384	0.429	0.203	0.0	0.723	181.277
SLAM	0.33	0.261	0.201	0.202	0.0	1.396	42.114
Fusion	0.403	0.345	0.345	0.209	0.0	0.768	156.054

Table A-1: Comparison of the accuracy of the proposed localization methods.

In Table A-1 the three different methods for estimating the position of the DSV are compared by calculating statistics for the errors that they produce. Comparing the Odometry-IMU and the Fusion rows, it is evident that the performance of the localization system improved by fusing the SLAM measurements, since the Root Mean Squared Error (RMSE) is lower when all measurements are fused. Although, the RMSE is the lowest for the SLAM algorithm, we cannot directy use its output since its Max error value is almost double than the other two methods.

A-3-1 Transformation between different coordinate frames

The transformation parameters between the trajectories of the MoCap system and the estimated one of my system were calculated with the least-squares estimation of transformation parameters between two point patterns [39].

The trajectories based on the MoCap system and the one produced by the proposed system were first cropped so both of them contain values in the same range.

Given two point patterns (set of points) $\{x_i\}$ and $\{y_i\}\,; i = 1, 2, \ldots, n$ in m-dimensional space, and we want to find the similarity transformation parameters (R: rotation, t: translation) giving the minimum value of the mean squared error $e^2(R, t)$ of these two point patterns. The dimensionality m is equal to 2 for pose and equal to 1 for the orientation.

$$e^2(R, t) = \frac{1}{n} \sum_{i=1}^{n} \|y_i - (Rx_i + t)\|^2 \tag{A-10}$$

Theorem: Let $X = \{x_1, x_2, \ldots, x_n\}$ and $Y = \{y_1, y_2, \ldots, y_n\}$ be corresponding point patterns in m-dimensional space. The minimum value ϵ^2 of the mean squared error

$$e^2(R, t) = \frac{1}{n} \sum_{i=1}^{n} \|y_i - (Rx_i + t)\|^2 \tag{A-11}$$

of these two point patterns with respect to the similarity transformation parameters (R: rotation, t: translation) is given as follows:

$$\varepsilon^2 = \sigma_y^2 - \frac{\mathrm{tr}(DS)^2}{\sigma_x^2} \tag{A-12}$$

where

$$\begin{aligned}
\mu_x &= \frac{1}{n}\sum_{i=1}^{n} x_i \\
\mu_y &= \frac{1}{n}\sum_{i=1}^{n} y_i \\
\sigma_x^2 &= \frac{1}{n}\sum_{i=1}^{n} \|x_i - \mu_x\|^2 \\
\sigma_y^2 &= \frac{1}{n}\sum_{i=1}^{n} \|y_i - \mu_y\|^2 \\
\Sigma_{xy} &= \frac{1}{n}\sum_{i=1}^{n} (y_i - \mu_y)(x_i - \mu_x)^T
\end{aligned} \tag{A-13}$$

and let a singular value decomposition of Σ_{xy} be $UDV^T (D = \operatorname{diag}(d_i), d_1 \geq d_2 \geq \cdots \geq d_m \geq 0)$, and

$$S = \begin{cases} I & \text{if} \det(\Sigma_{xy}) \geq 0 \\ \operatorname{diag}(1, 1, \ldots, 1, -1) & \text{if} \det(\Sigma_{xy}) < 0 \end{cases} \tag{A-14}$$

where $(\operatorname{rank}(\Sigma_{xy}) = m - 1)$.

Appendix B

Paper

Development of a Detection and Tracking of Moving Vehicles system for 2D LIDAR sensors

Konstantinos Konstantinidis, Mohsen Alirezaei, Sergio Grammatico

***Abstract*—This paper presents the development and evaluation of a Detection and Tracking of Moving Objects (DATMO) system, used for tracking nearby vehicles from a moving car. The developed system takes in raw 2D LIght Detection And Ranging (LIDAR) measurements as input and detects objects of interest by clustering them with the Adaptive Breakpoint Detector algorithm. The resulting clusters are fitted with oriented bounding boxes, by incorporating the Search-Based Rectangle Fitting algorithm. The tracking part of the system receives, extracted from the rectangles, L-shapes and associates them with already tracked vehicles using the Global Nearest Neighbor (GNN) algorithm. However, since LIDAR measures only the distance to surfaces that face the sensor, vehicle appearances change over time. In order to counteract tracking errors that originate from these changes, an L-shape switching algorithm is implemented. The kinematic poses of the tracked vehicles are estimated with two different tracking filters, a Kalman Filter, with a constant velocity model and an Unscented Kalman Filter (KF), with a Coordinated Turn model. The proposed system was evaluated in a simulation environment and the tests revealed that it can reliably estimate the position, speed, heading angle and dimensions of surrounding vehicles. Therefore, it can be reliably used in other research platforms to expand their environment perception capabilities.**

***Index Terms*—Detection, Tracking, DATMO, 2D LIDAR, ROS, Autonomous Vehicles.**

I. Introduction

In recent years, a lot of researchers have focused their efforts on providing solutions to the various problems that need to be addressed before vehicles can reliably perceive their surrounding environment and have divided environmental perception in three different tasks [1]. The first is localization, in which the vehicle localizes itself in the environment by establishing the spatial relationships between itself and stationary objects. The second one is mapping, which builds a map of the environment by establishing the spatial relationships between surrounding static objects. And the last task and the one that the developed system focuses on, is Detection and Tracking of Moving Objects (DATMO), which establishes the spatial and temporal relationships between the vehicle and moving objects. Therefore the aim of the proposed system is, given as input LIght Detection And Ranging (LIDAR) measurements to detect the surrounding moving vehicles and estimate their position (x, y), velocity (v_x, v_y), orientation (ψ), turn rate (ω) and dimensions (Length, Width).

DATMO systems for LIDAR sensors are designed based on three main approaches, the traditional, the model based and the grid based one [2]. The traditional approach, first divides the incoming sensor measurements into clusters and then associates them with objects from previous time instances. In more advanced systems, the clusters are fitted with geometric shapes whose center is then tracked with a parametric Bayesian filter [3], otherwise the geometric mean of each cluster is tracked [4]. The model based approach fits the sensor data directly onto geometric shape models by utilizing particle filters, which also handle data association [5], [6], [7]. Lastly, the grid based approach [8] is based around the construction and use of an occupancy grid, which models the space around the vehicle. The grid cells are then tracked using a Bayesian filter and in some systems, additional object level representations are fitted on top of the grid cells [9]. The development of the proposed system is based on the traditional approach, since it is the most modular and less computationally demanding of the three.

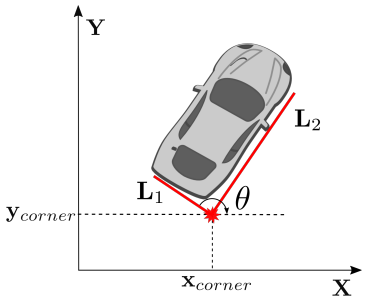
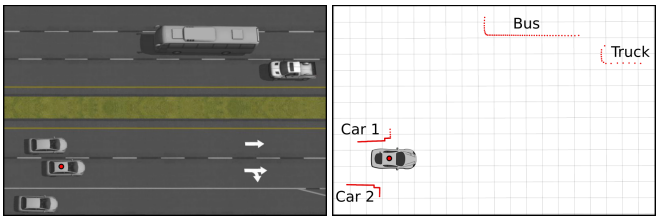


Fig. 1. The five measurements in an L-shape.

The operation of the developed system is based around extracting and tracking L-shapes, which are used to represent the surrounding vehicles (Fig. 1). L-shapes are defined based on five values, the position of their corner (x_{corner}, y_{corner}), their orientation (θ) and the lengths of their sides (L_1, L_2).

II. Detection of Moving Objects

The main goal of the detection stage is to differentiate moving objects from the LIDAR sensor measurements. LIDAR sensors calculate distance to neighboring objects by emitting a laser beam, capturing its reflection and calculating the distance by measuring the time of flight. The measurements of a LIDAR sensor can be better understood by examining Fig. 2. On the left, there is a screenshot from a simulation and on the right the resulting LIDAR measurements from a sensor on top of the ego vehicle. It should be noted, that this specific time instance will be used throughout this paper to explain and visualize the operation of the developed system.



(a) (b)

Fig. 2. Example of LIDAR data acquisition; (a) image of the simulation environment, (b) visual representation of the LIDAR data acquired at the same time instance.

The first step of the detection stage (Fig. 3) is a segmentation algorithm, which extracts clusters of LIDAR points from the raw LIDAR measurements. These clusters are then passed to a feature extraction algorithm, which extracts geometric shapes from the clusters. Common extracted geometric shapes are lines or rectangles for vehicles, circles for pedestrians and ellipses for bicycles and bikes [10]. Since the focus of the developed system is vehicle tracking, rectangles are fitted onto the clusters. Lastly, L-shapes are derived from the closest corner of every rectangle and those are passed to the tracking stage of the system, which will be presented in the next section.

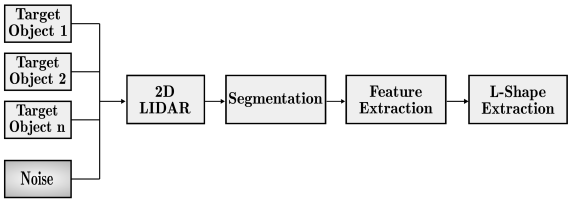


Fig. 3. Flowchart of the vehicle detection stage of the developed system.

A. Segmentation

The segmentation process is responsible for separating the raw LIDAR measurements (Fig. 2b) in groups that correspond to moving objects of the real world that need to be tracked. The algorithm used is the Adaptive Breakpoint Detector Algorithm [11], which clusters the 2D LIDAR point cloud of n points, $X \in R^{n\times 2}$, based on the euclidean distance between consecutive points. Consecutive points p_n and p_{n-1} are clustered together if their euclidean distance is lower than a predefined threshold distance $D_{\max}$.

$$\|p_n - p_{n-1}\| > D_{\max} \tag{1}$$

Otherwise, in case that (1) holds, a new cluster is started whose first point is p_n. In Fig. 4, we can see the threshold circle, which gets drawn around p_{n-1}, with a radius that equals to $D_{\max}$. In this diagram, the next point (p_n) is within the circle and the two points are clustered together. However, if the threshold distance $D_{\max}$ is fixed, the algorithm does not account for the fact that LIDAR point clouds become sparser

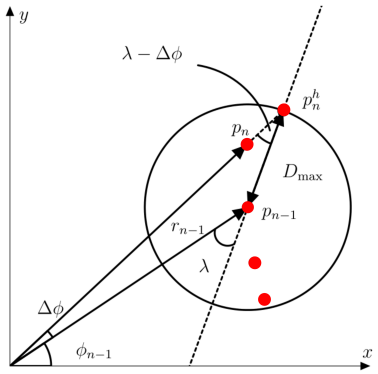


Fig. 4. Visualization of the Adaptive Breakpoint Detector Algorithm [12].

as the distance from the sensor increases. A way to overcome this limitation is by adapting the threshold distance $(D_{\max})$, according to the range distance r_n of the examined point. This is accomplished by drawing a line through the range point p_{n-1}, which represents the worst case for an incidence angle of a real world object that can be detected by the sensor. This line creates an angle λ with respect to the scanning angle ϕ_{n-1}. The maximum range distance r_n^h, for p_{n-1}, is calculated in the following way:

$$r_{n-1} \cdot \sin(\lambda) = r_n^h \cdot \sin(\lambda - \Delta\phi) \tag{2}$$

By reworking the equation above, $\|p_n^h - p_{n-1}\|$ is calculated, which can be used as a threshold distance $(D_{\max})$ in (1).

$$\|p_n^h - p_{n-1}\| = r_{n-1} \cdot \frac{\sin(\Delta\phi)}{\sin(\lambda - \Delta\phi)} \tag{3}$$

Lastly, because the sensor noise is not taken into account, problems can arise when the range distance is small. Therefore, the sensor error variance σ_r is added to the max distance

$$D_{\max} = \|p_n^h - p_{n-1}\| + \sigma_r. \tag{4}$$

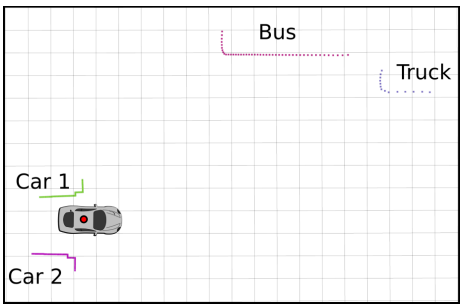


Fig. 5. Cluster segmentation with the Adaptive Breakpoint Detector algorithm.

In Fig. 5 the LIDAR points after the application of this algorithm are visualized and are drawn with a different color for every segmented cluster. We can observe that the algorithm created four different clusters which correspond accurately to the four surrounding vehicles of the simulation.

B. Feature Extraction

The purpose of the Feature Extraction process is to extract geometric shapes from the clustered points, which in this case are rectangles. The algorithm that was chosen and implemented for rectangle extraction is the Search-Based Rectangle Fitting algorithm [13], whose basic idea is to iterate through all possible directions and at each one; find a rectangle that contains all the LIDAR scan points. Afterwards, a performance score is calculated for each rectangle and the rectangle with the highest score is chosen as the best fitting rectangle.

The input of the algorithm is the n points of the examined cluster, $X \in R^{n\times 2}$, while its output are the line representations of the four edges of the fitted rectangle. The search space for θ ranges from 0° to 90°, because the two sides of a rectangle are orthogonal, and therefore only one edge needs to be calculated, since the other is $\theta + \pi/2$. In Fig. 6 an example

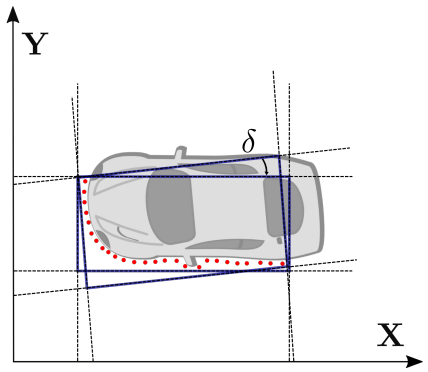


Fig. 6. Operation of the Search-Based Rectangle Fitting Algorithm.

of the algorithm's iterative nature is visualized, in which two rectangles that differ between them by an angle δ are fitted on the LIDAR range points. Although, both rectangles contain all the measurement points, one rectangle is better than the other at representing the vehicle that the points originated from and this is calculated by the performance score. The performance score used in this implementation is the point-to-edges closeness maximization criterion, which calculates how close the rectangle edges are to the LIDAR points.

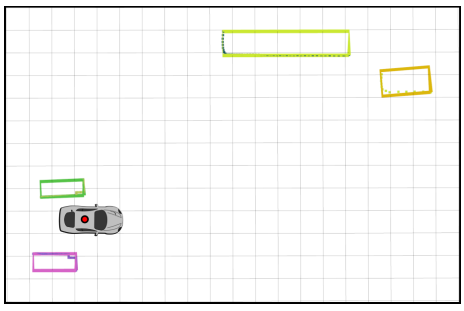


Fig. 7. Rectangle fitting with the Search-Based Rectangle Fitting algorithm.

In Fig. 7 the results of rectangle fitting in the developed system are visualized. It can be seen that the algorithm estimates the shape of the vehicles close to the sensor with high accuracy, but produces some error in the orientation estimation of the truck (orange rectangle).

C. L-shape Extraction

After every cluster of LIDAR points is fitted with a rectangle, an L-shape feature is extracted from every rectangle, mainly for two reasons. First, the information about the closest corner of a neighboring vehicle is important for collision avoidance systems and secondly by extracting L-shapes of the closest sides of neighboring vehicles, their appearance changes can be mitigated in later stages of the developed system.

L-shapes are extracted from the bounding rectangles by choosing as L-shape corner point, the corner point that is closest to the sensor. The two bounding box edges that connect to the corner point are named L_1 and L_2, by following a clockwise assignment convention, shown in Fig. 8. The orientation angle (θ) of the L-shape is defined as the orientation of L_1.

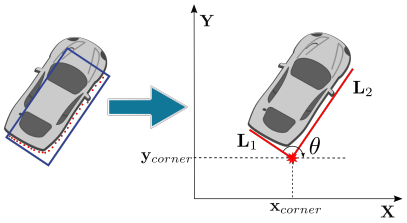


Fig. 8. Conversion of a fitted rectangle to an L-shape feature.

Summarizing the above, the L-shape feature contains five values that are extracted from the bounding box and which will be used in later stages for vehicle tracking. The position of the corner point (x_{corner}, y_{corner}), the lengths L_1, L_2 and the orientation angle θ.

III. Tracking of Moving Vehicles

The objective of the vehicle tracking stage is to estimate as accurately as possible the position, speed and dimensions of all detected vehicles. A flowchart of this stage is given in Fig. 9 and a brief explanation of its operation will be given below.

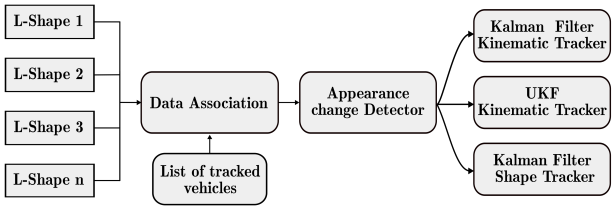


Fig. 9. Flowchart of the vehicle tracking stage of the developed system.

At the left side of the flowchart is the input into the tracking stage, which are the extracted by the detection subsystem L-shapes. The first process of the tracking stage is Data Association, in which the newly received L-shapes are associated

with tracked vehicles from previous time instances. After the L-shapes are associated with vehicles, it is investigated if the observed corner of the vehicles changed and with it the direction of the associated L-shape. If this is true, the three trackers are updated to reflect the change. Lastly, the position of the L-shape is used to update the two L-shape kinematic trackers and its dimensions and orientation are used for updating the shape tracker. The Kalman Filter (KF) kinematic tracker uses a linear vehicle motion model and its aimed at systems with low computational capabilities, while the Unscented Kalman Filter (UKF) tracker uses a nonlinear motion model and it is geared towards system with higher capabilities.

A. Data Association and Track Management

Data association is the process of associating detection results with already tracked objects by working out which observations were generated by which targets. Data association in multiple vehicle tracking is complicated because of the inherent uncertainty of sensor measurements and the fact that the number of observations does not necessarily correspond to the number of neighboring objects. Furthermore, the true number of surrounding vehicles is difficult to estimate since some vehicles might be temporarily occluded or unobserved.

Track management for multiple object tracking consists of deducing the number of surrounding objects and identifying if each observation corresponds to an already known object, to a new object in the scene or to a false measurement.

1) Data Association: In the proposed system the data association method used is the Global Nearest Neighbor (GNN) filter, which associates clusters with objects based on euclidean distance, while ensuring that each cluster is assigned to at most one object.

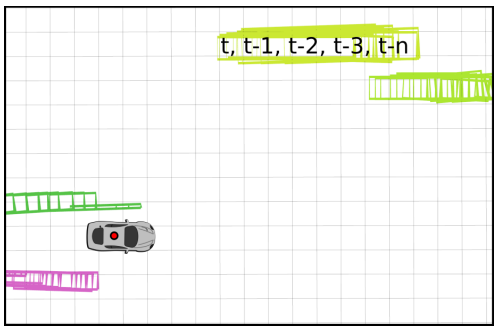


Fig. 10. Data association in the developed system.

Fig. 10 visualizes several consecutive time instances (t, $t-1, \ldots,$ $t-n$) from the simulation and the correct association of new measurements to already tracked vehicles can be observed. In case that there was an error, not all four object would have retained the same color throughout the time window, but there would be a color change at the time of the association error.

2) Track Management: Tracks is the name given to objects that are tracked by a DATMO system and track management is the process of managing the list of tracks. The main goal of track management is to reduce the amount of tracked objects, both for reducing the amount of computations performed at each timestep but also for preventing false data associations. The track management system that is used is simple in its design and operates in the following way. After every measurement update and clustering step, all the clusters not associated with any already tracked object are used to initiate new tracks. The tracks that are associated with newly detected clusters are unaffected, while the not associated tracks are immediately deleted.

B. L-shape Tracker

In order to track the position of the L-shape corner, the proposed system implements two solutions. The first one uses a Kalman Filter for tracking the corner of the L-shape, while the second one uses an Unscented Kalman Filter (UKF). The first approach is based on the work presented in [14], while the second one is novel. The system implements two solutions for two main reasons: the first being for comparing the accuracy of the Kalman Filter and the UKF in this particular application. And the second one is, providing the users of the system with options, so they can make a choice depending on the accuracy demanded by their application and the available computational resources of their platform. The dimensions (L_1, L_2), orientation (θ) and turn rate (ω) of the L-shape are tracked by a separate Kalman Filter.

1) Kalman Filter Kinematic Tracker: The Kalman Filter used for tracking the motion of the corner point uses a Constant Velocity (CV) model to estimate position and velocities, with the following state vector $\mathbf{x}_{CV}$.

$$x_{CV} = \begin{bmatrix} x_{corner} & y_{corner} & v_x & v_y \end{bmatrix}^T \tag{5}$$

The kinematic model A_{CV} that is used to track the position

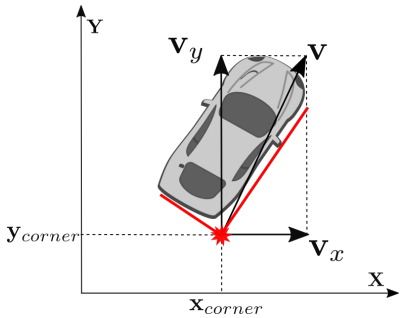


Fig. 11. The kinematic model models the motion of the corner point.

(x_{corner}, y_{corner}) and velocities v_x, v_y of the corner point, as show in Fig. 11, is the following:

$$A_{CV} = \begin{bmatrix} 1 & 0 & T & 0 \\ 0 & 1 & 0 & T \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6}$$

where T is the sampling time. Given that only the position of the corner point is measured, the measurement vector is:

$$z_{CV} = \begin{bmatrix} x_{corner} & y_{corner} \end{bmatrix}^T \tag{7}$$

and consequently the measurement model is the following:

$$H_{CV} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}. \tag{8}$$

2) UKF Kinematic Tracker: The kinematic tracker that uses a UKF has one main advantage over the one presented before and that is that nonlinear motion models can be used. The implemented one uses as kinematic function f_{CTM}, a Coordinated Turn Model (CTM) [15], which in addition to the position and speed tracks also the turn rate (ω). Therefore, the state vector $\mathbf{x}_{CTM}$ is:

$$\mathbf{x}_{CTM} = \begin{bmatrix} x & y & v_x & v_y & \omega \end{bmatrix}^T \tag{9}$$

And its kinematic function f_{CTM} is the following:

$$f_{CTM} = \begin{bmatrix} x + \frac{v_x}{\omega}\sin(\omega T) - \frac{v_y}{\omega}(1-\cos(\omega T)) \\ y + \frac{v_x}{\omega}(1-\cos(\omega T)) + \frac{v_y}{\omega}\sin(\omega T) \\ v_x\cos(\omega T) - v_y\sin(\omega T) \\ v_x\sin(\omega T) + v_y\cos(\omega T) \\ \omega \end{bmatrix}. \tag{10}$$

The measurement vector and model are similar to the ones used in the Kalman Filter, since the available measurements are the same.

$$\begin{aligned} z_{CTM} &= \begin{bmatrix} x_{corner} & y_{corner} \end{bmatrix}^T \\ H_{CTM} &= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}. \end{aligned} \tag{11}$$

3) Shape Tracker: The shape of the target vehicle is tracked using a Kalman Filter and a state vector composed of line lengths (L_1, L_2), the orientation of $L_1(\theta)$ and the turn rate (ω). Those states are visualized in Fig. 12 and are contained in vector $\mathbf{x}_s$.

$$\mathbf{x}_S = \begin{bmatrix} L_1 & L_2 & \theta & \omega \end{bmatrix}^T \tag{12}$$

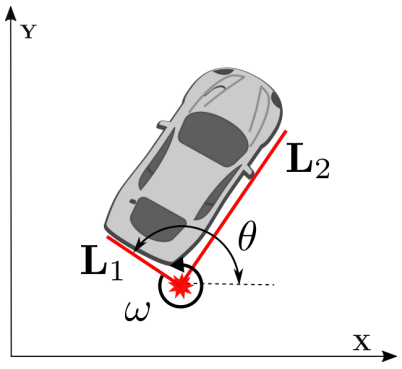


Fig. 12. The shape filter estimates the orientation, turn rate and size of the L-shape.

For estimating the vehicle's shape, a static model is applied to the line lengths (L_1, L_2) based on the assumption that the vehicle size does not change over time. For estimating the L-shape's yaw, and since the yaw rate does not change particularly fast, a constant turn rate model is chosen. The two above models are combined in a single process matrix:

$$A_S = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & T \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{13}$$

where T the sampling time and A_S is the process matrix containing the static model for the line lengths and the constant turn rate model for the orientation and the yaw rate.

Among the states of the shape model, only the yaw rate is not contained in the L-shape and therefore the measurement vector and model are the following:

$$\begin{aligned} z_S &= \begin{bmatrix} L_1 & L_2 & \theta \end{bmatrix}^T \\ H_S &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}. \end{aligned} \tag{14}$$

4) Corner Point Switching: The designed system is tracking L-shapes, which represent the closest corner of observed vehicles. However, while the ego vehicle and the observed ones are moving, it is expected that the closest corners of the other vehicles will be periodically changing and therefore the extracted L-shapes should also be changing. All the possible extracted L-shapes from a vehicle are drawn in Figure 13 and are marked by values C1 to C4.

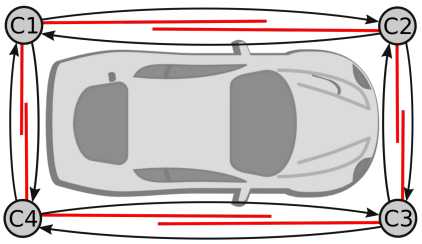
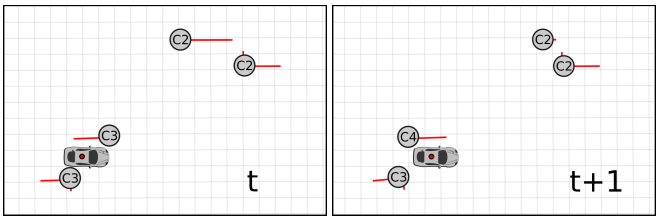


Fig. 13. Visualization of all the corner points of a vehicle (C1-C4) and of the clockwise and counter clockwise changes between them (black arrows).

An example of a corner change occurrence can be observed in Figure 14, in which two instances from the simulation are given side by side. In the first time instance (t), the overtaking vehicle at the left of the ego-vehicle is tracked by its front-right corner (C3). As it moves ahead in the second time instance $(t+1)$, the closest corner changes to the one at the vehicle's rear-right corner (C4).

In the proposed system, corner point changes were detected based on the Mahalanobis distances of the new measurement to the already tracked corner point and the two neighboring ones. If for example, the already tracked corner point is C3, with state vector $(\mathbf{c_3})$, then the state vectors of corner points C2 $(\mathbf{c_2})$ and C4 $(\mathbf{c_4})$ are calculated and their values are compared with the state vector of the new L-shape measurement $(\mathbf{m})$, via the Mahalanobis distance. The Mahalanobis distance



(a) (b)

Fig. 14. Clockwise change of closest corner point, from corner C3 to corner C4.

(D_M), which is a multivariate distance metric, is calculated based on the following formula:

$$\begin{aligned} D_M^2 &= (\mathbf{c}-\mathbf{m})^T \cdot P^{-1} \cdot (\mathbf{c}-\mathbf{m}) \\ \mathbf{c} &= [x_{corner},\ y_{corner},\ \theta,\ L_1,\ L_2,]^T \\ \mathbf{m} &= [x_{corner}, y_{corner}, \theta, L_1, L_2,]^T \end{aligned} \tag{15}$$

where P is the covariance matrix of the already tracked corner point.

If the shortest Mahalanobis distance is between the measurement vector $\mathbf{m}$ and the previously tracked corner vector, for example $\mathbf{c_3}$, then no corner switch took place. However, if the shortest distance is with $\mathbf{c_4}$, a clockwise switch took place and if it is with $\mathbf{c_2}$, a counter clockwise switch took place. In the example of Figure 14, we can observe that a clockwise corner switch between corners C3 and C4 occurred. In such cases, the filters that track the L-shape cannot be immediately updated because the new L-shape measurement corresponds to another corner than the one that the filter states are tracking. Therefore, the filter states should be altered, so that the represent the same corner with the new measurement.

In case that a clockwise switch is detected, the filter states are changed in the following way:

$$\begin{aligned} x_{\text{corner}}^{Cj} &= x_{\text{corner}}^{Ci} + L_1^{Ci}\cos\theta^{Ci} \\ y_{\text{corner}}^{Cj} &= y_{\text{corner}}^{Ci} + L_1^{Ci}\sin\theta^{Ci} \\ v_x^{Cj} &= v_x^{Ci} + L_1^{Ci}\omega\sin\theta^{Ci} \\ v_y^{Cj} &= v_y^{Ci} + L_1^{Ci}\omega\cos\theta^{Ci} \\ \theta^{Cj} &= \theta^{Ci} - \pi/2 \\ L_1^{Cj} &= L_2^{Ci} \\ L_2^{Cj} &= L_1^{Ci} \\ \text{where } &\begin{cases} j = i+1, & i<4 \\ j = 1, & i=4 \end{cases} \end{aligned} \tag{16}$$

where the superscripts Ci and Cj represents the corner number before and after the model change.

In case that a counter clockwise switch is detected the kinematic and shape states are changed in the following way:

$$\begin{aligned} x_{\text{corner}}^{Cj} &= x_{\text{corner}}^{Ci} + L_2^{Ci}\sin\theta^{Ci} \\ y_{\text{corner}}^{Cj} &= y_{\text{corner}}^{Ci} + L_2^{Ci}\cos\theta^{Ci} \\ v_x^{Cj} &= v_x^{Ci} + L_2^{Ci}\omega\cos\theta^{Ci} \\ v_y^{Cj} &= v_y^{Ci} + L_2^{Ci}\omega\sin\theta^{Ci} \\ \theta^{Cj} &= \theta^{Ci} + \pi/2 \\ L_1^{Cj} &= L_2^{Ci} \\ L_2^{Cj} &= L_1^{Ci} \\ \text{where } &\begin{cases} j = i-1, & i>1 \\ j = 4, & i=1 \end{cases} \end{aligned} \tag{17}$$

5) L-shape to Box Model Conversion: In the L-shape to box model conversion stage the kinematic state is changed from corner point motion to vehicle motion. The position of the center of the box model can be calculated by using the geometric information of the shape model

$$\begin{aligned} x_{center} &= x_{\text{corner}} + \varepsilon_x, \quad \text{where } \varepsilon_x = (L_1\cos\theta + L_2\sin\theta)/2 \\ y_{center} &= y_{\text{corner}} + \varepsilon_y, \quad \text{where } \varepsilon_y = (L_1\sin\theta - L_2\cos\theta)/2. \end{aligned} \tag{18}$$

Since the velocity of the corner point is the sum of the target velocity and the tangential velocity, the rotational motion of the corner point includes both translational velocity and rotational velocity. The rotational motion is assumed as a uniform circular motion, since a constant turn rate model for the shape model has already been assumed. Therefore, an equation for tangential velocity can be derived by using the distance from the center as the radius (r) of the circular motion.

$$\begin{aligned} v_{x,\text{center}} &= v_{x,\text{corner}} - r\omega\cos\alpha \\ v_{y,\text{center}} &= v_{y,\text{corner}} - r\omega\sin\alpha \\ &\text{where } \alpha = \tan^{-1}(\varepsilon_y/\varepsilon_x) - \pi/2. \end{aligned} \tag{19}$$

A vehicle's yaw (ψ) is typically difficult to estimate from an L-shape because of the ambiguities, since it is hard to judge which is the front part and which is the rear part of the vehicle. Therefore, it would be reasonable to keep the following four hypotheses:

$$\psi_i = \theta + i\frac{\pi}{2}, i \in \{0,1,2,3\} \tag{20}$$

where ψ_i represents the yaw angle of the vehicle [16]. Next, for calculating the actual yaw of the vehicle, these four angles are compared with the angle of the vehicle's speed and the one with the least absolute difference from the speed's orientation is chosen as the vehicle yaw. The turn rate of the box model is taken equal to the turn rate (ω) of the tracked L-shape.

In Fig. 15 the box models calculated by the system are visualized, while arrows are used to represent the estimated velocities of the surrounding vehicles. The starting point of those arrows are the estimated centers of the surrounding vehicles.

It can additionally be observed, that the dimensions of the box models are bigger than the extracted rectangles, since they are based on the estimations of the shape filter and that the

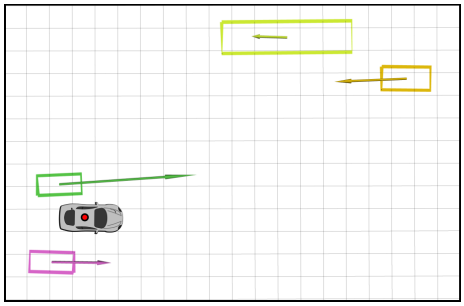


Fig. 15. Visualization of the estimated box models.

estimated orientation of the truck at the top right corner is closer to the true value, than the one estimated by the rectangle fitting algorithm (Fig. 7).

IV. Experimental Evaluation

In this section, a simulation experiment[1] that was used to test and evaluate the proposed system will be presented and the evaluation results will be explained. During this simulation experiment the output of the system and the reference measurements are both recorded and they are later analyzed to assess the accuracy of the proposed system. The reference measurements provided by the simulation environment are very accurate and with sufficiently higher frequency (100 Hz) than the estimations of the system (12 Hz) and therefore they are used as ground truth data. For evaluating the tracking performance, the system's estimates for the position (x, y), velocities (v_x, v_y), orientation (ψ), turn rate (ω) and dimensions (Length, Width) of the box models are plotted against the ground truth data, which also refer to the center of the vehicles.

In Fig. 2a, which is a snapshot of the simulation environment during this experiment, we can observe that there a total of five vehicles, from which the three at the bottom lanes are controlled via joystick input, while the bus and truck at the top lanes are following straight paths. The ego vehicle is the Prius in the middle lane and during this experiment it overtakes the car that drives on the emergency lane. At the same time, the third car which also starts in the middle lane, changes to the left lane, overtakes the ego-vehicle and then merges back in the middle lane in front of it.

The estimations of the system are plotted against the ground truth data on Fig. 16 for the car at the emergency lane and at Fig. 17 for the car at the left lane. At the first two rows of the figures there are graphs for the position (x, y) and velocities (v_x, v_y) of the tracked cars center, which are estimated by the kinematic filters. At the bottom two rows there are graphs of the orientation (ψ), turn rate (ω) and dimensions of the vehicles, which are estimated by the shape Kalman Filter.

In Fig. 16, we can observe that both filters produce identical estimates for the position (x, y) of the car in the emergency lane. However, the velocities estimated by the two filters are different, with the KF overshooting the reference and the UKF undershooting it. This can be explained, by the different process models that the filters employ. The orientation of the vehicle is estimated with high accuracy by the shape filter, after the vehicle starts moving. The turn rate however is not estimated accurately since its abrupt changes were not tracked correctly by the filter. Lastly, it is evident that the accuracy of length and width estimation, depends on which side the incoming measurements represent. Therefore, when the length is measured the length estimation is accurate and when the width is measured its estimation is more accurate. The two spikes at the beginning of those diagrams can be explained by the fact that, the orientation of the vehicle can not estimated when the vehicles are stationary.

The results for the overtaking car are drawn in Fig. 17, where it can be observed that the system has generally similar performance. However, in the (v_y) graph, a spike can be seen, which is attributed to two rapid corner switches that occur while the vehicles are almost stationary. In both cases, it is evident that the UKF offers no significant benefits to the KF, and that is probably due to the motion of the two vehicles, which is generally straight, while the model of the UKF estimates the kinematics of constant turns.

V. Conclusion

The DATMO system presented in this paper[2], encompasses the complete process of multi-object tracking; it receives 2D LIDAR measurements as input and estimates the kinematic state and dimensions of surrounding vehicles. The detection part of the system differentiates objects in the LIDAR measurements using the Adaptive Breakpoint Detector algorithm and calculates rectangles around the detected objects with the Search-Based Rectangle Fitting algorithm. The tracking part of the system receives L-shapes and associates them with already tracked vehicles, using the Global Nearest Neighbor algorithm. It estimates kinematic poses with two different tracking filters, a Kalman filter, with a constant velocity model and a UKF, with a nonlinear constant turn kinematic model.

The proposed system was evaluated in a simulation experiment and it was shown that it can reliably estimate the position, speed, heading angle, turn rate and dimensions of surrounding vehicles, in normal driving conditions. Since the system is able to run in a real-time manner, it is expected to be applicable for a variety of algorithm development projects using low-cost platforms.

References

[1] C.-C. Wang, C. Thorpe, S. Thrun, M. Hebert, and H. Durrant-Whyte, "Simultaneous localization, mapping and moving object tracking," *The International Journal of Robotics Research*, vol. 26, no. 9, pp. 889–916, 2007.

[2] A. Petrovskaya, M. Perrollaz, L. Oliveira, L. Spinello, R. Triebel, A. Makris, J.-D. Yoder, C. Laugier, U. Nunes, and P. Bessiere, "Awareness of road scene participants for autonomous driving," in *Handbook of Intelligent Vehicles*. Springer, 2012, pp. 1383–1432.

[1] A video of the experiment can be found at: https://youtu.be/JrbJjmIv730

[2] The complete system code can be found at:github.com/kostaskonkk/datmo

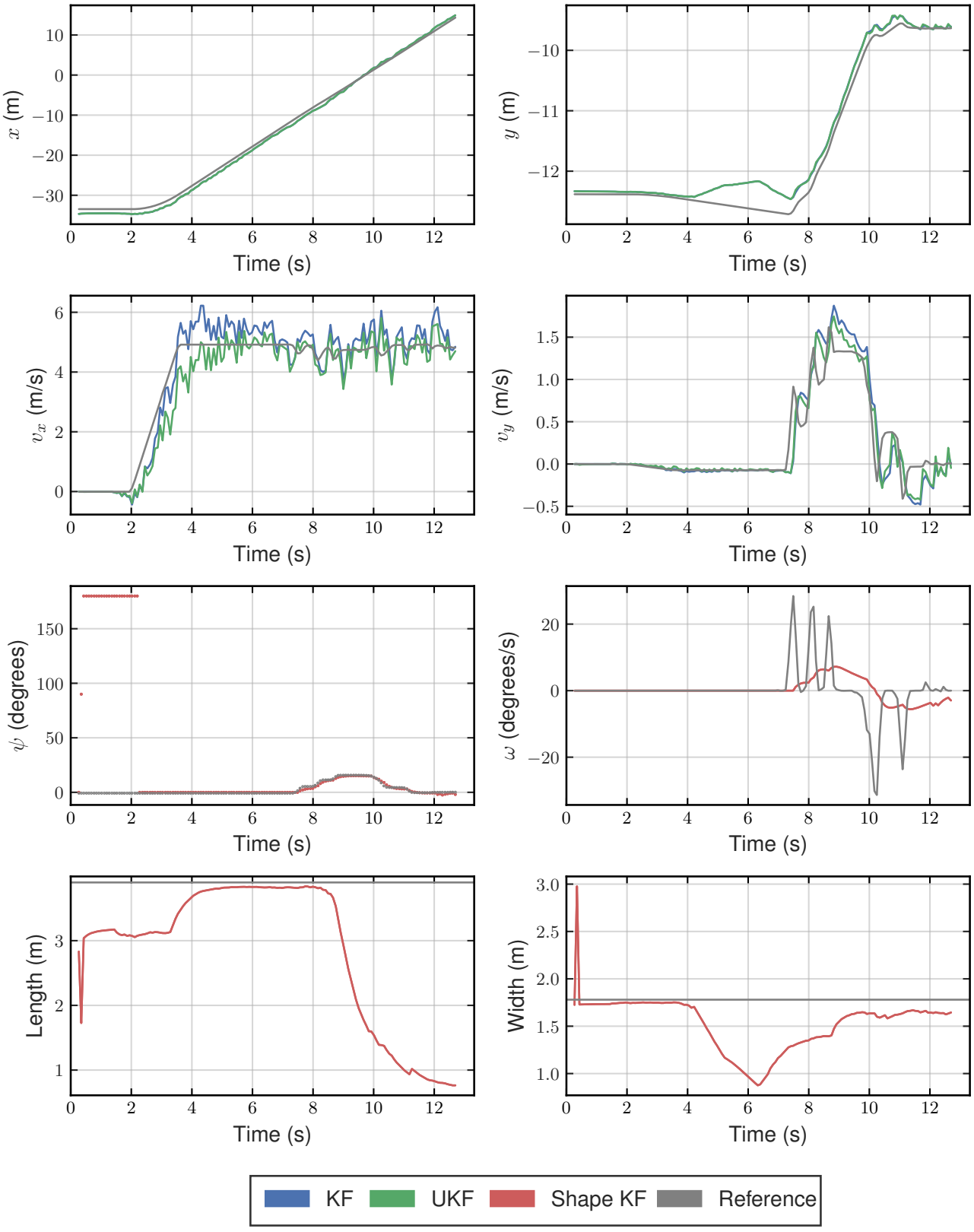


Fig. 16. Estimated states by the developed system against the simulation ground truth, for the car at the left of the ego-vehicle.

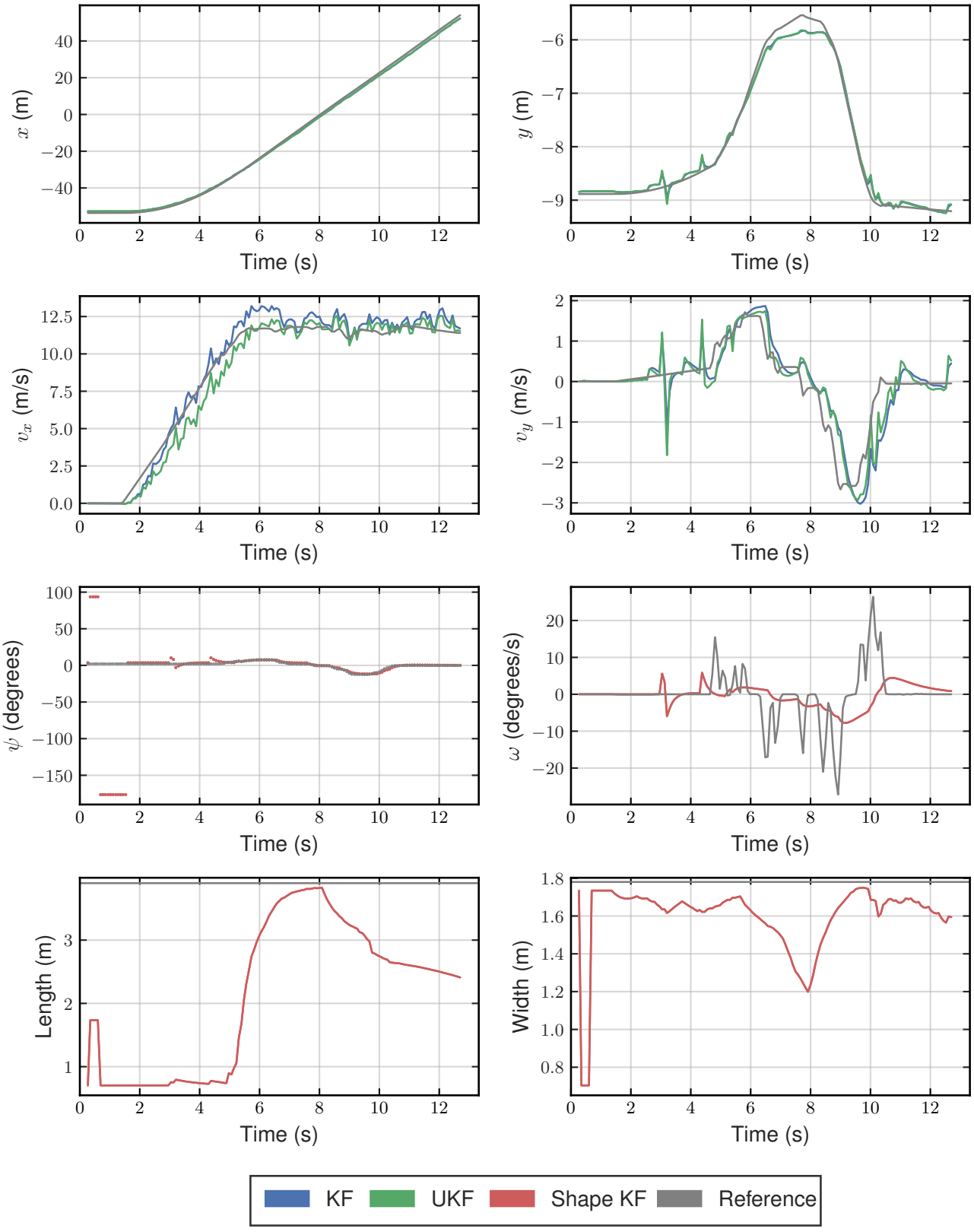


Fig. 17. Estimated states by the developed system against the simulation ground truth, for the car at the left of the ego-vehicle.

[3] J. Choi, S. Ulbrich, B. Lichte, and M. Maurer, "Multi-target tracking using a 3d-lidar sensor for autonomous vehicles," in *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*. IEEE, 2013, pp. 881–886.

[4] J. Leonard, J. How, S. Teller, M. Berger, S. Campbell, G. Fiore, L. Fletcher, E. Frazzoli, A. Huang, S. Karaman *et al.*, "A perception-driven autonomous urban vehicle," *Journal of Field Robotics*, vol. 25, no. 10, pp. 727–774, 2008.

[5] A. Petrovskaya and S. Thrun, "Model based vehicle tracking for autonomous driving in urban environments," *Proceedings of Robotics: Science and Systems IV, Zurich, Switzerland*, vol. 34, 2008.

[6] T. Chen, R. Wang, B. Dai, D. Liu, and J. Song, "Likelihood-field-model-based dynamic vehicle detection and tracking for self-driving," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 11, pp. 3142–3158, 2016.

[7] K. Granstrom, M. Baum, and S. Reuter, "Extended object tracking: Introduction, overview and applications," *arXiv preprint arXiv:1604.00970*, 2016.

[8] T.-D. Vu, J. Burlet, and O. Aycard, "Grid-based localization and local mapping with moving object detection and tracking," *Information Fusion*, vol. 12, no. 1, pp. 58–69, 2011.

[9] R. Danescu, F. Oniga, and S. Nedevschi, "Modeling and tracking the driving environment with a particle-based occupancy grid," *IEEE Transactions on Intelligent Transportation Systems*, vol. 12, no. 4, pp. 1331–1342, 2011.

[10] N. Kaempchen, M. Buehler, and K. Dietmayer, "Feature-level fusion for free-form object tracking using laserscanner and video," in *Intelligent vehicles symposium, 2005. Proceedings. IEEE*. IEEE, 2005, pp. 453–458.

[11] G. A. Borges and M.-J. Aldon, "Line extraction in 2d range images for mobile robotics," *Journal of intelligent and Robotic Systems*, vol. 40, no. 3, pp. 267–297, 2004.

[12] T. Johansson and O. Wellenstam, "Lidar clustering and shape extraction for automotive applications," Master's thesis, Chalmers University of Technology, 2017.

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

[14] D. Kim, K. Jo, M. Lee, and M. Sunwoo, "L-shape model switching-based precise motion tracking of moving vehicles using laser scanners," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 2, pp. 598–612, 2018.

[15] M. Roth, G. Hendeby, and F. Gustafsson, "Ekf/ukf maneuvering target tracking using coordinated turn models with polar/cartesian velocity," in *17th International Conference on Information Fusion (FUSION)*. IEEE, 2014, pp. 1–8.

[16] X. Shen, S. Pendleton, and M. H. Ang, "Efficient l-shape fitting of laser scanner data for vehicle pose estimation," in *2015 IEEE 7th International Conference on Cybernetics and Intelligent Systems (CIS) and IEEE Conference on Robotics, Automation and Mechatronics (RAM)*. IEEE, 2015, pp. 173–178.

Bibliography

[1] R. O. Chavez-Garcia and O. Aycard, "Multiple sensor fusion and classification for moving object detection and tracking," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 2, pp. 525–534, 2015.

[2] A. Petrovskaya, M. Perrollaz, L. Oliveira, L. Spinello, R. Triebel, A. Makris, J.-D. Yoder, C. Laugier, U. Nunes, and P. Bessiere, "Awareness of road scene participants for autonomous driving," in *Handbook of Intelligent Vehicles*, pp. 1383–1432, Springer, 2012.

[3] T.-D. Vu and O. Aycard, "Laser-based detection and tracking moving objects using data-driven markov chain monte carlo," in *2009 IEEE International Conference on Robotics and Automation*, pp. 3800–3806, IEEE, 2009.

[4] A. Nègre, L. Rummelhard, and C. Laugier, "Hybrid sampling bayesian occupancy filter," in *2014 IEEE Intelligent Vehicles Symposium Proceedings*, pp. 1307–1312, IEEE, 2014.

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

[6] T. Johansson and O. Wellenstam, "Lidar clustering and shape extraction for automotive applications," Master's thesis, Chalmers University of Technology, 2017.

[7] D. Kim, K. Jo, M. Lee, and M. Sunwoo, "L-shape model switching-based precise motion tracking of moving vehicles using laser scanners," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 2, pp. 598–612, 2018.

[8] A. Rachman, "3d-lidar multi object tracking for autonomous driving," Master's thesis, Delft University of Technology, 2017.

[9] E. A. Wan and R. Van Der Merwe, "The unscented kalman filter for nonlinear estimation," in *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No. 00EX373)*, pp. 153–158, Ieee, 2000.

[10] C.-C. Wang, C. Thorpe, S. Thrun, M. Hebert, and H. Durrant-Whyte, "Simultaneous localization, mapping and moving object tracking," *The International Journal of Robotics Research*, vol. 26, no. 9, pp. 889–916, 2007.

[11] J. Choi, S. Ulbrich, B. Lichte, and M. Maurer, "Multi-target tracking using a 3d-lidar sensor for autonomous vehicles," in *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*, pp. 881–886, IEEE, 2013.

[12] J. Leonard, J. How, S. Teller, M. Berger, S. Campbell, G. Fiore, L. Fletcher, E. Frazzoli, A. Huang, S. Karaman, *et al.*, "A perception-driven autonomous urban vehicle," *Journal of Field Robotics*, vol. 25, no. 10, pp. 727–774, 2008.

[13] A. Petrovskaya and S. Thrun, "Model based vehicle tracking for autonomous driving in urban environments," *Proceedings of Robotics: Science and Systems IV, Zurich, Switzerland*, vol. 34, 2008.

[14] T. Chen, R. Wang, B. Dai, D. Liu, and J. Song, "Likelihood-field-model-based dynamic vehicle detection and tracking for self-driving," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 11, pp. 3142–3158, 2016.

[15] K. Granstrom, M. Baum, and S. Reuter, "Extended object tracking: Introduction, overview and applications," *arXiv preprint arXiv:1604.00970*, 2016.

[16] T.-D. Vu, J. Burlet, and O. Aycard, "Grid-based localization and local mapping with moving object detection and tracking," *Information Fusion*, vol. 12, no. 1, pp. 58–69, 2011.

[17] R. Danescu, F. Oniga, and S. Nedevschi, "Modeling and tracking the driving environment with a particle-based occupancy grid," *IEEE Transactions on Intelligent Transportation Systems*, vol. 12, no. 4, pp. 1331–1342, 2011.

[18] H. P. Moravec, "Sensor fusion in certainty grids for mobile robots," in *Sensor devices and systems for robotics*, pp. 253–276, Springer, 1989.

[19] P. Bessière, C. Laugier, and R. Siegwart, *Probabilistic reasoning and decision making in sensory-motor systems*, vol. 46. springer, 2008.

[20] K. Mekhnacha, Y. Mao, D. Raulo, and C. Laugier, "Bayesian occupancy filter based" fast clustering-tracking" algorithm," 2008.

[21] N. Kaempchen, M. Buehler, and K. Dietmayer, "Feature-level fusion for free-form object tracking using laserscanner and video," in *Intelligent vehicles symposium, 2005. Proceedings. IEEE*, pp. 453–458, IEEE, 2005.

[22] P. Skrzypczynski, "Building geometrical map of environment using ir range finder data," in *Intelligent Autonomous Systems*, pp. 408–412, 1995.

[23] G. A. Borges and M.-J. Aldon, "Line extraction in 2d range images for mobile robotics," *Journal of intelligent and Robotic Systems*, vol. 40, no. 3, pp. 267–297, 2004.

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

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

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

[27] B. Anderson and J. B. Moore, "Optimal filtering," 1979.

[28] S. J. Julier and J. K. Uhlmann, "New extension of the kalman filter to nonlinear systems," in *Signal processing, sensor fusion, and target recognition VI*, vol. 3068, pp. 182–193, International Society for Optics and Photonics, 1997.

[29] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*. MIT press, 2005.

[30] J. J. LaViola, "A comparison of unscented and extended kalman filtering for estimating quaternion motion," in *Proceedings of the 2003 American Control Conference, 2003.*, vol. 3, pp. 2435–2440, IEEE, 2003.

[31] G. Zhai, H. Meng, and X. Wang, "A constant speed changing rate and constant turn rate model for maneuvering target tracking," *Sensors*, vol. 14, no. 3, pp. 5239–5253, 2014.

[32] M. Roth, G. Hendeby, and F. Gustafsson, "Ekf/ukf maneuvering target tracking using coordinated turn models with polar/cartesian velocity," in *17th International Conference on Information Fusion (FUSION)*, pp. 1–8, IEEE, 2014.

[33] T. Moore and D. Stouch, "A generalized extended kalman filter implementation for the robot operating system," in *Intelligent autonomous systems 13*, pp. 335–348, Springer, 2016.

[34] H. Zhao, X. Shao, K. Katabira, and R. Shibasaki, "Joint tracking and classification of moving objects at intersection using a single-row laser range scanner," in *2006 IEEE Intelligent Transportation Systems Conference*, pp. 287–294, IEEE, 2006.

[35] X. Shen, S. Pendleton, and M. H. Ang, "Efficient l-shape fitting of laser scanner data for vehicle pose estimation," in *2015 IEEE 7th International Conference on Cybernetics and Intelligent Systems (CIS) and IEEE Conference on Robotics, Automation and Mechatronics (RAM)*, pp. 173–178, IEEE, 2015.

[36] J. Rieken, R. Matthaei, and M. Maurer, "Toward perception-driven urban environment modeling for automated road vehicles," in *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, pp. 731–738, IEEE, 2015.

[37] M. Filipenko and I. Afanasyev, "Comparison of various slam systems for mobile robot in an indoor environment," in *2018 International Conference on Intelligent Systems (IS)*, pp. 400–407, IEEE, 2018.

[38] S. Kohlbrecher, O. Von Stryk, J. Meyer, and U. Klingauf, "A flexible and scalable slam system with full 3d motion estimation," in *2011 IEEE International Symposium on Safety, Security, and Rescue Robotics*, pp. 155–160, IEEE, 2011.

[39] S. Umeyama, "Least-squares estimation of transformation parameters between two point patterns," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, pp. 376–380, April 1991.

Glossary

List of Acronyms

DCSC	Delft Center for Systems and Control
DSV	Delft Scaled Vehicle
BOF	Bayesian Occupancy Filter
FCTA	Fast Clustering and Tracking Algorithm
DATMO	Detection and Tracking of Moving Objects
IMU	Inertial Measurement Unit
SLAM	Self Localization and Mapping
LIDAR	LIght Detection And Ranging
ROS	Robot Operating System
ALS	Autocovariance Least-Squares
NN	Nearest Neighbor
GNN	Global Nearest Neighbor
BARC	Berkeley Autonomous Race Car
JPDAF	Joint Probabilistic Data Association Filter
MHT	Multiple Hypothesis Tracking
MoCap	Motion Capture
KF	Kalman Filter
EKF	Extended Kalman Filter
UKF	Unscented Kalman Filter
RMSE	Root Mean Squared Error

List of Symbols

Latin Symbols

$\hat{x}$	State vector estimate
$\hat{z}$	Measument vector estimate
$\mathcal{X}$	Sigma point matrix
A_K	Linear kinematic model
A_S	Linear shape model
D_{max}	Maximum threshold distance
D_{split}	Accepted disimilarity between two lines
f	Nonlinear state-transition function
h	Nonlinear measurement function
H_K	Linear measurement model
H_S	Linear measurement model
K	Kalman gain
L	Length of tracked object
P	Estimate error covariance matrix
Q	Process noise covariance matrix
R	Measurement noise covariance matrix
T	Sampling time
v_x	Velocity of tracked object on global X axis
v_y	Velocity of tracked object on global Y axis
W	Weights matrix
W	Width of tracked object
x	Position of tracked object on global X axis
x_{center}	Position of the center of tracked object on global X axis
x_{corner}	Position of the corner of tracked object on global X axis
y	Position of tracked object on global Y axis
y_{center}	Position of the center of tracked object on global Y axis
y_{corner}	Position of the corner of tracked object on global Y axis
p	Single LIDAR measurement
r	Range of a LIDAR measurement

Greek Symbols

α	Angular resolution of a LIDAR sensor
β	Probability of measurement
δ	Steering angle

$\dot{\psi}$	Turn rate of tracked object
η	Normalization constant
κ	Scaling parameter
λ	Angle used by Adaptive Breakpoint Detector
ω	Turn rate of tracked object
ϕ	Scanning angle of a LIDAR measurement
ψ	Heading angle of tracked object
σ_r	Sensor noise of a LIDAR sensor
θ	Heading angle of tracked object