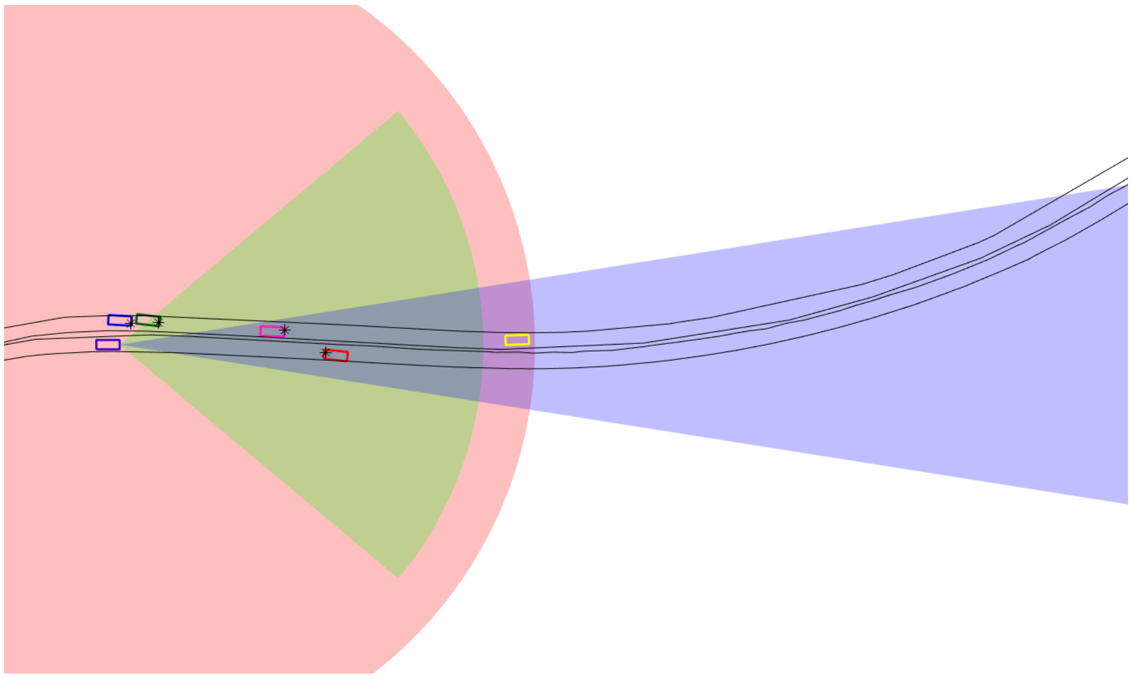
CHALMERS
UNIVERSITY OF TECHNOLOGY



Radar-LIDAR sensor fusion for vehicle tracking

Development of a more accurate and confident ground truth system for object detection surrounding an ego vehicle by fusing radar and LIDAR

Master's thesis in Systems, Control and Mechatronics

Joel Albinsson
Martin Hill

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2019

MASTER'S THESIS 2019

Radar-LIDAR sensor fusion for vehicle tracking

Development of a more accurate and confident ground truth system for object detection surrounding an ego vehicle by fusing radar and LIDAR

Joel Albinsson
Martin Hill

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Signals Processing and Biomedical Engineering
Signal Processing Group
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2019

Radar-LIDAR sensor fusion for vehicle tracking
Development of a more accurate and confident ground truth system for object detection surrounding an ego vehicle by fusing radar and LIDAR
JOEL ALBINSSON
MARTIN HILL

© JOEL ALBINSSON, 2019.
© MARTIN HILL, 2019.

Supervisor: Daniel Lovén Öberg, Volvo Cars
Examiner: Karl Granström, Chalmers University of Technology

Master's Thesis 2019
Department of Electrical Engineering
Division of Signals Processing and Biomedical Engineering
Signal Processing Group
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Top-down view of detected vehicles around ego car.

Typeset in LaTeX
Gothenburg, Sweden 2019

Radar-LIDAR sensor fusion for vehicle tracking
Development of a more accurate and confident ground truth system for object detection surrounding an ego vehicle by fusing radar and LIDAR
JOEL ALBINSSON
MARTIN HILL
Department of Department of Electrical Engineering
Chalmers University of Technology

Abstract

This thesis has been performed at Volvo Cars, at the department Prototype Vehicles, Data and Methods which supports development of active safety and advanced driver assistance system. It has been dedicated to exploring the feasibility and advantages of using various complementary sensors when generating reference data. Whenever a sensor based system is being developed a vital aspect of achieving optimal performance is to verify each sensor independently, and eventually also the system as a whole. Generating this reference in a dynamic environment such as a traffic environment is often, as in this case, achieved by an additional sensor-set, a reference system. To be able to verify sensor and system performance the reference must be capable of providing superior performance and robustness. A reference system as such will therefore benefit from making the most of available data by any and all means possible. In this thesis a sensor fusion between radar and LIDAR has been performed with the use of a Rauch-Tung-Striebel smoother. The code framework is created in a flexible way where different models and parameters can be switched out to compare the relative performance of various solutions. A data association algorithm using the nearest neighbour method is used to match detections from the two sensors. The implemented solution shows increased performance with smoother estimations, increased redundancy, and confidence in detections of surrounding vehicles as compared to the individual sensors.

Keywords: sensor fusion, tracking, track fusion, radar, LIDAR, Kalman Filter, RTS-smoother, reference system

Acknowledgements

We would like to extend our gratitude to our supervisor at Chalmers, Karl Granström, for support and good advice during the course of the project.

We would also like to thank Volvo Cars for letting us conduct our project with them and providing us with data. A special thanks goes out to the Prototype Vehicles, Data and Methods team for continuous support throughout the project. Especially to our supervisor Daniel Lovén Öberg who has helped us with getting the knowledge and resources needed to carry out the project.

Additionally we would like to thank some fellow students. Rasmus Lundin for help with technical issues. Erik Bohnsack and Adam Lilja for peer-reviewing and providing valuable insight during the writing process.
Finally we would like to extend our deepest gratitude to Jonas Harström and Daniel Eriksson for providing insightful questions after the presentation and also valuable written feedback on the report.

Joel Albinsson, Martin Hill, Gothenburg, June 2019

Contents

List of Figures

List of Tables

1 Introduction

This chapter describes the background, aim, purpose and delimitations of the project. The project is carried out in cooperation with the Prototype Vehicles, Data and Methods department at Volvo Cars.

1.1 Background

The main issue with safety related to travel by car always has and always will be the driver; human error is the cause of most traffic-related accidents [3]. To minimize the danger of inevitable human mistakes, automotive manufacturers have started to integrate an increasing number of sensors to allow vehicles to interpret its surroundings. The intention of giving the vehicle awareness of its surroundings is to aid the driver in averting easily avoidable accidents. There are already a variety of sensors integrated into many modern cars, ultrasonic sensors, cameras, and radar has become commonplace in the upper echelons of the automotive industry. Input from these sensors is used by the vehicle to interpret the surrounding traffic situation and assist the driver in situations where immediate intervention is necessary to mitigate or avoid an imminent collision. However, the sensors are never perfect, and the amount of scenarios that a car might encounter are almost endless.

Driver assistance systems and autonomous driving features are quickly becoming a requirement in the competitive automotive industry and in the pursuit of optimal performance developers are collecting massive amounts of data. Data from various driving scenarios is captured by the sensors integrated in many new cars and also by additional reference sensors used by test vehicles. The sensors included in the reference system can be chosen based on performance rather than size and cost constraints that must be met by the integrated sensors. The high-performance sensors and the integrated sensors simultaneously record the surrounding environment. This provides developers with a baseline against which they can compare the surroundings as interpreted by the sensors integrated in the vehicle. The mentioned baseline is referred to as the ground truth or reference frame since it is often used to validate the accuracy of the integrated sensors enabled by the increased performance over the integrated sensors.

Validating performance by comparing input from integrated sensors to an accurate model of the surrounding environment, generated by a reference system, enables more efficient development and testing of advanced driver assistance systems. This approach depends on the accuracy of the ground truth model generated from sensor data from the external sensors. If the model does not accurately depict the surroundings the recorded data loses its value. Using an inaccurate ground truth model might have a detrimental effect on development and validation of the integrated sensors, assuming the inaccuracies are unknown to developers. To capture an accurate ground truth model developers use a reference box. The reference box consists of a roof box mounted on the roof rack of the test-vehicles and is equipped with a multitude of high-end sensors.

Test-vehicles often face sub-optimal conditions where one or several of the sensors might not be able to perform optimally. Loss of performance can result in an inaccurate reference frame despite the accuracy of the individual sensors, limiting the value and accuracy of the collected data.

1.2 Aim

The aim of this master's thesis will be to increase the reliability and accuracy of the ground truth model generated from the proprietary reference box. By combining data from two sensors, the project group intends to provide increased system performance using existing hardware already integrated in the reference box. Improvements are made possible by the complementary abilities of radar and LIDAR which will be further discussed in Section 2 Hardware.

Evaluate tracking performance of the fused output generated using pre-recorded data and variuos solutions for sensor fusion. Collect additional data using reference system and a set of Real Time Kinematic (RTK) sensors [4] to provide a ground truth against which tracking performance of the sensor fusion output may be validated. The end goal is to provide a more accurate output in relation to the RTK ground truth than either of the two sensors are capable of providing on their own.

Increase the robustness of the system with the use of data from sensors with complementary abilities, see Section 2.1 and 2.2. Robustness of the fusion may be validated under conditions where one or both sensors perform badly, such as in heavy weather or over large distances.

1.3 Delimitations

The project will focus on only two sensors types: radar and LIDAR.

The sensor data used by the algorithms created during the course of this project has been preprocessed by a decoding software that takes raw sensor data from the reference box as input. From the reference box data, the decoding software generates a data structure with identified objects, their dynamic states (position, velocity and more), attributes, and more. The preprocessing that has been performed on the data was carried out by another in-house team, whom has since left the company. Due to communication and legal issues the details of this preprocessing has been unavailable and have thus been considered outside the scope of this project. The output from the preprocess however is tracked objects and this is what has been used as input to the implemented algorithm.

The hardware used to gather data shall remain unchanged to avoid adding additional expenses to the test equipment. The code created during the project should contribute to and be a part of a pre-existing toolchain. It is intended to be an optional module providing increased tracking performance if desired by developers.

The project is carried out as a master's thesis and thus the time is limited. Due to limitations in time and available resources; such as test vehicles and staff with the required expertise, it is impossible to guarantee that valid ground truth data will be attainable. The accuracy of the results can never be verified beyond the accuracy of the available reference. If such reference does not become available within the limited time a qualitative evaluation of the results will instead be performed. This qualitative evaluation together with the implemented solution should, hopefully, then be considered preparation for future work implementing sensor fusion to the existing solution.

2 Hardware

When creating a ground truth system for the area surrounding the car, sensor performance is crucial. Sensors are devices that by various means gather information about the state of a system. There are endless means of sensing the state of systems and each method has its inherent benefits and issues. Sensors that provide a ground truth against which other sensors are evaluated have to ensure higher accuracy than the sensors being evaluated. A reference frame also needs to remain accurate and perform well in difficult conditions such as heavy rain or snow to be able to see under which conditions the sensors being evaluated fail. The two sensor types used in this project are radar and LIDAR sensors which have complementary abilities discussed further in the following sections. These sensors are integrated in to a roof mounted box to get a good perspective over the surrounding environment.

2.1 Radar

The radar is a sensor which operates by transmitting pulses of radio waves and then measuring the signal that is reflected back, from the returning signal positions and motions of objects are gathered [5]. The radar in the reference box is a Continental, model ARS408-21 [1]. It is an advanced radar sensor with two operating modes; long-range and near-range. The long-range mode has a maximum range of 250 m with a narrow field of view, ±4°. The near mode has a wider field of view, ±40° and a shorter range of 70 m. The device is capable of switching between modes quickly, to the extent that it can operate in both modes almost simultaneously. By alternating between operating modes for every scanning sweep the sensor is capable of providing both long- and short range information with a narrow and wide field of view respectively. For a more precise visualization of the range and field of view see Figure 2.1.

The radar operates at a signal frequency of 77 GHz which is the frequency at which most new automotive radars are currently operating and it enables increased performance compared to the older 24 GHz [6]. The connection between the radar and the car uses a CAN interface which is the standard interface used in the automotive industry and also how all the sensors in the project reference box are communicating.

Radars can not only give the position but also the velocity of the object in only one measuring cycle by using the Doppler's Principle which gives the radial velocity [7]. One measuring cycle for the radar takes 72 ms during which it performs both near-

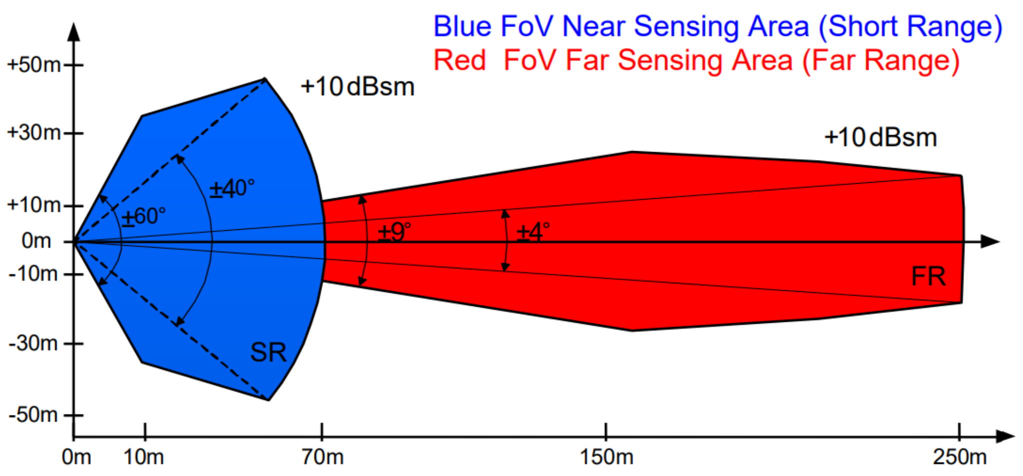

Blue FoV Near Sensing Area (Short Range)
Red FoV Far Sensing Area (Far Range)
+10 dBsm
+10 dBsm
±60°
±40°
±9°
±4°
SR
FR
+50m
+30m
+10m
0m
-10m
-30m
-50m
0m
10m
70m
150m
250m

Figure 2.1: Continental ARS408-21 performance at different ranges, image from Continental Datasheet [1]

and far-range measurements. The most relevant performance statistics for the radar sensor are presented in Table 2.1.

Resolution distance measuring	Up to 1.79 m far range, 0.39 m near range
Accuracy distance measuring	±0.40 m far range, ±0.10 m near range
Resolution azimuth angle	1.6° far range, 3.2°@0° / 4.5°@±45° / 12.3°@±60° near range
Accuracy azimuth angle	±0.1° far range, ±0.3°@0°/ ±1°@±45°/ ±5°@±60°near range
Velocity resolution	0.37 km/h far field, 0.43 km/h near range
Velocity accuracy	±0.1 km/h

Table 2.1: Continental ARS 408-21 measuring performance [1]

The radar has some benefits which the other sensors cannot match. One of the most important benefits of radar is its robustness both to lighting- and environmental conditions. Radar does not use emitted or external light when operating which makes it superior to, for example, cameras which are often used in cars and does not perform well in darkness, or LIDAR which struggles with reflective surfaces. Another strength with radar is its ability to perform well in harsh weather conditions where light based sensors sometimes struggle. In a study performed by Ryde and Hillier both radar and Laser scanners were tested in rainy and dusty conditions [8]. Results showed that the radar remained unaffected by the test conditions, heavy rain with 50–70 mm/h and dust with a 10 m visibility. The study demonstrated that the robustness of radar is excellent, however the positional accuracy was deemed too low for use in their intended application.

Lacking positional accuracy is one of the drawbacks with radar, its resolution and accuracy. It is not recommended to have the radar as the only sensor in an autonomous vehicle because of the inability to create an accurate map of objects around the car. The resolution will not allow the radar to distinguish for example: bicycles from pedestrians since it does not see minor differences in shapes and outlines of smaller targets. On near range it also has a problem doing accurate latitudinal localization as can be seen in Table 2.1. So without the help of cameras or LIDAR the localization of surrounding objects will not be accurate enough for the car to drive autonomously.

2.2 LIDAR

LIDARs measures distances to objects by measuring the distance to a fine grid of points equally spread around the sensor to create a point cloud. Each data point generated by a LIDAR sensor is calculated from the time between a pulse of laser light being emitted and the reflection of that light returning to the sensor. Knowing the speed of light and the time taken for a pulse of light to travel to and back from an object the distance to that point is calculated with reasonable accuracy for automotive purposes. The LIDAR that will be used in this project is from Velodyne, model HDL-64E [2]. This LIDAR utilizes 64 individual channels spread over a 26,9° vertical field of view. The horizontal field of view however is 360° since the whole sensor-head is mounted on a platform that spins continuously during operation. The device is capable of capturing between 1,3 and 2,2 million data points per second, and the horizontal resolution is a result of the capture rate and the user-defined rotation speed. The accuracy and other performance specifications are presented in Table 2.2.

Measurement range	Up to 120 m
Range accuracy	Up to ±5 cm, Generally ±2 cm
Field of View (vertical)	+2.0°to -24.9°(26.9°)
Angular resolution (vertical)	0.4°
Field of View (horizontal)	360°
Angular resolution (horizontal)	0.08°- 0.35°
Rotation rate	5 Hz - 20 Hz

Table 2.2: Velodyne HDL-64E S3 measuring performance [2]

With the 360° field of view, a LIDAR sensor is well suited for generating a reference frame since a single sensor easily captures everything around it with high precision. LIDAR also functions independently of external light and works just as well in pitch black darkness as in direct sunlight. It also has a very good resolution and accuracy for position measurements which makes it perfect for doing accurate localization and tracking. Since the sensing is performed with a fixed number of points each sweep, near objects will be covered with more laser points and thus greater accuracy than far objects of the same size. This makes LIDARs especially suited for near range detection.

While precision is high, LIDAR sensors have a few issues that inhibits it from being the one and only sensor needed. The first of these issues is that the detail in a scan is dependent on the number of gathered data points, thus, to generate a detailed image the sensor needs to rotate at a slower rate. While allowing more detail to be captured a slower rotation also results in a lower update frequency. In automotive applications cars are routinely traveling in excess of 30 m/s and when two oncoming cars traveling in opposite directions meet, their relative speed will be twice that. Due to the high speeds involved and the relatively low refresh rate of a LIDAR sensor, some movements will be too fast for the LIDAR to properly capture. LIDAR can only give estimates of the velocity of identified objects based on their shift in position between successive scans.

Another thing that must be taken into account when using LIDAR mounted on a moving vehicle is that it is a spinning sensor. This means that when it is moving while spinning the detected point cloud will be distorted since it will detect the last points of the revolution approximately 100 ms later, assuming a rotation speed of 10 Hz, than the points in the beginning of the revolution. This will give significant distortions to the resulting output as the vehicle speed increases. There are methods for correcting this problem with motion distortions [9].

LIDAR also encounters problems in conditions where visibility is limited, since LIDAR works by bouncing laser beams of objects around it, misreadings are common if there are a lot of debris in the air. In conditions where visibility is limited by fog, snow, rain, dust or similar the reliability of LIDAR scans drops significantly [8]. Without having any additional sensors in the system the reference system would be very vulnerable to tougher weather conditions.

2.3 Real Time Kinematic System

The roofbox is also equipped with a high accuracy Global Navigation Satellite System with Inertial Navigation System (GNSS/INS) for additional accuracy. This unit provides very precise position measurements by combining GNSS with additional information from an Internal Measurement Unit (IMU). The GNSS/INS allows for high accuracy measurements of ego motion and position in a world coordiante system. An IMU uses a combination of accelerometers and gyroscopes to determine changes in position as a result of the acceleration and rotation causing that change. Estimating position as a result of acceleration and rotation works well in some situations. It works especially well over short periods since during short periods minor noise in acceleration measurements will not accumulate to large errors in estimated position. The hardware used is an OXTS RT3000 [10].

The unit has the ability to provide additional accuracy when used with a calibrated base station in a Real-Time Kinematic (RTK) setup. When a base station is available the unit can measure position with an accuracy of ± 2 cm. The base station provides additional accuracy by also being connected to the same GNSS satellites used by the vehicles. However, unlike the mobile vehicles this base station is always stationary, thereby the position of that station is known and unchanging. Therefore the base station has a reference against which the GPS position can be compared, calculating the difference between these two positions gives a good estimation of the offset between GNSS position and the true position of the base station. By assuming that the offset between the true position and the GNSS provided position will be constant over the local area around the base station, the positions for every point can be compensated with this offset to increase accuracy. The same offset will then be communicated to the surrounding vehicles and applied to their GNSS position too, resulting in a very accurate GNSS positioning around the base station. The assumption that all GNSS receivers will experience the same offset as the base station is only valid if both are connected to the same GNSS satellites. The increased accuracy can thus only be guaranteed within a radius of 10-20 kilometres of the base station.

The data from the RTK can then be used in the postprocessing of the radar and LIDAR to make better use of the data these sensors generate, e.g. for ego motion compensation of the measurements to enable better tracking of the surrounding objects.

The RTK can also have other purposes during testing. Since it has very accurate position and velocity measurements it could be used as a ground truth to evaluate the accuracy of the roofbox output. This would require that both the ego vehicle and the surrounding vehicles has RTK systems installed and also that the testing will be performed close to a calibrated base station. System performance evaluation with RTK will be further described in Chapter 6.

3
Data

When doing sensor fusion or any kind of data fusion it is, of course, necessary to have data from two or more sources. The data can vary widely between various sensor fusion applications, from low level data points to higher level features. This is described in more detail in Chapter 5. In this thesis the data will mainly be positions and velocities of surrounding objects. The data provided as input and the processing that is performed before the fusion can occur is described in this chapter.

3.1 Preprocessing

The input data that has been used in this project is not raw data from the two sensors but rather preprocessed sensor data. It is often the case with complex sensors that the output data must be processed to reveal and extract valuable information. The process often includes several steps which are performed in succession to raw sensor data. For the radar the output is not raw data, as read from the manufacturers web page it already tracks objects. It therefore does not need the same amount of preprocessing, e.g. clustering, before it would work with the implemented code [1],[11]. For the LIDAR data however the first step is clustering of data points to identify objects in the point cloud. Each sensor update generates a new set of objects, these objects are then tracked with each sensor update to reveal their apparent movement. The following sections describe a few alternatives that may be used to perform data clustering and tracking. Due to the delimitations in Section 1.3 it is not certain that the raw data used in this project was preprocessed using the following methods, the impact of this is discussed in section 3.3.

3.1.1 Clustering

The LIDAR sensor returns data in form of a point cloud where every point essentially carries information about the distance between the sensor and the object reflecting the signal. These points must be clustered and sorted to reveal meaningful information about the surroundings. This transformation from point cloud into comprehensible and meaningful information is commonly performed with a clustering algorithm. There are many different clustering algorithms out there. One clustering method that is often used is the DBSCAN algorithm which is efficient and works well for larger datasets [12]. Another one that could be used is K-means clustering described in e.g., [13] which clusters faster than DBSCAN.

3.1.2 Tracking

The objects from the LIDAR clustering and the objects identified from the radar are then tracked. Tracking of objects is something that has been studied and applied in many and varied applications. When doing tracking for both radar and LIDAR it is suitable to use a Kalman filter [14]. The inner workings of the Kalman filters are described in Sections 5.2 and 5.3. When the processing is done after the data has been collected another possible method is a smoothing algorithm, the basics of which is the same with the difference being that also the future data is used to perform each estimation, smoothers are further discussed in Section 5.4. It is of value to understand how the initial tracking is usually done to better understand how this might affect the system.

3.2 Data structure

The data from the preprocessing is received in a Matlab-struct containing an, for the purposes of this project, abundance of information. The data structure contains information in a categorical structure displayed in Figure 3.1. Each slot in the bottom of the structure is a grid of values. Each row of those grids contains values for a point in time. Objects are identified with the grid in the id slot and on the same coordinates as a certain object id in the id grid, the corresponding states and attributes of that object may be found in the other grids.

The structure in Figure 3.1 is mostly self explanatory but some clarifications may aid understanding. First, the coordinate system has to be defined to understand how to interpret variables such as long (longitudinal), lat (latitudinal) and heading. These variables are to be interpreted as positions and angular heading in relation to the position and heading of the host-vehicle, Figure 3.2 shows how these variables are defined. The variable onRoad is a boolean which indicates if the object is on the road or not. A variable only available in the LIDAR data is the referencePos variable which indicates on what side of a detected vehicle the tracked point is placed. Both sensors tracks the middle of the closest side, for the radar this is almost always the front or the back since the radar used is a forward looking radar with a cone-shaped field of view. For the LIDAR this can differ since it has a field of view covering the full 360° around the vehicle. The referencePos variable has been used to compensate for the change of tracked point that occurs when the nearest side changes e.g. when the tracked side changes from front to right side to the rear of a tracked vehicle overtaking on the left side. The stdDev gives the standard deviation for a specific measurement, this is an output generated by the preprocessing, and is used when matching using Mahalanobis distance. Mahalanobis is further described in Section 4.2.

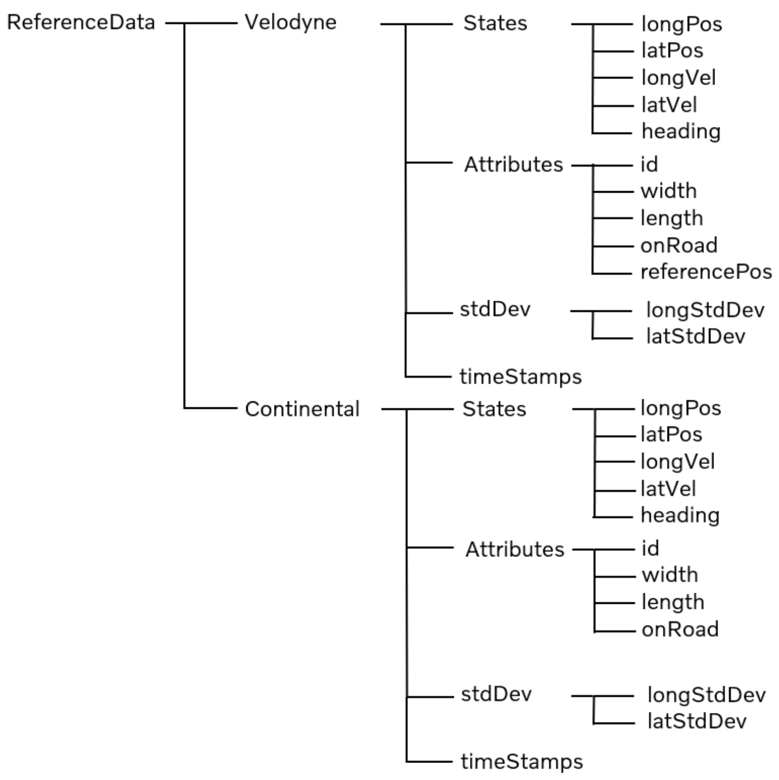


Figure 3.1: Data structure. Some of these are directly measured but most of them are calculated in the preprocessing.

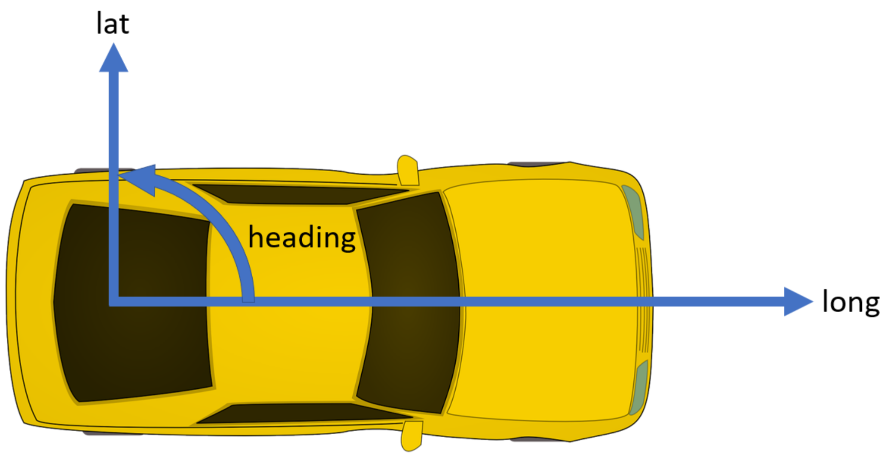


Figure 3.2: Definition of the coordinate system with origin in the middle of the rear axle. The heading is zero if forward facing and positive in the direction of the arrow

3.3 Implementation

The structure of the input data means that in order to get all data for a specific detection it is necessary to look through all matrices, this made it difficult and inefficient to work with. Therefore the first thing created was a data sorting code block, a track builder. The purpose of the track builder was to sort out all the individual tracks and creating a data structure for each track. This could be done with their ID:s that was assigned by the preprocessing. From the matrix structure the data that was required for the algorithm could be retrieved and the rest could be left out for the time being.

Something that was also implemented was a way to create filters for which tracks to save. For example a filter was used to filter out tracks that was seen for a certain amount of time. This could be used both for testing purposes and for sorting out noise. For testing purposes it is preferable to only sort out the object tracks seen for more than a couple of seconds. This gives a much smaller data set to work with and sorts out possible misreading. Using a smaller data set makes it possible for a human to check results compared to a video and has been necessary when developing the algorithm. Filtering out unwanted tracks also reduces the computational load which makes the process more effective. The filtering can also be done using other parts of the state such as position, velocity, but also the object attributes such as width, length or object classification.

4
Track association

Sensor fusion is a valuable tool but it is important that the correct points/features get matched together before any fusion algorithm is applied. If properties of different objects are mistaken as properties belonging to the same object the results of the fusion will be useless. When doing fusion of coordinate data it is valuable to have some kind of distance metric to be able to calculate how close two object detections from different sensors are to each other. The resulting distance is then used to determine if two detections are actually of the same object. Two of the metrics that may be used for matching are Euclidean and Mahalanobis distance [15], which will be introduced in the following sections. The distances calculated using these methods are not distances as such but rather a metric of deviations over all variables of the state expressed as a single metric.

4.1 Euclidean Distance

The standard approach to measure how similar two detections are to each other is the Euclidean distance between them. The Euclidean distance is calculated as:

$$d_{ED} = \sqrt{\left(\hat{\mathbf{x}}_i - \hat{\mathbf{x}}_j\right)^T \left(\hat{\mathbf{x}}_i - \hat{\mathbf{x}}_j\right)}, \tag{4.1}$$

where $\hat{\mathbf{x}}_i$ and $\hat{\mathbf{x}}_j$ are the state estimates from the two different detections. The Euclidian distance is used to calculate similarities between detections and determine which detections are most similar to each other. If the distance is under a certain threshold the detections are likely to be of the same object. This method works well assuming that there is some distance between the objects being detected. However, if several objects are close to each other in the state space, there can be some mismatching which may cause problems.

4.2 Mahalanobis Distance

A alternative distance metric first presented in [16] called the Mahalanobis distance is calculated as:

$$d_{MD} = \sqrt{\left(\hat{\mathbf{x}}_i - \hat{\mathbf{x}}_j\right)^T \left(\mathbf{P}_i + \mathbf{P}_j\right)^{-1} \left(\hat{\mathbf{x}}_i - \hat{\mathbf{x}}_j\right)}, \tag{4.2}$$

where the $\mathbf{P}_i$ and $\mathbf{P}_j$ are the covariance matrices corresponding to the two estimates $\hat{\mathbf{x}}_i$ and $\hat{\mathbf{x}}_j$. This distance metric is similar to the Euclidean but takes into account the covariances, which helps with matching not only the distance but also similarities in uncertainties. The Mahalanobis distance uses more of the available information to produce a valid match when there are many adjacent detections in the same area.

4.3 Implementation

When implementing the matching algorithm it is first important to look at the data available and how the matching may best be performed. In this case, as can be seen in Chapter 3, the data is tracks that has a lot of information about the objects over the time covered by that data set. From all available data one has to retrieve the most easily distinguishable data. In this case the position in 2D space is easily distinguishable data since two different vehicles will never be in the same location at the same time. The Euclidian or Mahalanobis distance for simultaneous tracks are calculated for all timestamps covered by both tracks. By then taking the root mean square (RMS) of all those state space distances a similarity score is derived. This score is then used to match tracks from the two sensors. This is described in Equation 4.3, where n is the number of data points in the track and d is the distance between the two tracks at that point in time.

$$RMS = \sqrt{\frac{1}{n}\left(d_1^2 + d_2^2 + \cdots + d_n^2\right)} \tag{4.3}$$

In this case the RMS value can be calculated with either the Euclidean or the Mahalanobis distance. The tracks are only matched over the time span during which they overlap which may result in many different shorter tracks from one sensor being matched to a longer track of the other sensor as long as the shorter tracks are all active during separate times. There are also tracks when the RMS value is high for all other tracks meaning that particular track does not have any match. This is very rare for tracks that have been seen for a longer time since the other sensor usually catches this object for at least some time, but can happen if the vehicle or object being detected is outside the field of view of the other sensor.

A simplified pseudocode describing the implemented matching solution can be seen in Algorithm 1. The function is implemented with an option on which states to use for matching *matchPosVel* which if true also matches based on velocity. Most commonly it only uses position but on occasion it is useful to also use the velocity when matching detections. It is vital that both inputs are synced in time so that only simultaneous detections are compared. This sync is made by the function *syncCurrentTracks* which returns the next two indicies where data should be retrieved. Then the distance between these two points is calculated with the distance function which can calculate the distance using either Euclidian or Mahalanobis distance. When either of the tracks ends no new distances are calculated, instead the root mean square error is checked against the current best match to see if the current match is better or worse. If the current match is better the algorithm saves that track pair as a match for fusion.

When every track have been checked against all the other tracks and their respectively best match has been found, the matched tracks are combined and become the input for the fusion algorithm. This combination simply takes both tracks and outputs one track with all data points in ascending order according to their time stamps. The resulting combined track has a higher number of data points from the radar since the update frequency for the radar is higher than that of the LIDAR. The data is sorted and combined so that the fusion algorithm made to be applicable to a single track may be applied to the new combined track without any changes. This solution allows the same code used for generating the covariance matrices used for data matching to be used to perform sensor fusion without requiring duplicated code.

Algorithm 1 Matching tracks

```
1:  $lidarTracks \leftarrow lidarPos$
2:  $radarTracks \leftarrow radarPos$
3:  if $matchPosVel = TRUE$ then
4:      $lidarTracks \leftarrow [lidarTracks, lidarVel]$
5:      $radarTracks \leftarrow [radarTracks, radarVel]$
    end
6:  for $i = 1 : nrOfRadarTracks$ do
7:      for $j = 1 : nrOfLidarTracks$ do
8:          $[rC, lC] \leftarrow syncCurrentTracks()$
9:          $radarTrack \leftarrow radarTracks(i)$
10:         $lidarTrack \leftarrow lidarTracks(j)$
11:         while $bothTracksStillActive()$ do
12:             $d \leftarrow [d, distance(radarTrack(rC)), lidarTrack(lC))]$
            end
13:         $d \leftarrow \sqrt{mean(d^2)}$
14:         $saveIfBestMatch(d)$
        end
    end
```

5 Sensor fusion

Sensor fusion can be described as merging data streams from multiple sensors to one unified output with all available data incorporated. Sensor fusion is an excellent way of making use of the complementary strength of several sensors in almost any application since it has many advantages when compared to processing each data source independently [17],[18]. The main advantages of sensor fusion:

- **Improved confidence:** When more than one sensor detects an object in the same location the confidence in that detection being true will increase. The confidence increases since it is unlikely that two or several sensors observing the same metric deviates in a similar way simultaneously, and thus the confidence increases.
- **Increased coverage:** Areas outside the field of view of one sensor may still be within the field of view of another sensor. The active sensor can thus provide input from that "hidden" area which gives the system as a whole the same field of view as the integrated sensors combined. Once the object becomes visible from both sensors the data streams are merged.
- **Improved accuracy:** When fusing data from multiple independent sensors the output can rely more on the measurements known to be most accurate. An example of how this may be achieved is by putting emphasis on longitudinal measurements while trusting less in latitudinal measurements from the radar, based on the known performance of that sensor.
- **Robustness:** Additional sensors contributing to the same measurement gives a certain redundancy where system performance can be maintained even if one of the sensors were to fail. This redundancy also keeps the system operational even if interference occurs since different kinds of sensors will be affected in different ways by environmental factors and other disturbances.

5.1 Levels of fusion

The fusion can be done in many different ways and on different levels. One way of describing the different levels of fusion is by dividing sensor fusion into three levels: data fusion, feature fusion, and decision fusion. This categorization may be broken down even further with a more detailed description of the gray zones between these three levels. In the following paragraphs, various forms of fusion are all grouped and described according to the different levels of fusion defined in [19].

5.1.1 Data fusion

The data fusion is the lowest level of fusion. One example of this would be how the human eyes fuse color information, just fusing the raw data from every color sensitive cone cell within the retina into a fused output that humans perceive as color vision. Data fusion on this level requires the sensor data to be very similar regarding compatible data rates, data dimensions, and formats. Performing data fusion with the same type of sensors can be convenient while trying to fuse the data from two different sensor types can be quite tricky. The sensor data would probably need some preprocessing since the data can be given in completely different formats and units, hence the fusion would no longer be in the lowest level of sensor data fusion.

There is also a mixed level in between the data fusion and the next level, feature fusion, where the input is data, but the output is on the feature level, see next section on feature fusion. One example of this fusion would be to fuse the intensities from two infrared bands of a multispectral scanner to compute the temperature of a surface.

5.1.2 Feature fusion

Feature fusion is when the fusion algorithm works with feature inputs and outputs instead of the raw sensor data. This can, for example, be all the properties of dynamic objects that can be deduced from raw sensor data. For example using the velocity from one sensor and the position from another to give the object a more detailed description.

Also on this level, there can be some connection to the next level: decision fusion described in the following section. For example in pattern recognition systems where the algorithm takes feature inputs from different sensors and then sends them through a classification network that outputs a classification, the decision to classify an object as such is based on features and the decision is the output.

5.1.3 Decision fusion

Decision fusion is fusion of a higher level than the data or feature fusion, to fuse the data on the decision level is always a feasible way of performing sensor fusion. Even if the sensors outputs different raw data and different features it can still be fused on the top level since the data can be preprocessed and transformed into a format suitable for fusion. Although this does require that decisions were made further down the chain either by the sensors independently or by some lower level fusion so that they output some kind of decision.

An example of decision fusion would be object classification where two or more sensors identify a single object and each independently classifying it as a car. The decision fusion algorithm would then assign the fused object to the car class based on decisions made with partial information further down the chain.

5.2 Recursive Bayesian estimation

Recursive Bayesian estimation, also known as Bayesian filtering, is a recursive method for estimating parameters with the use of prior information about that parameter and measurements from at least one sensor, but commonly several sensors see, e.g., [20]. The parameters of interest in a traffic environment would be at least: position, velocity, and direction in which surrounding objects are heading. This state would be described by a state x_k which is a vector that holds all information for one object at the discrete time k.

The measurements/observations from the sensors are all stored in a similar state, which also is a vector y_k with the same discrete time k. To perform Bayesian filtering all previous observations must be considered: $\mathrm{Y}_{1:k}$ is a matrix which contains all the measured data from system start $k = 1$

$$\mathrm{Y}_{1:k} = \{\mathrm{y}_1, \mathrm{y}_2, \ldots, \mathrm{y}_k\} \tag{5.1}$$

up until current time k. Two assumptions are made to perform Bayesian estimations, see, e.g., [21] where the first is that the state vectors stochastic process fulfills the Markov property. The Markov property refers to a memory-less model function where the future state only depends on the current state and not on all previous states, which is defined as

$$p\left(\mathbf{x}_k|\mathbf{x}_{k-1}, \ldots, \mathbf{x}_0\right) = p\left(\mathbf{x}_k|\mathbf{x}_{k-1}\right). \tag{5.2}$$

The second assumption that must be fulfilled is that current observation/measurement y_k is only dependent on the current state x_k, defined as

$$p\left(\mathbf{y}_k|\mathbf{x}_k, \ldots, \mathbf{x}_0\right) = p\left(\mathbf{y}_k|\mathbf{x}_k\right), \tag{5.3}$$

which is how most sensors work when no extra post processing is performed. For processes where these two assumptions are fulfilled, Bayesian filtering using recursive Bayesian estimations is applicable.

The Bayesian filter works recursively towards finding the posterior probability density function $p\left(\mathbf{x}_k|\mathbf{Y}_{1:k}\right)$. The posterior probability density function describes the probabilistic density distribution of the state $\mathbf{x}_k$, given all prior and current measurements $\mathbf{Y}_{1:k}$. From the probability density function the estimate $\hat{\mathbf{x}}_{k|k}$ can be computed based on different optimization criteria. One criterion could be the most likely value of $\mathbf{x}_k$ and it is called MAP, maximum a posteriori. MAP is defined as

$$\hat{\mathbf{x}}_{k|k} = \arg\max_{\mathbf{x}} p\left(\mathbf{x}_k|\mathbf{Y}_{1:k}\right), \tag{5.4}$$

where the posterior probability density function is used to generate the most likely state estimation.

The following are the necessary equations and definitions that are needed to compute the $p\left(\mathbf{x}_k|\mathbf{Y}_{1:k}\right)$ with the posterior density function from time $k-1$ together with the measurements from time k [20], [21]. The measurement data is split up into two: the current measurement and all the earlier ones. The posterior probability density function then takes the form:

$$p\left(\mathbf{x}_k|\mathbf{Y}_{1:k}\right) = p\left(\mathbf{x}_k|\mathbf{y}_k, \mathbf{Y}_{1:k-1}\right), \tag{5.5}$$

where the state is estimated based on current and all previous measurements. By using Bayes' law it can be rewritten as:

$$\begin{aligned} p\left(\mathbf{x}_k|\mathbf{y}_k, \mathbf{Y}_{1:k-1}\right) &= \frac{p\left(\mathbf{y}_k|\mathbf{x}_k, \mathbf{Y}_{1:k-1}\right) p\left(\mathbf{x}_k|\mathbf{Y}_{1:k-1}\right)}{p\left(\mathbf{y}_k|\mathbf{Y}_{1:k-1}\right)} \\ &= \frac{p\left(\mathbf{y}_k|\mathbf{x}_k\right) p\left(\mathbf{x}_k|\mathbf{Y}_{1:k-1}\right)}{p\left(\mathbf{y}_k|\mathbf{Y}_{1:k-1}\right)}, \end{aligned} \tag{5.6}$$

where the assumption that no new observations depends on earlier measurements is used to form the second row, this follows the assumption made in Equation 5.3. The term $p\left(\mathbf{y}_k|\mathbf{x}_k\right)$ requires knowledge of how to model the sensors with their physical properties. The sensors are modelled as

$$\mathbf{y}_k = h_k\left(\mathbf{x}_k, \mathbf{w}_k\right) \tag{5.7}$$

and can be both linear and nonlinear and has a measurement noise $\mathbf{w}_k$. The sensor model basically describes how any given state will look from the perspective of the sensors. The term $p\left(\mathbf{x}_k|\mathbf{Y}_{1:k-1}\right)$ is the predicted probability density and the denominator is there for normalization purposes. The denominator $p\left(\mathbf{y}_k|\mathbf{Y}_{1:k-1}\right)$ defines the probability of a measurement y_k given all previous measurements, however this term is not always used for describing posterior probability density and if left out the posterior probability density can be described as:

$$\text{Posterior} \propto \text{ Prior } \times \text{Likelihood}. \tag{5.8}$$

To be able to predict the next state from previous measurements and get the previously mentioned posterior probability density $p\left(\mathbf{x}_k|\mathbf{Y}_{1:k-1}\right)$ the Chapman-Kolmogorov equation is used:

$$\begin{aligned} p\left(\mathbf{x}_k|\mathbf{Y}_{1:k-1}\right) &= \int p\left(\mathbf{x}_k, \mathbf{x}_{k-1}|\mathbf{Y}_{1:k-1}\right) d\mathbf{x}_{k-1} \\ &= \int p\left(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{Y}_{1:k-1}\right) p\left(\mathbf{x}_{k-1}|\mathbf{Y}_{1:k-1}\right) d\mathbf{x}_{k-1} \\ &= \int p\left(\mathbf{x}_k|\mathbf{x}_{k-1}\right) p\left(\mathbf{x}_{k-1}|\mathbf{Y}_{1:k-1}\right) d\mathbf{x}_{k-1}. \end{aligned} \tag{5.9}$$

Here one of the earlier mentioned assumptions is used, the assumption being that the state vectors stochastic process fulfills the Markov property, that the future state of the process depends only on the current state, see Equation 5.2. The term $p\left(\mathbf{x}_{k-1}|\mathbf{Y}_{1:k-1}\right)$ is the posterior density function at the time $k-1$. The term $p\left(\mathbf{x}_k|\mathbf{x}_{k-1}\right)$ is the density that describes how the state transitions from one point in time to another. To accurately predict a state transition requires knowledge of the system and a set of equations that is capable of representing how the system is likely to behave in reality. The mathematical model used for this is usually referred to as a motion model is defined as

$$\mathbf{x}_k = f_{k-1}\left(\mathbf{x}_{k-1}, \mathbf{v}_{k-1}\right), \tag{5.10}$$

where f can be linear or non-linear set of equations describing the state transitions of the process. The term $\mathbf{v}_{k-1}$ is the process noise that is added and in effect accounts for inaccuracies in motion model. In the case where the variables are normally distributed and the transitions linear, recursive Bayesian estimations are performed using a Kalman filter, described in next section.

5.3 Kalman Filter

The Kalman filter, first developed in late 1950s by Rudolf E. Kalman [14], is a signal filter that allows a reasonable state prediction to be made even if the input describing that state is very noisy. The Kalman filter achieves this by making a few assumptions about the system, it assumes that both process model and measurement model are both linear with known expected values and covariances. With these assumptions the state $\mathbf{x}_k$ and the observations of that state $\mathbf{y}_k$ can be defined as

$$\mathbf{x}_k = \mathbf{F}_{k-1}\mathbf{x}_{k-1} + \mathbf{v}_{k-1} \tag{5.11}$$

$$\mathbf{y}_k = \mathbf{H}_k\mathbf{x}_k + \mathbf{w}_k, \tag{5.12}$$

where $\mathbf{v}_{k-1} \sim \mathcal{N}\left(\mathbf{0}, \mathbf{Q}_{k-1}\right)$ and $\mathbf{w}_k \sim \mathcal{N}\left(\mathbf{0}, \mathbf{R}_k\right)$ are Gaussian distributions with mean zero and covariances $\mathbf{Q}_{k-1}$ and $\mathbf{R}_k$ [20]. The process noise covariance $\mathbf{Q}$ contains information about the noise within the process. Along the diagonal of the matrix the entries represents the variance within each state e.g. how uncertain the process model is for that state variable. The off-diagonal entries in the covariance matrix represents the cross covariance between state variables e.g. how uncertainties in one state variable influences the other variables. The measurement noise covariance $\mathbf{R}$ contains information regarding the noise in measurements. Assuming that the prior density of the state $p\left(\mathbf{x}_0\right)$ is also Gaussian, so will the resulting posterior probability density function. A system fulfilling all of the above assumptions may be optimally filtered using the Kalman filter.

When using a Kalman filter, the posterior probability density at time $k-1$ is calculated from

$$p\left(\mathbf{x}_{k-1}|\mathbf{Y}_{1:k-1}\right) = \mathcal{N}\left(\mathbf{x}_{k-1}; \hat{\mathbf{x}}_{k-1|k-1}, \mathbf{P}_{k-1|k-1}\right), \tag{5.13}$$

where $\hat{\mathbf{x}}_{k-1|k-1}$ is the state estimate at time $k-1$ and $\mathbf{P}_{k-1|k-1}$ is the covariance matrix defining the uncertainties in that estimate. In the first step, the prediction step, the posterior (5.13) which is the best possible estimate using all available data is propagated through the process model to compute an estimate of $\mathbf{x}_k$ using all the data gathered until that point in time. Because a Gaussian density is completely described by its mean and covariance it is enough to calculate these parameters.

The mean and covariance of the predicted posterior state are thus calculated from the current best state estimate as:

$$\hat{\mathbf{x}}_{k|k-1} = \mathbf{F}_{k-1}\hat{\mathbf{x}}_{k-1|k-1} \tag{5.14}$$

$$\mathbf{P}_{k|k-1} = \mathbf{F}_{k-1}\mathbf{P}_{k-1|k-1}\mathbf{F}_{k-1}^T + \mathbf{Q}_{k-1}. \tag{5.15}$$

Next in the measurement update step, a measurement is made and the predicted state and the covariance are updated using that measurement $\mathbf{y}_k$. The result of this update step is a Gaussian posterior density $\mathcal{N}\left(\mathbf{x}_k; \hat{\mathbf{x}}_{k|k}, \mathbf{P}_{k|k}\right)$ computed according to:

$$\mathbf{S}_k = \mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}_k^T + \mathbf{R}_k \tag{5.16}$$

$$\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}_k^T\mathbf{S}_k^{-1} \tag{5.17}$$

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k\left(\mathbf{y}_k - \mathbf{H}_k\hat{\mathbf{x}}_{k|k-1}\right) \tag{5.18}$$

$$\mathbf{P}_{k|k} = \mathbf{P}_{k|k-1} - \mathbf{K}_k\mathbf{S}_k\mathbf{K}_k^T. \tag{5.19}$$

This process is iterated for every time step k to find the posterior probability density and thus the best state estimate at that point in time.

Basic Kalman filters are excellent in situations where the mentioned assumptions are valid but it struggles for non-linear models. For processes that are not approximately linear it might be more suitable to use an extended, unscented or cubature Kalman filter described in the following sections [20].

5.3.1 Extended Kalman Filter

The Extended Kalman filter (EKF) is an extension to the basic Kalman filter to handle non-linear filtering problems [20]. For a system where both the process and sensor models are nonlinear the state and measurement can be described by:

$$\begin{aligned} \mathbf{x}_k &= f\left(\mathbf{x}_{k-1}\right) + \mathbf{v}_{k-1} \\ \mathbf{y}_k &= h\left(\mathbf{x}_k\right) + \mathbf{w}_k \end{aligned}, \tag{5.20}$$

where $\mathbf{v}_{k-1} \sim \mathcal{N}\left(\mathbf{0}, \mathbf{Q}_{k-1}\right)$ and $\mathbf{w}_k \sim \mathcal{N}\left(\mathbf{0}, \mathbf{R}_k\right)$ are Gaussian distributions with mean zero and covariances $\mathbf{Q}_{k-1}$ and $\mathbf{R}_k$. It linearizes by a first order Taylor expansion around the expected value. Both the motion and measurement model can be calculated with

$$\hat{\mathbf{F}} = \left[\nabla_{\mathbf{x}_{k-1}} f\left(\mathbf{x}_{k-1}\right)^T\right]^T \Bigg|_{\mathbf{x}_{k-1}=\hat{\mathbf{x}}_{k-1|k-1}}, \tag{5.21}$$

$$\hat{\mathbf{H}} = \left[\nabla_{\mathbf{x}_k} h\left(\mathbf{x}_k\right)^T\right]^T \Bigg|_{\mathbf{x}_k=\hat{\mathbf{x}}_{k|k-1}}, \tag{5.22}$$

where

$$\nabla_{\mathbf{x}_k} \triangleq \left[\frac{\partial}{\partial \mathbf{x}_k(1)}, \ldots, \frac{\partial}{\partial \mathbf{x}_k(n)}\right]^T \tag{5.23}$$

is the partial derivations made [21]. The new equation for expected value and covariance then equates to

$$\hat{\mathbf{x}}_{k|k-1} = f(\hat{\mathbf{x}}_{k-1|k-1}) \tag{5.24}$$

$$\mathbf{P}_{k|k-1} = \hat{\mathbf{F}}\mathbf{P}_{k-1|k-1}\hat{\mathbf{F}}^T + \mathbf{Q}_{k-1}. \tag{5.25}$$

and $\hat{\mathbf{H}}$ is used in Equations 5.16-5.19 to create the now linearized filter update equations. The EKF is a good and easy to understand filter for handling non-linear systems since it works with standard linearization methods [20]. But as with all linearizations it only performs well if the problem does not have to many non-linearities. There can also be cases when the models are not easily differentiable and can therefore cause problems. Another weakness of EKF is that it becomes slow and computationally heavy to use as the number of variables in the state increase, as the number of partial derivatives to calculate increases with the square of dimensions in the state.

5.3.2 Sigma point methods

For non-linear problems where an EKF becomes inefficient there are other methods that has been proven to work effectively [22]. One possible method is the σ-point filter, which works by creating a fixed number of points with a zero mean distributed symmetrically according to the covariance matrix. By then adding the latest state estimate $\hat{x}$ to those points they get symmetrically distributed around $\hat{x}$. How they are distributed depends on the variables in Equation 5.31 and on which kind of σ-point filter is used. The two kinds of σ-point filters that has been used in this project are the Unscented Kalman filter and the Cubature Kalman filter, these methods will be further explained in the following sections.

Both of these σ-point methods are based on the unscented transformation [20],[22], the use of which is motivated by two ideas. The first idea is that it is easier to approximate a probability distribution than it is to approximate an arbitrary non-linear function or transformation. The second idea is that when a nonlinear mapping function $\mathbf{y} = h(\mathbf{x})$ maps a variable with a Gaussian distribution with known mean and covariance $\mathbf{x} \sim \mathcal{N}(\hat{\mathbf{x}}, \mathbf{P})$, the expected output value of that function can be approximated. The approximation works by evaluating $h(\mathbf{x})$ at a number of discrete σ-points according to

$$\hat{\mathbf{y}} = \mathbb{E}\{h(\mathbf{x})\} \approx \sum_{i=0}^{2n} W_m^i h\left(\mathcal{X}^{(i)}\right), \tag{5.26}$$

where $\mathcal{X}^{(i)}$ are σ-points and W_m^i are the mean weights associated with those points. The expected output $\hat{\mathbf{y}}$ is thus calculated as an aggregate of the transformed σ-points. By weighting the influence of each σ-point on the aggregate using the calculated weight-term W_c^i the expected mean $\hat{\mathbf{y}}$ may be used to compute the corresponding covariance according to:

$$\mathrm{Cov}\{\hat{\mathbf{y}}\} \approx \sum_{i=0}^{2n} W_c^i \left(h\left(\mathcal{X}^{(i)}\right) - \hat{\mathbf{y}}\right)\left(h\left(\mathcal{X}^{(i)}\right) - \hat{\mathbf{y}}\right)^T + \mathbf{R}_{k-1}. \tag{5.27}$$

By using this approach both the expected value and the covariance of a nonlinear mapping function with a Gaussian variable as input can be calculated without the use of derivatives.

5.3.2.1 Unscented Kalman Filter

A very important aspect of using the unscented Kalman filter is the choice of σ-points. The first σ-point is placed at the last estimated state so that $\mathcal{X}^{(0)} = \mathbf{x}_{k-1}$. The remaining $2n$ σ-points may be placed in various patterns, however, for all the results presented in Chapter 7 they have been placed according to

$$\begin{aligned} \mathcal{X}_{k-1}^{(0)} &= \mathbf{x}_{k-1} \\ \mathcal{X}_{k-1}^{(i)} &= \mathbf{x}_{k-1} + (\sqrt{(n+\lambda)P(:,i)})_i \quad \text{for } i = 1, \ldots, n \\ \mathcal{X}_{k-1}^{(i)} &= \mathbf{x}_{k-1} - (\sqrt{(n+\lambda)P(:,i)})_{i-n} \quad \text{for } i = n+1, \ldots, 2n, \end{aligned} \tag{5.28}$$

where they are symmetrically distributed around the first σ-point. By distributing the points so that each of the $2n$ points deviation from the mean $\mathbf{x}_{k-1}$ is cancelled out by a corresponding point on the opposite side of that estimated mean. The joint mean of all σ-points will thereby be equal to the estimated state $\mathbf{x}_{k-1}$. λ is a variable used for tuning σ-point placement and $P(:,i)$ is a column vector from the covariance matrix, further tuning parameters are defined in Equation 5.31.

The σ-points are then passed through the motion model which results in a predicted distribution of σ-points according to

$$\hat{\mathcal{X}}_k^{(i)} = f\left(\mathcal{X}_{k-1}^{(i)}\right), \quad i = 0, \ldots, 2n. \tag{5.29}$$

After the points have been passed through the motion model they are weighted before contributing to the prediction. The weight given to each σ-point is calculated according to

$$\begin{aligned} W_m^{(0)} &= \frac{\lambda}{n+\lambda} \\ W_c^{(0)} &= W_m^{(0)} + \left(1 - \alpha^2 + \beta\right) \\ W_m^{(i)} = W_c^{(i)} &= \frac{1}{2(n+\lambda)} \quad \text{for } i = 1, \ldots, 2n. \end{aligned} \tag{5.30}$$

The weights calculated above are generated from a set of variables λ, α, β and n, where n is the number of dimensions of the state. However the first three variables may be chosen and thereby used as tuning parameters for adjusting the spread of the σ-points and thus the performance of the filter. The parameters should however be restricted within certain ranges and the λ value is a calculated value, tuned through combination of the other parameters:

$$\begin{aligned} \kappa &\geq 0 \\ \alpha &\in (0, 1] \\ \lambda &= \alpha^2(n+\kappa) - n \\ \beta &= 2. \end{aligned} \tag{5.31}$$

It is suggested to restrict the tuning parameters to the ranges defined above to achieve stable and predictable performance of the filter [23].
The predicted state and corresponding covariance are then calculated according to:

$$\begin{aligned} \hat{\mathbf{x}}_k &= \sum_{i=0}^{2n} W_m^{(i)} \hat{\mathcal{X}}_k^{(i)} \\ \hat{\mathbf{P}}_k &= \sum_{i=0}^{2n} W_c^{(i)} \left(\hat{\mathcal{X}}_k^{(i)} - \hat{\mathbf{x}}_k\right)\left(\hat{\mathcal{X}}_k^{(i)} - \hat{\mathbf{x}}_k\right)^\top + \mathbf{Q}_{k-1}. \end{aligned} \tag{5.32}$$

The update step of the UKF first generates a new set of σ-points, the difference this time is that the predicted mean and covariance is used. The generated σ-points are thus spread symmetrically around the state predicted in the prediction step of the filter. The new σ-points are then passed through the measurement model to generate the predicted measurement according to the following equation.

$$\hat{\mathcal{Y}}_k^{(i)} = h\left(\hat{\mathcal{X}}_k^{(i)}\right), \quad i = 0, \dots, 2n \tag{5.33}$$

The covariances of the predicted measurement is described by

$$\begin{aligned} \hat{\mathbf{y}}_k &= \frac{1}{2n} \sum_{i=1}^{2n} \hat{\mathcal{Y}}_k^{(i)} \\ \mathbf{S}_k &= \sum_{i=0}^{2n} W_i^{(c)} \left(\hat{\mathcal{Y}}_k^{(i)} - \hat{\mathbf{y}}_k\right)\left(\hat{\mathcal{Y}}_k^{(i)} - \hat{\mathbf{y}}_k\right)^\top + \mathbf{R}_k \\ \mathbf{C}_k &= \sum_{i=0}^{2n} W_i^{(c)} \left(\hat{\mathcal{X}}_k^{(i)} - \hat{\mathbf{x}}_k\right)\left(\hat{\mathcal{Y}}_k^{(i)} - \hat{\mathbf{y}}_k\right)^\top, \end{aligned} \tag{5.34}$$

where $\mathbf{S}_k$ is the covariance of the predicted measurement and $\mathbf{C}_k$ is the cross-covariance between the predicted state and measurement.

Finally these covariances are used to compute the Kalman gain $\mathbf{K}_k$ which essentially defines how much the model should rely on the new measurement by taking the difference between those and the predicted measurements. Taking all of the above into account including the newest measurement in $\mathbf{y}_k$ the new state is calculated along with its corresponding covariance

$$\begin{aligned} \mathbf{K}_k &= \mathbf{C}_k \mathbf{S}_k^{-1} \\ \hat{\mathbf{x}}_k &= \hat{\mathbf{x}}_{k-1} + \mathbf{K}_k \left(\mathbf{y}_k - \hat{\mathbf{y}}_k\right) \\ \mathbf{P}_k &= \mathbf{P}_{k-1} - \mathbf{K}_k \mathbf{S}_k \mathbf{K}_k^\top. \end{aligned} \tag{5.35}$$

The unscented Kalman filter is a variation of estimation methods usually referred to as σ-point methods [24]. The unscented Kalman filter is one of the σ-point methods used in this project and it uses $\mathbf{2n+1}$ points, while the method described in the next section uses only $\mathbf{2n}$ points which has a certain influnce on performance.

5.3.2.2 Cubature Kalman Filter

The Cubature Kalman filter (CKF) works very similiar to the UKF with the two differences being how many σ-points that are created and the distribution of those points [20]. The σ-points are distributed around the estimated mean value according to

$$\begin{aligned} \mathcal{X}_{k-1}^{(i)} &= \mathbf{x}_{k-1} + \sqrt{n}\sqrt{\mathbf{P}(:,\mathbf{i})_{k-1}}, \quad i = 1, \dots, n, \\ \mathcal{X}_{k-1}^{(i)} &= \mathbf{x}_{k-1} - \sqrt{n}\sqrt{\mathbf{P}(:,\mathbf{i})_{k-1}}, \quad i = n+1, \dots, 2n, \end{aligned} \tag{5.36}$$

where $\mathbf{P}(:,\mathbf{i})_{k-1}$ is the i:th column of the covariance matrix. The obvious difference between CKF and UKF is that former one does not include $\mathcal{X}_{k-1}^{(0)}$ which for the UKF is the estimated mean, see Equation 5.28. The σ-points are then passed through the motion model according to

$$\hat{\mathcal{X}}_k^{(i)} = f\left(\mathcal{X}_{k-1}^{(i)}\right), \quad i = 1, \dots, 2n. \tag{5.37}$$

After passing the σ-points through the motion model they are weighted and added up to the predicted mean and covariance according to the following equations

$$\begin{aligned} \hat{\mathbf{x}}_k &= \frac{1}{2n}\sum_{i=0}^{2n}\hat{\mathcal{X}}_k^{(i)} \\ \hat{\mathbf{P}}_k &= \frac{1}{2n}\sum_{i=0}^{2n}\left(\hat{\mathcal{X}}_k^{(i)} - \hat{\mathbf{x}}_k\right)\left(\hat{\mathcal{X}}_k^{(i)} - \hat{\mathbf{x}}_k\right)^\top + \mathbf{Q}_{k-1}, \end{aligned} \tag{5.38}$$

where $\mathbf{Q}_{k-1}$ is the process noise covariance.

The update step starts with computing new σ-points around the predicted mean using the predicted covariance, these are then passed through the measurement model to get the predicted measurements

$$\hat{\mathcal{Y}}_k^{(i)} = h\left(\hat{\mathcal{X}}_k^{(i)}\right), \quad i = 1, \dots, 2n. \tag{5.39}$$

First the predicted measurement mean $\hat{\mathbf{y}}$, measurement covariance $\mathbf{S}_k$ and the cross-covariance between state and measurements $\mathbf{C}_k$ are computed as

$$\begin{aligned} \hat{\mathbf{y}}_k &= \frac{1}{2n}\sum_{i=1}^{2n}\hat{\mathcal{Y}}_k^{(i)} \\ \mathbf{S}_k &= \frac{1}{2n}\sum_{i=1}^{2n}\left(\hat{\mathcal{Y}}_k^{(i)} - \hat{\mathbf{y}}_k\right)\left(\hat{\mathcal{Y}}_k^{(i)} - \hat{\mathbf{y}}_k\right)^\top + \mathbf{R}_k \\ \mathbf{C}_k &= \frac{1}{2n}\sum_{i=1}^{2n}\left(\hat{\mathcal{X}}_k - \hat{\mathbf{x}}_k\right)\left(\hat{\mathcal{Y}}_k^{(i)} - \hat{\mathbf{y}}_k\right)^\top. \end{aligned} \tag{5.40}$$

Finally the filtered state $\mathbf{x}_k$ and the covariance $\mathbf{P}_k$ can be calculated with the Kalman filter gain $\mathbf{K}_k$ as

$$\begin{aligned} \mathbf{K}_k &= \mathbf{C}_k\mathbf{S}_k^{-1} \\ \mathbf{x}_k &= \hat{\mathbf{x}}_k + \mathbf{K}_k\left(\mathbf{y}_k - \hat{\mathbf{y}}_k\right) \\ \mathbf{P}_k &= \hat{\mathbf{P}}_k - \mathbf{K}_k\mathbf{S}_k\mathbf{K}_k^\top, \end{aligned} \tag{5.41}$$

where $\mathbf{y}_k$ are the measurements.

5.3.3 Motion Models

The motion model is a representation of the physical system that is being observed. This mathematical representation of the system takes as input the last state estimation and the time that has passed since that estimation was made, it then gives an estimation of the current state based on those inputs. The motion model is the core of the Kalman filter and is what enables predictions of the state to be made before that data becomes available [14]. The motion model can also be referred to with the wider term *process model* since the Kalman filters can be applied to processes other than pure motion. The equations of motion included in the motion model describes how the variables of the state will influence each other over time according to

$$\mathbf{x}_k = f\left(\mathbf{x}_{k-1}\right) + \mathbf{v}_{k-1}, \tag{5.42}$$

which also includes a noise term ($\mathbf{v}_{k-1}$) for the previous time. The set of equations included in the motion model varies but essentially always describes the relations within the state over time in a similar way to an equation of motion describing position in terms of velocity and acceleration over time. The accuracy of the prediction made with the motion model is also used to estimate the accuracy of the motion model, which varies over time as a consequence of the difference between actual measurements and predicted measurements. The motion model picked should accurately reflect the physical system that is being modeled and is therefore very case specific. In the case of modelling the motions of vehicles there are several suggested models identified during research of prior work. The models used over the course this project are described in further detail in the following sections.

5.3.3.1 Constant Acceleration Model

One of the simpler motion models that is applicable in a traffic scenario is the constant acceleration model. This model uses a state with six variables and describes movement within a 2D-plane [25]. The dimensions of the constant acceleration model state are given in the state vector as

$$X = [x, y, \dot{x}, \dot{y}, \ddot{x}, \ddot{y}]^T, \tag{5.43}$$

which is constituted of variables for position, velocity and acceleration in the x and y directions. The motion model is the following set of equations

$$f_{CA}(X) = \begin{bmatrix} x + T\dot{x} + \frac{T^2}{2}\ddot{x} \\ y + T\dot{y} + \frac{T^2}{2}\ddot{y} \\ \dot{x} + T\ddot{x} \\ \dot{y} + T\ddot{y} \\ \ddot{x} \\ \ddot{y} \end{bmatrix} \tag{5.44}$$

which describes the differential equations of motion along two perpendicular axis. In the motion model the first equation describes how the position along the x-axis is influenced over time by velocity and acceleration along that same axis. Line three describes how the velocity changes over time and is equal to the time derivative of the equation describing position. Line five describes how the acceleration along the x-axis is constant over time. This is the kernel of this motion model, the acceleration is assumed to be constant or close to it and changes in acceleration are accounted for by the added noise in Equation 5.42 rather than by the motion model itself which is where the name constant acceleration model is derived from.

5.3.3.2 Augmented Coordinated Turn Model

The augmented coordinated turn model uses a model that makes assumptions about the speed and turn rate rather than acceleration. There are two versions of this motion model, one in Cartesian coordinates and another one in polar coordinates [26]. There are similarities between Cartesian and polar such as the turn rate and speed being assumed to be constant and included as variables in the state. Another shared aspect between these models is the type of motion they describe, vehicles are assumed to move along their heading, which means they can never turn without also moving forward. The kind of motion these models predict is a good approximation of how a majority of vehicles move, without a turning rear-axle, slip-less motion may only occur along an axis perpendicular to and intersecting both front- and rear-axle. Further differences and similarities will be described in the following paragraphs.

Cartesian Coordinates
In the Cartesian coordinate system the coordinated turn model state is given as

$$X = [x, y, \dot{x}, \dot{y}, \omega]^T \tag{5.45}$$

and is similar to the constant acceleration model state but has the fifth state variable ω which describes the turn rate rather than the acceleration within the plane. The Cartesian coordinate version of the motion model used is

$$f_{\text{ACT}}(X) = \begin{bmatrix} x + \frac{\sin(\omega T)}{\omega}\dot{x} - \frac{1-\cos(\omega T)}{\omega}\dot{y} \\ y + \frac{1-\cos(\omega T)}{\omega}\dot{x} + \frac{\sin(\omega T)}{\omega}\dot{y} \\ \cos(\omega T)\dot{x} - \sin(\omega T)\dot{y} \\ \sin(\omega T)\dot{x} + \cos(\omega T)\dot{y} \\ \omega \end{bmatrix}, \tag{5.46}$$

where the turn rate is assumed to be constant and changes in ω will therefore be accounted for by the additive noise in the end of Equation 5.42. That additive noise is also how the model accounts for changes in velocity. The model assumes the absolute velocity to be constant while the distribution over the velocity components in x and y directions varies over time depending on the turn rate.

Polar Coordinates
The polar version of the coordinated turn model is a bit different with the state as

$$X = [x, y, v, \phi, \omega]^T, \tag{5.47}$$

where the position dimensions of the state remains unchanged but the remainder of the state is altered. Instead of having the speed of the vehicle broken down into two velocities there is only one absolute speed. The absolute speed in combination with the fourth state, the heading (ϕ), gives the same information as the third and fourth state variables in the Cartesian version of the motion model. The fifth component of the state also remains unchanged and is the turn rate e.g. how quickly the heading is changing, also known as the yaw rate. The polar coordinate version of the coordinated turn motion model has the equations of motion as:

$$f_{\mathrm{ACT2}}(X) = \begin{bmatrix} x + \left(\frac{2v}{\omega}\right)\sin\left(\frac{\omega T}{2}\right)\cos\left(\phi + \frac{\omega T}{2}\right) \\ y + \left(\frac{2v}{\omega}\right)\sin\left(\frac{\omega T}{2}\right)\sin\left(\phi + \frac{\omega T}{2}\right) \\ v \\ \phi + \omega T \\ \omega \end{bmatrix}. \tag{5.48}$$

This model describes movements very similar to the Cartesian coordinate version. Line four of the motion model defines how the, by the model assumed to be constant, turn rate will influence the heading over time.

5.3.3.3 Bicycle Model

This bicycle model state is more complex and attempts to use more of the information available from the LIDAR, namely the length and width of the detected vehicle [27]. The bicycle model state is defined as

$$X = [x, y, v, \varphi, \theta, \ell, w]^T \tag{5.49}$$

and is similar to the polar coordinated turn motion state with some additional variables, specifically the length and width of the tracked vehicle. It also has the fifth variable of the state as steering angle rather than the typical turn rate The model uses several additional variables not included in the state, but rather employs a couple of equations:

$$d_k = Tv_k \tag{5.50}$$

$$\beta_k = \frac{d_k}{\ell_k^w}\tan(\theta_k) \tag{5.51}$$

$$R_k = d_k/\beta_k \tag{5.52}$$

to define distance covered, turning angle and turning radius respectively.
The bicycle model unlike the other models in this report changes based on the state of the tracked object. While the estimated steering angle differs from zero, in other words when the tracked vehicle is not traveling in a straight line the motion model is described by a set of equations:

$$f_{\mathrm{BC}}(X) = \begin{bmatrix} x - R\sin(\varphi) + R\sin(\varphi + \beta) \\ y + R\cos(\varphi) + R\cos(\varphi + \beta) \\ [v, \varphi + \beta, \theta, \ell, w]^T \end{bmatrix}. \tag{5.53}$$

In this model the position varies according to estimated turning radius R, heading φ and turning angle β.

This model however is limited to estimations where the steering angle differs from zero since a value of zero for turning angle connotes that turning radius approaches infinity, in short the model breaks down. If conditions:

$$\lim_{\theta_k \to 0} \beta_k = 0, \quad \lim_{\theta_k \to 0} R_k = \infty \tag{5.54}$$

are fulfilled the motion model changes to a model more analogous to a constant velocity model. The motion model analogous to a constant velocity model:

$$f_{\mathrm{CS}}\left(\mathbf{x}_k\right) = \begin{bmatrix} x_k + d_k \cos\left(\varphi_k\right) \\ y_k + d_k \sin\left(\varphi_k\right) \\ \left[v_k, \varphi_k, \theta_k, \ell_k, w_k\right]^T \end{bmatrix} \tag{5.55}$$

assumes everything to be constant except the position variables which are influenced by the estimated heading and velocity according to the first two lines in the motion model. The other variables in the state are assumed to be constant and have their changes accounted for by the additive noise.

5.4 Smoothing

All the previous mentioned filtering techniques only considers current and previous measurements. Filters utilizing only previously gathered data is suitable for online applications e.g. controlling a process in real time. The reference system that this project is based upon, however, does not work on-line. The purpose of the algorithm created during this project is to evaluate the internal systems and sensors. All of this is performed after the data is collected and therefore does not need to work on-line. Without the need of filtering data in real-time more methods become applicable: for example smoothing. Smoothing applies much of the logic used for filtering but utilizes future and previous measurements instead of only the current and previous ones [20]. This yields much better results than using just a filter. The probability distribution can be calculated as $p(\mathbf{x}_k|\mathbf{Y}_{1:T})$ for any $k < T$. This T can either be the end time of the whole set or just some time bigger than the current time k.

The kind of smoother that has been used is the Rauch-Tung-Striebel smoother (RTS smoother). The RTS smoother works in two steps, first it performs forward filtering with the use of a Kalman filter. This is called the forward pass and it is performed in the exact same way as when working with only a Kalman filter. The forward pass is then followed by a backward recursion through the whole set, working its way backwards with all measurements and the filtered output. The update equations depends on what kind of Kalman filter that is used. The variations of smoothers used in this project will be described in further detail in the following sections.

5.4.1 Extended Rauch-Tung-Striebel Smoother

The Extended RTS smoother works by using the basic RTS smoother but changing the prediction equations to the first order Taylor series approximations [20]. There is of course the possibility to also use higher order approximations but these methods will not be covered here. The EKF RTS smoothing algorithm first does a forward filtering pass with the use of the standard EKF equations described in 5.3.1. Next it does the backward recursion for $k = T-1, \dots, 0$. Every iteration starts with predicting the state at time step $k+1$ according to

$$\begin{aligned}\hat{\mathbf{x}}_{k+1} &= \mathbf{f}\left(\mathbf{x}_k\right)\\ \hat{\mathbf{P}}_{k+1} &= \mathbf{F}_{\mathbf{x}}\left(\mathbf{x}_k\right)\mathbf{P}_k\mathbf{F}_{\mathbf{x}}^{\top}\left(\mathbf{x}_k\right) + \mathbf{Q}_k,\end{aligned} \tag{5.56}$$

where $\mathbf{F}_{\mathbf{x}}(\mathbf{x_k})$ is the Jacobian matrix of the process model evaluated at $\mathbf{x}_k$. The update equations for time step k then becomes

$$\begin{aligned}\mathbf{G}_k &= \mathbf{P}_k\mathbf{F}_{\mathbf{x}}^{\top}\left(\mathbf{x}_k\right)\left[\hat{\mathbf{P}}_{k+1}\right]^{-1}\\ \mathbf{x}_k^{\mathrm{s}} &= \mathbf{x}_k + \mathbf{G}_k\left[\mathbf{x}_{k+1}^{\mathrm{s}} - \hat{\mathbf{x}}_{k+1}\right]\mathbf{G}_k^{\top}\\ \mathbf{P}_k^{\mathrm{s}} &= \mathbf{P}_k + \mathbf{G}_k\left[\mathbf{P}_{k+1}^{\mathrm{s}} - \hat{\mathbf{P}}_{k+1}\right]\mathbf{G}_k^{\top}.\end{aligned} \tag{5.57}$$

With these update steps it works its way backwards recursively by comparing the smoothed result with the prediction from step $k+1$ to update the k:th step. This will give a smoother result than the one that is acquired from doing only the forward filtering since it has much more information to work with. A smoother result is desired when tracking vehicles since they tend to have a smooth movement.

5.4.2 Unscented Rauch-Tung-Striebel Smoother

The unscented RTS smoother first does a forward filtering pass as described in Section 5.3.2.1 where all the filtered results are calculated. These are then used in the backward recursion, for $k = T-1, \dots, 0$, which starts with calculating σ-points around the filtered mean $\mathbf{x}_k^f$. The filtered mean is derived from the forward pass for the current time step k. These σ-points are generated according to Equation 5.28 and then passed through the process model. Once passed through the process model the σ-points are transformed from current time to the next time step by

$$\hat{\mathcal{X}}_{k+1}^{(i)} = f\left(\mathcal{X}_k^{(i)}\right), \quad i = 0, \dots, 2n, \tag{5.58}$$

to get the transformed points surrounding the predicted mean. The weights are calculated as described by Equation 5.30. With the transformed σ-points $\hat{\mathcal{X}}_{k+1}^{(i)}$ the predicted mean $\hat{\mathbf{x}}_{k+1}$ along with predicted covariance $\hat{\mathbf{P}}_{k+1}$ and cross covariance $\mathbf{D}_{k+1}$ are calculated according to

$$\begin{aligned}\hat{\mathbf{x}}_{k+1} &= \sum_{i=0}^{2n} W_i^{(\mathrm{m})}\hat{\mathcal{X}}_{k+1}^{(i)}\\ \hat{\mathbf{P}}_{k+1} &= \sum_{i=0}^{2n} W_i^{(\mathrm{c})}\left(\hat{\mathcal{X}}_{k+1}^{(i)} - \hat{\mathbf{x}}_{k+1}\right)\left(\hat{\mathcal{X}}_{k+1}^{(i)} - \hat{\mathbf{x}}_{k+1}\right)^{\top} + \mathbf{Q}_k\\ \mathbf{D}_{k+1} &= \sum_{i=0}^{2n} W_i^{(\mathrm{c})}\left(\mathcal{X}_k^{(i)} - \mathbf{x}_k\right)\left(\hat{\mathcal{X}}_{k+1}^{(i)} - \hat{\mathbf{x}}_{k+1}\right)^{\top}.\end{aligned} \tag{5.59}$$

With the predicted mean and covariances the following equations can be solved as

$$\begin{aligned}\mathbf{G}_k &= \mathbf{D}_{k+1}\left[\hat{\mathbf{P}}_{k+1}\right]^{-1}\\ \mathbf{x}_k^{\mathrm{s}} &= \mathbf{x}_k + \mathbf{G}_k\left(\mathbf{x}_{k+1}^{\mathrm{s}} - \hat{\mathbf{x}}_{k+1}\right)\\ \mathbf{P}_k^{\mathrm{s}} &= \mathbf{P}_k + \mathbf{G}_k\left(\mathbf{P}_{k+1}^{\mathrm{s}} - \hat{\mathbf{P}}_{k+1}\right)\mathbf{G}_k^{\top}.\end{aligned} \tag{5.60}$$

Solving the equations above gives the gain $\mathbf{G}_k$, the smoothed mean $\mathbf{x}_k^{\mathrm{s}}$ and the covariance $\mathbf{P}_k^{\mathrm{s}}$ of the smoothed output. Unlike the output of the previously described filtering methods the smoothed output is generated based on all available data and thus likely to deviate less from the true state.

5.4.3 Cubature Rauch-Tung-Striebel Smoother

As with the unscented RTS smoother the Cubature Rauch-Tung-Striebel smoother first performs a forward pass according to the CKF process, explained in Section 5.3.2.2. The filtered results are saved and later used for the backwards smoothing pass. Every iteration starts with calculating σ-points as described in Equation 5.36, the σ-points are then passed through the motion model

$$\hat{\mathcal{X}}_{k+1}^{(i)} = f\left(\mathcal{X}_k^{(i)}\right), \quad i = 1, \ldots, 2n \tag{5.61}$$

to get the transformed σ-points. After passing through the motion model the transformed σ-points are spread around the predicted mean $\hat{\mathbf{x}}_{k+1}$. The spread and position of the transformed σ-points in relation to the predicted mean is then used to calculate the predicted covariance $\hat{\mathbf{P}}_{k+1}$ and cross covariance $\mathbf{D}_{k+1}$ as,

$$\begin{aligned}\hat{\mathbf{x}}_{k+1} &= \frac{1}{2n}\sum_{i=1}^{2n}\hat{\mathcal{X}}_{k+1}^{(i)}\\ \hat{\mathbf{P}}_{k+1} &= \frac{1}{2n}\sum_{i=1}^{2n}\left(\hat{\mathcal{X}}_{k+1}^{(i)} - \hat{\mathbf{x}}_{k+1}\right)\left(\hat{\mathcal{X}}_{k+1}^{(i)} - \hat{\mathbf{x}}_{k+1}\right)^{\top} + \mathbf{Q}_k\\ \mathbf{D}_{k+1} &= \frac{1}{2n}\sum_{i=1}^{2n}\left(\mathcal{X}_k^{(i)} - \mathbf{x}_k\right)\left(\hat{\mathcal{X}}_{k+1}^{(i)} - \hat{\mathbf{x}}_{k+1}\right)^{\top}.\end{aligned} \tag{5.62}$$

Using the predictions above the smoothing gain $\mathbf{G}_k$ is calculated and used to complete the prediction step. The smoothed mean $\mathbf{x}_k^{\mathrm{s}}$ and smoothed covariance $\mathbf{P}_k^{\mathrm{s}}$ can then be calculated according to

$$\begin{aligned}\mathbf{G}_k &= \mathbf{D}_{k+1}\left[\mathbf{P}_{k+1}^{-}\right]^{-1}\\ \mathbf{x}_k^{\mathrm{s}} &= \mathbf{x}_k + \mathbf{G}_k\left(\mathbf{x}_{k+1}^{\mathrm{s}} - \hat{\mathbf{x}}_{k+1}\right)\\ \mathbf{P}_k^{\mathrm{s}} &= \mathbf{P}_k + \mathbf{G}_k\left(\mathbf{P}_{k+1}^{\mathrm{s}} - \hat{\mathbf{P}}_{k+1}\right)\mathbf{G}_k^{\top}.\end{aligned} \tag{5.63}$$

5.5 Implementation

The aim with the implementation of this sensor fusion algorithm has been to find a well performing solution, with increased performance in both accuracy and reliability. This prompts the use of various performance metrics, the details of which are extensively explained in Section 6. Performance has to be compared for different variations of the fusion algorithm to determine which set of models, smoothers and tuning parameters provides the best performance. The ability to quickly test various solutions has been highly prioritized when creating the code. Therefore the code has been been implemented with a modular structure where it is easy to switch between models, parameters, etc.

The code has also been developed with the intention of it being an off-line script applied in post processing. This has made it possible to write code with longer computational time than what would be appropriate if it were meant to be run on-line. When working off-line it is no longer necessary to take the computational time into account when evaluating various algorithms. Assuming the computational time does not become unreasonable, better performance is always beneficial even if the performance gain is small and computational penalty is high.

All of the code has been implemented in Matlab since the data provided by the company came in MAT-files. The fusion algorithm is supposed to be an addition to the existing post processing and, therefore, it has been considered convenient to maintain the file format. Matlab is also a good tool for developing and testing new code with simple tools for visualizations.

The entirety of the implemented tool chain is visualized in Figure 5.1. In the following sections the implementation of the last three boxes of the tool chain will be described and explained.

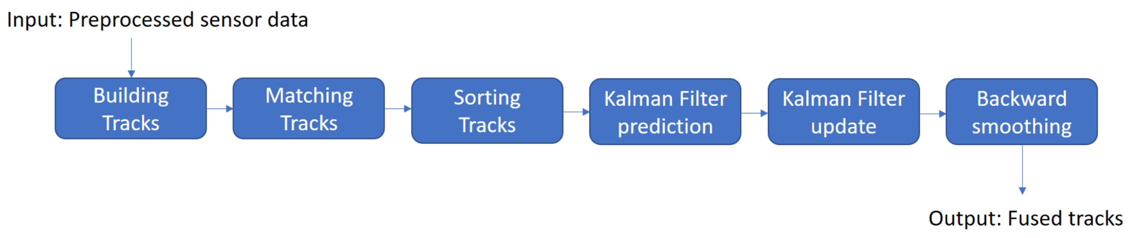


Figure 5.1: A block diagram of sensor fusion tool chain.

5.5.1 Kalman Filter Prediction

The prediction step considers information about the previous step to predict how it is likely to evolve until the current time step. The information needed for this state prediction is:

- prior mean x_{k-1}
- prior covariance P_{k-1}
- motion model $f()$
- time step length dt
- process noise covariance Q
- σ-points function $sigmaPoints()$
- which kind of Kalman filter to use *KFtype*

The algorithm recursively works its way through the whole time span, saving all the predictions as they are calculated. Pseudocode describing the implemented prediction function can be seen in Algorithm 2. Here there are three different versions depending on what kind of Kalman filter that is used. The EKF is fairly straight forward where x'_k is the result received from passing x_{k-1} through the Jacobian of the motion model.

The prediction step for the UKF starts with generating σ-points as described in Section 5.3.2.1. With the process model the algorithm then loops through all the σ-points, passing them through the motion model and adds up the product of the transformed points and their weights to the predicted mean value x_k.

Next, another for-loop then takes over, this loop uses the mean value and the transformed σ-points to generate the covariance of the prediction. The covariance matrix is generated using the state space distance between the predicted mean and the predicted σ-points. Since the σ-points are spread around the last estimated mean based on the covariance of that estimation, as seen in Equation 5.28, the transformed points are used to derive the predicted covariance, see Equation 5.32.

The prediction process for CKF is very similar to the UKF process described above. The only difference is how the σ-points are placed around the mean and that there are only $2n$ σ-points instead of the $2n + 1$ points used by the UKF.

Algorithm 2 Kalman Filter Prediction

```
 1: if $KFtype$ = $'EKF'$ then
 2:     $[x_k, x'_k] \leftarrow f(x_{k-1}, dt)$
 3:     $P_k \leftarrow x'_k \cdot P_k \cdot (x'_k)^T + Q$
 4: else if $KFtype$ = $'UKF'$ then
 5:     $n \leftarrow length(x_{k-1})$
 6:     $[SP_{k-1}, W] \leftarrow sigmaPoints(x_{k-1}, P_{k-1}, KFtype)$
 7:     $x_k \leftarrow 0$
 8:     for $i = 1 : 2 \cdot n + 1$ do
 9:         $SP_k \leftarrow f\,(SP_{k-1}(i), dt)$
10:         $x_k \leftarrow x_k + SP_k * W(i)$
        end
11:     $P_k \leftarrow Q$
12:     for $i = 1 : 2 \cdot n + 1$ do
13:         $SP_k \leftarrow f\,(SP_{k-1}(i), dt)$
14:         $P_k \leftarrow P_k + (SP_k - x_k) \cdot (SP_k - x_k)^T \cdot W(i)$
        end
15: else if $KFtype$ = $'CKF'$ then
16:     $n \leftarrow length(x_{k-1})$
17:     $[SP_{k-1}, W] \leftarrow sigmaPoints(x_{k-1}, P_{k-1}, KFtype)$
18:     $x_k \leftarrow 0$
19:     for $i = 1 : 2 \cdot n$ do
20:         $SP_k \leftarrow f\,(SP_{k-1}(i), dt)$
21:         $x_k \leftarrow x_k + SP_k * W(i)$
        end
22:     $P_k \leftarrow Q$
23:     for $i = 1 : 2 \cdot n$ do
24:         $SP_k \leftarrow f\,(SP_{k-1}(i), dt)$
25:         $P_k \leftarrow P_k + (SP_k - x_k) \cdot (SP_k - x_k)^T \cdot W(i)$
        end
    end
```

5.5.2 Kalman Filter Update

Once a prediction has been made using the physical models this predicted state is used to perform the update step which will be described in this section. The Kalman filter update is described with pseudocode in Algorithm 3. The inputs required to perform the update step are:

- predicted mean $\hat{x}$
- predicted covariance $\hat{P}$
- measurement noise covariance R
- measurements y
- measurement model h
- σ-points function $sigmaPoints()$
- which kind of Kalman filter that is used *KFtype*.

Similarly to the prediction step described in the previous section the update algorithm works recursively, updating the filtered output for every time step. For the update step the information from both the prediction step and the sensor data is used.

For the EKF it first passes the predicted mean $\hat{x}$ through the measurement model and also computes the Jacobian of the measurement model evaluated at that mean. The measurement model may differ depending on which sensor is providing the measurement(s). In sensor fusion the sensors used do not have to measure the same state variables. Using the Jacobian, the predicted covariance $\hat{P}$ and the measurement noise covariance R the innovation covariance S is computed. The innovation covariance together with the cross covariance are later used for computing the Kalman gain. The Kalman gain is used for updating the estimated state x depending on the accuracy of the prediction, essentially how good or bad the prediction is. Finally the covariance is updated with the use of the Kalman gain K and the innovation covariance S.

The update for the UKF starts with generating σ-points with the predicted mean $\hat{x}$ and covariance $\hat{P}$. All $2n + 1$ σ-points are passed through the measurement model h and then multiplied with the weights, received also from the $sigmaPoints$-function, and added up to the predicted measurement $\hat{y}$. With the predicted mean $\hat{x}$ and measurement $\hat{y}$ the cross covariance C and innovation covariance S are computed. The cross covariance depends on how much the σ-points are spread from both the predicted mean and the measurement. The innovation covariance only depends on the predicted measurements and the σ-points. With these covariances the Kalman gain is calculated and the mean and covariance are updated.

Algorithm 3 Kalman Filter Update

```
1: if $KFtype =$ 'EKF' then
2:     $[hx, hx'] \leftarrow h(\hat{x})$
3:     $S \leftarrow hx' \cdot \hat{P} \cdot (hx')^T + R$
4:     $K \leftarrow \hat{P} \cdot hx'/S$
5:     $x \leftarrow \hat{x} + K \cdot (y - hx)$
6:     $P \leftarrow \hat{P} - K \cdot S \cdot K^T$
7: else if $KFtype =$ 'UKF' then
8:     $n \leftarrow length(\hat{x})$
9:     $[SP, W] \leftarrow sigmaPoints(\hat{x}, \hat{P}, KFtype)$
10:    $\hat{y} \leftarrow 0$
11:    for $i = 1 : 2n + 1$ do
12:        $\hat{y} \leftarrow \hat{y} + h(SP(i)) \cdot W(i)$
       end
13:    $S \leftarrow R$
14:    $C \leftarrow 0$
15:    for $i = 1 : 2n + 1$ do
16:        $C \leftarrow C + (SP(i) - \hat{x}) \cdot (h(SP(i)) - \hat{y})^T \cdot W(i)$
17:        $S \leftarrow S + (h(SP(i) - \hat{y}) \cdot (h(SP(i)) - \hat{y})^T \cdot W(i)$
       end
18:    $K = (C/S)$
19:    $x \leftarrow \hat{x} + K \cdot (h(y) - \hat{y})$
20:    $P \leftarrow \hat{P} - K \cdot C^T$
21: else if $KFtype =$ 'CKF' then
22:    $n \leftarrow length(\hat{x})$
23:    $[SP, W] \leftarrow sigmaPoints(\hat{x}, \hat{P}, KFtype)$
24:    $\hat{y} \leftarrow 0$
25:    for $i = 1 : 2n$ do
26:        $\hat{y} \leftarrow \hat{y} + h(SP(i)) \cdot W(i)$
       end
27:    $S \leftarrow R$
28:    $C \leftarrow 0$
29:    for $i = 1 : 2n$ do
30:        $C \leftarrow C + (SP(i) - \hat{x}) \cdot (h(SP(i)) - \hat{y})^T \cdot W(i)$
31:        $S \leftarrow S + (h(SP(i) - \hat{y}) \cdot (h(SP(i)) - \hat{y})^T \cdot W(i)$
       end
32:    $K = (C/S)$
33:    $x \leftarrow \hat{x} + K \cdot (h(y) - \hat{y})$
34:    $P \leftarrow \hat{P} - K \cdot C^T$
   end
```

5.5.3 Rauch-Tung-Striebel Smoother

Last in the fusion toolchain the smoothing algorithm is implemented. The smoothing algorithm uses the results generated during the full forward filter pass to perform estimations, and was therefore implemented last when the previous parts were finalized. The smoothing function works backwards recursively and for every time step it takes as input:

- Smoothed mean at time $k+1$, x^s_{k+1}
- Smoothed covariance at time $k+1$, P^s_{k+1}
- Filtered mean at time k, x^f_k
- Filtered covariance at time k, P^f_k
- Predicted mean at time $k+1$, x^p_{k+1}
- Predicted covariance at time $k+1$, P^p_{k+1}
- Motion model, f
- Time step length, dt
- σ-points function, $sigmaPoints()$
- which kind of Kalman filter that is used, *KFtype*

The RTS process works backwards based on knowledge from the filtered data from the previous time step $k+1$. The first thing it does is to initialize the final values x^s_N, P^s_N with the last values from the filtered output. From this it starts to loop backwards $k = N-1, N-2, \ldots, 2, 1$ until it reaches the first measurement. For the EKF it starts with passing the filtered state for time step k through the motion model to receive the prediction and the Jacobian for k+1. The Jacobian is then used together with the covariances P^f_k and P^p_{k+1} to compute the gain G_k.

The UKF computes the gain in a different way, it first creates σ-points from the filtered mean and covariance for time k. Then a for-loop that runs for $2n+1$ times is used to create the covariance $P_{k|k+1}$ which is a covariance depending on time k and $k+1$ by comparing the σ-points with the mean for both of those time steps. Then together with that result and the predicted covariance P^p_{k+1} the gain G_k is computed. For the CKF this is done exactly the same as for the UKF except the number of σ-points and the weights used to create them.

Once the gain has been computed it is used to scale how much the filtered value should be changed depending on the difference between the predicted value x^p_{k+1} and the smoothed value x^s_{k+1}. The covariance is updated in the same manner but depending on the difference between the predicted and the smoothed covariances P^p_{k+1}, P^s_{k+1}. The gain is also squared for updating the covariances.

Algorithm 4 Rauch-Tung-Striebel Smoother

```
1:  $N \leftarrow nrOfTimeSteps$
2:  $x_N^s \leftarrow x_N^f$
3:  $P_N^s \leftarrow P_N^f$
4:  for k=N-1:1 do
5:      if $KFtype =$ 'EKF' then
6:          $[x_{k+1}^f, (x_{k+1}^f)'] \leftarrow f(x_k^f, dt)$
7:          $G_k \leftarrow P_k^f \cdot (x_{k+1}^f)'/P_{k+1}^p$
8:      else if $KFtype =$ 'UKF' then
9:          $[SP, W] \leftarrow sigmaPoints(x_k^f, P_k^f, KFtype)$
10:         $P_{k|k+1} \leftarrow 0$
11:         $n \leftarrow length(\hat{x})$
12:         for $i = 1 : 2n + 1$ do
13:             $P_{k|k+1} \leftarrow P_{k|k+1} + (SP(i) - x_k^f) \cdot (f(SP(i), dt) - x_{k+1}^p)^T \cdot W(i)$
            end
14:         $G_k \leftarrow P_{k|k+1}/P_{k+1}^p$
15:     else if $KFtype =$ 'CKF' then
16:         $[SP, W] \leftarrow sigmaPoints(x_k^f, P_k^f, KFtype)$
17:         $P_{k|k+1} \leftarrow 0$
18:         $n \leftarrow length(\hat{x})$
19:         for $i = 1 : 2n$ do
20:             $P_{k|k+1} \leftarrow P_{k|k+1} + (SP(i) - x_k^f) \cdot (f(SP(i), dt) - x_{k+1}^p)^T \cdot W(i)$
            end
21:         $G_k \leftarrow P_{k|k+1}/P_{k+1}^p$
        end
22:     $x_k^s \leftarrow x_k^f + G_k \cdot (x_{k+1}^s - x_{k+1}^p)$
23:     $P_k^s \leftarrow P_k^f + G_k \cdot (P_{k+1}^p - P_{k+1}^s) \cdot G_k^T$
    end
```

6
System Evaluation

When creating a new system a large portion of time will be spent creating the code to get it to a working state. After an acceptable working state is reached and the code works as intended the need for evaluating and possibly improving the system emerges. To be able to evaluate the performance of a system a structured testing process is required. In this chapter first the metrics used for evaluating performance will be described. After that the scenarios, during which all of the mentioned specifications can be evaluated, will be described.

6.1 Evaluation Criteria

Before evaluation could begin there was a need to define how the performance of the system would be evaluated. The performance of the system will be evaluated using the four criteria described in the following sections. Given a reference against which to compare the fused output, performance metrics for accuracy, confidence and resolution may be calculated. However even without reference data a qualitative evaluation may be performed.

6.1.1 Accuracy

Being able to improve the accuracy is of great importance for high end positional sensors such as radar and LIDAR. Therefore this has been chosen to be one of the things that should be improved with this sensor fusion system. The fused data should have a better accuracy then the two sensors can provide individually. To be able to measure any accuracy there has to be an external ground truth to compare with. This is where the RTK-data may be used to provide that reference. This is a sensor which the radar, LIDAR and fused data can be compared to and the accuracy of each of those outputs may then be calculated.

A good way of measuring the accuracy of sensors in an automotive application would be the use of Root Mean Square Errors (RMSE) described in Equation 4.3. The RMSE does not only find how far away the points are from the true values but also punishes large errors. This is desirable since a few large errors is something that would affect the final results more than many small errors.

6.1.2 Confidence

Measurement confidence is a property of measurements that defines how likely it is to be correct e.g. if several sources give the same or similar readings of some variable the confidence in those measurements being true increases. Having several sensors covering the same area will reduce the risk of objects being missed and thus increase the confidence in the mapping of the surrounding area being correct.

6.1.3 Resolution

The sensors, both the radar and the LIDAR, has a certain resolution which means that they both only update their measurements in discrete steps. The reality however is not discrete so it would be preferable to be able to represent the measurements continuously or at least with a better resolution. This is something that can be achieved with the use of an RTS smoother, or other smoothers, since it finds a smooth approximated curve in between those discrete points. A smooth curve is a much better approximation of how a vehicle is moving since those typically move in a smooth manner with low acceleration/deceleration.

The output resolution of the system may also achieve a higher temporal resolution than each of the input sensors could do separately. A higher refresh rate in measurement updates increase the amount of information over a given time period allowing the user to resolve more detail over a given time period.

6.1.4 Robustness

When discussing robustness many aspects may be taken into consideration. A system can be robust in a manner where it withstands different external noise, e.g weather, movement, force. It can also be a case where it is robust in a way that if it has faulty measurements it can handle those in a way where the output remains reasonably stable. Both of these definitions have been used when performing evaluation of the robustness of the system since both are important for the Reference box. It will have to perform well under bad weather conditions but also give good results even though the measurements sometimes can be faulty due to other reasons.

This is something that has been evaluated by checking scenarios where the data either has faulty measurement or was gathered during bad weather conditions. The robustness against faulty measurements was checked manually by reviewing plots of the fused output to see whether it remained stable during the faulty measurements. For robustness against bad weather conditions the fused output was checked for scenarios with heavy downpour.

6.2 Scenarios

Validation and evaluation of the fusion algorithm will entail the use of the hunter and target vehicles to enact a few typical traffic scenarios. The scenarios would be designed to evaluate various aspects of the fusion algorithm while not deviating to far from what may be considered normal traffic situations.

To fully evaluate the performance of the various fusion solutions implemented during the course of this project a more accurate reference is necessary. A reference system capable of recording position and motion with high enough accuracy is the RTK system described in Section 2.3. By equipping two vehicles with the same number of RT3000s, the position and motion of both vehicles may be recorded with high accuracy. Using the sets of data from both vehicles, their relative position may be calculated and used as a reference against which the fused output from the sensor fusion algorithm may be compared. The vehicle equipped with both RT3000 and the roof-mounted reference box will hence be referred to as the hunter. The second vehicle equipped with only the RT3000 setup will be referred to as the target.

Performing the scenarios and gathering new data with RTK reference for both hunter and target is a major undertaking. Therefore initial evaluation will be performed with suitable preexisting data gathered from test drives on public roads. This data was chosen to be similar to the scenarios planned.

6.2.1 Overtaking Scenario

The overtaking scenario is meant to replicate driving on a motor- or highway where the hunter passes to the left of a slower moving car. This maneuver might be thought of either as an overtaking or simply passing a car traveling in the right lane. The scenario is intended to test the algorithms performance in a scenario where both hunter and target are traveling at moderate to high speeds with a low relative velocity.

6.2.2 Following Scenario

The following scenario will replicate the most typical driving scenario where the hunter is traveling at low to moderate speeds behind the target vehicle. This scenario is similar to the overtaking scenario but rather than being performed on a straight motorway this scenario would be performed on a smaller less straight road, intended to replicate a typical country road with one lane in each direction. The relative speed between target and hunter will in this scenario be close to or equal to zero. This scenario will be useful in validating the ability of the fusion algorithms to compensate for motion and rotation of the hunter.

6.2.3 Oncoming Scenario

The most frequently encountered and also the most hazardous situation encountered on many roads is the oncoming scenario. This scenario may take almost any shape and form but the essentials that makes it dangerous is the high relative velocity between two oncoming vehicles. Most country roads generally have reasonable speed limits reducing the chances of a fatal outcome in case of run off road accidents or other accidents involving only one vehicle. However, these roads often lack a dividing barrier separating the lanes in one direction from those in the other direction. The lack of such a dividing barrier results in situations where two oncoming vehicles, both traveling at a moderate speed, may collide with a very high relative speed, increasing the risk of major injury or death of the occupants.

6.2.4 Bad Conditions Scenario

A common factor that all vehicles encounter at some point is bad weather. The most common result of bad weather conditions is reduced visibility, reduced visibility may be caused by rain, snow, fog, dust or something completely different. Regardless of cause, low visibility makes on-road travel more dangerous since it reduces the effective range of both sight and light based sensors, such as cameras and LIDAR. The bad weather scenario is intended to evaluate redundancy and the ability of the system to maintain an accurate target tracking in adverse conditions.

6.2.5 Stationary Observer Scenario

When tracking an object, and using a motion model to make predictions regarding the expected position of that object it is necessary to take into account how the observer has moved in between two consecutive observations. The motion of the hunter vehicle, hence referred to as ego-motion, will create an offset between the coordinate system in which the prediction is made and the coordinate system from which the following observation is made. The coordinate system in which the prediction is made is centered in the position of the hunter at the time of the most recent sensor update/observation, while the consecutive observation will be made from the then current position of the hunter. When the hunter is stationary the perspective from which two consecutive observations are made is the same. Therefore, when the hunter is immobile there is no need to compensate for the movement of the observer perspective. By having the hunter parked and motionless the motion models used to make prediction may be evaluated independently of the ability to compensate for ego-motion.

7
Results and Discussion

This chapter will present the results generated during the project. Initial intentions for evaluation is thoroughly described and discussed in the previous chapter. However, due to restructuring within the company efforts to gather, decode, post-process and evaluate the system using data with the necessary RTK-reference available has been futile. The implemented code has been designed for and intended to allow for evaluation using RTK-data, nonetheless in the absence of data and within the limited time frame of the project such evaluation has proven to be out of reach. Therefore this chapter will present results generated using pre-existing data gathered during test drives unrelated to this project. Because the data used has been gathered for other purposes it does not include the RTK reference data that would be necessary to evaluate increased accuracy or precision of the fused output. Without a valid and more accurate reference to compare the fused output to it becomes much more difficult to draw conclusions regarding the accuracy. To draw a conclusion regarding the accuracy of the fused output would require knowledge of the true relative position which cannot be known given the currently available data.

The data used in this chapter has been taken from servers where the company stores all the data recorded from test drives. From the copious amounts of data the scenarios and tracks presented as results in this chapter have been carefully picked to be similar to the scenarios described in the previous chapter. The chosen scenarios will be used to evaluate and draw conclusions regarding various combinations of tunable parameters, models and more.

Testing with different kinds of Kalman Filters has been performed. The EKF, UKF and CKF have all been tested on several data sets and produced such similar results, seen in Figure 7.1, that there is no need in showing the comparison further. The CKF version has been used for all the results presented below since it has the lowest computation time of the three methods.

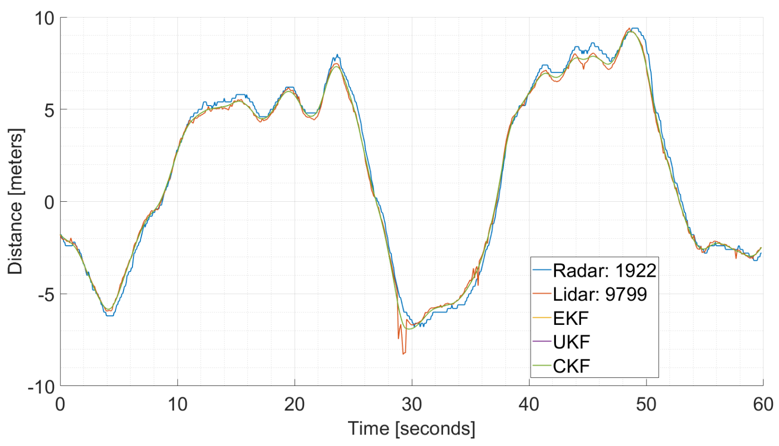


Figure 7.1: Comparison between the EKF, UKF and CKF, all of them very similar.

Overall the fusion algorithm is working really well and is capable of producing a reasonable and smooth fusion between the radar and LIDAR measurements when the measurements are stable. For this reason overall results is only shown in the first Scenario in Section 7.1. For the rest of the scenarios the focus will rather be on some special cases than on the overall results. The overall results are good and similar to the first scenarios results for all of the below scenarios.

7.1 Following Scenario

The first scenario takes place on a winding country road in California where the weather is clear, in this data set the hunter is following another test vehicle. The images in Figure 7.2 display three screenshots from the integrated camera. During the scenario there was also other oncoming vehicles present. The speed of the hunter and target vehicle is between 60-80 km/h.

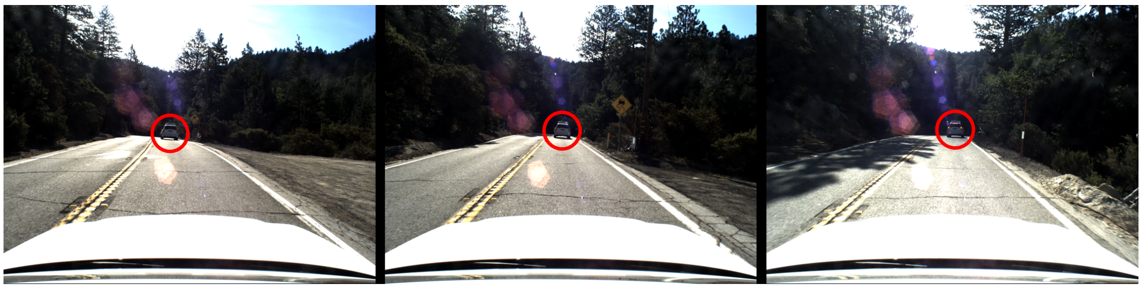
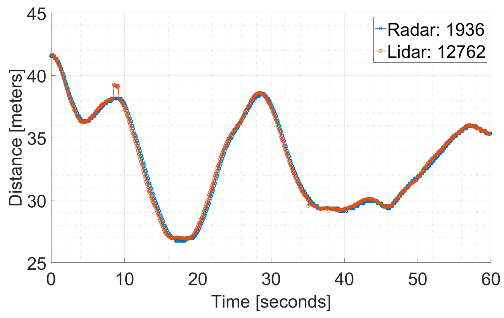
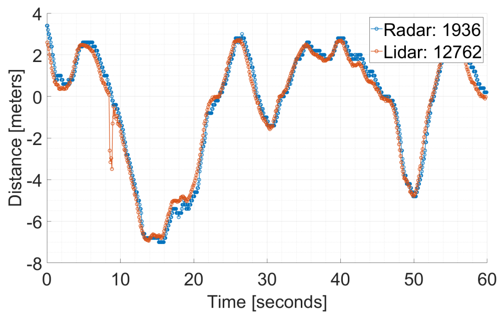


Figure 7.2: Camera frames of the following scenario covering the period during which the LIDAR gives faulty readings. Target vehicle is encircled in red.

The position of the vehicle in front in relation to the hunter over time is shown in the two graphs in Figure 7.3 where the relative distance in meters is displayed along the Y-axis and the time in seconds along the X-axis. The numbers after the radar and LIDAR is the track ID that was given in the input. In both these graphs it is apparent that something causes the LIDAR-sensor to return a couple of consecutive faulty readings between 8 and 9 seconds. How the algorithm deals with these faulty readings will be discussed below.



(a) Longitudinal measurements



(b) Latitudinal measurements

Figure 7.3: The longitudinal and latitudinal relative position of the target vehicle as seen by the radar and LIDAR.

The input data from the sensors appears correct and smooth in Figure 7.3 but upon closer inspection it becomes clear that this smoothness is an illusion of the scale used in the figure. In Figures 7.4-7.5 a cropped section of the longitudinal and latitudinal position graphs are shown. At this increased scale it becomes clear that especially the radar data is quite coarse. The ARS-408 radar used to gather this data has a lateral resolution of about 0.2 meters which is clear to see in the shape of the blue line in Figure 7.5. The lateral resolution of the LIDAR is higher but that line is not very smooth either.

Looking at the fusion results in Figure 7.4 for the different motion models it can be seen that they produce slightly different results. The coordinated turn with Cartesian coordinates and constant acceleration model produces very similar results, even to the point where it is hard to distinguish them from each other at the scale used in Figure 7.4. Both of them are producing very satisfying results with a smooth line following measurements from both sensors equally. For the coordinated turn with polar coordinates the output follows the two sensors with some smaller exceptions, just before 5 seconds. The smoothness of that curve however is not as good as it makes a subtle jump every time it gets a new measurement update. This it not a wanted result from the fusion with smoothing.

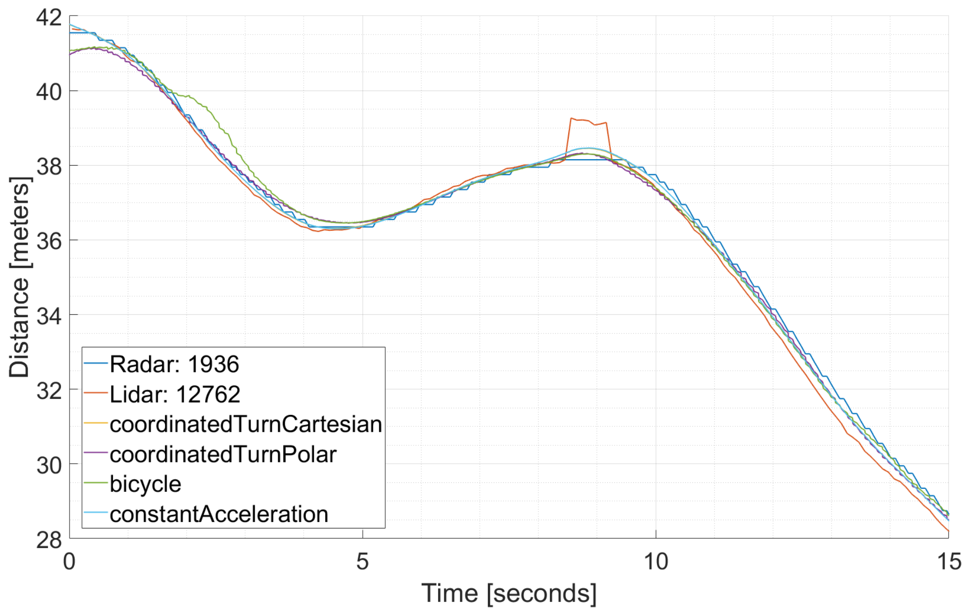


Figure 7.4: Fused longitudinal position with various motion models. The coordinatedTurnCartesian is covered by constantAcceleration since they are very similar.

The bicycle model is the model that follows the sensor measurements the least and it also gives the most uneven graph. At 2.5 seconds it deviates 1.08 metres from the LIDAR measurement. After 5 seconds though it starts to settle down and assumes a reasonable value and after that it follows the sensor values better. Despite evening out it still is not as smooth as the other motion models. The issue causing it to not fuse the signals appropriately is probably not within the motion model itself but rather some issues with the data. The predicament with the bicycle model is that it was found to be very difficult to tune to a suitable behaviour in this project. The process noise covariance matrix has to be extensively tweaked to get a good behaviour and it is a time consuming task. With more time spent on tuning the results could possibly have been better.

Looking at Figure 7.5 which holds the fusion results for the latitudinal position the coordinated turn motion with Cartesian coordinates curve can not be seen since it once again almost exactly coincides with the curve for the constant acceleration model. The results here as well as in Figure 7.4 are good with smooth lines following the sensors' measurements equally. The coordinated turn with polar coordinates is also showing good results with a smooth line; even smoother than for the longitudinal position. The bicycle models shows a very shaky result for the first 2 seconds but then it settles in to a reasonable result. It is still the most uneven of the graphs but this is probably due to the reasons discussed in the previous paragraph. Since no better tuning of the bicycle model was found this model will not be used in the latter scenarios since it does not perform as well as it has capacity for.

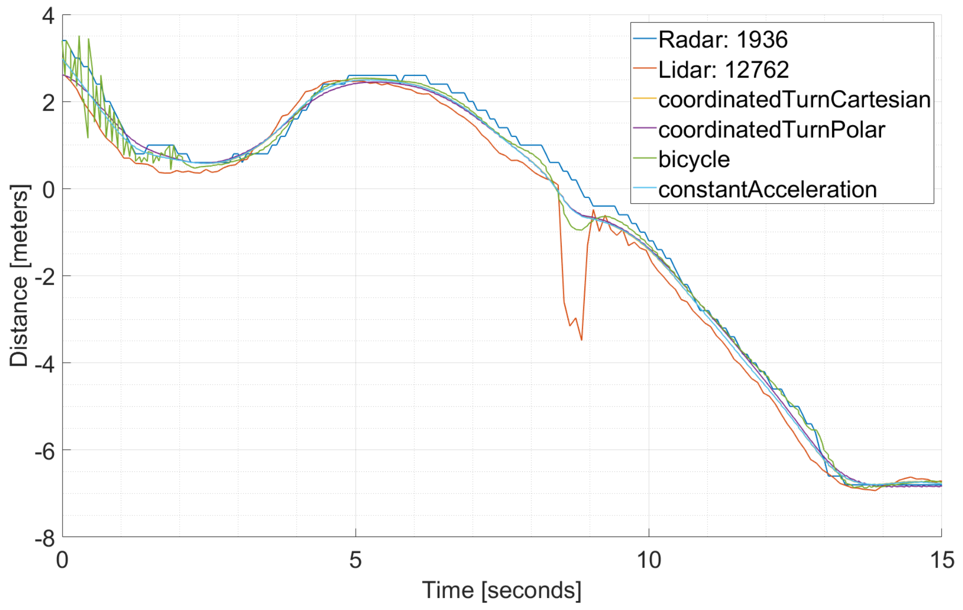


Figure 7.5: Fused latitudinal position with various motion models. The coordinatedTurnCartesian is covered by constantAcceleration since they are very similar.

In Figure 7.6 and 7.7 cropped versions of the data graphs are shown, these figures clearly show the faulty readings from the LIDAR sensor. Between 8 and 9 seconds the longitudinal position increases by about one meter during 7 measurement updates, simultaneously the latitudinal distance suddenly decreases by approximately 3 meters. The three images in Figure 7.2 shows the camera view just before, during and just after this issue and the video clearly shows how the tracked vehicle follows a smooth trajectory during this sequence. In the two cropped figures the result of various process noise levels are shown. The process noise is applied as a diagonal matrix with the same entry to every position. The result of having a very low process noise is that the system becomes more resilient against issues like this faulty reading. The yellow line, with the lowest process noise maintains its trajectory even when several consecutive measurements from one sensor places the target several meters away from its true position.

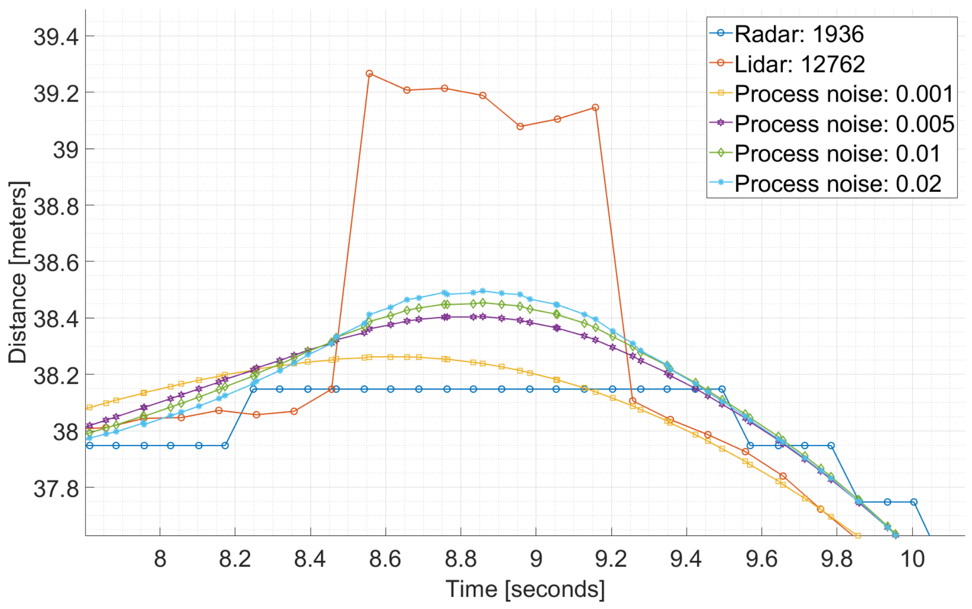


Figure 7.6: Longitudinal error and fused output with various levels of process noise. The motion model used is Coordinated turn with Cartesian coordinates

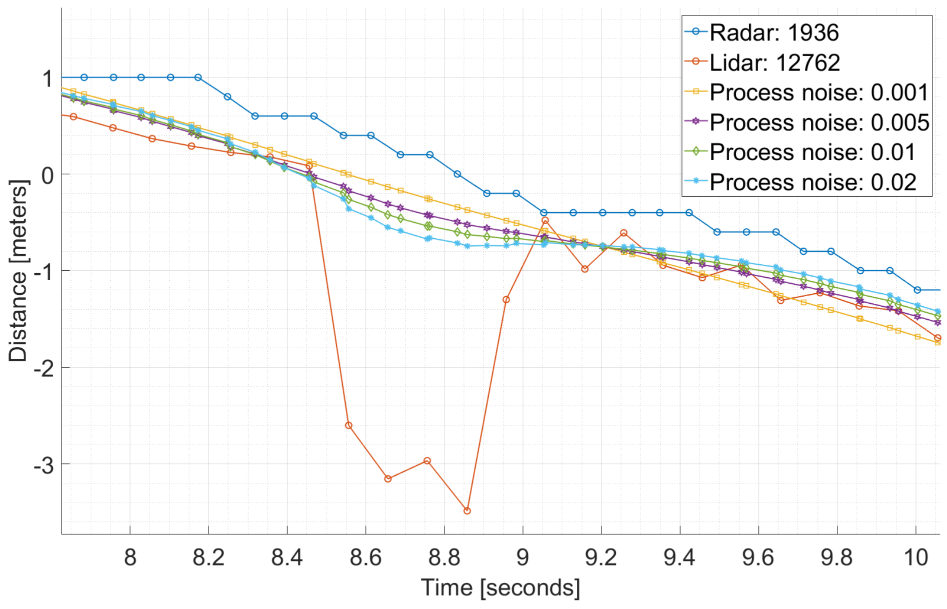


Figure 7.7: Latitudinal error and fused output with various levels of process noise. The motion model used is Coordinated turn with Cartesian coordinates

Decreasing the process noise to make the fusion less sensitive to bad readings does however have a cost. Lowering the process noise to such a low level introduces a considerable amount of latency in the tracking, this becomes obvious in Figure 7.8. In this graph it is easy to see that the fused output has difficulties keeping up with sudden changes even if both sensors measures that same change. It may also be concluded that less extreme process noise values quickly reduces this inertia to a more acceptable level.

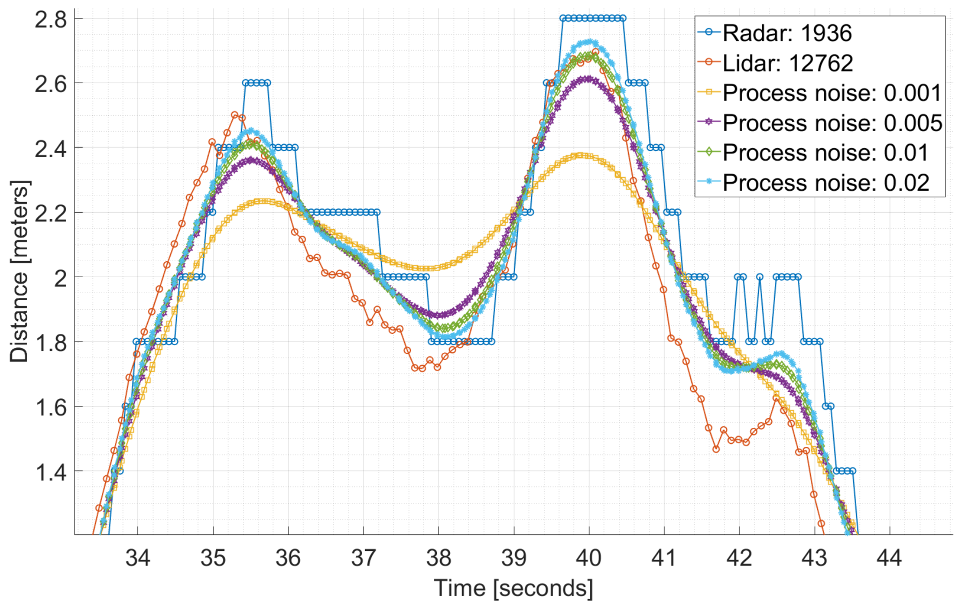


Figure 7.8: Latitudinal position showing unwanted effects of using low process noise

7.2 Overtaking Scenario

The overtaking scenario takes place on a straight piece of motorway where the hunter vehicle is traveling at a higher rate of speed than the car encircled in red, see Figure 7.9. The hunter is initially traveling in the same lane as the marked vehicle and leaves that lane as it approaches and then overtakes the tracked vehicle. This scenario represents most motorway scenarios in that all included vehicles are traveling at high speeds, ca. 100-120 km/h, and that the relative velocity between the vehicles are low to moderate.

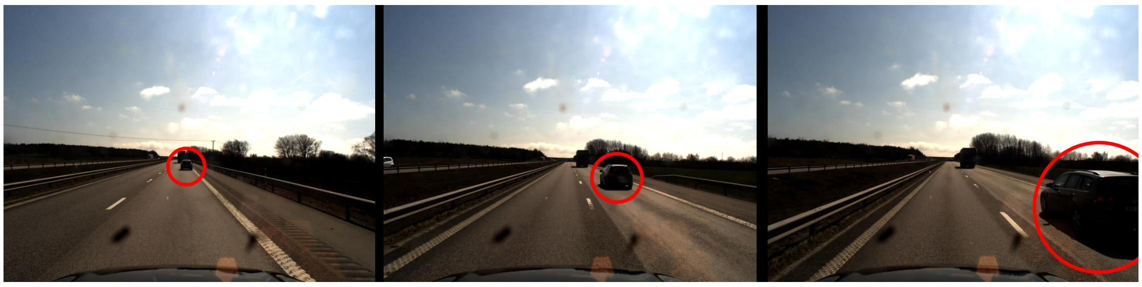
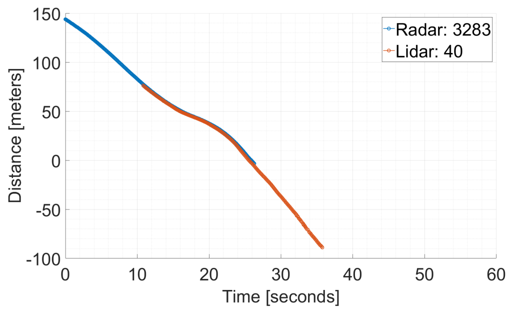
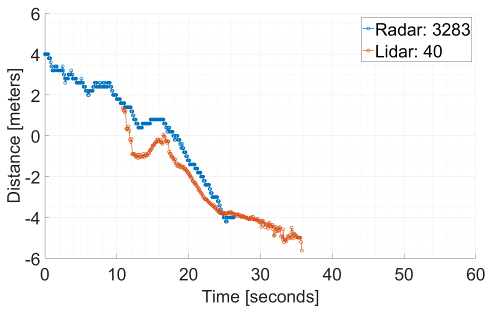


Figure 7.9: Figure shows how the hunter vehicle catches up to and changes lane to overtake a slower vehicle in the right lane

Longitudinal and latitudinal position over time is shown in Figure 7.10, as the figure shows the radar is the first of the two sensors to lock on to the vehicle due to its superior range. The vehicle is detected by the LIDAR when it is just below 75 meters away and the latitudinal position is initially very similar to that measured by the radar but almost immediately appears to detect some rightwards movement. The movement detected by the LIDAR is not confirmed by the radar nor the video. The latitudinal gap between the radar and LIDAR measurements then reduce and seem to converge as the hunter catches up and changes lanes to overtake the target vehicle. The longitudinal measurements are very similar for both sensors, with a comparatively small offset hardly visible in Figure 7.10. In this case similar measurements from separate sources gives increased confidence in the longitudinal distance since the measurements agree.



(a) Longitudinal measurements



(b) Latitudinal measurements

Figure 7.10: The longitudinal and latitudinal relative position of the target vehicle as seen by the radar and LIDAR.

The LIDAR and radar sensor used in this thesis have their inherent strengths and weaknesses, as discussed in the Hardware Chapter 2. One method of incorporating the knowledge about sensor performance into the fusion algorithm is by altering the measurement noise covariance accordingly. How the measurement noise covariance influences data interpretation is described in greater detail in Section 5.5.2, essentially it is used to describe the noise in various measurements. Figure 7.11 shows a cropped version of the latitudinal position graph, the three lines between the radar and LIDAR lines shows the result of using three versions of measurement noise covariances for radar measurement updates. The motion model used is the coordinated turn with Cartesian coordinates with its state shown in Equation 5.45. The three measurement noise covariance matrices used are defined as

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0.5 \end{bmatrix} \quad C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 10 \end{bmatrix}, \tag{7.1}$$

where the A matrix is the identity matrix which is the same covariance matrix used for the LIDAR measurement updates. Using the same covariance matrices for both sets of inputs produces the yellow line, see Figure 7.11 which follows both sources equally and therefore tends to end up close to the average of the two sources.

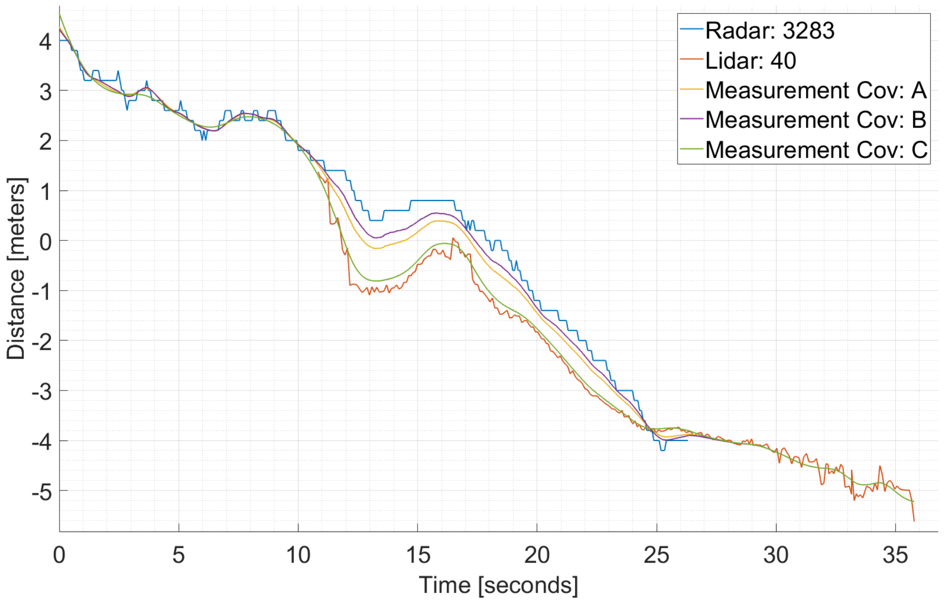


Figure 7.11: Results of fusion using the various measurement covariance matrices seen in Equation 7.1.

The remaining two matrices, B and C in Equation 7.1 defines the noise covariance matrices used when performing measurement updates using radar data in Figure 7.11. The B matrix has got two values set to 0.5, the cells set to this value are the ones defining the measurement variance for measurements of latitudinal position and velocity. By setting these values below one the algorithm will rely more on latitudinal measurements from the radar since their variances are set to a lower level. Figure 7.11 shows the reuslt of lower variance as the purple line ends up closer to the blue radar line than the yellow line does. Finally matrix C defines the variance of measurements in the latitudinal direction as 10 for both position and speed. This declares to the algorithm in the update step that measurements from the radar sensor regularly varies by up to 10 meters, which is an obvious exaggeration. However the result of using this exaggerated variation is that the fused output of the system becomes almost independent of radar measurements, see the green line in Figure 7.11.

These measurement noise covariances are not based on actual sensor performance data but rather used to show a method of minimizing the influence of unreliable measurements. The benefit of altering the noise covariance rather than decreasing the process noise is that it avoids the latency issues discussed in Section 7.1. By altering the measurement noise covariance it is also possible to extract all the valuable information from a sensor e.g. trusting the longitudinal measurements while almost ignoring the latitudinal measurements for the radar.

7.3 Oncoming Scenario

The oncoming scenario is a scenario where several oncoming vehicles are approaching with a speed of 60-80 km/h whilst the hunter vehicle is also traveling at similar speeds. The weather is clear and road ahead is slightly bent to the right as shown in the three frames in Figure 7.12. The results presented below will be the tracking of one of the oncoming vehicles, the one encircled in red in the Figure 7.12.

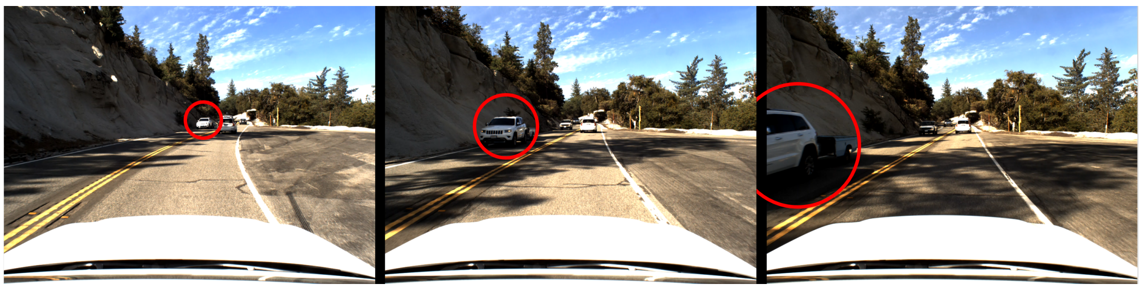
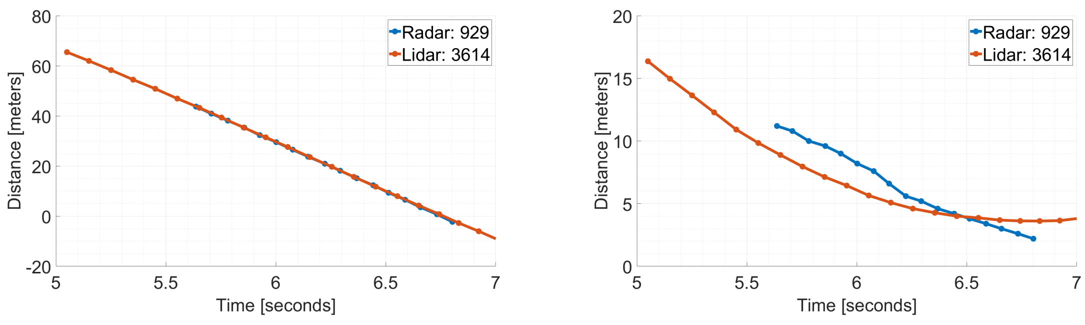


Figure 7.12: Three camera frames showing the oncoming target vehicle encircled in red. In the rightmost frame it can be seen that the vehicle is towing a trailer.

Examining the input data shown in Figure 7.13 the longitudinal positional measurements are very similar from the radar and LIDAR and the fused output along that axis will therefore be very good for all of the tested motion models. Therefore results regarding the latitudinal position will be the main consideration of this section since these measurements are differing by up to about 2 meters where it differs the most. This is notable considering that the target vehicle is only 2-10 metres away in lateral direction and up to 45 meters along the longitudinal axis during that time span. The LIDAR detects the vehicle approximately 0.5 seconds before the radar likely due to the oncoming vehicle being obscured by another vehicle.



(a) Longitudinal measurements **(b)** Latitudinal measurements

Figure 7.13: The longitudinal and latitudinal relative position of the target vehicle as seen by the radar and LIDAR.

In Figure 7.14 the fusion results for latitudinal position can be seen for the different motion models using the same process noise of 0.05 applied as a diagonal matrix with the same value for every state. The coordinated turn with Cartesian coordinates shows very similar results to the constant acceleration model. Both of these models shows a satisfactory and smooth fusion between the two signals where it listens equally to input from both sensors. Looking at the coordinated turn model with polar coordinates it can be seen that the output has quite uneven movements; this behaviour is not reflected in the video. Looking at the end of the time span, between 6.8-7 seconds, it is clear that the fused output almost disregards the measurements from the LIDAR despite being configured to follow input from both sensors equally. The problem here may be that all available data was used from both of the two sensors, this includes using the heading estimate from the radar. The heading estimate from the radar is a very crude estimate, it only measures heading in discrete steps of 0.4 radians. Using crude estimates to perform the update step of the Kalman filter is likely to cause issues since that input appears to remain stable until it changes suddenly and rapidly. These sudden, rapid changes are likely to cause the algorithm to repeatedly underestimate and then overestimate the heading or vice versa. The process model is intended to replicate reality and thus only allows for smooth changes in heading. Having large step wise changes in heading causes major mismatches between the input and the internal representation of the system.

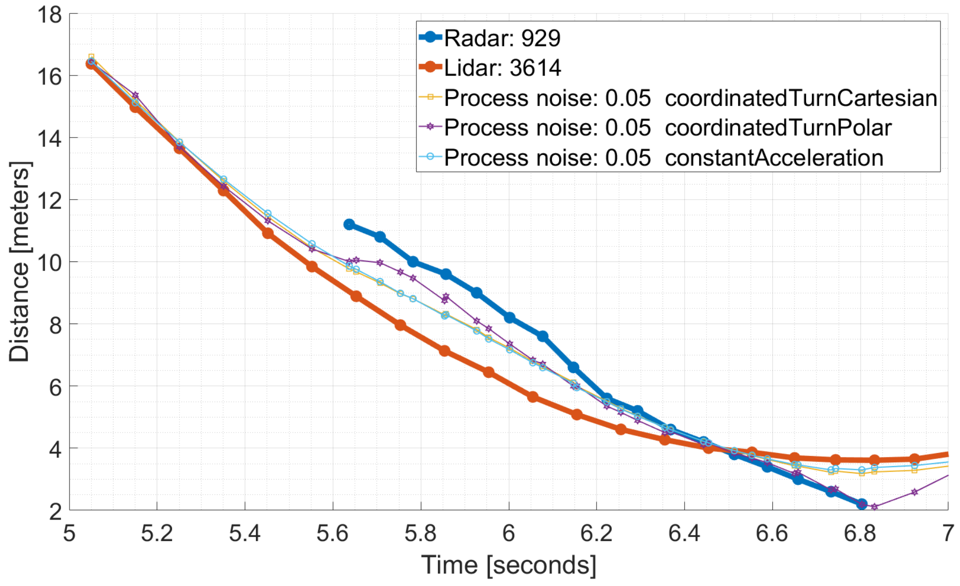


Figure 7.14: Latitudinal position. Raw data together using fusion with three different motion models with low process noise. Radar heading estimates used as input.

When trying the same scenario but this time completely disregarding the heading measurements from the radar the results shown in Figure 7.15. The only model that will have a different result is the coordinated turn with Polar coordinates since this is the only model originally using the heading measurements. Disregarding the heading measurement during radar updates but using it when performing updates using LIDAR data the resulting output is improved, see Figure 7.15. With this new approach the result is a smoother curve which also appears to more equally combine measurements from both sensors. This is desirable since it is not known which sensor actually provides the most accurate measurements in this sceanrio. Discussions regarding how to tune which sensors should be trusted at certain times is discussed in Section 7.2.

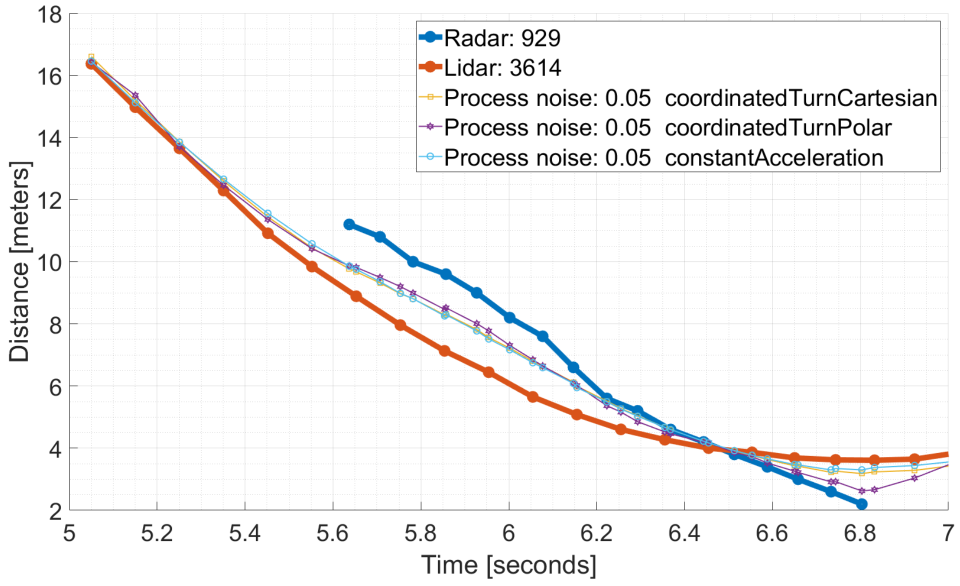


Figure 7.15: Latitudinal position. Raw data together using fusion with three different motion models with low process noise. Radar heading estimates not used as input.

For this scenario where there are many cars surrounding the ego vehicle at the same time it can be interesting to look at all the parallel tracks active at the time of the third frame in Figure 7.12. In Figure 7.16 a top-down view can be seen with the sensors field of view marked by the tinted areas, the fusion estimations of surrounding vehicles are shown as squares and the respective LIDAR detections as stars. Analyzing this figure from the left to right it can be seen that the vehicle just about to pass the ego car appears to be two vehicles. This is simply a detection of the trailer that the car is towing. For the trailer the heading estimate is a bit off but that is almost expected since the view of the trailer is almost entirely obstructed by the car towing it. The detection of the second oncoming car appears to be accurate both with position and heading. The latitudinal position estimation differs somewhat from the LIDAR, this is probably due to the estimation being based on both the radar and LIDAR data. The radar does not measure the latitudinal position as well as the LIDAR, see Chapter 2. For the third car the estimation is looking decent both with position and heading. The reason why there is no star corresponding to this estimation is that the LIDAR has not yet detected that car even though it is inside its field of view. This is something that happens in several cases and it shows the need for redundancy in the system. Just relying on one sensor is not always enough; some or a lot of overlap is beneficial since a single sensor cannot be expected to give reliable measurements 100% of the time. This is further supported by the results covered in the following section.

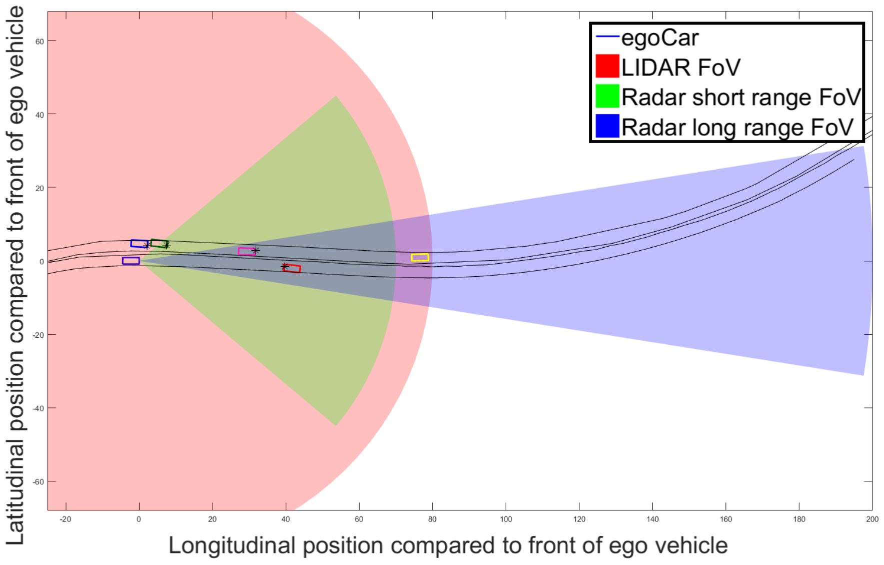


Figure 7.16: Top-down view of the ego car with surrounding vehicles. The squares are vehicles as estimated by the fusion algorithm. The stars are the position of the back of the car reported by the LIDAR. The black lines are the Lane markings as seen by the LIDAR

7.4 Bad Conditions Scenario

The bad condition scenario used was a scenario where the vehicle is driving on a wet road where snow mixed with rain is falling during the whole scenario. The speed of the vehicles is 60-70 km/h. The temperature is around 0 °C so water droplets and ice slush is building up on the lens of the camera as can be seen in the three frames in Figure 7.17. The target vehicle is the one encircled in red. In the first frame it is hard to see it through the camera but this is when the radar detects and begins tracking the oncoming target vehicle. In the second frame it is also hard to distinguish the vehicle but that frame is what the camera sees when the LIDAR detects and starts tracking the target vehicle. The final frame shows the camera view as target and hunter is just about to meet and clearly shows how the two vehicles are both traveling along fairly straight trajectories.

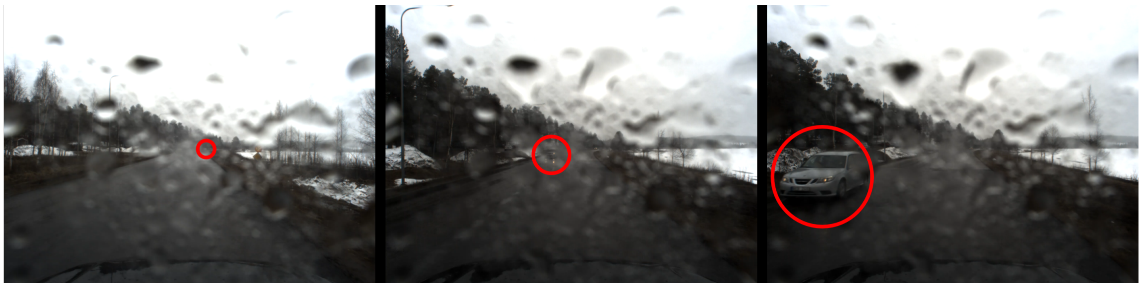
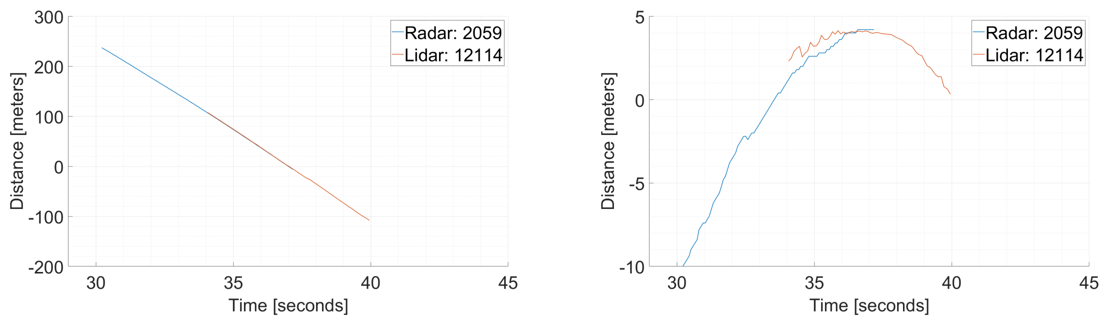


Figure 7.17: Three frames showing the oncoming vehicle encircled in red. The water obscures the oncoming vehicle in the first two frames

Looking at the input data shown in Figure 7.18a it can be seen that the radar detects the vehicle 237 metres away and the LIDAR detects it 108 metres away. This is close to what is expected from the two sensors according to their data sheets so along the longitudinal axis both sensors seem to perform well even though the sleet is there to disturb the LIDAR measurements. When examining Figure 7.18b the LIDAR appears to give some uneven measurements of the latitudinal position between 34-37 seconds. This time period covers the period during which the target vehicle is in front of the hunter. Once the vehicles pass each other and the longitudinal distance between them passes zero the uneveness decreases to the point where the measurements almost follow a smooth curve. This is something that can also be seen on other vehicle tracks during the same drive with the same conditions as well as other drives in similar weather conditions.

There might be several reasons contributing to bad measurements such as those in Figure 7.18b. However one of the more credible explanations of what might be affecting the measurements is the direction along which the vehicle is detected by the LIDAR. For objects in front of the moving hunter vehicle new drops of water and snow will hit the LIDAR at different spots every rotation. The drops and other debris hitting the lens could cause the lasers used by the LIDAR to distort an thus causing the measurements to return weird values. Since there will be new drops covering the lens of the LIDAR every rotation they will cause different distortions for every rotation. As the LIDAR sensor rotates most of the water and dirt on the surface of the lens will have been thrown off, due to centrifugal forces, before it is facing the direction along which the tracked vehicle may currently be detected. Also due to the speed of the hunter vehicle no new drops will hit the LIDAR lens when the lens is facing backwards since the hunter vehicle is moving forward. This would give a clearer lens for the LIDAR when facing backwards and could therefore be the reason why measurements of objects behind the hunter vehicle seems to be more stable. This might be the main contributor to the varying quality of measurements seen in the LIDAR data in Figure 7.18b, although there might be other external factors at play.



(a) Longitudinal measurements **(b)** Latitudinal measurements

Figure 7.18: The longitudinal and latitudinal relative position of the target vehicle as seen by the radar and LIDAR.

The fusion results can be seen in Figure 7.19. For the coordinated turn and constant acceleration models once again the results appears very similar with some small deviations. Both of them are performing well and manages to smoothly follow both the radar and the LIDAR measurements even though they are both slightly unstable and has some offset between them. For the coordinated turn model with polar coordinates the results appears to be unstable both before and after the vehicles has passed each other. This appears to be a result of bad heading estimates from the LIDAR. When checking the input data it appears that the LIDAR heading estimate is as crude as the radars heading estimates. This may be explained by the fact that the data used for this scenario was collected a couple of months prior to the other scenarios discussed above and might have gone through an earlier version of the postprocessing, one in which the postprocessing was not able to generate a stable heading estimation.

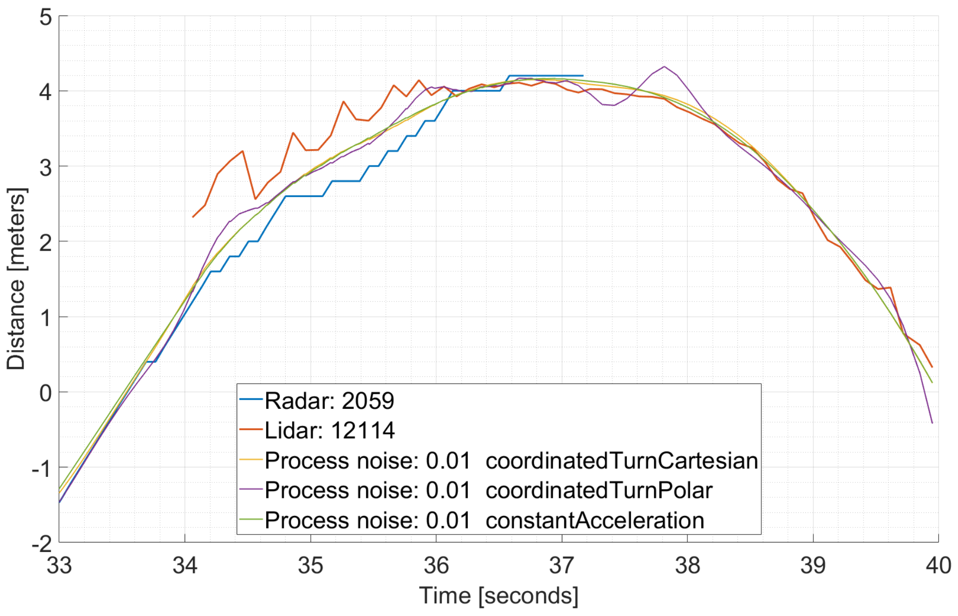


Figure 7.19: Latitudinal estimated position when observing heading

To fix this problem the heading estimate from the LIDAR can be disregarded and only updated trough the model without any measurements of that part of the state. This is what was done to create the result seen in Figure 7.20 where the coordinated turn model with polar coordinates gives a much better result than that shown in Figure 7.19. If unsure of the sensor's performance it may sometimes be a better choice to exclude that measurement from the measurement model when using a Kalman filter. The Kalman filter sometimes works better without measuring certain state variables, especially if the sensor keeps providing it with bad measurements.

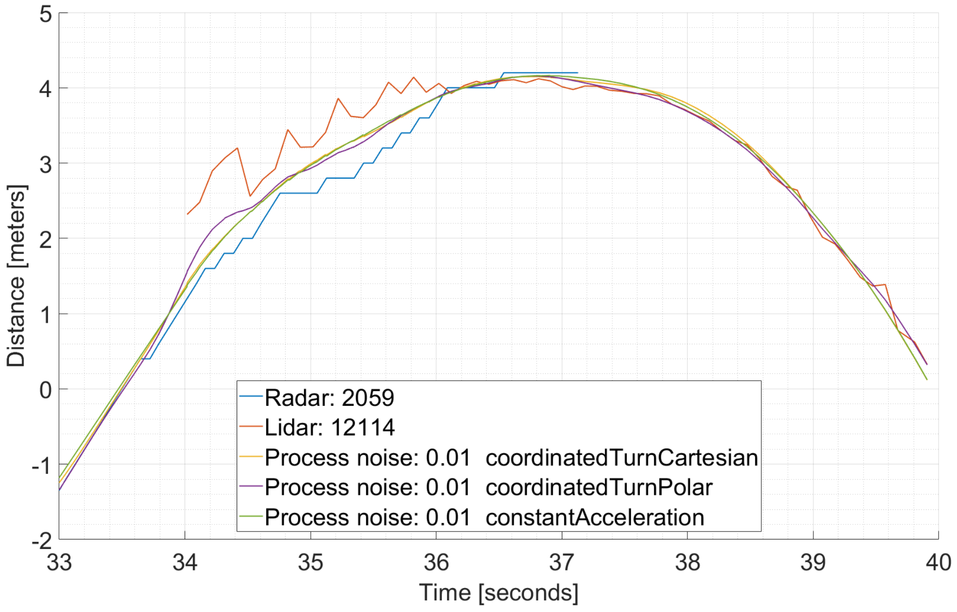


Figure 7.20: Latitudinal position estimate without observing heading from either sensor.

7.5 Stationary Observer Scenario

This scenario begins as the hunter vehicle approaches an intersection with a traffic light. As the car approaches the intersection the lights turns red and the vehicle stops. Before stopping completely the LIDAR detects a car to the left about to turn on to the road on which the hunter vehicle is traveling. As the hunter stops completely the other light turns green letting the car from the left on to the road, it enters the intersection before turning left and continuing along the road in the same direction as the hunter. A few frames of the scenario can be seen in Figure 7.21, the target car is encircled in red.

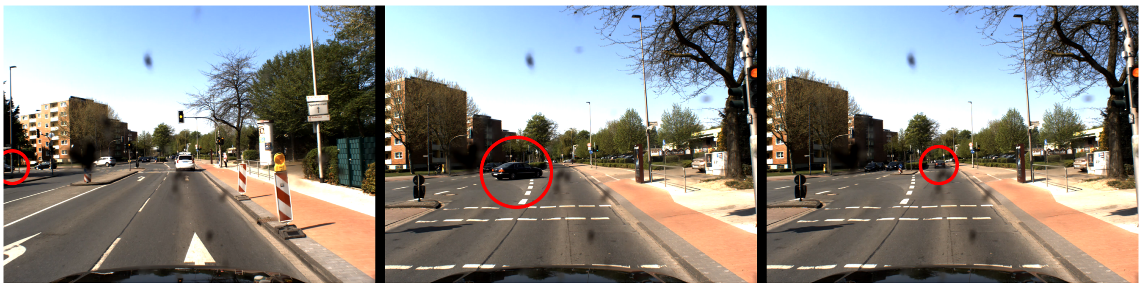
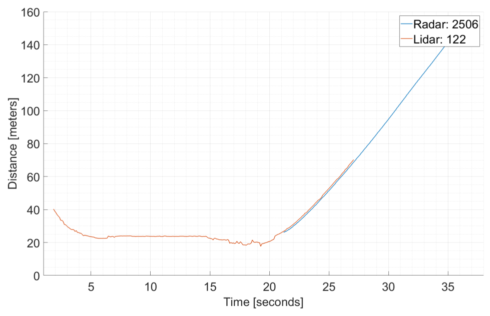
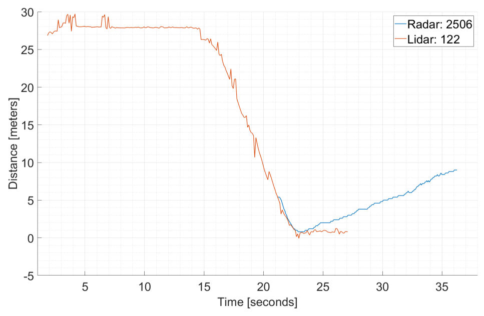


Figure 7.21: Frame sequence shows the car as it is discovered by the sensors, enters the intersection and starts heading down the road

As shown in Figure 7.22 the target vehicle is detected by the LIDAR a few seconds before the hunter comes to a stop which is why the longitudinal distance initially decreases before settling around 20 meters. The target vehicle then remains stationary for a couple of seconds before heading in to the intersection and into the radar sensors field of view. Both sensors then tracks the target as it finishes the turn on to the road and starts heading down that road, away from the hunter. Once the target passes beyond the 70 meter mark the LIDAR track ends while the radar sensor manages to maintain tracking to at least twice that distance.



(a) Longitudinal measurements



(b) Latitudinal measurements

Figure 7.22: The longitudinal and latitudinal relative position of the target vehicle as seen by the radar and LIDAR.

Figure 7.23 shows an enlarged view of the latitudinal position of the target over time. Here it is clear to see that the LIDAR tracking is plagued by faulty measurements through out, despite this the fused output remains stable with any of the tested motion models. Once the target is detected by the radar both measurements are used to estimate the latitudinal position and the fused output approaches the average of the two measured values.

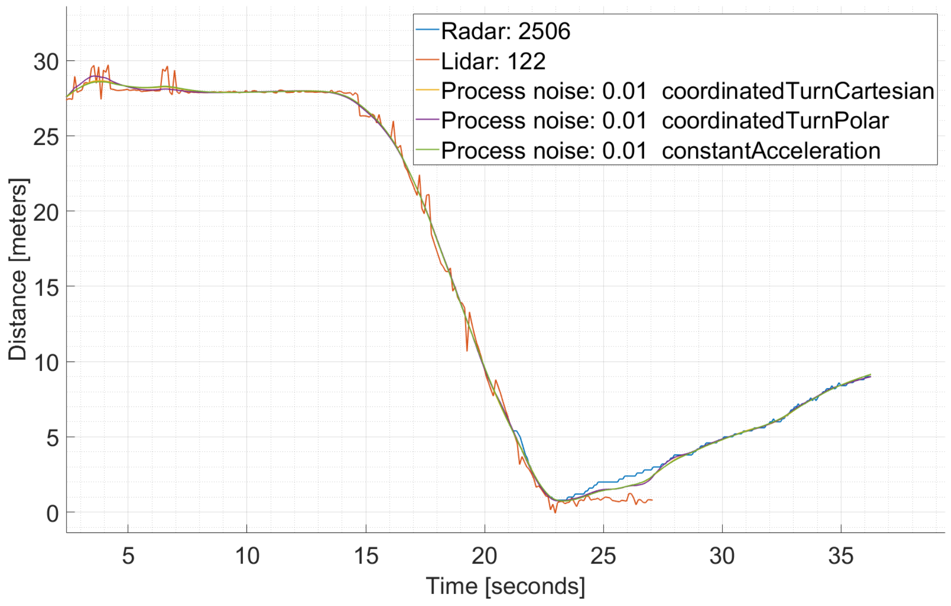


Figure 7.23: Latitudinal position of target entering from the left in the intersection, see Figure 7.21

In figure 7.24 the longitudinal position of the tracked target is shown. This graph shows that the LIDAR gives some faulty measurements along the longitudinal axis too. The fused output tracks the sensor measurements well and continues to track the radar measurements once the LIDAR loses its track of the target.

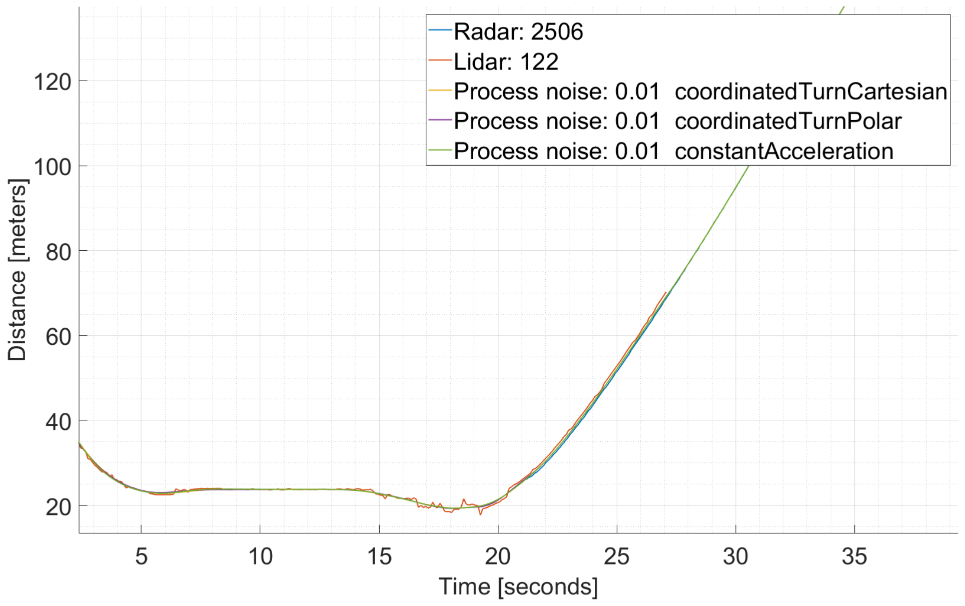


Figure 7.24: Longitudinal position of target entering from the left in the intersection, see Figure 7.21

8 Conclusion

The purpose of this project has been to implement a sensor fusion system using existing data. The goal with the system was to extract more value from the existing hardware setup. By using a modular approach when creating the code the system enables efficient evaluation and verification of various setups of the fusion algorithm, e.g switching motion models, measurement models and their respective tuneable parameters. Without the modularity it would have been difficult to find a good solution.

The purpose of the reference box is to record detailed information about the surrounding environment, e.g cars, trees, pedestrians, and other objects. The recorded information needs to maintain high quality in order to be a valid reference. In order to maintain a high quality output even in suboptimal conditions complementary sensors can provide a robustness that can not be matched by any of the individual sensors. Having a lot of sensor data from different sources may be inefficient during analysis, sensor fusion can then be a valuable tool to make the data more usable. As the results show both the radar and LIDAR provides accurate measurements in most situations but does sometimes give misreadings. In situations where one of the sensor inputs fail the RTS-smoother with its process model manages to mitigate this and generates a stable output.

For further development it would be beneficial to verify the fused output with an RTK reference, or similar reference solution. With such a reference the accuracy of the fused output can be verified with centimeter precision. Future work could be to continue the fusion of sensor data by fusing more of the data available from the radar and LIDAR. For example the object classification could be used to switch between motion models for pedestrians and vehicles. There are also additional sensors in the reference box which could be incorporated in the present solution to provide an even more complete reference system. The cameras could for example be used for classification purposes. With a well defined set of classes additional motion models could be added to account for different road-users movement, e.g pedestrians, animals, bicycles, cars and trucks.

Bibliography

[1] Continental, “Continental ARS 408-21.” [Online]. Available: https://www.continental-automotive.com/Landing-Pages/Industrial-Sensors/Products/ARS-408-21

[2] Velodyone LiDAR, “Velodyne HDL-64E.” [Online]. Available: https://velodynelidar.com/hdl-64e.html

[3] S. Singh, “Critical reasons for crashes investigated in the NMVCCS,” Washington DC, Tech. Rep. February, 2015. [Online]. Available: https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812115

[4] T. s. L. Oxford, “Real Time Kinematic Measurement Unit,” pp. 1–2, 2019.

[5] PathPartnerTech, “Understanding Radar for automotive (ADAS) solutions - PathPartnerTech,” 2018. [Online]. Available: https://www.pathpartnertech.com/understanding-radar-for-automotive-adas-solutions/

[6] B. Shaffer, “Why are automotive radar systems moving from 24GHz to 77GHz?” 2017. [Online]. Available: https://e2e.ti.com/blogs_/b/behind_the_wheel/archive/2017/10/25/why-are-automotive-radar-systems-moving-from-24ghz-to-77ghz#

[7] S. Merrill I, “An Introduction to Radar,” p. 11, 1962.

[8] J. Ryde and N. Hillier, “Performance of laser and radar ranging devices in adverse environmental conditions,” *Journal of Field Robotics*, vol. 26, no. 9, pp. 712–727, 2009.

[9] P. Merriaux, Y. Dupuis, R. Boutteau, P. Vasseur, and X. Savatier, “LiDAR point clouds correction acquired from a moving car based on CAN-bus data,” no. June, 2017. [Online]. Available: http://arxiv.org/abs/1706.05886

[10] Oxford Technical solutions Limited, “rt3000.” [Online]. Available: https://www.oxts.com/products/rt3000/?lang=sv

[11] H. Enders and T. Kruppi, “Radar PLC manual.” [Online]. Available: https://www.continental-automotive.com/getattachment/bffa64c8-8207-4883-9b5c-85316165824a/Radar-PLC-Manual-EN.pdf.pdf

[12] M. Ester, H.-P. Kriegel, S. Jorg, and X. Xu, “A Density-Based Clustering Algorithms for Discovering Clusters,” *Kdd*, vol. 96, no. 34, pp. 226–231, 1996.

[13] C. M. Bishop, *Pattern recognition and machine learning.* Springer-Verlag, Berlin, Heidelberg, 2006.

[14] R. E. Kalman, “A new approach to linear filtering and prediction problems,” *Journal of Basic Engineering*, vol. 82, no. 1, pp. 35–45, 1960.

[15] R. De Maesschalck, D. Jouan-Rimbaud, and D. L. Massart, “The Mahalanobis distance,” *Chemometrics and Intelligent Laboratory Systems*,

vol. 50, no. 1, pp. 1–18, 2000. [Online]. Available: https://www-sciencedirect-com.proxy.lib.chalmers.se/science/article/pii/S0169743999000477#!

[16] P. C. Mahalanobis, “On the generalized distance in statistics.” National Institute of Science of India, 1936.

[17] P. Grossman, “Multisensor data fusion,” *GEC Journal of Technology*, vol. 15, no. 1, pp. 27–37, 1998. [Online]. Available: https://www.researchgate.net/publication/293573229_Multisensor_data_fusion

[18] W. Elmenreich, “An introduction to sensor fusion,” Institut für Technische Informatik, Tech. Rep. August, 2002. [Online]. Available: https://www.researchgate.net/profile/Wilfried_Elmenreich/publication/267771481_An_Introduction_to_Sensor_Fusion/links/55d2e45908ae0a3417222dd9/An-Introduction-to-Sensor-Fusion.pdf

[19] B. V. Dasarathy, “Sensor fusion potential exploitation-innovative architectures and illustrative applications,” *Proceedings of the IEEE*, vol. 85, no. 1, pp. 24–38, 1997. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=554206

[20] S. Särkkä, *Bayesian Filtering and Smoothing.* Cambridge University Press, 2013.

[21] M. Lundgren, “Bayesian Filtering for Automotive Applications,” Ph.D. dissertation, Chalmers University of Technology, 2015.

[22] S. Julier, J. Uhlmann, and H. Durrant-Whyte, “A new approach for filtering nonlinear systems,” no. June, pp. 1628–1632, 2005.

[23] C. Stachniss, “Unscented Kalman Filter.” [Online]. Available: http://ais.informatik.uni-freiburg.de/teaching/ws13/mapping/pdf/slam06-ukf-4.pdf

[24] A. Berg and A. Käll, “Track-to-track Fusion for Multi-target Tracking Using Asynchronous and Delayed Data Master’s thesis in Systems, Control and Mechatronics,” Ph.D. dissertation, Chalmers University of Technology, 2017. [Online]. Available: http://publications.lib.chalmers.se/records/fulltext/250403/250403.pdf

[25] L. Svensson and J. Gunnarsson, “a New Motion Model for Tracking of Vehicles,” pp. 1376–1381, 2010.

[26] X. Yuan, F. Lian, and C. Han, “Models and algorithms for tracking target with coordinated turn motion,” *Mathematical Problems in Engineering*, vol. 2014, 2014.

[27] K. Granström, S. Reuter, D. Meissner;, and A. Scheel, “A multiple model PHD approach to tracking of cars under an assumed rectangular shape,” *17th International Conference on Information Fusion (FUSION)*, p. 1, 2014.