Mosser Alexander, BSc

Vehicle Environment Model for Autonomous Driving

MASTER'S THESIS

to achieve the university degree
Diplom-Ingenieur
Master's degree program: Electrical Engineering

submitted to
Graz University of Technology

Tutor
Univ.-Doz. Dipl.-Ing. Dr. techn. Daniel Watzenig
Institute of Electrical Measurement and Signal Processing [EMT]

Supervisor
Dipl.-Ing. Dr. techn. Michael Stolz
VIRTUAL VEHICLE Research Center

Graz, January 2016

Kurzfassung

Moderne Fahrerassistenzsysteme unterstützen den Lenker eines Fahrzeuges in schwierigen Verkehrssituationen, wo die menschliche Aufnahmefähigkeit nicht mehr ausreicht um das Szenario vollständig zu erfassen. Solche Methoden sollen durch ihren Einsatz die Verkehrssicherheit und den Fahrkomfort erhöhen. Einer der Kernbestandteile dieser Systeme ist die Abbildung der Fahrzeugumgebung, ein sogenanntes Umfeldmodell. Zurzeit verwenden die meisten Funktionen nur einen auf die Anwendung zugeschnitten Sensor, wie z.B. der Abstandsregeltempomat, der einen Radarsensor im Frontbereich benützt. Moderne Kraftfahrzeuge haben aber heutzutage eine Vielzahl an Sensoren in und an der Karosserie verbaut. Ziel ist es nun die Information von mehreren, die Umgebung erfassenden Sensoren zu bündeln und zu einem funktionsunabhängigen Modell zusammen zuführen. Das Modell kann als Basis für die unterschiedlichsten Assistenzsysteme eingesetzt werden. Die vorliegende Masterarbeit beschäftigt sich mit der Modellierung einer Umfelderfassung für Fahrzeuge.

Soll sich das Fahrzeug vorrangig autonom in einem zuvor unbekannten und undefinierten Raum bewegen, muss eine genaue Kenntnis über den Aufenthaltsort von statischen und dynamischen Objekten vorliegen. Da sich die Umgebung ständig ändert, muss sich auch das Modell dynamisch auf die immer wieder neuen Situationen einstellen können. Im Rahmen dieser Arbeit wurde ein Algorithmus entwickelt welcher ein umfassendes Abbild der Fahrzeugumgebung wiedergibt. Es sind sowohl die Anzahl, wie auch die Anordnung und die Auflösung der Sensoren einstellbar. Des Weiteren verfügt er über ein Trackingverfahren, welches über eine vorgebbare Zeit die Position der beobachteten Objekte bestimmen kann, selbst wenn keine aktuellen Messwerte vorliegen. Am Ende wird mit ausgewählten Beispielen die Funktions- und Anwendungsfähigkeit des Modelles validiert und die Ergebnisse diskutiert. Der vorgestellte Algorithmus eignet sich als Entwicklungs- und Simulationsumgebung für Fahrerassistenzsysteme. Mit dessen Hilfe die Reichweite und der Erfassungsbereich von unterschiedlichen Sensornetzwerken getestet und evaluiert werden kann. Mit dem vorgestellten Umgebungsmodell soll die Entwicklungszeit und die Prüfkosten von Assistenzsystemen verkürzt bzw. reduziert werden.

Abstract

Advanced driver assistance systems support the driver in difficult traffic situations where the limits of the human capacity show as the driver is no longer able to fully grasp the scenario. The use of such methods increases the traffic safety and driving comfort. One of the core components of these systems is the reproduction of the vehicle environment, the so-called environment model. Currently, most systems have one sensor fitted to the application, like the adaptive cruise control, which has a radar sensor that is observing the area in front of the vehicle. Nowadays modern cars have a variety of sensors, mounted in and on the vehicle body. The main purpose is to combine the information of several environment sensors to a function-independent model, which can be used as a basis for different assistance functions. In this Master thesis, an environment model for motor vehicles is developed.

If the car should be able to move primarily autonomous in a previously unknown and undefined space, a precise knowledge of the position of static and dynamic objects is required. Since the surrounding is constantly changing, the model must be able to dynamically adjust to the modified situation. In the course of this thesis, an algorithm is developed, which creates a comprehensive picture of the vehicle's surrounding. The number, placement and the resolution of the sensors are adjustable. Furthermore, it has a tracking unit which can observe the position. Over a changeable period of time, the movements of the objects are predicted, even if there are no current measurement values available. Finally, the functionality and usability of the model is validated with selected examples and the results are discussed. The proposed software is designed as a development and simulation environment for advanced driver assistance systems. The model can be used to test the range and detection area of different sensor networks and evaluate the outcomes. The presented environment model helps to save development time and reduce testing costs of assistance systems.

AFFIDAVIT

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis dissertation.

....
date (signature)

Acknowledgement

This work was accomplished at the VIRTUAL VEHICLE Research Center in Graz, Austria. The author would like to acknowledge the financial support of the COMET K2 - Competence Centers for Excellent Technologies Programme of the Austrian Federal Ministry for Transport, Innovation and Technology (bmvit), the Austrian Federal Ministry of Science, Research and Economy (bmwfw), the Province of Styria and the Styrian Business Promotion Agency (SFG).

I am heartily thankful to my supervisor, Michael Stolz from the VIRTUAL VEHICLE Research Center, whose encouragement, guidance and support from the start to the final level enabled me to develop an understanding of the project.

Furthermore I would like to bring my gratitude to Daniel Watzenig the head of the department E/E & Software from the VIRTUAL VEHICLE Research Center who gave me the opportunity to work on this thesis. I also would like to thank Markus Schratter and Georg Nestlinger, for there effort, cooperation and support. They were always available for my questions and gave generously of their time and knowledge.

I also would like to thank my parents for their moral and financial support to make this study possible. My family and friends for supporting and forcing me to finish this study.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Passenger cars are the most common products for individual mobility. One of their biggest advantages is that the driver can decide when and where he/she wants to go. However, the price for this is that the driver has to take over the vehicle guidance. During long highway rides and in traffic jams, controlling the vehicle can turn into an annoying task. Therefore, car manufacturers integrate technical systems to efficiently avoid or counteract dangerous situations. An assistance application supports the driver, guiding him in difficult situations and taking over control of the car to avoid accidents [4].

Nowadays many vehicles have integrated Driver Assistance Systems (DAS). Such applications should reduce or even eliminate the driver errors and improve efficiency in traffic and transport. What are assistance system in general? Answers can be found in [5] and [6]. There are two different types of assistance systems:

- Conventional Driver Assistance Systems
- Advanced Driver Assistance Systems.

Conventional Driver Assistance Systems support the driver in situations which are easy to measure or to detect, like the Antilock Braking System (ABS) or the Electronic Stability Program (ESP).

Advanced Driver Assistance Systems (ADAS) have sensors which observe the surrounding area of the car. With the help of these sensors and a corresponding signal processing, a detailed environment model (EnvM) can be built. For example, Adaptive Cruise Control (ACC) or Lane Departure Warning (LDW). Figure 1 gives an overview of state of the art systems.

1.1 Motivation

The basis of every ADAS is the reproduction of the environment. Depending on the complexity of the assistance function, the model is based on information of several sensors and extensive signal processing algorithms. One of the current methods, for instance, is the ACC. To realize a cruise control system, "only" a radar sensor on the front side of the vehicle is needed. A system that could execute an overtake manoeuvre autonomously will require additional sensors, which are mounted all around the vehicle body. As a consequence a program that is able to process and merge data from the sensors is needed [7]. This is called a sensor data fusion (SDF) and it results in a complex

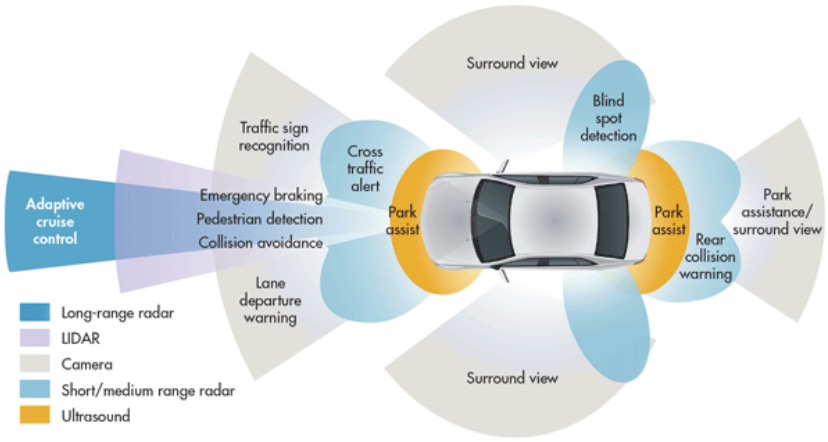

Figure 1.1: Common assistance functions are illustrated with name and detection area [1]. The used sensor types of each system are also mentioned.

EnvM.

Taking into account the complexity of DAS and the traffic scenarios, testing on the road is not economically efficient any more. Therefore virtual test environments are developed. Even by the same execution of the test under exactly the same conditions, repeatability is not given in practice because there are numerous potential and sometimes unknown or unaccounted influences. For this reason, the reproducibility of the results is impossible [8]. This leads to a high motivation for implementing a detailed surrounding model to reduce the test and development costs of assistance systems. With the EnvM the implemented functionalities can be simulated, tested, and the results validated. The test can be repeated and the outcome is always the same.

1.2 Problem Statement

This thesis is focused on developing some key aspects of an EnvM for an ADAS, mainly enhanced for testing and sensor fusion. State of the art vehicles have several environment sensors integrated. If information is available from more than one sensor, it needs to be merged in order to get a complex image of the surrounding area. In this context, complex means the model is able to handle the dynamic motions of the objects in its range. For example, when the ego vehicle (EV) is overtaken by another car, the other traffic member's motion is tracked with the support of the target tracking. The advantage is that the position and velocity of the car is always known. With this dynamic information of other objects, the EnvM can be used to predict other traffic participants, which is needed for a highway assistance system. The highway assistant is able to guide the EV autonomously on a motorway. This function reduces and increases the vehicle's velocity according to the driving situation and, if required it is able to overtake other traffic members.

The inputs to the model are the data from the sensors, which are mounted on the vehicle's body. The fusion merges the measurements from each sensor and calculates the dynamic environment information. The output of the model is detailed information about the other road users, which can be used as input for different assistance systems, see figure 1.2.

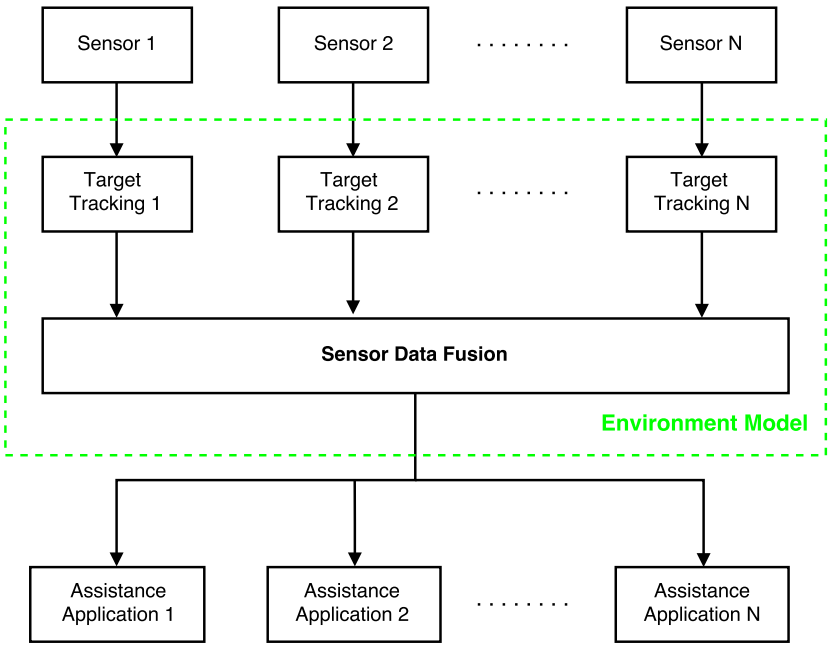

Figure 1.2: Principal concept of the environment model. The number of connected sensors is adjustable.

1.3 State of the Art

The number of assistance functions integrated in cars increases constantly with every new vehicle generation introduced [9]. Original equipment manufacturers (OEM), automobile industry suppliers and universities spend much effort and money in researching and developing new strategies and methods to build EnvM's [6], [10]. Assistance systems that operate fully or partly autonomous have high requirements to the model concerning to accuracy and real-time of the provided data. Models available today are optimized for applications that support the driver, but do not control or guide the vehicle [5], [11]. The next step in the development of surrounding models is to extend them, so that they can be used for assistance functions which make autonomous driving manoeuvres [4]. In this thesis an EnvM, that can be used as a test and simulation environment for autonomous driving functions is developed.

1.4 Main Contribution

- The sensor model extension considers the physical visibility of the detected targets, it can be used for different types of sensors and has an adjustable resolution. The incoming sensor data is processed to be serve as input for the EnvM.
- A dynamic object model in combination with a target tracking algorithm on sensor level is explained and developed.
- A sensor data fusion algorithm, which can handle both synchronous and asynchronous sensors, is implemented.
- The models and algorithms can be used to test and simulate the sensor configuration and advanced assistance systems.
- The presented model is validated with simulation use cases.

1.5 Structure of the Thesis

This thesis consists of six chapters. The first chapter is the introduction, where a general overview about assistance functions is given. In the second chapter, a sensor model extension is implemented and tested. The developed expansion can be used for different types of sensors and the resolution is adjustable. The third chapter gives an overview about the theory behind the fusion of the data from several sensors. Various merging concepts from literature are presented with their advantages and disadvantages. In the fourth chapter, the fusion architecture used in the thesis is explained and its implementation is described. The fusion is a two stage process: At the first stage, an object model is built and a tracking algorithm on sensor level is developed. Each target in the sensor's range is observed with the routine. In the next stage, the obtained information from the first stage is merged together to build the global object track. The fusion process is very difficult due to different sensor sample rates. This is solved by synchronizing the sample rates in an additional step. In the chapter five, "Validation Examples" the implemented software from chapter four is tested with typical use case scenarios. Afterwards, the results of the tests are evaluated. In the last chapter, "Conclusion and Perspective" the outputs of the thesis are reflected critically. Moreover, the advantages and disadvantages of the EnvM are pointed out. Concluding the thesis an outlook of what can be done in future is given.

Chapter 2

Sensor Model Extension

2.1 Introduction

Virtual testing becomes more and more important for the development of ADAS. Due to the strong interactions of the environment and vehicles with automated driving functions, environment simulation and sensor modelling are key factors for development. This chapter introduces a generic sensor model extension for simulation environments with the focus on visibility and masking. Figure 2.2 shows a typical use case of the extension. The model is based on a sensor which has integrated a signal processing, as illustrated figure 2.1. For more information on the underlying sensor model, please refer to [12]. The sensor data that has already been converted is the input of the model. Inside the model, an algorithm is implemented, that checks if the objects in the sensors field of view (FoV) are visible or masked by other objects. Simulation environments only detect if the object is in the sensor range or not, but do not consider object masking and visibility. In this special case, the sensor model extension decides if the object is viewable or not. Output of the expansion are all physically visible objects within the FoV.

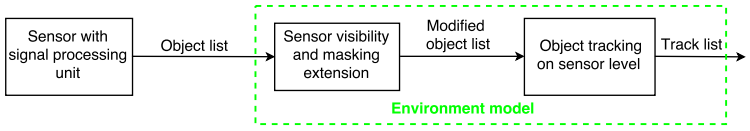

Figure 2.1: Illustration of the sensor model extension.

2.1.1 Requirements of the Sensor Model Extension

Requirements can be summarized as follows:

- Must be usable for different kind of sensors and independent from the sensor sample rate
- Must have a parameter where the sensor resolution can be set
- Must consider the physical visibility of the objects in the sensor range
- Must determine how many objects are detected in the sensor range
- Should have a parameter to adjust the percentage for when an object is physically visible

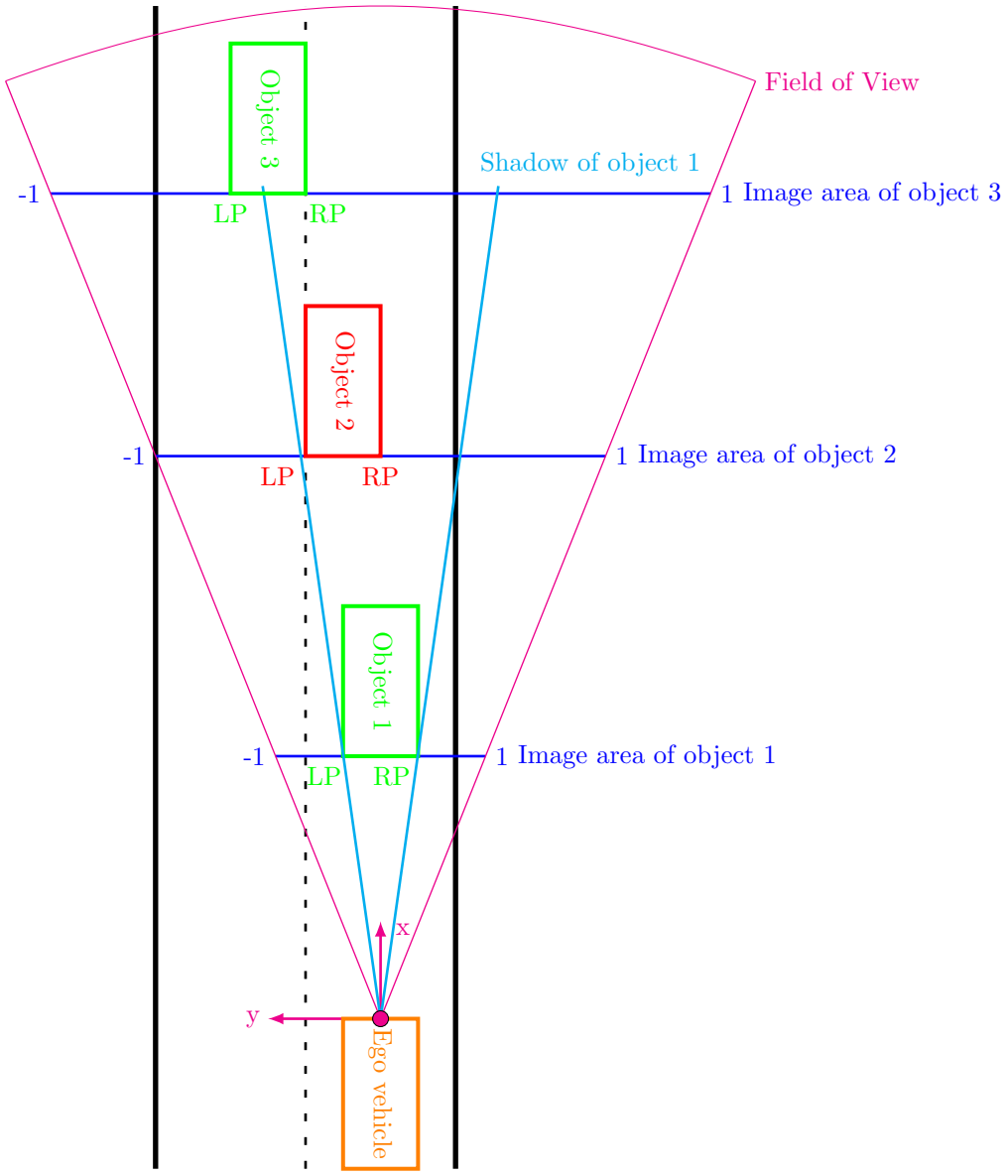

Figure 2.2: Use case scenario with 3 objects in the sensors field of view. Object 1 is visible, object 2 is fully masked by object 1 and object 3 is partly visible, partly masked by object 1.

2.2 Sensor Model Extension

Depending on the requirements in section 2.1, a sensor model extension is developed. All *must* requirements have to be included in the model and the *should* requirement can be considered. The result can be used for different types (Radar, Camera, Ultrasound etc.) of sensors with unequal sample rates. Deducing from the requirements of the model, it has two adjustable parameters **resolution** and **min visible**. The possible settings of these variables are explained later on in the section. Precondition for the model extension are basic sensor functions in the simulation environment, providing sensor raw data. In general the signal processing has to fulfill the following

conditions: the position (x and y coordinates) and y dimension of each object in the sensor range must be known. For the y extent of the object, also the leftest (**LP**) and rightest point (**RP**) must be identified and normed to the width of the field of view (FoV), see figure 2.2 and 2.3. The normed width of the sensor range always goes from minus one to one regardless on which x position the object is.

Visibility Checking:

The resolution of the sensor is modelled with the help of an array. The array partitions the object from its leftest to its rightest point. With the parameter **resolution**, the number of cells of the array is defined. The number of cells of the array is related to the angle resolution of the real sensor. One advantage of this solution is that the user can easily change the resolution of the simulated sensor. This ensures that the extension can be used for sensors with different resolutions. Furthermore the effect of giving the sensor a higher or lower resolution can be simulated and the results can be compared.

A "nice to have" requirement is to implement a parameter that defines the percentage at which a partly masked object is visible or not. For this requirement, there must be at least one fixed value of percentage of the parameter **min visible** so that the algorithm will recognize the object. Therefore, the percentage value is converted into a number of cells of the resolution array. Then the algorithm sorts the objects in the sensor range beginning with the object that is furthest away from the EV. The next step is to compare if the objects are masking each other. The program checks for each cell of the resolution array if it is hidden by another object. If so, the corresponding cells are set to zero. Visible cells are set to one and masked cells are set to zero as shown in figure 2.3. In the end the cells of each object are summed up and when the result of this summation is less than the value of the parameter **min visible**, the object is not visible. The function of the algorithm will be explained in detail in section 2.3.

2.2.1 Adjustable Parameters of the Sensor Model Extension

- **Resolution:** With this parameter, the angle resolution of the sensor can be set. In the model extension an object is represented as an array. This array marks the y dimension of the object in the sensor range. For instance, **resolution** is set to 19, this means the y extent of the object is divided into 20 cells (Keep in mind that C starts to count at 0.), see figure 2.3. If the parameter is increased, also the resolution will increase. If the parameter is reduced, this will lead to a reduction of the resolution.

- **Min visible:** This parameter defines which percentage of the object's y dimension must be physically visual that the algorithm recognizes the object as detected. For example, the parameter has a value of 60. If more than 40% of object 2 are hidden by object 1, object 2 will not be visible. As a result object 2 is not detected by the sensor. If the value of this parameter is increased, a higher percentage of the y dimension of the object has to be viewable to detect it. If the parameter **min visible** is decreased, a lower percentage of the object needs to be physically visible to detect it. With this parameter the quality of object detection of the sensor can be tested and simulated.

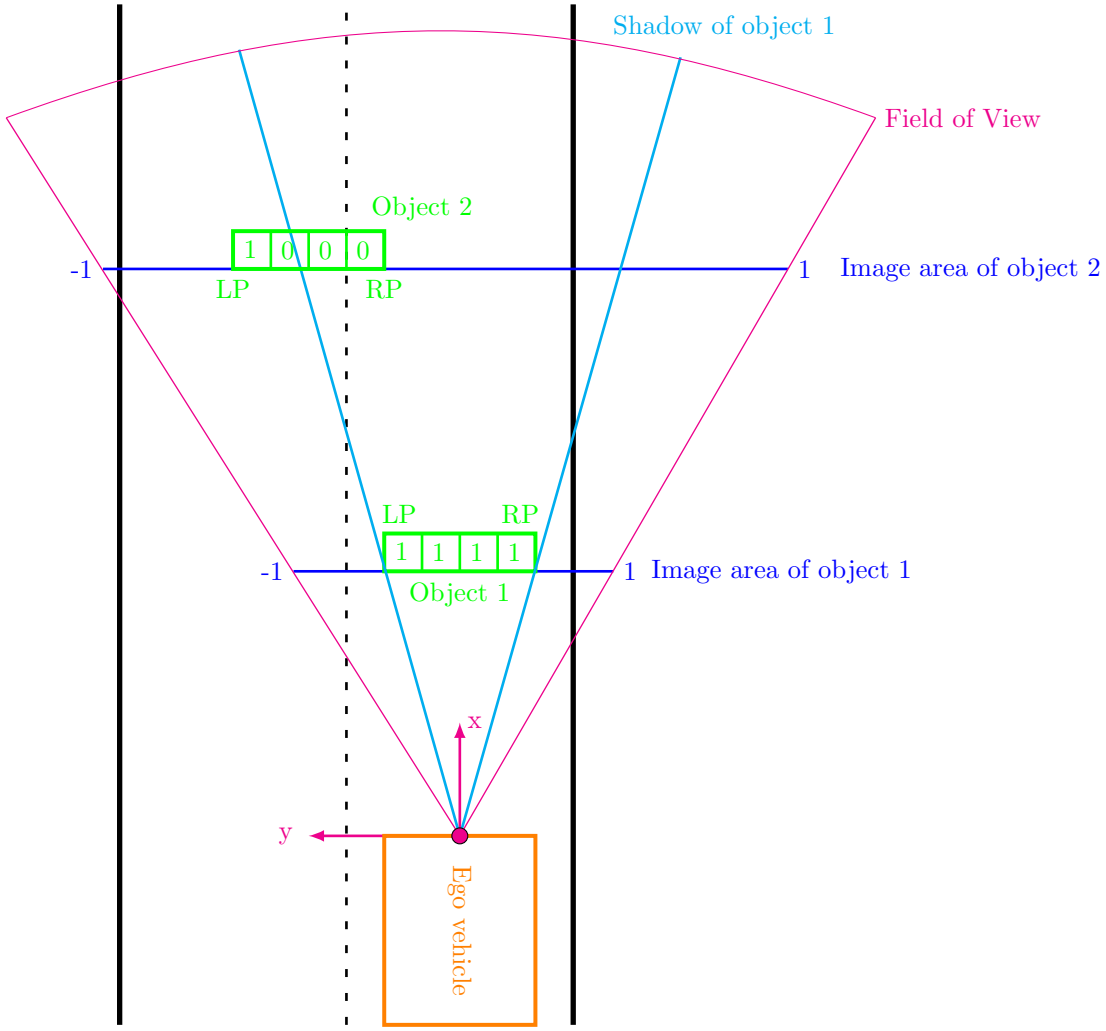

Figure 2.3: Schematic of the sensor model.

2.2.2 Assumptions and Boundaries of the Extension

The algorithm only considers the object's y dimension but not its x and z dimension. This behaviour can lead to the following situation: In front of the EV are a truck and a passenger car, with the passenger car driving between the truck and the EV. A human driver is able to recognize the truck because it looks over the car in z direction. The sensor model extension is not able to determine the z component of the vehicles and so the visibility check will result in the truck being hidden by the passenger car and, therefore, the truck would not be detected. A possible solution would be a detection algorithm for the kind of object. The output of the algorithm would be the type of object (passenger car, truck, bus, motorcycle, pedestrian etc.) and this information would be additionally saved to the object information. The model extension could then use this information and so detect the truck in front of the passenger car.

2.3 Implementation Details

The sensor visibility checking is implemented in C and called Object Detection Algorithm (ODA). For reasons of clarity, no pieces of C code are shown here but flowcharts are used instead to illustrate the structure of the program. The algorithm is needed because the object is detected as soon as it is within the range of the sensor in simulation environment, whether or not the object is physically visible. For example, two cars (= objects in the algorithm) are driving in the same lane in front of the EV. In that case, the car which is further away from the EV is not seen, because the other object which is closer to the EV is masking it. The task of the ODA is to detect if an object is physically visible. If so, the object detection flag is set to one. Figure 2.4 shows the flowchart of the algorithm.

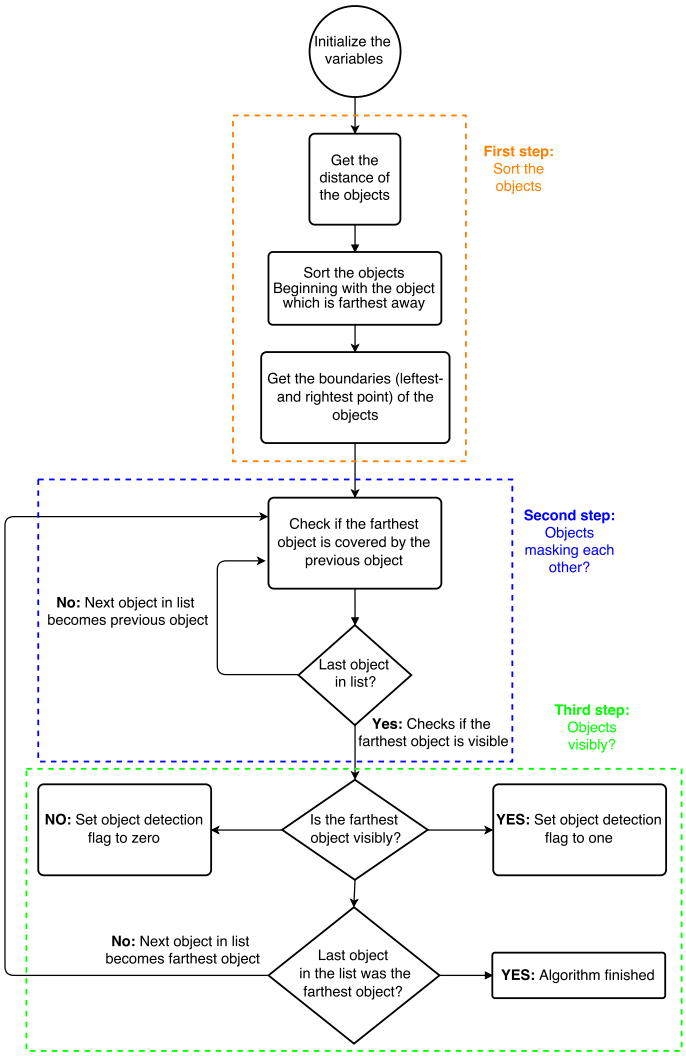

Figure 2.4: Flowchart of the object detection algorithm.

The algorithm consist of three main steps:

- Sort the objects, beginning with the object which is furthest away from the EV.
- Check if the object is covered by any other objects.
- Check if the object is visible or not and set the object detection flag accordingly.

First step

First of all, get all objects which are detected by the sensor and store them in a list. The list is sorted such that the object which is furthest away from the EV comes first in the list. Then get the boundaries (LP and RP) of each object, i.e. the area of the sensor range the object covers in y direction and save them to the list.

Second step

The next step is to determine if an object is masking another object, starting with the first item in the list. Therefore, the area which the object covers in the FoV is converted to an array with a changeable number of cells, that is to say the resolution of the sensor. The array is initialized with all cells set to one in the beginning. Then the area of the first element in the list is compared with the second element in the list. If the two areas are overlapping each other, the cells of the first object which are overlapped are set to zero. These cells are hidden by the second object. Then the array of the first object in the list is matched to the array of the next object in the list. If the two arrays are also overlapping each other, the corresponding cells of the first object are set to zero. This is repeated with every object in the list. When the first item in the list was compared to every object in the list, the algorithm moves to the final step.

Third step

Lastly the algorithm determines if the object is visible or not and sets the object detection flag depending on that. In the previous step, the cells of the object array have either set to one or zero. One means the cell is visible, zero that the cell is masked by another target. With the adjustable parameter (**min visible**), it can be set which percentage (cells) of the object has be filled with the value one so that the object is detected as visible. To be able to compare the value of the parameter (**min visible**) with the sum over the array the value has to be converted into a value of number of cells. When the built sum over the array is greater than the converted parameter value, the object is viewable and the object detection flag is set to one. If the sum is smaller than the parameter, the object is hidden and the object detection flag is set to zero. After that the algorithm jumps to the previous step and compares all objects besides the first one to the second item in the list. In other words, now it is checked if the second object is viewable or not. This is repeated until the program has reached the last object in the list. The last object in the list is automatically visible.

2.3.1 Example

To make it more clear how the algorithm is working, here is an example. Imagine the following situation: EV is driving in the right lane (gray BMW) of a two lane highway. In front of the EV, are three other traffic members driving. Two vehicles (blue car and truck) are in the same lane as

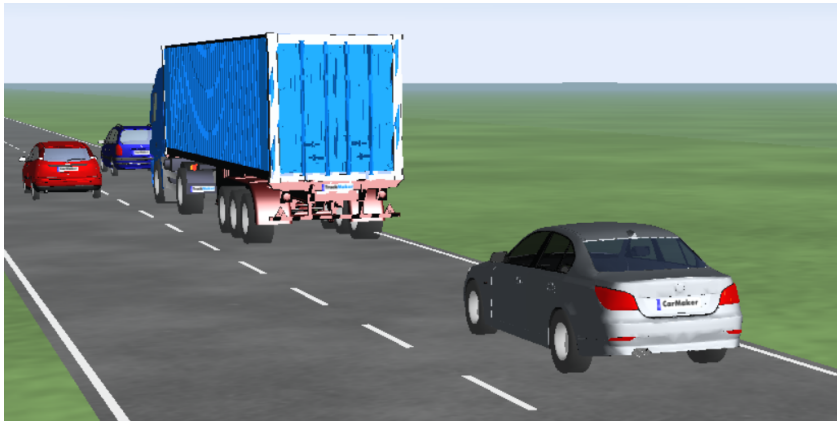

Figure 2.5: Side view of the scenario. Shows all member of the traffic situation.

the EV. One car (red) is in the left lane. The described scenario is illustrated in figure 2.5.

Figure 2.6 gives a better impression of the driver's view. The driver is not able to see the blue car which is driving in front of the truck. The expected result of the algorithm is that the blue car is not detected and that the truck and the red car are visible.

Figure 2.6: Driver's view of the scenario. Only the red car and the truck can be seen.

The following figures will illustrate the output of every step of the algorithm. Figure 2.7 shows the result of the **first step** of the algorithm, the sorted list of objects with areas which the objects are covering in the y direction in the sensor range. The target which is farthest away from the vehicle is listed first (see X in figure 2.7). The first element in the list is the blue car (ID: 2), followed by the red car (ID: 3) and the truck (ID: 4). The last item in the list is filled with default values because in the scenario, there are only three objects defined. The program is able to handle up to 100 objects, the example here is done with three targets.

```
--------------------------------
Sorted List with Points
X: 36.385 ID: 2 LP: -0.042 RP: 0.054
X: 26.385 ID: 3 LP: -0.307 RP: -0.151
X: 11.379 ID: 4 LP: -0.207 RP: 0.243
X: -500.000 ID: -1 LP: -2.000 RP: -2.000
--------------------------------
```

Figure 2.7: List with sorted objects and boundaries.

The **second step** checks if the object is hidden by another object. In this case, it is checked if the target with ID: 2 is covered by any other object. The result shows that object two is completely hidden by the object with ID: 4, (see figure 2.8). The array consists of only zeros. Target two is the blue car which is driving in front of the truck and there is no possibility that the driver of the EV can see this car.

```
4 Complete Hidden: far Obj ID: 2 pre Obj ID: 4
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
--------------------------------
```

Figure 2.8: Array of the object with ID: 2 (blue car).

The algorithm has checked if target two is hidden by another object. Now the algorithm checks if the next element in the list (object with ID: 3) is masked by any object. The result of this execution is displayed in figure 2.9. Object three is partly covered by object four. If target three is detected, will be determined in the next step.

In **step number three**, the object detection flag will be set, according to if the object is visible or not. For this case, the array of the objects consists of twenty cells and an object is detected if 50% of the y dimension are visible. As mentioned before, the number of cells (parameter **resolution**) and the percentage of visibility (parameter **min visible**) of the object are adjustable. This means that in our case, there must at least be 10 cells with the value one in the array so that the detection flag will be set to one. The sum over the array of the target with ID: 3 is 13, see figure 2.9.

```
2 Left: far Obj ID: 3 pre Obj ID: 4
 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0,
--------------------------------
```

Figure 2.9: Array of the object with ID: 3 (red car).

Figure 2.10 displays the result of the ODA. The object with ID: 2 (now with ID:-1, first element in the list) is not detected and so the flag is set to zero. The target with ID: 3 is partly visible, but enough so that the object detection flag is set to one. This is because the sum of the array of object three is 13 and for the detection the sum must be greater than or equal 10. The object with ID: 4 (the truck) is also visible and the detection flag is set to one.

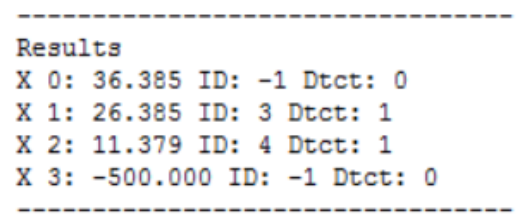

Figure 2.10: Result of the Object Detection Algorithm

2.3.2 Results and Conclusion

As expected, the blue car (first element in the list) is not detected by the algorithm and the detection flag is zero (Dtct: 0). The truck (ID: 4) which is driving directly in front of the EV (gray car) is recognized by the ODA and the flag is set to one (Dtct: 1). The red car (ID: 3) has enough cells of the object array viewable so the object detection flag must be one.

The sensor model extension determines if the objects in the sensor's FoV are masking each other. The extension has two main parameters to tune, the resolution of the sensor and visibility of the targets. With the number of cells in the object array, the angle resolution of the sensor is defined. If the object is sufficiently visible, i.e. the percentage of the object's visibility is high enough according to the parameter **min visible**, then the detection flag will be set to one. This means that the object is detected by the model extension.

Chapter 3

Sensor Data Fusion

3.1 Introduction

According to targets and requirements to a data merging algorithm, there are several architectures possible. The chosen structure has a great influence on the performance and accuracy. In this chapter, important parts of the theory behind **sensor data fusion (SDF)** are explained. Different approaches used in the literature are compared and the advantages and disadvantages are discussed. Finally the architecture which is used in the thesis is explained.

A measurement (observation) is the registration of the environment with a sensor. The result of this operation is a vector where the measurement date and the detected characteristics (position, velocity, etc.) are stored. With the help of the vector, an estimation of dynamic processes is possible. The observation reduces the real world to the measurement characteristics [5].

An ADAS needs a detailed EnvM which describes the surrounding area of the vehicle. With mathematical equations, these relevant objects are specified. This behaviour and movement are described with a model. The so called **object model** is an abstraction of the real world and will be explained in detail in section 4.1.2.

The target of the object tracking is to use collected measurements to get a result which is better than a single measurement. When all data are used to track one object this is called **single target tracking (STT)**. If there is more than one object to be tracked, it is multiple target tracking. In **multiple target tracking (MTT)**, the measurements have to be assigned to the right track. This process is named **association**.

Often the information from one sensor is not detailed enough, therefore, a sensor network is needed. The combination of measurements from different sensors to define an object's state is called **sensor data fusion** [11]. The fusion can have different motivations, which are listed here.

- Increase the accuracy of the estimation through the use of more sensors with the same sensor range
- Expand the total field of view with sensors with different detection areas
- Increase the drop out stability by using multiple sensors
- Enhance the robustness against environmental influences by using different sensor types (Radar, Ultrasound, etc.)

- Get more detailed information with the combination of different sensors
- Improve obtaining of information winning using a sensor network

Regarding the characteristics of the used sensors, there are two different types of sensor networks defined. If the sensors all have the same physical measurement principle, it is a **homogeneous sensor network**. The disadvantage of this network is that all sensors have the same limitations. In a **heterogeneous network**, different type of sensors are in use. The advantage of a heterogeneous network is that the weakness of one sensor can be compensated by the strength of another sensor. This leads to three different general fusion structures [13]:

1. **Competitive Fusion:** The data comes from identical sensors with the same field of view. Through increasing the amount of sensors the accuracy of the fusion gets better. If one sensor falls out, there is no influence on the detection area and feature space observed.

2. **Complementary Fusion:** If there are no sensor ranges overlapping each other or the sensors all are measuring different parameters, this is called complementary fusion. Different parameters are, for example, one sensor is detecting the width of an object while another sensor is measuring the height of the object. When one sensor is defect there is always a loss of information.

3. **Cooperate Fusion:** If the combination of sensor data leads to a new measurement characteristic, this is called cooperate fusion. For instance, one sensor is measuring the width of an object and another sensor is observing the height of the same object. Through the combination of both measurements the geometry of the object is determined.

In practice, fusion structures are most times a mixture of the different structures described above, like the sensor network used in the thesis: some sensors do have an overlapping FoV and some sensors do not. This results in a composition of competitive and complementary fusion.

3.2 Classification of the SDF

Up to date there is no standardized classification for SDF of environment sensors for assistance systems. This section gives an overview about the used approaches. Due to the great number of possible structures of data fusion, only the important ones are presented here. Every classification concept is described, and advantages, disadvantages and characteristics of each concept are highlighted [5], [8].

3.2.1 Classification According to the Place of the Fusion

The classification in **decentralized**, **central** and **hybrid** is done according to where the fusion takes place. This notation is often used for object tracking applications.

Figure 3.1 shows the structure of a **decentralized fusion**. In every sensor, the object tracking is done individually and then the tracks are handed over to the global fusion, where the results are merged together. For the target tracking, this structure is very beneficial with respect to minimization of the estimation error of the state variables. According to the modularity of this

architecture, sensors can simply be added to or removed from the system without changing the global fusion unit which is an big advantage. However, this structure also has some limitations, all sensors must measure the same characteristic (position, velocity, etc.) of an object.

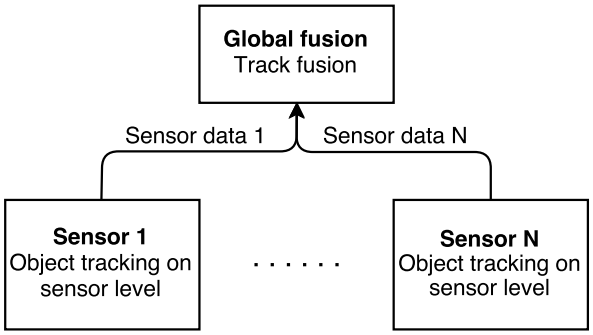

Figure 3.1: Structure of the decentralized fusion.

If a Kalman Filter is used for tracking, the system model of the filter has to be the same for all sensors. For a modular system this is an advantage, it is easy to add a sensor to the network: copy the block sensor, connect it to the block global fusion and the new sensor is integrated to the system. For removing a sensor it is the opposite way: disconnect the sensor block and delete it. According to the target tracking which is already done on sensor level, the data transfer between sensor and fusion is reduced. Another advantage is the rather easy implementation of different samplerates.

In the **central fusion**, figure 3.2, the data are only slightly edited on sensor level. The object tracking and combination takes place in the fusion block. This makes the structure quite inflexible, since for adding or removing sensors the fusion routine has to be modified each time. Another disadvantage is the high amount of data which must be transferred between the sensor and the fusion.

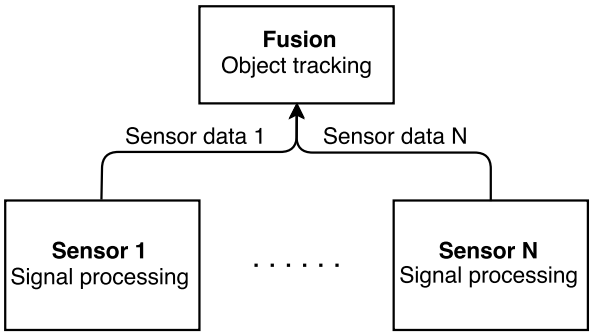

Figure 3.2: Structure of the centralized fusion.

In **hybrid fusion**, central and decentralized fusion are combined. The fusion block is able to

receive low edit data (central fusion) or already object tracks (decentralized fusion). The following example illustrates hybrid fusion: the detection area of the whole system is split into different sensor networks, e.g. all sensors which are mounted at the back of the vehicle are in one sensor network, all sensors positioned at the car's front are in another sensor network. Each sensor network is built up in the central structure. Now the networks are combined together to detect the whole surround area of the EV. This combination is done in decentralized structure. As a result, the structure of the total data fusion is a hybrid one.

3.2.2 Classification According to the Edition Level of the Input Data

The classification in fusion on **row data**, **characteristic** or **decision level** results from the edition level of the insert data to the merging process. This kind of division is often used by object specification algorithms, which detect for example if the target is a passenger car, truck, bus, etc.

Within **fusion on raw data level**, data is only minimally edited and merged in the sensor resolution available at the central unit. This architecture is used for example in an image processing algorithm, where the information of the different spectra of light is considered. An advantage of this method is that the total information of the sensor is used. The fusion software can be specialized to this information. The disadvantages of fusion on raw data level are the high data transfer between the sensor and the fusion and the difficult expandability and changeability of the program.

In the **fusion on characteristic level**, first the parameters (like position, velocity, etc.) are determined and then the fusion takes place. Consequently, the transfer volume between the sensor and the fusion is reduced. Therefore, a loss of information is accepted.

Fusion on decision level means that there is already a signal processing performed on sensor level. In the signal processing unit a decision is taken. This could be e.g. an object classification, like of which type (bus, truck, etc.) the target is. The determined type is then send to the fusion where it can be combined with information from other sensors. Such a combination can be done with an object track. The result is then a track where the class of the item is also known.

3.2.3 Classification According to the Sensor Sample Rate

According to the sensor sample rate there are two divisions: **synchronized** or **unsynchronized**. The classification is based on when the sensors are collecting data.

In **synchronized fusion**, all sensors are measuring at the same point in time, so there is no time delay between the several measurements. An advantage is that the time behaviour of the system is deterministic and predictable. However, a disadvantage is the additional synchronization effort either in hardware or software. If the sensor network is using heterogeneous sensors, a synchronized architecture cannot be used because the sensors do not have the same sample rate.

In **unsynchronized fusion** the sensors are measuring at different points in time. The advantage is that heterogeneous sensors can be used. According to the different sample times of the sensors, a synchronization step is necessary before the fusion.

3.2.4 Event Based Fusion Classification

Regarding to the event that must happen to activate the fusion there are three different types distinguished in literature: **new data** available, a **certain data constellation** and an **external event**.

If there is **new data** available from the sensor, the fusion will get active. Depending on if the network has synchronized or unsynchronized sensors, the fusion algorithm must have a strategy to cover measurements which do not arrive in the correct time order. In a decentralized structure, the latest merged data can be fed back to the sensors. Then the sensors have the latest data and can include them in their signal processing.

When using the method **certain data constellation**, the fusion routine must have a temporary memory. Such a "certain data constellation" arises when data is available from a predefined sensor. Because of waiting for a certain event, the fusion always has a time delay.

When the results of the fusion are not fed back to the sensors, the fusion can be started by an **external event**. This method has the advantage that the sample rate of the merging can be chosen and fitted to the working process. However, the adaptation of the fusion sample rate leads to a loss of accuracy of the tracking process.

3.2.5 Classification According to the Input Data

Regarding the kind of the input data to the fusion process, there are three partitions possible: **original input data**, **filtered input data** and **predicted input data**.

In the classification **original input data**, the information from a sensor comes to the fusion program without filtering and with a time delay. This makes an optimize track estimation possible.

If **filtered input data** is used, like for example in the decentralized fusion, under certain circumstances an optimized estimation is possible. When the filtered data is treated like unfiltered data and is passed on to another filter, the result is a filter chain. This leads to a time delay through the filters.

Another possibility is the usage of **predicted input data** for the fusion algorithm. This practice is often used when data is collected for a certain time point. The predicted measurements of the sensors are then fused to one data.

3.2.6 Classification According to Processing Time Step

This classification is done considering at which time step the fusion is executed. There are two different divisions: **parallel fusion** and **sequential fusion**.

In **parallel fusion** the measurements integrated to the fusion in one execution step. In **sequential fusion**, the measurements are added in multiple time steps.

3.2.7 Conclusion

There are many different ways to classify measurement and fusion. In practice it is necessary to use combinations to describe a fusion algorithm. This subsection should give an idea and an overview of the mainly used terms. For example the approach used in this thesis is built in **decentralized structure**, with **unsynchronized sensors**, triggered by an **external event**, with **predicted input data** and is **parallelly executed**.

3.3 Multiple Target Tracking

Since there can be more than one object in the FoV of every sensor, only the multiple target tracking is relevant for this thesis. In contrast to the single target tracking, the measurements have to be associated with the right object. There are two kinds of target trackings, either the tracking is done with **one sensor** or there is a **network of several sensors** used to track the objects.

3.3.1 Multiple Target Tracking on Sensor Level

Figure 3.3 shows the typical steps of a recursive multiple target tracking algorithm on sensor level. The processing consists of seven parts. In the sensor detection area, there can be several objects. The aim of the tracking program is to associate the measurements with the according tracks. This is difficult if the objects are making lane change manoeuvres and the tracks cross each other. On sensor level, there exists one object for one track. Literature to the topic can be found in [13] and [14].

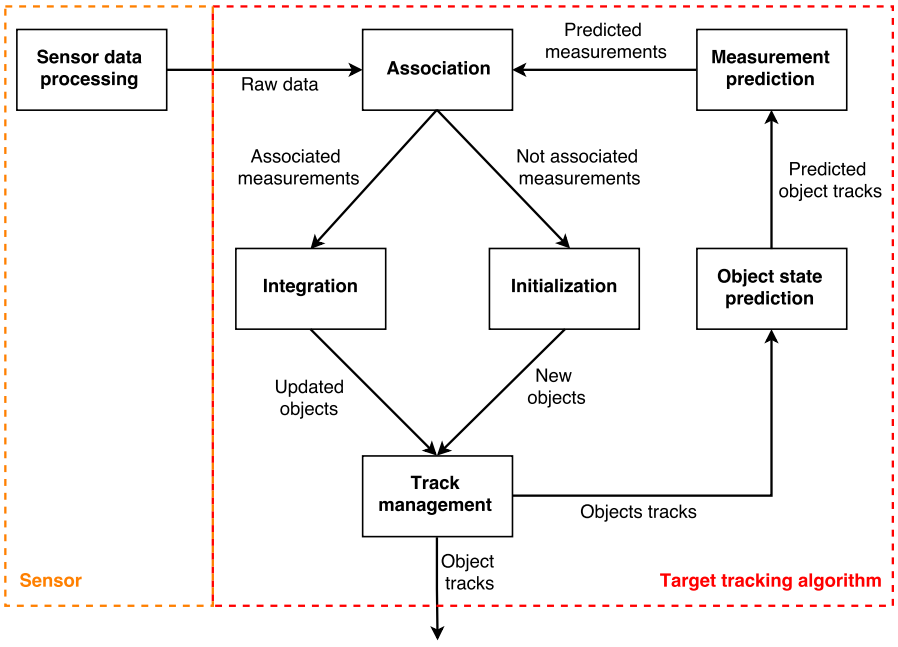

Figure 3.3: Architecture of the multiple target tracking on sensor level.

- In the **sensor data processing** the sensor raw data is converted into measurement data, like distance, velocity, etc. This adaptation is done most times by a signal processing unit which is integrated in the sensor and afterwards the data is sent to the tracking algorithm. For the tracking algorithm, these measurement data are the raw data for the object tracks.
- With the help of a distance range, the **association** between the predicted measurement values and the raw data (current measurements) is done. The association decides which sensor data is used to update existing object tracks and which data initializes new tracks.
- The **integration** executes the state variable update of the objects. Since the state vectors of the objects and measurements do not need to be in the same feature space, the values of step measurement processed during prediction are considered here. It is possible to add a weighting in this step according to the accuracy and reliability of the measurements and the object states.
- If some measurements are not associated to any track, they are used to initialize new object tracks. The new tracks will be supplied in the next execution step to prediction process. This part of the program is called **initialization**.
- A further part of the algorithm is the **track management**. Tracks which are not associate with new measurement values over a certain time period are deleted. Also the merging of several tracks to one can be done here.
- In **object state prediction**, the state variables from the previous execution step are predicted to the current measurement time point. This means that the movements of the objects are predicted with the values from the last estimation.
- It is possible that the measurements are not in the same feature space as the objects. In other words, maybe only the position is measured but the object state variable consists of the position and velocity. The speed is then calculated by the algorithm and not based on measurements. In the **measurement prediction** the object state variables are integrated to the feature space of the measurements.

The steps which are described here are necessary for MTT on sensor level. Since modern cars have more then one sensor, the structure must be expanded to handle the data from a sensor network. The MTT with a sensor network is explained in the next section.

3.3.2 Multiple Target Tracking with a Sensor Network

Since modern cars have more than one sensor, the mulitple target tracking is not done with one separate sensor but with a network of sensors. The data of all sensors must be integrated by the state estimate. The state estimate of the object is now a combination of data from different sensors. The merging of data from more then one sensor is called **sensor data fusion**, which has already been explained at the beginning of this chapter. In this subsection, fusion algorithms for the architectures from section 3.2 are described. Each concept is based on the recursive algorithm from the MTT on sensor level. According to the system architecture and the used sensors, the steps from the MTT on sensor level are adapted and, if necessary, some additional steps are added. The content of this subsection is based on [13] and [14].

Synchronous Sensors

A sensor network with synchronous sensors has the big advantage that all measurements are available at the same point in time. This makes the combination of the data easier. Since the measurements do not have a time offset they can be integrated in the same iteration step to the tracking system, as shown in figure 3.4. In the figure, an example for tracking with a network

consisting of two synchronous sensors is shown. **M** stands for the measurements from the sensors and **F** for the SDF. Every additional measurement value results in an increase of information which reduces the inaccuracy of the object track. Only one object model for time prediction of the global tracks is needed. The sensors deliver raw data. Tracking algorithms for synchronous sensors can be implemented **parallel** or **sequential fusion**.

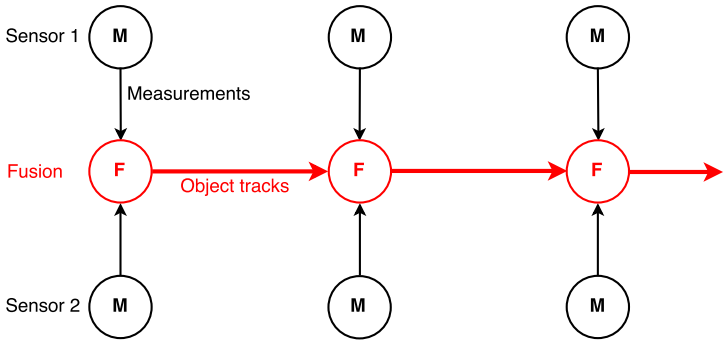

Figure 3.4: Fusion structure with two synchronous sensors.

Parallel Fusion: All measurements which belong to one object are introduced simultaneously to the fusion. The integration of the data is done in one execution step. The parts of the single sensor tracking are passed as follows.

- All sensors measure synchronously. Then the signal processing, abstraction and interpretation of the raw data is done for each sensor. These actions take place in the sensor processing unit and afterwards the data is sent to the tracking program. For the tracking algorithm, the already processed sensor measurements are raw data.
- The **association** is done for each sensor and each object in the FoV. If an object is in several sensor ranges, then multiple measurements are used for one track.
- For objects with only one measurement value, the integration takes directly place. If an object has several measurements, these are summarized to a multilevel vector. Then the vector is integrated to the program in one execution step. This part of the algorithm is called **integration**.
- The **initialization** is rather complex in MTT with a sensor network: not every measurement which is not associated has to create a new track because the object is already tracked by another sensor. When an object is observed from several sensors, all the according measurements can be taken to initialize a new track.
- The merging of tracks plays a big role in the MTT with a sensor network. Since the objects can be detected from several sensors, it is possible that it is not detected that the measurements belong to the same track. The task of the **track management** is to fuse such tracks together to one track and delete unused object paths.
- The **object state prediction** is done in the same way as in MMT on sensor level. The state variables of the objects are predicted based on the last state estimation.
- The **measurement prediction** takes place separately for every sensor. If an object is in several detection areas, several measurements will be predicted. This means that an object

track can be based on more than one measurement. The number of measurements can maximally be as high as the number of sensors.

Sequential Fusion: When the measurements are integrated in order and not parallel to the **object state predict**, the process is called **sequential fusion**. In this case the data is not introduced to the fusion in one execution step with a multilevel measurement vector. Instead, first the state prediction is done with the first associated measurement, then with the second associated measurement and so on. The amount of integration steps can not be higher than the number of sensors in the network.

Please note: the order in which the data is integrated to the algorithm can influence the performance of the state prediction. The best way to introduce data is to start with the most precise measurement [13].

Unsynchronized Sensors

Synchronous sensor networks are addressed in scientific papers. But in practice they are hardly realizable because the sensor types used for the environment recognition have very different physical measurement methods so that a synchronization is in most cases not possible or not reasonable. The sample rate of a synchronization must be oriented on the slowest sensor's sample time. This means that the information density of the faster sensors would be lost. In a network with **unsynchronized sensors** there are time delays, since the measurements can not be integrated to the fusion in the order they appear. This makes target tracking more complex. The object state prediction is by a recursive algorithm only in positive time direction (to the future) possible. The fusion should be able to handle measurement data which arrives late. The time delay between the real time and last object state update should be as small as possible.

Prediction to a Common Fusion Time Point: Up to now the explained algorithms use a central unit where the fusion takes place. Such systems are called **central level fusion (CLF)**. The sensor data are sent to the central unit in which the steps necessary for the object tracking are executed. Another opportunity is the **sensor level fusion (SLF)**. In this method, the target tracking is already done on sensor level, like in the MTT on sensor level. Each sensor has its own tracking algorithm which can be specialized for the sensor. The central unit's task is to fuse the object tracks from the sensors together to one global track, as shown in figure 3.5. The figure shows a target tracking architecture with prediction to a common fusion time point. **M** marks the measurements, **F** symbolizes the MTT on sensor level and **GF** is the global tracking where the several tracks of the object are merged together to one track. The central data processing unit makes a track to track association. Due to the prediction to a common fusion time point, the sample rates of the sensors and the global fusion can be independent. As a result, the sample time of the global fusion can be defined by the user.

Since every sensor has its own object model, it is possible to predict the state variable to each time point in the future. Consequently, the fusion is able to combine data from sensors with different sample rates. Moreover, the object models can be unequal. This means that the different object models do not have to observe the same characteristics of an object (position, velocity, width, etc.). According to that, the object data must be equalized to the same features. The two necessary steps of the fusion are the time and territorial data assimilation, since it is only reasonable to merge same object parameters.

The advantages of this method are that the model is special for each sensor, optimizing object model and algorithm for the target tracking. When every sensor has its own object model, these

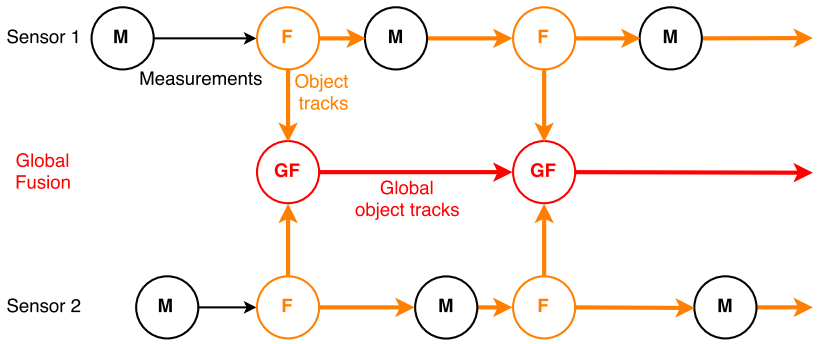

Figure 3.5: Structure with two unsynchronized sensors and prediction to a common fusion time point.

models can consider the strength and weaknesses of the different sensors. The workload of the central unit is decreased, since the tracking is already done on sensor level and here only several tracks have to be fused to one global track. Also the bus load is reduced because the sensors send only the object tracks to the central unit and not the complete measurement data. On the other hand, the calculation actions in the sensor increases. The information from other sensors is not considered by the tracking, since the tracking algorithm only has the data from one sensor. Considering that the sensors do not have to have the same sample rate, the data has to be predicted to the fusion time point. Because of the prediction, new measurements are not immediately integrated in the fusion.

Retrodiction of the Measurement: In fusion with synchronous sensors and the prediction to a common fusion time point, the measurements are brought in periodically and not immediately. To cover the disadvantages of the prediction which uses both methods, it would be better to bring in the measurements sequentially. When the sensors have sample rates which do not overlap each other and the processing and transfer times are very small, this method can be implemented easily, as shown in figure 3.6. **M** symbolizes the measurements and **F** is the fusion. As can be seen in the figure, the measurements arrive sequentially and the sample times do not interfere each other. Imagine the following situation: a measurement value arrives which is older than the last measurement that was integrated in the fusion. To introduce this measurement to the fusion, the prediction has to go backwards in time, which is called **retrodiction**. For this retrodiction, mathematical solutions are available but they are hard to realize and additional memory is needed.

Retrodiction only considers the consequences of old measurements to the state prediction. The following critical points are not covered: An old measurement can initialize a new track which has already been deleted by the track management. This track would have an influence to the following association steps. If the integration is working with a changeable number of states, an old measurement can create a new feature to the state variable. This changing of the size of the state variable would have an impact to the following execution steps. The biggest advantage of this method is that the measurements are immediately used in the fusion and so there is no time delay between the sensor raw data and the tracks.

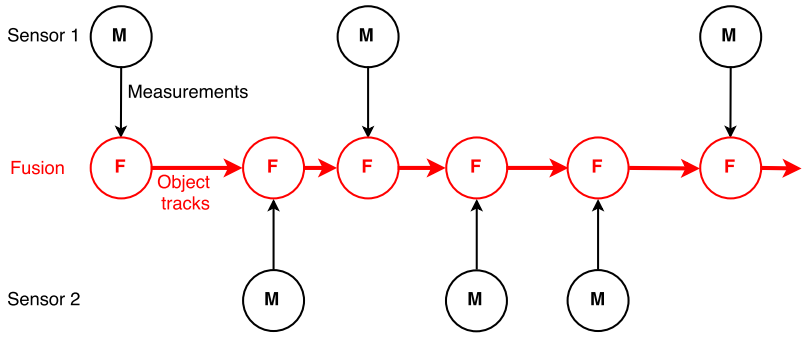

Figure 3.6: Structure with two unsynchronized sensors and retrodiction.

3.3.3 Conclusion and Comparison of Common Fusion Types

With a sensor network, the FoV of the environment model can be expanded. When sensor ranges are overlapping each other, the accuracy and reliability of the object tracks are increased and also the drop out stability is improved. If the sensors have different sample rates, a synchronization is necessary, which rises the complexity of the fusion. More sensors create more data which has to be transferred and processed. Compared to a MTT on sensor level, the complexity and effort of the fusion are much higher.

If the network is only using **synchronous sensors**, all measurements are available at the same time point, and so a synchronization step is unnecessary. This reduces the time delay between the data and the tracks. Resulting from that the system is real-time capable and has a high accuracy, see [5]. Also only one object model for the combination of the different sensor data is needed. The modularity of the system is restricted because if a sensor is added or removed from the network, the object model must be adapted. A network with synchronous sensors has hardly any practical importance because due to different physical measurement principles of the sensors, they will ever have a common sample time. The measurements of the tracking algorithm can be integrated in two ways, either **parallel** or **sequential**. For parallel integration, one execution step is necessary, which has a positive effect to the process time. In sequential integration, there may be as many iterations needed as there are sensors in the network. This possibly increases the processing time of the fusion.

Because of the necessary synchronization at networks with **unsynchronized sensors**, they always have a time delay. However, this software architecture is suitable to be used in applications for the real world, see [13], [14]. There are two different ways to implement an algorithm for the target tracking. In the **prediction to a common fusion time point**, the object tracks are updated to the fusion time point. Since the time difference from every sensor to the fusion varies, the tracking is done on sensor level. This leads to the following advantages and disadvantages:

+ Sample times of the sensors and fusion are independent from each other
+ Object model and tracking algorithm are specialized to strength and weaknesses of each sensor
+ Behaviour of the fusion with different sample rates can be simulated and compared
+ Workload of the central fusion is reduced

+ Data transfer between the sensors and the fusion is decreased
- Calculation effort on sensor level increases
- Information from other sensors is not considered in the object tracking on sensor level

The real-time response of the system and the accuracy of the target tracks are reduced, but the modularity is increased with this method. To overcome the time delay by tracking with unsynchronized sensors, the second method **retrodiction of the measurements** was introduced. This option's big advantage is that the measurements go directly to integration and so there is no time delay between data and track. Since prediction back in time (retrodiction) is also necessary, the following problems occur:

- Old measurement can create a new track or new object parameters
- Additional memory is needed to guarantee the retrodiction

Because of the retrodiction, additional memory is needed and the complexity of the algorithm increases.

3.4 Used Fusion Architecture in the Thesis

In this subsection the architecture of the SDF used in the thesis is described, based on [15] and [10]. The requirements to the software have a great influence on the chosen architecture and the integrated algorithms. According to the general targets of the fusion, listed in section 3.1, and the targets of this thesis in section 1.4, the requirements are summarized.

3.4.1 Requirements for the Sensor Data Fusion

General Requirements:

The general requirements can be summarized as follows:

- Must have a modular structure, so that it is easy to add or remove sensors
- Must be able to deal with different types of sensors (e.g.: Radar, Camera, etc.) with unequal sample rates
- The system must be real-time capable. ADAS applications need to get the data from the EnvM with a defined maximum latency. Execution effort must have an upper bound
- The system should be robust against disturbances, like if a sensor value for some samples is missing or measurements are noisy
- Should be usable for the development, testing and simulation of assistance systems

A multiple target tracking with a network of unsynchronized sensors is the best solution to fit the above defined requirements. According to this the fusion is a two stage process, the first step is the object tracking on sensor level and the second stage is where the global object tracks are fused. On each level a separate algorithm is needed, fulfilling the following requirements:

Requirements for the Object Tracking on Sensor Level:

- Build up an object model which handles the dynamic movements of vehicles in typical traffic situations
- Reduce the noise and cover the drop-out of measurements
- The object tracking time should be adjustable with a parameter
- Should be reusable for different sensor sample rates

Requirements for the Global Sensor Data Fusion:

- Flexible in the number of sensors connected to the system (⇒ easily possible to add or remove sensors)
- Merge the object tracks from the different sensors to one global object track
- Synchronize the different sensor sample rates to one global sample rate

In figure 3.7, the structure used for the SDF is illustrated. The input data for the object tracking comes via a sensorbus from a sensor model. After the object tracks are created on sensor level, they are sent to the global tracking unit and the several target tracks are merged to one global track. The output of the data fusion algorithm must be one track for each object, which is detected by any of the sensor mounted on the vehicle body. Since it is possible that the detection area of some sensors are overlapping, there exist more tracks for the same object on sensor level. The global tracking unit must now fuse these tracks together to one track. A challenge is to merge the target tracks from sensors with different sample rates together: for this a time synchronisation is necessary.

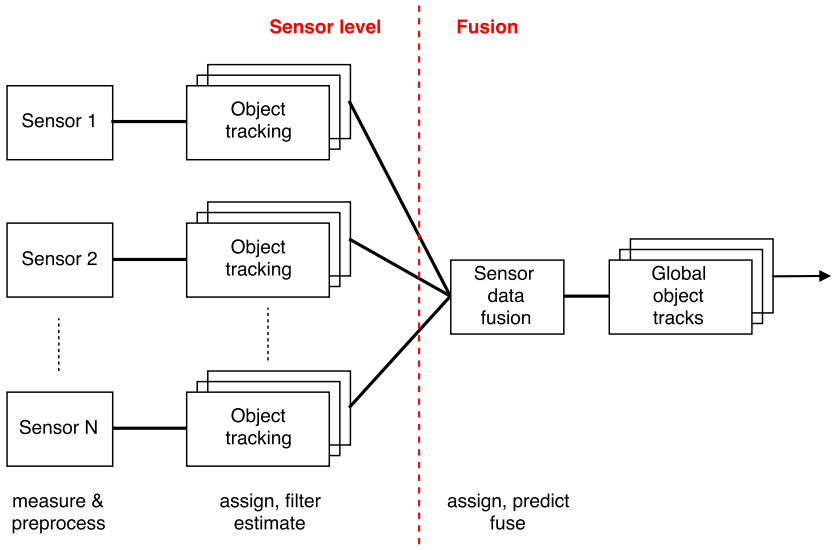

Figure 3.7: Structure of the sensor data fusion

Chapter 4

Implementation of the Sensor Data Fusion

4.1 Introduction

In this chapter the development and implementation of the algorithms for the sensor data fusion are described and explained. The theory from the previous chapter is now converted into executable code. The inputs for the programs come from a sensor model extension or directly from the simulation environment, as can be seen in figure 4.1. The two developed routines are implemented in separated Matlab Functions in Simulink.

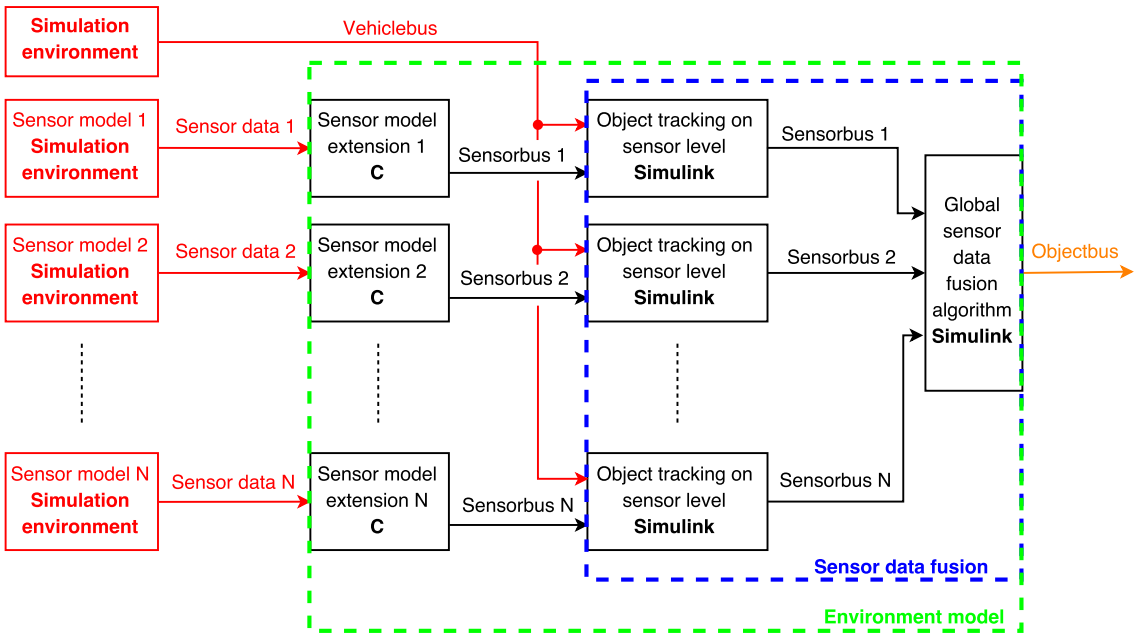

Figure 4.1: Structure of the environment model.

The fusion is a two stage process: first, the object tracking on sensor level is done. A software is developed which is able to track all detected objects in the sensor range and is called **object tracking algorithm on sensor level**. Each object has its own tracking filter. These tracks are then handed over to the second stage, the **global sensor data fusion algorithm**.

As the whole software development is done in interaction with CarMaker (simulation environment), the different user manuals [12], [16], [17] and [18] are background information for the coding. The output of the SDF is the objectbus, which has a defined structure. The bus contains the parameters of the objects which are detected by any sensor of the EV. In figure 4.1 the architecture of the complete EnvM is shown. Each block shows the name and the programming language of the corresponding routine. The interfaces to the model are marked with colors: inputs are red and outputs are orange. The sensor model extension development is already explained in chapter 2. The designing process of the other parts is described here. Every sensor has its own extension model and object tracking. Before starting with the implementation, some basic concepts have to be explained.

4.1.1 Coordinate Systems

Coordinate systems define the position of a point or an object in space. The three systems which are interesting for this thesis are all Cartesian based and shown in figure 4.2. More details about the systems used in CarMaker can be found in [12] and [17].

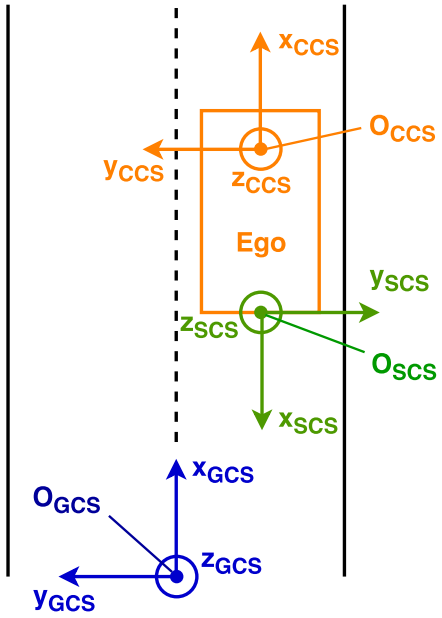

Figure 4.2: The three different coordinate systems used in the thesis.

Global Coordinate System (GCS): This system has a fixed origin in the virtual world of CarMaker and is situated on the horizontal surface of the road. O_{GCS} is the origin and it is placed in the middle of the street, like in figure 4.2. The symbols x_{GCS}, y_{GCS} and z_{GCS} are the three axis. Since it is a left handed coordinate system, z is directed upwards.

Sensor Coordinate System (SCS): The origin O_{SCS} of this system is situated where the sensor is mounted on the vehicle body. The x_{SCS} coordinate points in forward direction from sensor point of view, y_{SCS} points left from x direction and z_{SCS} is directed upwards. In figure 4.2 the sensor is placed in the middle of the back side with a FoV behind the vehicle.

Car Coordinate System (CCS): This system is fixed to the EV, which means it performs all translative and rotatory driving movements of the vehicle. The x_{CCS} coordinate points in forward driving direction, y_{CCS} is pointing to the left and z_{CCS} shows upwards. The origin O_{CCS} of the system can be set arbitrary on the car. In the thesis, the origin is set to the middle of the front axle. All algorithms introduced later use the CCS.

4.1.2 Object Model

The object model describes the dynamic driving movements of the traffic members surrounding the EV. There are many different ways to model cars depending on the usage. ESP and ABS work with detailed dynamic vehicle models [19]. Target tracking applications reduce the object to a point moving on a path. The model used in the thesis is based on CarMaker and the sensors. State of the art driving assistance sensors are able to deliver the distance to the objects in x and y direction, the velocity of the objects and sometimes additionally an estimation for the size of the object (object classification). In this thesis, a point model for the objects is chosen. This means the whole car is concentrated to one point. The point model is taken because for the other models, more information about the object like for example the mass or the geometry is needed and such details are not available at the sensors.

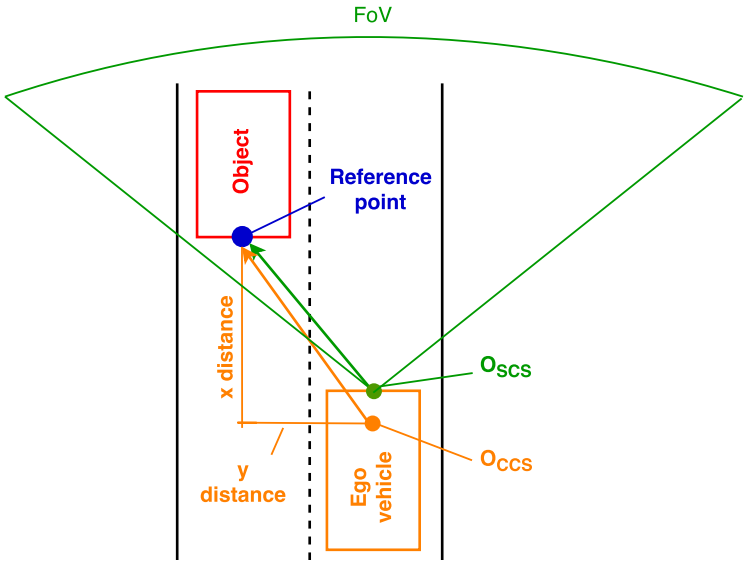

Figure 4.3: Illustration of the sensor setting reference point.

In CarMaker, the traffic objects are modelled as rectangular and the sensors measure the distance to and the velocity of the objects. There are several settings possible to which point of the target the sensor will determine its measurement values, see [12]. The method used for all sensors in the thesis is called **reference point**. By this option the middlepoint of object's back side is taken, as can be seen in figure 4.3. The distances and velocities which come from the sensors are

all in SCS. Since the total EnvM is calculated related to the car based coordinate system, the distances and speeds from the sensors have to be transformed into it. In other words the gaps used in the thesis are the difference between the O_{CCS} and the **reference point** of the object. The velocity is the difference between the speed of the EV and the object. If, for example, the EV is driving at $50\,km/h$ and the object at $70\,km/h$, the velocity in CCS is $20\,km/h$.

Now that the background for using a point model for the object has been explained, the equations which describe the dynamic behaviour are presented. For more information about the kinematic equations please see [14] and [20]. The movement of a point in time on a path with a constant translational acceleration in one direction is specified with

$$s(t) = s_0 + v(t)\ t - \frac{a\ t^2}{2}. \tag{4.1}$$

Symbol s_0 is the initial position of the point, $v(t)$ is the velocity at a specific point in time t and a is the acceleration. Converting the formula from the continuous time to the discrete time leads to

$$s_k = s_{k-1} + v_k\ t_{Sample} - \frac{a_k\ t_{Sample}^2}{2}. \tag{4.2}$$

The sample time is the time difference between two time steps $t_{Sample} = t_k - t_{k-1}$. With the common vehicle accelerations and sample times of the sensors, the term $\frac{a_k\ t_{Sample}^2}{2}$ gets negligible small and is removed in order to save computing time, see table 4.1. Additionally the acceleration of the objects is not available as measurement and it would be necessary to have it calculated by an algorithm.

0 to 100 km/h in time	**Acceleration**	**Sample time**	**Term result**
$3\,s$	$9.26\,m/s^2$	$0.001\,s$	$0.0046\,mm$
$12\,s$	$2.31\,m/s^2$	$0.001\,s$	$0.0012\,mm$
$3\,s$	$9.26\,m/s^2$	$0.01\,s$	$0.46\,mm$
$12\,s$	$2.31\,m/s^2$	$0.01\,s$	$0.12\,mm$
$3\,s$	$9.26\,m/s^2$	$0.1\,s$	$46\,mm$
$12\,s$	$2.31\,m/s^2$	$0.1\,s$	$12\,mm$

Table 4.1: Table confirms that the acceleration term can be removed from equation 4.2, compare results in column three. The values of the acceleration are chosen to go from sportive to common cars and related to [2] and [3]. The sample times cover the range of typical sample rates of sensors used for environment detection.

It is suggested that the velocity stays constant over one sample step $v_k = v_{k-1}$. This assumption reduces the equation to

$$s_k = s_{k-1} + v_{k-1}\ t_{Sample}. \tag{4.3}$$

Replacing s through x leads to the motion equations of the object model used in the thesis for the x direction

$$\begin{aligned} x_k &= x_{k-1} + v_{x_{k-1}}\ t_{Sample} \\ v_{x_k} &= v_{x_{k-1}}. \end{aligned} \tag{4.4}$$

For the y direction, the same equations are valid, just substitute x with y. To get a more compact form, the model can also be written in matrix notation

$$\begin{bmatrix} x_k \\ v_{x_k} \\ y_k \\ v_{y_k} \end{bmatrix} = \begin{bmatrix} 1 & t_{Sample} & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & t_{Sample} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{k-1} \\ v_{x_{k-1}} \\ y_{k-1} \\ v_{y_{k-1}} \end{bmatrix}. \tag{4.5}$$

When using letters for the matrices, the equation gets even more compact

$$\mathbf{x_k} = \mathbf{A}\ \mathbf{x_{k-1}}. \tag{4.6}$$

Symbol $\mathbf{x_k}$ is the current state variable of the current time step, $\mathbf{A}$ is the system matrix and $\mathbf{x_{k-1}}$ is the state variable of the previous time step. The state variable consists of the following parameters:

$$\mathbf{x_k} = \begin{bmatrix} x_k \\ v_{x_k} \\ y_k \\ v_{y_k} \end{bmatrix},$$

which describe the position and velocity of the object in x and y direction.

4.1.3 Conclusion and Boundaries of the Object Model

Based on the object model the dynamic movements of the targets will be tracked. If measurements are available, the achieved performance is satisfying. The model has a weakness when no new data is available from the sensors. Then the state variable is predicted based on the object model. Since it is assumed that the velocity remains constant, the whole forecasting is done with the last available speed measurement. When the velocity of the target changes, the algorithm is not able to track the object correctly. Considering the small prediction time the resulting difference between input and track can be tolerated. A possibility to reduce the aberration between the real movements and the tracked motion when there are no measurements is to include the acceleration in the model, like in equation 4.1. Another opportunity is to consider the acceleration of the EV and then correct the state variable.

The SDF algorithm has to be built in a modular form according to the requirements, see section 3.4.1. Therefore it is better to use the same object model for all tracking routines on sensor level. This reduces the steps needed for adding sensors to or removing them from the network, which is an advantage. If a different model is taken for each sensor the time for changing the amount of sensors in the network would dramatically increase. Another advantage of the modularity is that it can check and simulate different sensor configurations. Configuration means in this case: the number of sensors fixed on the vehicle body, the position where the sensors are mounted and the types of sensors which are used. A disadvantage of the modularity of the system is that the models are universal and not fitted to the speciality of each sensor. The object track algorithm must be designed that it can be used for different kinds of sensors with unequal sample rates.

Assumptions of the Object Model

For a faster and easier understanding, the assumptions of the object model are summarized as follows:

- All algorithms are working in the car based coordinate system (**CCS**).
- All sensors are set to the measurement method **reference point**.
- Distances and velocities are the difference between the object's reference point and the origin of car coordinate system.
- A point model is used as object model.
- The velocity remains constant over one sample step.

4.1.4 Used Buses

The algorithms are connected via buses holding the information they have calculated, illustrated in figure 4.1. There are three different buses used, the **vehiclebus**, the **sensorbus** and the **objectbus**. The buses are introduced to define the interfaces between parts of the software and to guarantee that each section has the current and same information of the environment objects. The signals of the used buses are listed below. It is, of course, possible to add or remove variables if needed.

Vehiclebus

The vehiclebus contains information about the EV, as shown in table 4.2.

Parameter	Description	Unit
v	Velocity of the EV	m/s
a_x	Acceleration of the EV in x direction	m/s^2
a_y	Acceleration of the EV in y direction	m/s^2
yawrate	Steering rate of the EV	rad/s

Table 4.2: Parameters of the vehiclebus.

Sensorbus

In the sensorbus there is information about the objects from the sensor's point of view. This bus is used to transfer sensor data regarding objects from sensor level to fusion level. The bus is hierarchically structured. The number of objects in the sensorbus can be defined and must have the same value as defined in the parameter **number of objects** (for more information see 4.2.1). Note that each sensor used in the EnvM has its own bus, holding several objects.

Parameter	Description	Unit
N	Number of objects detected by the sensor	-
Update	Flag that shows when there is new sensor data available	-
ID	ID of the sensor	-
Object 0	Substructure object data sensorbus of object 0	-
Object 1	Substructure object data sensorbus of object 1	-
⋮	⋮	⋮
Object N	Substructure object data sensorbus of object N	-

Table 4.3: Parameters of the sensorbus.

In table 4.4 the parameters of the substructure of the sensorbus are shown.

Parameter	Description	Unit
ID	ID of the object. Shows when the object is detected by the sensor otherwise it is -1.	-
MeasurementTimePoint	Time point when the measurement was taken	s
TrackActive	Flag that shows when the track is active. Value: Object ID == track active; -1 == track not active.	-
x	Distance from the EV to the objects reference point in x direction in **CCS**	m
y	Distance from the EV to the objects reference point in y direction in **CCS**	m
z	Distance from the EV to the objects reference point in z direction in **CCS**	m
v_x	Difference velocity from the EV to the object in x direction in **CCS** (longitudinal velocity)	m/s
v_y	Difference velocity from the EV to the object in y direction in **CCS** (lateral velocity)	m/s
v_z	Difference velocity from the EV to the object in z direction in **CCS**	m/s
alpha	Angle between the EV and the object in **CCS**	rad
yawrate	Yaw rate of the EV in **CCS**	rad/s
NSensor	ID of the sensor for number of sensors calculation	-

Table 4.4: Parameters of the object data within the sensorbus.

Objectbus

The objectbus is the output of the EnvM, which means it is the defined interface to other algorithms. In this bus there is all the information regarding the detected targets. This bus also has a hierarchical layout and contains a object data sublayer.

Parameter	Description	Unit
N	Number of objects detected	-
SampleTime	Sample time of simulation environment	s
Object 0	Substructure object data objectbus of object 0	-
Object 1	Substructure object data objectbus of object 1	-
⋮	⋮	⋮
Object N	Substructure object data objectbus of object N	-

Table 4.5: Parameters of the objectbus.

Table 4.6 shows the parameter of the substructure of the objectbus.

Parameter	Description	Unit
TrackActive	Flag that shows when the track is active. Value: Object ID == track active; -1 == track not active.	-
ID	ID of the object. Shows when the object is detected by any sensor otherwise it is -1.	-
Type	Defines the object type. Value: -1 == not valid; 0 == not defined; 1 == passenger car; 2 == truck; 3 == motorcycle; 4 == pedestrian; 5 == bus.	-
Width	Width of the object	m
Height	Height of the object	m
Bright	Bright of the object	m
x	Distance from the EV to the objects reference point in x direction in **CCS**	m
y	Distance from the EV to the objects reference point in y direction in **CCS**	m
z	Distance from the EV to the objects reference point in z direction in **CCS**	m
v_x	Difference velocity from the EV to the object in x direction in **CCS** (longitudinal velocity)	m/s
v_y	Difference velocity from the EV to the object in y direction in **CCS** (lateral velocity)	m/s
v_z	Difference velocity from the EV to the object in z direction in **CCS**	m/s
a_x	Difference acceleration from the EV to the object in x direction in **CCS** (longitudinal acceleration)	m/s^2
a_y	Difference acceleration from the EV to the object in y direction in **CCS** (lateral acceleration)	m/s^2
a_z	Difference acceleration from the EV to the object in z direction in **CCS**	m/s^2
alpha	Angle between the EV and the object in **CCS**	rad
yawrate	Yaw rate of the EV in **CCS**	rad/s
NSensor	Number of sensors which have detected the object	-
Sensor ID	ID's of the sensors who have detected the object	-
yLane	Current y distance to the object from the middle line of the street.	m

Table 4.6: Parameters of the object data within the objectbus.

4.2 Object Tracking Algorithm on Sensor Level (OTASL)

The algorithm is implemented as a Matlab Function in Simulink. For reasons of clarity no pieces of code are shown, instead, flowcharts and examples are used to get an idea of the routine. The inputs are the sensorbus, the vehiclebus and simulated measurement noise. The output is the modified sensorbus, see figure 4.4. Modified means in this case that the distances (x, y) and velocities (v_x, v_y) are adapted. According to whether or not the object is detected, the **measurement time point** and the **track active flag** are set in the sensorbus. In a disturbance simulation, measurement noise at the state variables is added with normally distributed white noise. For each state variable, an own noise can be defined. The OTASL is executed with the sensor sample rate which can be different for each sensor.

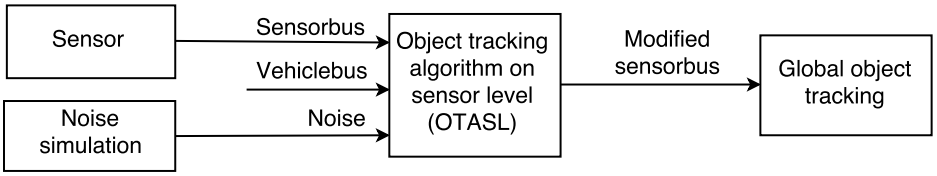

Figure 4.4: Illustration of the object tracking algorithm on sensor level.

Assistance systems usually use state prediction based on measurements from the sensor to calculate the position and velocity of the other traffic members. Different types of Kalman Filters are often used in practice, since they can be implemented and tuned easily. A further option would be a particle filter, which is more precise but also has a higher calculation and implementation effort. More information to this topic can be found in [4] and [21]. In this thesis, a Kalman Filter is used for the object tracking on sensor level. Since it is a dynamic state predictor, the object model from section 4.1.2 can be used. Every object in the sensor range has its own tracking filter.

The algorithm is designed according to the requirements from section 3.4.1 and figure 4.5 shows its structure. The first execution step of the program initializes all variables. Then depending on **IF** there is new sensor data available or not, the software either checks **IF** there is an active object track or confirms to the section **finish algorithm**. If there is a new measurement available and an object track is detected, the program jumps to the **track managing part**. When there are no active tracks, the next section is **write to bus**. The **track management** checks if the incoming data belongs to an already active track or if it has to initialize a new track. Then the **Kalman Filter** is calculated, doing a **correction of the state variables**. If the EV is changing its velocity or it is steering, the state variables are corrected accordingly. The improved state vector is handed over to the **Kalman Filter for each object**. Here the estimated tracks of the objects are determined. Now the new coordinates and velocities are written to the bus and the algorithm finishes for this execution step. In the next step, the routine starts at the section **IF** sensor data is available or not. Depending on this decision, the algorithm works through the according steps. The single steps are described in detail in the following subsections.

4.2.1 Adjustable Parameters of OTASL

The meaning and setting possibilities of the parameters of the OTASL are explained in the following subsections.

- **Number of Objects:** This parameter defines the number of objects the sensor is able to handle. The number of objects must be the same for all sensors used in the network. All sensors use the predefined structure of the bus. It is important that the defined amount of objects is the same in the algorithm and in the sensorbus.

- **Tracking Samples:** This parameter defines for how long the routine tracks an object when there is no measurement available. If there is no new sensor data for the object, a counter is started. The counter is raised in the next execution step if no measurement is available. However, if there is a measurement, the counter is reset to zero. When the counter reaches the value specified in the tracking samples parameter, the object track will be deleted. For example tracking samples is set to 50 and the sensor has a sample rate of $0.04\,s$, then the maximum time the target would be tracked is 50*$0.04\,s$=$2\,s$. Please note when setting this parameter: If the sensor has a slow sample rate, do not make the tracking samples too long otherwise the target will be tracked a long time without sensor data.

- **Threshold Longitudinal Correction:** The idea behind this parameter is that if there is a noise on the longitudinal acceleration a_x, the correction should not become active because of this disturbance. The improvement should only work when the EV is really accelerating or braking and not because there is a noise on the signal. If the parameter is set to $0.1\,m/s^2$, the correction only gets active when: $|a_x| > 0.1\,m/s^2$. Increasing the value of the parameter will increase the robustness against disturbances and decrease the sensitivity of the correction. Decreasing the threshold of the longitudinal correction will lead to a decrease of robustness against noise and increase sensitivity.

- **Threshold Steering Correction:** The idea behind this parameter is that when there is a disturbance on the yaw rate $\dot{\Theta}$, the correction should not become active because of this noise. The improvement should only work when the EV is really steering and not because there is a disturbance on the signal. If the parameter is set to $0.02\,rad/s$, the correction only gets active when: $|\dot{\Theta}| > 0.02\,rad/s$. Increasing the value of the parameter will increase the robustness against noise and decrease the sensitivity of the improvement. Decreasing the threshold of the steering correction will lead to a decrease of robustness against noise and increase the sensitivity.

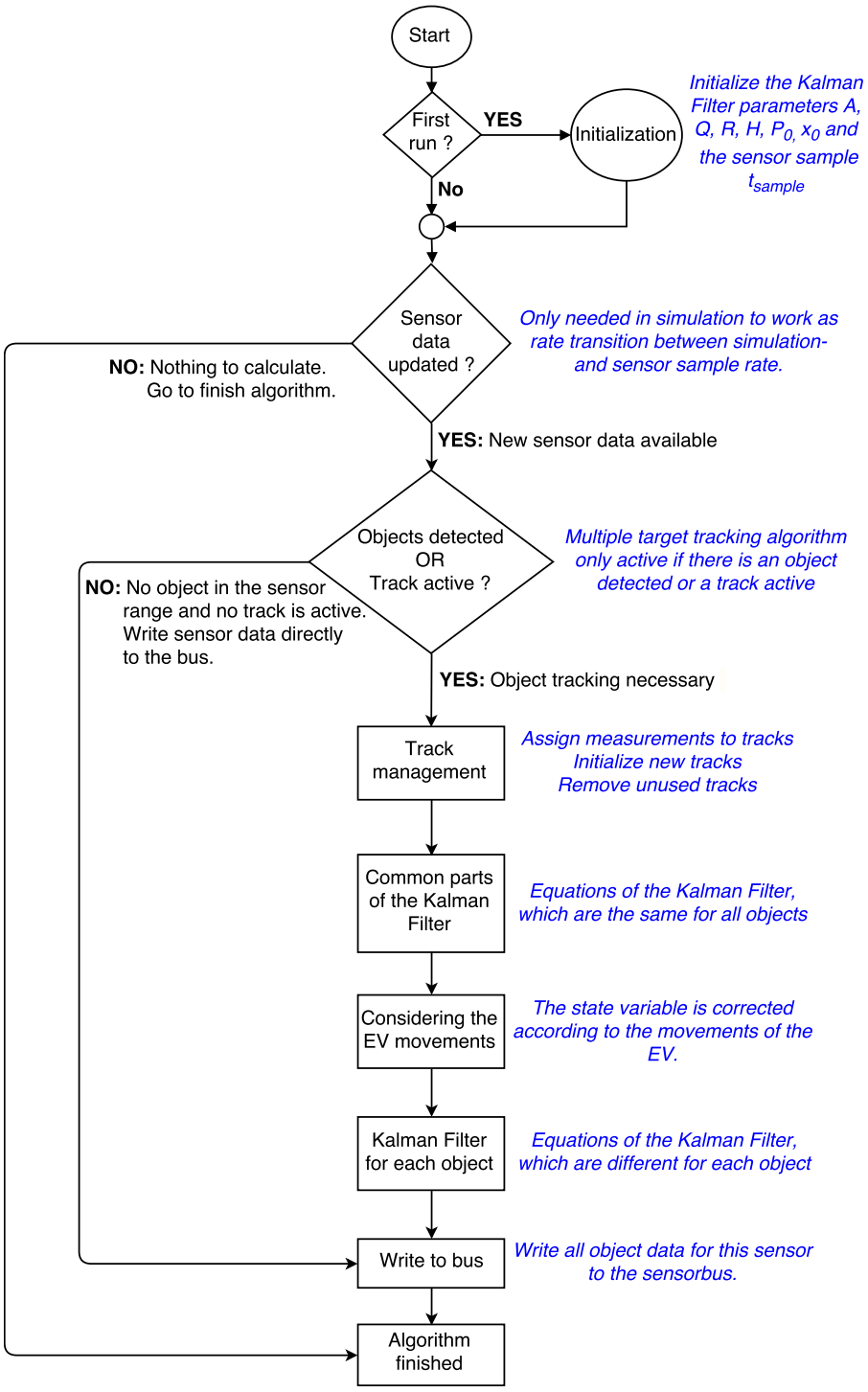

Figure 4.5: Flowchart of the multiple target tracking algorithm on sensor level. Execution is done repeatedly for each time step where new data is available. Normally this is done for each sensor using a fixed constant sample time.

4.2.2 Initialization of the Algorithm

The first part of the algorithm is the initialization of the variables and the Kalman Filter, where the system matrix (**A**), the covariance matrix of the model failure (**Q**), the covariance matrix of the measurement noise (**R**) and the measurement matrix (**H**), and the sensor sample time used in the program are set. Note that this part of the software is only iterated once when the routine is executed for the first time. The rest of the algorithm will be processed every execution step depending on if the corresponding conditions are true.

Determination of the Sensor Sample Time

The program is able to determine the sample rate of the sensor. Therefore the signal update of the sensorbus is needed. Every iteration step the signal update is raised about the sensor sample time. The software builds the difference of the update signal between the first and second execution step, which gives the sample time of the sensor. The advantage of this automatically determined time step is that the algorithm gets it directly from the simulation and the user does not have to set it additionally in the routine, which can be easily forgotten. A disadvantage of this method is that the working process starts at the second time step because the first sensor step is needed for determining the sample rate.

4.2.3 Rate Transition in Simulation

The simulation environment is working with a different sample time as the real sensor. To guarantee deterministic time behaviour of the sensor simulation, the multiple target tracking algorithm has to be executed only to the sensor sample time points. Therefore an **IF** decision (**sensor data updated?**) is implemented in the routine. If there is new sensor data available on the sensorbus (**YES**), the object tracking algorithm becomes active. Otherwise the old values are held on the sensorbus (**NO**), as can be seen in figure 4.5. Note that this rate transition is only necessary in the simulation.

Let us assume that there is new sensor data available, which does not automatically means that there is also an object in the sensor FoV. The next step is to check if there is an object detected or an track active (**objects detected OR track active?**). If one of these conditions is fulfilled, the algorithm saves the time point (when new data is available) to the signal **measurement time point** of the sensorbus and goes to the next part **track management**. The time point of the measurement is needed later on in the global SDF. All objects which are detected or tracked by the sensor then have the same time point. When no condition is true, the program jumps to **write to bus**.

4.2.4 Track Management

In this subsection, first the tasks of track management are summarized and then their implementation in the program is described. The track management has the following tasks:

- Associate the measurements to the tracks
- Initialize new tracks
- Delete unused tracks

Association of the Measurements

At the beginning of the program it is checked if an object already has an active track or if it is new. Therefore the algorithm compares the ID's of the already detected objects with the ID of the

new measurement. If there is an object found with the same ID, the measurement belongs to an existing track and the program jumps to the next section **common part of the Kalman Filter**. If the measurement is not assigned to an track, the algorithm goes to the next part **Initialize a new Track**.

Initialize a New Track

If an object is detected for the first time, it will be **initialized** in the so-called object track list. In this list the object ID and track active flag are stored. So if the item is recognized for the first time, the object ID and track active flag are set, as shown in table 4.7. In the table, the objects with ID 0, 1 and 3 are active. The size of the list defines how many objects can be tracked by the routine. The size is adjustable with the parameter **number of objects**, in this example the algorithm can handle five traffic members. Note that the position of the target in the object list depends on its ID. The object with ID = 0 will always be in the first row of the list, followed by object with ID = 1 and so on. Also the state variables **x** of the initialized objects are set to the measurement values **z**. This step has the big advantage that there is no transient effect by the Kalman Filter. When the item is recognized for the first time, the signal **track active** of the sensorbus is set to the object ID. The signal **ID** of the sensorbus shows when an object is detected (measured) by the sensor while the signal **track active** is set as long as the object is tracked. The algorithm has the ability to predict the track of the object over an adjustable amount of time when there is no measurement available. This means that the signal **track active** can still be set while the signal **ID** is already reset to default value.

Object ID	Track Active Flag
0	1
1	1
-1	-1
3	1
-1	-1

Table 4.7: Example for an initialized object track list.

Delete unused Tracks

Imagine the following situation: There is no new measurement and the object track is still active. In this case the signal **ID** of the sensorbus has the default value while the signal **track active** is still set to the object ID and a counter starts. If there is still no data available in the next execution step, the counter is increased by one. The counter is increased as long as there is a current measurement available or the counter has reached the threshold to delete the object track. The threshold of the counter can be set with the parameter **tracking samples**. If one of these cases happens the software behaves differently. If there is a new measurement on the bus, the counter is set to zero and the track remains active. However, if the counter has reached the threshold, the object track will become inactive and it is reset. Also, the corresponding element in the object track list is reset to default values. This means that the object ID gets the value -1 and the track active flag is set to 0. In table 4.8 the object with ID 1 is deleted. Also, the signal **track active** in the sensorbus is reset to its initial value (-1). Now both signals **ID** and **track active** of the sensorbus have the default values. The next section of the algorithm is the **common part of the Kalman Filter**.

Object ID	Track Active Flag
0	1
-1	-1
-1	-1
3	1
-1	-1

Table 4.8: Object with ID 1 is deleted from the list.

4.2.5 Kalman Filter for Linear Models

This subsection gives an introduction to the Kalman Filter for linear models. The Kalman Filter consists of five equations. Literature on this topic can be found in [21] and [22]. Calculation is processed in the following order:

1. **Prediction of the state variable:** The estimate of the previous time point $\hat{\mathbf{x}}_k$ is used as input and the prediction of this at the current time point $\hat{\mathbf{x}}_{k+1}^-$ is returned as result:

$$\hat{\mathbf{x}}_k^- = \mathbf{A}\ \hat{\mathbf{x}}_{k-1}. \tag{4.7}$$

   This equation reflects the dynamic object model from section 4.1.2 with $\mathbf{A}$ being the system matrix.

2. Step two calculates the **prediction of the error covariance:**

$$\mathbf{P}_k^- = \mathbf{A}\ \mathbf{P}_{k-1}\ \mathbf{A}^T + \mathbf{Q}. \tag{4.8}$$

   $\mathbf{P}_k$ is the estimate error covariance of the previous time point and $\mathbf{Q}$ is the covariance matrix of the model noise.

3. The **kalman gain** is not constant but updated each time step through the following formula:

$$\mathbf{K}_k = \mathbf{P}_k^-\ \mathbf{H}^T\ (\mathbf{H}\ \mathbf{P}_k^-\ \mathbf{H}^T + \mathbf{R})^{-1}. \tag{4.9}$$

   The symbol $\mathbf{H}$ is the state to measurement matrix and $\mathbf{R}$ is the covariance matrix of the measurement noise.

4. In this step, **estimation of the state variable** is computed:

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_k^- + \mathbf{K}_k\ (\mathbf{z}_k - \mathbf{H}\ \hat{\mathbf{x}}_k^-). \tag{4.10}$$

   The estimate is obtained from the sum of the prediction $\hat{\mathbf{x}}_k^-$ and the measurement ($\mathbf{z}_k$) with appropriate weighting.

5. In the last step the **estimation of the error covariance** is determined:

$$\mathbf{P}_k = \mathbf{P}_k^- - \mathbf{K}_k\ \mathbf{H}\ \mathbf{P}_k^-. \tag{4.11}$$

   The error covariance is a measure for the accuracy of the state estimate. If $\mathbf{P}_k$ is large, then the failure of the estimate is large, and if the error covariance is small the failure of the estimate is small as well.

Setting the Matrices Q and R

In Kalman Filter, the variance of model and measurement noise are the only things needed to be known since the noises are assumed to be normally distributed and uncorrelated. The matrices **Q** and **R** are diagonal matrices with the noise variance in the main diagonal. The values for the measurement variances can be obtained from the sensor data sheet or, if not available, they must be calculated from measurements. The variances for the model noise are set based on expertise of the object model. Changing the value of those two matrices influences the performance of the filter. If **Q** is increased, the Kalman gain increases and the measurement values are stronger contributed in the estimation process. Decreasing **Q** will reduce the variation of the estimate and the influence of the measurement. The matrix **R** influences the filter in the opposite way. Those matrices are design factors for the Kalman Filter.

4.2.6 Common Parts of the Kalman Filter

For the implementation the filter is split into two parts: **Common parts of the Kalman Filter** and **Kalman Filter of each object**. The reasons for doing this are highlighted in this subsection. In the software, the sequence of the equations which describe the filter is changed in order to save computing time. In the order in which the formulas are shown here, the functionality of the Kalman Filter is easier to understand and explain. In the section **common parts of the Kalman Filter**, the following three equations 4.8, 4.9 and 4.11 are calculated. The reason why the equations are the same for all objects is that they all use the same object model. Since they are all getting the data from the same sensor the measurement noise is for all objects equal. There is no need to determine these equations separately for each target. The equations 4.7 and 4.10 will be included in the routine later on.

4.2.7 Considering the Ego Vehicle Movements

The idea of considering the EV movements can be explained as follows: Since the velocity and the yaw rate (steering) of the vehicle are known, this information can be used to improve the state vector in advance. If the state variables are updated before they are handed over to the Kalman Filter, the filter does not have to identify this from the measurements. So the performance of the filter can be improved.

There are two different corrections:

- Longitudinal correction (in cooperation with the EV acceleration)
- Steering correction (in cooperation with the EV steering)

Longitudinal correction: Imagine this situation: Another car is driving with a constant velocity in front of the EV. Now the driver of the EV pushes the accelerator pedal and the distance to the front vehicle becomes smaller. When the driver is braking, the distance to the objects gets bigger. These are simple cases but when the car in front of the EV also changes its velocity, it gets more difficult. Then the following situation can happen: the driver of the EV is increasing the speed, but the vehicle in front is accelerating even stronger and the distance becomes bigger. To cover such cases, it is required to take the difference velocity between EV and the car which is determined with this equation [12]

$$v_{k_{uncorr}} = v_{k_{Car}} - v_{k_{Ego}}. \tag{4.12}$$

For the longitudinal correction, the following assumptions are made:

- The velocity of the car is assumed to remain constant. The symbol $v_{k_{uncorr}}$ is the estimated difference velocity in CCS from the previous calculation step, $v_{k_{Car}}$ is the absolute speed of the object in GCS and $v_{k_{Ego}}$ is the absolute velocity of the EV in GCS.
- A change in the EV's velocity directly influences the difference velocity, since the speed of the object is assumed to be constant ($a_{k_{Car}} = 0$). This leads to the following condition: $a_k = a_{k_{Car}} - a_{kEgo} \Rightarrow a_k = -a_{kEgo}$.

The next step is to correct the difference distance and speed in x direction according to the contribution of the EV longitudinal acceleration $a_{k_{Ego}}$. Multiplying the acceleration with the sensor sample time gives the contribution of the change of the difference velocity

$$\Delta v_k = -a_{k_{Ego}}\ t_{Sensor}. \tag{4.13}$$

The difference velocity is corrected with the change value

$$v_{k_{corr}} = v_{k_{uncorr}} - a_{k_{Ego}}\ t_{Sensor}. \tag{4.14}$$

The algorithm is working with difference velocities and distances. The sign of the speed is positive if the distance between EV and objects becomes bigger, and negative if the distance becomes smaller. The corrected distance can be determined with

$$x_{k_{corr}} = x_{k_{uncorr}} - a_{k_{Ego}}\ t^2_{Sensor}. \tag{4.15}$$

This correction is only done for the x coordinate of the state variables, therefore it is called **longitudinal correction**. The improved velocity and distance are handed over to the Kalman Filter. The distance and the speed in y direction is improved with the next correction.

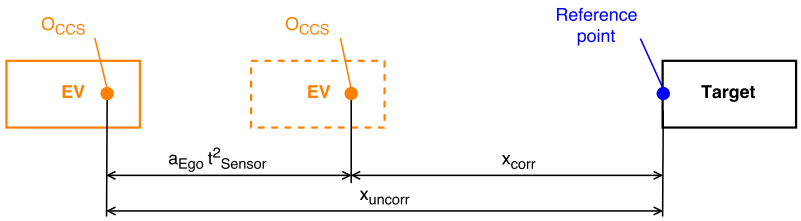

Figure 4.6: Illustration of the longitudinal correction. The distance between the target and the EV gets smaller.

Steering correction: The correction considers the information if the EV is steering. This means that if the driver is making some steering action, the state variable $\mathbf{x}$ will be corrected accordingly. For this correction the yaw rate $\dot{\Theta}$ is used. The yaw angle Θ gives the rotation between the global coordinate system and the CCS, see figure 4.7. The symbol α is the angle between the object and the EV.

For the steering correction the following assumptions are made:

- The position of the target remains constant.
- If the EV is steering, the yaw angle changes its value. Since the object stays at its position, the angle α is changing with the same rate as the yaw angle. This leads to the enclosed condition: $\dot{\Theta} = -\dot{\alpha}$.

How does the correction work in detail? With the help of the tangential velocity v_{T_k}, the corrected difference velocity between the object and the EV will be calculated. The tangential velocity is determined using the equation:

$$v_{T_k} = r_k \ \dot{\Theta}_k. \tag{4.16}$$

Before the tangential velocity can be defined, the absolute value of the difference distance r_k needs to be calculated

$$r_k = \sqrt{x_k^2 + y_k^2}. \tag{4.17}$$

The next step is to split the tangential speed into the two coordinates (x and y) of the CCS. Therefore the angle α to the target is determined

$$\alpha_k = \arctan\ (\frac{y_k}{x_k}). \tag{4.18}$$

Now the velocity can be separated into an x and y part. X component of the tangential velocity

$$v_{T_x},_k = v_{T_k} \ \sin\ \alpha_k \tag{4.19}$$

and y component of the tangential velocity

$$v_{T_y},_k = v_{T_k} \ \cos\ \alpha_k. \tag{4.20}$$

Since the yaw angle is changing, the angle α has to be corrected to determine the improved x and y components of the distance with the result. The correction is done with the following equation:

$$\alpha_{k_{corr}} = \alpha_k - \dot{\Theta}_k \ t_{Sensor}. \tag{4.21}$$

With the improved angle $\alpha_{k_{corr}}$, the changed distances (= $x_{k_{corr}}$ and $y_{k_{corr}}$) for both components can be calculated:

$$x_{k_{corr}} = r_k \ \cos\ \alpha_{k_{corr}}, \tag{4.22}$$

$$y_{k_{corr}} = r_k \ \sin\ \alpha_{k_{corr}}. \tag{4.23}$$

Now the **steering correction** of the state variable can be calculated, see figure 4.7. By the steering correction all state variables are corrected.

Equations of the Steering Considering of the EV

With the following formulas, the movement of the EV is considered in the state variables.

Correction of the x coordinate:

$$x_{k_{corr}} = r_k \ \cos(\alpha_k - \dot{\Theta} \ t_{Sensor}) \tag{4.24}$$

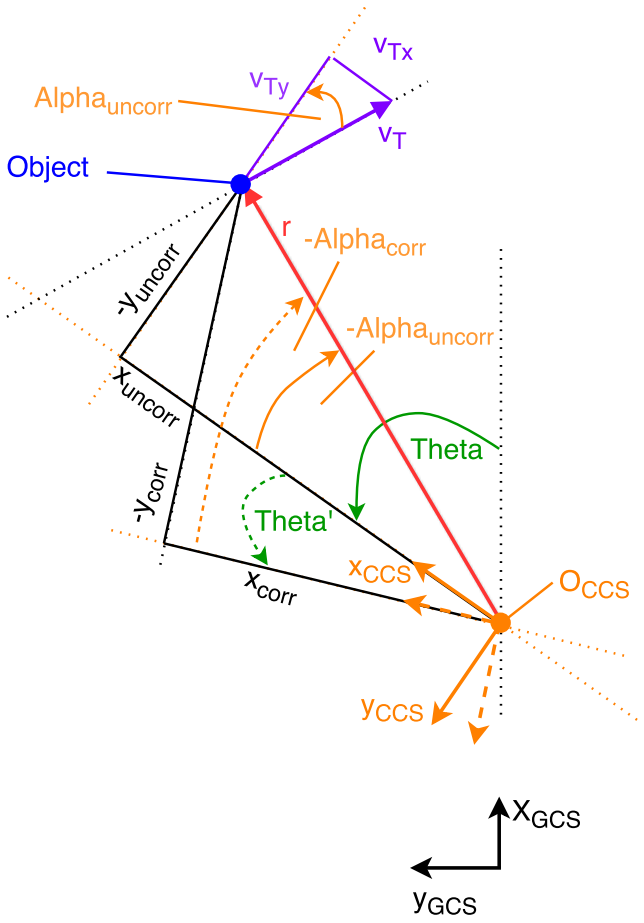

Figure 4.7: Illustration of the steering correction for distances. The EV is steering to the left and therefore the angle α is changing. The distances (x_{uncorr} and -y_{uncorr}) are improved through the corrected angle $-\alpha_{corr}$.

Correction of the x velocity:

$$v_{x_{corr},k} = v_{x_{uncorr},k} + v_{T_x,k} = v_{x_{uncorr},k} + r_k \ \dot{\Theta}_k \ \sin(\alpha_k) \tag{4.25}$$

Correction of the y coordinate:

$$y_{k_{corr}} = r_k \ \sin(\alpha_k - \dot{\Theta} \ t_{Sensor}) \tag{4.26}$$

Correction of the y velocity:

$$v_{y_{corr},k} = v_{y_{uncorr},k} + v_{T_y,k} = v_{y_{uncorr},k} + r_k \ \dot{\Theta}_k \ \cos(\alpha_k) \tag{4.27}$$

After both corrections are done, the algorithm goes to the next part, the **Kalman filter for each object**.

4.2.8 Kalman Filter for Each Object

This part of the algorithm is done in a loop separately for every object. The target standing on the first place in the object list is executed first, then the element on the second place is calculated and so on until the last object in the list is reached. The calculations are only done if the track is active, meaning there must be a value unequal to -1 in the track active column, as shown in table 4.8. In the table two object tracks are active.

For the object tracking on sensor level, the following equations 4.7 and 4.10 from subsection 4.2.5 of the Kalman Filter are used. Depending on whether or not there are measurements available for the track, the estimate of the state variable is calculated differently. When there is new sensor data, the estimate is determined with formula 4.10 otherwise the prediction of the state vector is the estimate $\hat{\mathbf{x}}_\mathbf{k} = \hat{\mathbf{x}}_\mathbf{k}^-$.

After the new position of the object is determined, the changed values have to be sent to the sensorbus. This is done in the last step of the program **write to bus**.

4.2.9 Write to Bus

Write to bus is the last section of the algorithm. Here, are two cases distinguished: If there is new sensor data available, the routine gets active and works through all the above described steps. The object variables can be modified and these changed values are written to the sensorbus. In the other case, there is no new measurement available and the values from one execution step before are held on the sensorbus.

4.2.10 Conclusion and Boundaries

The structure of the algorithm is designed that it can be used for different types of sensors. For each state variable $\mathbf{x}$ of the object model, a particular measurement noise can be simulated. Changing the values of the matrices $\mathbf{Q}$ and $\mathbf{R}$ influences the disturbance behaviour of the Kalman Filter. The algorithm is able to track the objects for an adjustable amount of samples if there is no measurement available. This time range can be set with the parameter **tracking samples**. Because of the special design of the software, there is no transient effect when a track is initialized. To improve the performance of the routine, a **longitudinal** and a **steering correction** are implemented in the algorithm. The **longitudinal correction** considers if the EV is accelerating or braking and improves the difference distance in x direction. The corrected distance is then handed over to the filter. If the EV is steering, the second correction gets active. All state parameters (x, v_x, y and v_y) are improved and afterwards given to the tracking. There is a threshold parameter (**threshold longitudinal correction** and **threshold steering correction**) for each correction to determine when it gets active. The idea behind these parameters is that the improvements should not work because of the noise that might be on the signals. The correction should only get active for accelerating, braking or steering manoeuvres of the EV. A disadvantage is, that the algorithm uses the difference distance and velocity between the EV and detected vehicles but does not consider the street course at this point. This means that with this approach, the motion of the objects is decoupled from the road course.

4.3 Global Sensor Data Fusion Algorithm (GSDFA)

This algorithm is responsible for the sensor data fusion on global level. The requirements for the algorithm are listed in section 3.4.1. The program is designed and developed according to those specifications. The implementation is done in a Matlab Function in Simulink. For reasons of clarity, no pieces of code are shown here; instead, are flowcharts and examples used to get an overview of the algorithm. The inputs to the GSDFA are the sensorbuses; the number of the buses can change and depends on how many sensors are connected to the global fusion. The structure of the algorithm has to be flexible so that the amount of sensors linked to it can be changed easily. The number of connected sensors is set with the parameter **number of sensors**. The output of the fusion is the objectbus with the global object tracks in it, see figure 4.8.

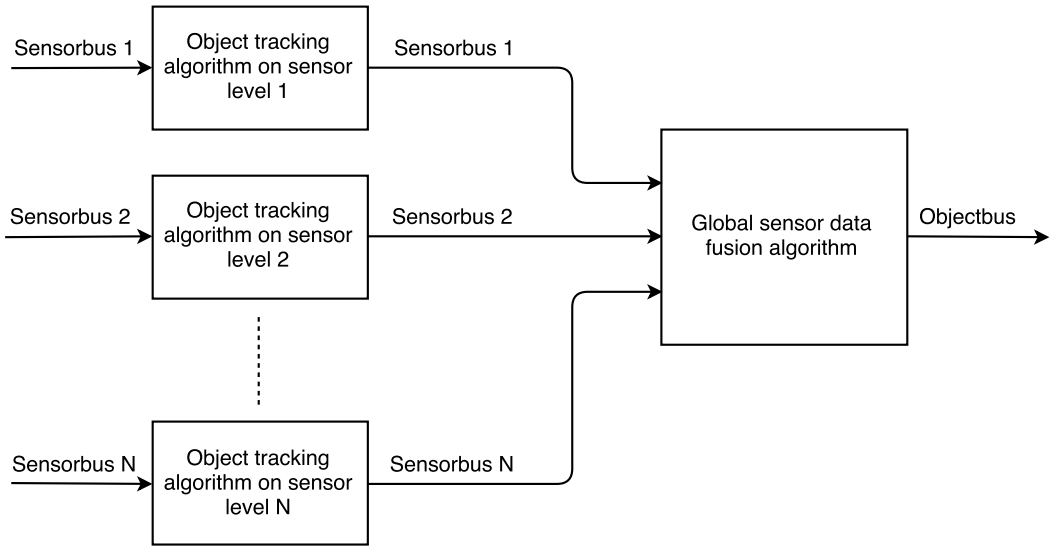

Figure 4.8: Structure of the GSDFA.

Since the sensors do not have the same sample rates, the SDF is a two stage process. In a first step the measurements from the sensors are updated to the fusion time point, can be seen in figure 4.10. In the second step the fusion takes place. Finally the object tracks from the sensorbuses are merged together to the objectbus. The global fusion works with its own adjustable sampling rate. This has the advantage that the objectbus is updated with a fixed sample rate which guarantees a deterministic time behavior.

As already mentioned, the routine consists of two main steps: the update section and fusion section. The steps themselves are divided into smaller parts. In the following, all parts of the algorithm are explained. The first part is the **initialization of the variables** where all used variables of the algorithm are set to their default values. Then the software checks **IF** the object is detected **OR** tracked from any sensor which is mounted on the vehicle body. If so, the next step is **update to fusion time point**. In case the item is not recognized from the sensor network, the algorithm continues with the section **write to bus**. In the section update, the required sensor data are refreshed to the fusion time point and the object count list is set as well. After those two steps are done, the routine jumps to **assign index**. In this step, the corresponding indexes

on which place in the sensorbus to find the track are written to the object count list. The next part, **average building**, is the heart of the program and here the SDF is done. All the paths for the same object from the sensors are fused together to one global object track. Finally in the step **write to bus**, the track is written to the objectbus. If the object is not detected or tracked by any sensor, the default values will be send to the bus. The steps described here are executed in every iteration of the global fusion.

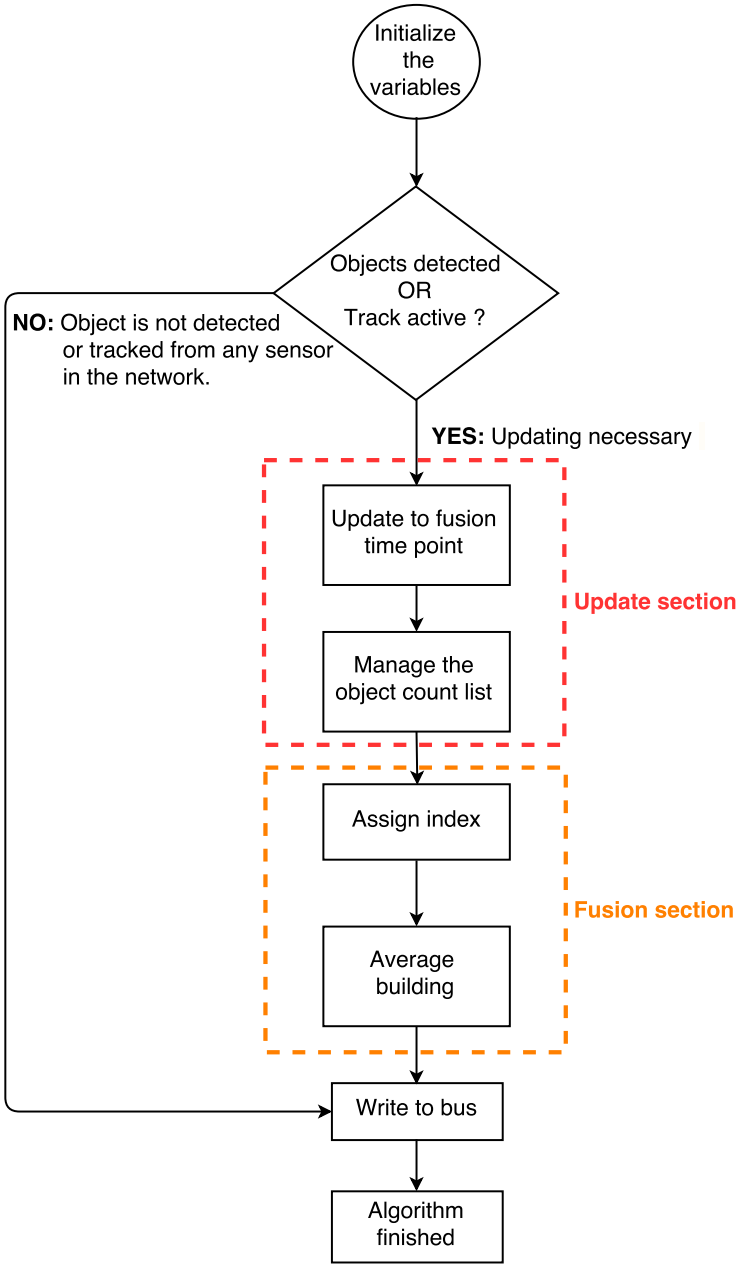

Figure 4.9: Flowchart of the global sensor data fusion.

4.3.1 Adjustable Parameters of GSDFA

- **Number of Objects:** This parameter has the same name and task as the parameter in the object tracking algorithm. For more details see section 4.2.1.

- **Fusion Sample Time:** This parameter defines the sample time of the GSDFA. The used sensors do not need to have the same sample rates. To get a deterministic time behaviour for the global fusion, a particular independent sample rate is set with the parameter. Increasing this parameter will increase the time between two execution steps of the program. This means that the algorithm will work slower. Decreasing the value of the parameter will reduce the time between two iteration steps, as a consequence, the routine will work faster. For instance, the fusion is sample time is set to 0.01 s and increasing the time to 0.02 s makes the software work slower. Reducing the sample time to 0.009 s makes the algorithm run faster.

- **Number of Sensors:** This parameter defines the amount of sensors connected to the algorithm. Since the program must be able to handle an adjustable number of sensors, this parameter is introduced. This has the advantage that the behaviour of different sensor network configurations can be tested and simulated. Increasing the parameter also increases the number of sensors linked to the routine. Decreasing the parameter reduces the amount of sensors connected to the algorithm. If, for example, the parameter value is 5, then five sensors are linked to global fusion.

- **Number of Detection Sensors:** This parameter sets the amount of sensors which can simultaneously detect or track an object. For sure the question will come up: Why is not the parameter **number of sensors** used for this? The answer is that it is hardly ever possible that all sensor which are mounted to the vehicle body can recognize the same target at once. In order to reduce the size of the object count list and memory capacity, this parameter is introduced. Increasing the value of the parameter directly raises the number of sensors that can track one object at the same time. Decreasing the parameter will reduce the amount of sensors which are able to recognize a target synchronous. For instance, the parameter value is three like in the example of section 4.3.5, which means that three sensors can simultaneously detect or track the same object. It is important to note that the value should not be too small, for then the following could happen: The value is set to two but four sensors have recognized the track. Then the safety part of section 4.3.6 gets active, the algorithm will be stopped and a warning will appear in the Matlab command window. The safety section is implemented to prevent the user from losing information about the object because the value of the parameter is too small.

4.3.2 Initialize the Variables

In this section of the algorithm, all used variables are initialized with their default values. It is important to note that this is done only once when the program is executed the first time. Here, the object count list is defined. The list consists of an adjustable amount of rows and columns. The amount of rows defines the number of tracks that the routine is able to handle, this can be set with the parameter **number of objects**. The usage of the parameter will be explained later in the section 4.3.1. In the first column, the object ID from sensorbus is written and in the second one the number of sensors that have detected or tracked the object. The additional columns define how many sensors can recognize a target simultaneously. These additional columns are also adjustable with the parameter **number of detection sensors**. The variable object list is initialized as follows: All columns beside the second one are filled with -1. The second column is set to 0 at

first.

Object ID	Number of Sensors	Index of 1st Sensor	Index of 2nd Sensor	Index of 3rd Sensor
-1	0	-1	-1	-1
-1	0	-1	-1	-1
-1	0	-1	-1	-1

Table 4.9: Object count list initialized with default values.

Table 4.9 is an example for an object count list filled with default values. In this example, **number of objects** and **number of detection sensors** are set to three. As a result, the table has three rows and five (three plus two) columns. Please note that the first row is not implemented in the code, it is just added here to facilitate the understanding.

When all variables are defined, the program checks **IF** the object is detected or tracked by any sensor which is mounted on the vehicle body. Depending on this decision, the algorithm either goes to **write to bus** or **update to fusion time point**. If the object is not recognized from the sensor network (**NO**), then the next executed section is **write to bus**. If the object is detected (**YES**), the routine jumps to the part **update to fusion time point**.

4.3.3 Update to Fusion Time Point

In this part of the algorithm, the sensor data are updated to the fusion time point. Every sensor is working with its own specific sample rate. In the fusion, information from more than one sensor is merged. Moreover, the GSDFA has a different sampling rate to the sensors. In figure 4.10, the timing behaviour of two sensors and the fusion shown as an example. To fuse data from different measurement times, it is necessary to refresh the measurements to the **fusion time point**. Otherwise, this would lead to a failure in the sensor data merging, since the objects are moving. In the tracking algorithm on sensor level, the measurement time point of each sensor is saved to the sensorbus. In the global fusion routine, the **measurement time point** is subtracted from the **fusion time point**

$$\Delta t = t_{Fusion} - t_{Measurement}. \tag{4.28}$$

The result Δt is used to update the state variables x and y of each track in the sensorbus with the object model. For more information about the model, please see section 4.1.2. In this step, only the distances are updated because the object model is not able to predict the velocities. The next step is to set the objects adjustment in the object count list.

4.3.4 Manage the Object Count List

The tasks of this part of the program are:

- Initialize new tracks
- Hold existing tracks
- Delete unused tracks

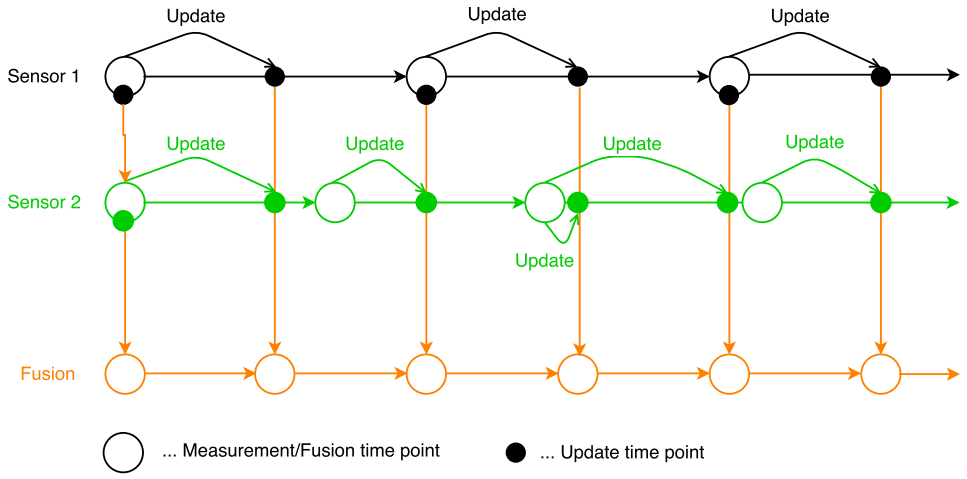

Figure 4.10: Workflow of the GSDFA.

If the algorithm is executed the first time, the object list comes directly from the initialization section and is filled with the default values, see table 4.9. Otherwise the first column of the object count list comes with the values from the previous execution step, as shown in table 4.10. In this case, the object ID symbolizes that the track is active. All other columns are reset every time step. This is done because the number and sensors who detect or track the object can be different in every iteration.

Object ID	**Number of Sensors**	**Index of 1st Sensor**	**Index of 2nd Sensor**	**Index of 3rd Sensor**
6	0	-1	-1	-1
10	0	-1	-1	-1
-1	0	-1	-1	-1

Table 4.10: Input list to the section **manage the object count list**.

Initialize a New Track

The algorithm has checked the object count list and there is no active track with this object ID, but the object is detected by a sensor. Therefore a new track must be initialized. The program uses the first free row of the list and writes the object ID in the first column and sets the second column to one. For instance, the track with ID two is initialized in the fourth row of table 4.11. The value one in the column number of sensors indicates that the object is recognized by one sensor. The tracks in the second and third row are already active from previous execution steps, which can be seen in the second column. The zeros in the second column symbolize that the track has already been active.

Object ID	Number of Sensors	Index of 1st Sensor	Index of 2nd Sensor	Index of 3rd Sensor
6	0	-1	-1	-1
10	0	-1	-1	-1
2	1	-1	-1	-1

Table 4.11: In the last row, a new track is initialized in the object count list.

The last thing to do is to set the **track active** flag of the objectbus to the value of the object ID. This flag shows that the target is detected or tracked by at least one sensor of the vehicle.

Hold the Track

If a track is already active, then it is checked in this section by how many sensors it is detected or tracked. The result is written in the corresponding row of the column "number of sensors" in the object count list. For example, in table 4.12, the track with ID six is recognized by three sensors (second row). Object ten is not detected by any sensor, which is shown by the value zero in the column number of sensors. Object two is recognized by one sensor.

Object ID	Number of Sensors	Index of 1st Sensor	Index of 2nd Sensor	Index of 3rd Sensor
6	3	-1	-1	-1
10	0	-1	-1	-1
2	1	-1	-1	-1

Table 4.12: The tracks of the object with ID 6 and 2 are based on sensors.

In all active tracks, the signal track active of the objectbus is set to the object ID, which symbolizes that the object is recognized from the sensor network.

Delete Unused Tracks

Every track which is not used any more is deleted from the object count list to make the place free for a new track. When no sensor has detected or tracked the object, the corresponding second column of the list is zero. In this case, the object ID is deleted from the first column and the default value minus one is written instead. The track is removed and in the next iteration, a new track can be assigned to the row. In table 4.13, the track in the third row is deleted and the column object ID is set to minus one.

Object ID	Number of Sensors	Index of 1st Sensor	Index of 2nd Sensor	Index of 3rd Sensor
6	3	-1	-1	-1
-1	0	-1	-1	-1
2	1	-1	-1	-1

Table 4.13: The track in the third row is deleted from the object count list.

In the next section, **assign index**, the columns up the second are filled with the indexes from the sensorbus.

4.3.5 Assign Index

In this section of the algorithm, the object count list is completed. In every row where an active track is stored, the columns starting from the third one are filled with the indexes of the object's location in sensorbus. The number of sensors that can recognized a target is adjustable with the parameter **number of detection sensors**. The value in the second column indicates how many indexes are saved. If the value of the column number of sensors higher than the parameter value, the indexes which are situated first in the sensorbus are taken. To get a better understanding, the following table shows an example. As input, the table 4.13 is used and the index columns are completed in this section. Table 4.14 illustrates how the result can look like.

Object ID	Number of Sensors	Index of 1st Sensor	Index of 2nd Sensor	Index of 3rd Sensor
6	3	2	11	14
-1	0	-1	-1	-1
2	1	5	-1	-1

Table 4.14: Object count list with assigned indexes.

Also, a safety feature is implemented in this section. When the algorithm is assigning the indexes to the columns of the list, an internal counter is started. At the end, the value of the internal counter and the column number of sensors must be the same, otherwise the averaging in the next section is not executed. In case the values are unequal, the algorithm is stopped and a warning appears in the Matlab command window. With the completed object count list, the global object tracks can be calculated in the next part.

4.3.6 Averaging

This section of the software is the fusion and the tracks of the objectbus are determined. The SDF is implemented as an averaging method. The updated to **fusion time point** distances (x, y) and velocities (v_x, v_y) of the sensors are summed up and divided through the value of the second column of the list

$$x_{Global} = \frac{\sum_{i=1}^{N} x_{i_{Sensor}}}{N}. \tag{4.29}$$

The equation is shown exemplary for distance in x direction, for all other state variables it is the same, just replace the letter x with the corresponding variable. N is the number of sensors that have detected or tracked the object, and it is stored in the second column of object count list.

As mentioned in the section **assign index**, the averaging is not executed when the value of the internal counter is unequal to the value of the second column of the object count list. If this happens, the algorithm is interrupted and a warning is displayed at the Matlab command window. When the global is fusion finished, the program comes to the last part **write to bus**.

4.3.7 Write to Bus

According to if the object is detected or tracked by any sensor of the vehicle, the modified data is sent to the objectbus. If the track is not recognized, the default values are written to the objectbus. Besides the section **initialize the variables**, all described parts of the program are executed every iteration.

4.3.8 Conclusion and Boundaries

On account of the fusion's flexible structure, it can handle an adjustable amount of sensors linked to it. The number of sensors connect to the program can be set with the parameter **number of sensors**. To guarantee deterministic time behaviour, a particular independent sample time can be set for the algorithm with the parameter **fusion sample time**. This leads to the following advantage, the behaviour of different sensor network configurations can be simulated and the results are compared. By validating the results, it is possible to say which configuration is best for an ADAS application. With the parameter **number of objects**, the amount of different objects that the routine is able to handle can be changed. Please remember that the value of this parameter has to be the equal to the value of the same parameter of the OTASL, as otherwise there would be a problem with the indexing of the sensorbus.

The software consists of two main steps and in every section, there are things which can be improved in the future. In the following, these things are explained and it is mentioned shortly how to improve them. In the update section, the sensor data is predicted with the object model to the fusion time point. Since the model is only able to predict the distances, an improvement suggestion is to expand the model so that it is able to predict the velocities. For more information, please refer to section 4.1.2.

In the fusion section the SDF is done. In this case, the merging is implemented with an averaging. This means the data of the sensors is summed up and divided through the number of sensors which have detected the object. A suggested improvement would be to weight the measurements before they are summed up. The data can be weighted according to the following points:

- **How old the latest measurement time point is:** Older measurements are not as confidential and reliable as more recent ones.

- **If the object is predicted the by tracking algorithm on sensor level:** Measured data is more precise and trustworthy than tracked data.

- **How confidential the sensor is according to accuracy and resolution:** For example it is known that cameras have a better lateral resolution than radar sensors. Therefore the radar has a better longitudinal solution than the camera. Such information can be included in the weighting to improve the fusion.

The shortcomings of the software and their respective improvement suggestions become evident in the validation chapter 5.

Chapter 5

Validation Examples

5.1 Introduction

The function development is completed and the next step is to test and validate the software. Unfortunately, there are no real measurement data available for the evaluation, so the assessment is done with simulated values. The test cases prove the implemented functionalities of the EnvM and demonstrate the performance of the algorithm. All shown experiments are done on the motorway because the implemented functions will be used for a motorway assistance system. The targets of the motorway assistant are: to control the lateral and longitudinal movement of the EV automatedly with and without other vehicles in the current lane and in relation to other vehicles. In other words, to control the distance to other traffic members, to adjust the EV's velocity if necessary and to overtake other cars autonomously. The model is also able to handle traffic situations in other surroundings.

Input to the EnvM are the traffic scenarios, which are set up in CarMaker. Information on how to define and create situations can be found in [17]. The output of the simulation are the global object tracks, which are based on the fused data from all surrounding detection sensors. In figure 5.1 the test structure is illustrated.

Figure 5.1: The test simulation environment.

According to the architecture of the algorithm, first the OTASL and then the GSDFA are checked with several examples. The following test cases are executed in this chapter:

1. **Test case for the longitudinal correction**
2. **Test case for the steering correction**
3. **Test case for the target tracking**
4. **Static test case of the global data fusion**

5. **Dynamic test case of the global data fusion**

6. **Test case for noisy input signals**

The test cases are chosen to prove the correctness of the functionalities, to support the understanding of the working process and to show the capabilities of the algorithm.

5.2 Test Cases of the Object Tracking Algorithm on Sensor Level

The following test cases in this subsection prove the functionalities of the tracking algorithm and force the understanding of its working process. First there will be examples which test the **longitudinal** and the **steering correction**. The last test case shows how the **OTASL** works. For all cases in this subsection, the same EV with the camera sensor is used. The simulations are done in CarMaker and Simulink. When there are no disturbances on the input signals, the results are easier to compare due to that the noise simulation is deactivated. At the beginning of each test case the underlying traffic scenario is described.

5.2.1 Used Ego Vehicle and Parameter Values

On the EV, a camera is mounted on the top in the middle of the windscreen, as illustrated in figure 5.2. The red point indicates where the camera is positioned.

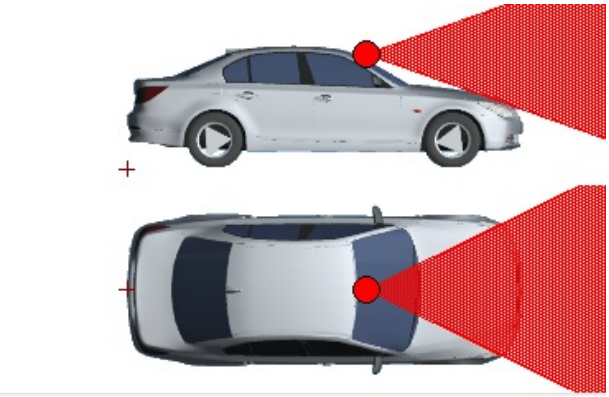

Figure 5.2: Position of the camera sensor on the vehicle.

For the examples, the following versions of the supporting programs are used: Matlab Version: R2011a 32Bits and CarMaker Version: 4.0. The used parameter values of the tracking routine on sensor level, which remain constant during all test cases are shown in table 5.1. The value of the thresholds for the **longitudinal** and **steering correction** are different for every example.

Parameter	**Values**
Sample time of CarMaker	0.001 s
Number of Objects	5
Tracking Samples	80

Table 5.1: Parameter values for the test cases of the OTASL.

5.2.2 Test Case for the Longitudinal Correction

This example demonstrates the functionality of the **longitudinal correction**. The parameter **threshold longitudinal correction** is set to $0.2\,m/s^2$. In this test case, only the longitudinal correction is tested. To turn the steering correction off, the parameter **threshold steering correction** is set to $100\,rad/s$. The test is taking in place on a straight two lane highway where the object is standing on the left lane, $130\,m$ before the EV. The EV starts on the right lane with a velocity of $1.39\,m/s$ and makes the following drive manoeuvres:

1. Accelerating up to $11.23\,m/s$ at time $4\,s$
2. Braking until it has a velocity of $1.39\,m/s$ at $8\,s$
3. Speeding up to $9.72\,m/s$ at $12\,s$
4. Reducing the velocity until standstill

The described motions of the EV can be seen in the bottom diagram of figure 5.3. In the plot, the difference velocity between the object and the EV is displayed. In this test case, the speed of the object is always zero, which explains the negative sign of the velocity in the bottom diagram. Since the **longitudinal correction** is only improving the x difference distance, the EV does not make any steering actions. That is the reason why the distance and velocity in y direction are constant and therefore those variables are not shown in the following figures.

The expected result of this test case is that the difference distance between the EV and the target gets smaller over time. The difference velocity v_x will change its value according to the driver's accelerating and braking actions. In the top diagram of figure 5.3, the object detection flag is displayed. Since the target is in the FoV of the camera during the whole example, the flag always has the value zero, which is the Object ID. In the second digram, the longitudinal acceleration of the EV over time is shown. In diagram number three and four, the difference distance and velocity in x direction are illustrated. In these two diagrams, there are always three signals displayed: the blue line is the input from CarMaker, the red signal is from the target tracking algorithm of the camera without correction and the green line is from the tracking program with **longitudinal correction**. Indicated the legend in the right corner of digram three. The main target is to follow the blue signal as good as possible. When looking at figure 5.3, most times only the green signal is visible; this is because the other signals are laying under it. As a first conclusion it can be said that the tracking algorithm follows the input signal. The next step is to check if the **longitudinal correction** is working acceptably. Therefore it is necessary to zoom into the plots of the difference distance in x direction, see figure 5.4.

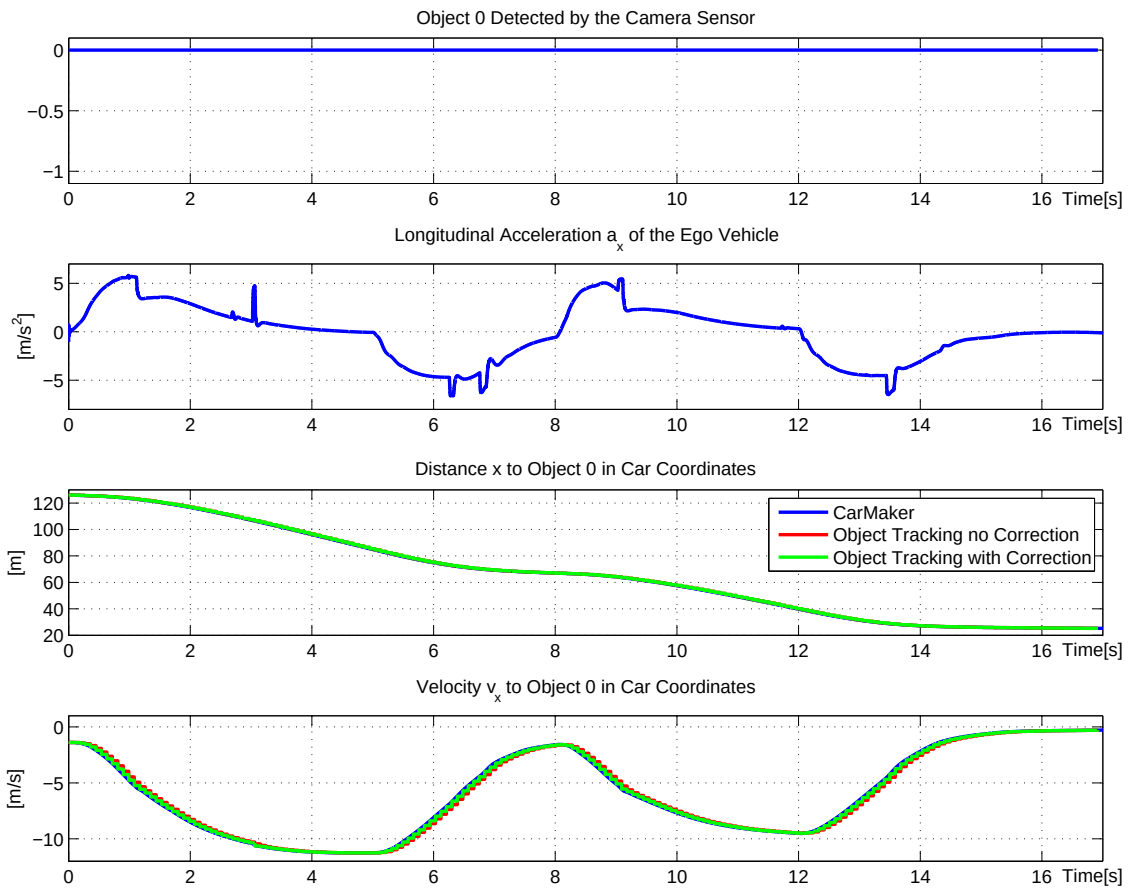

Figure 5.3: Signal sequences of the longitudinal correction.

In figure 5.4, the signals are not overlapping any more. The zoom is chosen such that one sample step of the tracking algorithm is shown. The part of the signal sequence is selected randomly; however, this can be done at any time frame of the signal the result is the same everywhere. As mentioned earlier, the blue signal comes from CarMaker, which is working with a sample time of $0.001\,s$. The tracking routine is updated every $0.09\,s$ which can be seen in the figure.

Results of the Longitudinal Correction

At $9.9\,s$ in figure 5.4, the tracking software on sensor level is calculating the new x value. Without **longitudinal correction** (red signal), the new value is $58.31\,m$. With correction, the new value is $58.29\,m$ (green signal). When comparing the red and the green signal, one can see that the green signal is closer to the blue input signal. At $9.9\,s$ the CarMaker signal has a value of $59.29\,m$. The difference between the blue signal and the algorithm without correction is $59.29\,m$ - $58.31\,m$ = $0.02\,m$ at $9.9\,s$. The difference between the green and the blue signal is $59.29\,m$ - $59.29\,m$ = $0\,m$. The improvement in the x direction is about $0.02\,m$. The correction of the velocity, displayed in the bottom plot of the figure, is more significant. The value of the speed without correction is -7.36 m/s and with correction it is -7.47 m/s at $9.9\,s$. The input signal has a value of -7.48 m/s at $9.9\,s$. As conclusion can be said the **longitudinal correction** brings the velocity in x direction closer to

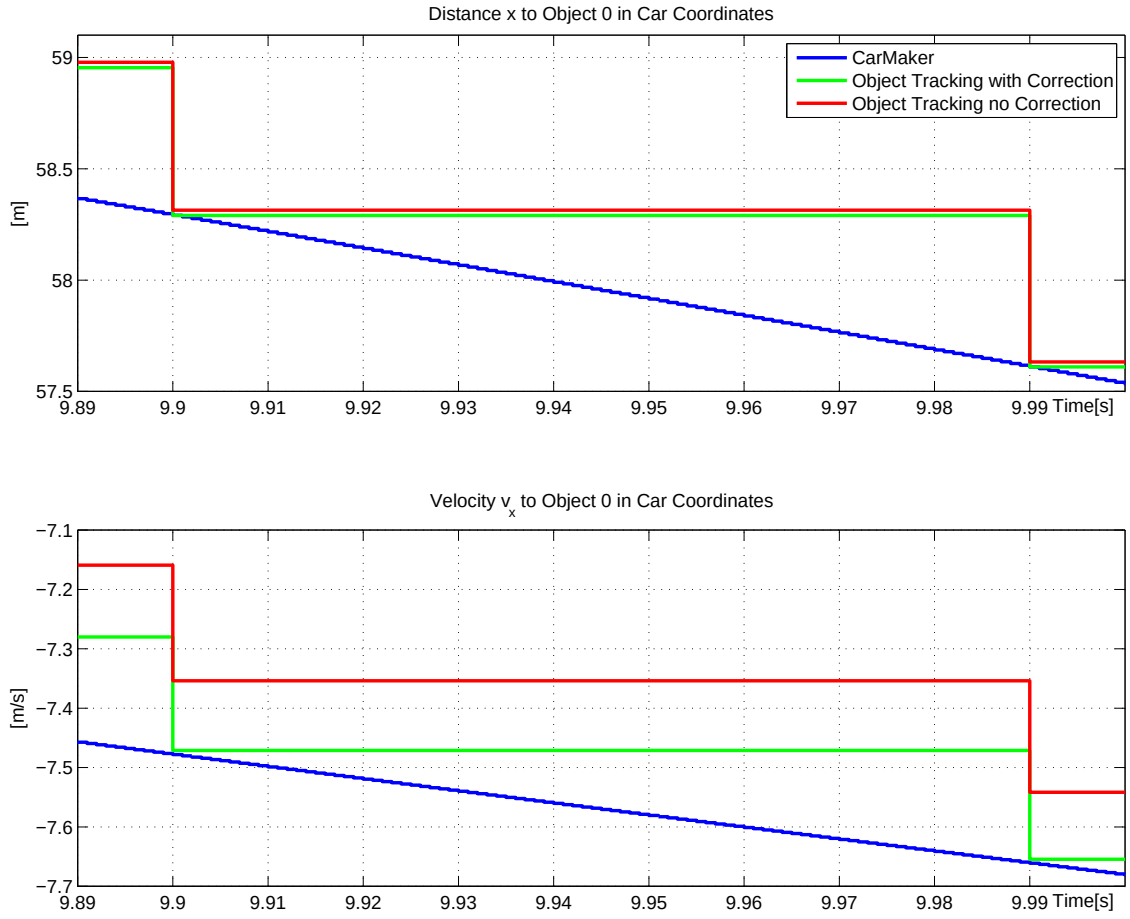

Figure 5.4: Illustration of the results of the longitudinal correction.

the CarMaker signal. The difference between the speed without correction and the input signal is: -7.36 m/s + 7.48 m/s = 0.12 m/s. The difference between the velocity with correction and the input signal is: -7.37 m/s + 7.48 m/s = 0.01 m/s. The improvement in number is: 0.12 m/s - 0.01 m/s = 0.11 m/s. With the support of the **longitudinal correction**, the velocity is about 0.11 m/s closer to the CarMaker signal.

5.2.3 Test Case for the Steering Correction

With this example, the functionality of the **steering correction** is tested. The test is done on a straight two lane highway, where the target is standing on the right lane, 126 m before the EV. The EV starts on the right lane with a velocity of 5.56 m/s and makes the following steering actions:

1. Changing to the left lane
2. Driving back to the right lane
3. Changing again to the left lane
4. Moving back to the right lane

The EV and the object have the same velocity v_x and so the distance in x direction reamins constant. This is done to avoid that the **longitudinal correction** gets active. The thresholds are set to the following values: **threshold longitudinal correction** $= 100\,m/s^2$ and **threshold steering correction** $= 0.05\,rad/s^2$. Here, are all state variables (x, v_x, y and v_y) improved and therefore all are plotted in the following figures.

The expected result of the example is that through the correction activities, the algorithm is able to react faster to steering manoeuvres of the EV. In figure 5.5, the signal sequences of the test case are plotted. In the top diagram, the object detection flag is displayed. Since the target is in the range of the camera during the whole test, the signal always has the value of the object ID, which is zero. In the second plot, the yaw rate $\dot{\Theta}$ is illustrated. From the third diagram on, there are three signals plotted. The blue line comes directly from CarMaker, the red signal is the target track without improvement and the green line belongs to the tracking with correction. Indicated in the legend in the right top corner of figure 5.5. In the plot number three to six, are the state variables over time displayed. The main target is to follow the input signal as good as possible. Compared to that criterion the algorithm is doing fine. Most of the time only the green sequence is visible because the other signals are laying under it. To see how the correction is working in detail, it is necessary to zoom into the waveforms, which is shown in figure 5.6.

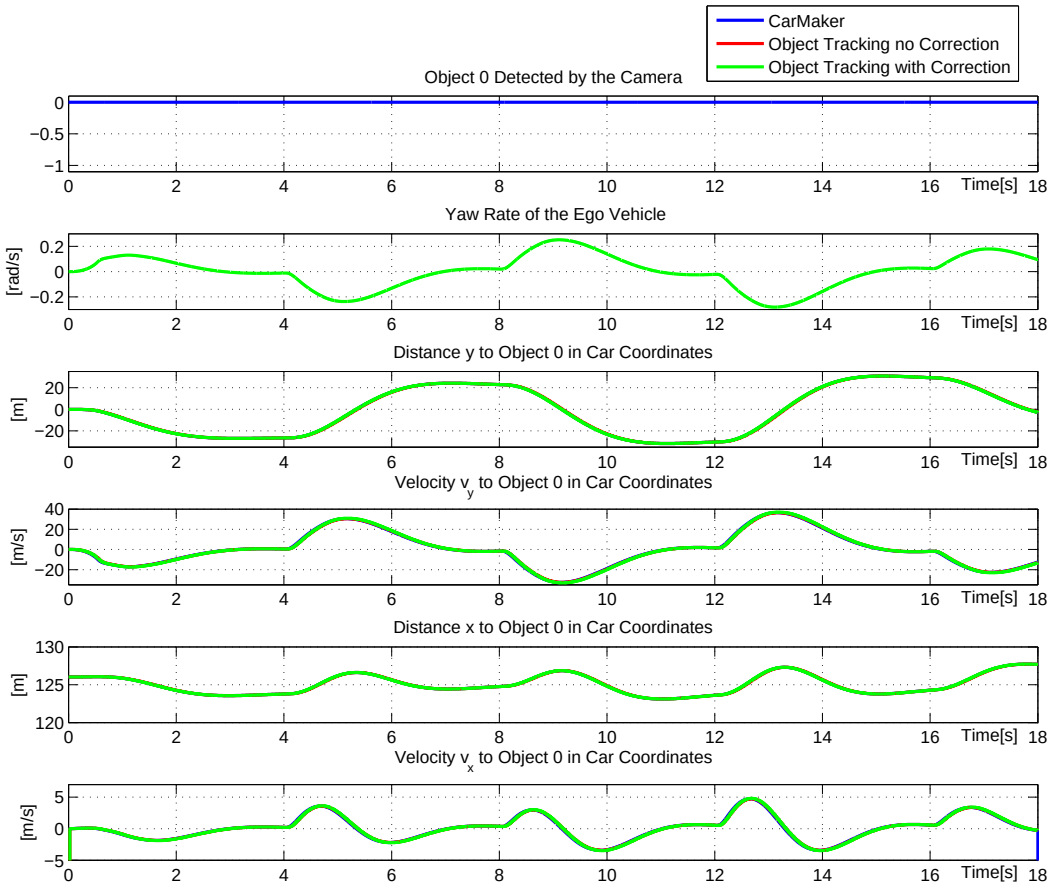

Figure 5.5: Signal sequences of the steering correction.

In figure 5.6, the signals do not overlap each other any more. The zoom is chosen such that there is one sample step of the target tracking algorithm illustrated. The part of the signal sequence is selected randomly; however, this can be done at any time frame of the signal as the result is the same everywhere. Remember that CarMaker is working with a sample time of 0.001 s and the camera has in this example a sample rate of 0.025 s. The **steering correction** improves all four variables of the state vector. Since the function of the correction is the same for x and y direction, only the distance and speed in y direction are displayed here.

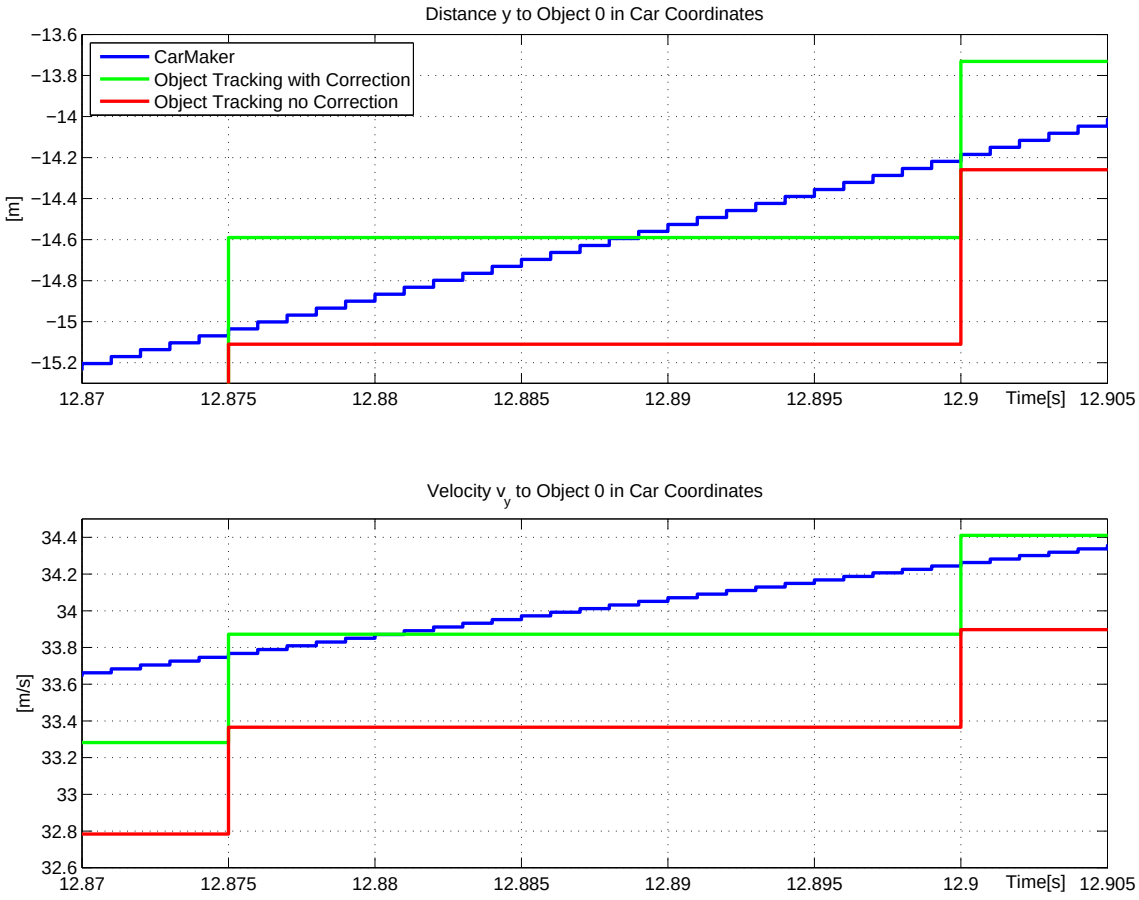

Figure 5.6: Illustration of the results of the steering correction.

Results of the Steering Correction

Looking at the scopes of figure 5.6, it can be seen that the **steering correction** has improved the performance of the OTASL compared to that without correction. The green signal is closer to the blue signal than the red signal in both plots. At 12.9 s in the top plot, the CarMaker signal has a value of -14.18 m, the tracking algorithm with correction has a value of -13.73 m and the algorithm without improvement has a value of -14.26 m. From that the difference between the blue- and the green signal is -14.18 m + 13.73 m = -0.45 m, the aberration between blue- and the red signal is -14.18 m + 14.26 m = 0.08 m. Comparing the two calculated differences, the algorithm without correction (red) is closer to the blue input line than the that one with correction (green). But in the middle over a camera sample step the green signal is closer to the blue signal, than the red signal. As a conclusion can be said, that the median error over one sample step for the algorithm with correction is lower than the error without correction.

In the bottom diagram, the velocity v_y in CCS is displayed over the time. At 12.9 s the input signal has a value of 34.26 m/s. The algorithm without correction has a value of 34.41 m/s and the algorithm with correction has a value of 33.9 m/s at 12.9 s. Comparing the differences between CarMaker and the algorithm with- and without correction leads to the following results: The aberration between the blue and red signal is 34.26 m/s- 33.9 m/s= 0.36 m/s and the difference between blue the and the green signal is 34.26 m/s- 34.41 m/s= -0.15 m/s. As a result, the velocity with **steering correction** is about (-0.15 m/s + 0.36 m/s =) 0.21 m/s closer to the CarMaker signal.

5.2.4 Test Case for the Object Tracking Algorithm on Sensor Level

This test case shows the functionality and the performance of the target tracking routine. The experiment is done on a straight two lane motorway. The EV is standing during the whole test and there are two moving objects. Object 1 is driving away from the EV in the same lane. In figure 5.7, the driving trajectories of both objects are illustrated. Object 0 is starting on the left lane, driving to the right lane and then overtaking object 1, where it also has to make some lane changing actions. At about 20 m in x direction target 1 is not visible any more for the driver because it is hidden by the other traffic member. Both objects are starting the movements 5 m in front of the EV.

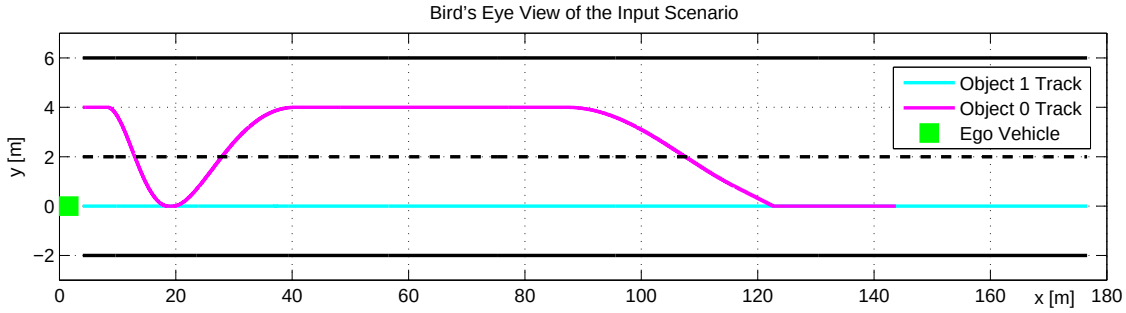

Figure 5.7: Bird's eye view of the object trajectories.

The test shows that the program is able to track targets over an adjustable amount of time, even if they are not visible for the sensor. Moreover, the track management system is tested, if it can create new tracks, handle the tracks when the object is hidden and delete the track when the object is out of the FoV of the sensor. At a round 20 m in x direction, object 1 is hidden by target

0, but the algorithm should be able to track target 1 as long that the track is not deleted. At a distance 120 m in x direction, the track of object 0 must be deleted because it is hidden by object 1 for the rest of the test.

In figure 5.8, the signal sequences of object 0 are plotted. In the first diagram, the object detection flag is displayed, which is 0 (= object ID) until it is hidden by target 1 at about 15 s. In the second plot, the sequence of the track active flag is shown. The value is 0 until the track becomes inactive (= -1) at about 17.5 s. By comparing the detection flag with the track active flag, it can be seen if there are no measurements available but the track is still active. After the predefined time goes by, which can be set with the parameter **tracking samples**, the track is deleted. From diagram number three on to last diagram, two signals are displayed in each plot. The blue signal is the input, which comes from CarMaker, the red signal is from the OTASL, see the legend in the left corner of diagram three. In diagram three to six, only the red signal is visible because the blue line is hidden beneath is. In plot number three, the difference distance in x direction is illustrated, in diagram four the difference velocity v_x, in the next plot the difference distance y and in the last diagram, the velocity v_y over time.

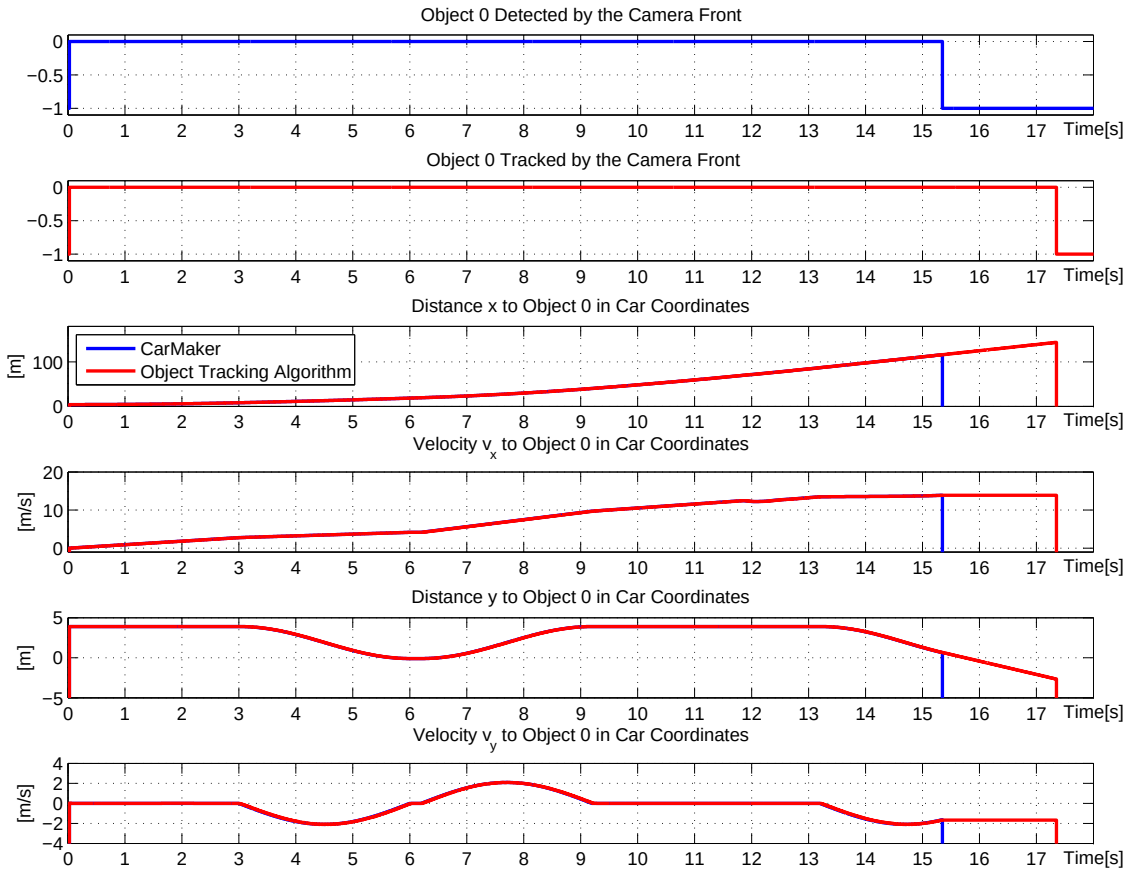

Figure 5.8: Signal sequences of object 0.

In the next figure 5.9, the signal sequences of object 1 are displayed. In the diagrams of the figure, the same signals are plotted as in figure 5.8, the only difference is the recognized object. Because of this, the detection flag and the track active flag have a value of 1 (= object ID) here when they are active.

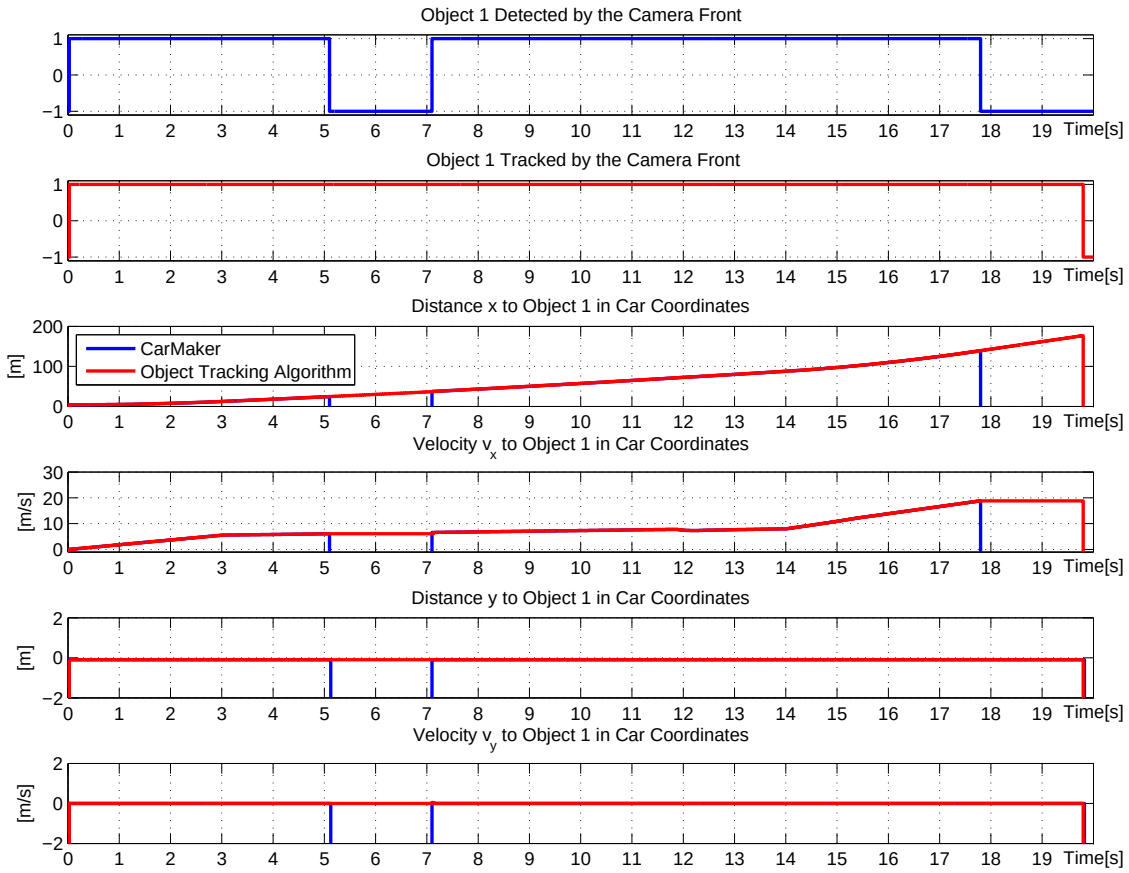

Figure 5.9: Signal sequences of object 1.

Results of the Object Tracking Algorithm on Sensor Level

As mentioned above, the signals of object 0 are plotted in figure 5.8. Comparing the first two diagrams, it can be seen when the object is tracked by the algorithm. When the detection flag has the value -1 and the track active flag has the value 0, the movement of the target is predicted by the routine. This is the case between 15 s and 17.5 s. In the plots (three to six) where the state variables are displayed, the blue signals are set to their default values (-500) at about 15 s. After this time point and until the red signals are set to their default values (-500) at 17.5 s, the track is held by the prediction of Kalman Filter. In diagram five, the lane changing manoeuvres of object 0 are displayed. When the object is tracked at about 15 s, it seems that the car is leaving the highway. This is because only the position is updated, but the velocity remains constant on the last available measurement value. The algorithm is calculating the new difference distance based on the last measured speed. As a result, it seems to the algorithm that target 0 is driving farther in minus y direction.

Comparing diagram one and two, it can be seen when the target is detected or tracked in figure 5.9. Between 5 s and 7 s, when object 1 is hidden by a another traffic member, the track is still active. Since the object does not change the lane and its velocity, the predicted x (diagram three) and y (diagram five) distances correlate with input from CarMaker. The results are satisfying. The plots number one and two show that the track managing section of the algorithm is working correctly. The track is initialized at the beginning, then held between 5 s and 7 s (no new sensor data), and deleted after about 19.5 s.

In conclusion, can be said that the program is able to track objects over an adjustable time range with the parameter **tracking samples**. The tracking has some weaknesses, according to the underlying object model it is not able to recognize a change of the velocity. The track management part of the software is able to initialize, hold and delete tracks. The results are acceptable based on the requirements made for the target tracking on sensor level in section 3.4.1.

5.3 Test cases for the Global Sensor Data Fusion

Now that the target tracking algorithm is validated, the next part of the environment software is tested and the results are discussed. In this subsection, three different examples to prove the **GSDFA** are described. The first two test cases check if the SDF is working correctly and if it achieves desired results. The last test case analyzes the behaviour of the system if there is noise on the input signals.

5.3.1 Used Ego Vehicle and Parameters

The same car as in the section 5.2 is used, with the difference that here, several sensors are observing the surrounding area. The following sensor setup is used for the test cases:

- Long range (LR) radar
- Mid range (MR) radar
- Camera
- Three short range (SR) radar

The positions of the sensors are illustrated in figure 5.10. The placement and type of sensors are chosen according to state of the art configurations, discussed in [23] and [24].

The sensors used for the validation examples, (see table 5.2) have the following parameters and belong to [8], [25] and [26].

Sensorname	**Sample Rate [s]**	**Range [m]**	**Horizontal angle [°]**	**Vertical angle [°]**
Radar Front LR	0.066	200	17	4.3
Radar Front MR	0.066	60	56	4.3
Radar Back	0.04	65	120	24
Radar Side Left	0.04	65	120	24
Radar Side Right	0.04	65	120	24
Camera Front	0.028	140	52	39

Table 5.2: Parameters of the environment sensors.

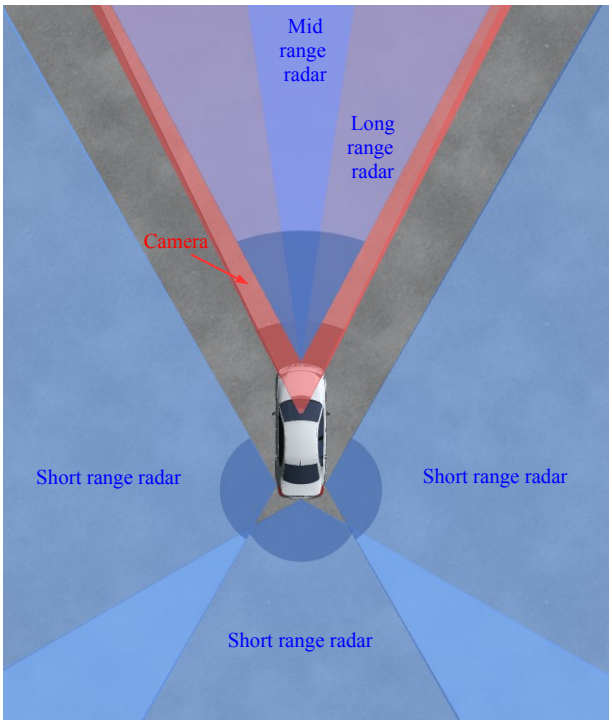

Figure 5.10: Positions of the sensors on the ego vehicle body.

Table 5.3 shows the used parameter settings of the tracking routine on sensor level and global data fusion.

Parameter	**Values**
Number of objects	5
Tracking samples	30
Threshold longitudinal correction	$0.2\,m/s^2$
Threshold steering correction	$0.02\,rad/s$
Fusion sample time	$20\,ms$
Number of sensors	6
Number of detection sensor	6

Table 5.3: Parameter values for the validation examples of the GSDFA.

5.3.2 Static Test Case for the Global Data Fusion

This test case proves the ability of the software to merge several tracks from the sensors to one global track. If the objects are hiding each other, the algorithm predicts the movements of the objects over an adjustable period of time. The experiment takes place on a straight two lane motorway with two traffic members. The EV is standing on the right lane during the whole test. Object 1 starts in front of the EV in the right lane and is moving away with a constant velocity. Object 0 begins on the left lane, overtakes the EV and then changes to the right path

where it hides object 1. After a short time, object 0 switches back to the left lane to overtake car number 1. When the passing manoeuvre is finished, object 0 changes back to the right lane where it is not visible any more for the driver of the EV. The described scenario can be seen in figure 5.11.

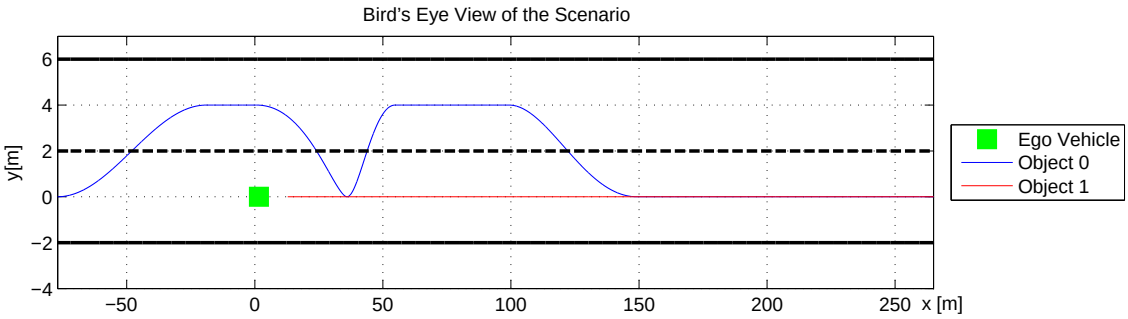

Figure 5.11: Illustration of the input trajectories of the ego vehicle and the objects.

Expected results of the test case:

- The target tracks of the algorithm should be close to the real driving manoeuvres of the traffic objects.
- If the targets are hiding each other, the algorithm should predict the movement of the object.
- If no measurement is available over a certain period of time, the track must be deleted.
- The accuracy of the tracks should increase with the number of sensors that have recognized the object.

The top plot of figure 5.12 shows the number of sensors that have detected object 0. If the value of the signal is greater than 0, the object recognized from at least one sensor. It can be seen that the target is tracked until 19 s and deleted afterwards. From diagram number two on, there are two signals in it: the input signal from CarMaker (blue) and one coordinate of the global track of object 0 (green) are plotted, as indicated in the legend in the top left corner. Object 0 is starting behind the EV at -73 m and at 0.5 s. Then, it overtaking the EV (0 m and 4 s) and finally drives away. At 16 s object 0 is hidden by target 1; as a result, the blue signal goes to the default value (-500). The track is then predicted from the algorithm until 19 s and deleted afterwards, the green signals is set to default value. Also, at time point 19 s the signal number of sensors gets the value 0. The described movements and behaviour relate to plot number two. In the diagram three, the lane changing manoeuvres of the target are displayed. At the value 0 m, the EV and the object are in the same lane; at 4 m, the target is on the right lane of the highway. As a first conclusion, it can be said that the track follows the input signal without great deviation. Moreover, the track is deleted if it is not confirmed by any measurement over a certain time.

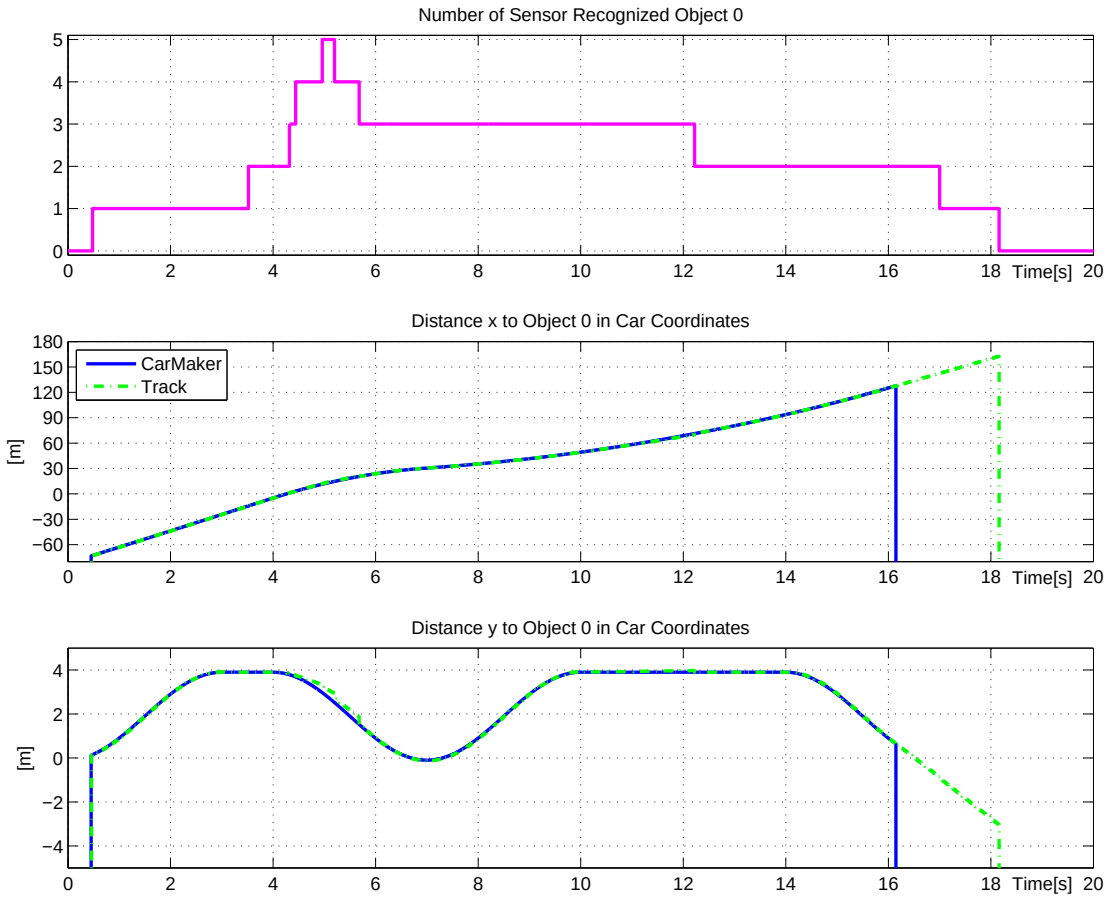

Figure 5.12: Signal sequences of the x and y coordinates of object track 0 over time in CCS.

Figure 5.13 illustrates the signal sequences of object 1. In the diagrams, are the same signals plotted as in figure 5.12. As object 1 is drive in the same lane as the EV during the whole test, the value of the y position stays constant at $0\,m$, as can be seen in plot three. At $21\,s$, the target leaves the FoV of the EnvM and the CarMaker signals of the diagram two and three are set to their default values. Note that the LR radar has a range of $200\,m$ in x direction. The track is deleted at $23\,s$, therefore the green signal goes to the default value. Between $6\,s$ and $8\,s$, object 1 is hidden by object 0, during that time range the track is predicted by the algorithm. The prediction meets the requirements, since no inconsistencies between the track and the new measurement at $8\,s$ occur.

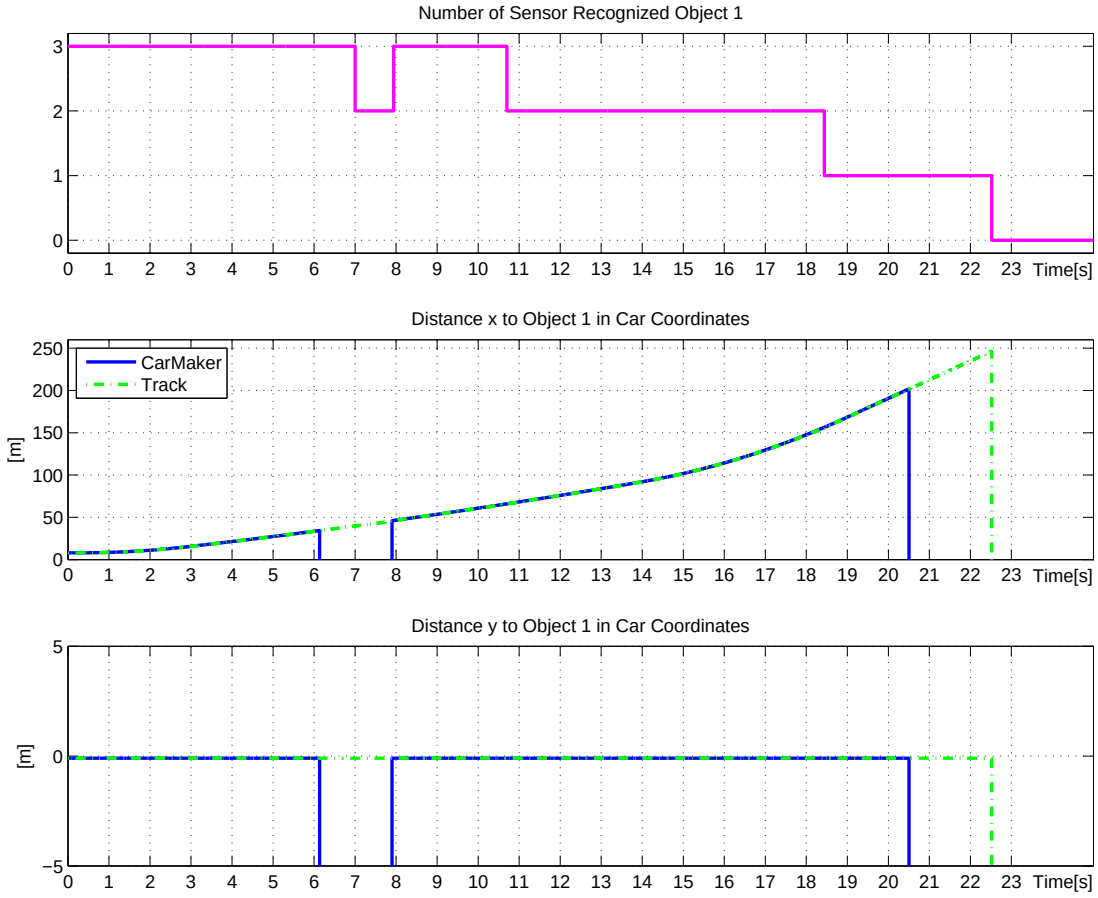

Figure 5.13: Signal sequences of the x and y coordinates of object track 1 over time in CCS.

Figure 5.14 explains how the tracks from several sensors are fused to one. Therefore the signals of the camera and the radar front MR are added to the plots. Due to the sample times of those sensors and the fusion, the sequences are zoomed in. The plots in the figure display the signals from 15.29 s to 15.42 s. The time point and the object track are chosen randomly. Remember that the fusion is a two stage process. In the first step, the measurements are updated to fusion time point. In the second step, the updated tracks are fused to one with an averaging. The blue signal is the input from CarMaker, the black signal is the track of the LR radar sensor, the red signal is the track of the camera, and the green signal is the global track. In the figure, the sample rates of the sensors and the fusion according to the settings from table 5.2 can be seen. The goal is to come as close as possible to the input signal. For example, at 15.34 s the radar and camera track are predicted to the fusion time point and then the routine merges the target tracks together to one. The difference between each sensor track and the CarMaker signal is much bigger than the difference between the global track to the input. From that follows that through the fusion the precision is increased.

Results of the Static Test Case for the Global Data Fusion

In figure 5.15, the global tracks of object 0 and 1 are illustrated. The following signals are displayed: the green point is the EV, the blue signal is the input trajectory of object 0, the red line is

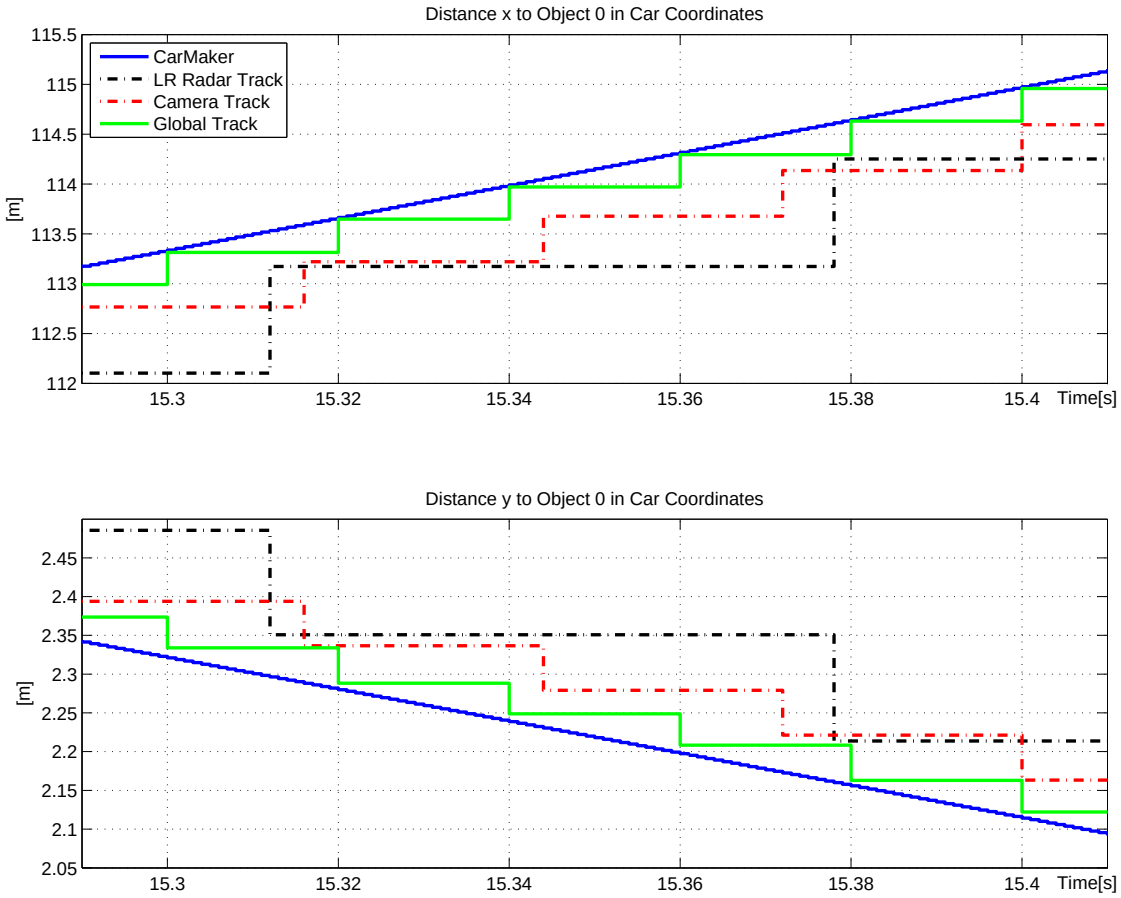

Figure 5.14: Fusion process with two unsynchronized sensors. The tracks of the sensors LR radar (black) and camera (red) are fused to one global track (green).

the input trajectory of object 1, the magenta signal is the object track 0, and the cyan signal is the object track 1. If the tracks are dashed, it means that no measurements are available at this time and that the movements of the objects are predicted by the algorithm. For better visibility, track 1 is slightly shifted downwards. Normally the tracks are overlapping each other at about 150 m in x direction. The example shows one weakness of the algorithm: since the object model is not able to update the velocity, the whole prediction is based on the last measured speed. This explains why, at about 150 m in x the track of object 0 goes further in negative y direction and leaves the motorway, although, in reality, object 0 drives in front of object 1.

Considering the tracking of object 1, the result is good, there is no difference between the track and real object movements. The routine is able to follow the given track during lane changing actions, as can be seen in object track 0. Through the use of a network of sensors, the outcomes get more precise. If the object leaves the detection area of the EnvM, it is tracked over an adjustable period of time and then the track is deleted. This behaviour can be observed at the end of track 1.

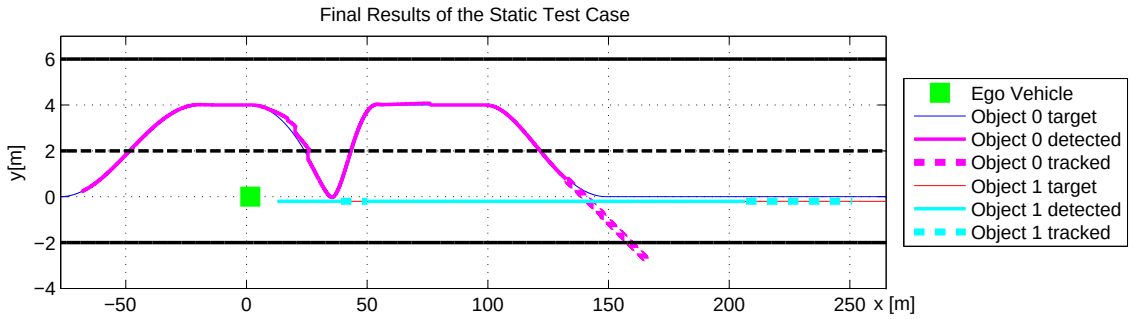

Figure 5.15: Result tracks of target zero and target one in car coordinates.

5.3.3 Dynamic Test Case of the Global Fusion

In this test case the EV is making dynamic driving manoeuvres to show that the software can handle complex traffic situations on the highway. Here, is as test environment a three lane motorway with a straight part and a curve used. This illustrates that the EnvM can track objects on both sides of the vehicle. As input a situation with three traffic objects is taken. Object 0 (blue line) is driving on the left lane and then overtakes object 1 in the middle lane and the EV. Object 1 (red signal) is driving in the middle lane. Towards the end of the test case, it overtakes object 2, afterwards it reaches the end of the test track and disappears. Object 2 (bold black signal) moves on the right lane with constant speed during the whole time. The EV (green line) is starting in the left lane. Towards the end of the experiment it switches to the right lane and overtakes object 2. The driving trajectories of the traffic members and the EV are illustrated in figure 5.16.

Expected results of the test case:

- The movement of the EV should not influence the tracking.
- The target tracks should be close to the input trajectories.
- During the passing manoeuvre, object 0 is hidden by object 1 over a period of time. When target 0 is not visible, it should be tracked by the algorithm, so that the position should be known at any time.
- Tracks must be deactivated when the objects leave the sensor network detection area.

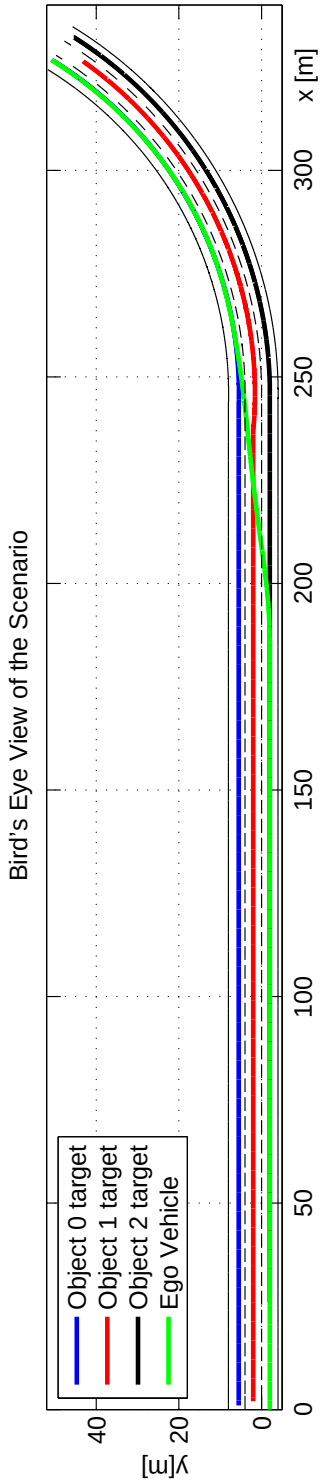

Figure 5.16: Illustration of the input trajectories of the ego vehicle and the three objects.

In figure 5.17, the results of the tracking are illustrated for object 0. The number of sensors that have detected the object is plotted over time in the first diagram. In plot two, the x distance to the object in the CCS is displayed over the time. The blue signal is the input and the green line is the global track, as it is shown in the legend in the left top corner. In diagram three, the y distance is plotted over time in the CCS. In the last plot, the position of target 0 in the GCS is shown. Again, the blue signal is the input from CarMaker, the green and red lines are the tracks from the algorithm. The green color indicates that the track is based on measurements, while red means it is predicted by the routine. Object 0 is driving in the left lane, overtaking the car in the middle lane and the EV simultaneously. Between $1\,s$ and $3\,s$, object 0 is hidden by object 1. During this time gap, the movements of object 0 are predicted, which can be seen in diagram two and three, as the blue signals go to their default value, and in diagram four, as the track's color is red between $30\,m$ and $80\,m$ in x direction. The forecast of the track fits to the input signal. When target 0 is driving in the curve, it is also not visible for the EV, but here the prediction does not fit to the blue input trajectory, as can be seen in diagram four at about $300\,m$ in x direction. The difference comes from the fact that the prediction is based on the last available measured velocity. It can be seen that the quality of the track depends strongly on the availability of sensor data and the dynamics of the object.

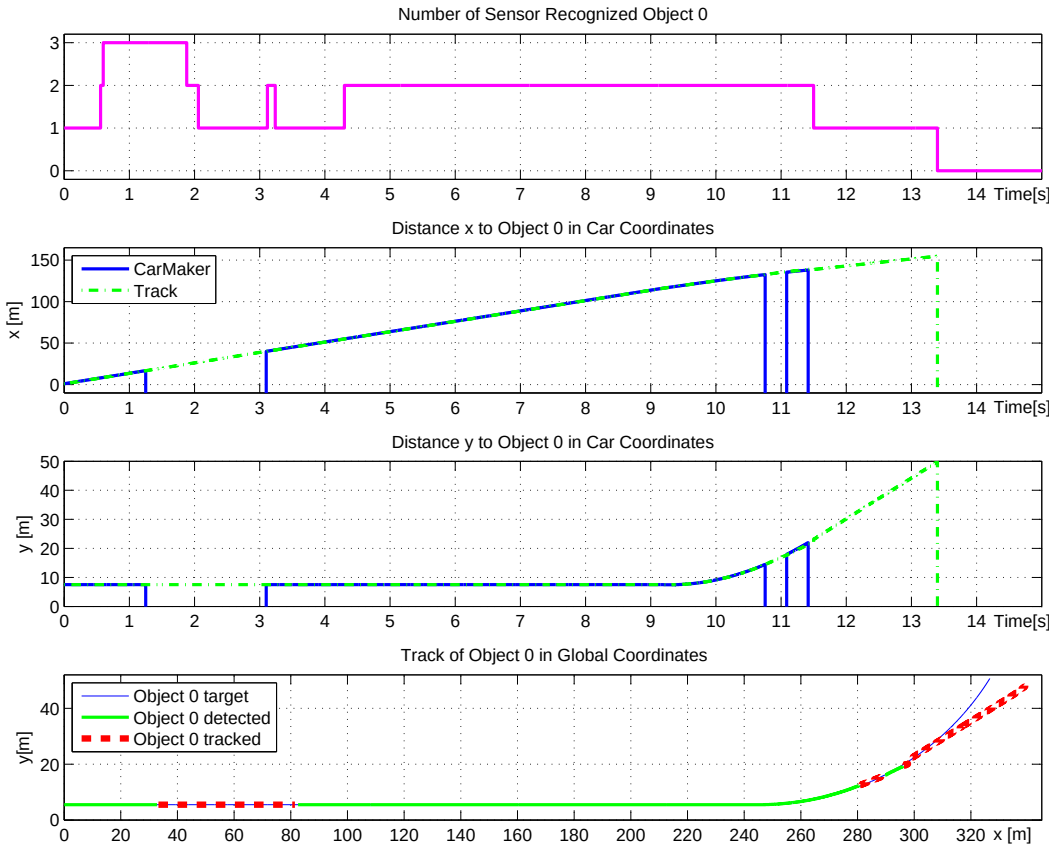

Figure 5.17: Signal sequences of object 0.

In figure 5.18, the results of the tracking of object 1 are illustrated. Compared to object 0 in figure 5.17, different colors are used for the signals in plot four. As it can be seen from the diagrams, object 1 is always detected by same sensors until 22 s. At this time point, the object leaves the test course and the values of the blue input signals in plot two and three are set to default. The red trajectory in plot four is the input and the signals with the colors magenta and cyan belong to the object track. Magenta indicates that the track is based on measurements, while cyan highlights when it is predicted. Since the red input signal is not visible because it is hidden beneath the track, there is no deviation between input and output. At 22 s the track is forecast until the end of the example.

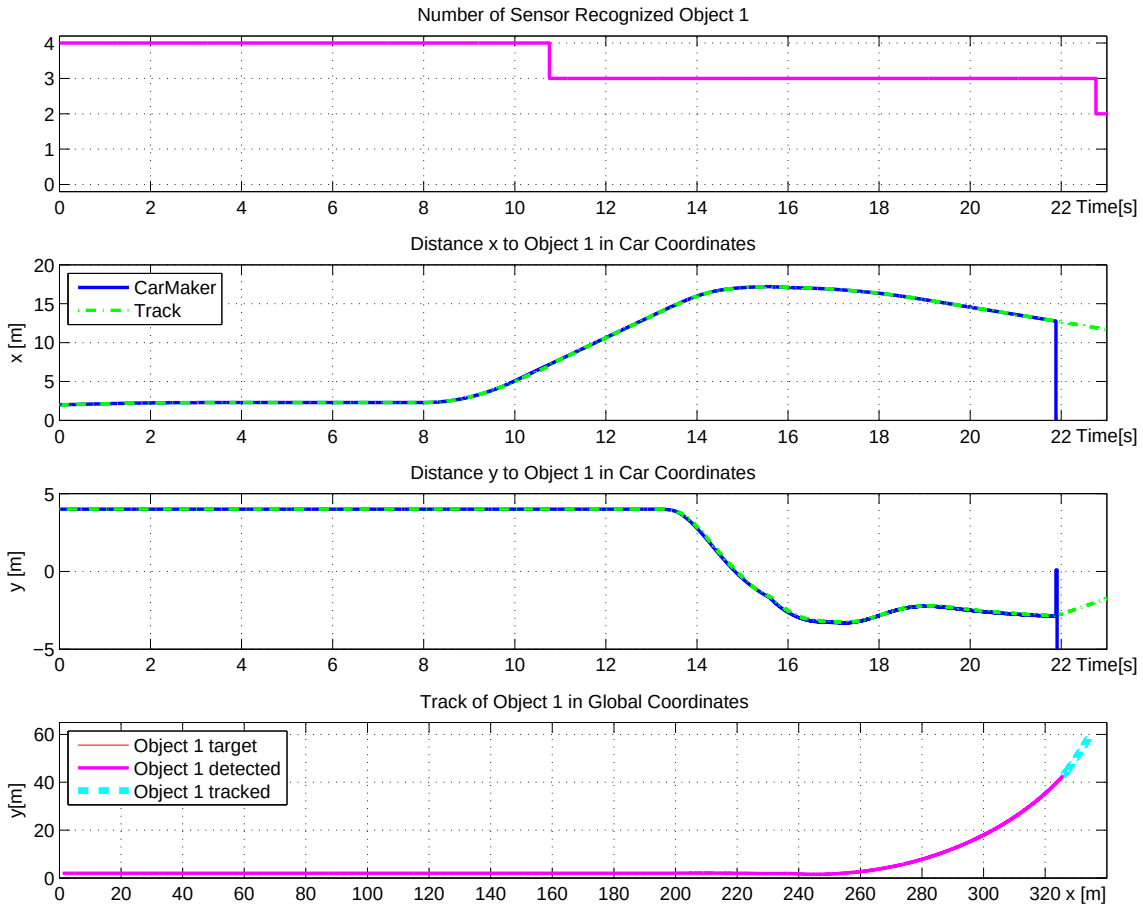

Figure 5.18: Signal sequences of object 1.

In figure 5.19, the results of the tracking of object 2 are illustrated. Compared to object 0 in figure 5.17, different colors are used for the signals in plot four. As it can be seen from the diagrams, the object is detected by the sensors during the whole test case. The achieved result looks good with an exception between time 15 s and 17 s, see plot three. At 13 s the EV is starting its passing manoeuvre and therefore changes from the right lane to the left lane. At about 15 s object 2 leaves the FoV of the SR radar side right, but the sensor tracks the object with the last available measured velocity over an adjustable period of time. This is the reason for the deviation from the green signal to the blue input signal. When the SR radar side right sensor stops the

prediction of the position at time 16.5 s, the green signals goes to the blue line again. To avoid such failures in future, a weighting of the tracks according to if they are based on measurements or not can help. The black trajectory in plot four is the input and the signal with the color cyan is the object track. Most of the time the black signal is not visible because it is hidden beneath the cyan track, it can only be seen at about 250 m in x direction. This is because of the aberration of the y track, described earlier.

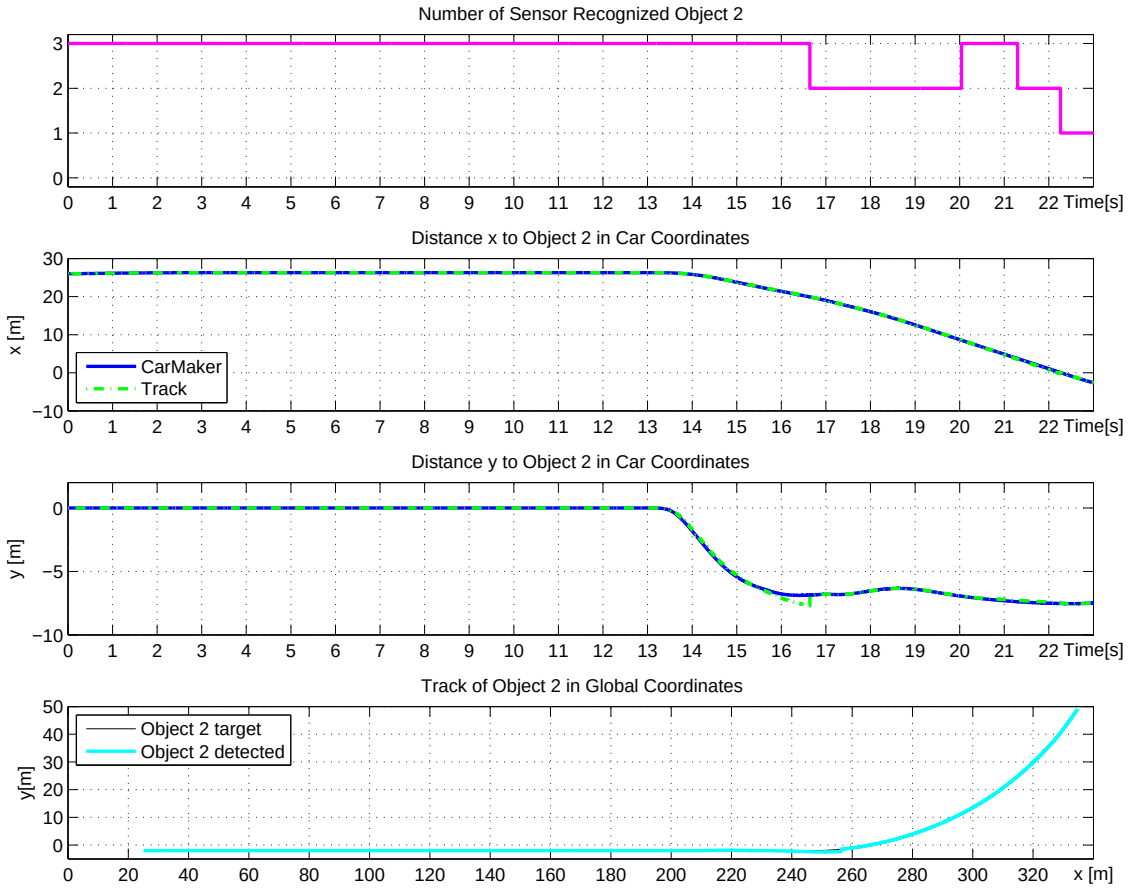

Figure 5.19: Signal sequences of object 2.

Results of the Dynamic Test Case for the Global Data Fusion

The movement of the EV does not have an influence on the quality of the target tracks, compared to the results of the static example. The achieved outputs satisfy the claimed results from the beginning of this subsection. A weakness occurs if the tracked object is changing its velocity during prediction. Then the algorithm is not able to follow the target movements correctly, which can be seen in the fourth plot of figure 5.17 and in the third diagram of figure 5.19. Possibilities to reduce the failure: decrease the prediction time or implement a weighting for the tracks according to if they are based on measurements or not.

5.3.4 Test Case for the Global Data Fusion with Noisy Input Signals

For this validation example, disturbances are laid over the input trajectories to simulate real signals, which always have a noise on it. This test checks the behaviour of the EnvM with disturbances on the inputs. It is easier to show and explain the functionalities of the algorithm by using ideal signals, which is the reason why in all previous test cases the noise simulation was switched off.

The disturbance is generated in the noise simulation block and afterwards added to the input signals, see figure 4.4. It is possible to create a separate disturbance for every sensor. The noise block needs the power density spectra (PSD) of the disturbance as input, and the output will be normally distributed white noise. Since there are no measurements or information available about the disturbances, and for simplification in this thesis, the PDS are chosen in order to generate a fluctuation. In this test case, all sensors beside the camera use the same noise settings, as can be seen in the following table.

PDS for distance noise	PDS for velocity noise	Sensors
0.008	0.0008	Radars
0.001	0.0009	Camera

Table 5.4: PDS values for the noise simulation.

Thw input traffic scenario is the same as the one used in the dynamic test case; for detailed information see subsection 5.3.3. The only difference is that here, a noise is on the input signals.

Expected results of the test case:

- Kalman Filter should smooth the noisy input signals.
- The disturbed input signals should not influence the result of the fusion. The sequence of the target tracks must be equally to case 5.3.3.

To show how the algorithm handles noisy input signals, the tracks of object 1 from the LR radar and the camera are taken. The sensors and the object are chosen randomly. It is not necessary to display the signal sequences of all sensors for all traffic members. In figure 5.20, the signal sequences of object 1 on sensor level are illustrated. In the first two plots, the blue signal is the ideal input, the red line is the input with disturbances and the green signal is the track of the LR radar. With the matrices $\mathbf{R}$ and $\mathbf{Q}$, the performance of the Kalman Filter can be adapted according to the noise on the signals. The difficulty is to find the balance between smoothing the disturbances and following the input. According to those requirements the values of the matrices are set. The smoothing will be evaluated in a figure, where the signal sequences are zoomed into to see the noise better. The object track is close to the blue signal, which means the filter is able to follow the input even when there are disturbances on it.

In the diagrams three to four, the signal sequences of object 1 are plotted from the camera. The blue signal is the ideal input, the cyan line is the input with noise on it and the magenta signal is the track from the camera. Since all sensors use the same object model, also the settings of the matrices $\mathbf{R}$ and $\mathbf{Q}$ are the same for all. As can be seen in plot three and four, the algorithm is able to track the object, as there is no difference between the blue and the magenta signal. The blue signal is hidden behind the other signals.

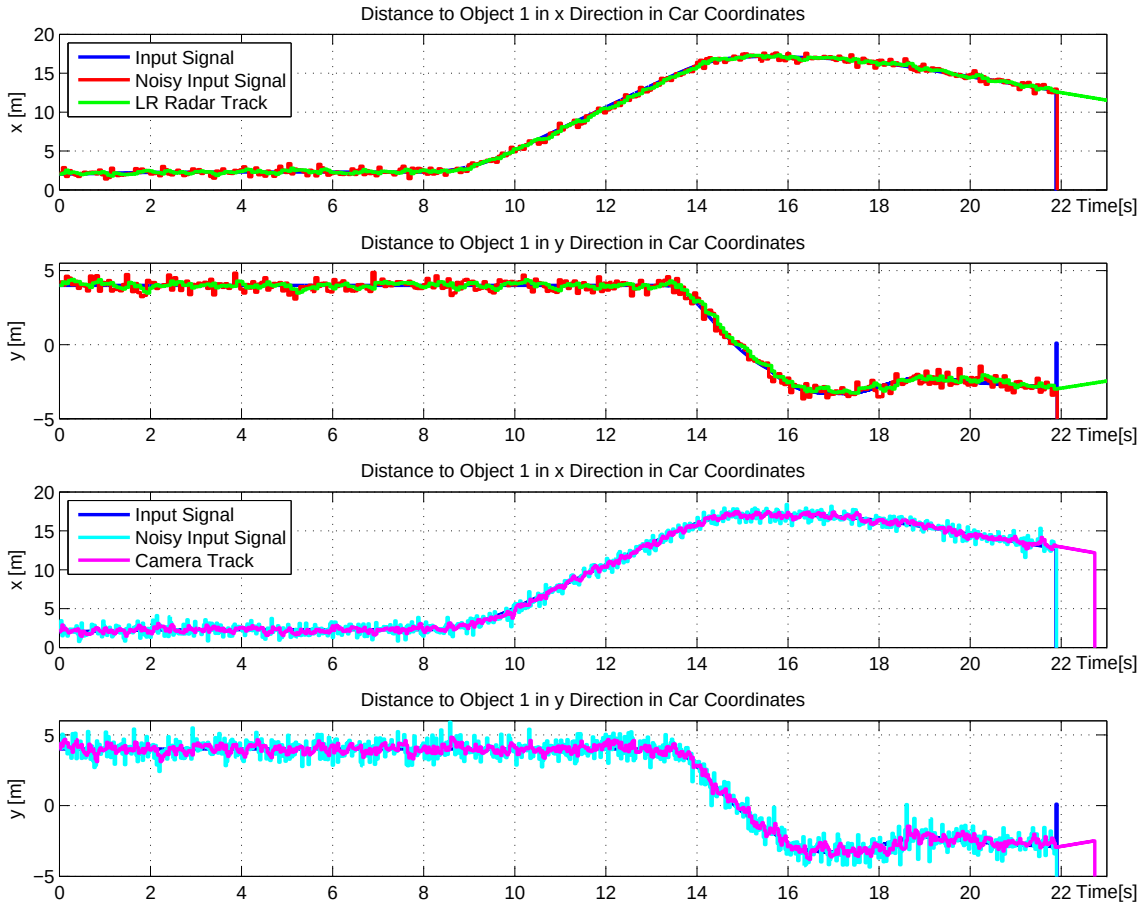

Figure 5.20: Noisy signal sequences of the LR radar and camera for object 1.

In figure 5.21 the same signals are illustrated as in figure 5.20, the only difference being that here the sequences are zoomed in to get a better view of the noise. Comparing diagram two and four, it is obvious that the LR radar and the camera have disturbances with different PSD's. The amplitude of the noise on the camera's input is bigger than on the radar front's input. The different PSD were chosen on purpose, to point out that the algorithm can handle different noises for each sensor. In order to smooth the disturbances, the elements of the main diagonal of the matrices $\mathbf{Q}$ are set to 1.5 and of $\mathbf{R}$ to 6. The achieved results with thess settings are satisfying and the disturbances are smoothed, as it is illustrated in all the diagrams in the figure 5.21.

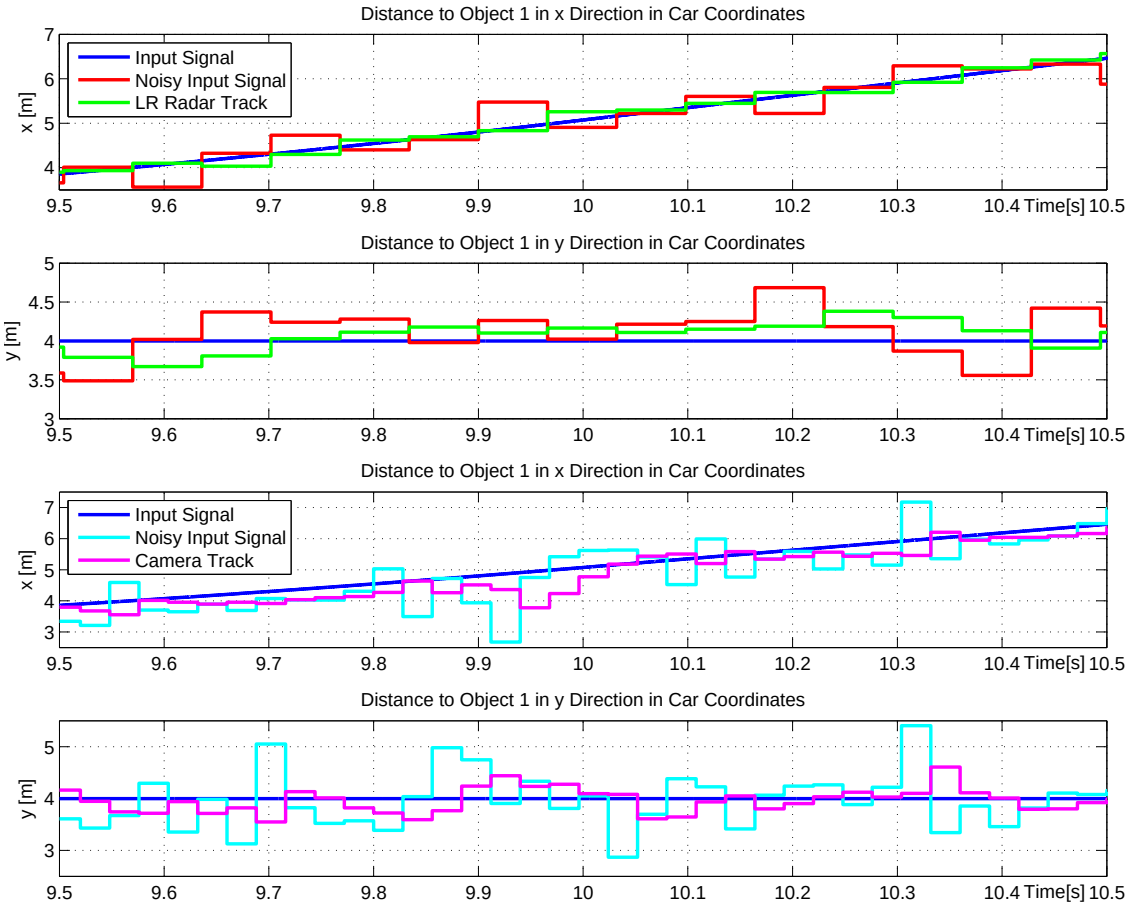

Figure 5.21: Detail view of the noise on the sequences of object 1.

Results of the Example with Noisy Input Signals

The object tracks on sensor level are handed over to the SDF to built the global tracks, illustrated in figure 5.22. In the top diagram, the input trajectory (blue) and the track (green and red) of object 0 are displayed, as it can be seen in the legend in the left corner. The colors of the track indicate the following: green is based on measurement and red is prediction of the object's movement. In plot two and three, the same signals are shown for the other objects but in different colors; please refer to the legends in the top left corner of each diagram.

Looking at the diagrams of the figure, one can see that there are deviations between the input signals and the tracks, which results from the sensor noise. Comparing the signals of the object tracks to the results of the previous test case, they are equal. Especially the times when the object trajectories are detected or tracked are the same, as can be seen in the fourth diagram of figure 5.17, 5.18 and 5.19, respectively. In fact, the noise on the input signals has little influence on the quality of the calculated tracks of the algorithm.

To improve the performance of the smoothing, each sensor should get a separate $\mathbf{R}$ and $\mathbf{Q}$ matrix, which is set according to the measurement noise. This activity will have a positive effect

to the quality of the output tracks.

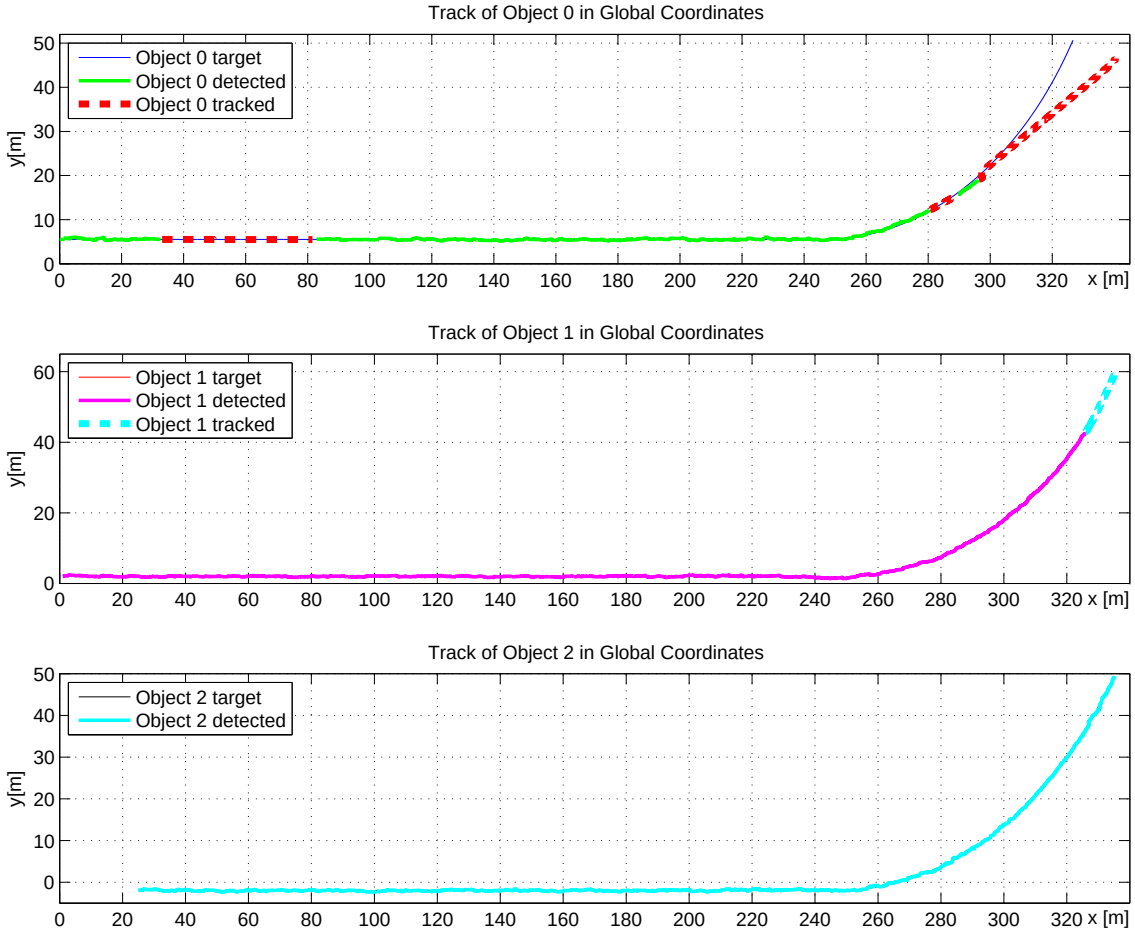

Figure 5.22: Results of the object tracks in global coordinates.

In figure 5.23 the deviation between the input trajectories and the calculated object tracks is shown. In the first plot, the error of object 0 is illustrated. Towards the end of the test case the failure between the input and the object track gets bigger. At about $300\,m$ in x direction the track is predicted with the latest velocity available. But the speed changes and so the prediction is not able to follow the input signal. In diagram two, the error of the object track 1 is plotted. The error fluctuate at round $0\,m$, this deviation only comes from the sensor noise. The last plot illustrates the difference between the input signal and object track 2. The error here also comes from the disturbance of the input signal.

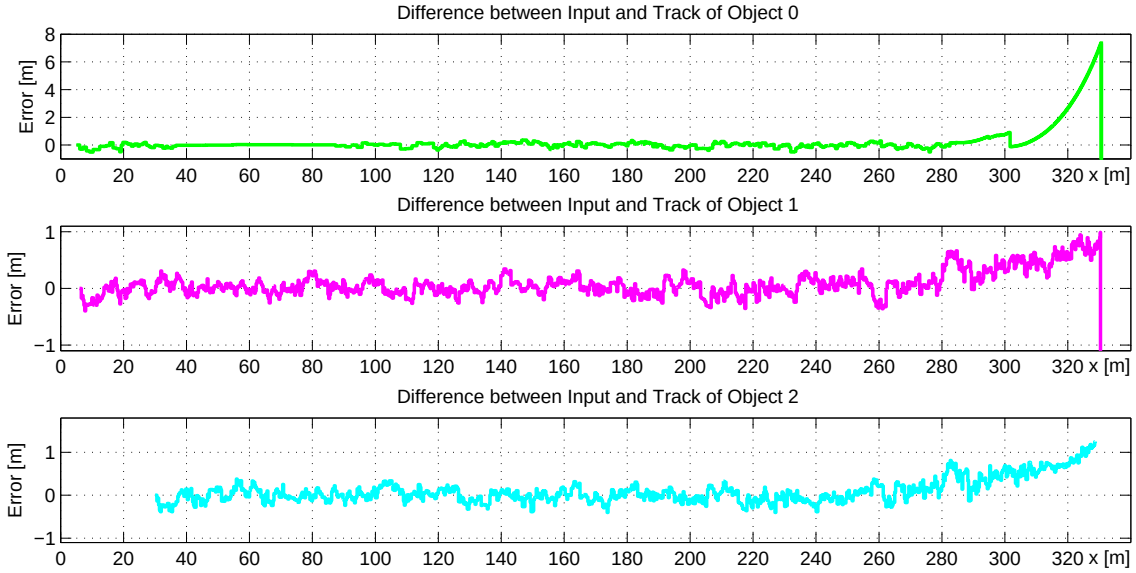

Figure 5.23: Deviation between the input and the tracks of object 0, 1 and 2.

Chapter 6

Conclusion and Perspective

Advanced driver assistance systems support the driver in difficult traffic situations where the limits of the human capacity show as the driver is no longer able to fully grasp the scenario. The main advantage of these systems is that they increase the traffic safety and driving comfort. With the level of automation the use of assistance systems is increasing; this leads to higher requirements for environment modelling. If the car should be able to move primarily autonomous in a previously unknown and undefined space, a precise knowledge of the position of static and dynamic objects is required.

The task of this thesis is to implement an environment model which can be used as a test and development environment for a motorway assistance system. The model consists of three main parts: The sensor model extension evaluates whether or not the detected objects from the simulation environment are masking each other. The extension has two adjustable parameters: with the first parameter, the angle resolution of the sensor can be set. The other parameter defines the percentage at which a partially masked object is detected. The next part is the object tracking on sensor level. Every sensor has its own tracking algorithm. All recognized objects in the sensor's field of view are tracked by a multiple target tracking routine. The algorithm is able to predict the movements of the objects over an adjustable amount of time when there are no measurements available. With the prediction, the drop-out of measurements can be compensated and if the object is shortly masked by other traffic members, the motion of the object is forecast. To improve the performance of the target tracking algorithm, the movements of the ego vehicle are considered. The last part of the environment model is the sensor data fusion. In this part, the different object tracks from the sensors are merged to one global track. The data fusion is a two stage process: In the first step, the tracks from the sensors are updated to fusion time point. This has to be done because a heterogeneous sensor network is used. In the second stage, the various object tracks are fused to one track. At the end, every object which was detected by the sensors has one global output track.

In the last chapter of the thesis, the results of the validation examples are evaluated and discussed. The shown test case illustrates that the environment model is able to handle complex traffic scenarios on the highway. The examples also point out a weakness of the developed algorithm. According to the underlying object model, the tracking routine is not able to update the velocity of the objects in prediction state. In other words, the position of the object is updated with the last measured speed available. If the velocity of the object changes, then the tracked motion deviates from the real movement of the object. To overcome this error in future applications, a weighting could be added. The tracks should be weighted according to if they are based on measurements or if they are predicted by the algorithm.

Appendix A

Explanation of terms

A.1 Abbreviation

DAS	Driver Assistance System
ABS	Antilock Braking System
ESP	Electronic Stability Program
ADAS	Advantage Driver Assistance System
EnvM	Environment Model
ACC	Adaptive Cruse Control
LDW	Lane Departure Warning
SDF	Sensor Data Fusion
EV	Ego Vehicle
OEM	Original Equipment Manufacturer
LP	Leftest Point
RP	Rightest Point
FoV	Field of View
ODA	Object Detection Algorithm
STT	Single Target Tracking
MTT	Multiple Target Tracking
CLF	Central Level Fusion
SLF	Sensor Level Fusion
GF	Global Fusion
GCS	Global Coordinate System
SCS	Sensor Coordinate System
CCS	Car Coordinate System
OTASL	Object Tracking Algorithm on Sensor Level
GSDFA	Global Sensor Data Fusion Algorithm
LR	Long Range
MR	Mid Range
SR	Short Range
PDS	Power Density Spectra

A.2 Symbols

Coordinates Systems

O_{SCS}	Origin of the sensor based coordinate system
x_{SCS}	X-axis of the sensor based coordinate system
y_{SCS}	Y-axis of the sensor based coordinate system
z_{SCS}	Z-axis of the sensor based coordinate system
O_{CCS}	Origin of the car based coordinate system
x_{CCS}	X-axis of the car based coordinate system
y_{CCS}	Y-axis of the car based coordinate system
z_{CCS}	Z-axis of the car based coordinate system
O_{GCS}	Origin of the global coordinate system
x_{GCS}	X-axis of the global coordinate system
y_{GCS}	Y-axis of the global coordinate system
z_{GCS}	Z-axis of the global coordinate system

Variables

t	Time	s
$s(t)$	Position at time t	m
s_0	Initial position	m
$v(t)$	Velocity at time t	m/s
a	Acceleration	m/s^2
s_k	Position of the current time step	m
s_{k-1}	Position of the previous time step	m
v_k	Velocity of the current time step	m/s
v_{k-1}	Velocity of the previous time step	m/s
t_{Sample}	Time between current- and previous time step	s
t_k	Current time point	s
t_{k-1}	Previous time point	s
x_k	Distance between object and ego vehicle in x direction in the car based coordinate system	m
v_{x_k}	Difference velocity between object and ego vehicle in x direction in the car based coordinate system	m/s
y_k	Distance between object and ego vehicle in y direction in the car based coordinate system	m
v_{y_k}	Difference velocity between object and ego vehicle in y direction in the car based coordinate system	m/s
$\mathbf{x}_k$	State variable	
$\mathbf{A}$	System matrix	
$\mathbf{Q}$	Covariance matrix of the model failure	
$\mathbf{R}$	Covariance matrix of the measurement noise	

$\mathbf{H}$	Measurement matrix	
$\hat{\mathbf{x}}_k^-$	Prediction of the state variable at current time step	
$\hat{\mathbf{x}}_{k-1}$	Estimate of the state variable at previous time step	
$\mathbf{P}_k^-$	Prediction of the error covariance at current time step	
$\mathbf{K_k}$	Kalman gain at current time step	
$\hat{\mathbf{x}}_k$	Estimate of the state variable at current time step	
$v_{k_{uncorr}}$	Uncorrected difference velocity at current time step in the car based coordinate system	m/s
$v_{k_{Car}}$	Velocity of the car at current time step	m/s
$v_{k_{Ego}}$	Velocity of the ego vehicle at current time step	m/s
a_k	Difference acceleration at current time step	m/s^2
$a_{k_{Car}}$	Acceleration of the car at current time step	m/s^2
$a_{k_{Ego}}$	Acceleration of the ego vehicle at current time step	m/s^2
Δv_k	Change of the difference velocity at current time step	m/s
t_{Sensor}	Sample rate of the sensor	s
$v_{k_{corr}}$	Corrected difference velocity at current time step in the car based coordinate system	m/s
$x_{k_{corr}}$	Corrected distance between object and ego vehicle in x direction in the car based coordinate system	m
$x_{k_{uncorr}}$	Uncorrected distance between object and ego vehicle in x direction in the car based coordinate system	m
v_{T_k}	Tangential velocity at current time step	m/s
r_k	Absolute difference distance between object and ego vehicle at current time step	m
$\dot{\Theta}_k$	Yaw rate of the ego vehicle at current time step	rad/s
α_k	Angle between object and ego vehicle at current time step	rad
$v_{T_x},_k$	X-component of the tangential velocity at current time step	m/s
$v_{T_y},_k$	Y-component of the tangential velocity at current time step	m/s
$\alpha_{k_{corr}}$	Corrected angle between object and ego vehicle at current time step	rad
$v_{x_{corr}},_k$	Corrected difference velocity at current time step in x direction in the car based coordinate system	m/s
$v_{x_{uncorr}},_k$	Uncorrected difference velocity at current time step in x direction in the car based coordinate system	m/s
$y_{k_{corr}}$	Corrected distance between object and ego vehicle in y direction in the car based coordinate system	m
$y_{k_{uncorr}}$	Uncorrected distance between object and ego vehicle in y direction in the car based coordinate system	m
$v_{y_{corr}},_k$	Corrected difference velocity at current time step in x direction in the car based coordinate system	m/s
$v_{y_{uncorr}},_k$	Uncorrected difference velocity at current time step in x direction in the car based coordinate system	m/s
Δt	Difference time between fusion time point and measurement time point	s
t_{Fusion}	Time point of the fusion	s
$t_{Measurement}$	Time point of the measurement	s
N	Number of sensors	
x_{Global}	Global distance between object and ego vehicle in x direction in the car based coordinate system	m
$x_{i_{Sensor}}$	Distance between the object and the ego vehicle in x direction of the ith sensor	m

Bibliography

[1] William Wong. Electrionic Design. http://electronicdesign.com/iot/internet-things-here-stay/, 2015. [2015-11-29].

[2] Automobili Lamborghini Holding S. p. A. http://www.lamborghini.com/en/models/aventador-lp-700-4-roadster/technical-specifications/, 2015. [2015-12-05].

[3] Volkswagen AG. http://www.volkswagen.de/de/models/golf{_}7/trimlevel{_}overview.s9{_}trimlevel{_}detail.suffix.html/der-golf1{~}2Ftrendline.html{\#}/tab=24da160131fd7fb3520a62a7c0c435cc/, 2015. [2015-12-05].

[4] Simon Steinmeyer. *Probabilistische Fahrzeugumfeldschätzung für Fahrerassistenzsysteme*. PhD thesis, Technischen Universität Carolo-Wilhelmina zu Braunschweig, 2014.

[5] Michael Darms. *Eine Basis-Systemarchitektur zur Sensordatenfusion von Umfeldsensoren für Fahrerassistenzsysteme*. PhD thesis, 2007.

[6] Markus Maurer, J Christian Gerdes Barbara Lenz, and Hermann Winner Hrsg. *Autonomes Fahren: Technische, rechtliche und gesellschaftliche Aspekte*. Springer Berlin Heidelberg, 2015.

[7] Nico Dziubek, Hermann Winner, Matthias Becker, and Stefan Leinen. Sensordatenfusion zur hochgenauen Ortung von Kraftfahrzeugen mit integrierter Genauigkeits- und Integritätsbewertung der Sensorsignale. In *5. Tagung Fahrerassistenz München*, pages 1–19, München, 2012.

[8] Hermann Winner, Stephan Hakuli, and Gabriele Wolf. *Handbuch Fahrerassistenzsysteme: Grundlagen, Komponenten und Systeme für aktive Sicherheit und Komfort*. Springer-Verlag, 2015.

[9] Ioannis Papadakis. Fahrerassistenzsysteme: Überblick und aktueller Trends. Technical report, 2007.

[10] Volker von Holt. *Integrale Multisensorielle Fahrumgebungserfassung nach dem 4D-Ansatz*. PhD thesis, Universität der Bundeswehr München, 2004.

[11] Dipl-Inf Mathias Haberjahn. *Multilevel Datenfusion konkurrierender Sensoren in der Fahrzeugumfelderfassung*. PhD thesis, Humboldt-Universität zu Berlin, 2013.

[12] IPG Documentation. *CarMaker Reference Manual 4.0*. IPG Automotive GmbH, D-76185 Karlsruhe, 2012.

[13] Dirk Stüker. *Heterogene Sensordatenfusion zur robusten Objektverfolgung im automobilen Straßenverkehr*. PhD thesis, 2003.

[14] Antje Westenberger. *Simultane Zustands- und Existenzschätzung mit chronologisch ungeordneten Sensordaten für die Fahrzeugumfelderfassung*. PhD thesis, Universität Ulm, 2011.

[15] Fredrik Gustafsson. *Statistical Sensor Fusion.* Professional Pub Serv, 2013.

[16] IPG Documentation. *CarMaker Programmer's Guide Version 4.0.* IPG Automotive GmbH, D-76185 Karlsruhe, 2012.

[17] IPG Documentation. *CarMaker Quick Start Guide Version 4.0.* IPG Automotive GmbH, D-76185 Karlsruhe, 2012.

[18] IPG Documentation. *CarMaker User's Guide Version 4.0.* IPG Automotive GmbH, D-76185 Karlsruhe, 2012.

[19] Dieter Schramm, Manfred Hiller, and Roberto Bardini. *Vehicle Dynamics: Modeling and Simulation.* Springer, 2014.

[20] Dara W. Childs and Andrew P. Conkey. *Dynamics in Engineering Practice, Eleventh Edition.* CRC Press, 2015.

[21] Branke Ristic, Sanjeev Arulampalam, and Neil Gordon. *Beyond the Kalman Filter: Particle Filters for Tracking Applications.* Artech House, 2004.

[22] Phil Kim and Lynn Huh. *Kalman Filter for Beginners: With MATLAB Examples.* CreateSpace Independent Publishing Platform, 2011.

[23] Markus Schöttle. Zukunft der Fahrerassistenz mit neuen E/E-Architekturen. *ATZ Elektronik*, (04):8–15, 2011.

[24] M. Aeberhard, S. Rauch, M. Bahram, G. Tanzmeister, J. Thomas, Y. Pilat, F. Homm, W. Huber, and N. Kaempchen. Experience, Results and Lessons Learned from Automated Driving on Germany's Highways. *Intelligent Transportation Systems Magazine, IEEE*, 7(1):42–57, 2015.

[25] Kameras retten Leben - Sensorlösungen für zukünftige Fahrerassistenz- und Sicherheitssysteme in Fahrzeugen. In *Automotive-Innovationsforum 2014*, pages 1–24, Kostal, 2014.

[26] James Scoltock. Technology - Automotive Engineer. `http://ae-plus.com/technology/trws-mono-camera-sees-through-the-darkness/`, 2014. [2015-11-17].