# Camera and radar based object tracking for cooperative driving

Stoffels, W.J.

*Award date:*
2021

Department of Mechanical Engineering
Dynamics and Control Research Group
System and Control

# Camera and radar based object tracking for cooperative driving

Master thesis

W.J. (Nikky) Stoffels (0849144)
DC-number: 2020.112

*Supervisors:*
dr. ir. T.P.J. van der Sande (TU/e)
Prof. dr. H. Nijmeijer (TU/e)

Department: Mechanical Engineering
Masters program: Systems and Control
Research group: Dynamics and Control

Eindhoven, Monday 1st March, 2021

EINDHOVEN UNIVERSITY OF TECHNOLOGY

## Declaration concerning the TU/e Code of Scientific Conduct for the Master's thesis

I have read the TU/e Code of Scientific Conduct[i].

I hereby declare that my Master's thesis has been carried out in accordance with the rules of the TU/e Code of Scientific Conduct

| | |
|---|---|
| Date | 23-11-2020 |
| Name | W.J. Stoffels |
| ID-number | 0849144 |
| Signature | WJStoffels |

*Insert this document in your Master Thesis report (2nd page) and submit it on Sharepoint*

[i] See: http://www.tue.nl/en/university/about-the-university/integrity/scientific-integrity/
The Netherlands Code of Conduct for Academic Practice of the VSNU can be found here also.
More information about scientific integrity is published on the websites of TU/e and VSNU

Version 202007

# Abstract

Autonomous vehicles are a current trend in research and development within the field of transportation. When considering automated driving, environmental awareness of the automated vehicle is one of the most important aspects. To sense the environment the vehicle needs to be equipped with sensors. These sensors are the key components of self-driving vehicles and are required to control the vehicle. For the vehicle to be able to drive automatically, it needs to be able to track objects over time.

The goal of this graduation project is to develop an object tracking algorithm that uses data generated by a radar and a camera sensor. To be able to track an object over time, detection, prediction and data association are required. Since the radar sensor can produce multiple detections for a single object, a clustering algorithm has been designed that creates clusters based on the difference in position and velocity between all actors. To predict the position and velocity of the object that is being tracked, an extended Kalman filter with a constant velocity model is used. To be able to correlate observation data generated by the sensors in the current-time step with already existing objects obtained from the estimations from the previous time steps a global nearest neighbour(GNN) method is used.

Since it was not possible to use real measurement data from a radar and camera sensor, synthetic measurement data has been used. To be able to test the developed object tracking algorithm four different test scenarios have been designed. In those four test scenarios several situations which can occur when driving on urban and provincial roads have been tested. The scenario consists of straight and curved roads with one or three actors, representing a car, truck and a bicycle. The obtained tracking results from these four test scenarios have been compared to a ground truth data set, which consists of the actual position and velocity of each actor and the generated measurement data by the radar and camera sensors.

The results obtained from these four test scenarios show that the object tracking algorithm is able to track all the different actors. For the actors driving in front of the ego vehicle the best tracking results are obtained, with small errors between the track and the reference of the actor. For the actors that are driving next to the ego vehicle (oncoming or overtaking vehicles) the errors between the track and the reference of the actor are slightly larger compared to the vehicles driving in front of the ego car. This is due to the fact that radar clusters are created at different locations on the actor, since the radar detects more of the side of these actors. This results in more tracks, located at different parts of the actor. The developed clustering algorithm is able to generate correct clusters, as long as the road users are not driving too close to each other.

# Acknowledgements

I would like to gratefully acknowledge various people who guided and supported me during my graduation project. First of all I would like to thank my supervisor dr.ir. Tom van der Sande for his guidance during the graduation project. The meetings we had helped me a lot to continue and improve the graduation project.
Secondly, I would like to thank my mentor Prof. dr. Henk Nijmeijer for the feedback and interesting discussions during the progress meetings.
Thirdly, I would like to thank all the members of 'the famous AES lab' for the fun we had during the first few months of the graduation project. Furthermore I want to thank my parents Jan and Simone and my brother Jimmy and sister Shanna for their support throughout the project and the cosiness during the months that I had to work from home due to the Corona pandemic.
Finally, I specially want to thank my boyfriend Glenn, for the wonderful years we already have had together since we met during the first week at the university.

# Contents

# List of Symbols

| Symbol | Description | Unit |
|---|---|---|
| $\theta_c$ | Elevation angle measurement of the camera | ° |
| $\theta_r$ | Elevation angle measurement of the radar | ° |
| $\sigma^2_{\phi r}$ | Variance of the azimuthal angle of the radar | ° |
| $\bar{\sigma}^2_{\phi r}$ | Variance of the azimuthal angle of the radar cluster | ° |
| $\sigma^2_{x_c}$ | Variance in $x$ for the camera | $\mathrm{m}^2$ |
| $\sigma^2_{y_c}$ | Variance in $y$ for the camera | $\mathrm{m}^2$ |
| $\sigma_{x_c}\sigma_{y_c}$ | Covariance of $xy$ for the camera | $\mathrm{m}^2$ |
| $\sigma^2_{rr}$ | Variance of the range of the radar | $\mathrm{m}^2$ |
| $\bar{\sigma}^2_{rr}$ | Variance of the range of the radar cluster | $\mathrm{m}^2$ |
| $\sigma^2_{\dot{r}r}$ | Variance of range rate of the radar | $\mathrm{m/s}^2$ |
| $\bar{\sigma}^2_{\dot{r}r}$ | Variance of the range rate of the radar cluster | $\mathrm{m/s}^2$ |
| $\sigma^2_{v_{x0}}$ | Initial state covariance in $v_x$ | $\mathrm{m/s}^2$ |
| $\sigma^2_{v_{y0}}$ | Initial state covariance in $v_y$ | $\mathrm{m/s}^2$ |
| $\sigma^2_{x0}$ | Initial state covariance for $x$ | $\mathrm{m}^2$ |
| $\sigma^2_{y0}$ | Initial state covariance for $y$ | $\mathrm{m}^2$ |
| $\phi_c$ | Azimuth angle of the camera | ° |
| $\phi_r$ | Azimuth angle measurement of the radar | ° |
| $\bar{\phi}_r$ | Azimuth angle of the core point of the radar cluster | ° |
| $B_k$ | Control input matrix at time $k$ | - |
| $C$ | Clustering matrix | - |
| $C_{ij}$ | Cost matrix | - |
| $c_{ij}$ | Cost element of the cost matrix | - |
| $d_{M,ij}$ | Mahalanobis distance between point $i$ and $j$ | - |
| $d_{ij}$ | Euclidean distance between radar point $i$ and $j$ | m |
| $D_{ij}$ | Boolean value for distance threshold | - |
| $d_{threshold}$ | Distance threshold value for clustering | m |
| $\mathbf{d}$ | Euclidean distance matrix | - |
| $\mathbf{F}_k$ | State transition matrix at time $k$ | - |
| $G_{threshold}$ | Gating threshold value | - |
| $\mathbf{G}$ | Process noise prediction matrix | - |
| $\mathbf{H}_k$ | Observation matrix at time $k$ | - |
| $\mathbf{h}(\mathbf{x}_k)$ | Non-linear measurement function | - |
| $\mathbf{I}$ | Identity matrix | - |
| $i$ | Index number | - |
| $J$ | Jacobian matrix | - |
| $j$ | Index number | - |
| $\mathbf{K}_k$ | Discrete Kalman gain at time $k$ | - |
| $k$ | Discrete time step | s |
| $M$ | Tuning parameter of the multi-object tracker | - |
| $N$ | Tuning parameter of the multi-object tracker | - |

| | | |
|---|---|---|
| $n$ | Number of clusters | - |
| $P_{coasted}$ | Number of times a the tracker is allowed to coast a confirmed track | - |
| $\mathbf{P}_{k\|k}$ | Updated covariance estimate at time $k$ | - |
| $\mathbf{P}_{k\|k-1}$ | Predicted state covariance at time $k$ | - |
| $\mathbf{Q}_k$ | Process noise covariance matrix at time $k$ | - |
| $q_x$ | Variance of possible change in $x$-direction | - |
| $q_y$ | Variance of possible change in $y$-direction | - |
| $r_c$ | Radial distance measurement of the camera | m |
| $\dot{r}_c$ | Radial velocity measurement of the camera | m/s |
| $R_{c,cart}$ | Covariance matrix of the camera in Cartesian coordinates | - |
| $R_k$ | Measurement noise covariance matrix at time $k$ | - |
| $R_r$ | Covariance matrix of the radar | - |
| $r_r$ | Radial distance measurement of the radar | m |
| $R_{r,cluster}$ | Measurement noise covariance matrix for the radar cluster | - |
| $\dot{r}_r$ | Radial velocity measurement of the radar | m/s |
| $\bar{r}_r$ | Radial distance of the core point of the radar cluster | m |
| $\bar{\dot{r}}_r$ | Radial velocity of the core point of the radar cluster | m/s |
| $\mathbf{S}_k$ | Innovation covariance at time $k$ | - |
| $T$ | Sample time | s |
| $\mathbf{u}_k$ | Input vector | - |
| $\mathbf{v}$ | Velocity matrix | - |
| $\mathbf{v}_k$ | Measurement noise vector at time $k$ | - |
| $V_{r,ij}$ | Boolean value for velocity threshold | - |
| $v_{r,ij}$ | Velocity difference of the radar between point $i$ and $j$ | m/s |
| $v_{threshold}$ | Velocity threshold value for clustering | m/s |
| $v_{x0}$ | Initial velocity in $x$-direction | m/s |
| $v_x$ | Velocity in $x$-direction of the vehicle | m/s |
| $v_{y0}$ | Initial velocity in $y$-direction | m/s |
| $v_y$ | Velocity in $y$-direction of the vehicle | m/s |
| $\mathbf{w}_k$ | Process noise vector at time $k$ | - |
| $x$ | $x$ position tracker | m |
| $x_{ego}$ | $x$-position of ego vehicle | m |
| $x_0$ | Initial $x$-position of the tracker | m |
| $x_c$ | $x$-position of camera measurement | m |
| $x^i_{offset}$ | Sensor offset in $x$-direct of sensor $i$ ion | m |
| $x_k$ | State vector at time $k$ | - |
| $\bar{x}_r$ | $x$-position of the core point of the radar cluster | m |
| $\hat{\mathbf{x}}_{k\|k}$ | Predicted state estimate at time $k$ | - |
| $x_{world}$ | $x$-position in world coordinates | m |
| $y$ | $y$ position tracker | m |
| $y_c$ | $y$-position of camera measurement | m |
| $y_{ego}$ | $y$-position of ego vehicle | m |
| $\tilde{\mathbf{y}}_k$ | Innovation or measurement residual at time $k$ | - |
| $y^i_{offset}$ | Sensor offset in $y$-direction of sensor $i$ | m |
| $\bar{y}_r$ | $y$-position of the core point of the radar cluster | m |
| $y_{world}$ | $y$-position in world coordinates | m |
| $y_0$ | Initial $y$-position of the tracker | m |
| $\mathbf{z}_k$ | Measurement vector at time $k$ | - |
| $z_{combined,k}$ | Structure that consists of combined radar and camera data | - |
| $z_{c_{combined},k}$ | Structure that consists of combined camera data | - |
| $z_{r_{combined},k}$ | Structure that consists of combined radar data | - |

Chapter 1

# Introduction

Autonomous vehicles are a current trend in research and development within the field of transportation. When considering automated driving, environmental awareness of the automated vehicle is one of the most important aspects. For the vehicle to be able to drive automatically, it needs to be able to detect and subsequently track objects over time. This means the vehicle knows where objects are and can predict their evolution over time. To be able to sense the environment the vehicle needs to be equipped with a number of perception systems and sensors. These sensors are the key components of self-driving vehicles and are required to be able to control the vehicle. Sensors present in a modern vehicle can be divided into a number of subgroups: internal sensors measuring the motion of the vehicle, sensors that are communicating with other vehicles and with the infrastructure and external sensors measuring the objects surrounding the vehicle [28].
Tracking an object is not the same as detecting an object. Detection is the process of locating an object of interest in a single frame. Target tracking associates detections of objects across multiple frames. To be able to track objects, detection, prediction and data association are required [36].

Detection is done by the sensors present on the vehicle. All the sensors used for (self-driving) vehicles have their own strengths and weaknesses and create an image of the driving environment in their own way. This results in a large set of different views of the same environment. However, to control the automated vehicle, this has to be one coherent image. Therefore, measurement data from various sensors needs to be merged into one usable data set. Merging different data sets is also called sensor fusion. By fusing data from different sensors, the advantages of each sensor can be combined.
Prediction and estimation is done by a filter that will determine the state of the target under movement given the observations or measurements. The state estimation phase is a common stage in data fusion algorithms because the target's observation could come from different sensors, because the final goal is to obtain a global target state from the observations [5]. There are a lot of methods available for sensor fusion, but one of the most popular algorithms in data fusion is the Kalman filter [9]. The objective of sensor fusion is to create a more accurate and reliable state estimation compared to only using a single sensor.
Another crucial part of the tracking algorithm is data association. This part will correlate observation data obtained in a current time-step with already existing objects, thanks to their estimation from the previous time step. A data association algorithm serves to extract object measurements from the clutter and to separate multiple objects [24]. By designing suitable algorithms for prediction and data association, an algorithm can be created that is able to estimate the states of multiple targets over time. Such an algorithm is of importance for several applications used in automated vehicles, such as forward collision warning (FCW) and path planning [53].

## 1.1 Research objective

The goal of the graduation project is to create an algorithm that is able to track multiple objects. The tracking algorithm will receive measurements from a radar and a camera sensor. Both sensors will have different outputs, due to their own sensor characteristics. The measurement data generated by these sensors is used to estimate the state of the object and track the object over time. The tracking algorithm will be tested for different scenarios. The graduation project has the following research objective:

*Develop and test an algorithm that is able to track multiple objects by fusing onboard radar and camera sensor data.*

To be able to reach the goal of the graduation project, the project is divided in several tasks:

- Obtain measurement data and adapt the data in such a way that it can be used as an input for the tracking algorithm[1].
- Obtain an algorithm that is able to cluster data generated by the radar sensor.
- Determine which algorithm can be used to associate the measurement data.
- Determine which filter can be used to estimate the state of the objects.
- Model the object tracking algorithm and test and validate the tracking model by making use of a simulation environment.

## 1.2 Outline

In Chapter 2 a literature review with respect to object tracking is given. In this chapter several methods that can be used for radar clustering and object tracking are discussed. In Chapter 3 the radar and camera sensors are discussed in more detail. Chapter 4 describes the object tracking algorithm and all the different parts of the model such as clustering, state estimation and data association. In Chapter 5 several test scenarios are described. Furthermore, it is described how the parameters that are part of the object tracking algorithm are determined and tuned. In Chapter 6 the designed scenarios are used to test the obtained object tracking algorithm. The results for each scenario are discussed with different sensor characteristics to be able to draw a more detailed conclusion about the performance of the object tracking algorithm. Lastly, conclusions and recommendations are given in Chapter 7.

[1]At first the plan was to use the radar and camera sensor that are mounted on a Renault Twizy and to perform measurements with the Twizy to test the object tracking algorithm. These Renaults Twizy's are part of the i-Cave project, a research program of the TU/e that addresses current transportation challenges regarding throughput and safety with an integrated approach to automated and cooperative driving. But during the project there suddenly was the COVID-19 pandemic, also known as the Corona-virus pandemic, which started at the end of 2019. To control the spread of the virus, public buildings were closed among which our university, which forced all the employees and students to work from home for the rest of the year. Since the progress of the spread of the virus is very insecure and it was not clear when the university could reopened for students, it is decided to use only synthetic measurement data and test the object tracking algorithm in a simulation environment only.

# Chapter 2

# Literature review

In this chapter literature with respect to object tracking will be discussed. To be able to track objects over time, detections, predictions and data association are required [36]. First, it will be discussed why a radar and camera sensor are used and how these sensors obtain their measurements. Since radar measurements can consist of multiple detections that can originate from the same object, a clustering method can be used to group together the measurements that originate from the same object. In this chapter several methods are discussed that can be used for clustering. Furthermore, several methods are discussed that can be used to fuse the obtained radar and camera data and to predict the state of the objects over time. Finally, several data association methods will be discussed that can be used to link the obtained state predictions to the correct track of each object.

## 2.1 Radar and camera sensors

A key part of self-driving vehicles are the sensors. Autonomous vehicles need sensors to collect data about the vehicle and its environment. Sensors in modern vehicles can be divided into a number of subgroups: internal sensors measuring the motion of the vehicle, external sensors measuring the objects surrounding the vehicle and sensors which are able to communicate with other vehicles and with the infrastructure [28]. Examples of external sensors are radar sensors, lidars and camera systems. Often a combination of radar and camera systems are used to combine the advantages of both sensors. In [51] a systematic review of perception systems such a a radar and camera sensors is given. An overview of the properties of these two sensors is given in Table 2.1 [28].

| | Radar | Camera |
|---|---|---|
| Detects | Other vehicles, pedestrians, cyclists and stationary objects | Other vehicles, pedestrians/cyclists, lane markings |
| Classifies object | No | Yes |
| Azimuth angle | Medium accuracy | High accuracy |
| Range | Very high accuracy | Low accuracy |
| Range rate | Very high accuracy | Not |
| Field of view | narrow | wide |
| Weather conditions | Less sensitive | Sensitive to bad weather conditions |

Table 2.1: Radar and camera properties with respect to object detection [28]

In [43] the working of an automotive radar is discussed in detail. Radar stands for Radio Detection and Ranging and consists of a transmitter and a receiver. Radar operates by transmitting radio frequency electromagnetic waves and analyses the signals reflected from objects surrounding the vehicle. By measuring the time delay and phase shift of received signals, the angle, radial distance and radial velocity of an object can be determined. While other sensors measure the velocity by calculating the difference between two data points, radar uses the Doppler effect to measure the velocity directly. The fact that the velocity is measured directly by the radar sensor can be useful for sensor fusion. Some disadvantages of radar sensors are the lack of precision, its reduced Field of View (FOV) and the production of false positives due to reflections compared to a camera sensor [51].

In [40] stereo vision systems are discussed. Humans acquire information about the location and other properties of objects in the environment from their eyes. Humans are able to see depth due to the difference between images formed by the left and right eye. Stereo vision systems are inspired by the biological process to extract three-dimensional information from digital images, which can be used to perform 3D reconstructions, tracking, and detection of objects [42]. A stereo camera consists of two cameras of the same type and with the same specifications. The distance to an object can be measured when the object is within the overlapping field of view of the two cameras. Two images from both cameras are merged to generate a disparity map. The disparity map in stereo vision represents the depth of an object. The main disadvantage of vision systems is that a camera is sensitive for variations in lighting and certain weather conditions. Therefore often a combination of radar and camera sensors is used to increase its robustness [51].

## 2.2 Radar clustering

Data generated by a radar sensor can consist of multiple detections at different angles and ranges which originate from the same target. Clustering methods are used to group together data according to some kind of similarities in one or more variables [45]. A distance function between the data points is usually used as a measure of similarity [14]. In [46] various clustering methods and algorithms are discussed and compared. Clustering algorithms can be divided in four different categories: hierarchical based methods, partition based methods, density-based methods and grid-based methods. Both hierarchical and partition-based methods are highly efficient with normal shaped clusters. But for arbitrary shaped clusters and detecting outliers, density or grid-based methods are more efficient.
A well-known clustering method is the K-Means algorithm [2]. This clustering algorithm is a distance-based algorithm, where the distances between all the data points are calculated to assign a point to a cluster. The biggest challenge for the K-Means method is to define the number of cluster ”K”, since the number of clusters is often not known beforehand. Therefore in the field of radar signal processing the most common algorithm is the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [13]. The advantage of the DBSCAN algorithm over the K-means algorithm, is that a fewer number of parameters is used for the algorithm and no prior knowledge is necessary about the number of clusters. Furthermore, there is no limit for the dimensions and size of the input data of the clustering algorithm. A disadvantage of the DBSCAN algorithm is that in some cases it is difficult to determine an appropriate distance threshold value for the clusters to be met. Furthermore the DBSCAN is not suited for situations where the densities of the clusters are varying a lot, since it is hard to determine one parameter set to get the correct clusters for the complete data set.
In [45] and [13] an extension of the DBSCAN algorithm, the Grid-based DBSCAN (GB-DBSCAN) is discussed. In [32], it is proven that the GB-DBSCAN algorithm will produce similar results compared to the DBSCAN algorithm, but in theory the GB-DBSCAN algorithm should be faster than the DBSCAN algorithm. The main challenge for the Grid-Based DBSCAN algorithm is to determine an appropriate size of the grid structure. If the size of the grid is not the correct size, incorrect clusters will be generated.
In [19] an extended density-based clustering algorithm for radar applications is discussed. For this clustering method, clusters will not be created based on position only, but also velocity is taken into account. Density-based clustering alone is not always sufficient for radar applications, as a density-based clustering algorithm only based on position, without velocity measurements, can give faulty results. From [19] it can be concluded that including velocity-based clustering besides position-based clustering gives better clustering results than only using a density-based clustering algorithm.

## 2.3 Sensor fusion

As described above radar and camera sensors both have their advantages and disadvantages. By combining the observations from both sensors a more complete, robust and precise representation of the environment of interest can be obtained. Sensor fusion is the process of using information from several (different) sensors to compute an estimate of the state of a dynamic system [28]. According to [20] there are a several advantages of using multiple sensors for data fusion instead of using a single sensor system:

- **Increased reliability**: by using multiple sensors the reliability can increase, since each sensor performance different under different circumstances.
- **Extended spatial coverage**: Each sensor covers a different area. By using multiple sensors the coverage

area can increase.

- **Extended temporal coverage**: Each sensor may have a different update rate. By interpolating sensor updates the temporal coverage can increase.
- **Increased confidence**: By combining sensor data, confidence will increase since it will be less sensitive for uncertainties.
- **Reduced uncertainty**: By combining sensors the overall uncertainty of the system can be reduced since each sensor has its own uncertainty.
- **Increased resolution**: By combining observations from multiple sensors the resolution of the measurements can increase.

So by combining sensor data, the estimation of the states can be improved. Since the measurement data from the sensors will contain random errors, that might come from e.g. hardware limitations, environment noise or technical limitations and failures [7], an algorithm is needed that combines the available measurement data. There are different algorithms available for different levels of fusion [17]. At low-level fusion, fusion takes place at data-level and raw output data from the sensors is used. Fusion at this level can be used in real-time applications. Sensor fusion at medium level is also called feature fusion. Data fusion at this level is realized by concatenating the feature points obtained from multiple sources, to result in a feature with higher discrimination [48]. High-level fusion takes place at the decision level and uses a symbolic representation of the process parameters. For each level of fusion different algorithms for sensor fusion can be used. An overview of several algorithms for the different levels can be seen in Table 2.2, which is based on [29].

| **Low-level fusion** | **Medium level fusion** | **High level fusion** |
|---|---|---|
| **Estimation methods** | **Classification methods** | **Inference methods** |
| *Recursive*: | Parametric templates | Bayesian inference |
| Kalman filter | Cluster analysis | Particle filters |
| Extended Kalman filter | K-means clustering | Dempster-Shafer theory |
| Unscented Kalman filter | Leaning vector quantization | Expert system |
| | Kohonen feature map | Fuzzy logic |
| *Non-recursive:* | Artificial neural network | |
| Weighted average | Support vector machines | |
| Least squares | | |
| | | |
| Covariance-based | | |
| Cross covariance | | |
| Covariance intersection | | |
| Covariance union | | |

Table 2.2: Classification of fusion algorithms

From Table 2.2 it can be concluded low-level fusion algorithms make use of estimation methods like a Kalman filter that uses raw measurement data. While higher level fusion methods are not using raw measurement data directly. Therefore low-level sensor fusion has several advantages compared with higher-level fusion:

- It is possible to exploit information from one sensor while interpreting sensor data from another sensor. In general this results in an improvement of the robustness of the algorithm [55].
- The information loss is minimized [6].
- It reduces the risk of adopting inconsistent modelling assumptions while processing data from different sensors [10].

One of the disadvantages of low-level sensor fusion is that a lot of data needs to be transferred from the sensor to the platform that runs the estimation algorithm. An advantage of higher-level sensor fusion compared to low-level sensor fusion is that a limited amount of data needs be transferred [12]. But to accomplish this the sensors must be able to process data on-board and deliver only data at tracked object level. The disadvantages of higher-level sensor fusion are the opposite of the advantages of the low-level sensor data fusion. Since (raw) sensor data is available from both the radar and camera sensors, sensor fusion can take place at the lowest level. Therefore the linear Kalman filter, extended Kalman filter and the unscented Kalman filter will be discussed and explained in more detail below.

### 2.3.1 Kalman filter

The Kalman filter is one of the most widely used filters for tracking moving objects [9]. With the filter the state of the detected objects can be estimated by making use of the measurements [8]. The state of the object can consist of the position, velocity or acceleration of the object. The filter algorithm involves two stages: the prediction stage and the measurement update [16]. It uses a model of how the states change over time and a model of how the observations from the sensors are related.

**Kalman filter**
The linear Kalman filter can be used for linear dynamical systems. In linear form, the states of the detected object can be described by the following model:

$$\mathbf{x}_k = \mathbf{F}_k \mathbf{x}_{k-1} + \mathbf{B}_k \mathbf{u}_k + \mathbf{w}_k \tag{2.1}$$

where $\mathbf{x}_k$ is the state vector, $\mathbf{F}_k$ is the state transition matrix, $\mathbf{B}_k$ is the input matrix, $\mathbf{u}_k$ the input vector and $\mathbf{w}_k$ the process noise vector. The process noise is assumed to have zero mean Gaussian noise with a covariance given by the covariance matrix $\mathbf{Q}_k$. In general for vehicle applications several motion models are used such as random, constant velocity, constant acceleration and Singer models [7].

Measurements generated by the sensors in linear form can be described by the following observation model:

$$\mathbf{z}_k = \mathbf{H}_k \mathbf{x}_k + \mathbf{v}_k \tag{2.2}$$

where $\mathbf{z}_k$ contains the vector of measurements, $\mathbf{H}_k$ is the observation matrix and $\mathbf{v}_k$ is the vector that contains the measurement noise. The process noise $\mathbf{w}_k$ and measurement noise $\mathbf{v}_k$ are independent from each other. It is also assumed that the measurement noise is zero mean Gaussian noise with a covariance matrix given by $\mathbf{R}_k$.

The state estimate can be predicted using the following equation:

$$\hat{\mathbf{x}}_{k|k-1} = \mathbf{F}_k \hat{\mathbf{x}}_{k-1|k-1} + \mathbf{B}_k \mathbf{u}_k \tag{2.3}$$

where $\hat{\mathbf{x}}_{k|k-1}$ is the predicted state vector. The calculation of the predicted state estimate is based on the state estimate and the input of the previous time step.

The second part of the prediction stage consists of predicting the covariance of the state. For calculating the predicted state covariance, the predicted state covariance of the previous time step is used and can be calculated by the following equation:

$$\mathbf{P}_{k|k-1} = \mathbf{F}_{\mathbf{P}k-1|k-1} \mathbf{F}_k^T + \mathbf{Q}_k \tag{2.4}$$

where $\mathbf{P}_{k|k-1}$ is the predicted state covariance, that contains information about the variances and covariances of the states and $\mathbf{Q_k}$ is the process noise covariance matrix. The diagonal terms of the matrix are the variances associated with the corresponding terms in the state vector. The off-diagonal terms are the covariances between the terms present in the state vector.

In the update stage of the algorithm the first step is to determine the innovation or measurement pre-fit residual $\tilde{\mathbf{y}}_k$. This residual can be calculated by:

$$\tilde{\mathbf{y}}_k = \mathbf{z}_k - \mathbf{H}_k \hat{\mathbf{x}}_{k|k-1}. \tag{2.5}$$

The next step in the update stage is to calculate the innovation covariance $\mathbf{S}_k$ by:

$$\mathbf{S}_k = \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^T + \mathbf{R}_k. \tag{2.6}$$

where $\mathbf{R_k}$ represents the measurement noise covariance matrix. The predication and update stage are combined using the Kalman gain $\mathbf{K}_k$, which is calculated to minimize the mean-square error of the state estimate. The Kalman gain can be calculated by using the residual of the covariance calculated in (2.6) by:

$$\mathbf{K}_k = \mathbf{P}_{k|k-1} \mathbf{H}_k^T \mathbf{S}_k^{-1}. \tag{2.7}$$

The calculated Kalman gain says something about how much the estimate of the system needs to be changed by a given measurement.

Finally, the state and covariance estimates will be updated by:

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \tilde{\mathbf{y}}_k \tag{2.8}$$

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_{k|k-1} \tag{2.9}$$

where $\hat{\mathbf{x}}_{k|k}$ is the updated state estimate, $\mathbf{P}_{k|k}$ is the updated covariance estimate and $\mathbf{I}$ is the identity matrix.

The Kalman filter works according to a cycle which starts with an estimate, generation of a prediction, obtainment of an observation and then an update of the prediction to an estimate [11]. The filter uses the process model to generate a prediction and the observation model generates an update. In Figure 2.1 an overview of the operation of a Kalman filter is shown.

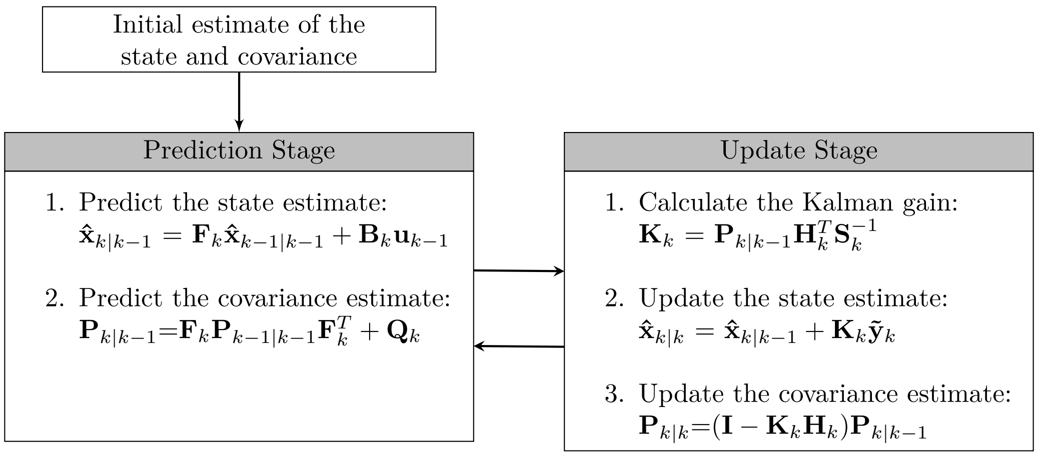

Initial estimate of the state and covariance
Prediction Stage
1. Predict the state estimate: $\hat{\mathbf{x}}_{k|k-1} = \mathbf{F}_k \hat{\mathbf{x}}_{k-1|k-1} + \mathbf{B}_k \mathbf{u}_{k-1}$
2. Predict the covariance estimate: $\mathbf{P}_{k|k-1} = \mathbf{F}_k \mathbf{P}_{k-1|k-1} \mathbf{F}_k^T + \mathbf{Q}_k$
Update Stage
1. Calculate the Kalman gain: $\mathbf{K}_k = \mathbf{P}_{k|k-1} \mathbf{H}_k^T \mathbf{S}_k^{-1}$
2. Update the state estimate: $\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \tilde{\mathbf{y}}_k$
3. Update the covariance estimate: $\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_{k|k-1}$

Figure 2.1: Overview of the operation of the Kalman Filter

An advantage of the Kalman filter is that it is suited for real-time problems, since it has a high computational efficiency. This high computational efficiency is due to the fact that it only needs information about the previous state [17]. The disadvantage of the linear Kalman filter is that it is restricted to linear system dynamics and it assumes that the probability density function is Gaussian. In reality often observations are not linear since the motion of the object often is non-linear, but there can also be non-linearties in the measurement model [21]. Therefore the linear Kalman filter discussed above often cannot be used in tracking problems. If a linear Kalman filter (KF) is not suitable other variants of the Kalman filter can be used such as the extended Kalman filter (EKF) and unscented Kalman filter (UKF).

**Extended Kalman filter**
The extended Kalman Filter is the most popular non-linear estimation algorithm due to its computational efficiency [3]. The extended Kalman filter is obtained by linearizing the signal model about the current state estimate and using the linear Kalman filter to predict the state estimate of the next time step. Therefore, the state and observation models do not need to be linear functions and can be of the form:

$$\mathbf{x}_k = \mathbf{f}(\mathbf{x}_{k-1}, \mathbf{u}_k) + \mathbf{w}_k \tag{2.10}$$

$$\mathbf{z}_k = \mathbf{h}(\mathbf{x}_k) + \mathbf{v}_k \tag{2.11}$$

The non-linearities present in the system are approximated by a first-order linearized version of the nonlinear model around the last state estimate [49]. Linearization of the system is done by calculating the following Jacobians:

$$\mathbf{F}_k = \left.\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right|_{\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_k} \tag{2.12}$$

$$\mathbf{H}_k = \left.\frac{\partial \mathbf{h}}{\partial \mathbf{x}}\right|_{\hat{\mathbf{x}}_{k-1|k-1}} \tag{2.13}$$

The rest of the equations used for the prediction and update stages are similar to the linear Kalman filter discussed earlier. As stated in [49], the Extended Kalman filter is not an optimal filter since it is implemented based on a set of first-order approximations. Due to the fact that the extended Kalman filter is not optimal, it is possible that the filter diverges if the linearization of the estimate of the state is not approximated precisely enough compared to the exact model.

**Unscented Kalman filter**
The unscented Kalman filter is an improved version of the extended Kalman filter for highly non-linear systems and addresses the problems of sub optimal performance and divergence of the filter [52]. The difference between EKF and UKF is that in EKF only one point is used for linearization, while for the UKF several points, called sigma points are used. By taking multiple points the approximation will be better. This set of sigma points is a set of carefully chosen sample points, which captures the true mean and covariance of the Gaussian random variables [52]. The UKF achieves higher accuracy than the first-order accuracy obtained for the EKF, while the computational complexity of the UKF is of the same order as that of the EKF [52].

## 2.4 Data association

In multi-object tracking algorithms data association plays an important role for the success of the object tracking algorithm [56]. If there is more than one object to track, the problem of associating measurements with the correct object arises. The goal of data association is to establish the set of observations or measurements that originate from the same object over time [5]. Data association is a key step in object tracking, because the object tracking algorithm will behave incorrectly if the data association step does not work coherently [5]. There are several algorithms that can be used for data association. In [37] several techniques and methods are discussed that can be used to solve the data association problem.

The nearest neighbour (NN) method is the simplest data association algorithm available [30]. The nearest neighbour method uses a measure of the distances between all objects and all the detections to assign each measurement to its nearest object. The nearest neighbour algorithm is a single-target data association method, since multiple detections can be assigned to a single target. The nearest neighbour method can be extended to multiple targets by using the global nearest neighbour (GNN) method [4]. The nearest neighbour method is only locally optimal. When using the global nearest neighbour method, distances between all measurements and tracks are considered when calculating the association assignments [4]. The global nearest neighbour method minimizes the total distance of all the possible associations between object and measurement. The distances between all the objects and detections are determined and a cost matrix is constructed from these distances. This cost matrix is used to assign each detection to a track if possible. The GNN approach works well if the sensor data is not too noisy and the observations and known objects are sufficiently far apart from each other [7].

The probabilistic data association method combines all the potential candidates for association to one track into a single statistic model, taking into account the statistical distribution of the track errors and clutter

[7]. The algorithm is able to associate different measurements to one specific target. The PDA algorithm will provide a poor performance when the targets are too close to each other. The PDA algorithm is a single-target data association method and behaves incorrectly when there are multiple targets. Another disadvantage of the PDA algorithm is that it assumes that there is already a track established. Therefore other algorithms must be provided that are able to initialize and delete tracks if needed. The probabilistic data association (PDA) method can be extended for multiple objects to the joint probabilistic data association (JPDA) method. A disadvantage of the JPDA method is, that also this method needs an algorithm that is able to initialize and delete tracks. Another disadvantage of JPDA compared with PDA is that it is computationally more expensive when it is applied for environments where multiple targets are present.

One of the most popular data association methods is the multiple hypothesis tracking method [12]. The multiple hypothesis tracking algorithm stores all the hypothesis associations from several frames until the information is considered enough to take an association decision [7]. The MHT method was developed to be able to track multiple targets in a cluttered environment. The main disadvantage of the MHT method is that it has a high computational cost, which grows exponentially with the number of tracks and measurements [5]. This computational cost can be reduced by limiting the growth in track hypotheses. To limit this growth a pruning strategy based on the likelihood can be implemented [11].

Both the GNN and JPDA method can have difficulties with data association in an environment where objects are close to each other. In such a situation there are multiple objects and multiple detections, which result in ambiguities. The GNN method gives an optimal solution is such a situation [25]. Furthermore, it can be concluded that in terms of computation time the GNN method is the most efficient compared to the computation time required for the JPDA and MHT method.

## 2.5 Summary

In this chapter a literature review is given with respect to all the key components of an object tracking algorithm. First an overview is given of the advantages of using a radar and camera sensor. Since radar data can consist of multiple detections originating from the same target, it can be useful to cluster the generated radar data. Within this chapter several clustering algorithms have been discussed. From the literature it can be concluded that a clustering algorithm that generates clusters based on position and velocity gives the best clustering result. To be able to fuse the generated radar and camera measurements a Kalman filter can be used. An overview and discussion of several Kalman filters have been given in this chapter. For non-linear systems the EKF or UKF can be used for estimation. The EKF is the most popular estimation algorithm filter for non-linear systems, while the UKF often is used for highly non-linear systems. Furthermore, several commonly used data association techniques have been discussed. From the discussed data association methods it can be concluded that the GNN method has the lowest computational load and it works well if the objects are not too close to each other.

# Chapter 3

# Sensor data

To be able to track objects over time detections are necessary. For this graduation project a radar and camera sensor are used to generate the required sensor data that will be used as an input for the object tracking algorithm. First the radar and camera sensors that are used on the Twizy will be discussed. Since it was no longer possible to generate real measurement data, synthetic radar and camera data has been used during the project. In this chapter it is explained how this synthetic data has been obtained from a designed scenario and a radar and camera sensor generator.

## 3.1 Radar and camera sensor

The Renault Twizy used in the i-Cave project is equipped with one single radar sensor mounted on the front bumper of the vehicle and a camera which is mounted on top of the vehicle, as can be seen in Figure 3.1. In the future the vehicle will be equipped with five radar sensors, such that 360° around the vehicle is covered.

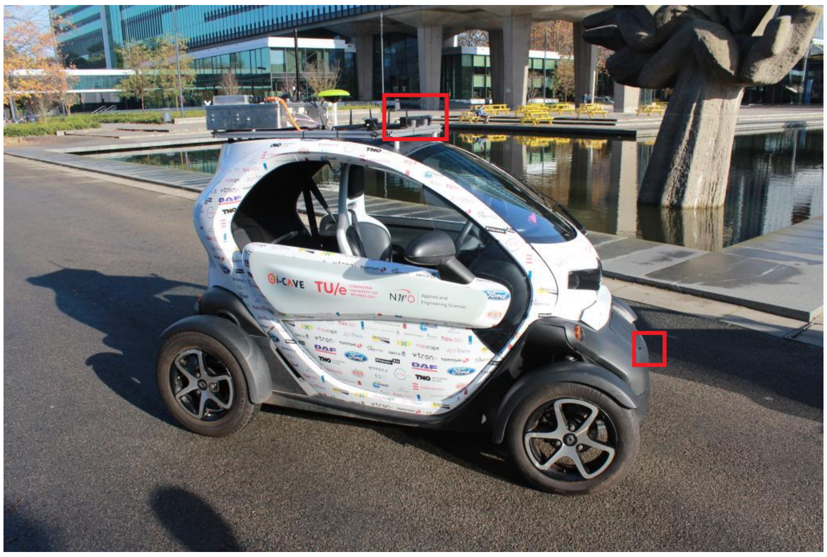


Figure 3.1: Renault Twizy used in the i-Cave project

The radar that is used for the project is a NXP Cocoon radar [39]. A radar such as the Cocoon receives reflections of objects. Data generated by such a sensor can consist of multiple detections originating from the same target. The Cocoon radar offers different modes for short range and ultra short range configurations. For each of these modes two non-overlapping frequency spectra are available to avoid interference [38]. In Table 3.1 the specifications for the ultra short and short range configurations can be found.

| | Ultra short range configuration | Short range configuration |
|---|---|---|
| Effective bandwidth | 1500 MHz | 275 MHz |
| Effective center frequency | 79 GHz | 78 GHz |
| Measurement distance | 12.8 m | 69.8 m |
| Minimum distance | 0.1 m | 0.75 m |
| Distance resolution | 0.1 m | 0.55 m |
| Maximum relative approaching velocity | 250 km/h | 250 km/h |
| Maximum relative receding velocity | 150 km/h | 150 km/h |
| Velocity resolution | 1 km/h | 1 km/h |
| Azimuth field of view (FOV) at 1 m | ± 60° | ± 60° |
| Elevation field of view (FOV) at 1 m | ± 10° | ± 10° |
| Angular resolution | No | No |
| System cycle | 70 ms | 70 ms |

Table 3.1: Specifications for the NXP Cocoon radar sensor for two configurations [38]

The Cocoon radar sensor outputs are given in spherical coordinates. In Figure 3.2 an overview of what the radar sensor measures can be seen.

The outputs that are generated by the radar sensor are the:

- Azimuth angle ($\phi_r$): angle between the longitudinal axis ($x$-axis) and a detected object
- Elevation angle ($\theta_r$): angle between the vertical axis ($z$-axis) and the detected object
- Radial distance ($r_r$): distance between the radar and detected object
- Radial velocity ($\dot{r}_r$): velocity of the detected object in the direction of the radial distance.

The stereo-vision system that is mounted on the Renault Twizy consists of two Leopard imaging LI-AR0231-GMSL-060H cameras. In Table 3.2 some specifications of the camera can be seen.

| | Camera specification |
|---|---|
| Maximum resolution | 1928H x 1208V |
| Pixel size | 3 µm x 3 µm |
| Full resolution | 30 fps |
| Clock input | 27 MHz |
| Base width of stereo camera ($B$) | 30 cm |
| Azimuth field of view (FOV) | ± 30° |
| Elevation field of view (FOV) | ± 15° |

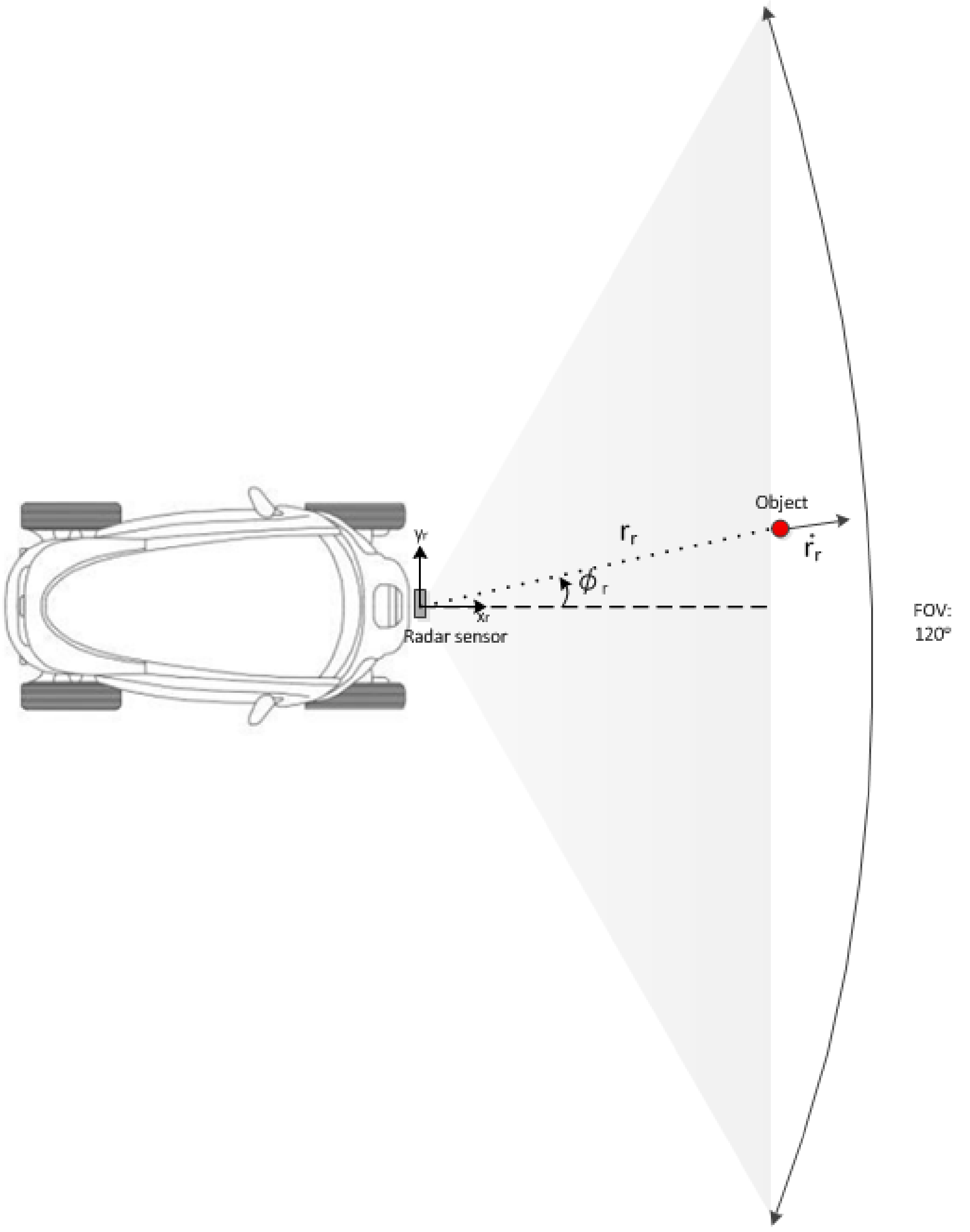
Table 3.2: Specifications of the Leopard imaging LI-AR0231-GMSL-060H camera [27]

The camera outputs are given in spherical coordinates. In Figure 3.3 an overview of what the camera sensors measure can be seen. The frame of the left camera is used as the reference frame.

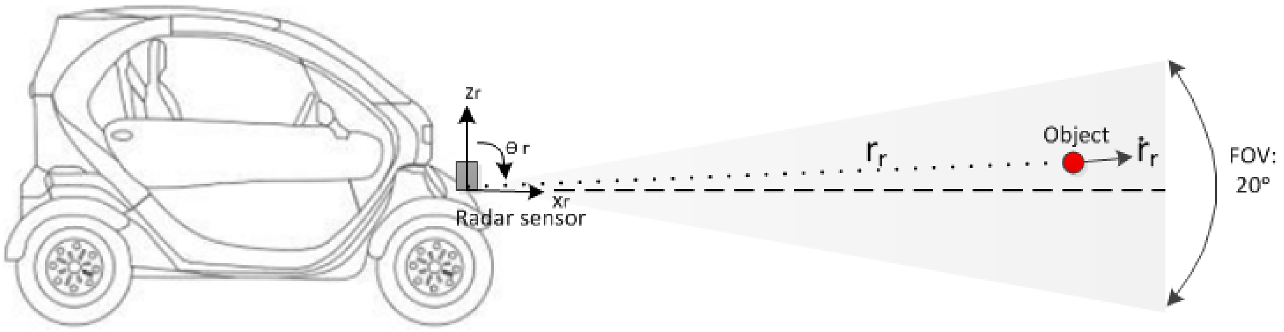
The data generated by the camera is already partially pre-processed and filtered on a Nvidia DrivePX2 [22]. Due to some of this processing the vision system is able to detect six different object classes: cars, road signs, bicycles, trucks, persons and traffic lights. After processing the following outputs can be obtained per object from the camera sensor:

- Azimuth angle ($\phi_c$): angle between the optical axis of the left camera and the object
- Elevation angle ($\theta_c$): angle between the optical axis and the object
- Radial distance ($r_c$): distance between the left camera and the object
- Object class ID; 0: Cars, 1: Road signs, 2: Bicycles, 3: Trucks, 4: Persons, 5: Traffic lights

These are the outputs of the camera that can be used for the object tracking algorithm. No information with respect to the radial velocity is available.

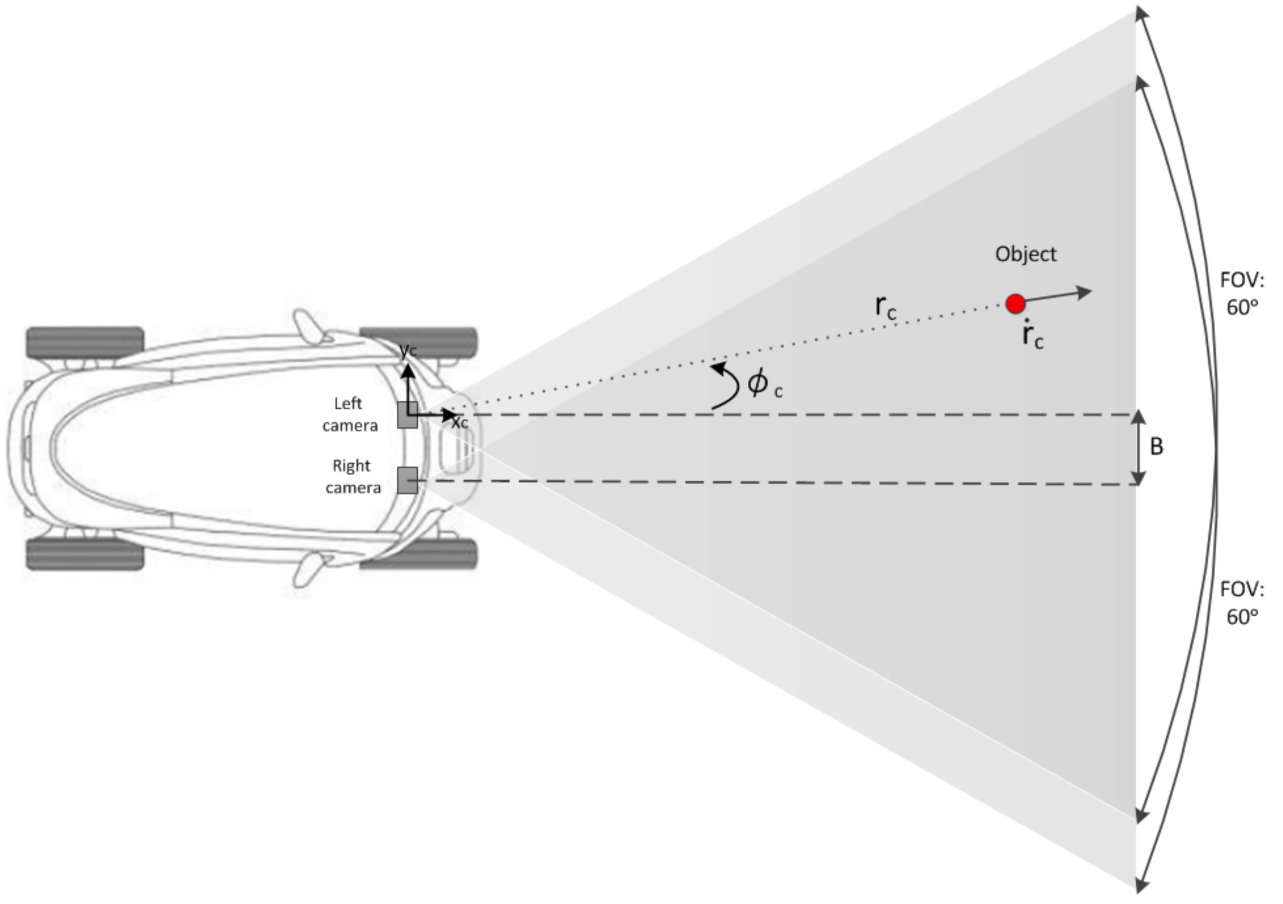


(a) Top view radar measurement

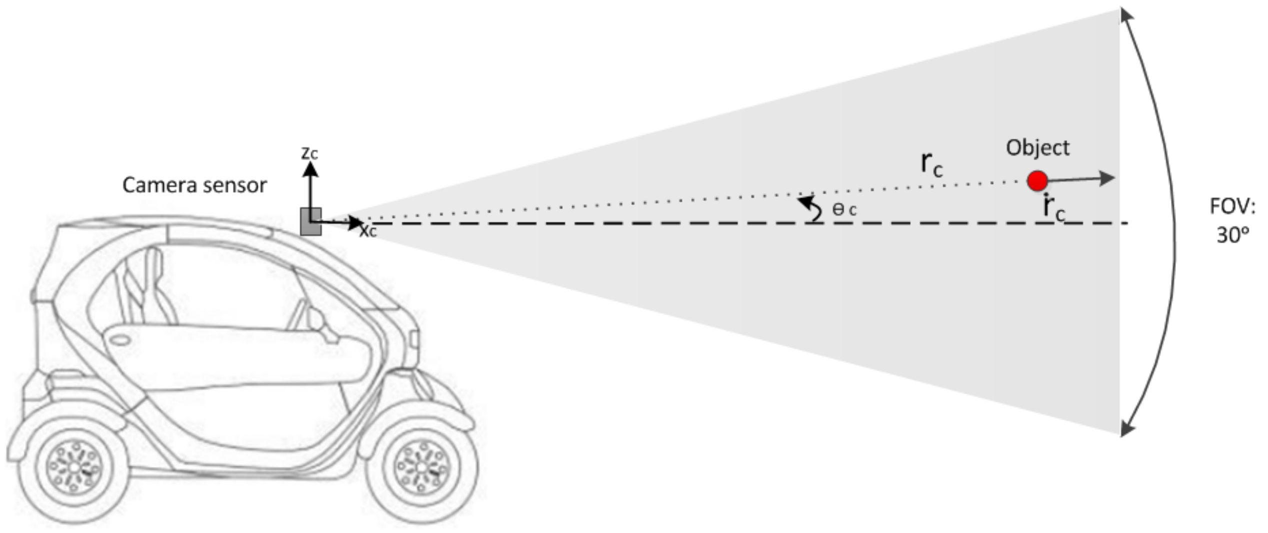


(b) Side view radar measurement

Figure 3.2: Overview of radar measurement



(a) Top view camera measurement

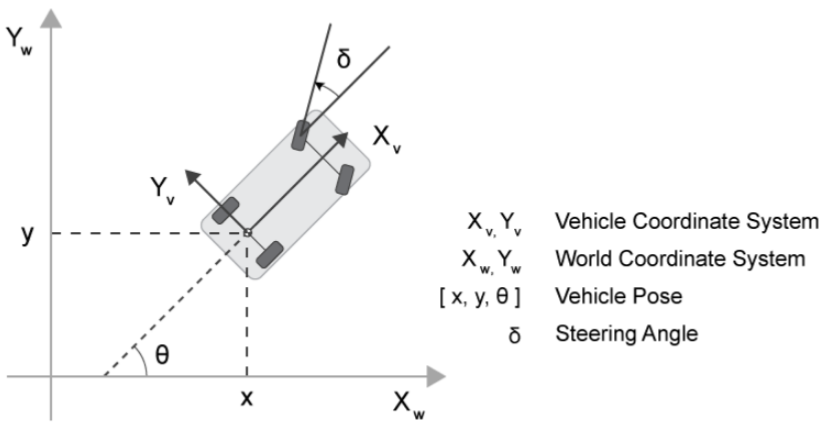


(b) Side view camera measurement

Figure 3.3: Overview of camera measurement

## 3.2 Driving scenario

Since no real measurement data can be generated, synthetic measurement data will be used instead. The radar and camera sensor generator will generate virtual measurement data based on a driving scenario. Such a driving scenario can be created with a driving scenario designer. Available actors for a driving scenario are cars, trucks, cyclists, pedestrians and barriers. To be able to simulate the behavior of the actors, the trajectory of each actor needs to be defined. The most important actor is called the ego vehicle. The ego vehicle will be the vehicle that is equipped with the desired sensors and will be used to track the other actors and obstacles present in the driving scenario. Based on the driving scenario, measurements will be generated that will be used as an input for the object tracking model. The real position and velocity of each actor are generated by the driving scenario reader and will be used as a ground truth data set. An overview of all the outputs generated by the driving scenario reader can be found in Appendix A. All the actors are moving in the world coordinate frame, which is a fixed frame given by $(x_w, y_w)$. The generated positions and velocities of each actor are obtained with respect to the vehicle coordinate system given by $(x_v, y_v)$, which is a coordinate frame placed on the ground below the midpoint of the rear axle. An overview of both coordinate systems can be seen in Figure 3.4.



$Y_w$
$X_w$
$X_v$
$Y_v$
δ
θ
y
x
$X_v, Y_v$ Vehicle Coordinate System
$X_w, Y_w$ World Coordinate System
[ x, y, θ ] Vehicle Pose
δ Steering Angle

Figure 3.4: Vehicle coordinate system and world coordinate system [34]

## 3.3 Sensors models

Based on the simulated poses and velocities of the actors, radar and camera measurements will be generated. Radar and camera detections are generate by a radar and camera sensor generator which generate measurement noise, true detections and false positives from a statistical model. The statistical model uses random numbers created by a random number generator. Depending on the field of view of the sensors, the random numbers and other parameter settings, radar and camera measurements will be generated. The measurements from the sensors will be obtained with respect to the ISO coordinate system as can be seen in Figure 3.5.

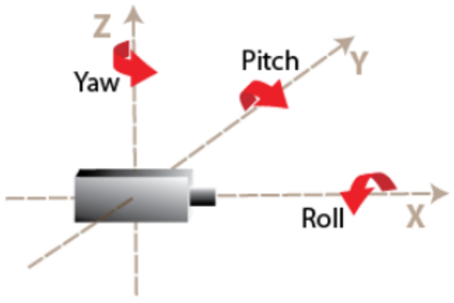


Figure 3.5: Coordinate frame of the radar and camera sensor [34]

Since it is assumed that all the actors move in the 2D-plane and the motions in $x$ and $y$-direction are assumed to be uncorrelated from the motion in the $z$-direction, measurements and motions in the $z$-direction will be ignored during the rest of the report. For the radar and camera sensor generators some of the parameter settings will be based on the known sensor characteristics of the radar and camera that are mounted on the Twizy, such as the time interval between sensor updates and the update rates. A complete overview of the radar and camera sensor settings that will be used is shown in Appendix D.

Radar measurements obtained from the radar sensor generator consist of:

- Azimuth angle ($\phi_r$) in degrees: measured from the $x$-axis towards the $y$-axis.
- Radial distance ($r_r$) in meters: measured in the direction of the detection.
- Radial velocity ($\dot{r}_r$) in meters per second: range rate is positive for a target moving away from the sensor and negative for a target moving towards the sensor.

Each radar detection will be represented by a measurement noise covariance matrix. This matrix contains the covariances in each direction. Based on the resolution settings of the radar and the position of the object, a covariance matrix will be created. It is assumed that all the measured variables are uncorrelated and therefore the output of the covariance matrix will be of the following form:

$$R_r = \begin{bmatrix} \sigma^2_{\phi r} & 0 & 0 \\ 0 & \sigma^2_{rr} & 0 \\ 0 & 0 & \sigma^2_{\dot{r}r} \end{bmatrix} \tag{3.1}$$

where $\sigma^2_{\phi r}$, $\sigma^2_{rr}$ and $\sigma^2_{\dot{r}r}$, are the variances of $\phi_r$, $r_r$ and $\dot{r}_r$ respectively.

Camera measurements from the vision generator are obtained in Cartesian coordinates $[x_c \; y_c \; \dot{x}_c \; \dot{y}_c]^T$, since the camera sensor generator is not able to output the measurements in spherical coordinates. To be able to fuse the radar and camera detections by making use of the same Kalman filter, the camera detections are transformed from Cartesian to spherical coordinates. This transformation is done by the following equations [41]:

$$\begin{aligned} \phi_c &= \tan^{-1}\left(\frac{y_c}{x_c}\right) \\ r_c &= \sqrt{x_c^2 + y_c^2} \\ \dot{r}_c &= \frac{x_c \dot{x}_c + y_c \dot{y}_c}{\sqrt{x_c^2 + y_c^2}} \end{aligned} \tag{3.2}$$

with

- $x_c$ and $y_c$, the measured position of the detection in Cartesian coordinates.
- $\dot{x}_c$ and $\dot{y}_c$, the measured velocities of the detection in Cartesian coordinates.
- Azimuth angle ($\phi_c$) in degrees: measured from the $x$-axis towards the $y$-axis.
- Radial distance ($r_c$) in meters: measured in the direction of the detection.
- Radial velocity ($\dot{r}_c$) in meters per second. The radial velocity is positive for a target moving away from the sensor and negative for a target moving towards the camera. The radial velocity is determined from the position and velocities in the $x$ and $y$-direction.

Also for the camera a measurement noise covariance matrix for each detection is given by the sensor model. Based on parameter settings of the camera, a measurement noise covariance matrix of the following form is generated by the sensor model:

$$R_{c,cart} = \begin{bmatrix} \sigma^2_{x_c} & \sigma_{x_c}\sigma_{y_c} & 0 & 0 \\ \sigma_{y_c}\sigma_{x_c} & \sigma^2_{y_c} & 0 & 0 \\ 0 & 0 & \sigma^2_{\dot{x}_c} & \sigma_{\dot{x}_c}\sigma_{\dot{y}_c} \\ 0 & 0 & \sigma_{\dot{y}_c}\sigma_{\dot{x}_c} & \sigma^2_{\dot{y}_c} \end{bmatrix}. \tag{3.3}$$

Since pixel positions and velocities obtained from the camera are transformed into camera coordinates and velocities, the measurement noise covariance matrix is not diagonal. Since this output measurement covariance matrix is given in Cartesian coordinates and the measurements are transformed from Cartesian to spherical coordinates, also the measurement noise covariance matrix needs to be transformed to spherical coordinates. As described in [18], the measurement noise covariance can be transformed from Cartesian to spherical coordinates by:

$$R_{c,sph} = JR_{c,cart}J^T \tag{3.4}$$

where the Jacobian $J$ can be calculated by taking the derivatives of the equations given in (3.2). After the transformation the measurement noise covariance matrix becomes a full matrix, which means that the camera outputs $\phi_c$, $r_c$ and $\dot{r}_c$ are correlated.

Similar to the positions on the Twizy, the radar sensor used in the object tracking model will be placed in the middle of the front bumper and the camera sensor on top of the roof. In Figure 3.6 an overview of how the radar and camera measurements are generated, is shown. An overview of all the outputs that can be obtained from the radar and camera detection generators can be found in Appendix A.

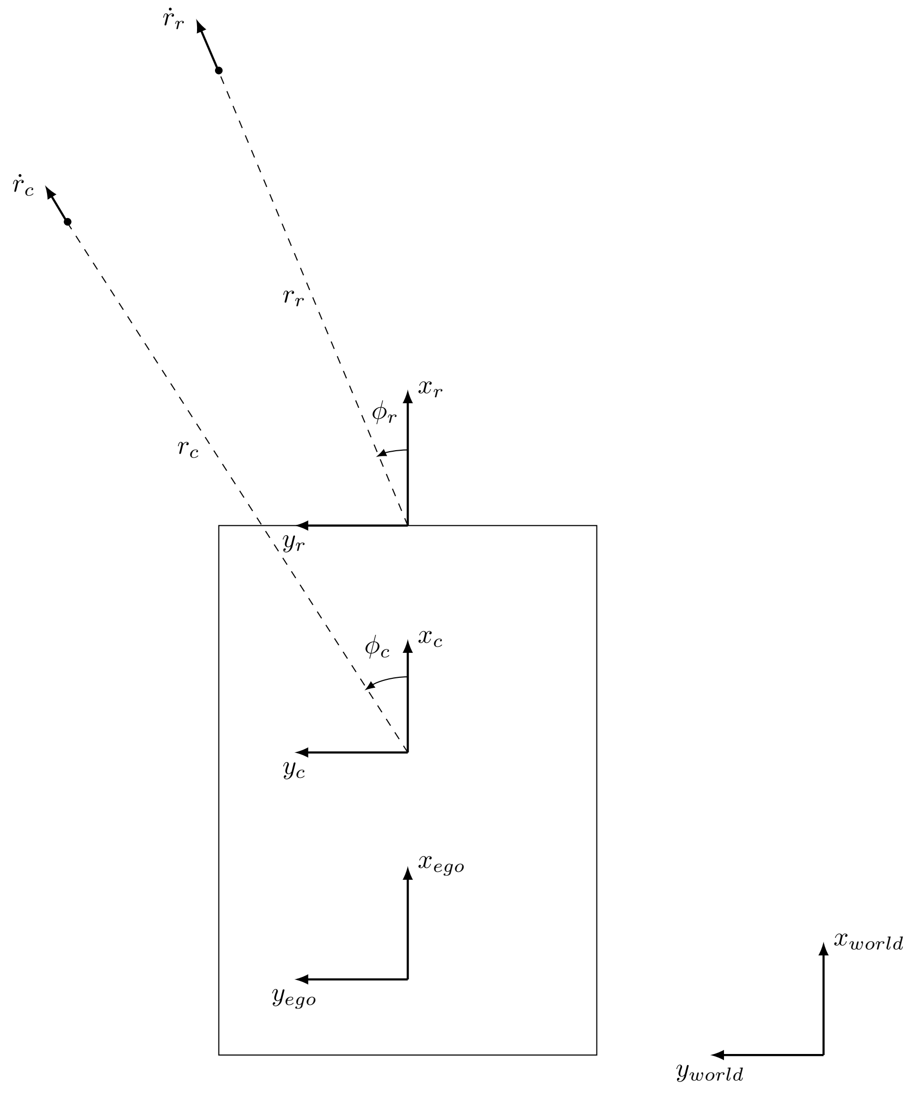


Figure 3.6: Measurement overview top view

## 3.4 Summary

In this chapter the radar and camera sensors which are mounted on the Twizy have been discussed in more detail. Furthermore, it is described how synthetic radar and camera data will be created based on a driving scenario and sensor models. An overview of all the outputs of the radar and camera sensor that will be used for the object tracking model has been given.

# Chapter 4

# Object tracking model

In Chapter 2 literature have been discussed with respect to several necessary parts of an object tracking algorithm. The object tracking algorithm will consist of several parts such as radar clustering, data association and a Kalman filter. The input for the clustering algorithm will consist of radar and camera measurements. In Figure 4.1 an overview of the tracking model can be seen.

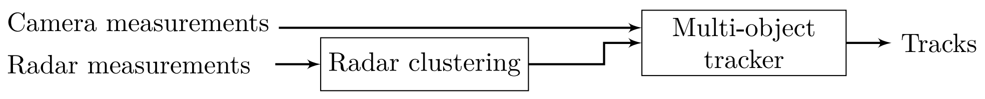


Figure 4.1: Object tracking model overview

In this chapter all the parts of the object tracking model will be discussed in more detail. First the working of the clustering algorithm will be discussed. Second, it will be discussed how the clustered radar data and the camera measurements are combined to a single input for the multi-object tracker. Finally, it will be discussed how tracks are obtained with the multi-object tracker, which consists of a Kalman filter and a data association algorithm.

## 4.1 Radar clustering

Data generated by a radar sensor can consist of multiple detections with a different range, range rate or angle, which can be originating from the same target. To be able to group measurement data that is related to the same object together, a radar clustering algorithm can be used. In Chapter 2 several clustering algorithms have been described. One of the most widely used clustering algorithms is the DBSCAN algorithm [13]. This clustering algorithm creates clusters based on position only. But clustering only based on position can give inaccurate results, for example when multiple objects are too close to each other. Then these detections together will form a cluster and information about the separate objects will be lost. Therefore, a clustering algorithm that creates clusters based on position and velocity have been discussed in [19]. Since the radar both measures the position and velocity described by $\phi_r$, $r_r$ and $\dot{r}_r$ in spherical coordinates, a position and velocity-based clustering algorithm based on the algorithm discussed in [19] will be used in the object tracking model.

To create clusters based on the position of the radar measurements, the Euclidean distance between each detection needs to be calculated. The Euclidean distance between point $i$ and point $j$ can be calculated as:

$$d_{ij} = \sqrt{(x_{r,i} - x_{r,j})^2 + (y_{r,i} - y_{r,j})^2} \tag{4.1}$$

with the $x$ and $y$-position of each radar detection in Cartesian coordinates, $x_r$ and $y_r$ given by:

$$\begin{aligned} x_r &= r_r \cos(\phi_r) \cos(\theta_r) \\ y_r &= r_r \sin(\phi_r) \cos(\theta_r). \end{aligned} \tag{4.2}$$

By computing all the Euclidean distances between $n$ detections, a Euclidean distance matrix can be created. Since distance is symmetric, only the upper triangular matrix is filled with Euclidean distances. The diagonal elements of the matrix are zero, representing distances from a detection to itself, therefore the Euclidean distance matrix $\mathbf{d}$ looks as follows:

$$\mathbf{d} = \begin{bmatrix} 0 & d_{12} & d_{13} & \dots & d_{1n} \\ 0 & 0 & d_{23} & \dots & d_{2n} \\ 0 & 0 & 0 & \dots & d_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 0 \end{bmatrix}. \tag{4.3}$$

To determine which points will form a cluster based on position, all the elements of the distance matrix $\mathbf{d}$ are compared to a distance threshold value $d_{threshold}$ as follows:

$$D_{ij} = \begin{cases} 0 & \text{if } d_{ij} > d_{threshold} \\ 1 & \text{if } d_{ij} \leq d_{threshold}. \end{cases} \tag{4.4}$$

If a distance $d_{ij}$ is below or equal to the threshold value $d_{threshold}$, then this point can be part of the first cluster and $D_{ij}$ becomes 1. When points are not within a distance $d_{threshold}$ from each other, then these points will not form a cluster and $D_{ij}$ becomes 0. Based on the threshold value a new matrix can be generated consisting of zeros and ones.

Assuming that the velocity difference between two objects is larger than the velocity difference between detections of the same object, position based clustering in combination with velocity based clustering can improve the clustering algorithm. Therefore, as discussed in [19] also the measured velocity will be used to create clusters. Since it is not possible to convert the radial velocity into linear velocity, because of missing measurement data with respect to the change of the azimuth angle over time, the radial velocity $\dot{r}_r$ will be used to calculate the velocity difference. Therefore the velocity difference $v_{r,ij}$ between all detections $i$ and detections $j$ is calculated by:

$$v_{r,ij} = \dot{r}_{r,i} - \dot{r}_{r,j} \tag{4.5}$$

Similar to the position, the velocity differences between all the $n$ detections can be stored in a velocity matrix. This velocity matrix $\mathbf{v}$ has the same structure as the distance matrix and looks as follows:

$$\mathbf{v} = \begin{bmatrix} 0 & v_{r,12} & v_{r,13} & \dots & v_{r,1n} \\ 0 & 0 & v_{r,23} & \dots & vr_{2n} \\ 0 & 0 & 0 & \dots & v_{r,3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 0 \end{bmatrix} \tag{4.6}$$

All the elements of the velocity matrix $\mathbf{v}$ are compared to a velocity threshold value $v_{r,threshold}$ as follows:

$$V_{r,ij} = \begin{cases} 0 & \text{if } v_{r,ij} > v_{r,threshold} \\ 1 & \text{if } v_{r,ij} \leq v_{r,threshold}. \end{cases} \tag{4.7}$$

Based on the threshold value a new Boolean matrix can be generated consisting of zeros and ones. To create clusters based on position and velocity a combined cluster matrix $C$ is obtained as follows:

$$C = \begin{bmatrix} 2 & 0 & 0 & \dots & 0 \\ 0 & 2 & 0 & \dots & 0 \\ 0 & 0 & 2 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 2 \end{bmatrix} + \begin{bmatrix} 0 & D_{12} & D_{13} & \dots & D_{1n} \\ 0 & 0 & D_{23} & \dots & D_{2n} \\ 0 & 0 & 0 & \dots & D_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 0 \end{bmatrix} + \begin{bmatrix} 0 & V_{r,12} & V_{r,13} & \dots & V_{r,1n} \\ 0 & 0 & V_{r,23} & \dots & Vr_{2n} \\ 0 & 0 & 0 & \dots & V_{r,3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 0 \end{bmatrix} \tag{4.8}$$

which consists of zeros, ones or twos. By looping over the rows and columns of the clustering matrix $C$ it will be determined which detections form a cluster together. A zero in the matrix means that the detection will not be part of the cluster, since both the position and velocity threshold values are not met. A 1 in the matrix means that the detection only meets one of the two conditions and therefore will not be part of the cluster, since it is necessary that the detection satisfies both threshold values. The value two in the matrix means that the detection meets both the position and velocity threshold values and will be part of the cluster. If a detection is already assigned to a cluster, this detection cannot be part of another cluster. Single detections that are not assigned to a clusters will form a separate "cluster", consisting of only one single detection.

Since each cluster consists of a combination of detections the new centroid of the cluster needs to be determined, which is the middle of the cluster. In [31] cluster centroids are calculated based on a weighted average. But since it is difficult to give the points in a cluster different weightings, all the points in a cluster will have the same weighting and the centroid of the cluster is determined by the sum of all the points within the cluster dived by the number of points in the cluster. Therefore the new core point in Cartesian coordinates can be calculated as:

$$\begin{aligned} \bar{x}_r &= \frac{\sum^{n} x_r}{n} \\ \bar{y}_r &= \frac{\sum^{n} y_r}{n} \end{aligned} \tag{4.9}$$

where $\bar{x}_r$ and $\bar{y}_r$ represent the position of the core point in Cartesian coordinates. From the positions in Cartesian coordinates also the position of each cluster in spherical coordinates can be calculated by the following equations:

$$\begin{aligned} \bar{\phi}_r &= \tan^{-1}\left(\frac{\bar{y}_r}{\bar{x}_r}\right) \\ \bar{r}_r &= \sqrt{\bar{x}_r^2 + \bar{y}_r^2 + \bar{z}_r^2} \end{aligned} \tag{4.10}$$

where $\bar{\phi}_r$ and $\bar{r}_r$ together represent the position of each cluster in spherical coordinates. Assuming that the detections that form a cluster are originating from the same object, it can be assumed that the radial velocity of these detections are almost of the same order of magnitude and in a similar direction. Therefore the radial velocity of the cluster is calculated by:

$$\bar{\dot{r}}_r = \frac{\sum^{n} \bar{\dot{r}}_r}{n} \tag{4.11}$$

where $\bar{\dot{r}}_r$ represents the radial velocity of each cluster.

Since the quality of the results obtained by the Kalman filter are depending on the accuracy of the measurement noise covariance matrix it is important to obtain an accurate representation of the measurement noise covariance of each cluster. Therefore, a combined variance will be calculated for each cluster in the developed clustering algorithm, which takes into account the spread of the measurements within the cluster. The new variances of each cluster $\bar{\sigma}^2_{\phi r}$, $\bar{\sigma}^2_{rr}$ and $\bar{\sigma}^2_{\dot{r}r}$ can be calculated by the following equations [15]:

$$\begin{aligned}
\bar{\sigma}^2_{\phi r} &= \frac{\sum^n \sigma^2_{\phi_r} + \sum^n (\phi_r - \bar{\phi}_r)^2}{n} \\
\bar{\sigma}^2_{rr} &= \frac{\sum^n \sigma^2_{rr} + \sum^n (r_r - \bar{r}_r)^2}{n} \\
\bar{\sigma}^2_{\dot{r}r} &= \frac{\sum^n \sigma^2_{\dot{r}_r} + \sum^n (\dot{r}_r - \bar{\dot{r}}_r)^2}{n}
\end{aligned} \tag{4.12}$$

By using this equation a new combined variance can be calculated as the mean of the cluster taking into account the spread of all the points within that cluster. The combined variance consists of the original variance of each detection and a part that depends on the spread of the points within the cluster with respect to the centroid of the cluster. A large spread of the detections within a cluster will result in a higher variance. On the other hand if the detections within a cluster are very close to each other, the combined variance will be almost equal to the average of the original variance. It is not necessary that the variances and means of each group are equal. A proof of (4.12) can be found in Appendix B.1. After calculating the new variances for each cluster the following measurement noise covariance matrix can be obtained:

$$R_{r,cluster} = \begin{bmatrix} \bar{\sigma}^2_{\phi r} & 0 & 0 \\ 0 & \bar{\sigma}^2_{rr} & 0 \\ 0 & 0 & \bar{\sigma}^2_{\dot{r}r} \end{bmatrix} \tag{4.13}$$

The following outputs of each cluster are created by the clustering algorithm:

- Position and velocity of each radar cluster in spherical coordinates: $\bar{\phi}_r$, $\bar{r}_r$ and $\bar{\dot{r}}_r$
- Variances of each radar cluster in spherical coordinates: $\bar{\sigma}^2_{\phi_r}$, $\bar{\sigma}^2_r$ and $\bar{\sigma}^2_{\dot{r}_r}$

These outputs will be used in the next part of the tracking algorithm which is detection concatenation. An example of how the clustering algorithm works and how the variances of the clusters change can be found in Appendix B.2.

## 4.2 Combining radar and camera data

Before data association and filtering can take place, all the radar and camera data and measurement noise covariance matrices need to be combined into one output signal. To be able to combine the clustered radar detections and camera detections, first an empty structure is created. This structure is similar to the structure of the individual sensor outputs. All the radar outputs for each time step $k$ are represented by $z_{r_{combined},k}$, which consists of the clustered radar outputs $\bar{\phi}_r$, $\bar{r}_r$ and $\bar{\dot{r}}_r$ and the measurement noise covariance matrix $R_{r,cluster}$. The camera outputs for each time $k$ are represented by $z_{c_{combined},k}$, which consists of the radar outputs $\phi_c$, $r_c$ and $\dot{r}_c$ and the measurement noise covariance matrix $R_{c,cart}$. Depending on the update rate of each sensor, the combined output of the radar clusters and camera measurements looks as follows:

$$z_{combined,k} = \begin{cases} z_{r_{combined},k} & \text{if there are only radar detections available at time step } k \\ z_{c_{combined},k} & \text{if there are only camera detections available at time step } k \\ \begin{bmatrix} z^T_{r_{combined},k} & z^T_{c_{combined},k} \end{bmatrix} & \text{if there are radar and camera detections available at time step } k \end{cases} \tag{4.14}$$

The output structure $z_{combined,k}$ consists of all the relevant information of each sensor at time step $k$ that is necessary for the Kalman filter, such as the measurement and the measurement noise covariance matrix.

## 4.3 Multi-object tracker

The multi-object tracker consists of multiple elements [36]. Figure 4.2 gives a structural overview of all the elements that are part of the multi-object tracker. In real world applications, the functions of these elements can overlap considerably. However, this figure illustrates how the multi-object tracker can be divided in different elements.

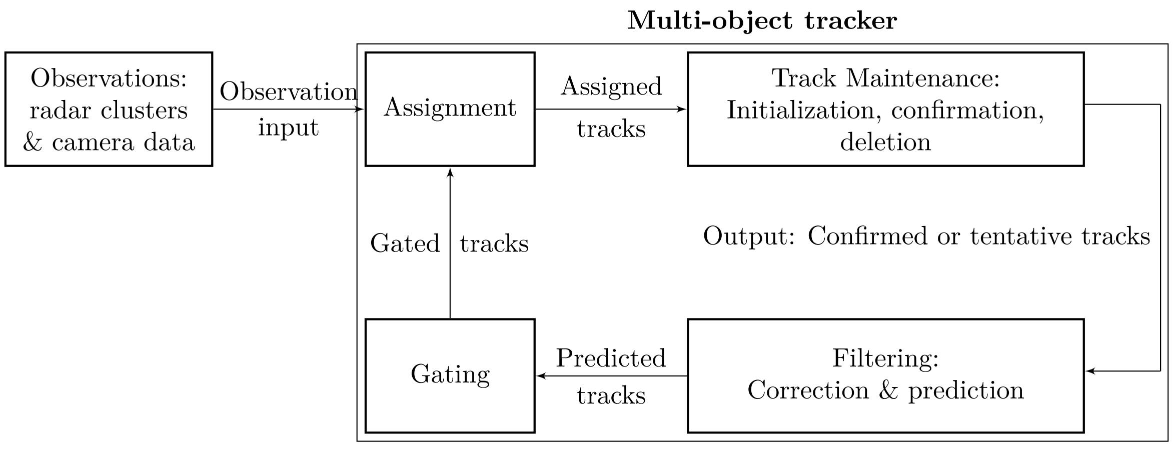

Multi-object tracker
Observations: radar clusters & camera data
Observation input
Assignment
Assigned tracks
Track Maintenance: Initialization, confirmation, deletion
Gated tracks
Output: Confirmed or tentative tracks
Gating
Predicted tracks
Filtering: Correction & prediction

Figure 4.2: Elements of the multi-object tracker

The input of the multi-object tracker consists of the combined information generated by the radar clustering algorithm and the camera. The output of the multi-object tracker block will consist of information about the state of the track (confirmed or tentative) and other information such as the state covariance and the number of obtained tracks. An overview of all the outputs of the multi-object tracker block is shown in Appendix A. Within the multi-object tracker block assignment, track maintenance, filtering and gating will take place. Tracks (confirmed or tentative) will be generated in the following way [36]:

1. A Kalman filter predicts the confirmed or tentative tracks from the previous time step to the current time step (Kalman filter).
2. The multi-object tracker will use this predicted state estimate and predicted state covariance to form a validation gate around the predicted track (gating).
3. The detections that are within the gate of a track are considered as candidates that can be assigned to a track (assignment).
4. Based on the assignment the tracker will execute track maintenance: initialization, confirmation or deletion (track maintenance).
5. The obtained confirmed and tentative tracks will be predicted to the next time step.

In this section all the elements of the multi-object tracker will be discussed in more detail.

### 4.3.1 Kalman filter

As discussed in Chapter 2 the Kalman filter is the most widely used filter for tracking algorithms. The main goal of the filtering algorithm is to predict tracks for the current time step. Also information about the predicted state covariance is obtained from the Kalman filter. This information will be used for gating and data association that are also part of the multi-object tracker. The Kalman filter is also used to correct the predicted states by using the assigned detections. Within the object tracking algorithm non-linearities are occurring due to the fact that the measurements are in spherical coordinates and the state vector is in Cartesian coordinates. Since it is possible to convert spherical coordinates to Cartesian coordinates and vice versa and the Jacobian of this state transition can be obtained, an extended Kalman filter will be used. As given in Figure 2.1 the Kalman filter consists of a prediction and update stage and an initialization part. In this section the discrete extended Kalman filter that will be used for tracking will be explained in more detail.

**Motion model**
There are a lot of different motion models available [50]. But the most common forms of target motion models used for object tracking are the constant velocity model (CV) and constant acceleration (CA) model [44]. As long as no maneuver is performed by the object, a simple filter such as the constant velocity model can be used for tracking. If the objects are following a maneuvering trajectory, a constant acceleration model can be used [47]. Since it is the goal to finally use the designed object tracking model for a cooperative driving project with the Twizys, for now a constant velocity model will be used. Also the designed test scenarios that will be used to test the object tracking model will not contain any difficult manoeuvres or abrupt changes in the situation. By using a constant velocity model it is assumed that the velocity is constant over a time interval $T$, where $T$ is chosen to be 0.01 seconds. As long as this time interval is small enough and the car is not accelerating too hard, it can be assumed that during this time interval the velocity is not varying that much and a nearly constant velocity can be assumed. For the constant velocity model it is assumed that the motion in the $x$, $y$ and $z$ direction are uncorrelated. Since a vehicle typically moves in the $x$-$y$-plane, the $z$-direction will be ignored and a 2D Kalman filter will be used.

The motion of the vehicle can be described with a linear state space representation described as

$$\hat{\mathbf{x}}_{k-1} = \mathbf{F}_k \hat{\mathbf{x}}_k + \mathbf{G}\mathbf{w}_k \tag{4.15}$$

where the state $\hat{\mathbf{x}}_k$ is described by:

$$\hat{\mathbf{x}}_k = \begin{bmatrix} x & v_x & y & v_y \end{bmatrix}^T \tag{4.16}$$

the state transition matrix $\mathbf{F}_k$ that describes how the motion of the object evolves over time is given by [33]:

$$\mathbf{F}_k = \left[\begin{array}{cc|cc} 1 & T & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 1 & T \\ 0 & 0 & 0 & 1 \end{array}\right] \tag{4.17}$$

and the process noise prediction matrix $\mathbf{G}$ for a constant velocity model is given by [50]:

$$\mathbf{G} = \begin{bmatrix} \frac{1}{2}T^2 & 0 \\ T & 0 \\ 0 & \frac{1}{2}T^2 \\ 0 & T \end{bmatrix}. \tag{4.18}$$

The process noise $\mathbf{w}_k$ represents the parts of the process that are not taken into account in the model. For the constant velocity this will be the change in the velocity in each direction, since constant velocity is assumed. The process noise is assumed to have zero mean Gaussian noise with the variances of $w_x$ and $w_y$ given by:

$$\mathbf{q} = \begin{bmatrix} q_x & 0 \\ 0 & q_y \end{bmatrix} \tag{4.19}$$

where $q_x$ and $q_y$ represent the variances of the possible changes in the velocities in the $x$ and $y$-direction. This matrix is diagonal, since it is assumed that the motion in the $x$ and $y$-direction are uncorrelated. The process noise covariance matrix $\mathbf{Q}$ can now be obtained by [50]:

$$\mathbf{Q} = \mathbf{G}\mathbf{q}\mathbf{G}^T. \tag{4.20}$$

**Prediction and update stage**
During the prediction stage of the Kalman filter both the state and state covariance are predicted from the time step $k-1$ to the current time step $k$. Since it is assumed that the process noise is white and zero-mean, the predicted state $\hat{\mathbf{x}}_{k|k-1}$ can be calculated by:

$$\hat{\mathbf{x}}_{k|k-1} = \mathbf{F}_k \hat{\mathbf{x}}_{k-1|k-1}. \tag{4.21}$$

Besides predicting the state of an object the corresponding state covariance matrix also needs to be predicted. The state covariance matrix for the discrete extended Kalman filter is predicted by:

$$\mathbf{P}_{k|k-1} = \mathbf{F}_k \mathbf{P}_{k-1|k-1} \mathbf{F}_k^T + \mathbf{Q}. \tag{4.22}$$

The predicted state covariance matrix will be obtained based on the defined motion model, the state covariance matrix generated in the update stage of the previous time step and the the process noise matrix $\mathbf{Q}$ that can be calculated using (4.20).

Besides the predicting stage there is also an update stage, that will make use of the data that is generated by the radar and camera sensor. The state estimation will be compared with the measurement data by calculating the measurement residual by:

$$\tilde{\mathbf{y}}_k = \mathbf{z}_k - \mathbf{h}(\hat{\mathbf{x}}_{k|k-1}) \tag{4.23}$$

where the measurement state $\mathbf{z}_k$ is defined as

$$\mathbf{z}_k = \begin{bmatrix} \phi & r & \dot{r} \end{bmatrix}^T \tag{4.24}$$

and consists of the radar and/or camera measurement. Since each column of $\mathbf{z}_k$ consists of a detection, each column will be passed to the Kalman filter. Since the measurement state is given in spherical coordinates and the model state is in Cartesian coordinates a non-linear model will be used. The non-linear measurement function $\mathbf{h}(\hat{\mathbf{x}}_k)$ is defined as [23]:

$$\mathbf{h}(\hat{\mathbf{x}}_k) = \begin{bmatrix} \tan^{-1}\left(\frac{y - y^i_{offset}}{x - x^i_{offset}}\right) \\ \sqrt{(x - x^i_{offset})^2 + (y - y^i_{offset})^2} \\ \frac{(x - x^i_{offset})\dot{x} + (y - y^i_{offset})\dot{y}}{\sqrt{(x - x^i_{offset})^2 + (y - y^i_{offset})^2}} \end{bmatrix}. \tag{4.25}$$

To be able to calculate the measurement residual, first the observation model needs to be translated from Cartesian coordinates in the ego car frame, to spherical coordinates in the sensor frames. This is done by applying a state transformation and correction of the states with the offsets $x^i_{offset}$ and $y^i_{offset}$ of the corresponding radar or camera sensor $i$.

Besides the innovation measurement also the covariance of the states needs to be updated. The innovation covariance matrix will be calculated by the predicted state covariance matrix and will be updated with the current measurement as follows:

$$\mathbf{S}_k = \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^T + \mathbf{R}_k. \tag{4.26}$$

where $\mathbf{H}_k$ is the Jacobian of the non-linear measurement model which can be calculated by:

$$\mathbf{H}_k = \frac{\partial \mathbf{h}(\mathbf{x})}{\partial \mathbf{x}}\bigg|_{\hat{\mathbf{x}}_{k|k-1}} = \left.\begin{bmatrix} \frac{\partial \phi}{\partial x} & \frac{\partial \phi}{\partial v_x} & \frac{\partial \phi}{\partial y} & \frac{\partial \phi}{\partial v_y} \\ \frac{\partial r}{\partial x} & \frac{\partial r}{\partial v_x} & \frac{\partial r}{\partial y} & \frac{\partial r}{\partial v_y} \\ \frac{\partial \dot{r}}{\partial x} & \frac{\partial \dot{r}}{\partial v_x} & \frac{\partial \dot{r}}{\partial y} & \frac{\partial \dot{r}}{\partial v_y} \end{bmatrix}\right|_{\hat{\mathbf{x}}_{k|k-1}} \tag{4.27}$$

and the measurement noise covariance matrix $\mathbf{R_k}$ is given by (3.4) or (4.13) depending from which sensor the measurement is coming from.

The next step in the updating part of the Kalman filter is calculating the Kalman gain. This Kalman gain can be calculated from the predicted state covariance, the innovated state covariance and the linearized measurement model by the following equation:

$$\mathbf{K}_k = \mathbf{P}_{k|k-1} \mathbf{H}_k^T \mathbf{S}_k^{-1}. \tag{4.28}$$

Depending on the how accurate the measurements are and how large the covariance is, a Kalman gain will be obtained. This Kalman gain determines how much the state and state covariance are updated by the measurement or predicted state. Accurate measurements and low covariances will result in a high Kalman gain. In this situation the update of the state will depend more on the measurement compared to the predicted state. The last step of the Kalman filter is to update the state and covariance based on the Kalman gain. The state and covariance are updated as follows:

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \tilde{\mathbf{y}}_k \tag{4.29}$$

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_{k|k-1}. \tag{4.30}$$

As long as measurement data is generated by a sensor, the Kalman filter will keep predicting and updating the state and covariance. But before the Kalman filter can start predicting and updating, first the states and covariances need to be initialized. The initialization process of the state and covariance is described in Appendix C. By using a Kalman filter, state estimations are obtained for each detection. To be able to generate tracks of each object, each detection needs to be linked to a track. This association will be done by gating and data association algorithms.

### 4.3.2 Gating

The first step before data association can take place is gating. Gating is applied to determine which detections are unlikely to be assigned to a certain track. Gating is not a necessary part of the multi-object tracker, but when gating is applied the computational load can be reduced [26]. The gate will form a region, and if a detection is outside this region, this detection will be ignored for the assignment part. As discussed in [26] different gating shapes are available, such as a rectangular gate, a spherical gate and an ellipsoidal gate. In [54] it is proven that an ellipsoidal gate is optimal in the sense of minimal volume and it has the highest probability that a detection falls within the correct gate. Therefore an ellipsoidal gate will be used for the gating algorithm. Such an ellipsoidal validation gate is formed around the center of the predicted measurement by the residual covariance matrix $S$. From the eigenvectors and eigenvalues of the covariance matrix an ellipse can be obtained, see Appendix B.2. To determine if a detection is part of a gate, the Mahalanobis distance needs to be determined. This distance will be the distance between the predicted detection and the ellipsoidal gate obtained from the residual covariance matrix. If this distance is less than or equal to a certain threshold value, then this detection satisfies the gate. The Mahalanobis distance $d^2_{M,ij}$ between a detection $j$ and a track $i$ can be calculated as follows:

$$d^2_{M,ij} = \tilde{\mathbf{y}}^T_{ij}\mathbf{S}^{-1}_i\tilde{\mathbf{y}}_{ij} \tag{4.31}$$

where $\tilde{\mathbf{y}}_{ij}$ is the measurement residual and $\mathbf{S}_i$ is the residual covariance matrix that are both calculated by the Kalman filter. This Mahalanobis distance will be compared to a gating threshold value $G_{threshold}$ as follows:

$$\tilde{\mathbf{y}}^T_{ij}\mathbf{S}^{-1}_i\tilde{\mathbf{y}}_{ij} \leq G_{threshold} \tag{4.32}$$

If the gate size is increased, the probability that a detection falls within the gate is higher. But when the gate size is too large, detections from other objects fall within the gate. This results in a large computational load and the tracking performance can decrease because of a large probability that the detection is assigned to the incorrect track. On the other hand, if the gate size becomes smaller, the number of detections within the gate decreases which results in a reduced computational load. But if the gate is too small, tracking performance can become poor since it can happen that detections are not within the gate and no assignment can take place.

### 4.3.3 Assignment of detections to tracks

After gates are formed data association can take place. Data association or also called assignment is an important step in object tracking. It is essential to assign detections to tracks in a correct way, so these detections can be used to update or create a new track. In Chapter 2 several data association algorithms are explained, such as the (G)NN, (J)PDA and MHT method. Each method has their own advantages and disadvantages. Due to its simplicity and the reduced computational load the GNN method will be used as an assignment method in the tracking algorithm. Another advantage of the GNN method over the JPDA method is the ability to initialize tracks, so no separate algorithm is needed.

**GNN method**
The GNN method takes the output of the gating algorithm and assigns the global nearest observations to existing tracks or creates new track hypotheses for unassigned detections. As mentioned in Chapter 2 the GNN approach works well if the detections are sufficiently far apart from each other. So if a single detection is gated to a single track, the assignment can be immediately made. However when detections are too close to each other, it can happen that there is more than one detection within a track's gate or a detection is in the gates of more than one track. In such a situation a conflict occurs. Such a conflict situation can be seen in Figure 4.3.

In Figure 4.3 it can be seen that there are multiple detections within the gates of the tracks T1, T2 and T3, and that detections D3 and D4 are within the gates of multiple tracks. Since gating cannot solve this problem, the data association will solve this problem using probabilistic methods [25]. The GNN method will assign detections to tracks minimizing some distance criterion. To solve the problem assignment a cost matrix $C_{ij}$ needs to be generated. This cost matrix will look as follows:

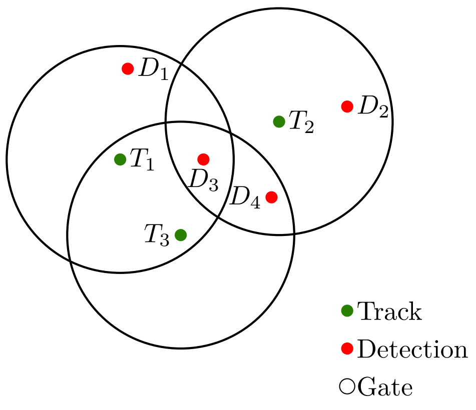


Figure 4.3: GNN with association conflict

$$C_{ij} = \begin{bmatrix} c_{11} & c_{12} & c_{13} & \dots & c_{1m} \\ c_{21} & c_{22} & c_{23} & \dots & c_{2m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & c_{n3} & \dots & c_{nm} \end{bmatrix} \tag{4.33}$$

where each element of the cost matrix can be calculated as:

$$c_{ij} = d_{M,ij} + \ln(|\mathbf{S}_i|) \tag{4.34}$$

where the Mahalanobis distance $d_{M,ij}$ can be calculated by (4.32). The desired solution of the assignment problem is the one that minimizes the summed cost [25]. If a detection is not allowed to be assigned to a certain track, then the element of the cost matrix becomes infinite which results in a large cost and therefore will be ignored. Given the cost matrix element $c_{ij}$ find and assignment $Z = \{z_{ij}\}$ that minimizes [35]

$$J = \sum_{i=0}^{n} \sum_{j=0}^{m} c_{ij} z_{ij} \tag{4.35}$$

with the following two constraints:

$$\begin{aligned} \sum_{i=0}^{m} z_{ij} &= 1, \forall j \\ \sum_{j=0}^{n} z_{ij} &= 1, \forall i \end{aligned} \tag{4.36}$$

where the first constraint means that each detection cannot be assigned to more than one track and the second constraint means that each track cannot be assigned to more than one detection. Finally, the assignment problem assigns one detection to each track. If there are detections that are left unassigned, then the tracker will use this detection to initialize a new track.

### 4.3.4 Track maintenance

To update, create or delete a track, track maintenance is necessary. The tracker consists of initialized and confirmed tracks. Based on certain conditions, an initialized track can become a confirmed track. The three following situations can occur:

- An initialized track becomes a confirmed track. If a detection cannot be assigned to an already existing track, an initialized track is created. Such a track can become a confirmed track if an object has been detected $M$ times in $N$ updates, with $M \leq N$.
- If a track is coasted $P_{coasted}$ times, then this track will be deleted.
- A track is updated if the assignment algorithm keeps assigning new detections to that track.

The parameters $M$, $N$ and $P_{coasted}$ need to be chosen based on the update rate of the sensors and the tracker algorithm, to update, create or delete tracks correctly. The confirmed tracks will be the output of the object tracking algorithm and will be compared with some ground truth information to draw conclusions about the generated tracks.

## 4.4 Summary

In this chapter the object tracking model has been discussed which consists of several elements. First an extended version of the DBSCAN algorithm has been discussed. With the developed clustering algorithm, clusters are created based on position and velocity. Besides the new core points of the clusters, also the measurement noise covariance matrix of each cluster has been calculated by an average of the variance of each individual point within the cluster. Furthermore, all the important components that are part of the multi-object tracker block have been discussed such as gating, assignment, track maintenance and filtering. For the filtering part an extended Kalman filter has been used with a constant velocity motion model. The estimated states have to be associated to existing tracks, which is done by gating, assignment and track maintenance algorithms. Finally, the confirmed tracks generated by the object tracking algorithm will be compared with ground truth information obtained from designed test scenarios. The test scenarios and corresponding results will be discussed in the following chapters.

# Chapter 5

# Test scenarios and parameter settings of the object tracking algorithm

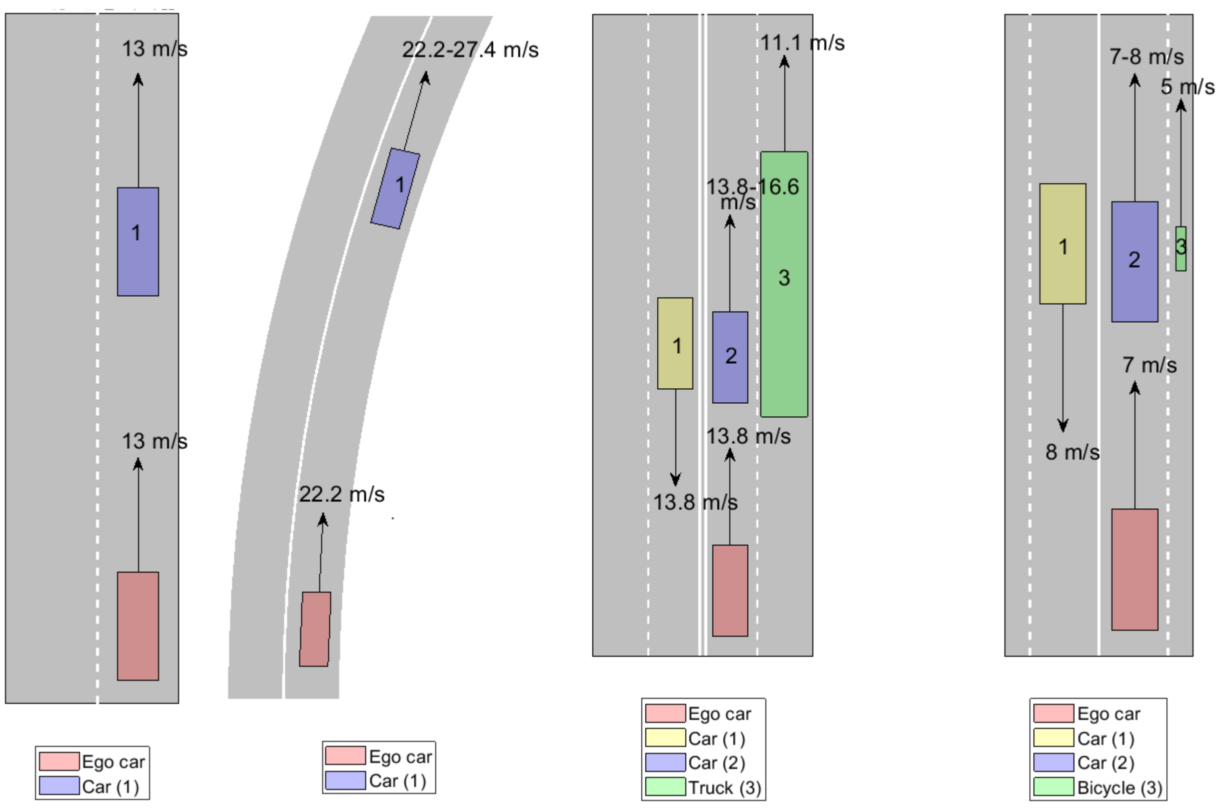
In this chapter several designed test scenarios and the settings and tuning of the parameters of the object tracking algorithm will be discussed. First the four different test scenarios will be discussed and described in more detail. Those test scenarios will be used to test the performance of the object tracking algorithm under different circumstances. Finally, the settings and tuning of the parameters of the object tracking algorithm will be discussed.

## 5.1 Test scenarios

To be able to test the object tracking algorithm, four different test scenarios are designed. The four different test scenarios are designed to test the object tracking algorithm for common driving scenarios on urban and provincial roads with different actors. Those different test scenarios will be used as an input for the object tracking algorithm. The sample time of the scenarios is 0.01 seconds, which is equal to the sample time $T$ of the object tracking algorithm. Based on those scenarios, the radar and camera sensor generators will generate measurements. The positions and velocities of all the actors in the test scenarios will be used as a ground truth data set to be able to determine the performance of the object tracking algorithm. An overview of the four designed test scenarios can be seen in Figure 5.1.

**Scenario 1**
Scenario 1 represents the most simple test scenario. In scenario 1 there is only one car in front of the ego car. Both cars are driving in a straight line with a constant velocity of 13 m/s. This test scenario has been chosen to test how the tracker performs in an easy situation, with constant velocity. In this scenario clustering is necessary to cluster multiple detections on the vehicle and to distinguish the detections on the vehicle from the false detections. An overview of test scenario 1 is given in Figure 5.1a.

**Scenario 2**
Scenario 2 represents a curved provincial road. In this scenario there is only one car in front of the ego car. Both cars are following a part of a circle with a radius of 300 m. The car in front of the ego vehicle is accelerating with 0.25 m/s$^2$ from 22.2 m/s up to 24.2 m/s. This scenario will be used to determine if the object tracking algorithm is able to track the accelerating vehicle, while it is driving on a curved road. Similar to scenario 1, clustering is relevant to generate a single cluster from multiple detections on the vehicle. An overview of test scenario 2 is given in Figure 5.1b.



(a) Scenario 1 (b) Scenario 2 (c) Scenario 3 (d) Scenario 4

Figure 5.1: Illustration of the test scenarios

**Scenario 3**
Scenario 3 represents an urban road. This scenario is designed to test if the object tracking algorithm is able to track multiple vehicles. There will be a moment in this scenario where all the three vehicles are almost on a line. At that moment it is important that the clustering algorithm is working properly and is able to create the correct clusters for each actor. This scenario consists of three vehicles where the first vehicle represents an oncoming car, which has a constant velocity of 13.8 m/s. The second car is driving in a straight line on the left lane in front of the ego car and is accelerating with 0.28 m/s$^2$ from 13.8 m/s up to 16.6 m/s. The last actor present in the scenario is a truck which is driving at the right lane. This truck is driving with a constant velocity of 11.1 m/s. Due to the difference in velocity, both the ego car and car 2 will pass the truck. An overview of test scenario 3 is given in Figure 5.1c.

**Scenario 4**
Similar to scenario 3 also scenario 4 represents an urban road. This scenario is designed to test if the object tracking algorithm is able to track multiple vehicles and a bicycle that is riding close to a car. Since in this scenario actors are also driving close to each other, clustering will be important. Scenario 4 consists of two cars and a bicycle, where car 1 is an oncoming car, with a constant velocity of 8 m/s. The second car is driving in front of the ego car and is acceleration with 0.1 m/s$^2$ from 7 m/s up to 8 m/s. The third actor is a bicycle that is driving very close to car 2 with a constant velocity of 5 m/s. Both the ego car and car 2 will pass the bicycle over time, while car 1 is approaching. There will be a moment where all the three actors are almost in a straight line. An overview of test scenario 4 is given in Figure 5.1d.

## 5.2 Parameter settings of the object tracking model

To be able to test and improve the performance of the object tracking algorithm it is necessary to select the correct parameters for the radar clustering and track maintenance algorithm. Furthermore, the process noise covariance matrix of the Kalman filter needs to be tuned.

Radar clusters are created based on two parameters, a distance threshold value $d_{threshold}$ and a velocity threshold value $v_{threshold}$. To determine the value for $v_{threshold}$, the relative radial velocities generated by the radar generator for all the four scenarios have been studied. In Figure 5.2 all the generated measurements are plotted. As can be seen there are multiple detections per time step which need to be clustered. It can be seen that for scenario 3 and 4 the spread in the generated velocities is larger compared to scenarios 1 and 2. A zoomed in plot of the part with the largest spread of scenario 3 and 4 is shown in Figure 5.3. The value for $v_{threshold}$ will be based on the largest delta between two velocity measurements at the same time step, which is equal to 11.5 m/s.

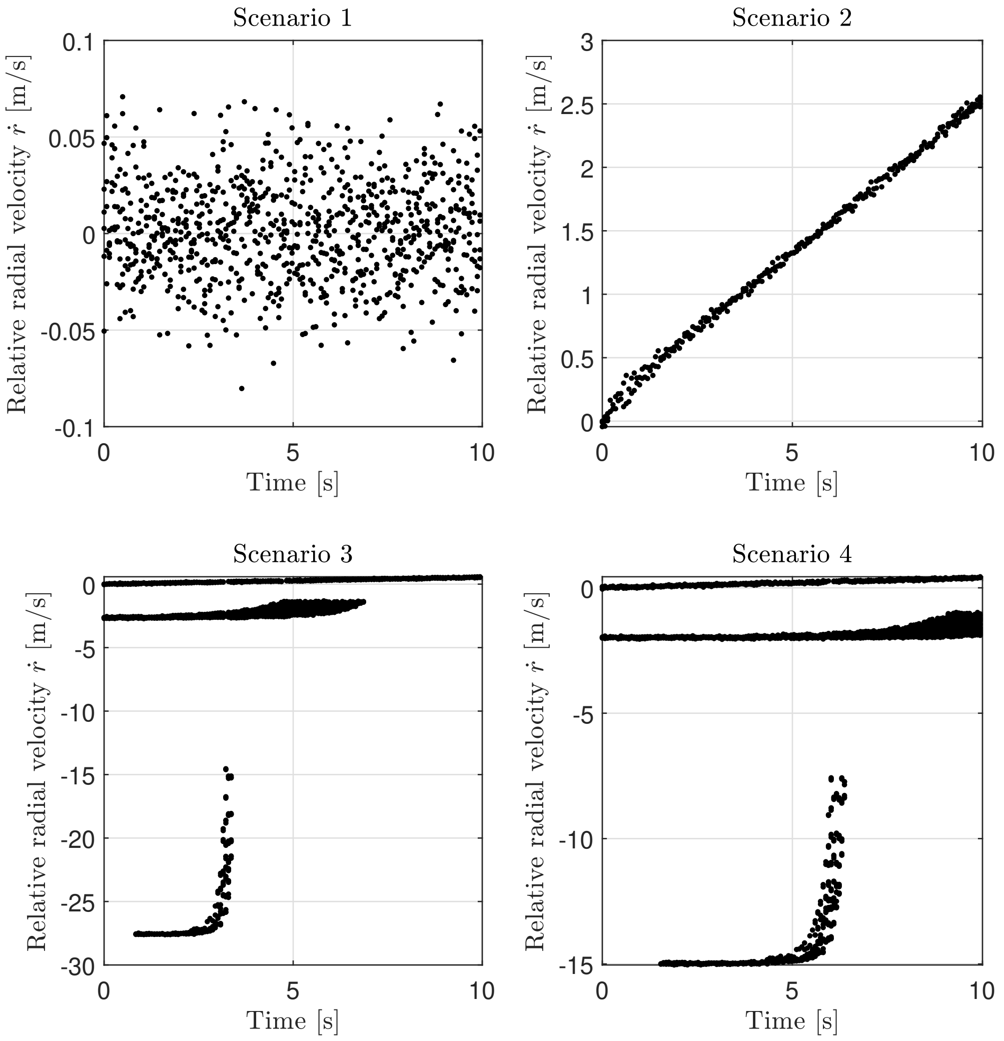


Figure 5.2: Generated relative radial velocity radar measurements for all the scenarios

Since scenario 3 and 4 consist of multiple vehicles, clustering is important for these two scenarios. Therefore a value for the threshold $d_{threshold}$ is obtained by looking to a crucial time step in scenario 3 and 4, where the three actors are almost driving at the same $x$-position. Since very often the detections cover the whole rear or front of a vehicle (especially when a vehicle is driving in front of the ego car), $d_{threshold}$ is set to 2.5 m, which is equal to the width of a truck. In this way the cluster algorithm is able to obtain one cluster if all the detections

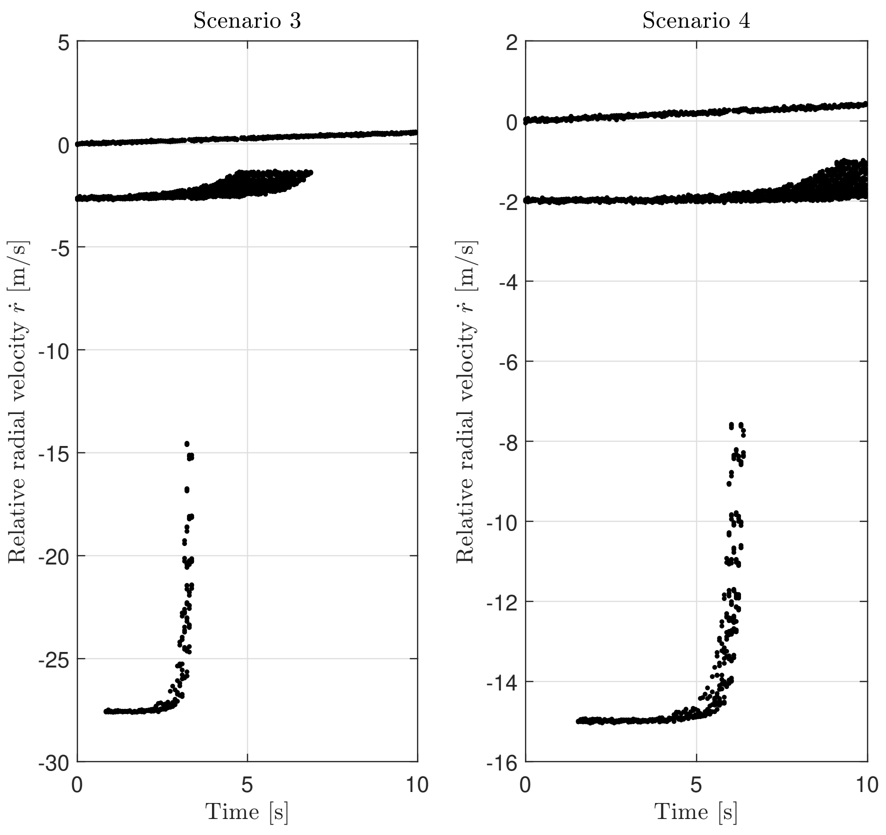


Figure 5.3: Zoomed in plot of the generated relative radial velocity radar measurements for scenario 3 and 4

are at the front or rear of a vehicle and even if there are some measurements at the side of the vehicle. To determine if the clustering algorithm is able to obtain the correct clusters with the created parameter settings, clustering is applied to a moment in time where clustering is crucial. The obtained results for scenario 3 and 4 can be seen in Figure 5.4 and 5.5.

As can be seen from Figure 5.4 and 5.5 the correct clusters are created in both scenarios with $d_{threshold} = 2.5$ m and $v_{threshold} = 11.5$ m/s. By setting the distance threshold value equal to 2.5 m, the cluster algorithm is able to create clusters that consist of some measurements at the front or rear of a vehicle including a part of the side of the vehicle.

As discussed earlier there are several parameters necessary for gating and track performance. The gating threshold value $G_{threshold}$ will determine if a detection will be assigned to a track or not. By increasing the threshold value more detections are allowed to be in the gate. With a bird's-eye scope it is possible to plot the results of the tracking algorithm simulation in real time. The value for $G_{threshold}$ is determined by looking at the real-time behaviour of the tracker with the bird's-eye scope. By setting $G_{threshold}$ equal to 50, all the relevant detections are assigned to a track and the noisy measurements far away from a vehicle are ignored.
For track performance the parameters are $M$, $N$ and $P_{coasted}$ respectively. The values of these parameters are depending on the update rate of the camera and radar sensor which are set equal to 0.1 second and 0.07 second and the update rate of the tracker $T$ which is equal to 0.01 seconds. For parameter $N$ the number of times the tracker needs to update before it confirms a track need to be considered. By setting $N$ to 10 the tracker is allowed to confirm a track each 0.1 seconds, if at least one radar or camera detection is received during that time interval. Parameter $M$ takes into account the probability of object detection for the sensors and is set to 2. When increasing $M$ more false detections will be assigned to tracks, which is undesired. By decreasing $N$ the tracker will confirm a track very fast, which results in a lot of tracks. By increasing $N$ it takes more time before a track is confirmed, so it can happen that for a while no track is created and information gets lost. Parameter $P_{coasted}$ determines how many times a track is allowed to be coasted before it is deleted. By setting $P_{coasted}$ to 21, the tracker is allowed to coast a track for 0.21 seconds if no detection is received anymore. This means

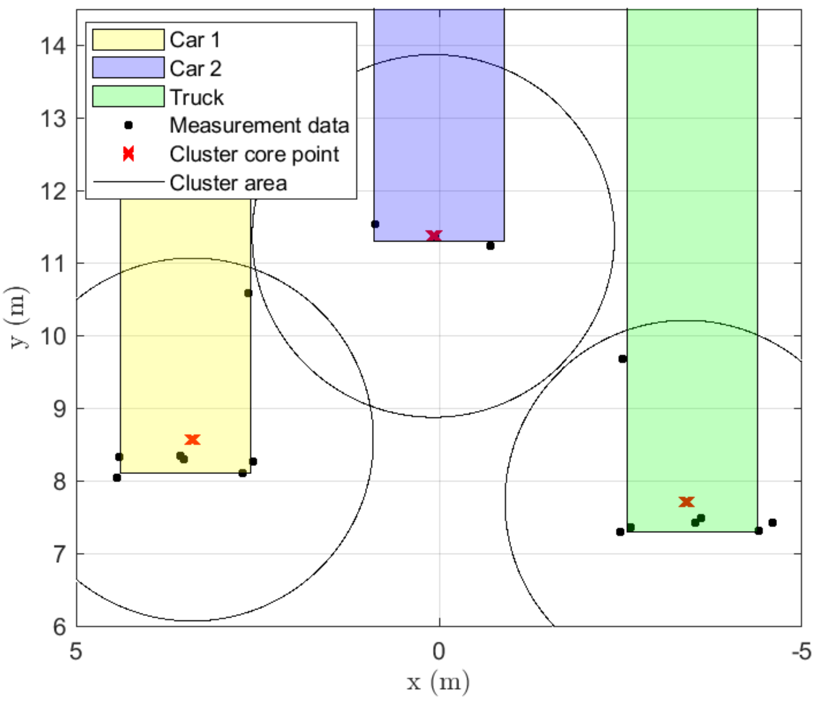


Figure 5.4: Obtained clusters for a crucial time step for scenario 3 with the selected parameters

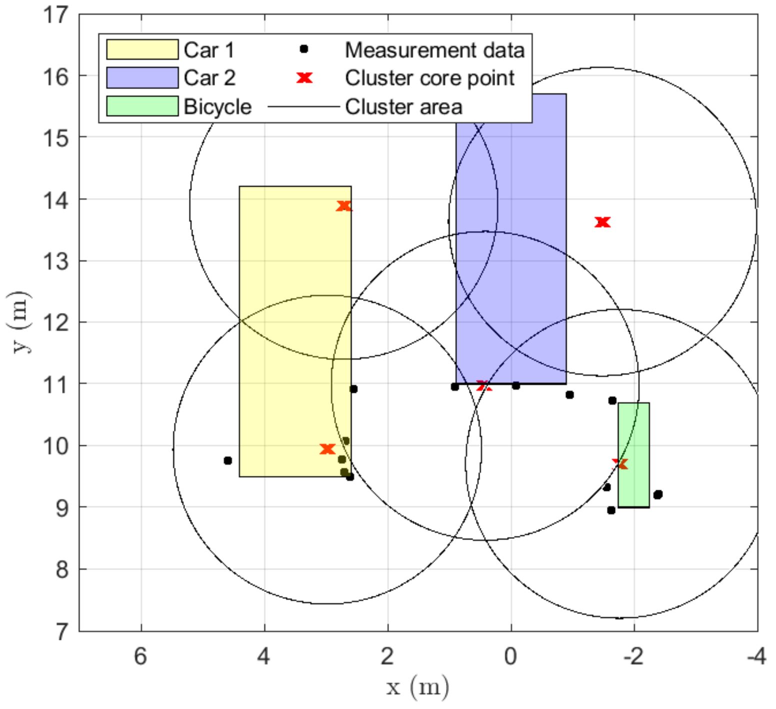


Figure 5.5: Obtained clusters for a crucial time step for scenario 4 with the selected parameters

that if two times in a row no radar or camera detections are received, the tracker removes the track. To prevent the tracker from creating a single track consisting of multiple actors or neglecting measurements located on at the side of the vehicle, the tracker is allowed to create more than one track for each actor. Therefore is the maximum number of tracks set to 6. An overview of all the parameters that are part of the multi-object tracker and their values are shown in Table 5.1.

| | |
|---|---|
| Threshold for assigning detections to tracks $G_{threshold}$ | 50 |
| $M$ and $N$ for the $M$-out-of-$N$ confirmation | [2,10] |
| Number of times a confirmed track is coasted $P_{coasted}$ | 21 |
| Maximum number of tracks | 6 |
| Maximum number of sensors | 2 |

Table 5.1: Multi-object tracker parameters

The last parameter that needs to determined is the process noise covariance matrix $\mathbf{Q}$ of the Kalman filter. The process noise covariance matrix can be calculated by (4.20) by tuning the variances $q_x$ and $q_y$ that are part of matrix $q$. To simplify the tuning process it is assumed that $q_x$ and $q_y$ are equal. The optimal values for $q_x$ and $q_y$ are determined by calculating the root mean square error (RMSE) between the ground truth and the obtained track for scenario 1. The results for different variances are shown in Table 5.2.

| $q_x = q_y$ [m/s$^2$] | RMSE $x$ [m] | RMSE $y$ [m] | RMSE $v_x$ [m/s] | RMSE $v_y$ [m/s] | Total RSME |
|---|---|---|---|---|---|
| 0.01 | 0.0336 | 0.1209 | 0.0796 | 1.2851 | 1.5192 |
| 0.025 | 0.0116 | 0.0210 | 0.0051 | 0.0289 | 0.0666 |
| 0.05 | 0.0162 | 0.0109 | 0.0088 | 0.0267 | 0.0626 |
| 0.075 | 0.0198 | 0.0329 | 0.0094 | 0.0420 | 0.1041 |
| 0.1 | 0.0130 | 0.0338 | 0.0070 | 0.0531 | 0.1069 |
| 0.25 | 0.0175 | 0.0359 | 0.0102 | 0.0817 | 0.1453 |
| 0.5 | 0.0177 | 0.0228 | 0.0085 | 0.0737 | 0.1227 |
| 1 | 0.0166 | 0.0246 | 0.0104 | 0.0944 | 0.1460 |

Table 5.2: RSME results for different values of $q_x = q_y$

From Table 5.2 it can be seen that for $q_x = q_y = 0.05$ m/s$^2$, the smallest total RMSE value is obtained. Therefore the values of $q_x$ and $q_y$ are set to 0.05 m/s$^2$, and $\mathbf{Q}$ is calculated by (4.19), with the sample time $T$ equal to 0.01 s.

## 5.3 Summary

In this chapter four different test scenarios have been discussed. These four different test scenarios will be used to test the performance of the object tracking algorithm. All the four test scenarios represent a different traffic situation which include different actors consisting of cars, a truck and a bicycle. Based on these test scenarios radar and camera data will be generated. Additionally, the position and velocity of the actors in the scenario will be used as a ground truth data set. Furthermore, the settings and tuning of several parameters that are part of the clustering algorithm and the multi-object tracker have been discussed.

# Chapter 6

# Testing results of the object tracking algorithm

In this chapter the measurement data generated by the radar and camera sensor generators and the obtained tracking results for each scenario, will be compared to a ground truth data set. This ground truth data set is generated from the scenario designer and consist of the actual position and velocity of each actor. Based on the measurement data and generated tracks, conclusions will be drawn about the performance of the object tracker.

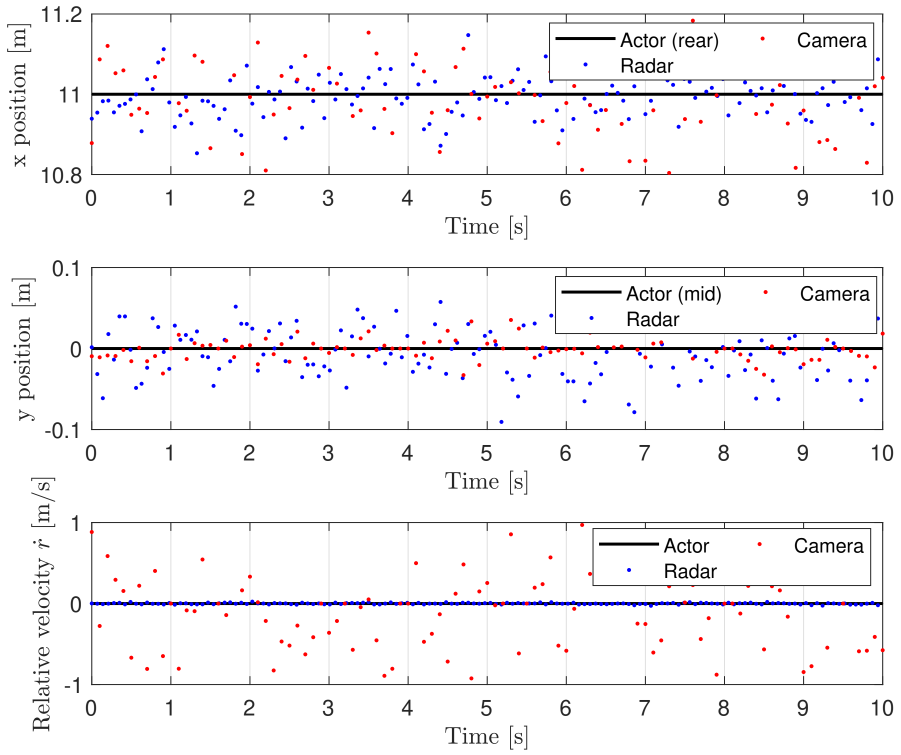
## 6.1 Scenario 1

Scenario 1 represents the most simple example and consists of one actor driving in front of the ego vehicle, while both cars maintain the same speed. The generated measurement results for scenario 1 are shown in Figure 6.1a. From this figure it is shown that camera and radar cluster measurements are generated during the whole simulation. The radar cluster measurements are spread more in the $y$-direction compared to the camera detections, while the radar gives better measurement results for the $x$-position and the relative radial velocity. Since the radar does not measure linear velocities, the radial velocity is shown, to be able to compare the radar and camera data with the reference velocity of the actor. Based on these detections, tracks are obtained.
In Figure 6.1b the tracks are compared with the actual position and velocity of the vehicle. As can be seen from this figure, the multi-object tracking algorithm is able to track the position and velocity of the actor very accurate. An overview of the RMSE values and the maximal deviation between track and the reference of the actor is given in Table 6.1.

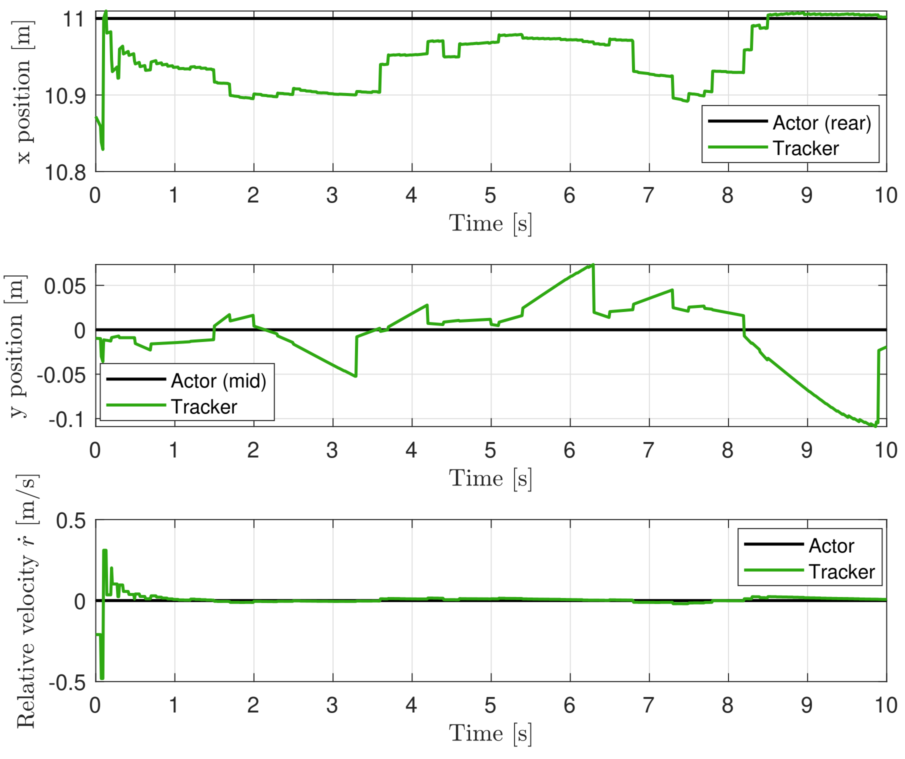
| Error | $x$ | $y$ | $\dot{r}$ |
|---|---|---|---|
| RMSE | 0.06 m | 0.04 m | 0.01 m/s |
| Maximum | 0.11 m | 0.11 m | 0.04 m/s |

Table 6.1: RMSE and maximal errors between track and actor for scenario 1 after initialization

As shown in Figure 6.1b the tracker has a relative large error during the first 0.1 seconds, which is due to the initialization of the tracker, which is done as described in Appendix C. It can be seen that combining radar and camera data results in a better performance than is obtained when only one of the two sensors is used instead. Since the radar gives better measurements in $x$-direction and for the velocity, while the camera gives better measurement results in the $y$-direction. So both advantages are combined to obtain more accurate results. Despite the large spread in velocity measurements by the camera, the track is able to follow the reference very closely with a small deviation from the reference. In Figure E.1 in Appendix, E a combination of the above plots is shown. From the Figures E.2 and E.3 in Appendix E it can be concluded that for scenario 1 correct clusters are created, since for each time step radar detections are generated and one cluster is generated around the middle of the car $y = 0$. From the results for scenario 1, it can be concluded that the tracking algorithm is able to track a vehicle in front of the ego vehicle, while both vehicles drive with the same speed with a very small error with respect to the reference.



(a) Measurement results for scenario 1 (car in front of ego car)



(b) Tracker results for scenario 1 (car in front of ego car)

Figure 6.1: Results for scenario 1

## 6.2 Scenario 2

To determine if the object tracking algorithm is also able to track a vehicle that is driving in front of the ego vehicle while it is accelerating and not driving in a straight line, scenario 2 has been designed. In Figure 6.2a the generated camera and radar cluster measurements are shown.

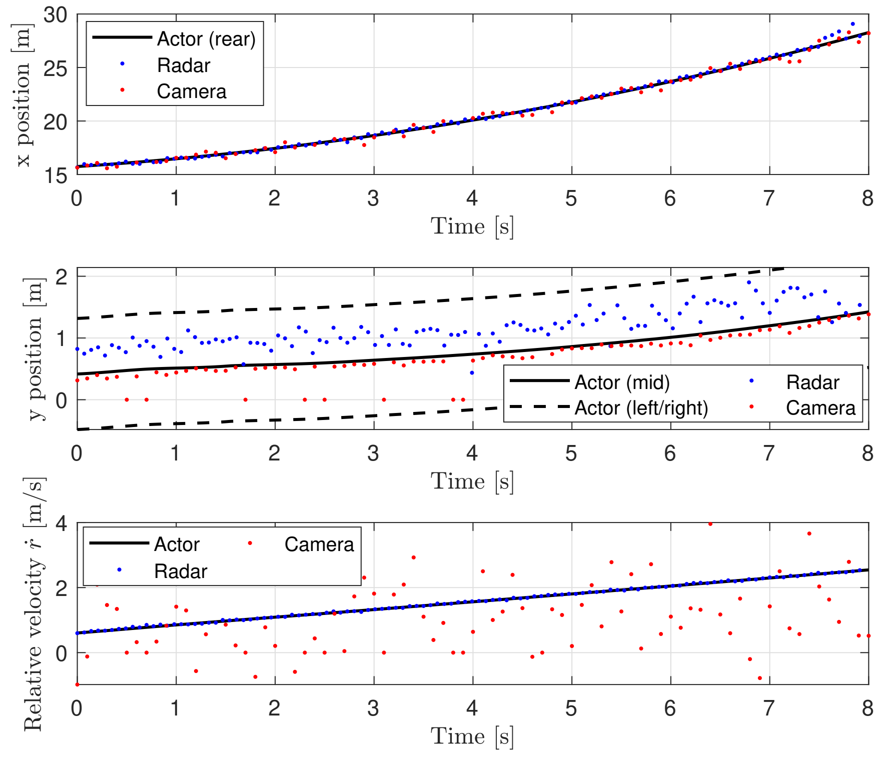
As shown in Figure 6.2a, radar and camera measurements are generated during the complete simulation. For the $x$-position similar measurement results are generated by both the radar and the camera. For the $y$-position it can be seen that there is some more variation in the measurement data generated by the radar and camera. The camera detections are closer to the mid-rear of the car, while the radar detections are a bit more at the left of the rear-side of the car. Furthermore it can be seen that the radar velocity measurements are more accurate compared to the generated camera measurements. From the generated measurements, tracks are obtained as can be seen in Figure 6.2b.

As can be seen from Figure 6.2b the tracker is able to follow the actual position and velocity of the vehicle with a small error. In Figure E.4 in Appendix E a combination of the above plots is shown. An overview of the RMSE values and the maximal errors are shown in Table 6.2.

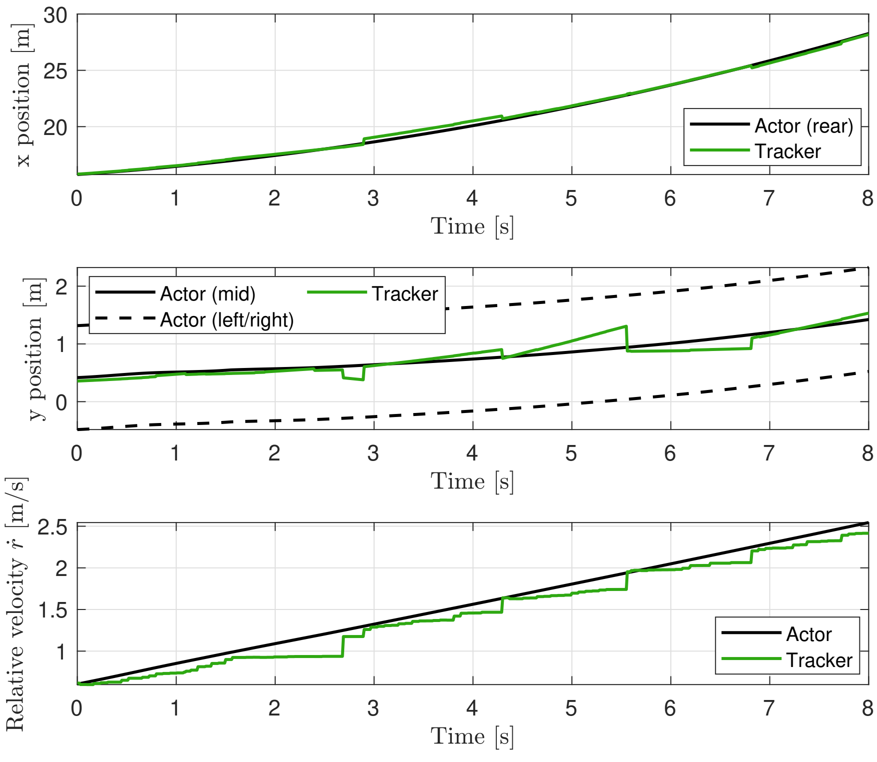
| Error | $x$ | $y$ | $\dot{r}$ |
|---|---|---|---|
| RMSE | 0.20 m | 0.12 m | 0.12 m/s |
| Maximum | 0.40m | 0.37 m | 0.31 m/s |

Table 6.2: RMSE and maximal errors between track and actor for scenario 2

Despite some offset of the radar detections from the mid-rear of the vehicle and a less accurate camera measurement result for velocity, the tracker is able to generate tracks at the mid-rear of the vehicle with a small deviation in the velocity. From the Figures E.5 and E.6 in Appendix E it can be concluded that during the first 7 seconds a single cluster is created. Later on in the scenario some more clusters are formed which is due to the fact that the side of the vehicle becomes more visible, since the vehicle is accelerating away from the ego vehicle and the vehicles are making a turn. Therefore, the radar generates more detections at the side of the vehicle, which results in more than one cluster. From the results obtained from scenario 2, it can be concluded that the tracking algorithm is able to track a vehicle in front of the ego vehicle, while it is driving on a curved road and accelerating away from the ego vehicle with small errors.



(a) Measurement results for scenario 2 (car in front of ego car)



(b) Tracker results for scenario 2 (car in front of ego car)

Figure 6.2: Results for scenario 2

## 6.3 Scenario 3

Test scenario 3 will be used to determine if the object tracking algorithm is able to track multiple vehicles, consisting of two cars and a truck. Since the actors are driving close to each other it is important that the clustering algorithm is able to create the correct cluster for each actor.

**Actor 1**
In scenario 3, actor 1 represents an oncoming car driving in a straight line with constant speed. The generated measurements for actor 1 are shown in Figure 6.3a.

From Figure 6.3a it can be seen that actor 1 is only detected for several seconds. First the radar detects the vehicle and later on from 1.7 seconds also the camera detects the vehicle. After several seconds the vehicle has passed the ego car and no detections are generated any more. Furthermore, it can be seen that the radar generated detections at the side of the vehicle, while the camera generates detections at the middle of the mid-front of the vehicle. Similar to the previous results the camera velocity measurements have a large deviation from the actual velocity compared to the velocity results generated by the radar. Based on these detections, tracks are obtained as can be seen in Figure 6.3b.

From Figure 6.3b it can be seen that the tracker is able to track the vehicle as long as radar or camera measurements are generated. The $y$-direction of the track is most of the time located at the side of the vehicle, which is caused by the radar detections as can be seen in Figure 6.3a. In the $x$-direction the track is located around 1.2 meters in front of the vehicle which is caused by the deviation from the reference of the radar detections. During the last 0.5 seconds the velocities generated by the radar and camera sensor start to deviate more from the reference, which also results in a larger deviation between track and reference during these last 0.5 seconds. An overview of the RMSE values and maximum errors is shown in Table 6.3.

| Error | $x$ | $y$ | $\dot{r}$ |
|---|---|---|---|
| RMSE | 1.19 m | 0.62 m | 0.29 m/s |
| Maximum | 1.38 m | 1.33 m | 4.98 m/s |

Table 6.3: RMSE and maximal errors between track and actor for scenario 3, actor 1

**Actor 2**
In scenario 3, actor 2 represents a car that is driving in front of the ego vehicle in a straight line, while it is accelerating away from the ego car. The generated radar cluster and camera measurements are shown in Figure 6.4a.

As shown in Figure 6.4a radar clusters and camera measurements are generated during the complete simulation. Between 1-3 seconds some radar measurements for the $y$-position are more located at the side compared to the rest of the measurements. Similar to the rest of the scenarios the generated velocity measurements generated by the camera are less accurate compared to the radar cluster measurements. Based on the generated measurements, tracks are obtained as is shown in Figure 6.4b.

As shown in Figure 6.4b and Figure E.8 in Appendix E the tracker is able to follow the vehicle that is driving in front of the ego vehicle during the complete simulation. During the first 1.5 seconds a track is generated which starts to move from the mid-rear to the side of the rear of the vehicle. This is due to some radar measurements which are located more at the side of the car. After 1.5 seconds the tracker is able to follow the reference with a small error. An overview of the RMSE values and the maximum error is given in Table 6.4.

| Error | $x$ | $y$ | $\dot{r}$ |
|---|---|---|---|
| RMSE | 0.07 m | 0.12 m | 0.13 m/s |
| Maximum | 0.17 m | 0.98 m | 0.96 m/s |

Table 6.4: RMSE and maximal errors between track and actor for scenario 3, actor 2

**Actor 3**
In scenario 2, actor 3 represents a truck that is driving next to the ego car. The truck is driving in a straight line with a constant speed, while it is overtaken by the ego car during the simulation. In Figure 6.5a the generated

radar clusters and camera measurements are shown.

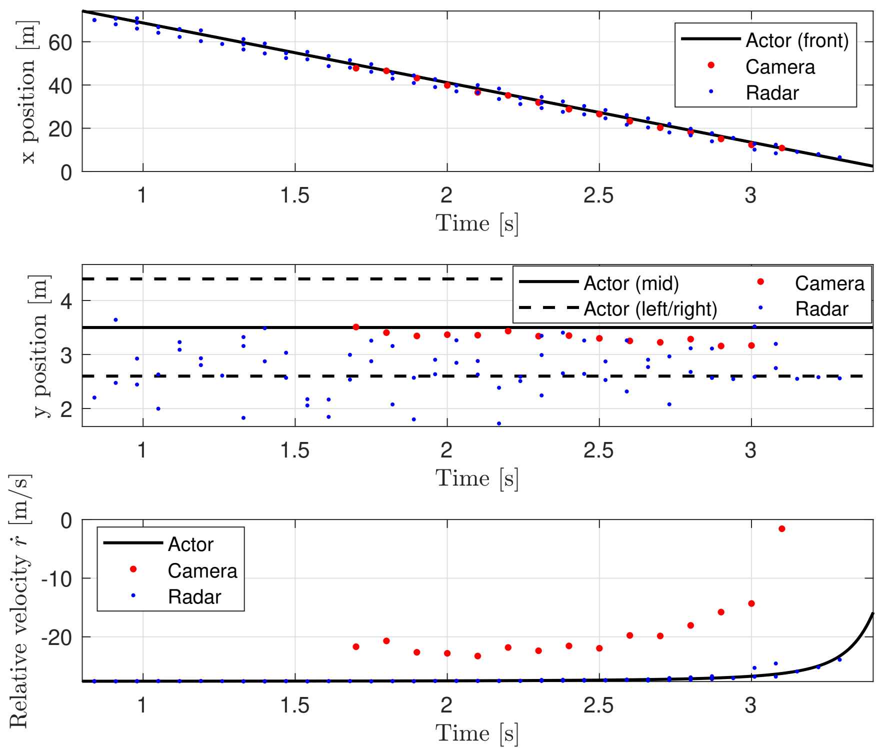
From Figure 6.5a it can be seen that the truck is driving in the field of view of the camera between 0-7 seconds and between 0-3.5 seconds in the field of view of the camera. Furthermore, it can be seen that for the radar most of the time two or more clusters are created. There are clusters which are close to the mid-rear of the truck and clusters which are more located at the side of the truck. Since the truck is very long, more radar detection are generated at the side of the truck compared to a car. Therefore a cluster distance threshold of 2.5 meter results in more than one cluster for the truck. In Figure 6.5b the obtained tracks for this actor are shown.

From Figure 6.5b it can be concluded that multiple tracks are generated for the truck. In Figure E.9 in Appendix E a combined plot of the measurements and tracks is shown. From that figure it can be concluded that the multiple tracks are generated due to multiple radar clusters. From 5 seconds onward there is one track which starts to deviate from the reference. At this moment the rear of the truck is moving out of the field of view of the radar sensor and the measurements in the $y$-direction move from the middle of the truck to the side of the truck. Since the tracks are tracking a different part of the truck, the maximum error from the reference position (mid-rear) is much larger compared to the other actors. An overview of the average and maximum errors between the reference and the tracks is given in Table 6.5.

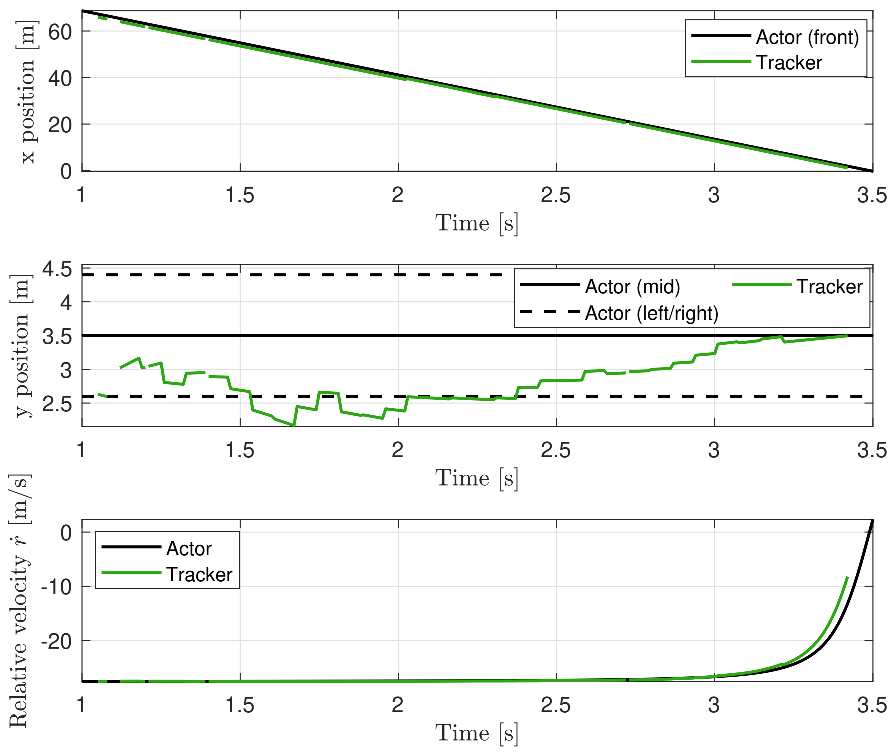
| Error | $x$ | $y$ | $\dot{r}$ |
|---|---|---|---|
| RMSE | 1.19 m | 0.58 m | 0.47 m/s |
| Maximum | 6.83 m | 3.03 m | 2.10 m/s |

Table 6.5: RMSE and maximal errors between track and actor for scenario 3, actor 3

**Conclusion scenario 3**
From the obtained results for scenario 3 it can be concluded that the object-tracking algorithm is able to track multiple vehicles consisting of two cars and a truck. For actor 1 and 3 larger RMSE values and maximum errors are obtained compared to actor 2. This is due to some radar and camera detections which are more located at the side of the car or truck compared to actor 2, where the detections are located at the mid-rear of the car. This can also be seen in Figure E.10 in Appendix E, where the position of all the measurements and clusters is shown. By looking at Figures 6.3a, 6.4a and 6.5a it can be concluded that for all the actors, radar clusters are created as long as the actor is in the field of view of the sensor. In Figure E.11 in Appendix E the number of created radar clusters is shown. From this figure it can be concluded that the maximum number of clusters is equal to 6, while the maximum number of radar detections is equal to 30. Between 0-7 seconds multiple clusters are created since multiple actors are detected during this time interval. After 7 seconds only actor 2 is detected since the other actors are out of the field of view of the radar, which results in a single cluster for the measurements for actor 2.

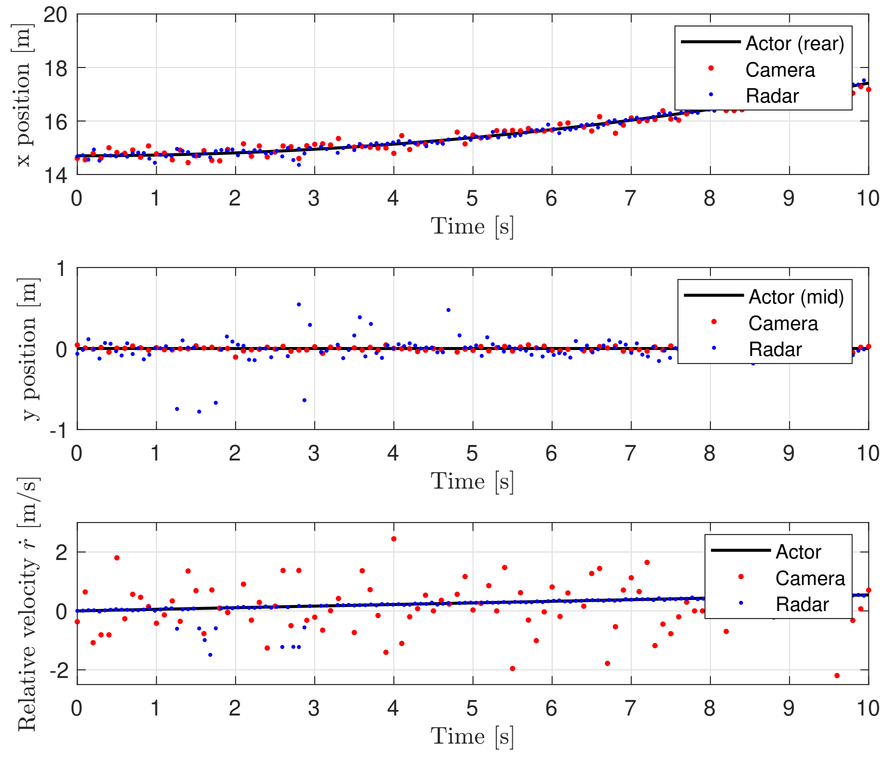


(a) Measurement results for scenario 3, actor 1 (oncoming car)

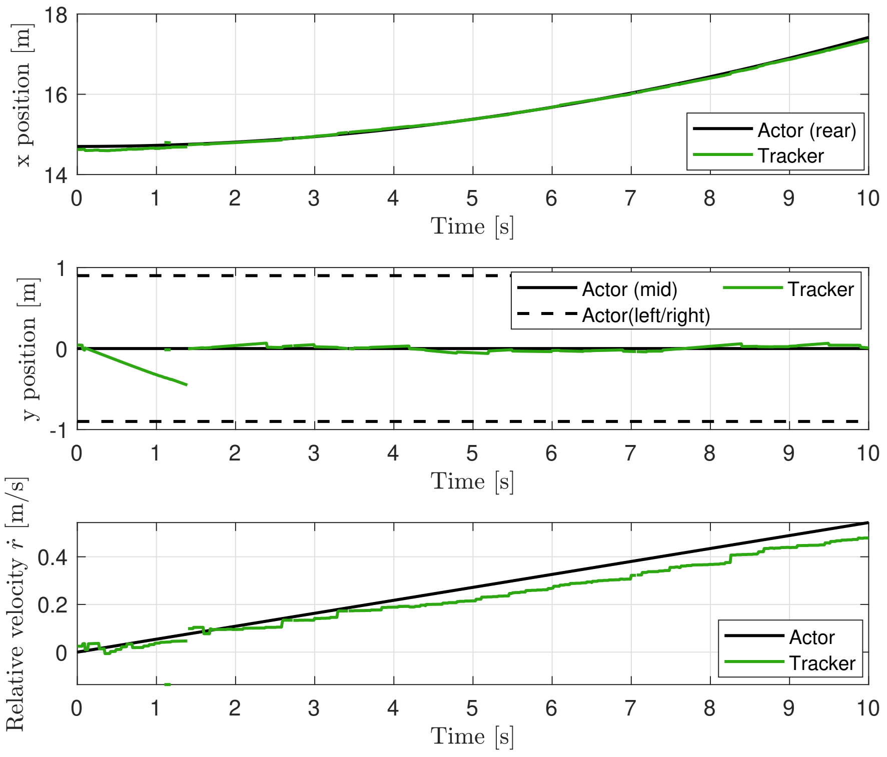


(b) Tracker results for scenario 3, actor 1 (oncoming car)

Figure 6.3: Results for scenario 3, actor 1

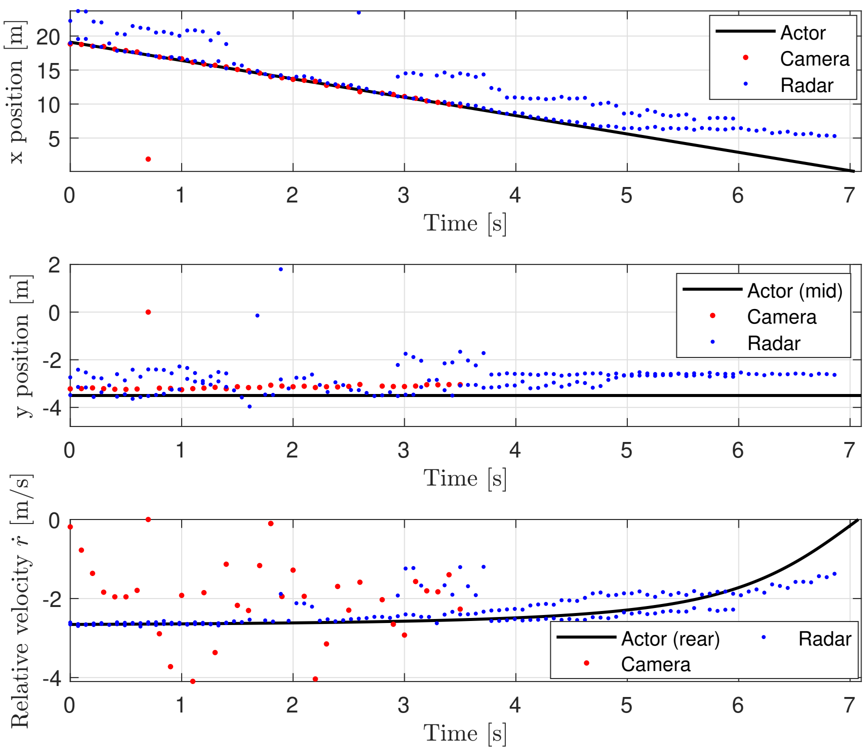


(a) Measurement results for scenario 3, actor 2 (car in front of ego vehicle)

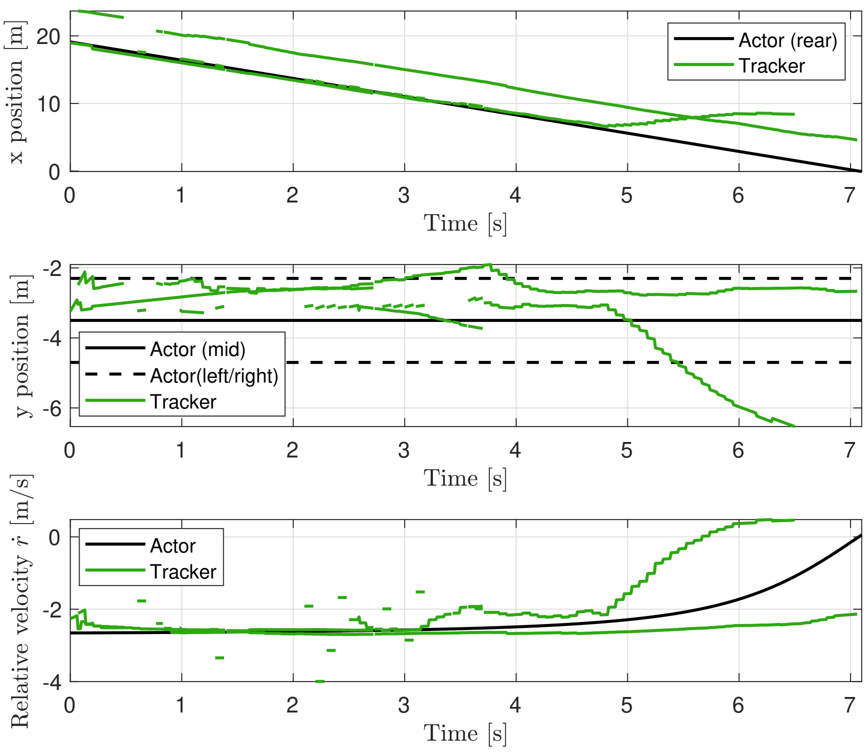


(b) Tracker results for scenario 3, actor 2 (car in front of ego vehicle)

Figure 6.4: Results for scenario 3, actor 2



(a) Measurement results for scenario 3, actor 3 (truck)



(b) Tracker results for scenario 3, actor 3 (truck)

Figure 6.5: Results for scenario 3, actor 3

## 6.4 Scenario 4

Test scenario 4 is the last test scenario that will be used to test if the object tracking algorithm is also able to detect a bicycle besides cars and trucks as in the other test scenarios. Similar as for scenario 3, clustering will be important since the three actors in this scenario are driving close to each other.

**Actor 1**
Similar as in scenario 3, actor 1 in this scenario represents an oncoming car driving in a straight line with constant speed. The generated radar clusters and camera measurements for actor 1 are shown in Figure 6.6a.

From Figure 6.6a it can be seen that for actor 1 multiple radar clusters are created. By looking at the generated radar clusters for the $x$ and $y$-positions it can be seen that clusters are created at the side of front of the vehicle but also more to the rear-side of the vehicle. First the radar detects the vehicle and later on also the camera is able to detect the vehicle. After a few seconds the vehicle has moved out of the field of view of the sensor and no detections are generated anymore. In Figure 6.6b the obtained tracks for actor 1 are shown.

As can be seen from Figure 6.6b and Figure E.12 from Appendix E multiple tracks are generated at a different location. This is due to the radar clusters which are located at the side of the front and more at the rear-side of the vehicle. At 1.7 seconds there is a large deviation in the $y$-position from the reference which is due to a radar detection which is located at the other side of the vehicle. At 5.8 seconds the front of the vehicle is out of the field of view of the radar and only a track located at the side of the vehicle is generated. The track located more at the rear side of the vehicle results in a large average and maximum error between track and reference, since the reference of the vehicle is located at the front of the vehicle. An overview of the RMSE values and maximum errors is given in Table 6.6.

| Error | $x$ | $y$ | $\dot{r}$ |
|---|---|---|---|
| RMSE | 2.83 m | 0.74 m | 2.54 m/s |
| Maximum | 5.48 m | 1.82 m | 10.11 m/s |

Table 6.6: RMSE and maximal errors between track and actor for scenario 4, actor 1

**Actor 2**
Actor 2 in scenario 4 represents a car that is driving in front of the ego vehicle while it is accelerating away from the ego car. The generated radar clusters and camera measurements for actor 2 are shown in Figure 6.7a.

From Figure 6.7a it can be seen that the radar and camera are able to detect the vehicle during the whole simulation. Around 5 seconds some radar cluster measurements are located more to the right of the vehicle. Around this time step the bicycle (actor 3) is riding very close to actor 2 and the clustering algorithm has some difficulties to generate correct clusters for actor 2. Since clusters are created which consist of measurements located at actor 2 and located at actor 3, the new core point of the cluster is in the middle of the two actors. In Figure 6.7b the obtained tracks for actor 2 are shown.

From Figure 6.7b it can be concluded that the tracker is able to track actor 2 very close to the reference. Since the camera detections are located very close to the mid-rear of the vehicle, the tracker is able to follow the ground truth very closely, despite some incorrect located clusters around 5 seconds. An overview of the average and maximum errors between the track and ground truth data is given in Table 6.7.

| Error | $x$ | $y$ | $\dot{r}$ |
|---|---|---|---|
| RMSE | 0.08 m | 0.12 m | 0.13 m/s |
| Maximum | 0.20 m | 0.44 m | 0.59 m/s |

Table 6.7: RMSE and maximal errors between track and actor for scenario 4, actor 2

**Actor 3**
Actor 3 in scenario 4 represents a bicycle, which is the most challenging actor in all the scenarios because of the size of the actor. The size of a bicycle is much smaller compared to a car or truck, which makes it difficult for the sensors to detect this actor. Furthermore, the bicycle is driving close to actor 2, which makes it difficult for the clustering algorithm to generate correct clusters for both actors. In Figure 6.8a the generated radar clusters

and camera measurements are shown.

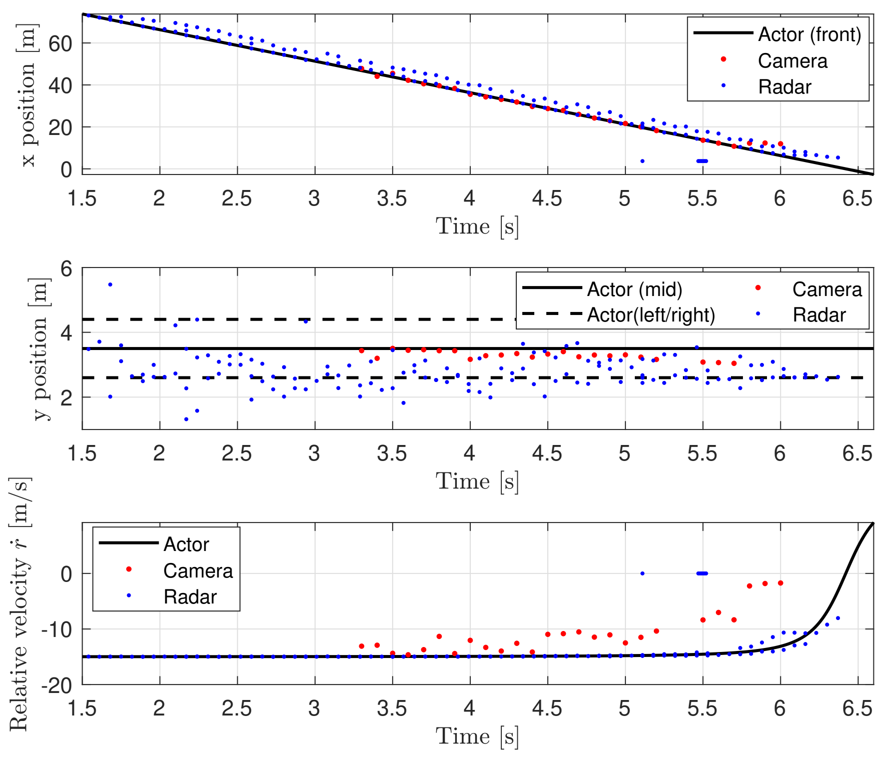
From Figure 6.8a it can be seen that the radar has some difficulties with detecting the bicycle, which results in radar clusters located at the rear of the bicycle, but also around 5 meters in front of the bicycle as can be seen when looking at the $x$-position. Since the bicycle is very small and driving very close to actor 2 the generated radar clusters are located between those actors. The camera is better able to detect the bicycle at the correct location compared to the radar, while the radar is better able to measure the correct velocity of the bicycle compared to the camera. In Figure 6.8b the obtained tracks for actor 3 are shown.

As can be seen from Figure 6.8b the tracker is able to track the bicycle, but with a relative large error for the $y$-position, which is due to the location of the radar measurements and clusters. The peaks in the track velocity around 2.5 seconds and 6 seconds is due to some incorrect clusters. The velocity of actor 2 is positive since it is driving away from the ego vehicle and the relative velocity of the bicycle is negative since it is overtaken by the ego car. Incorrect radar clusters for the bicycle consist of some detections originating from actor 2, which results in a higher relative velocity for the new core point of the cluster. An overview of the average and maximum error between the tracks and the ground truth for actor 3 is given in Table 6.8.

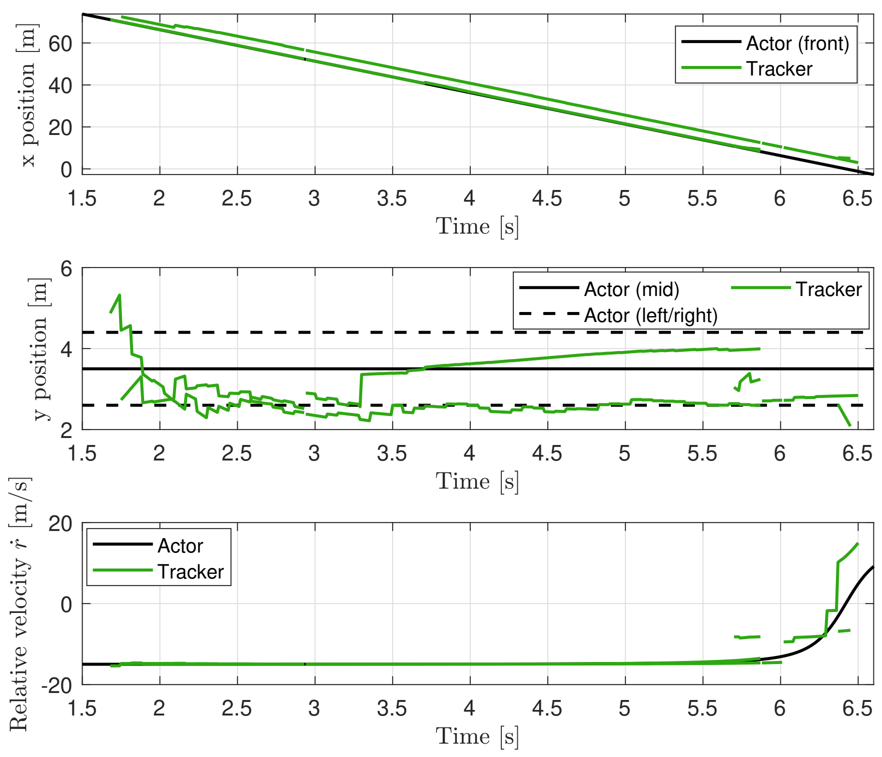
| Error | $x$ | $y$ | $\dot{r}$ |
|---|---|---|---|
| RMSE | 0.27 m | 0.26 m | 0.31 m/s |
| Maximum | 2.05 m | 0.94 m | 1.87 m/s |

Table 6.8: RMSE and maximal errors between track and actor for scenario 4, actor 3

**Conclusion scenario 4**
From the results for scenario 4 it can be concluded that the object-tracking algorithm is able to track all the actors in scenario 4, consisting of two cars and a bicycle. Similar as for scenario 3 larger RMSE values and maximum errors are obtained for the actors 1 and 3. For actor 1 multiple tracks are generated, located at the front of the vehicle and more to the rear of the vehicle. The tracker has some difficulties to estimate the the correct $y$-position for the bicycle. This is due to some incorrect clusters, since the bicycle is driving very close to actor 2. In Figure E.15 in Appendix E the clusters are shown. From this figure it can be concluded that the radar measurements and clusters for actor 2 and 3 are very close to each other and clusters are created which are located between the two actors. From Figure E.16 in Appendix E it can be concluded that multiple clusters are created during the simulation. Between 1.5-6.5 seconds more clusters are generated since during that time period all the three actors are detected by the radar. Between 0-1.5 seconds and 6.5-10 seconds only actors 2 and 3 are detected by the radar, which results in a reduced number of clusters.

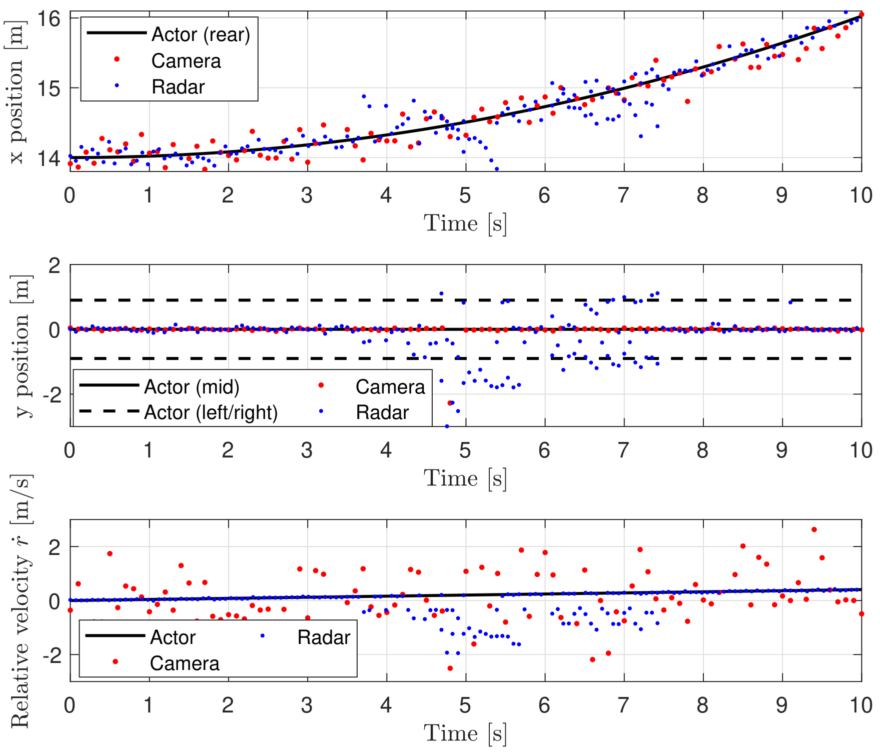


(a) Measurement results for scenario 4, actor 1 (oncoming car)

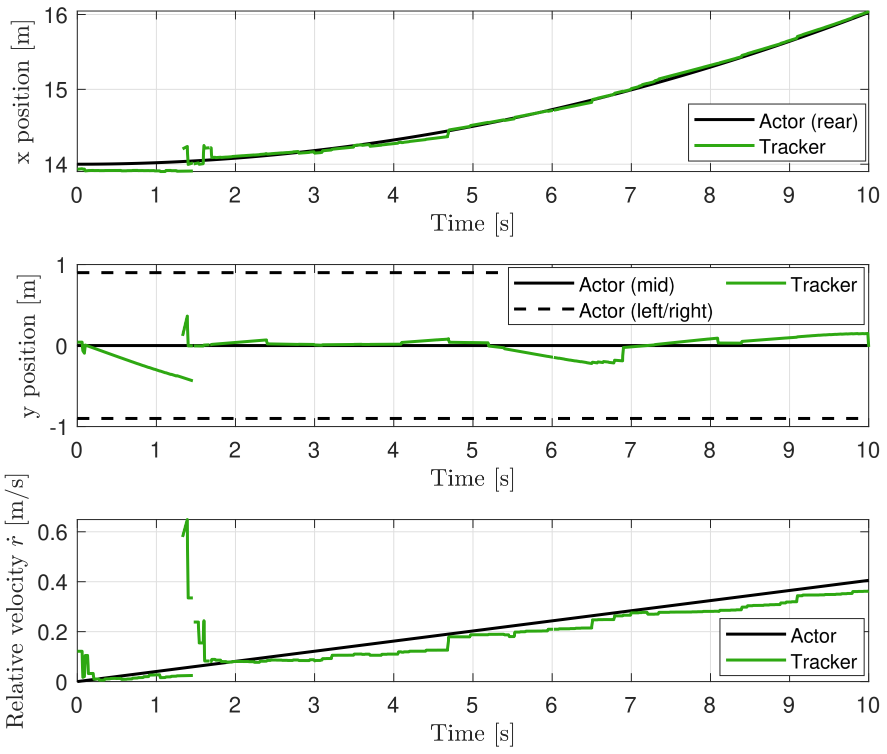


(b) Tracker results for scenario 4, actor 1 (oncoming car)

Figure 6.6: Results for scenario 4, actor 1

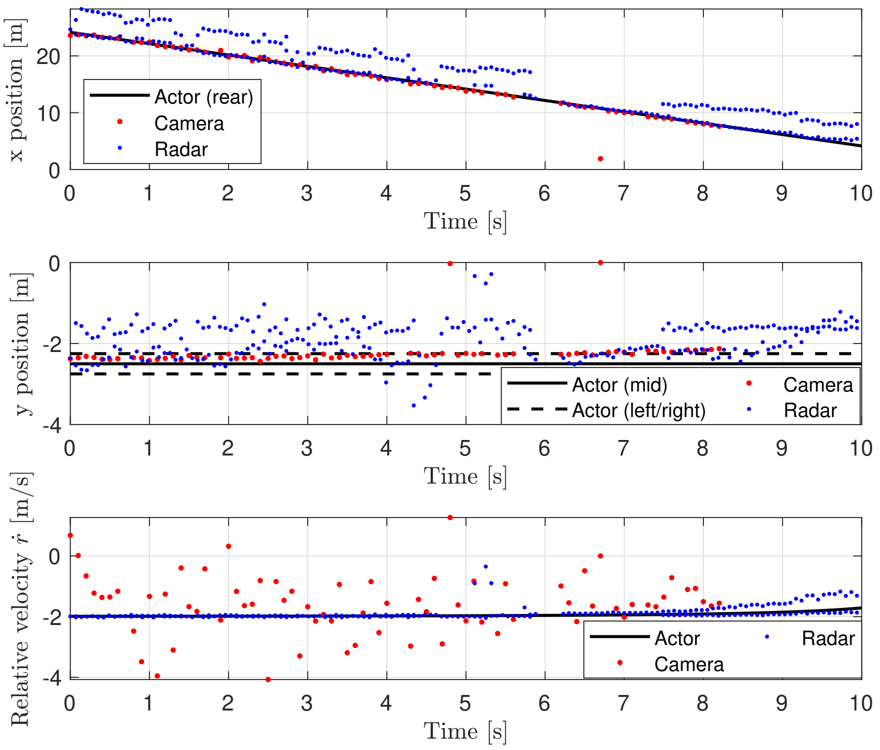


(a) Measurement results for scenario 4, actor 2 (car in front of ego vehicle)

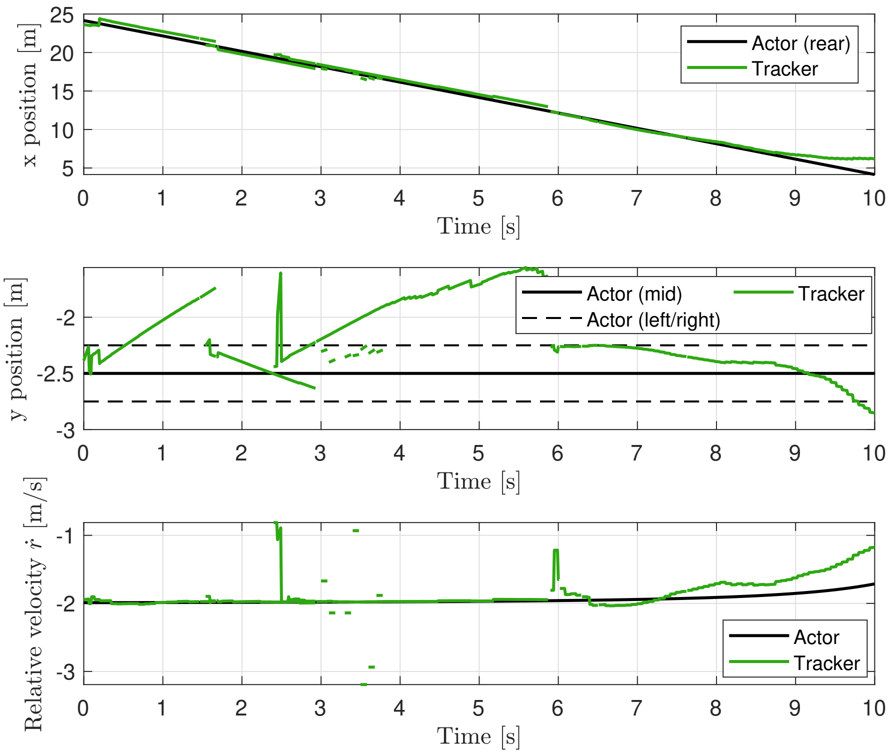


(b) Tracker results for scenario 4, actor 2 (car in front of ego vehicle)

Figure 6.7: Results for scenario 4, actor 2



(a) Measurement results for scenario 4, actor 3 (bicycle)



(b) Tracker results for scenario 4, actor 3 (bicycle)

Figure 6.8: Results for scenario 4, actor 3

## 6.5 Summary

In this chapter the test results of the object tracking algorithm have been discussed. The object tracking algorithm has been tested for four different test scenarios. From these four scenarios it can be concluded that the object tracking algorithm is able to detect all the actors present in the scenarios. The best tracking results with the smallest errors are obtained for the actors driving in front of the ego vehicle. Oncoming vehicles, the truck and the bicycle are also detected, but since more detections are generated at the side of these actors compared to the vehicles driving in front of the ego vehicle more tracks are generated located at different positions, which results in a larger RMSE value. By looking at the clustering results it can be concluded that most of the time correct clusters are created. Since in scenario 4 actor 2 and 3 are driving very close to each other, the clustering algorithm has some difficulties in obtaining the correct clusters. But despite some incorrect clusters, the tracker is still able to track those actors.

# Chapter 7

# Conclusion and recommendations

In this final chapter the conclusion and recommendations of this thesis are given.

## 7.1 Conclusion

The goal of this graduation project was to develop and test an algorithm that is able to track multiple objects by fusing radar and camera data. An object tracking model has been developed which consists of radar and camera sensor generators, a radar clustering algorithm and a multi-object tracker. Since no real measurements could be generated for testing the tracking algorithm, radar and camera detections have been generated based on a designed driving scenario and specific sensor characteristics. Since the radar can produce multiple detections representing one vehicle, a radar clustering algorithm has been developed. With this algorithm clusters are created based on the difference in position and velocity for the generated measurement results. Based on the clustered radar measurements and the camera measurements, state predictions have been obtained by an extended Kalman filter. By assuming that no large accelerations will occur during a sample time period and no highly dynamic manoeuvres will occur, a constant velocity model has been used to predict the position and velocity for the next time step. To be able to associate the predicted state estimations from the current time step to the already existing track obtained from the previous time steps, gating and the GNN method have been used. Finally, tracks have been generated which consist of estimated positions and velocities of the detected objects.

To be able to test the object tracking algorithm four different test scenarios have been designed. Each scenario represents a situation that often occurs in urban or provincial areas. In the different scenarios one or three actors are used. These actors represent cars, a truck and a bicycle to test if the object tracking algorithm is able to track different and multiple vehicles at the same time. Furthermore, the parameters present in the model, such as the threshold values for the clustering algorithm, parameters in the multi-object tracker block and the Q matrix of the Kalman have been determined to obtain the best results possible.

To determine the performance of the multi-object tracking algorithm, the tracking results for each actor are compared to the generated radar and camera measurements and a ground truth data set, which consists of the actual position and velocity of the actor present in the test scenario.
From the first test scenario it can be concluded that the tracker is able to track the vehicle's position and velocity with a small error between the track and the reference of the actor. At the beginning of the scenario some relative large errors occur, which is due to the initialization of the tracker.
For the seconds scenario, which consists of an accelerating vehicle driving on a curved road, it can be concluded that the tracker is able to track the vehicle. The errors between track and reference for scenario 2 are larger compared to the errors of scenario 1, but still the errors are small. The thirth and fourth scenario are designed to test if the object tracking algorithm is able to detect multiple actors at the same time. From both scenarios it can be concluded that similar tracking results are obtained. The best tracking result in both scenarios is obtained for the car in front of the ego car. The tracking algorithm is able to track the vehicle during the complete simulation time with small errors. For the actors 1 and 3 larger errors are obtained compared to actor

2. This is due to the fact that there are radar clusters generated at different locations at the actor, since the radar detects more of the side of these actors instead of only the rear as for actor 2. This results in more tracks, located at different parts of the actor. By looking at the radar cluster results it can be concluded that most of the time correct cluster are created. Since in scenario 4 actors 2 and 3 are driving very close to each other, the radar clustering algorithm has some difficulties to create the correct tracks. But despite some incorrect clusters, the tracker is still able to track both actors.
Overall it can be concluded that the object tracking algorithm is able to detect all the actors in the four scenarios. The best results are obtained for the actors driving in front of the ego car. Since the final goal is to use the object tracking algorithm for cooperative driving, that will be the most important actor that needs to be tracked.

## 7.2 Recommendations

The tracking algorithm is able to detect multiple objects, but to improve the object tracking algorithm and for further testing, several recommendations should be taken into account. At first it would be better if only one single track per actor can be generated instead of multiple tracks that are now obtained in the scenarios 3 and 4. Only for scenarios 1 and 2, one track is generated. But if the scenarios become more dynamic or more complex, the object tracking algorithm is not able to obtain a single track. To solve this problem the clustering algorithm needs to be improved. Better clustering results may be obtained by better tuning of the cluster parameters and look for other possible clusters shapes or categories.
A second recommendation is to test more complex and dynamic situations to determine if the extended Kalman filter is able to track those dynamic manoeuvres. For the designed test scenarios a constant velocity model was good enough, but if situations and manoeuvres become more complex probably other models such as a constant acceleration or maybe even an interacting multiple model (IMM model), which makes use of a combination of multiple motion models depending on the situation, will improve the performance of the object tracking algorithm.
During the project parts of the multi-object tracker block from MATLAB have been used. Since it is very complex to understand all the code behind this block and to figure out where something is within the code, it is very difficult to adapt things within the code. Therefore it is recommended to obtain all the algorithms used for data association, gating, Kalman filtering and track maintenance separately. In that way it will be easier to make changes to the model and to see what is happening in every part of the model. It will also become more obvious how to tune the model properly, since it will be easier to see what is happening with all the outputs of the different algorithms.
The last recommendation will be to test the tracking algorithm with real measurement data generated from some tests with the Twizy and use this measurement data as an input for the model. In this way it can be determined if the object tracking algorithm is also working properly with real measurement data instead of the measurement data generated by the sensor generator blocks in the model. The final step then will be to implement the object tracking algorithm on the Twizy and test the object tracking algorithm in real situations and real-time.

# Bibliography

[1] B. Dasgupta A.M Goon, M.K Gupta. *Fundamentals of Statistics*, volume 1. SAGE Publications, 1968.

[2] Preeti Arora, Deepali, and Shipra Varshney. Analysis of k-means and k-medoids algorithm for big data. *Procedia Computer Science*, 78:507–512, 12 2016.

[3] Sanat Biswas. *Computationally Efficient Non-linear Kalman Filters for On-board Space Vehicle Navigation.* PhD thesis, The university of New South Wales, Syndney, Australia, 7 2017.

[4] F.J. Botha, C.E. van Daalen, and J. Treurnicht. Data fusion of radar and stereo vision for detection and tracking of moving objects. pages 1–7, Nov 2016.

[5] Federico Castanedo. A review of data fusion techniques. *The Scientific World Journal*, 2013:704504, 10 2013.

[6] K. C. Chang, Tian Zhi, and R. K. Saha. Performance evaluation of track fusion with information matrix filter. *IEEE Transactions on Aerospace and Electronic Systems*, 38(2):455–466, April 2002.

[7] R. Omar Chavez-Garcia. *Multiple Sensor Fusion for Detection, Classification and Tracking of Moving Objects in Driving Environments.* PhD thesis, Université de Grenoble, 2014.

[8] Xiao Dong Chen, Xiao Wang, and Jianhua Xuan. Tracking multiple moving objects using unscented kalman filtering techniques. *ArXiv*, abs/1802.01235, 2018.

[9] Jeremy Cohen. Sensor fusion. *Towards Data Science*, 2018. Paris School of AI Head Dean of France.

[10] M. Darms and H. Winner. A modular system architecture for sensor data processing of adas applications. In *IEEE Proceedings. Intelligent Vehicles Symposium, 2005.*, pages 729–734, June 2005.

[11] Henderson T.C. Durrant-Whyte H. *Multisensor Data Fusion.* Springer, Cham, Berlin Heidelberg, 2016.

[12] J. Elfring, R.P.W. Appeldoorn, S. van den Dries, and M.R.J.A.E. Kwakkernaat. Effective world modeling: multisensor data fusion methodology for automated driving. *Sensors*, 16(10), 2016.

[13] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, pages 226–231. AAAI Press, 1996.

[14] Martin Ester and Jörg Sander. Knowledge discovery in databases. 01 2000.

[15] Annabel Ness Evans. *Using Basic Statistics in the Behavioral and Social Sciences.* SAGE Publications, 2014.

[16] R. Faragher. Understanding the basis of the kalman filter via a simple and intuitive derivation [lecture notes]. *IEEE Signal Processing Magazine*, 29(5):128–132, Sep. 2012.

[17] M. L. Fung, M. Z. Q. Chen, and Y. H. Chen. Sensor fusion: A review of methods and applications. In *2017 29th Chinese Control And Decision Conference (CCDC)*, pages 3853–3860, May 2017.

[18] Anton J. Haug. *Bayesian Estimation and Tracking: A Practical Guide*, volume 1. Wiley, 2012.

[19] L.G.M. Hendrix. Extended density-based clustering algorithm for automobile radar applications. BEP report, Technical University Eindhoven, ID 1012441.

[20] Shaun M. Howard. Deep learning for sensor fusion. Master's thesis, Department of Eletrical Engineering and Computer Science, Case Western Reserve University, 2017.

[21] Eric Jones, Maria Scalzo, Adnan Bubalo, Mark Alford, and Benjamin Arthur. Measures of nonlinearity for single target tracking problems. *Proceedings of the SPIE Signal Processing, Sensor Fusion and Target Recognition XX*, 05 2011.

[22] N. Kemsaram, A. Das, and G. Dubbelman. An integrated framework for autonomous driving: Object detection, lane detection, and free space detection. In *2019 Third World Conference on Smart Trends in Systems Security and Sustainablity (WorldS4)*, pages 260–265, 2019.

[23] M. Klompmakers. Object tracking for autonomous and cooperative driving. Master's thesis, Department of Mechanical Engineering, Technical University of Technology Eindhoven, 2020.

[24] Pawel Kmiotek. Multi-sensor data fusion for representing and tracking dynamic objects. 12 2009.

[25] Pavlina Konstantinova, Alexander Udvarev, and Tzvetan Semerdjiev. A study of a target tracking algorithm using global nearest neighbor approach. 01 2003.

[26] Y. Kosuge and T. Matsuzaki. The optimum gate shape and threshold for target tracking. In *SICE 2003 Annual Conference*, volume 2, pages 2152–2157 Vol.2, 2003.

[27] LEOPARD IMAGING INC. *LI-AR0231-GSML-xxH Data Sheet.*

[28] Christian Lundquist. *Sensor Fusion for Automotive Applications.* PhD thesis, Department of Electrical Engineering, Linköping University, SE-581 83 Linköping, Sweden, 2011.

[29] R. C. Luo, C. C. Chang, and C. C. Lai. Multisensor fusion and integration: Theories, applications, and its perspectives. *IEEE Sensors Journal*, pages 3122–3138, Dec 2011.

[30] Ronald P. S. Mahler. *Statistical Multisource-Multitarget Information Fusion.* Artech House, Inc., Norwood, MA, USA, 2007.

[31] M. S. Mahmud, M. M. Rahman, and M. N. Akhtar. Improvement of k-means clustering algorithm with better initial centroids based on weighted average. In *2012 7th International Conference on Electrical and Computer Engineering*, pages 647–650, 2012.

[32] Shaaban Mahran and Khaled Mahar. Using grid for accelerating density-based clustering. *2008 8th IEEE International Conference on Computer and Information Technology*, pages 35–40, 2008.

[33] MATLAB. Constant velocity motion model. `https://uk.mathworks.com/help/fusion/ref/constvel.html`, 2020.

[34] MATLAB. Coordinate systems in automated driving toolbox. `https://uk.mathworks.com/help/driving/ug/coordinate-systems.html`, 2020.

[35] MATLAB. Introduction to assignment methods in tracking systems. `https://uk.mathworks.com/help/fusion/ug/introduction-to-assignment-methods-in-tracking-systems.html`, 2020.

[36] MATLAB. Multi object tracking. `https://uk.mathworks.com/help/vision/ug/multiple-object-tracking.html`, 2020.

[37] J. McMillan and Sang Seok Lim. Data association algorithms for multiple target tracking. page 37, 07 1990.

[38] NXP. *Cocoon user manual*, 2017. Revision 2.2.

[39] NXP. Nxp announces a complete suite of radar sensor solutions that can surround vehicles in a 360-degree safety cocoon. 12 2020.

[40] Andrew O' Riordan, Thomas Newe, Daniel Toal, and Gerard Dooly. Stereo vision sensing: Review of existing systems. 12 2018.

[41] S. Omar and S. Winberg. Multisensor data fusion: Target tracking with a doppler radar and an electro-optic camera. In *2011 IEEE International Conference on Control System, Computing and Engineering*, pages 210–215, 2011.

[42] Luis Ortiz, Elizabeth Cabrera, and Luiz Gonçalves. Depth data error modeling of the zed 3d vision sensor from stereolabs. *Electronic Letters on Computer Vision and Image Analysis*, 17, 04 2018.

[43] Michael Parker. *Digital Signal Processing 101.* Newnes, 2017.

[44] V Parthasarathi Naidu. *Kinematic and image data fusion with application to tracking of moving target.* PhD thesis, Department of Electronics, University of Mysore, 2007.

[45] Susan Luu Pia Lidman. Clustering, shape extraction and velocity estimation applied to radar detections. Master's thesis, Department of Eletrical Engineering, Division of Systems and Control, Chalmers University of Technology, 2018.

[46] R. Kavitha R. Tamilselvi, B. Sivasakthi. A comparison of various clustering methods and algorithms in data mining. May 2015.

[47] K.V. Ramachandra. *Kalman Filtering Techniques for Radar Tracking.* Marcel Dekker Inc., 200.

[48] A. Rattani, D. R. Kisku, M. Bicego, and M. Tistarelli. Feature level fusion of face and fingerprint biometrics. In *2007 First IEEE International Conference on Biometrics: Theory, Applications, and Systems*, pages 1–6, Sep. 2007.

[49] Maria Isabel Ribeiro. Kalman and extended kalman filters: Concept, derivation and properties. Master's thesis, Institute for Systems and Robotics, 1049-001 Lisboa Portugal, 2004.

[50] X. Rong Li and V. P. Jilkov. Survey of maneuvering target tracking. part i. dynamic models. *IEEE Transactions on Aerospace and Electronic Systems*, 39(4):1333–1364, 2003.

[51] Francisca Rosique, Pedro Navarro Lorente, Carlos Fernandez, and Antonio Padilla. A systematic review of perception system and simulators for autonomous vehicles research. *Sensors*, 19:648, 02 2019.

[52] E. A. Wan and R. Van Der Merwe. The unscented kalman filter for nonlinear estimation. In *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium*, pages 153–158, Oct 2000.

[53] Y. Wei, H. Meng, H. Zhang, and X. Wang. Vehicle frontal collision warning system based on improved target tracking and threat assessment. In *2007 IEEE Intelligent Transportation Systems Conference*, pages 167–172, 2007.

[54] T.E. Fortmann Y. Bar-Shalom. *Tracking and Data Association.* Academic Press, New York, 1988.

[55] Ya Xue and D. Morrell. Target tracking and data fusion using multiple adaptive foveal sensors. In *Sixth International Conference of Information Fusion, 2003. Proceedings of the*, volume 1, pages 326–333, July 2003.

[56] Desai U.B. Zaveri M.A., Merchant S.N. *Data Association for Multiple Target Tracking: An Optimization Approach.* Springer, Berlin, Heidelberg, 2004.

Appendix A

# Outputs of the scenario reader, sensor generators and multi-object tracker

Outputs of the scenario reader (scenario actor poses):

| | | |
|---|---|---|
| | NumActors | Number of actors (ego vehicle excluded) |
| | Time | Current simulation time [s] |
| Actors | ActorID | Scenario-definied actor ID |
| | Position | Position of the actor [x y z] in [m] |
| | Velocity | Velocity of the actor [$v_x$ $v_y$ $v_z$] in [m/s] |
| | Roll | Roll angle of the actor in [°] |
| | Pitch | Pitch angle of the actor in [°] |
| | Yaw | Yaw angle of the actor in [°] |
| | AngularVelocity | Angular velocity of the actor [$\omega_x$ $\omega_y$ $\omega_z$] in [$°/s$] |

Table A.1: Output of the scenario reader

Outputs of the vision and radar generators and detection concatenation:

| | | |
|---|---|---|
| | NumDetections | Number of detections |
| | IsValidTime | True at times the sensor updates (Boolean) |
| Object detection | Time | Current simulation time [s] |
| | Measurement | Position and velocity of the measurement in spherical or Cartesian coordinates |
| | MeasurementNoise | Measurement noise covariance matrix |
| | SensorIndex | Unique sensor index |
| | ObjectClassID | Object classification |
| Measurement parameters | Frame | Frame used to report measurements (spherical or Cartesian) |
| | OriginPosition | Offset of the sensor origin from the ego vehicle origin in [m] |
| | Orientation | Orientation of the sensor coordinate system with respect to the ego vehicle coordinate system, derived from the yaw angle, pitch angle and roll angle of the sensor |
| | HasVelocity | Indicates whether measurement contain velocity (boolean) |
| | HasEveleration (only for radar) | Indicates whether measurements contain elevation components (Boolean) |
| ObjectAttributes | TargetIndex | Identifier of the actor. For false alarms this value is negative |
| | SNR (only for radar) | Signal-to-noise ratio of the detection in [dB] |

Table A.2: Output of the camera and radar generators and detection concatenation

Outputs of the multi-object tracker:

| | | |
|---|---|---|
| | NumTracks | Number of tracks |
| Track | TrackID | Unique integer that identifies the track |
| | Time | Current simulation time [s] |
| | Age | Number of time the track survived updating |
| | State | Values of the state vector at the update time |
| | StateCovariance | Uncertainty covariance matrix |
| | IsConfirmed | True is the track is assumed to be of a real target (boolean) |
| ObjectAttributes | TargetIndex | Identifier of the track |
| | SNR | Signal-to-noise ratio of the track |

Table A.3: Output of the multi-object tracker

# Appendix B

# Clustering

## B.1 Proof of the combined variances for a cluster

The equations described in (4.12) that can be used to calculate the combined variance for a cluster is coming from the sum of squares of deviations. The proof of this equation is as follows [1]:

**Theorem**
Let a cluster consists of the points mean points $\bar{x}_1$, $\bar{x}_2$, ..., $\bar{x}_n$, that each consists of $i = 1, ..., n_1$, $j = 1, ..., n_2$ and $k = 1, .., n_n$ number of points. The standard deviations of each cluster are given by $\sigma_{x1}$, $\sigma_{x2}$, ..., $\sigma_{xn}$. The mean (core point) of the cluster is described by $\bar{x}$, see Figure B.1.

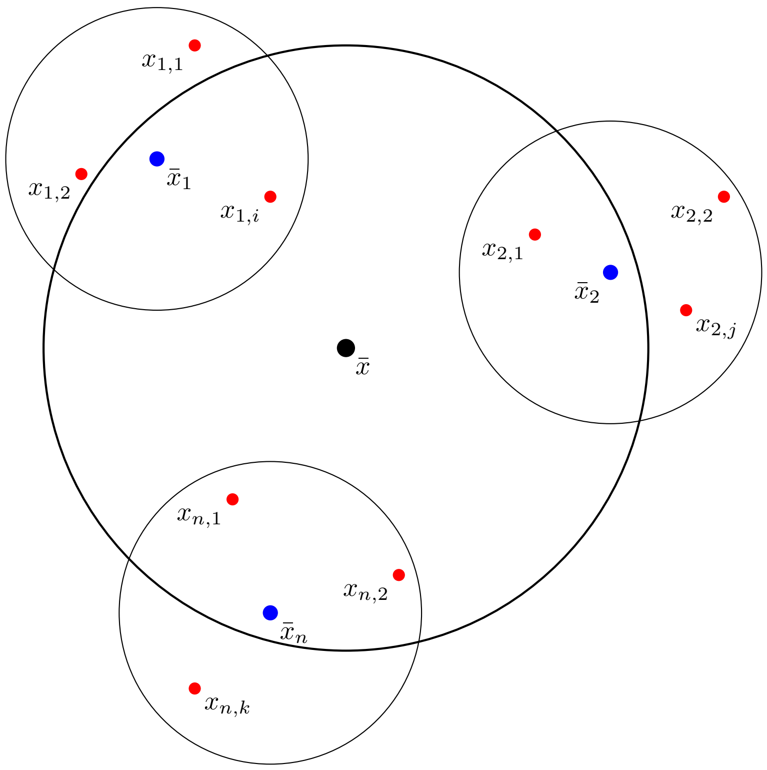


Figure B.1: Overview of a cluster

Then the combined standard deviation of the cluster can be calculated by:

$$\sum_i (x_{1,i}-\bar{x})^2+\sum_j (x_{2,j}-\bar{x})^2+...+\sum_k (x_{n,k}-\bar{x})^2 = n_1\sigma_{x1}^2+n_1(\bar{x}_1-\bar{x})^2+n_2\sigma_{x2}^2+n_1(\bar{x}_2-\bar{x})^2+n_n\sigma_{x2}^2+n_n(\bar{x}_n-\bar{x})^2 \tag{B.1}$$

**Proof**

$$\begin{aligned}
\sum_i^{n_1}(x_{1,i}-\bar{x})^2 &= \sum_i^{n_1}\Big((x_{1,i}-\bar{x}_1)+(\bar{x}_1-\bar{x})\Big)^2 \\
&= \sum_i^{n_1}(x_{1,i}-\bar{x}_1)^2+\sum_{i=1}^{n_1}(\bar{x}_1-\bar{x})^2+2\sum_{i=1}^{n_1}\Big((x_{1,i}-\bar{x}_1)(\bar{x}_1-\bar{x})\Big) \\
&= \sum_{j=1}^{n_1}(x_{1,i}-\bar{x}_1)^2+\sum_{i=1}^{n_1}(\bar{x}_1-\bar{x})^2+2(\bar{x}_1-\bar{X})\sum_{i=1}^{n_1}(x_{1,i}-\bar{x}_1) \\
&= \sum_{i=1}^{n_1}(x_{1,i}-\bar{x}_1)^2+\sum_{i=1}^{n_1}(\bar{x}_1-\bar{x})^2+2(\bar{x}_1-\bar{X})\Big(\sum_{i=1}^{n_1}x_{1,i}-\sum_{i=1}^{n_1}\bar{x}_1\Big) \\
&= \sum_i^{n_1}(x_{1,i}-\bar{x}_1)^2+\sum_{i=1}^{n_1}(\bar{x}_1-\bar{x})^2 \\
&= \sum_i^{n_1}(x_{1,i}-\bar{x}_1)^2+n_1(\bar{x}_1-\bar{x})^2 \\
&= n_1\sigma_{x1}^2+n_1(\bar{x}_1-\bar{x})^2
\end{aligned}$$

## B.2 Example of the radar clustering algorithm

To illustrate how the clustering algorithm works, an example will be discussed in this Appendix. The situation that will be used for this example is shown in Figure B.2.

The following inputs are used for this clustering algorithm example:

- $\phi_r$ = [21.9 18 14 3.1 -0.1 -3 -13 -14.7 -18] in °
  $r_r$ = [11 10.8 10.5 11 10.8 10.9 11 10.2 10.7] in m
  $\dot{r}_r$ = [13.5 13.49 13.51 0 0.01 -0.01 3 3.01 2.99 ] in m/s
- $R_r = \begin{bmatrix} \sigma_{\phi r}^2 & 0 & 0 \\ 0 & \sigma_{rr}^2 & 0 \\ 0 & 0 & \sigma_{\dot{r}r}^2 \end{bmatrix} = \begin{bmatrix} 0.0175^2 & 0 & 0 \\ 0 & 0.55^2 & 0 \\ 0 & 0 & 0.2778^2 \end{bmatrix}$
- $d_{threshold} = 1.8$
  $v_{r,threshold} = 0.5$

First the Euclidean distances are calculated by (4.1), which results in the following distance matrix:

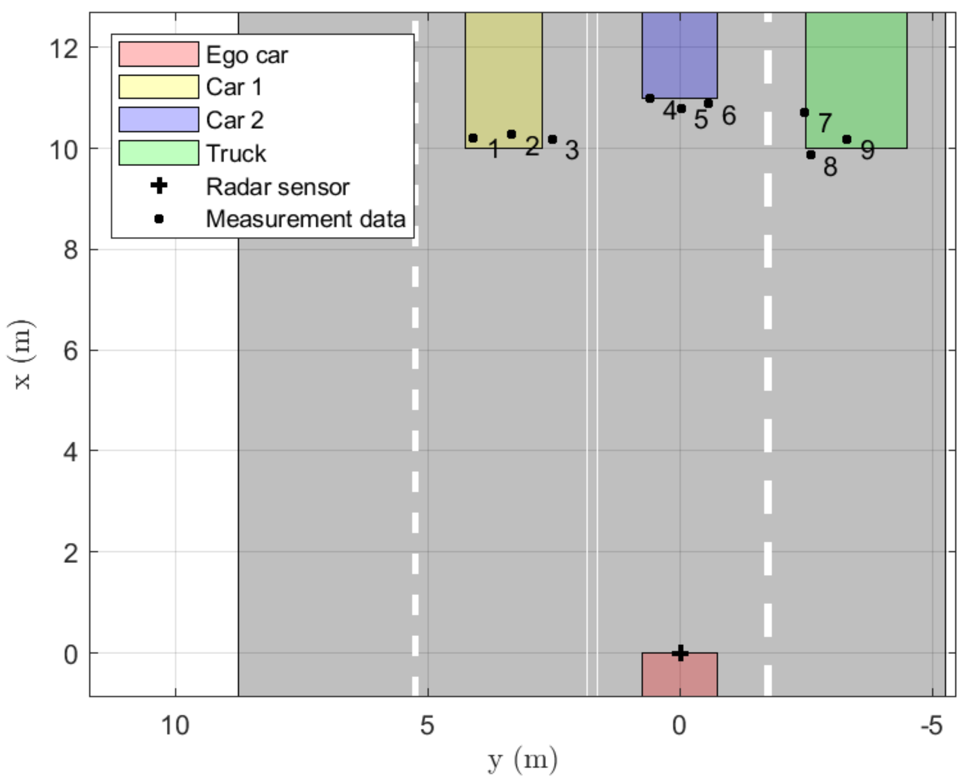


Figure B.2: Overview of the situation in the $xy$-plane

$$D = \begin{bmatrix} 0 & 0.7683 & 1.5628 & 3.5932 & 4.1643 & 4.7224 & 6.5972 & 6.6998 & 7.4094 \\ 0 & 0 & 0.8015 & 2.8336 & 3.39557 & 3.9557 & 5.8290 & 5.9336 & 6.6445 \\ 0 & 0 & 0 & 2.1018 & 2.6312 & 3.1878 & 5.0426 & 5.1386 & 5.8467 \\ 0 & 0 & 0 & 0 & 0.6407 & 1.1695 & 3.0808 & 3.3737 & 3.9841 \\ 0 & 0 & 0 & 0 & 0 & 0.5581 & 2.4570 & 2.7339 & 3.3463 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1.9113 & 2.2605 & 2.863 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.8595 & 0.9929 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.7823 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \leq 1.8$$

Then the velocity differences between the detections are calculated by (4.5), which results in the following velocity matrix:

$$V_r = \begin{bmatrix} 0 & 0.01 & 0.01 & 13.5 & 13.49 & 13.51 & 10.5 & 10.49 & 10.51 \\ 0 & 0 & 0.02 & 13.49 & 13.48 & 13.5 & 10.49 & 10.48 & 10.5 \\ 0 & 0 & 0 & 13.51 & 13.5 & 13.52 & 10.51 & 10.5 & 10.52 \\ 0 & 0 & 0 & 0 & 0.01 & 00.01 & 3 & 3.01 & 2.98 \\ 0 & 0 & 0 & 0 & 0 & 0.02 & 2.99 & 3 & 2.98 \\ 0 & 0 & 0 & 0 & 0 & 0 & 3.01 & 3.02 & 3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.01 & 0.01 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.02 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \leq 0.5$$

In Table B.1 it can be seen which cluster are obtained by each method:

| | Distance based | Velocity based | Distance and velocity based |
|---|---|---|---|
| Cluster 1: | 1,2,3 | 1,2,3 | 1,2,3 |
| Cluster 2: | 4,5,6 | 4,5,6 | 4,5,6 |
| Cluster 3: | 7,8,9 | 7,8,9 | 7,8,9 |

Table B.1: Obtained clusters

So, based on position and velocity three clusters are obtained. Where cluster 1 consists of the points: 1,2,3, cluster 2 of points 4,5,6 and cluster 3 of points 7,8,9.

Now for each cluster the new core point in Cartesian and spherical coordinates is determined by (4.9) and (4.10). The resulting coordinates can be seen in Table B.2.

| | Cartesian coordinates $[\bar{x}_r, \bar{y}_r]$ | Spherical coordinates $[\bar{\phi}_r, \bar{r}_r, \bar{\dot{r}}_r]$ |
|---|---|---|
| Cluster 1: | [10.2219, 3,3268] | [0.3146, 10.7497 13.5] |
| Cluster 2: | [10.8896, 0.0019] | [1.75005e-4, 10.8896, 0] |
| Cluster 3: | [10.2535, -2.7898] | [-0.2656, 10.6262, 3] |

Table B.2: Obtained coordinates of each cluster in Cartesian and spherical coordinates

In Figure B.3 the obtained clusters for this example can be seen:

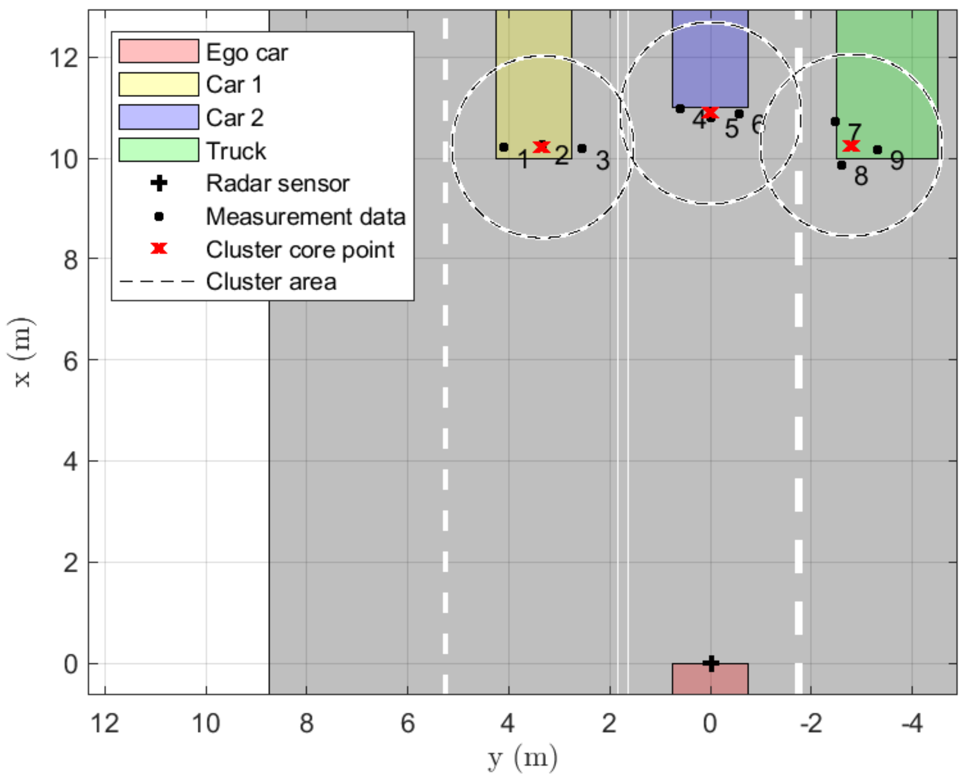


Figure B.3: Top view of obtained clusters

The last step is to calculate the measurement noise covariance matrix for each cluster, which can be done by (4.12). In Table B.3 the results of the new variances of each cluster are shown.

| | $\bar{\sigma}^2_{\phi r}$ | $\bar{\sigma}^2_{rr}$ | $\bar{\sigma}^2_{\dot{r}r}$ |
|---|---|---|---|
| Original: | 0.0003 | 0.3025 | 0.0772 |
| Cluster 1: | 0.0035 | 0.3450 | 0.0772 |
| Cluster 2: | 0.0022 | 0.3093 | 0.0772 |
| Cluster 3: | 0.0016 | 0.4114 | 0.0772 |

Table B.3: Obtained variances for each cluster

**Plotting of the measurement noise covariance matrix**
To show how the measurement noise covariance matrices of the clusters are changed compared to the original measurement data, first the measurement noise covariance matrices needs to be transformed from the spherical to the Cartesian coordinate system. This can be done by the following matrix transformation:

$$R_{cart} = JR_{sph}J^T \tag{B.2}$$

where $R_{cart}$ is the measurement noise covariance matrix in Cartesian coordinates, $J$ the Jacobian of the transformation matrix obtained from (4.2) and $R_{sph}$ the measurement noise covariance matrix in spherical coordinates.
By eigendecompostion of a matrix its eigenvalues and eigenvector can be determined. The eigenvalues and eigenvector of the measurement noise covariance matrix can be obtained by the following equation:

$$R_{cart} = V\Lambda V^{-1} \tag{B.3}$$

where $V$ is the matrix whose columns consists of the eigenvectors of $R_{cart}$ and $\Lambda$ is a diagonal matrix whose non-zero elements are the corresponding eigenvalues. In the 3D situation, three eigenvalues and three eigenvectors will be obtained. The eigenvectors of the covariance matrix will represent the directions of the largest variance of the data, while the eigenvalues represent the magnitude of this variances in those directions, see Figure B.4. So this means matrix $V$ represents a rotation matrix and $\Lambda$ represent a scaling matrix.

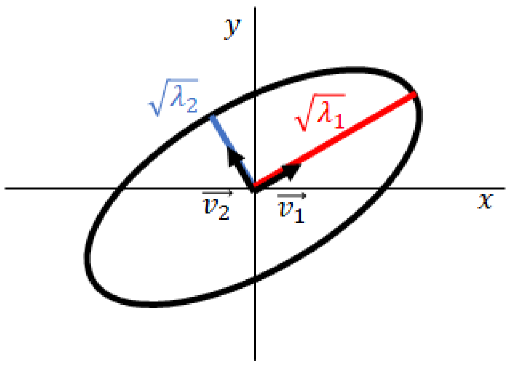


Figure B.4: 2D ellipse of a covariance matrix

By applying this transformation to a unit sphere, the measurement noise error ellipse represented by $x_{ellipse}$ and $y_{ellipse}$ can be obtained by the following equation:

$$\begin{bmatrix} x_{ellipse} \\ y_{ellipse} \end{bmatrix} = \begin{bmatrix} v_1 & v_2 \end{bmatrix} \begin{bmatrix} \sqrt{\lambda_1}\ cos(\phi) \\ \sqrt{\lambda_2}\ cos(\phi) \end{bmatrix} \qquad -\pi/2 \leq \phi \leq \pi/2 \tag{B.4}$$

where $v_1$ and $v_2$ are the eigenvectors and $\lambda_1$ and $\lambda_2$ are the eigenvalues.
By applying this transformation to the measurement noise covariance matrix of the original measurement data and the measurement noise covariance matrix of the clusters, it can be illustrated how the measurement noise

is changed for each cluster as can be seen in Figure B.5. From these figures it can be concluded that the size and the direction of each ellipse is different depending on how the detections in that cluster are spaced.

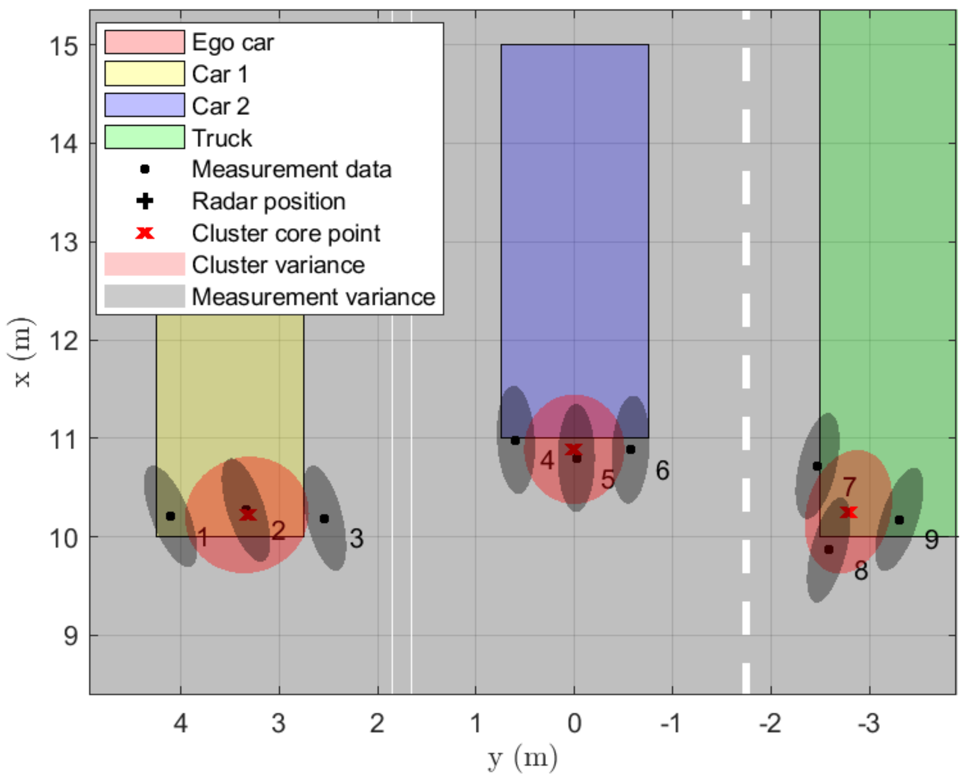


Figure B.5: Overview of the clustering algorithm in the $xy$-plane

# Appendix C

# Initialization of the Kalman filter

Before the Kalman filter can start predicting and updating, first the states and state covariances need to be initialized. This is done by using the measurement and measurement noise. Since the measurements are in spherical coordinates and the states are in Cartesian coordinates, the measurements that will be used initialization first needs to be translated from spherical to Cartesian coordinates.

The initial positions $x_0$ and $y_0$ are obtained by applying the transformation from spherical to Cartesian coordinates and correcting it with the sensor offset by the following the equation:

$$\begin{bmatrix} x_0 \\ y_0 \end{bmatrix} = \begin{bmatrix} r\cos(\phi) \\ r\sin(\phi) \end{bmatrix} + \begin{bmatrix} x^i_{offset} \\ y^i_{offset} \end{bmatrix} \tag{C.1}$$

The initial linear velocities $v_{x,0}$ and $v_{y,0}$ are obtained by taking the derivative of (C.1) by the following equation:

$$\begin{bmatrix} v_{x,0} \\ v_{y,0} \end{bmatrix} = \begin{bmatrix} \dot{x}_0 \\ \dot{y}_0 \end{bmatrix} = \begin{bmatrix} \frac{x_0}{\sqrt{x_0^2+y_0^2}} \cdot \dot{r} \\ \frac{y_0}{\sqrt{x_0^2+y_0^2}} \cdot \dot{r} \end{bmatrix} \tag{C.2}$$

For the initial velocities it is assumed that $\dot{\phi}_0$ is equal to zero for the initialization, since these rate of change of both angles is not measured and

Besides the positions and velocities also the state covariance needs to be initialized. This is done as follows:

$$\begin{bmatrix} \sigma^2_{x,0} & 0 & 0 & 0 \\ 0 & \sigma^2_{v_{x,0}} & 0 & 0 \\ 0 & 0 & \sigma^2_{y,0} & 0 \\ 0 & 0 & 0 & \sigma^2_{v_{y,0}} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \sigma^2_{x,0} & 0 \\ 0 & \sigma^2_{y,0} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \sigma^2_{v_{x,0}} & 0 \\ 0 & \sigma^2_{v_{y,0}} \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{C.3}$$

The position covariance is calculated as follows:

$$\begin{bmatrix} \sigma^2_{x,0} & 0 \\ 0 & \sigma^2_{y,0} \end{bmatrix} = J \begin{bmatrix} \sigma^2_{\phi} & 0 \\ 0 & \sigma^2_{r} \end{bmatrix} J^T \tag{C.4}$$

where the Jacobian $J$ can be calculated by:

$$J = \begin{bmatrix} \frac{\partial x}{\partial \phi} & \frac{\partial x}{\partial r} \\ \frac{\partial y}{\partial \phi} & \frac{\partial y}{\partial r} \end{bmatrix} \tag{C.5}$$

The velocity covariance can be obtained as follows:

$$\begin{bmatrix} \sigma^2_{v_{x,0}} & 0 \\ 0 & \sigma^2_{v_{y,0}} \end{bmatrix} = R_{rot} \begin{bmatrix} \sigma^2_{\dot{r}} & 0 \\ 0 & \sigma^2_L \end{bmatrix} {R_{rot}}^T \tag{C.6}$$

where $R_{rot}$ is a rotation matrix that is used to rotate the variances to frame of the sensor and is given by:

$$R_{rot} = \begin{bmatrix} \cos(\phi) & -\sin(\phi) \\ \sin(\phi) & \cos(\phi) \end{bmatrix} \tag{C.7}$$

The obtained initial states and covariances will be used as a starting point for the Kalman filter.

# Appendix D

# Sensor parameter settings

| | |
|---|---|
| Unique identifier of the sensor | 1 |
| Required time interval between sensor updates | 0.07 s |
| Sensor's (x,y) position | (3.7,0) m |
| Sensor's height | 0.5 m |
| Yaw angle of sensor mounted on ego vehicle | 0° |
| Pitch angle of sensor mounted on ego vehicle | 0° |
| Roll angle of sensor mounted on ego vehicle | 0° |
| Maximum number of reported detections | 30 |

(a) Radar parameters

| | |
|---|---|
| Azimuthal resolution of radar | 6° |
| Elevation resolution of radar | 10° |
| Range resolution of radar | 3 m |
| Range rate resolution of radar | 0.5 m/s |
| Fraction azimuthal bias component of radar | 0.1 |
| Fractional elevation bias component or radar | 0.1 |
| Fractional range bias component of radar | 0.05 |
| Fractional range rate bias component of radar | 0.05 |
| Total angular field of view for radar | (120°, 20°) |
| Maximum detection range | 70 m |
| Minimum and maximum range rates that can be reported | (-100,100) m/s |
| Detection probability | 0.9 |
| Rate at which false alarms are reported | $1{\cdot}10^{-6}$ |
| Range where detection probability is achieved | 100 m |
| Radar cross section at which detection probability is achieved | 0 dBsm |
| Enable elevation angle measurements | true |
| Enable range rate measurements | true |
| Add noise to measurements | true |
| Enable false detections | true |
| Enable occlusion | true |

(b) Radar measurements

Table D.1: Parameters radar sensor generator

| | |
|---|---|
| Unique identifier of sensor | 2 |
| Types of detections generated by sensor | Object only |
| Required interval between sensor updates | 0.1 s |
| Sensor's (x,y) position | (1.9,0) m |
| Sensor's height | 1.3 m |
| Yaw angle of sensor mounted on ego vehicle | 0° |
| Pitch angle of sensor mounted on ego vehicle | 0° |
| Roll angle of sensor mounted on ego vehicle | 0° |
| Maximum number of reported detections | 15 |

(a) Camera parameters

| | |
|---|---|
| Maximum detection range | 100 m |
| Bounding box accuracy | 5 pixels |
| Smoothing filter noise intensity | 5 $m/s^2$ |
| Maximum detectable object speed | 50 m/s |
| Maximum allowed occlusion for detector | 0.5 |
| Minimum detectable image size of an object | (15,15) pixels |
| Probability of detecting a target | 0.9 |
| Number of false positives per image | 0.1 |
| Add noise to measurement | true |

(b) Camera measurements

| | |
|---|---|
| Focal length | (800,800) pixels |
| Optical center of the camera | (320,240) pixels |
| Image size produced by the camera | (480,640) pixels |
| Radial distortion coefficients | (0,0) |
| Tangential distortion coefficients | (0,0) |
| Skew of the camera axes | 0 |

(c) Camera intrinsics

Table D.2: Parameters camera sensor generator

# Appendix E

# Test results

## E.1 Scenario 1

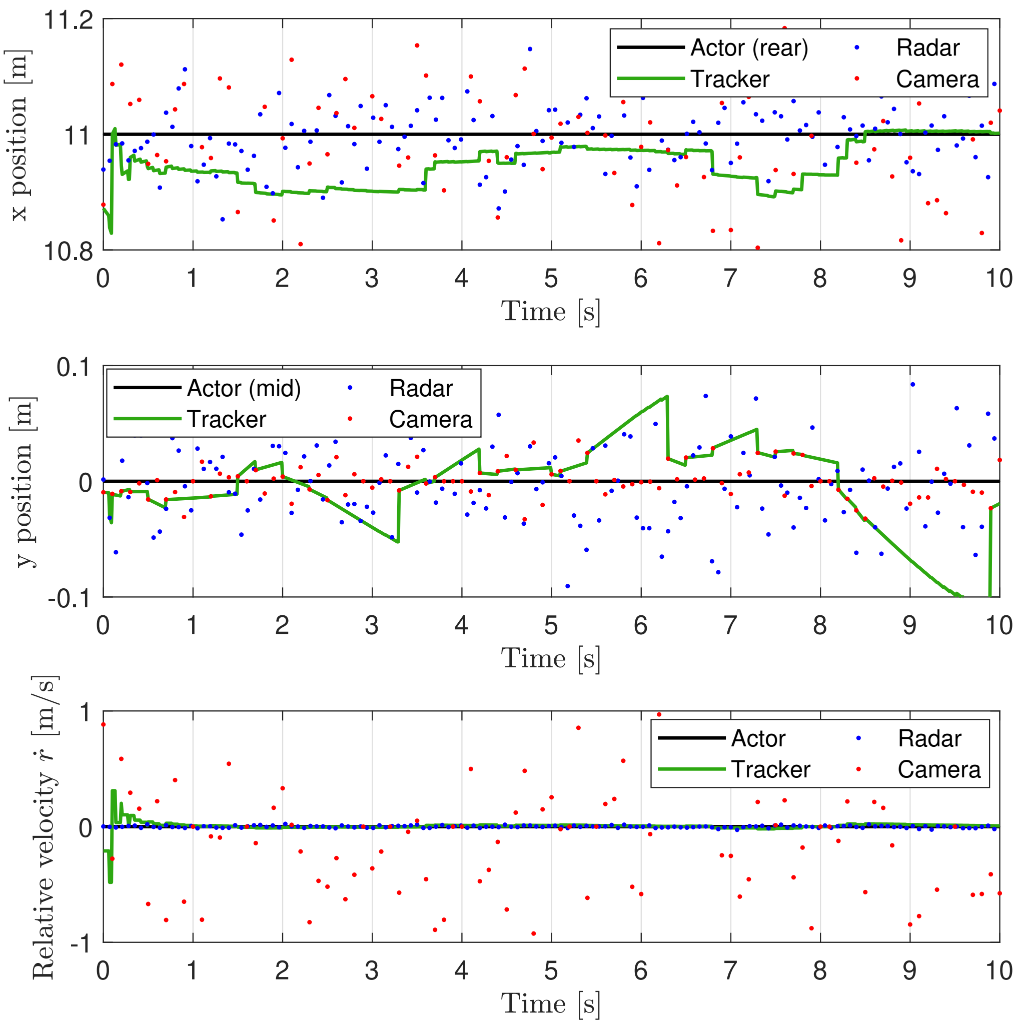


Figure E.1: Measurement and tracker results for scenario 1

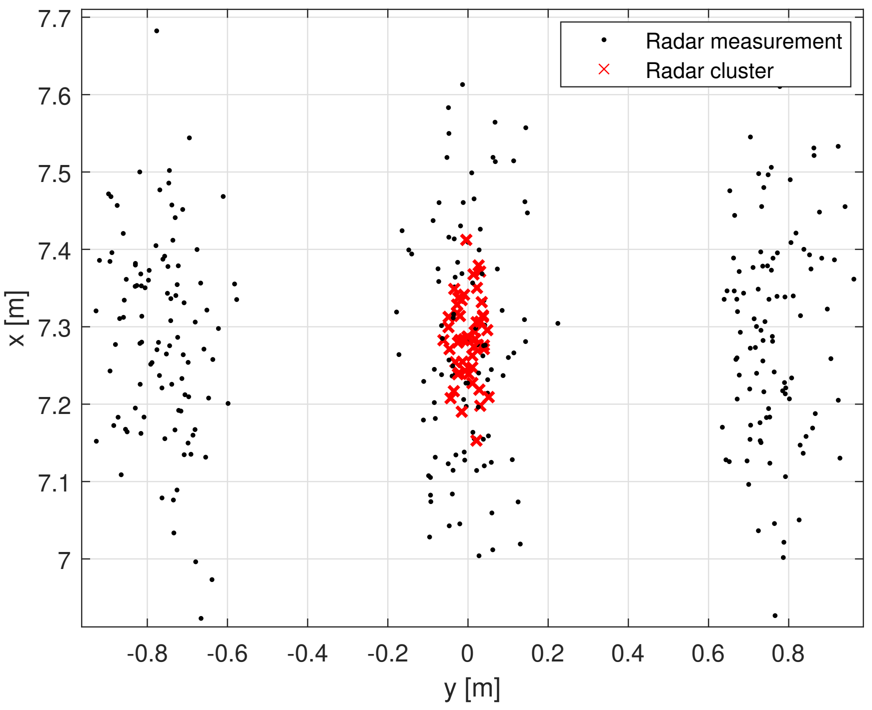


Figure E.2: Obtained clusters for scenario 1

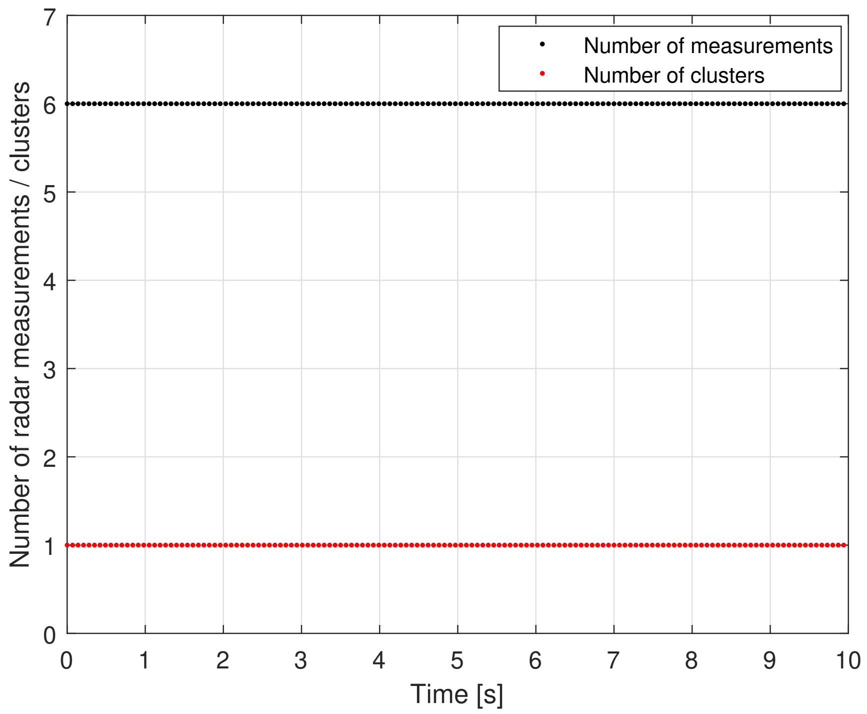


Figure E.3: Number of obtained clusters per time step for scenario 1

## E.2 Scenario 2

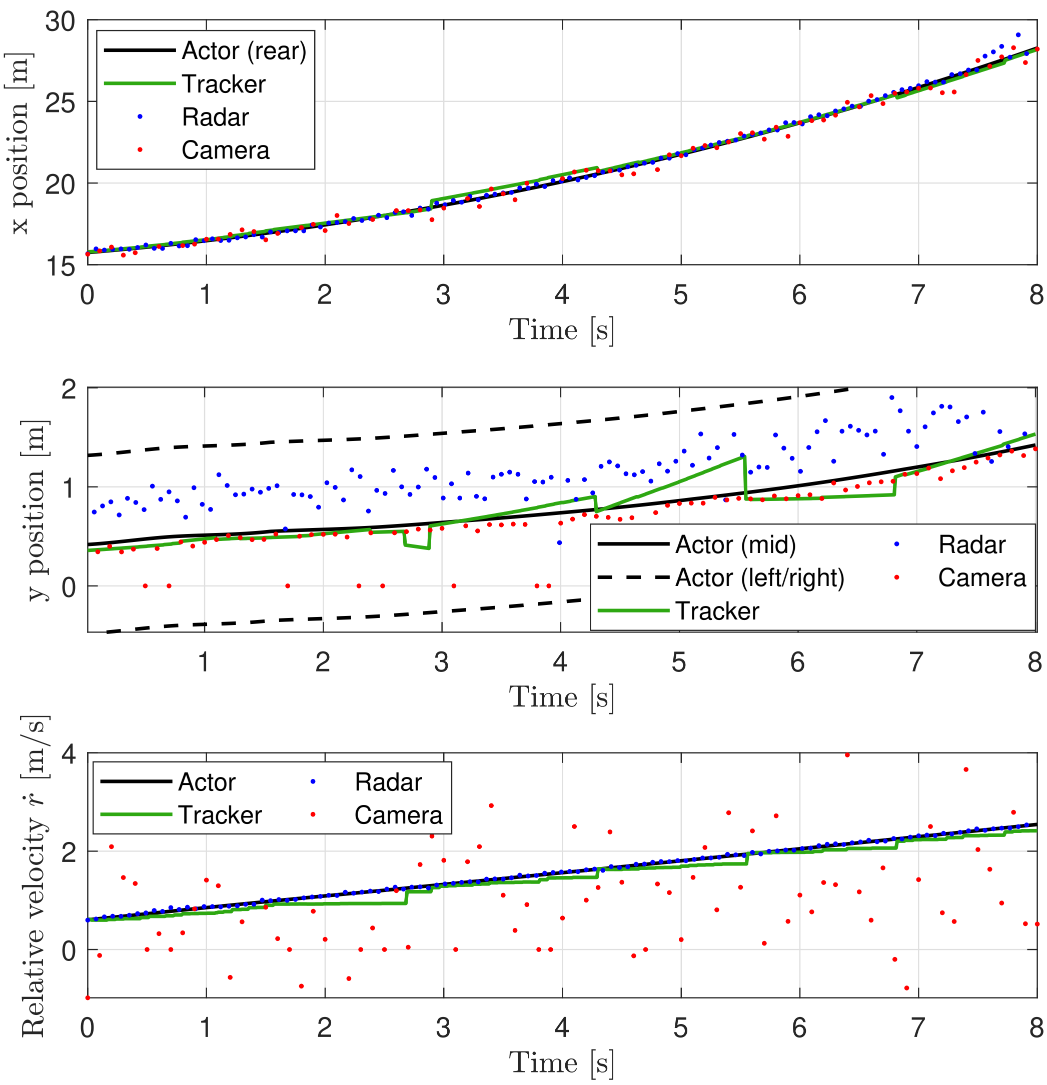


Figure E.4: Measurement and tracker results for scenario 2

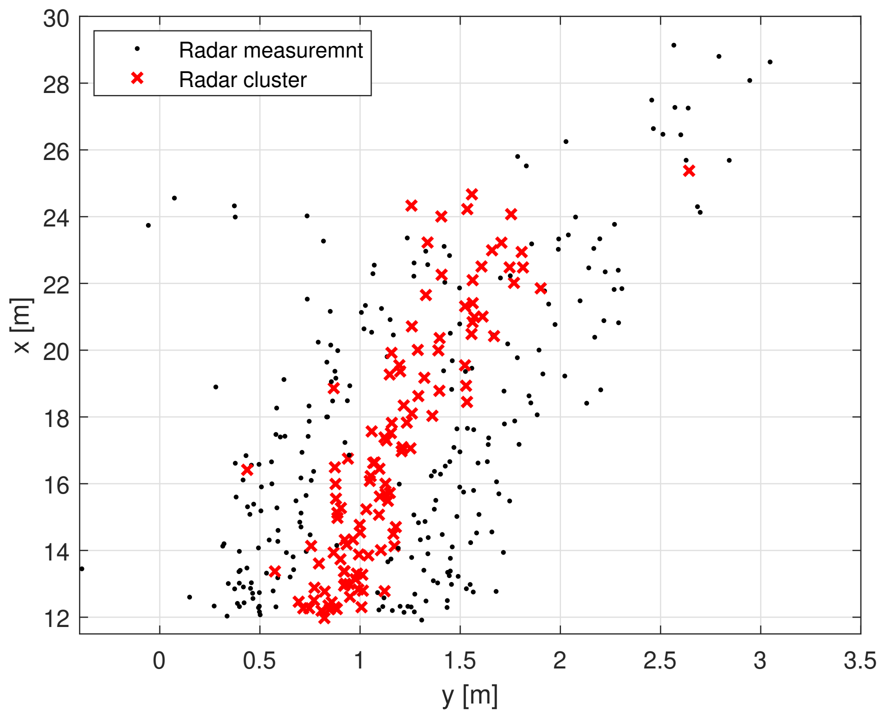


Figure E.5: Obtained clusters for scenario 2

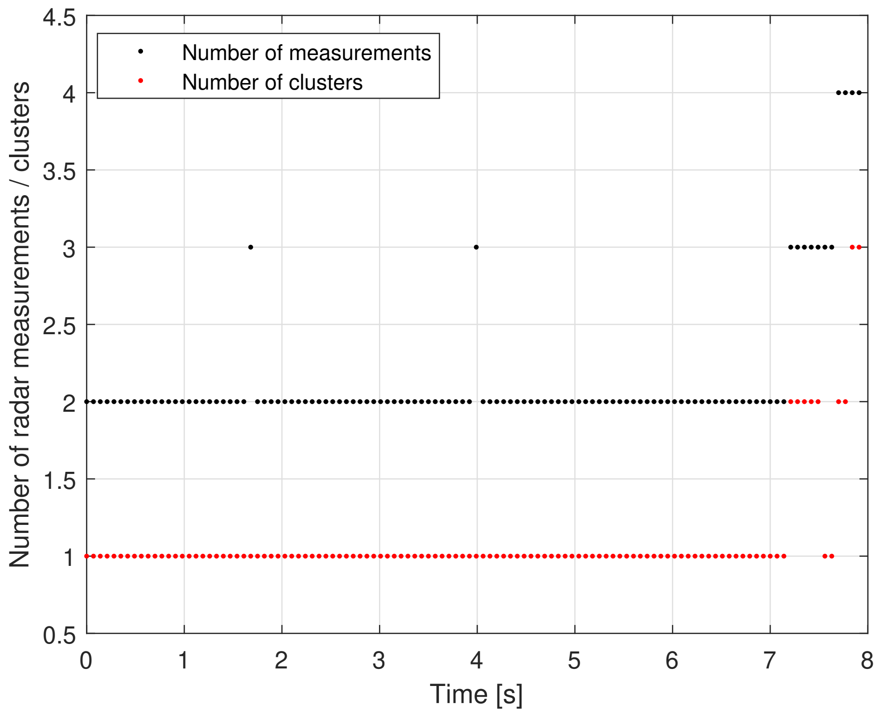


Figure E.6: Number of obtained clusters per time step for scenario 2

## E.3 Scenario 3

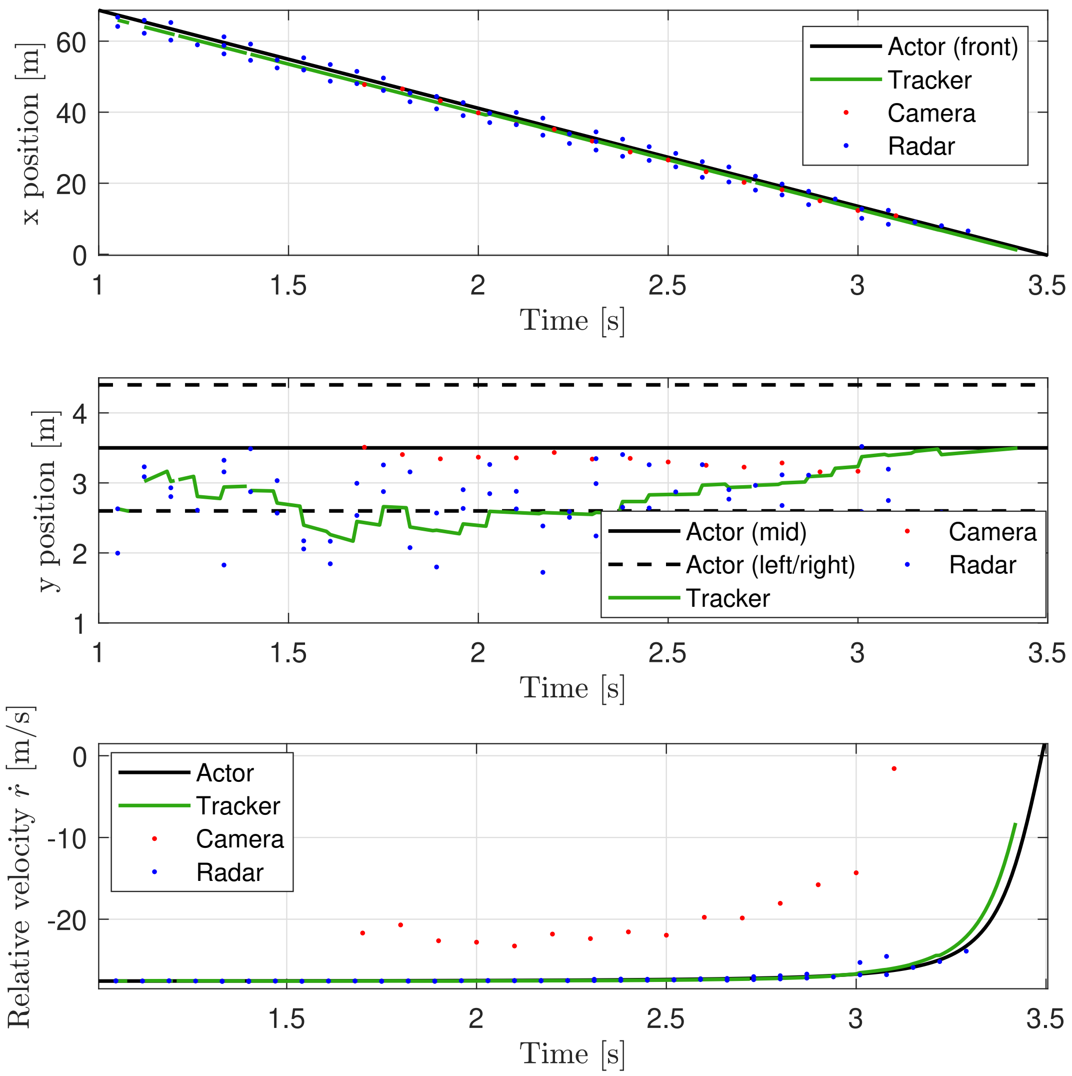


Figure E.7: Measurement and tracker results for scenario 3, actor 1

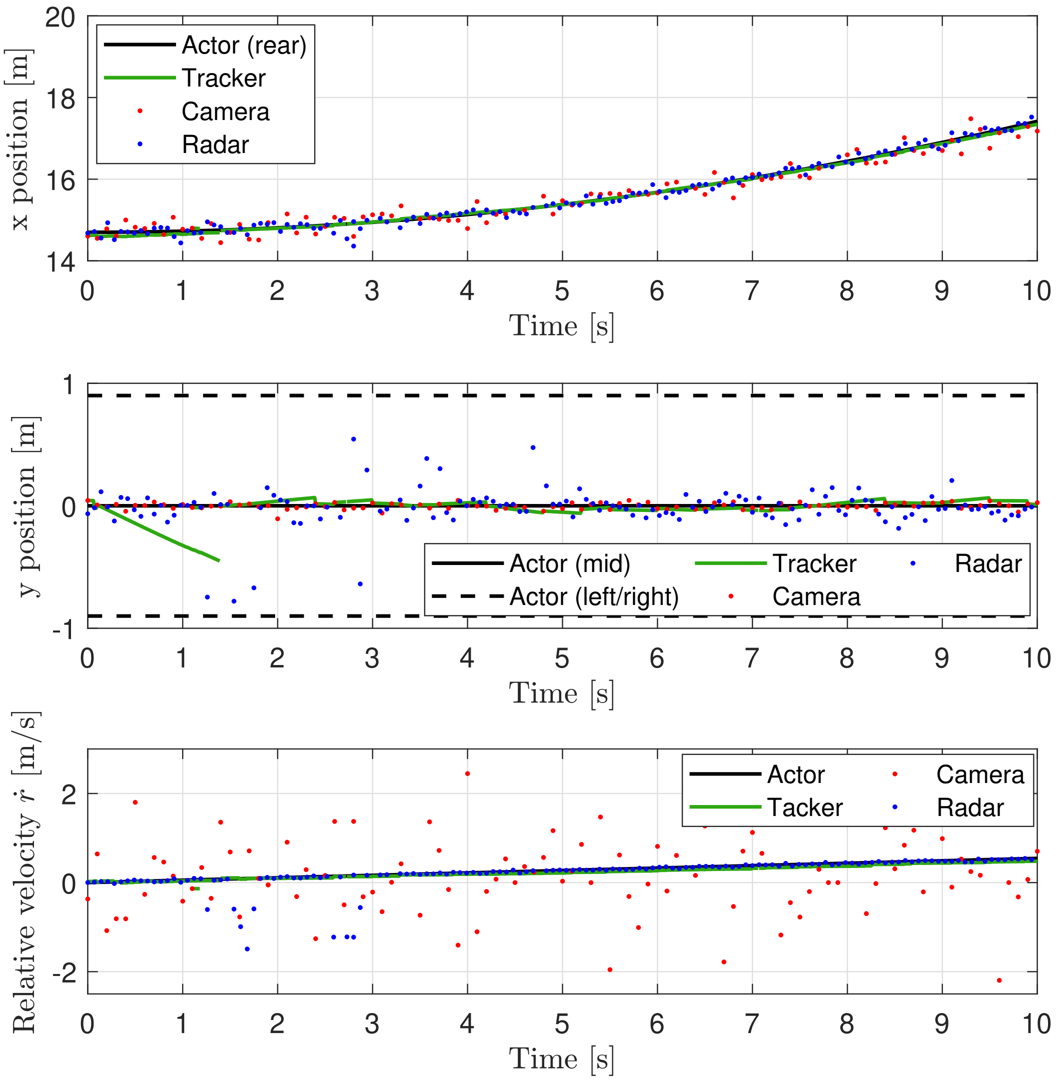


Figure E.8: Measurement and tracker results for scenario 3, actor 2

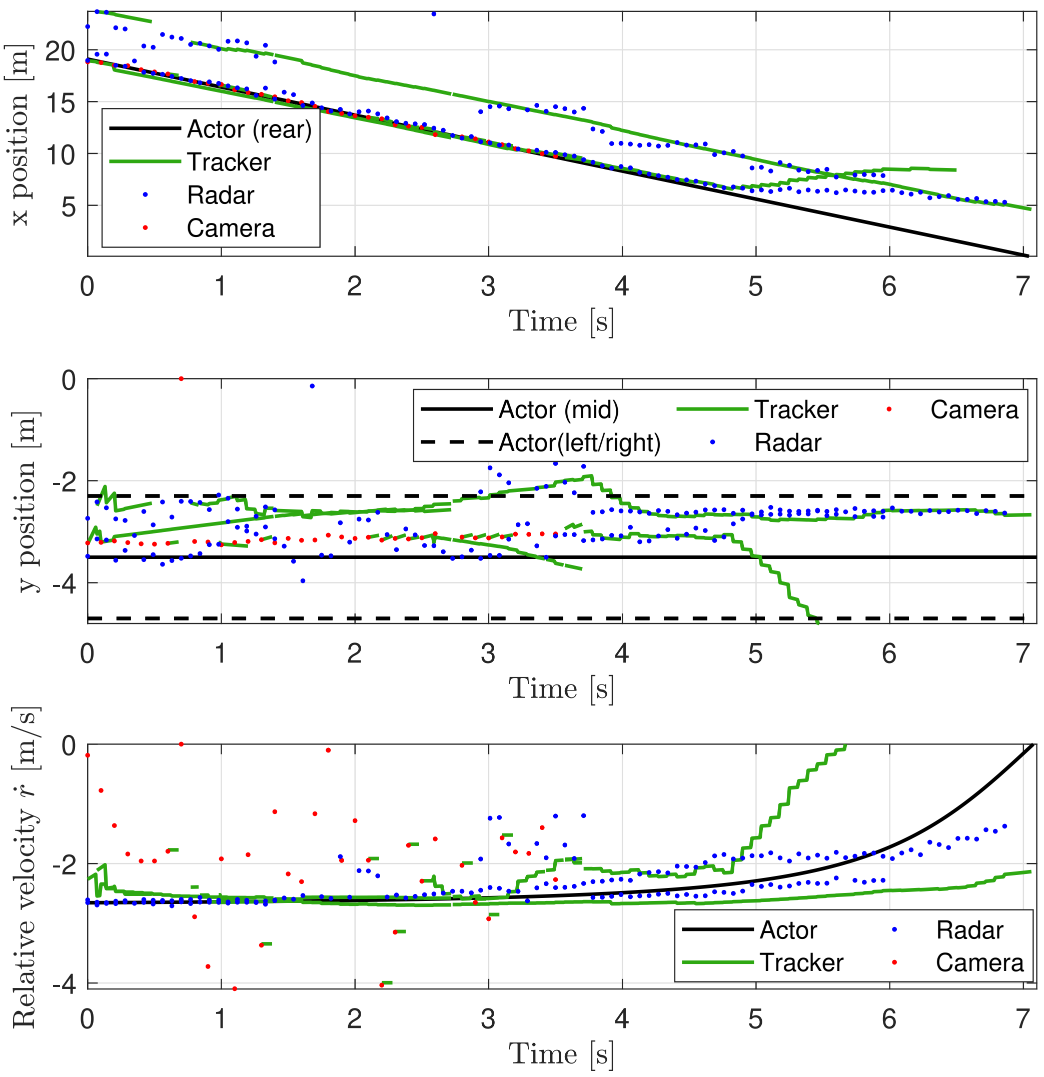


Figure E.9: Measurement and tracker results for scenario 3, actor 3

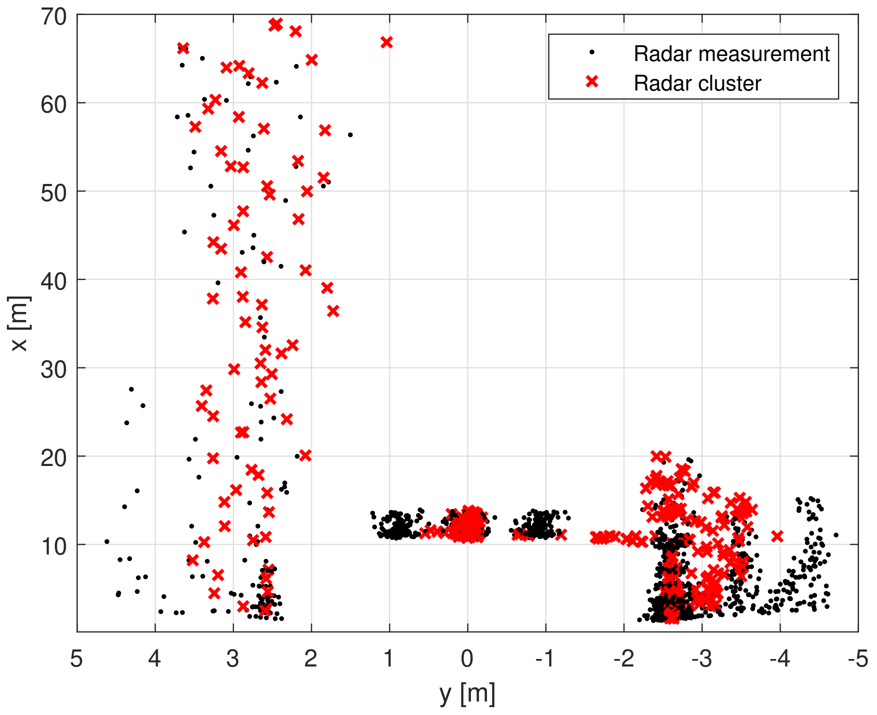


Figure E.10: Obtained clusters for scenario 3

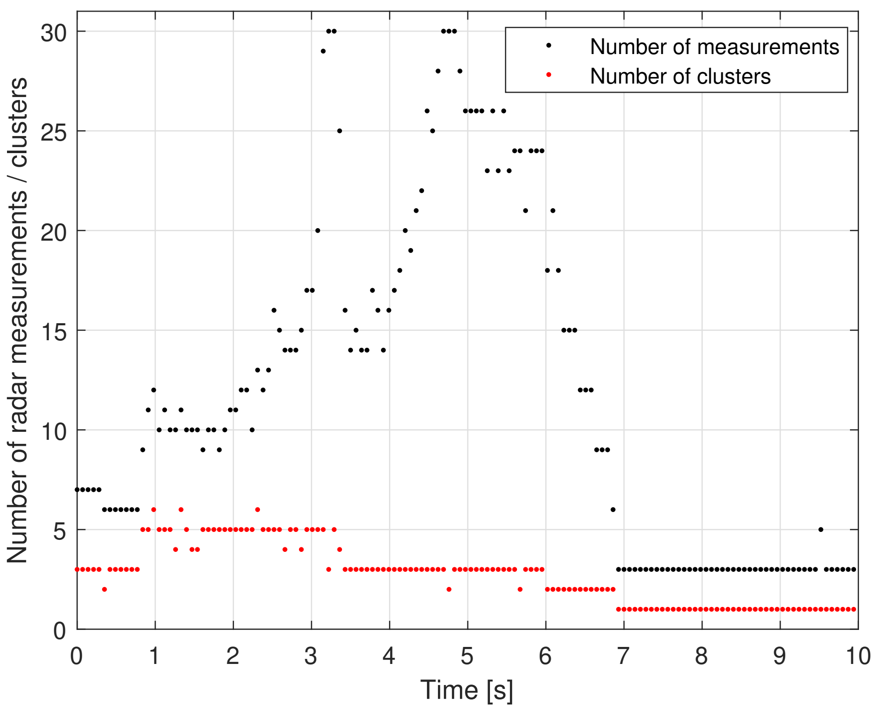


Figure E.11: Number of obtained clusters per time step for scenario 3

## E.4 Scenario 4

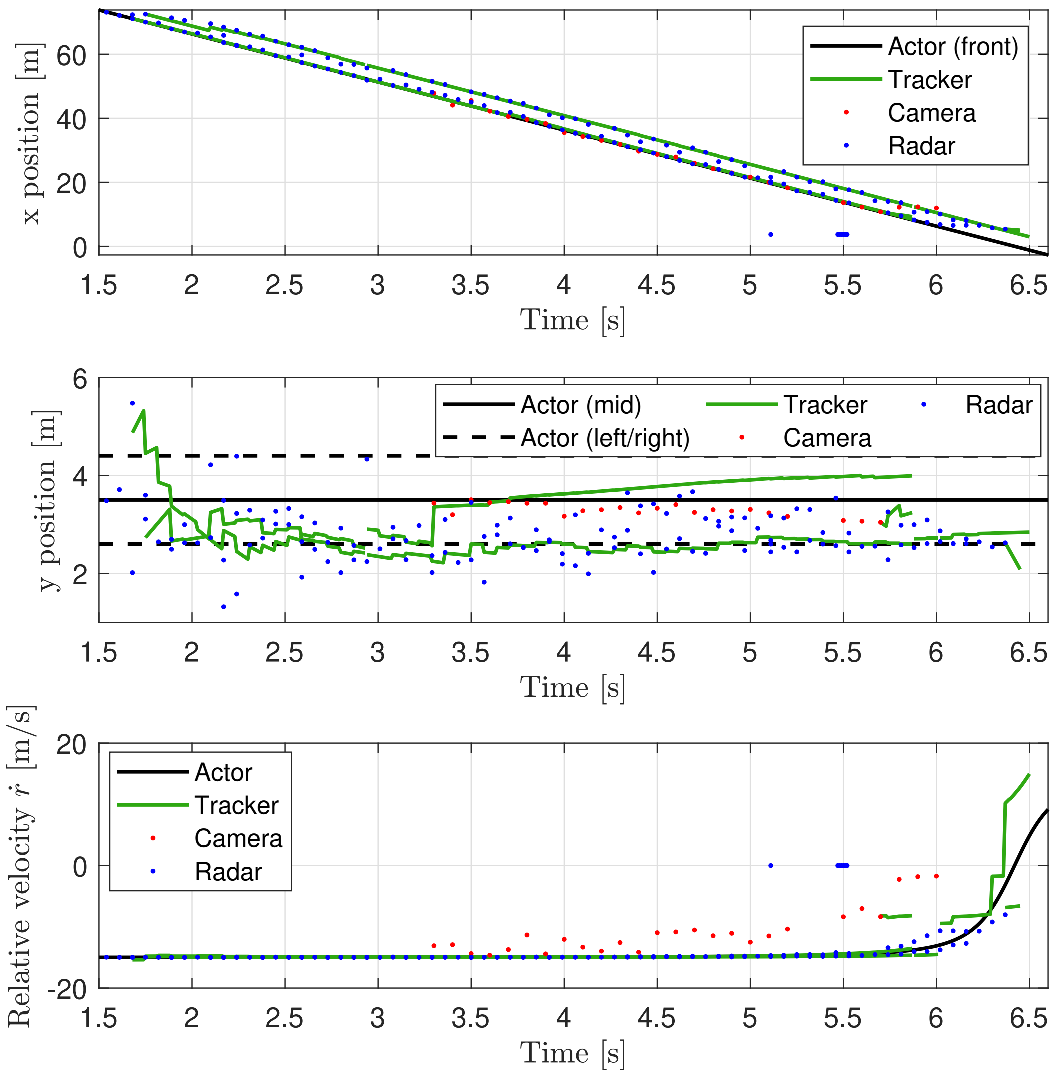


Figure E.12: Measurement and tracker results for scenario 4, actor 1

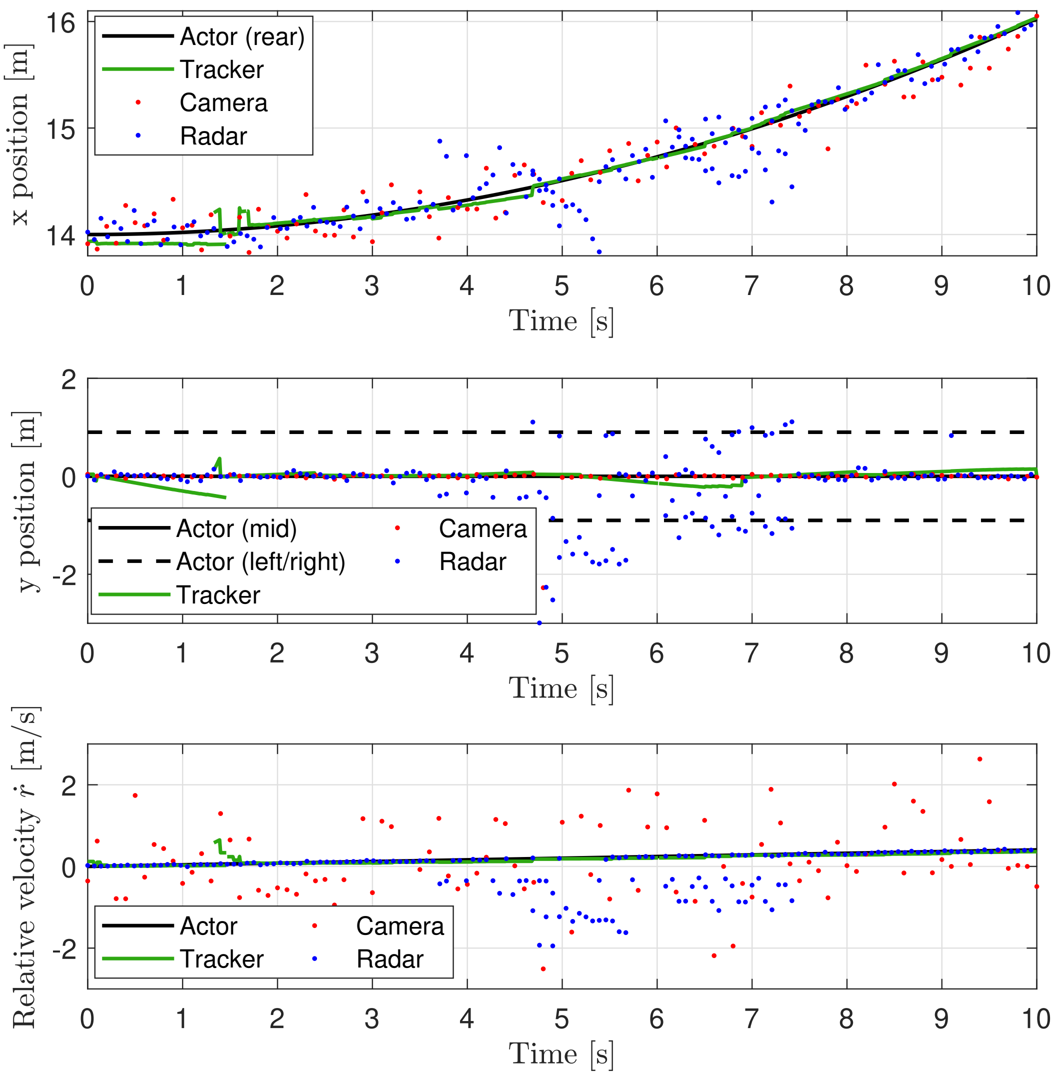


Figure E.13: Measurement and tracker results for scenario 4, actor 2

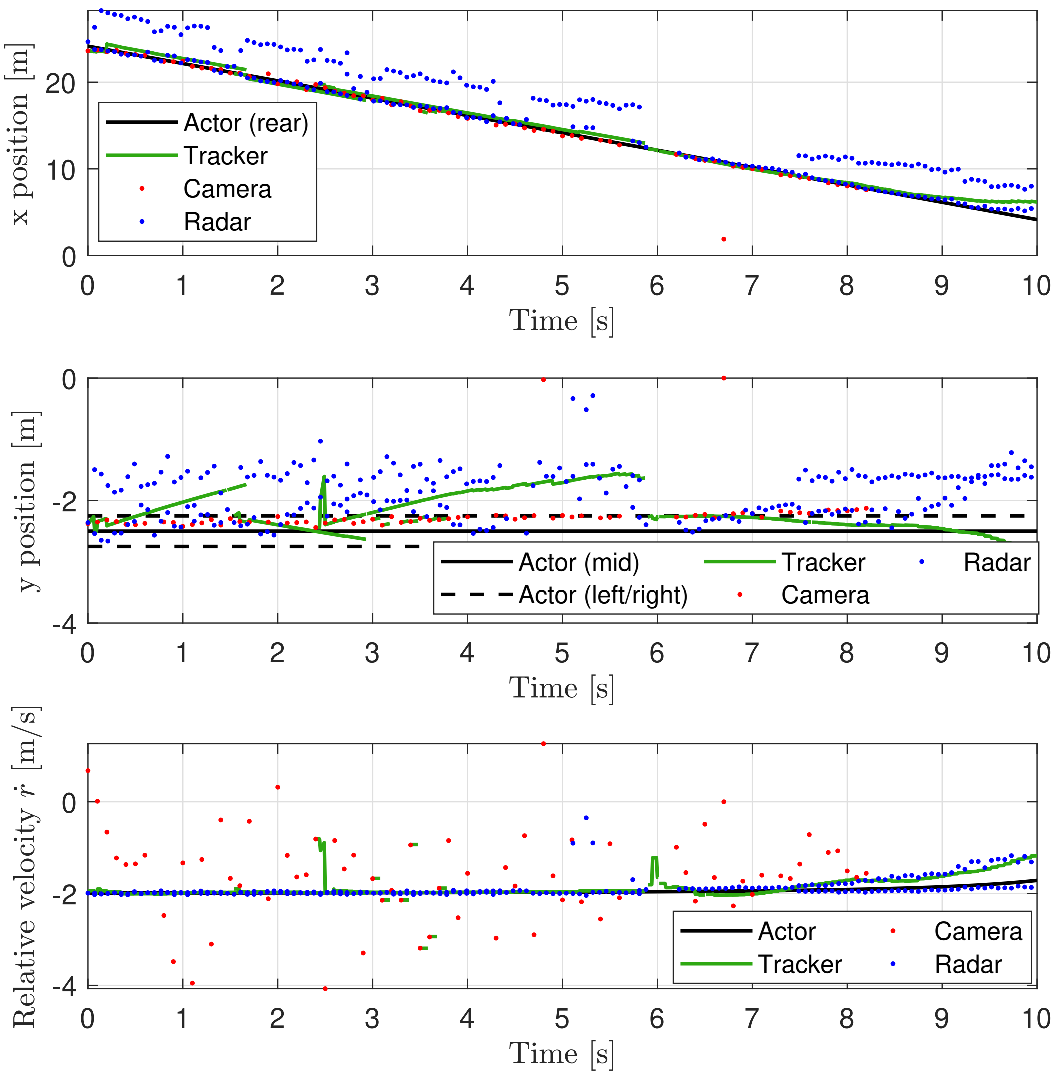


Figure E.14: Measurement and tracker results for scenario 4, actor 3

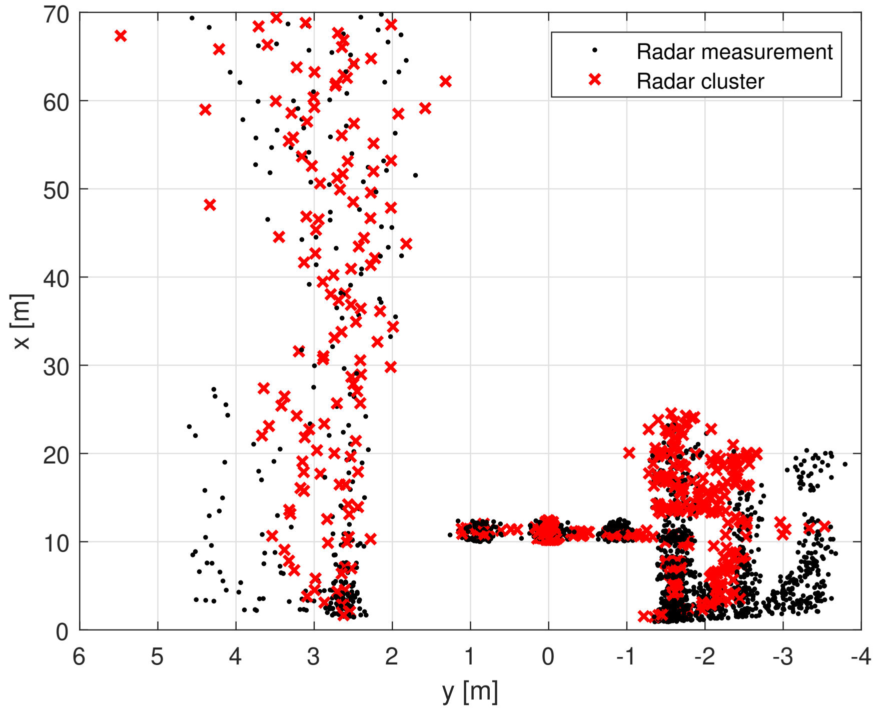


Figure E.15: Obtained clusters for scenario 4

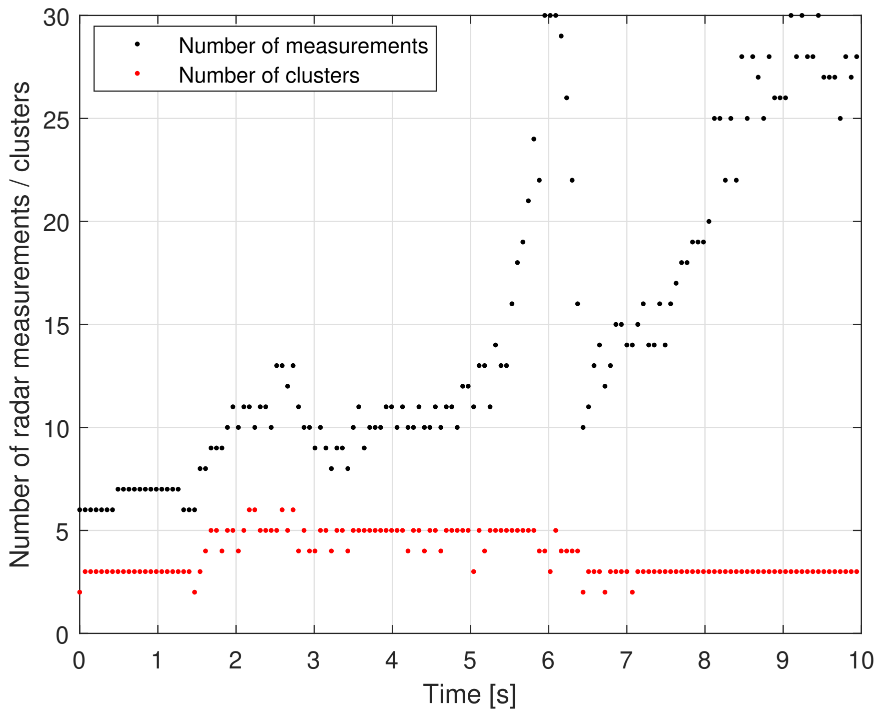


Figure E.16: Number of obtained clusters per time step for scenario 4