Constructing a region of interest using map information for object tracking in autonomous vehicles

Author:	**Randy Stakelbeek**
Master:	Bio-Mechanical Design
Track:	Automotive Human Factors
Student number:	1179381
E-mail:	rstakelbeek@gmail.com
Submitted:	29-06-2016
Supervisor:	Ir. Joris F. M. Domhof
Exam committee:	Prof. Dr. Ir. Pieter P. Jonker
	Dr. Ir. Riender Happee
	Ir. Joris F. M. Domhof
	Dr. Oleg A. Krasnov

TU Delft

Image created with PreScan®

Acknowledgements

This thesis would not have been possible without the help of some very special people. First of all I would like to thank all my supervisors and professors that helped me along the way. Thanks to Dr. Ir. Riender Happee I got an internship at one of the most distinguished research companies in The Netherlands, TNO Automotive, where I started working on semi-autonomous vehicles. After this internship, I also did my literature research and this thesis in the field of autonomous vehicles, with the DAVI project at the Delft University of Technology.

Special thanks goes to Ir. Joris Domhof who kept confidence in my work at all times, guided me, and got me motivated to finish this research. Thanks also to Ir. Stefano Arrigoni, for helping me get part of the data in the correct format. Thanks to your work on your MATLAB functions I could test my hypothesis using recorded data, instead of just simulated data.

To my exam committee; Prof. Dr. Ir. Pieter Jonker, Dr. Ir. Riender Happee, Ir. Joris Domhof, and Dr. Oleg Krasnov. Thank you for taking the time to review my work and giving me valuable feedback. I appreciate you taking time from your busy schedules to do this.

Next to the official guidance, I got a lot of support from my parents, both financially and at times when things didn't go as planned. Thanks for always believing in me and helping me with everything. I also need to thank all of my fellow students, who helped me during courses, with assignments, or just made me feel better by hanging out.

Special thanks has to go to Ir. Maarten Griffioen, together with whom I did many courses and even did a nearly 30-day streak of working at the faculty to get all the work done for many courses (FYI this is thirty full days in a row, including weekends!). Thanks for always being there to support me. Another special fellow student was Emma de Bont. We only did one project together, but we stuck together afterwards and shared our highs and lows. Thank you so much for being there.

During my work on the DAVI project I got to know two very special guests from Italy; Paolo Gottardis and Simone Stefano Manazza. I had great fun showing you guys the good parts of Delft and the rest of The Netherlands. Thank you Paolo for letting us stay in your mountain house in Forni di Sopra (Italy). To Simone, a special thanks for giving me one of the best days of my life, driving me around in *your* car. I hope we can all meet again many times to catch up on all the memories of things we did together.

The last special thanks goes to another fellow student; Dennis Priester. We worked together on some robotics courses, but I mainly want to thank you for all the things outside the study. Especially the caching-days and gaming-nights were always great.

Without the support of these people I would not have been able to do this research and create this report. To you and all I forgot to mention: Thank you so much!

Abstract

The Delft University of Technology intends to aid in the development of autonomous vehicles by building their own experiment platform in the DAVI project. To be able to make valid decisions, an autonomous vehicle needs to know its surroundings. Many sensors are used to detect any objects within range, but the sensor information has to be processed to form a map of the environment with all objects of importance on it; this process is called tracking.

Current methods of tracking and associating objects are accurate enough to be safely used in an autonomous vehicle, but require significant computation power. Increasing the speed of the algorithms without losing performance is a real challenge. As with many methods, the *joint probabilistic data association* (JPDA) algorithm gets exponentially slower with more objects and more measurements. It is therefore proposed that a method that splits the survey area into smaller regions could lead to a faster algorithm.

In this research, a region is dynamically constructed while the vehicle drives around using extended maps that contain all the information on the infrastructure (eHorizon). The region is created by merging polygons that are defined by the road shape. Objects within the region are tracked with a JPDA algorithm, while objects further away (outside the region) are tracked with a fast and simple nearest neighbor Kalman filter. Once an object gets inside the region, the track will automatically be handled by the JPDA algorithm.

Experiments show that the region tracking algorithm performs faster than a JPDA algorithm without clustering and manages to keep track of objects in challenging environments. Compared to a clustering JPDA algorithm the processing times are slightly higher, but the region trackers shows more robustness in densely cluttered scenarios where large clusters mean more processing time for the clustering algorithm.

Table of contents

List of figures

List of tables

1 Introduction

Ever since means of transportation were invented, people are trying to travel from one place to another without having to do any of the operating themselves. The most obvious examples are chariots or horse-carts that were driven by a dedicated driver, while the passengers could sit back and enjoy the travel. This is of course an example of what is now known as a taxi: You tell the driver where you want to go and he will take you there.

A drawback of this concept is that the driver is a person, who needs money as a wage and who will not be available at all times. Even as far back as the days of Leonardo da Vinci (approximately 1478), some form of automation was thought of to replace the driver [1]. Da Vinci's cart could be pre-programmed to drive a certain distance, turn and drive on. There is no recorded history of his invention ever being built in his days, but an enthusiastic group of scientist recreated his cart in 2004, proofing his concept is feasible [2].

Comfort and ease of use remained a main reason for fantasizing about automatic cars for a long time during the 20th century. Many drawings and sketches can be found from people in the 1920s-1950s that show a futuristic picture where people enjoy games or a newspaper while in an automatic vehicle. One advertorial displaying these visions, from the American *Central Power and Light Company*, is shown in Figure 1.

Figure 1. A family playing a game while traveling on a highway. This is a concept art from 1950 of a driverless car [3].

With computers invented and getting ever more powerful, this futuristic vision became more realistic. The focus and reason for developing an autonomous car was no longer comfort and entertainment, but safety. Automatic systems were introduced throughout the final decades of the twentieth century and carries on till today. Think of systems like cruise control, anti-lock braking systems, electronic stability control and voice navigation systems.

All of these systems have the goal of making driving easier or safer, but none of them can take complete control over the vehicle. Many automotive companies want to change that and are planning to introduce vehicles with autonomous systems as soon as 2020 [4]. The Delft University of

Technology intends to aid in the development of such vehicles and initiated the DAVI project (Dutch Automated Vehicle Initiative). The aim being to develop systems for autonomous cars and evaluate their performance and impact on the driver/operator as well as the environment.

To succeed in designing an autonomous vehicle, many systems need to be developed, one of which is the object tracking system. This system will interpret the sensor data and combine it into a single *map* of the environment with all the objects on it. Other systems can use this *map* to evaluate the safety of the vehicle and make driving decisions.

Tracking of objects is not a novel technology, aeronautic radars have been used for many decades and they all use some degree of tracking. In automotive applications there is less experience; only halfway during the 1990s were systems developed that needed some kind of perception of the environment. These systems were the first ACC systems (Automatic Cruise Control), which are able to maintain a certain distance from a leading vehicle.

The first systems were focused on single driving tasks (like distance keeping) and did not require an extensive tracker, because there are not many objects in the intended observation area. However as more and more tasks are aided, more sensor information is required, specifically a broader field of view (FOV). The simple trackers used in early driver aids are not able to cope with the high number of objects within the sensor range of typically around 200 m [5].

In line with the DAVI project, a survey was done on tracking algorithms that can be used in the vehicle [6], an abstract of which can be found in Chapter 2. From this research can be concluded that an accurate tracker like the JPDA (Joint Probabilistic Data Association) filter works very well, but computation time scales exponentially with the number of targets. In case of an autonomous vehicle, many objects are expected in the vicinity of the car, so a solution must be found to reduce the number of objects, or otherwise increase the speed of the algorithms.

One way of limiting the number of objects is filtering out the less important ones by constructing a region of interest. The objects inside this region can be tracked with a computationally heavy algorithm, while the other objects can be tracked with a much simpler algorithm. There are of course many ways to construct a region of interest and this thesis will be about finding an acceptable solution for this problem.

The questions to be asked are whether enough of the surroundings can be covered with the regions, how efficient they are used and what kind of challenges will arise from certain solutions. Five solutions are evaluated in Chapter 3, one of which is an approach not seen before in autonomous vehicle. In this chapter a comparison will be made of the methods and the following question will be answered.

> *What is the best way to divide the surroundings of an autonomous vehicle into smaller regions, with the aim of limiting the number of objects per region, to increase processing speed of the tracking-/association algorithm?*

From this analysis the conclusion is drawn that the novel method that uses extended maps to generate regions that follow the infrastructure boundaries should be investigated in greater detail. This is done in Chapter 4 and a complete description will be given about several rules to construct

the regions in many scenarios. The aim is to come up with a (limited) set of rules that will work in all traffic scenarios. The question in this chapter will be:

> *How can a novel method using extended maps be implemented for region splitting in an autonomous vehicle?*

In Chapter 5 the method is implemented in a MATLAB model and tested in several scenarios and the results will be presented. A discussion and conclusions are given in Chapter 6, and final recommendations are made for future research and implementation in a real autonomous vehicle.

With this research I hope that the future vision people had in the early twentieth century is one step closer to being implemented in road vehicles in the near future. And that soon we will be able to play a game on our way to visit family in the safe environment of our personal autonomous vehicle.

2 State of the art

Tracking of objects is not something new and many techniques have been used over the decades to improve accuracy of the trackers and to reduce computation times. The first systems to use tracking algorithms were aeronautical radars, which were used to track airplanes and monitor the air traffic, especially around airfields. Of course there were also military applications; to detect and track air movement of opposing parties.

For automotive purposes, tracking has not been around for very long, but with increasing technology ever more demand is put on detecting objects in the vicinity and making sense of more and more data. A literature survey was carried out [6] on several tracking and association techniques. Section 2.1 will summarize this report and use its conclusions to further develop a tracking strategy for an autonomous vehicle.

Section 2.2 will briefly describe how the Joint Probabilistic Data Association (JPDA) filter used in this project was created and demonstrate its workings. A tool to generate a realistic dataset of an automotive environment, with the aim to test the performance of the trackers, is presented in Section 2.3. The main problem for the JPDA algorithm is discussed in Section 2.4 and an existing solution is presented in Section 2.5. A brief introduction is made to a novel method for tracking which will be described in detail in Chapter 3. To compare the different methods, a performance indicator needs to be chosen, this is discussed in Section 2.6.

2.1 Tracking and association algorithms

This section will summarize the findings of a literature survey [6] conducted in line with the DAVI project about tracking algorithms and architectures to use in an autonomous vehicle. The problem of tracking objects in an automotive environment can be separated into two main parts: The system architecture to use and the data-processing algorithm itself. In this, the data-processing algorithm contains all processing done between the received sensor information and the output 'map' containing all objects around the vehicle.

Before describing how a tracking and association algorithm works, Section 2.1.1 is used to explain what encompasses this term with regards to this project. Section 2.1.2 will give the summary of existing system architectures and their respective drawbacks and benefits. The filtering and association methods are discussed in Section 2.1.3 and the conclusions and recommendations of [6] are summarized in Section 2.1.4.

2.1.1 What is a tracking algorithm?

The basic function of a tracking algorithm in an autonomous vehicle is to take the (raw) sensor data and combine this all into one *map* of the surroundings of the car, containing all objects. Based on this map, another algorithm is able to determine the safest route, avoid obstacles, and maintain the comfort for the passengers.

A tracking algorithm, or *tracker*, can be divided into several sub-systems that all do their own little task and then parse the data to the next part. For this project, the sensor data will be the input of the tracker and the output will consist of a list of (tracked) objects with all their relevant properties. The output can thus be used by decision making algorithms in the autonomous vehicle.

The DAVI research vehicle will be equipped with multiple radars [10], cameras, ultrasonic sensors, and an advanced GPS system with extended map information (eHorizon, [11]). Aside from this data all the normal vehicle data such as forward speed and accelerations (yaw, pitch, roll) are also within the reach of the tracker.

The input data for the algorithm may not always be the raw sensor data. For instance; in this thesis no object detection in camera images will be performed. Instead, it will be assumed that this process is executed by a dedicated processor and the data from the camera will be a list of possible targets and their location and other properties (if available). A similar approach could be taken with the ultrasonic sensors: A dedicated algorithm can combine the information from all sensors and output only the detected object location. An example of how such a processor may work can be seen in Figure 2.

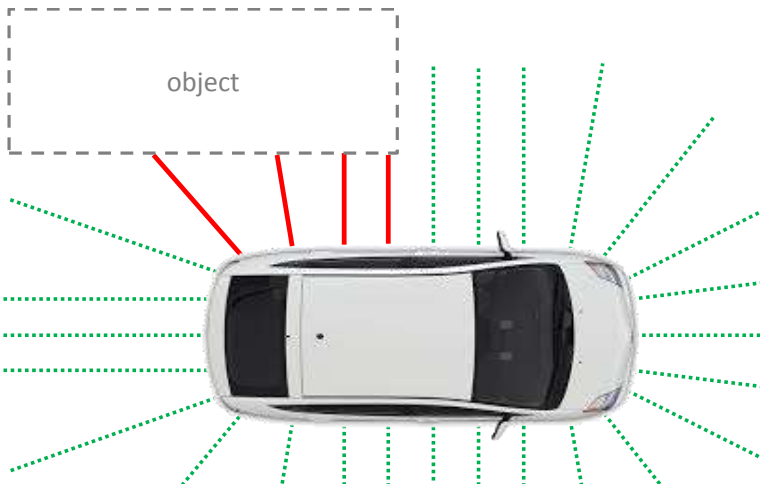

Figure 2. An example of how many ultrasonic sensors could be used to detect objects. Image taken from [6].

The input of the tracking algorithm will be the pre-processed sensor data. These measurements will be translated into a Cartesian World coordinate frame, such as an UTM-frame, because for this thesis all objects will be tracked in a World frame. The conversion must be performed, since all sensors are attached to the vehicle, which can also move around in the World frame.

The reason to use a coordinate system fixed to the World, instead of one connected to the vehicle is that in this way the objects can easily be overlaid on a geographic map of the area. Stationary objects in the World, like traffic signs, buildings and trees, will also be stationary in the output of the tracking algorithm. Note that it is not important for the tracker to know in which coordinate system it is working, as long as it is consistently used in the same system.

Of course it *is* important to take into account the measurement errors. When a measurement is translated from the sensor coordinate frame, into the World coordinate frame, the positioning error of the car has to be added onto the measurement error. Conversely, if the tracking algorithm would use a moving coordinate system (attached to the vehicle) the positioning error would have to be added to the location of the origin of this coordinate system. Either way; all measurement errors contribute to the outcome of the tracking algorithm, regardless of the choice of the coordinate system.

Objects that have been tracked before and are therefore stored in the vehicle memory are called *tracks*. These tracks are defined in the World coordinate frame, just like the converted measurements are. A very important step in the tracker is to combine the measurements and the tracks and determine which measurement may have come from which tracked object. This process is called *association*.

In order to associate the measurements with the tracks, a prediction of the position of the tracks must be made, which is done using a linear motion model. The next step is to match the measurements to these new estimated positions and see how well they match. This process can be very easy for a human to carry out, but causes all kinds of problems when a computer tries it. More on the why and how can be read in Section 2.1.3.

Once the tracks and measurements are connected together, the filtering process can begin. This means updating the state and covariance of the tracks for the next time step. A final step for the

tracking algorithm is to perform maintenance on the tracks: New tracks are initiated where new measurements occur and old tracks that had no updates for some time have to be deleted. After the track maintenance, the output of the tracker is complete. A schematic overview of this is shown in Figure 3.

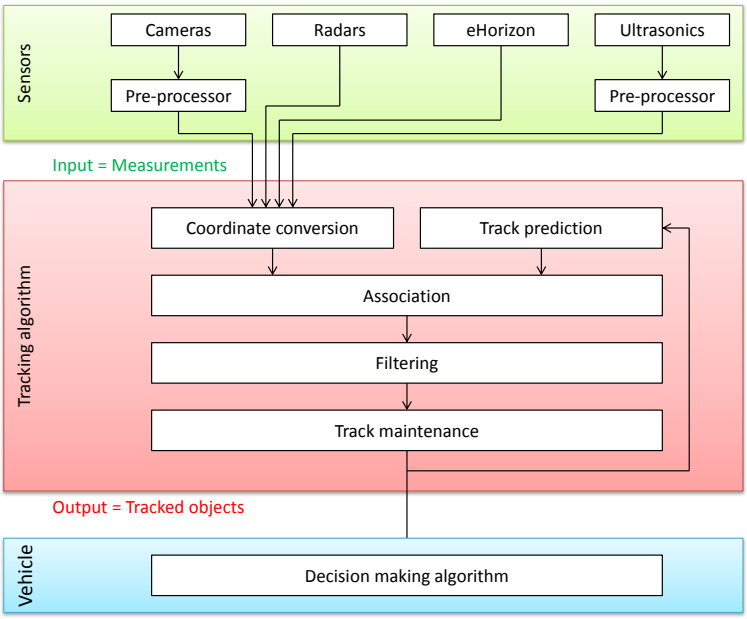

Figure 3. A schematic overview of a tracking algorithm. In this research the input will be the (pre-processed) sensor data and the output is the list of tracks. The parts bordered with dashed lines are not part of this research.

The different parts within the red solid rectangle of Figure 3 will be discussed in Section 2.1.3, but before that, the next section will take a deeper look into the general architecture of the algorithm.

2.1.2 System architecture

There are three main categories of tracking architectures according to [6], although some other sources (for example: [7]) may define more, different structures, these can always be summarized within the three main architectures: centralized, de-centralized and hybrid. The choice of which architecture is the best depends on what application it is used for, according to Hall [12]. This section describes the three main architectures.

The *centralized* architecture is the most basic one, because all the data is collected and processed by a single algorithm at once (see Figure 4). This approach was used in the early stages of tracking when multiple sensors were used for the first time. In this method it is very easy to combine sensor information at the lowest level to increase tracking performance, but it comes at the cost of a high bandwidth requirement to send the unprocessed data from the sensor to the tracking algorithm.

Another drawback comes into play when many objects exist and many sensors are used: the processing of all this data at once requires a lot of memory and leads to large processing times. In the early years of tracking this was not a real issue, because few sensors were used and those sensors did not output many objects. For the DAVI project the idea is to use five radars, each capable of detecting up to 94 targets, five cameras, ultrasonic sensors and of course all the normal vehicle

data. It is infeasible to have a centralized architecture with this amount of data at this point in time, simply because computers are not fast enough and memory is not large enough.

An example of a centralized architecture used in automotive systems is Volvo's CWAB-PD system [13], a system to maintain a safe headway and automatically brake for crossing pedestrians. The system uses a radar and a camera as complementary sensors. Another project using a similar setup, but this time with a far-infrared camera, is project EUCLIDE [14;15]. Both these systems work well within their respective design parameters, but have much less sensors and thus less data to process than a fully autonomous vehicle would have.

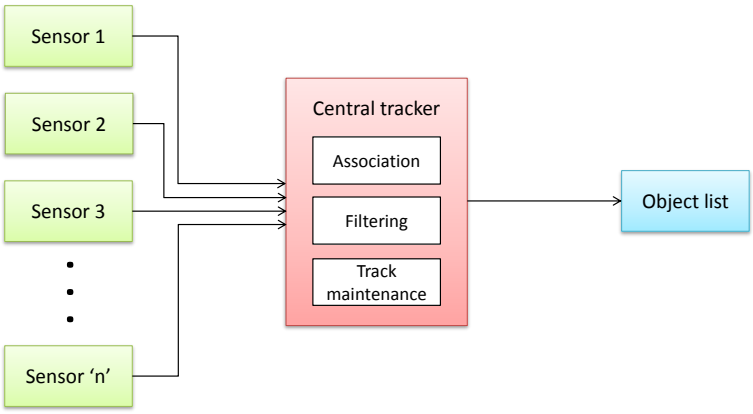

Figure 4. An example of a centralized tracking architecture. All sensor data is sent to a central point, where it is processed at the same time.

In order to reduce the amount of memory required and with that also reduce the computation times, the *de-centralized* method was developed [7;16] (Figure 5). In this method, the tracking takes place closer to the sensor. In fact, there are as many trackers as there are sensors. The data is tracked on a per-sensor basis and only the tracks are communicated to a central processer that fuses these tracks together into one output.

The obvious benefit of this is the reduction in memory requirements and processing times, but separating the sensors from each other prevents them from working together on the same object. For instance; a camera and a radar are complementary sensors. Radar is strong in distance measurements, but lacks in angular accuracy, whereas a camera can estimate an angle to an object very well, but a distance measurement is less accurate. By having these two sensors survey the same *field of view* (FOV), the measurements can be combined, resulting in a combined measurement that is more accurate than either sensor on its own could be.

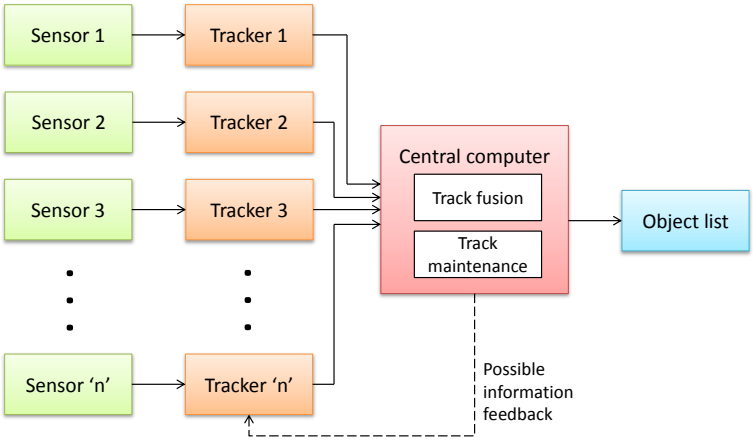

Figure 5. An example of a de-centralized architecture. Every sensor has its own tracker and the central computer only adds those together and performs the track maintenance. Some individual trackers may use information of previous tracks.

This is however not possible in a purely de-centralized architecture, because the tracking takes place *before* fusing the data together. So the individual trackers only have access to the raw data of their own sensor. A feedback from the central computer to the tracker *is* possible, but the information will be the tracks of the previous time-step, not the raw data of another sensor. An example of a track-feedback that may be beneficial is: The tracks may be used to predict the position of an object in the next camera frame. This in turn can be used by an object recognition algorithm in the tracker of a camera sensor, because now the general location of an object is known, and thus not the entire image has to be scanned for objects.

By using a de-centralized architecture, not only are the memory requirements and processing time reduced, but the system also is more robust than a centralized architecture. If an execution error occurs in a centralized algorithm, the whole system is down, but when an error occurs in one of the trackers of a de-centralized system, only that part of the system fails, while the rest remains operational. This *will* lead to performance degradation, but not to a complete failure.

Care must be taken when the tracks of the different trackers are combined, because whenever tracks are fused that belong to the same object, they are correlated. This will violate the assumption of uncorrelated error statistics that is present in most fusion algorithms. This problem, and solutions are described in [17]. Although solutions are given, it is easy to make errors in its implementation and prevention of the problem would be preferred.

The third architecture discussed is the *hybrid*, which contains all systems that do not belong in the previous categories. Mainly these will be systems where part of the layout is centralized, but another part is de-centralized. Figure 6 shows an example of a hybrid architecture, although some systems may look very different than this example, the idea is that some parts act like a centralized structure, while others are de-centralized.

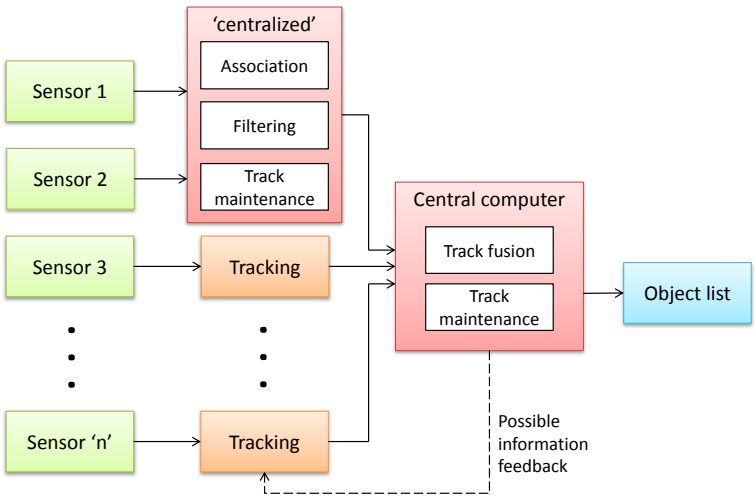

Figure 6. A schematic example of a hybrid architecture. Some parts can be recognized as centralized (sensor 1 and 2), whereas some other parts act like a de-centralized architecture.

An much applied example of a hybrid architecture in the automotive sector is to use a camera and radar at the front of the car in a centralized manner, while these tracks will later be merged with data from a rear-facing radar as with a de-centralized approach. Many, if not all, automotive tracking systems are of a hybrid nature.

Having both centralized and de-centralized parts means that the benefits of both structures can be used, typically leading to a system that is more accurate (because of collaboration between sensors) and faster (because of more efficient use of processing resources) than either of the techniques are on their own.

However, there are also some downsides to the hybrid architecture. Most importantly; the sensor structure needs to be defined, so that the system 'knows' which sensor data to treat as centralized and which as de-centralized [18]. Hall described it as follows: *"While the hybrid architecture provides the most flexibility, it also requires overhead to monitor the fusion process and select between data and state vector fusion."* [12].

Despite the overhead that the hybrid architecture has, practically all of the teams in the 2007 DARPA Urban Challenge opted to use this system layout [19-21]. This challenge was organized to demonstrate the advance of technology for autonomous vehicles. Teams, mostly universities, were asked to design an autonomous vehicle and test it in a real-life environment. Many different sorts of hybrid architectures were used, designed with several strategies in mind.

The properties of all three architectures are summarized in Table 1. Since the vehicle in this project will be equipped with many sensors, the centralized architecture can be ruled out. The choice for the *hybrid architecture* is made, because it will give more flexibility to develop sensors and sub-systems for specific purposes. The extra overhead is expected to be negligible compared to the gains that can be made with a more flexible system.

Architecture	Benefits	Drawbacks
Centralized	• Connecting data from different sensors is easy • Simple implementation	• Processing is time- and memory-consuming • Required communication bandwidth
De-centralized	• Less communication bandwidth required • Processing can be done in parallel • System can be made modular and robust	• Collaboration between sensors difficult • Correlation between sensor tracks
Hybrid	• Flexible and adaptable architecture • Benefits of centralized and de-centralized methods	• Central managing processor needed • System structure changes with sensor upgrade/change

Table 1. Properties of the three different architectures.

2.1.3 Filtering and association techniques

The main part of a tracker is the association of measurements with tracks and the filtering that takes place after that. This section will elaborate on which techniques are currently used. First the filtering will be discussed, after which some modern association methods will be presented.

Filtering data is not much more than comparing a prediction with a measurement and correct for the error that is present. Generally the prediction will be made using a linear motion model of the process. The most well-known filtering process is the Kalman filter, introduced in the 1970s [22]. There are some older methods that are simpler, such as the alpha-beta filter [23;24], but the performance of these filters is much less than the Kalman filter. Generally they are not used anymore for high-precision applications.

There are also some methods that are capable to cope with non-linear systems, such as the extended Kalman filter (EKF) [25]. The non-linear process model can however lead to problems, especially when the model is not very accurate, or the noise levels on the measurements are high. Of course the non-linear models also bring more computations with them, leading to a generally slower filter.

An interactive multiple model (IMM) approach [7;26;27] is sometimes used to try and solve the problems with maneuvering objects. In essence this works as a normal Kalman filter, but with multiple linear process models, instead of just one. A Markov chain is used to predict which model represents the measurements best. When three different process models are used, the processing time of the filter is approximately four times higher than a single model Kalman filter [7].

All considering, the standard Kalman filter seems to be the best choice for the vast majority of problems, including tracking in an automotive environment. In fact, the filter is so popular that all of the most common association techniques are based on the standard Kalman filter, with a single linear model. Figure 7 shows a schematic representation of the Kalman filter process.

Starting on the right side of the figure, a prediction is made using a linear motion model, based on the state from the previous time step. First a prediction is made for the state ($\hat{x}_{k|k-1}$), in the case of this project the position and velocity of an object. Then a prediction for the covariance matrix ($\hat{P}_{k|k-1}$) is made, indicating the expected error margins of the state.

These predictions are compared to the measurements (z_k) in the bottom of the left side of the figure(leading to $\tilde{y}_k$ and S_k). The Kalman gain (K_k) is computed, which is basically the correction factor to be applied to the previously made predictions. And finally the state ($\hat{x}_{k|k}$) and covariance

($\hat{P}_{k|k}$) are updated using the Kalman gain. The updated state and covariance are the output of the filter and will also be used in the next time step to make a new prediction.

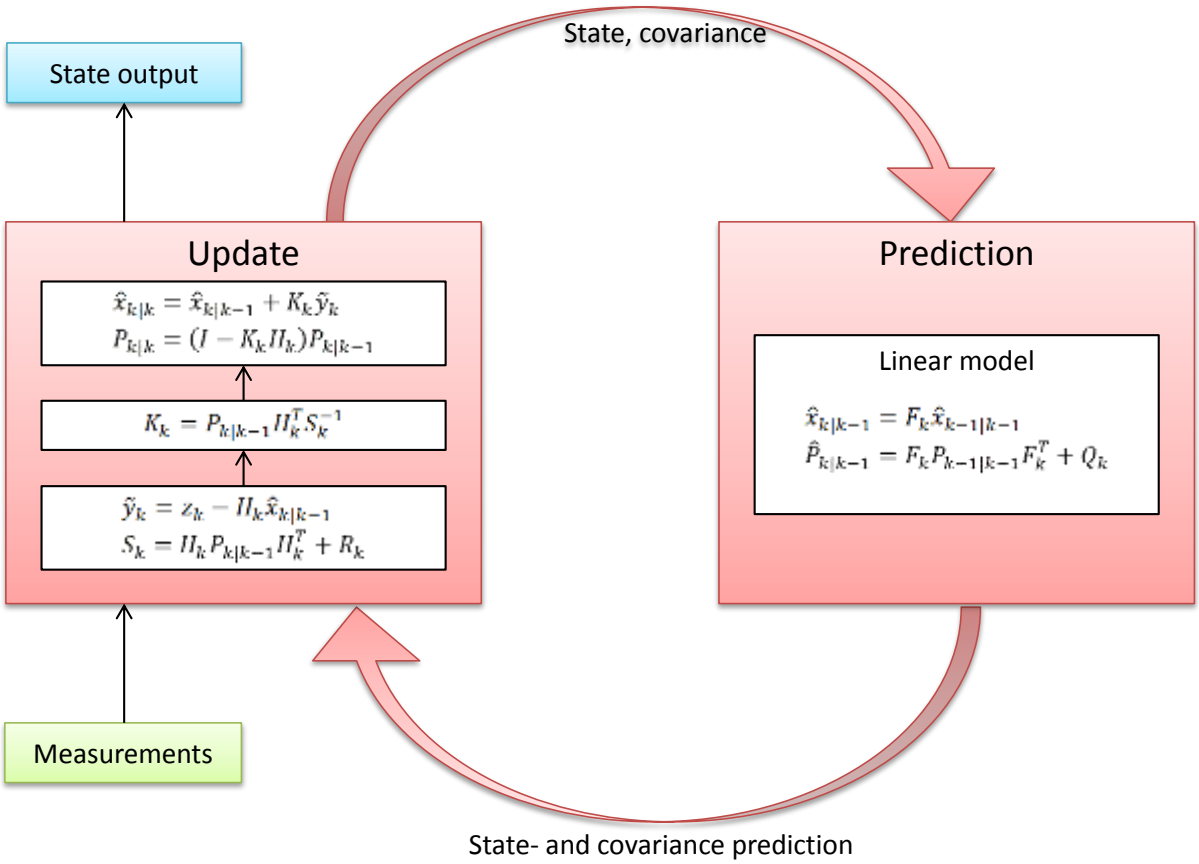

Figure 7. A schematic representation of the Kalman filter process. First a prediction is made, this is tested against the measurements, a correction is made and the output is given. Image taken from [6].

Before filtering can take place, a decision must be made about which measurement could belong to which (known) track. This process is called association and is generally the most time-consuming process in a tracker.

The nearest neighbor (NN) approach is the most simple technique there is. The measurement that is observed to be the closest to the predicted location of an object is assumed to be the correct one. Figure 8 depicts one track prediction (circle) with five measurements (crosses). The measurement indicated with an arrow will be used in the filtering process. This method works very well in a low-clutter environment, where the measurements are sufficiently accurate and consistent. However, when the measurements are noisy, missing or there are many false measurements, the method can lead to incorrect results very easily.

Using the NN approach also prevents having multiple measurements for one track, for instance when there are multiple sensors that cover the same FOV. The method usually only takes the position into account, which could lead to problems when a false measurement with an entirely different velocity vector happens to be closest to the predicted location of the track.

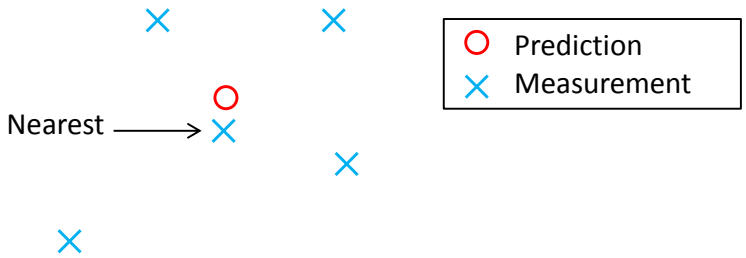

Figure 8. The nearest measurement will be taken as the true measurement in a nearest neighbor (NN) association process.

A new method of association was introduced by Yaakov Bar-Shalom [7;28] called the *probabilistic data association* filter (PDA). For this method a confidence ellipse is created around the predicted track, which is based on the covariance matrix. All measurements within this ellipse are weighted according to their likelihood of belonging to the track, and all will have a part in the final prediction of the algorithm (Figure 9).

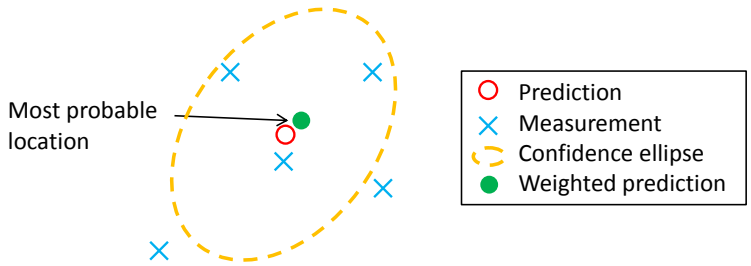

Figure 9. All the measurements within the confidence ellipse are weighted, creating a prediction that can depend on multiple measurements.

In more detail the process is as follows: First of all the measurements within the confidence ellipse (m_k) the probabilities (β) of belonging to the object are calculated (Equation (1)). The likelihood function ($\mathcal{L}$) is in essence a scaled normal-function, meaning the measurements closer to the object are weighted much more heavily than the ones close to the edge of the confidence ellipse. The detection probability (P_D) and the probability of the measurement existing within the confidence ellipse (P_G) are used to help scale the probabilities properly.

$$\beta_i(k) = \begin{cases} \dfrac{\mathcal{L}_i(k)}{1 - P_D P_G + \sum_{j=1}^{m_k} \mathcal{L}_j(k)} & i = 1, \dots, m_k \\ \dfrac{1 - P_D P_G}{1 - P_D P_G + \sum_{j=1}^{m_k} \mathcal{L}_j(k)} & i = 0 \end{cases}$$
$$with:$$
$$\mathcal{L}_i(k) = \frac{\mathcal{N}\left[z_i(k); z_{k|k-1}, \mathrm{S_k}\right]\mathrm{P_D}}{\lambda}$$
$$P_G = P\{z_{k-1} \in \mathcal{V}(\mathrm{k}-1, \gamma)\}$$
$$P_D = detection\ probability \tag{1}$$

Although the filtering process is not different than in the Kalman filter, the updating uses slightly different parameters. Instead of using the Kalman gain and the measurement error, the weighted equivalents are used. In Equation (2) the update formula is posted, where the Kalman gain is replaced with weighted gains and the measurement error is replaced with the weighted average of

all measurement errors within the confidence ellipse. Note that the possibility of none of the measurements belonging to the object is also taken into account.

$$\begin{aligned}
&\hat{x}_{k|k} = \hat{x}_{k|k-1} + W_k \cdot v_k \\
&with: \\
&v_k = \sum_{i=1}^{m_k} \beta_i(k) \cdot v_{i,k} \\
&v_{i,k} = z_{i,k} - \tilde{z}_{k|k-1} \\
&and \\
&W_k = P_{k|k-1} \cdot H'_k \cdot S_k^{-1}
\end{aligned} \tag{2}$$

The Covariance update also changes to encompass the weighted equivalents of the original Kalman function (Equation (3)).

$$\begin{aligned}
&P_{k|k} = \beta_0(k) \cdot P_{k|k-1} + [1 - \beta_0(k)] \cdot P^c_{k|k} + \tilde{P}_k \\
&with: \\
&P^c_{k|k} = P_{k|k-1} - W_k \cdot S_k \cdot W'_k \\
&\tilde{P}_k = W_k \cdot \left[\sum_{i=1}^{m_k} \beta_i(k) \cdot v_{i,k} \cdot v'_{i,k} - v_k \cdot v'_k \right] \cdot W'_k
\end{aligned} \tag{3}$$

The output of the PDA filter are more robust than the standard Kalman filter while requiring approximately twice the computation power [7;28]. In a cluttered environment this is therefore a good choice, especially in traffic situations where safety plays an important role. A shortcoming of this method is that, like the Kalman filter, it is actually an algorithm for only one object. If more objects need to be tracked, every object will have its own tracker.

In order to deal with more than one object the joint probabilistic data association filter (JPDA) was developed [7;29;30]. This method does not only allow multiple objects to be tracked at the same time, but also computes the most likely distribution of the measurements among the tracks. To illustrate the workings, a simple scenario is sketched in Figure 10.

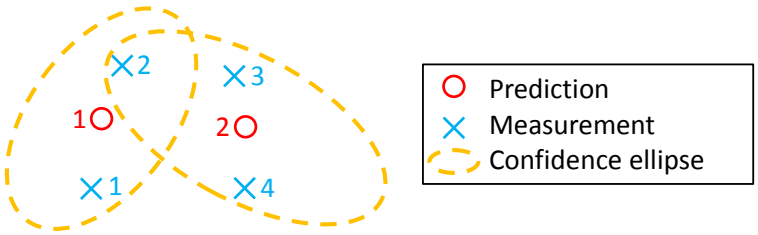

Figure 10. When more objects can share the same measurements, a joint PDA can be used. This scenario is used in the text as an example. Based on an image from [7] .

After determining the predictions and the confidence ellipses of the tracks, a validation matrix is constructed (Ω). This matrix has as many rows as there are measurements and the number of columns is the number of tracks plus one. The first column represents *track 0*, or the case that a measurement is an artifact or false measurement and thus does not belong to any real target.

In the validation matrix a 1 is written when a measurement is within the confidence ellipse of a track, otherwise a 0 is added. This means that the first column, representing a measurement not belonging to any track is always filled with ones. Following this principle, the validation matrix can be constructed for this scenarios as shown in Equation (4).

$$\Omega = \begin{array}{cccc} & t & 0 & 1 & 2 \\ j & & & & \\ 1 & & 1 & 1 & 0 \\ 2 & & 1 & 1 & 1 \\ 3 & & 1 & 0 & 1 \\ 4 & & 1 & 0 & 1 \end{array} \tag{4}$$

From the validation matrix the scenarios are extracted and their respective matrices are formed. To do this, two main rules are applied to the original validation matrix. 1) There can only be one unit-value per row of the matrix. This represents the fact that a measurement can only belong to one track at a time. 2) There can only be one unit-value per column of the matrix. This can be explained as there can be only one measurement per track, so no double measurements.

As an example, three of the resulting scenario matrices are shown in Equation (5). From these matrices it can be seen that the second track has any of the last three measurements, but never more than one at the same time; there is never more than one 1 in any column. Only three of the matrices are given, but in reality there are 11 possible scenarios.

$$\hat{\Omega}(1) = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \hat{\Omega}(2) = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}, \hat{\Omega}(3) = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5}$$

For every scenario, a probability is computed, which indicates the likelihood of a specific scenario to occur. Using the formulas in Equation (6), the probabilities are calculated. Note that this value has much to do with the distance between the measurements and the tracks, similar to the PDA calculations described before.

$$\begin{aligned} &\mathrm{P}\{\theta(\mathrm{k})|\mathrm{Z}^{\mathrm{k}}\} = \frac{1}{\mathrm{c}} p[Z(k)|\theta(k), m(k), Z^{k-1}] P\{\theta(k)|m(k)\} \\ &with: \\ &p[Z(k)|\theta(k), m(k), Z^{k-1}] = p\left[z_j(k)\middle|\theta_{jt_j}(k), Z^{k-1}\right] \\ &p\left[z_j(k)\middle|\theta_{jt_j}(k), Z^{k-1}\right] = \mathcal{N}\left[z_j(k); \hat{z}^{t_j}(k|k-1), S^{t_j}(k)\right] \end{aligned} \tag{6}$$

After having obtained these event probabilities, the algorithm continues using a normal PDA process, where the measurements within the confidence ellipses of the tracks are weighted according to their distance from the track estimation. A schematic representation of the whole JPDA algorithms can be seen in Figure 11.

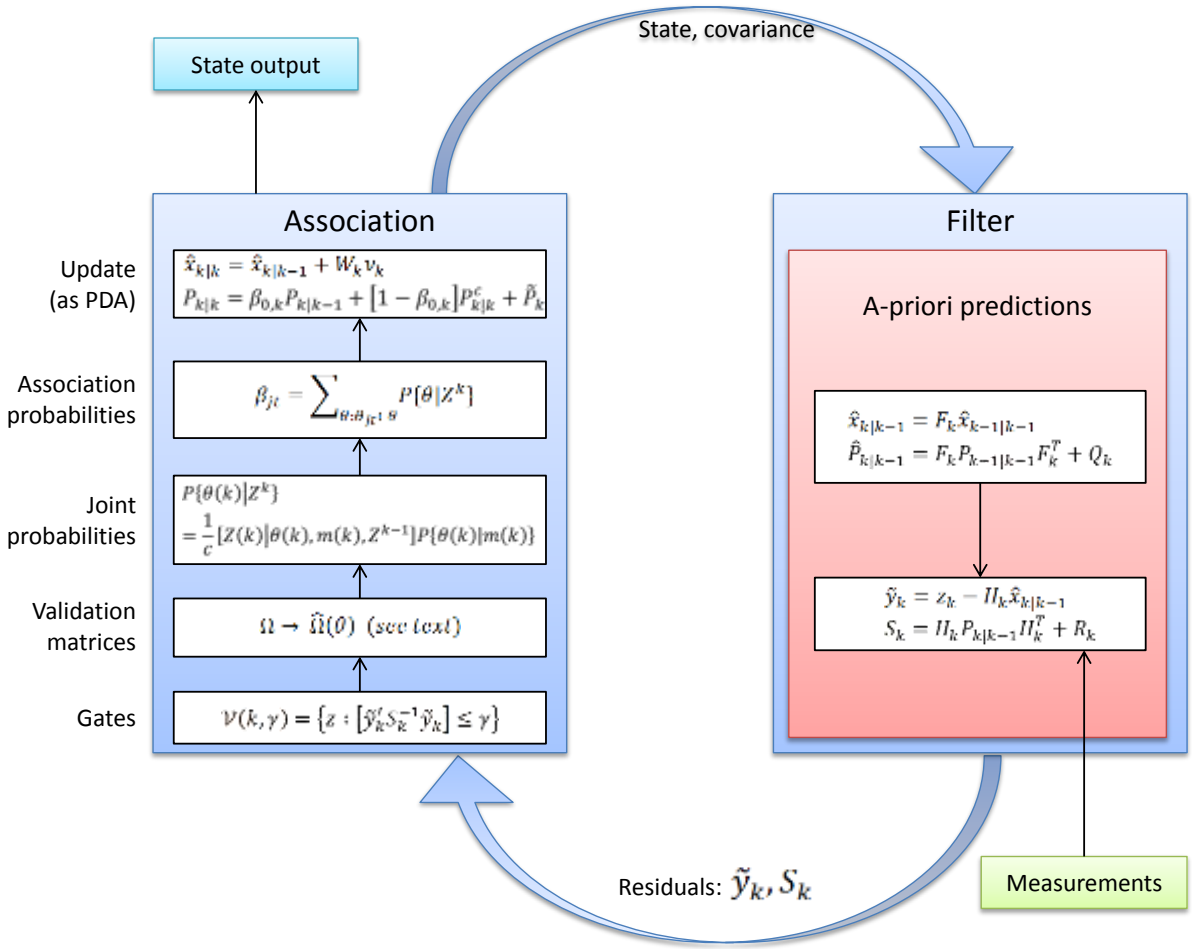

Figure 11. Schematic representation of the JPDA process. The old tracks and measurements are used to predict the new states, gating is performed, the validation matrices are constructed and then the event probabilities calculated. Note that the filter on the right (red rectangle) contains a normal Kalman filter process. Image taken from [6].

From this figure it is also visible that the association part of the algorithm takes quite a bit of space. This is not only true in this schematic, but also in the implementation and processing times of the different parts of the algorithm. Where a standard Kalman filter with nearest neighbor association can be programmed within ten lines of code, the JPDA requires up to a tenfold of that. Concerning speed; especially the probability calculations are heavy on computation times. In the next section some possibilities of overcoming this problem are presented.

Of course there are not only drawbacks to JPDA. The benefit over NN is that it works much better in cluttered environments *and* it is better capable of keeping track of objects that are close together or cross each other's paths [7;30]. More on the performance of the JPDA algorithm, including some experiments can be found in Sections 2.2-2.6.

2.1.4 Improving computation times using simplifications

There are several ways to solve the shortcomings of the JPDA filter. In [6] the focus has mainly been on reducing the computation times of the probability equations, but it is also possible to split the surroundings into region and have a tracker for each area. In this section the findings of [6] are presented. Clustering and region solutions are described in Sections 2.5 and Chapter 3.

The two main methods of reducing the processing time of the probability equations in JPDA are more strict gating, and simplifying the formulas. With stricter gating [31;32], it is possible to have less measurements within the confidence ellipses of tracks and thus there are less scenarios and less computations to be done. This method is prone to missing important situations, especially when the gates are set near the limit. It is easy to exclude a measurement that should have belonged to a track, but is just outside the gate.

A method based on this theory is to only compute the probabilities for the *n* closest measurements and assume that the closeness represents the likelihood of a measurement belonging to that track. As with the strict gating, this order statistics approach can be susceptible to missing out on important data.

In order to not throw away measurements, but increase the speed in other ways, many simplifications of the probability calculations are proposed [33-36]. What they all have in common is that the probabilities are no longer calculated using Bayesian formulas or Gaussian distribution assumptions, but use much simpler equations. For instance the *fast JPDA* (FJPDA) algorithm [35] will scale down the probabilities whenever the measurement is within more than one confidence ellipse.

$$\bar{\beta}_{\mathrm{j}}^{\mathrm{t}}(k) = \left(1 - \frac{n-1}{H \cdot N_t}\right) \beta_j^t(k) \tag{7}$$

With H a weighting coefficient and N_t the number of gates the target is in. With n also the maximum number of measurements to take into account can be limited, leading to a much faster algorithm. Although no experiments were performed in the literature survey [6], A large scale comparison was conducted by Pulford [37]. The conclusions from that study show that most of the methods indeed reduce the computational complexity and thus should run faster on the same hardware. However, nearly all of the faster methods also show a deficit in performance.

In the recommendations of [6] The *cheap JPDA* (CJPDA) method [33] is selected as the preferred algorithm to use in future work, because it can run slightly faster than a normal JPDA filter due to simplifications and a mild form of gating, while still maintaining a relatively good performance. However, the performance gain is still limited compared to what other methods show, such as clustering or dividing the measurement area into regions. For the purpose of this thesis, a JPDA filter was constructed to test whether clustering and region-splitting can indeed increase performance.

The development and proof of concept of the JPDA algorithm is presented in Section 2.2, an experiment with realistic simulation data is conducted to show the JPDA works in traffic scenarios (Section 2.3). During these simulations some problems occurred, which will be described in Section 2.4, while an existing solution to these problems (clustering) is presented in Section 2.5.

2.2 Creating a JPDA algorithm

To be able to test the effect of clustering and using regions on computation times, a base JPDA algorithm was written. In this section the process of creating a MATLAB function that can track using the JPDA method. Several simple tests are performed, to check the workings and the performance against results from other researches.

In order to compare the results in this thesis with those previously obtained by other researchers, a simple Kalman filter is created. It can be directly compared with the JPDA algorithm, because the inputs, motion model, and measurement model are the same. All other parameters required for the filtering, such as noise estimations are also the same.

2.2.1 The basic form of the function

The MATLAB functions are built to have three inputs: The old data, the new data, and the fixed parameters. The old data (tracks) consists of the information about objects that were previously tracked. The new information is the set of measurements, if needed converted to the same coordinate frame as the tracks. The fixed parameters contain the motion model, measurement model, estimated noise levels and other parameters that are needed by the tracking algorithms. The output is the new set of tracked objects.

By structuring all the tracking algorithms with these inputs and outputs, it gets easier to exchange modules with different algorithms. If a different tracking algorithm is required, the call to the module can be changed, but the rest of the code (initializing, variable scaling, etc.) will remain the same. A more detailed description of the used MATLAB code can be found in Appendix A.

2.2.2 Proof of concept testing

The Kalman filter and JPDA algorithm are tested using an artificial dataset with three objects. The algorithms are using the previously explained methods (Section 2.1.3). The goal of this test is to show the accuracy of the trackers and test their base performance. The object motion is simulated in MATLAB, giving the *ground truth* motion. Then a measurement set is constructed by adding some noise to the ground truth data. The paths of the objects are described in Table 2.

Property	Value / Description	Units
Object 1 start position	[0, 0]	m
Object 1 velocity	[20, 10]	m/s
Object 2 start position	[0 100]	m
Object 2 velocity	[20 -5]	m/s
Object 3 start position	[0, 50]	m
Object 3 linear velocity	[20, 2.5]	m/s
Object 3 sine wavelength	200	m
Object 3 sine amplitude	50	m
Position noise (all directions)	Independent Gaussian noise, 5 standard deviation	m
Velocity noise (all directions)	Independent Gaussian noise, 2.5 standard deviation	m/s
Measurement frequency	15	Hz
Simulation time	10	s

Table 2. Properties used for the proof of concept simulations. The values and units are chosen to represent an automotive-like scenario.

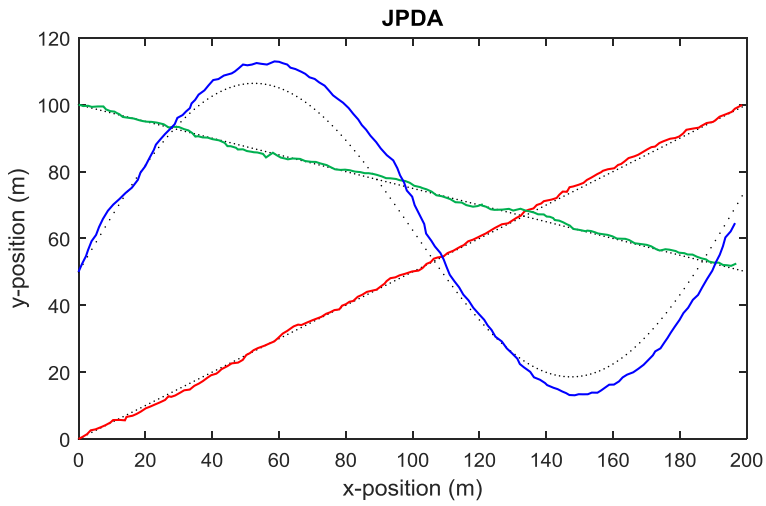

Figure 12. Results of the JPDA tracker with the proof of concept data. Black dots are ground truth positions, colored lines the tracked positions.

The results of the JPDA tracker with this dataset are shown in Figure 12. There is some overshoot at the tips of the sine-shaped path, but overall the colored lines (tracks) are very close to the ground truth (black dots). The results for the Kalman filter are not visualized, because they is no visible difference. To show how close both trackers are, the root mean square (RMS) error is calculated for all objects. These errors are shown in Table 3. The errors are very close to each other, so the conclusion can be drawn that with perfect association, the performance of the trackers is equal.

	Kalman filter error (RMS, m)	**JPDA filter error (RMS, m)**
Object 1 (red)	1.31	1.26
Object 2 (green)	1.36	1.33
Object 3 (blue)	2.32	2.47

Table 3. RMS errors for the three objects as tracked with the Kalman filter and JPDA algorithm.

In the previous experiment the measurements were ordered in such a way that the first measurement always corresponded with the first track. To demonstrate the capability of the association part of the JPDA filter, the same measurement set is used in the next experiment, but now the measurements are placed in a random order. Unsurprisingly this gave the exact same solution and the exact same values for the RMS errors.

After establishing that the accuracy of the JPDA filter is in accordance with the expectations, the question remains how fast it is. According to literature [7] a JPDA filter requires about three times the computation time compared to a Kalman filter. Figure 13 shows the processing times over the entire simulation. The JPDA algorithm has a more variable cycle time, because sometimes there are more objects within a confidence ellipse, which increases the number of calculations to be made.

Taking the averages of the cycle times gives a good idea of the performance deficit of the JPDA filter compared with the Kalman filter. The JPDA algorithm is approximately 3.3 times slower $\left(\frac{t_{JPDA}}{t_{KF}} = \frac{5.817 \cdot 10^{-4}\ s}{1.739 \cdot 10^{-4}\ s} \cong 3.3 \times\right)$, but it should be noted that the algorithm also performs the association, whereas the Kalman filter does not. The value of 3.3 times slower is close to the expected increase in cycle times [7].

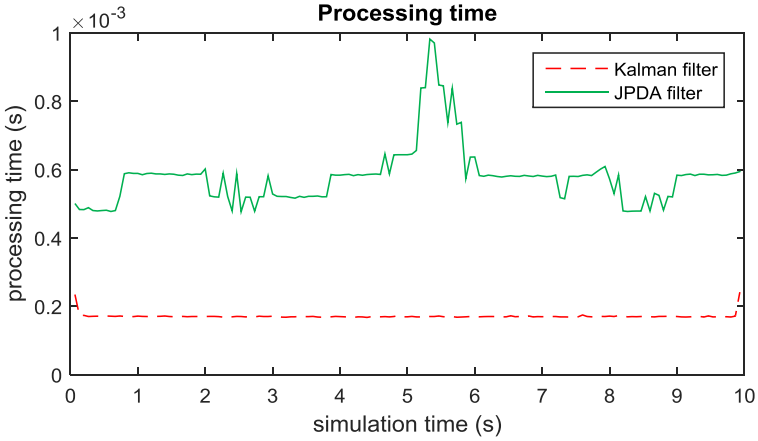

Figure 13. Processing times of the Kalman filter (red dashed) and JPDA filter (green solid) during the proof of concept simulation.

Another test for the algorithm is to see how it copes with false measurements. These measurements are artifacts of the sensor or the environment that generate measurements in places where there is no actual object, such as radar reflections off buildings. It is simulated by adding a random number of measurements with random position and velocity to the set of measurements. Table 4 summarizes the properties of the false measurements.

	Value	**Units**	**distribution**
Number of false targets per time step	1 – 15	[-]	Random uniform distribution
x-positions	0 – 200	m	Random uniform distribution
y-positions	0 – 150	m	Random uniform distribution
x-velocities	-30 – 30	m/s	Random uniform distribution
y-velocities	-30 – 30	m/s	Random uniform distribution

Table 4. Specifications of simulated ghost measurements.

In Figure 14 some of the intermediate results are shown. First and foremost, it can be seen that the results are only changed a little compared to the scenario without false measurements. Although the errors are slightly larger (1.75, 1.74 and 2.60 m RMS errors), the algorithm never loses the objects. The figure also shows the randomness in the number of false measurements and the positions; sometimes there are many false alarms, sometimes only a few; some of the false measurements are close to the objects and some far away.

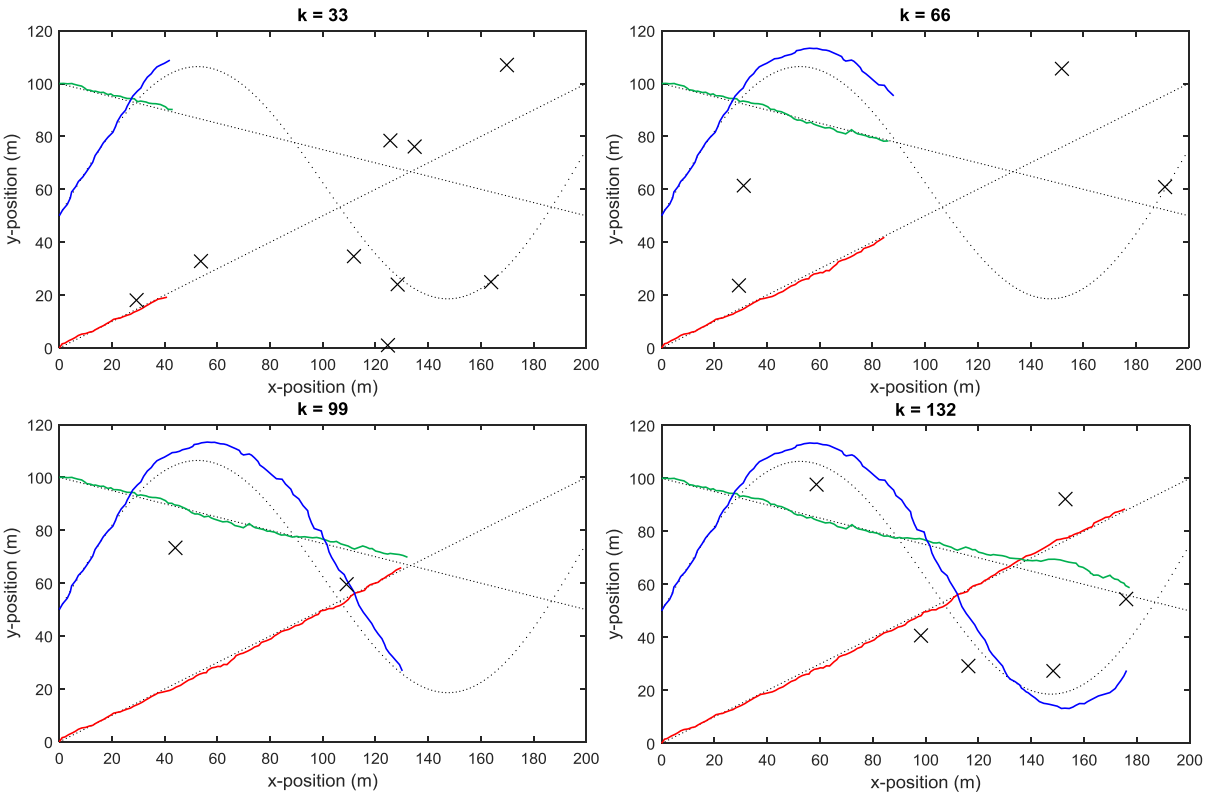

Figure 14. Intermediate results of the JPDA filter for the scenario with false measurements. The graphs display the situation after every 33rd frame (out of 150). Black crosses are false measurements, the correct measurements are not shown.

2.2.3 Track maintenance

An important part of tracking is track maintenance; knowing when to start a new track and determining when an object is no longer of importance. In order to do this, the tracks are accompanied by a counter. The counter is increased whenever a track has a measurement associated to it and decreases if this is not the case.

A new track will be initialized whenever one of two situations occurs. First of all if there are no known tracks in memory, while there are measurements. And secondly, if a measurement is not used during the association phase of the JPDA tracking algorithm. If one of these scenarios are true, a subroutine is called that will take the measurements and summarize them into a series of tracks.

If an object is not measured, the track is updated by using a filter action only. This means that the linear motion model dictates the prediction in the next time step. Since the linear model is an approximation, this could lead to a large difference between the track in memory and the real-life object. In order to make sure that these situations do no occur, the counter will not add indefinitely. A maximum number has to be defined and the counter is updated as shown in Equation (8).

$$\begin{aligned} counter(k) &= \min(counter(k-1)+1; counterMax) \quad && if\ track\ associated \\ counter(k) &= \min(counter(k-1)-1; counterMax) \quad && if\ track\ not\ associated \end{aligned} \tag{8}$$

The parameter `counterMax` can be set as high or low as the user requires, usually it is sufficient for a track to stay in memory for about one second after its last measurement association. In practice this comes down to a maximum of 10 to 30 for the counter, depending on the sensor data rate.

After the tracking and association process is finished all track counters are checked. If a counter has reached zero, the track is deleted. A track is thus deleted if there have been no associations for a specified time.

This method of track maintenance is not the most high-tech solution, but it is robust enough for the purpose of this research. In this thesis it is not the tracker itself that gets the main focus, rather the possibility of increasing the processing speed by using clustering or regions. The fact that the JPDA algorithm may not be the most efficient is less important, since all the algorithms will use the same JPDA base.

2.2.4 Algorithm assumptions

There are a few assumption made while creating the JPDA algorithm for this project. These assumption mean that the algorithm is not perfect, nor is it the most efficient JPDA filter currently in use. The focus of this thesis is to show the difference in computation times for several clustering and region splitting techniques. As long as the JPDA base algorithm used for all those methods is the same, conclusion can be drawn about the relative processing times.

For this project, the algorithm will track objects in a Cartesian World frame. In real life the UTM method of positioning can be used. Using this frame means that the autonomous host vehicle (ego vehicle) will also move and thus the sensors are not stationary. This brings complications which are solved by having two main assumptions.

First of all the confidence ellipses which act as gates for the association of measurements are always aligned with the axes of the World frame. In reality the confidence ellipses turn with the direction of travel of the tracked objects, or with the specified accuracies of the sensors. Implementing these rotations should not lead to any problems, but to keep the focus on the clustering / region splitting, it has been decided to assume that the effect of turning the confidence ellipses is negligible.

A second assumption is that the GPS data of the host vehicle is perfect. This data is used to convert the measurements, which are in the sensor coordinate frame, through the vehicle coordinate frame, into the World coordinate frame. The accuracies of the sensor measurements are of course translated through these frames, but the errors in the ego vehicle position, heading and speed are neglected. The errors in GPS data that are present in real data can of course be estimated and compensated for.

Since it is not possible to do experiments with a real vehicle within the time-frame of this research, and therefore the algorithms will only be tested with simulated data, it has been decided to assume perfect GPS data and show the effect of clustering and region splitting in a more controlled environment. However; some of the data (eHorizon) used by the region tracker will be recorded from an actual road test. More on this will be discussed in Section 5.1.

2.3 Realistic simulations with PreScan

More realistic scenarios are simulated using the software tool PreScan, which is normally used to test *advanced driver assistance system* (ADAS) controllers. For this section two environments were created and used to generate more realistic simulated measurements than the simple three objects from the previous examples (Section 2.2). More details on the PreScan software can be found in 0.

PreScan can be used to create a three dimensional environment with road segments, vehicles and stationary objects. Figure 15 shows a screen-capture of the highway scenario with annotations of some interesting features. All kinds of objects are available to create scenarios, such as (moving) vehicles, stationary objects (buildings) and all kinds of roadside objects like guardrails or overhead gantries for traffic signs.

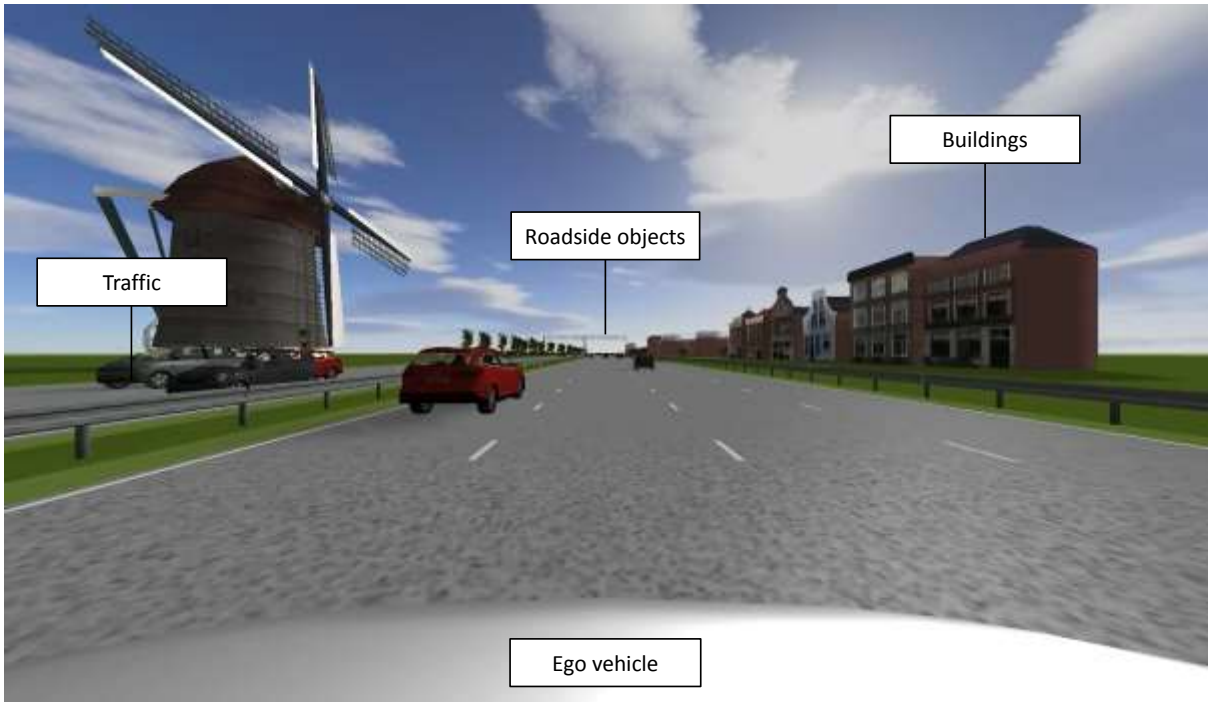

Figure 15. A screen-capture from the PreScan software with annotations.

The main feature of PreScan is however that it can be used to generate sensor data from these scenarios. The sensors can be placed on the host vehicle and the sensor parameters such as sample frequency, accuracy and field of view can be adjusted. For ADAS applications there is also the possibility to act on the information while the simulation is running, so that the performance of these systems can be tested. Since this research does not require a feedback loop to act on the environment, PreScan is only used to generate datasets of realistic traffic situations.

2.3.1 Highway scenario

The first scenario is a simple highway, with several cars entering and exiting the main road and some overtaking taking place. Some of the frames are visible in Figure 16 with the tracked objects also plotted. The blue dot is the host vehicle, with the grey dotted lines representing the FOV of the radar sensor used. Black crosses are the locations of the measurements and the other colored dots are the tracked objects, with their velocity vectors indicated as blue lines.

When first initiated an object is red, as can be seen in the image of frame one. After the counter for the tracks reach a certain number, the color changes to orange and when the counter reaches its maximum value, the dots turn green. In frame ten the objects are tracked long enough to turn orange and in frame 38 they are green. Also in this frame, a new object enters the FOV of the host vehicle; it is initiated as a red dot.

This slowly overtaking object is still visible in frame 100, where it is tracked for some time and is represented as a green dot. Another vehicle enters the highway, this time using an on-ramp, coming from the right side. In the last frame several thing happen: One new car enters, again overtaking the host vehicle on the left side. One car uses an off-ramp to exit the highway and one vehicle is overtaken by the host vehicle. All of those three cars are indicated with an orange color. In the case of the entering car it is because it is only tracked for a limited number of frames. But in the case of the exiting vehicles; their counter has decreased while there were no measurement updates for some time.

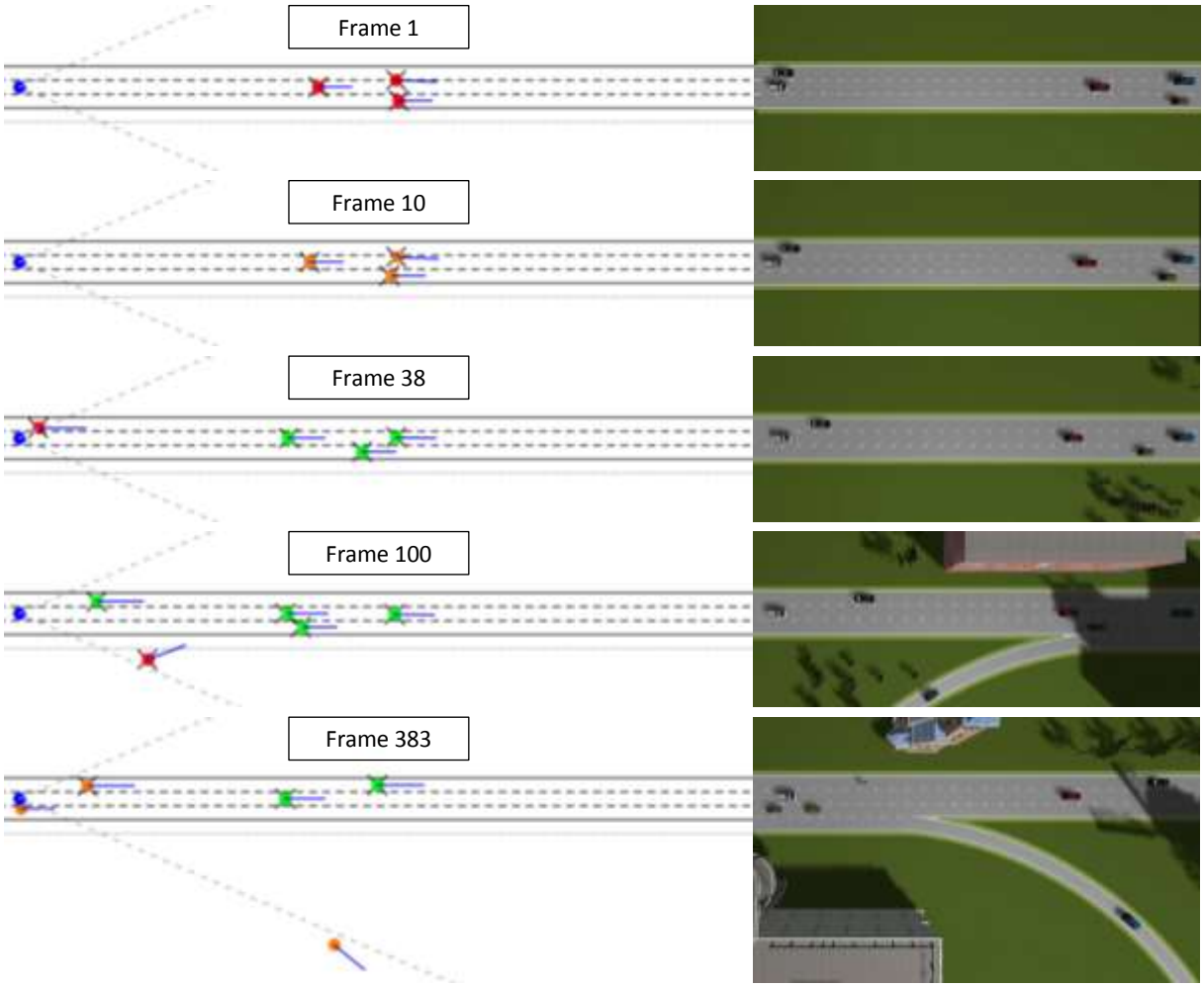

Figure 16. Top views of different frames (1, 10, 38, 100 and 383) of the JPDA tracker in the highway scenario. Left: MATLAB data, right: PreScan top view.

2.3.2 Urban scenario

A second scenario to verify the tracker in an automotive environment is an urban environment. The vehicle speed is much lower and the corners tighter. There are now also people walking in the

scenario. Figure 17 shows the results of this scenario, where in this case the traveled path of the host vehicle is represented by black dots. Whenever an object has reached its maximum with the counter, the green dot is plotted, but is not removed afterwards. This means that in the plot it is always visible where an object has been detected. This is only a matter of plotting, the tracks are still deleted in the algorithm once the counter reaches zero.

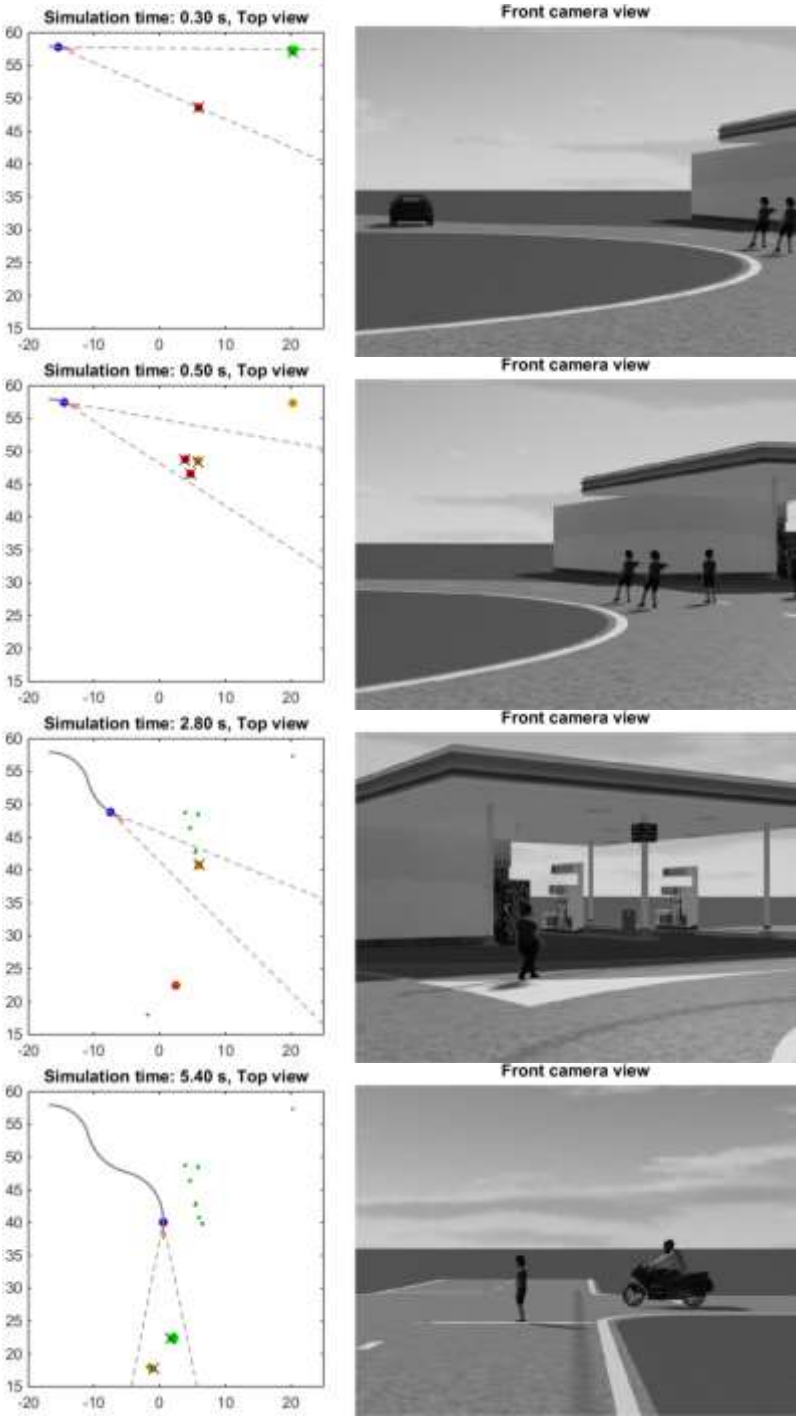

Figure 17. Four screen captures from the urban scenario. On the right side the camera view from inside the car, on the left; the data as tracked by the algorithm.

From the final image in Figure 17 it is clear that all object were at one point tracked by the JPDA filter. This is an indication that the tracker can perform adequately in urban environments.

2.4 The problems with JPDA in practice

In the previous section the basic workings of the JPDA algorithm have been demonstrated. The problem mentioned in many of the literature [7-9] available is that the JPDA filter will become slow when many objects are tracked, or the data is very cluttered. To show this effect, some experiments are conducted. The computer used for these simulations is a PC running Windows 7 64-bit with an Intel Core i5-2400 CPU at 3.10 GHz. There is 8 GB of RAM available and the simulations are run in MATLAB R2014b 64-bit.

2.4.1 Increasing number of objects

For the first experiment an object is generated at a random location, with a random speed and heading. This object will continue this motion until the experiment is over. The experiment time is set to 10 seconds with a sample frequency of 15 Hz, leading to a total of 151 points.

From this ground truth data a measurement is made by introducing a random perturbation on the state of all points. These measurements will go into the JPDA algorithm, which should lead to the tracking of the object. The parameters for the object are summarized in Table 5.

Parameter	Value
Start position X	0 – 30
Start position Y	0 – 30
Start speed	10 – 25
Start heading	0 – 360°
Position noise	0.1
Velocity noise	0.05
Simulation time	10 s
Sample frequency	15 Hz

Table 5. Parameters for the experiment to show processing speed of the JPDA algorithm versus number of objects.

While doing this experiment, the processing time of the JPDA function is recorded and plotted in Figure 18. The initializing of tracks takes longer than the tracking, but the consistency of the processing time is good. Leaving out the initializing, the average cycle time is 4.3×10^{-4} s. This on its own does not mean anything, because it is strongly dependent on the computer used. However, it is a good base to compare the processing time when more objects are tracked.

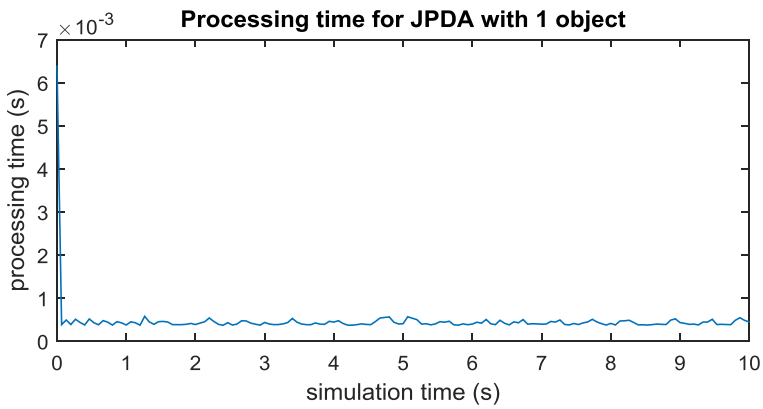

Figure 18. Processing time of JPDA algorithm with one object. The initializing takes more time, but the tracking shows a very consistent processing time.

The experiment was continued by increasing the number of objects by one, until a total of 15 objects were simulated. Note that there is only a little noise on the measurements, but there are no false measurements, which leads to a very consistent cycle time, since every track will be updated with only one measurement. In fact, the number of possible scenarios can be computed directly from the number of objects. Every measurement within the confidence ellipse of an object (which is one) can either belong to the track, or not belong to the track, giving two possibilities per track. Adding everything together, the number of scenarios the tracker considers is 2^n. Also note that it is possible that some of the tracks cross each other, since the paths are generated randomly.

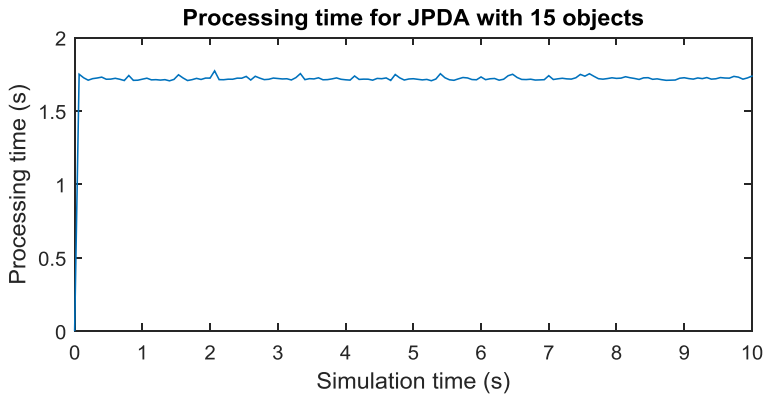

Figure 19. Processing time of JPDA algorithm with 15 objects. After the peak in the beginning, the processing times are very consistent.

In Figure 19 the same information as in Figure 18 is plotted, but now for 15 objects. The initialization still takes roughly the same amount of time, but the tracking is much slower: The average processing time is now 1.71 s. Apart from the track initialization in the first frame, the cycle time is again very consistent, as expected. The initializing of the tracks takes around the same time as with one object, which in this case is much shorter than the average cycle time.

The interesting result of this experiment is the comparison of the processing times for different number of objects. This is shown in the graphs in Figure 20; on the left the linear scaled graph and on the right the same data, but plotted with a logarithmic scale on the vertical axis. In the right side graph there is also a red dashed line, connecting the processing times of the scenarios with one and fifteen objects. The measured cycle times are very close to this line, indicating that the algorithm get exponentially slower when the number of objects increases.

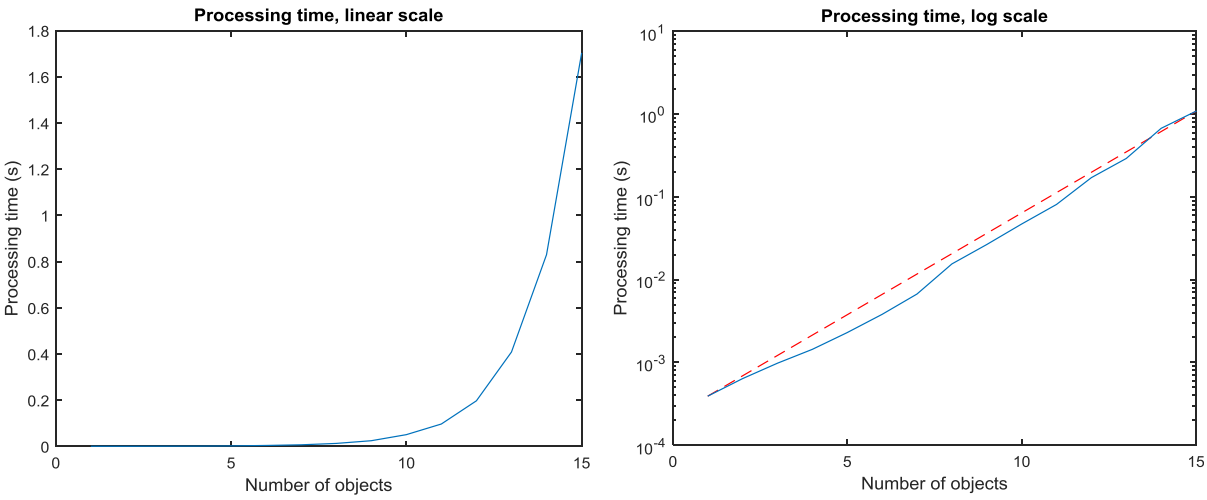

Figure 20. Processing time for different number of objects. Left: linear scales, the processing time increases rapidly with increasing number of objects. Right: logarithmic vertical axis, the red dashed line indicates the exponential increase of processing time for an increasing number of objects.

2.4.2 Objects or hypotheses?

The number of objects is however not the only reason for high processing time of the algorithm, which is indicated by the next experiment. There is only one object in all of these experiments, but instead of one measurement for this object, this number increases. For every frame the measurement is stored multiple times, but all measurements have a different random noise added to it. This can be compared to having multiple sensors in real life, or a sensor having trouble to detect a target as a single object.

The tracker will associate all measurement to the single object, but it will compute the probabilities for every measurement for it to belong to the object. This means the number of calculations will go up, while the number of objects stays the same. Figure 21 shows this for having between one and seventeen measurements for the single object.

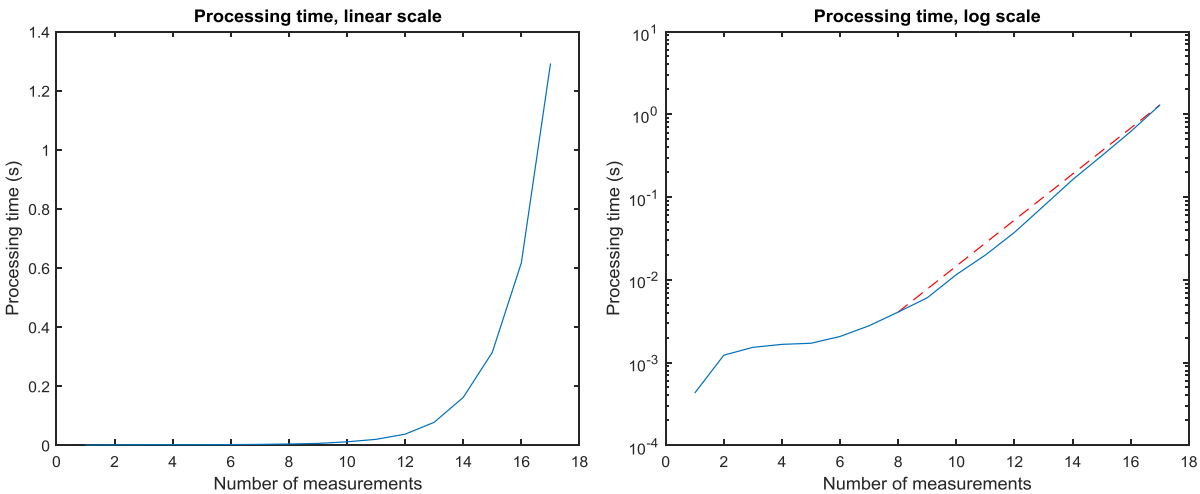

Figure 21.Processing time for different number of measurements of a single object. Left: linear scales, the processing time increases rapidly with increasing number of measurements. Right: logarithmic vertical axis, the red dashed line indicates the exponential increase of processing time for an increasing number of measurements.

Instead of having the number of objects on the horizontal axis, it would be better to have the number of scenarios the algorithm will consider. This is often referred to as the number of hypotheses. When plotting the processing time versus the number of hypotheses, it is useful to do this on a log-log scale, because both the processing time and the number of hypotheses will increase exponentially with the increasing number of objects and measurements.

The results of the Section 2.4.1 should therefore be presented in a different graph, as shown in Figure 22. The data still shows up as a straight line, because in this situation there are no wrong measurements. In this case the number of hypotheses is 2^n, where n is the number of objects. This is true, because every measurement is either correct for the track, or it is designated as a wrong measurement.

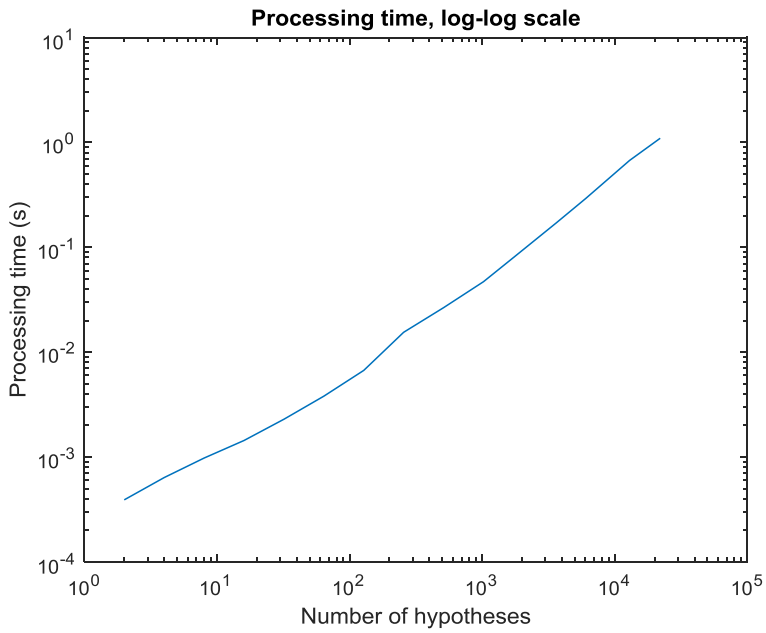

Figure 22. Data from Section 2.4.1 plotted on a log-log scale, with the number of hypotheses on the horizontal axis, instead of the number of objects.

2.4.3 Cluttered environment

In the previous two sections, the problems for the number of objects and the number of measurements were taken on separately. In this section the tracker is exposed to a more life-like situation where several objects need to be tracked while there is a substantial number of false measurements (random noise).

For this purpose, a number of tracks is initialized in the same area as before, but the measurements of those tracks will stop whenever they reach the borders of this area, comparable with an object that moves out of range of the sensor. Five objects will be initialized in the first frame, while another 45 objects are generated at a random time in the first half of the simulation. In every frame there will be ten false measurements, or random noise instances.

Figure 23 shows the processing time for every frame versus the number of scenarios considered by the tracker in that frame. The trend that was visible in the previous examples is also seen here; the processing time increases exponentially with the number of scenarios.

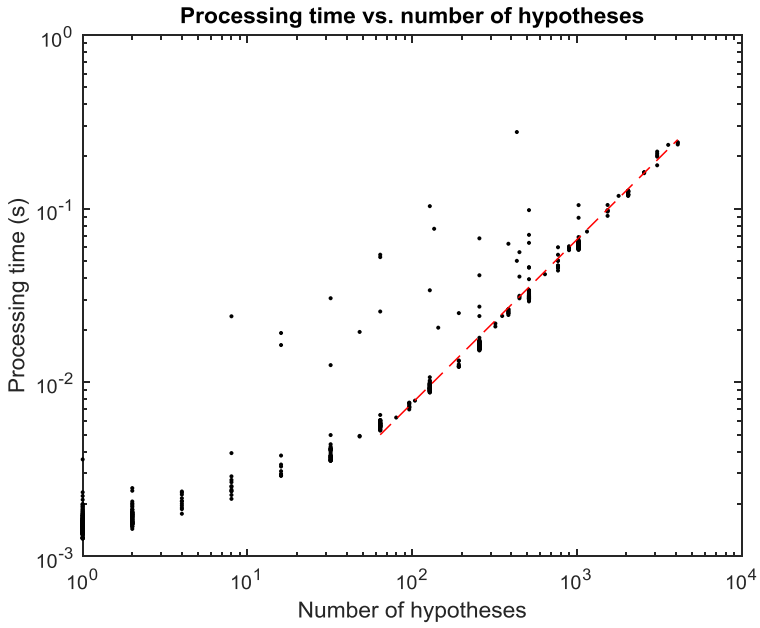

Figure 23. Processing time for every frame in the cluttered environment scenario. Every data-point is represented by a black dot, a straight line (red dashed) is plotted to indicate an exponential behavior for increasing number of hypotheses.

As an indication of the performance of the tracker, the true number of objects is plotted in the same figure as the number of *confirmed* tracks. In this case, a track is designated as confirmed whenever it is tracked for three frames or more. Figure 24 shows the number of objects (ground truth) minus the number of tracks, where the vector of number of tracks is shifted three frames, to counter the effect of track confirmation. In an ideal situation, the number of tracks and objects should be equal and the graph would show a straight line at zero. In this simulation approximately 2% of the frames have a non-zero value in this graph, but all of the frames show a difference of one or less.

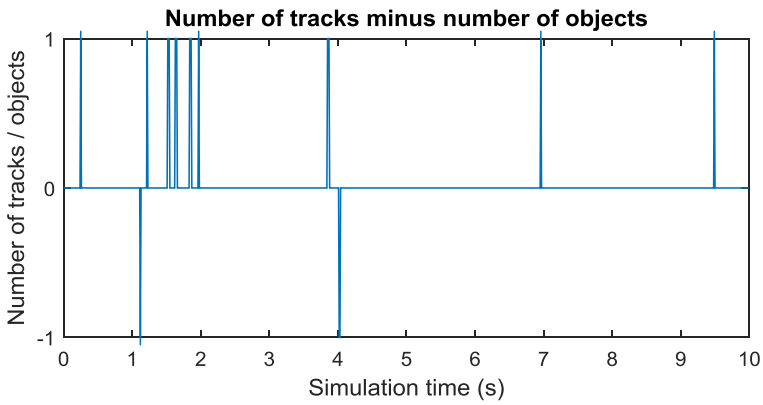

Figure 24. Number of tracks minus number of objects. The vector containing the number of tracks is shifted three frames, to counter the effect of track confirmation after three frames.

To show the situation of the tracker during the simulation, some of the frames have been imaged in Figure 25, namely after 1, 3, 5 and 7 seconds. In these images, the ground truth is represented by black dots, the measurements at this time-step are shown as crosses. Confirmed tracks are plotted as green dots, but stay in the image once they are plotted. This means that a confirmed track will eventually show up as a line of green dots. Unconfirmed tracks are shown as red and orange dots, but will disappear in the next frame.

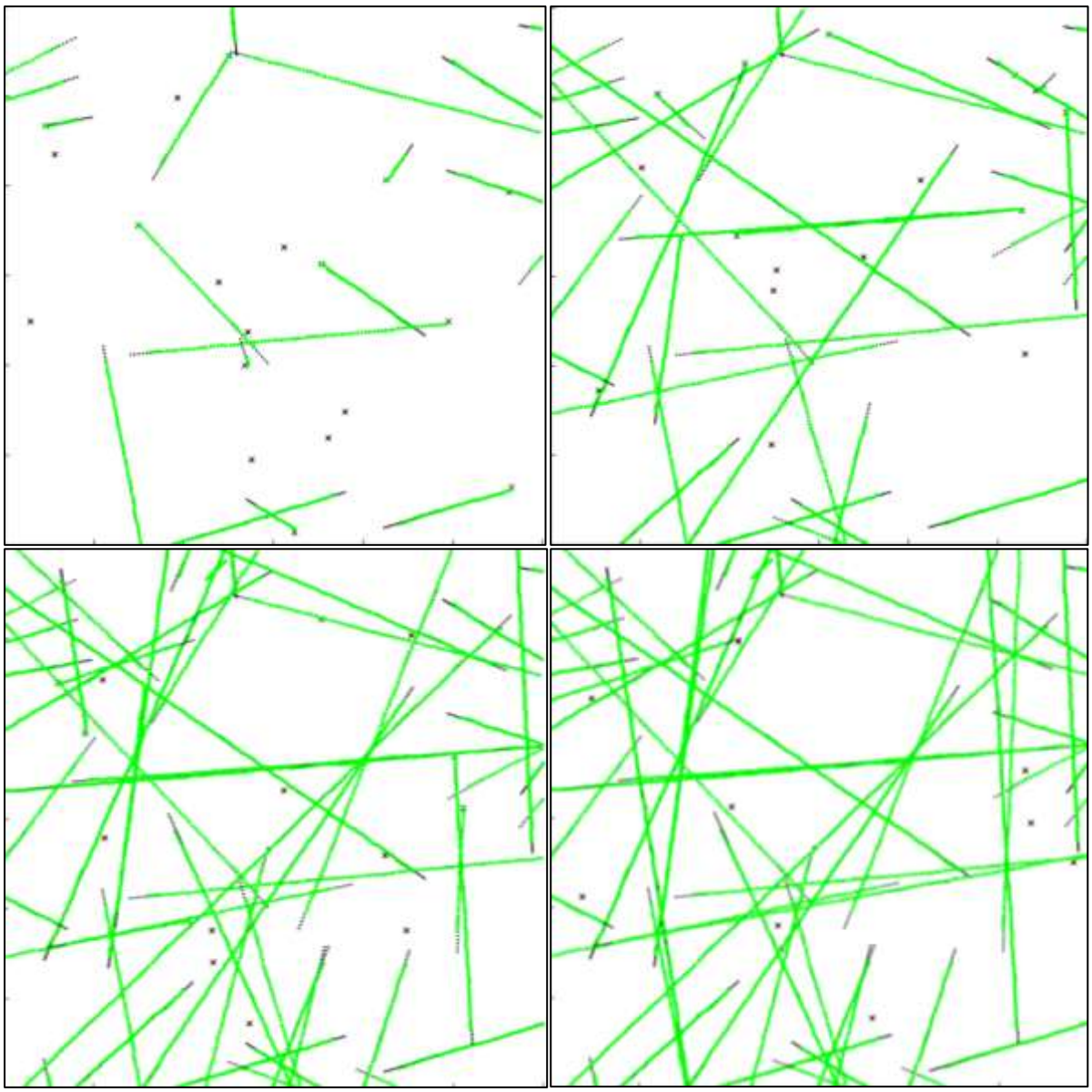

Figure 25. Frames from the cluttered environment experiment, taken after 1 (top-left), 3 (top-right), 5 (bottom-left) and 7 (bottom-right) seconds of simulation. Black dots are ground truth objects, black crosses are measurements in the respective frame, green dots / lines are confirmed tracks.

From this experiment the conclusion can be drawn that on this computer, the algorithm can run in real-time with up to about 10 objects and 10 false measurements per frame. Any more will lead to processing times that are higher than the sample rate of the sensor off 15Hz would allow.

Since the purpose of this tracker is for it to be used in an autonomous vehicle, which is surrounded by (many) more objects, it is clear that some method must be found to decrease the processing time. This will be discussed in the next sections.

2.5 Solving the issue of performance loss with many objects

Although processing power of computers is increasing at a staggering rate, JPDA remains a computationally heavy algorithm, especially when many objects are tracked. This issue can be solved by limiting the number of objects to be associated and tracked. The question is how to do this without deleting data that turns out to be important.

Since computation times increase exponentially with the number of tracks and measurements, the observation can be made that two algorithms running on parts of the surroundings will perform faster than one algorithm observing the entire area. The idea can be extended to more than two areas, provided that every area has its own JPDA algorithm.

By structuring the areas in a smart way, an upper limit to the number of objects within every region can be defined, ensuring that the tracker will remain fast enough for its purpose. For an autonomous vehicle, the lower limit of processing speed should be 10 Hz; everything slower will mean that there is insufficient time to respond to traffic situations and act accordingly. This limit in turn can be translated to an upper limit for the number of objects in a region. The experiment from Section 2.4.3 shows that an algorithm can associate and track up to 10-15 objects while still being sufficiently fast.

A first attempt at splitting the data is trying to find clusters of objects within the sensor area and run a JPDA on each cluster. A very elegant method of clustering was introduced by Dezert and Bar-Shalom [38]. The clustering part is called the *clustering decomposition algorithm* (CDA). The JPDA+CDA method will be used to compare against a novel method later on. The existing clustering method will be explained and evaluated in the next section.

The novel approach will feature separate regions that divide the area around the vehicle into smaller parts. The idea of multiple regions with their own trackers could solve the processing speed issues, but the question remains of how to construct these regions in such a way that all the important parts of the surroundings are covered. The algorithm will run in a road vehicle and many systems exist that can interpret the surroundings, giving useful information and giving pointers to construct the regions. In Chapter 3 several methods will be described and a trade-off is made on which approaches to elaborate on and use for further testing.

2.5.1 Clustering decomposition algorithm (CDA)

This section will focus on an existing clustering algorithm, to compare against a novel method which will be introduced in Chapter 3. The CDA method and the example given in this section are based on a paper by Dezert and Bar-Shalom [38].

To start off, a scenario is devised as can be seen in Figure 26, where seven tracks are present (red circles, with orange confidence ellipses), along with ten measurements (blue crosses). For a human it is quite easy to see that there are three groups that can be clustered together (green dash-dot rectangles). It is much harder to let a computer know this fact. Note that track number 3 (t_3) is not assigned a cluster, because there are no measurements within its confidence ellipse.

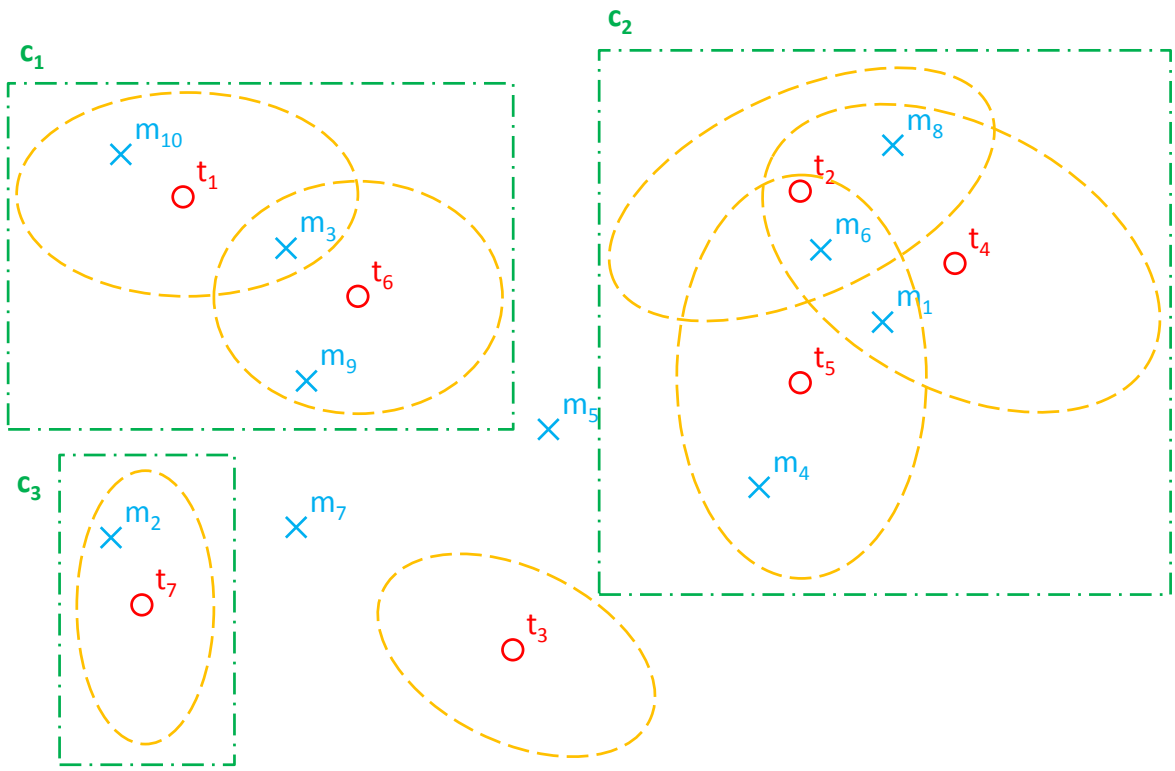

Figure 26. Example of a clustered environment. Red circles are the estimated positions of the tracks, blue crosses are the measurements. The orange dashed ellipses represent the confidence ellipses and the green dash-dotted rectangles indicate the clusters.

The clustering method works by parsing the validation matrix, as constructed according to the JPDA method, several times to create the clusters. The first step is to delete any rows or columns in the validation matrix that do not contain any 1. The null-columns represent the tracks that have no measurement within their gate (track 3 in Figure 26). The null-rows are the measurements that do not belong to any track, such as measurements 5 and 7 in Figure 26. During this process, the first column of the validation matrix is disregarded, since it only contains ones and represents the possibility of a measurement belonging to no track. Figure 27 shows the validation matrix before and after this first step.

$$
\begin{array}{c}
\begin{array}{cccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\
\begin{bmatrix}
1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \end{array}
\end{array}
\Longrightarrow
\begin{array}{c}
\begin{array}{cccccc} 1 & 2 & 4 & 5 & 6 & 7 \end{array} \\
\begin{bmatrix}
0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 6 \\ 8 \\ 9 \\ 10 \end{array}
\end{array}
$$

Figure 27. First step of the clustering algorithm: remove the first column of the validation matrix and then remove all rows and columns containing only zeros, indicated by the red dashed rectangles in the left matrix.

After this initial step the algorithm looks at the first column and all rows with a 1 in this column are merged together. The original rows are remove and replaced by the merged row. In the implementation for this project, this merged row is added to the bottom of the matrix. This process is visualized in Figure 28.

$$
\begin{array}{c}
\begin{matrix} 1 & 2 & 4 & 5 & 6 & 7 \end{matrix} \\
\begin{bmatrix}
0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
\end{array}
\Longrightarrow
\begin{array}{c}
\begin{matrix} 1 & 2 & 4 & 5 & 6 & 7 \end{matrix} \\
\begin{bmatrix}
0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 1 & 0
\end{bmatrix}
\end{array}
$$

Figure 28. Merging of the rows with a 1 in the first column. The rows to merge (left) and merged row (right) are indicated with a red dashed rectangle around them.

This process is then repeated for the second column, then the third column, until the matrix has been parsed for all columns. The intermediate matrices in this process are shown in Figure 29. When the process is finished, the remaining matrix represents the clusters that are visible in Figure 26. Indeed, comparing the matrix with the situation in Figure 26 it can be seen that track 2, 4 and 5 form a cluster, as do track 1 and 6, while the final cluster exists only of track 7.

$$
\begin{array}{c}
\begin{matrix} 1 & 2 & 4 & 5 & 6 & 7 \end{matrix} \\
\begin{bmatrix}
0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 1 & 1 & 0 & 0
\end{bmatrix}
\end{array}
\longrightarrow
\begin{array}{c}
\begin{matrix} 1 & 2 & 4 & 5 & 6 & 7 \end{matrix} \\
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 1 & 1 & 0 & 0
\end{bmatrix}
\end{array}
\longrightarrow
\begin{array}{c}
\begin{matrix} 1 & 2 & 4 & 5 & 6 & 7 \end{matrix} \\
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 1 & 1 & 0 & 0
\end{bmatrix}
\end{array}
$$

$$
\begin{array}{c}
\begin{matrix} 1 & 2 & 4 & 5 & 6 & 7 \end{matrix} \\
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & 1 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0
\end{bmatrix}
\end{array}
\longrightarrow
\begin{array}{c}
\begin{matrix} 1 & 2 & 4 & 5 & 6 & 7 \end{matrix} \\
\boxed{\begin{bmatrix}
0 & 1 & 1 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}}
\end{array}
$$

Figure 29. The next steps of the clustering algorithm, leading to the matrix with clusters, marked with a green dashed box.

To test the performance against the JPDA algorithm without clustering, the scenario described above is implemented and tested. To make the processing time more reliable, the scenario is run 1000 times with the same data and the average of the processing time is calculated.

Note that the results of this experiment should be interpreted appropriately: this example is a very good case for clustering, real-life situations may see a different performance gain than this

experiment shows. The clusters within the JPDA+CDA algorithm are evaluated one at a time. This can potentially be sped up by processing each cluster in parallel.

The processing time for the JPDA algorithm without clustering is 0.037 seconds as recorded by MATLAB, whereas the processing time for the clustering algorithm is 0.0051 seconds, leading to a 7.3 times faster processing speed. Looking at the number of scenarios the algorithms have to process, this difference is not very surprising. The normal JPDA has to process 512 scenarios, where the JPDA+CDA has to cope with 32, 8 and 2 scenarios for the clusters.

	Processing time (s)	**Improvement**	**Scenarios**
Normal JPDA	0.037	-	512
JPDA + CDA	0.0051	7.3x	32 + 8 + 2

Table 6. Improvement in cycle time for the clustering algorithm. NOTE: these results are for this scenario only and may not give a good representation of real life performance gain.

The experiment in this section indicates that clustering increase the speed of the tracker by as much as 7.3 times, however this is only the case in this specific example. Using real data could lead to different results. This will be evaluated at a later stage, when the comparison is made between the normal JPDA, JPDA with CDA and JPDA with the novel regions method. In the remainder of this work the JPDA+CDA algorithm will be named *clustering JPDA*.

2.6 Performance indicator

When a new technique is introduced, it must be proven to work and it should be shown that the new method is better than the one previously used. Ever since the introduction of tracking algorithms this has been a difficult issue. How *do* you measure the performance of an object tracker? Surely processing speed is one measure, but accuracy and repeatability are also important.

In [37;39] the *receiver operating characteristic* (ROC) used in signal processing is modified into a *tracker operating characteristic* (TOC) to estimate the performance of a tracking algorithm. More specifically; A modified Riccati equation is derived to incorporate the false alarm rate and detection probabilities into the tracking performance. The equation is then used to compare against curves of the TOC to determine the true performance of the system.

This method can give an indication of how well one tracking algorithm performs compared to another, but as described in the previous paragraph, the method gives an approximation of the performance, not an exact or optimal metric. Furthermore, the computation of the TOC-curves is quite extensive, which makes this method impossible to implement on a live tracker such as an autonomous vehicle at this moment.

The method works by iterating the modified Riccati equation of the covariance matrix until it converges for several detection probabilities (P_D) and false alarm probabilities (P_F). This is then used to create the contour plots of the TOC. By graphically comparing the TOC curves with the ROC contour of a certain signal-to-noise ratio of the sensor system, the required P_D and P_F can be determined for a (near)-optimal functioning tracker.

Since the TOC does not give an exact number and the processing time required is substantial, it is decided to use a less general, more simplistic indicator of the tracker performance. The focus of this thesis is to find a way to increase the speed of a tracking and association algorithm in an environment with many objects. All the algorithms will be presented with the same data and the main metric for comparison will be the average and maximum processing time. The average processing time can give an indication of the time gain for a certain method over another one, whereas the maximum processing time can indicate whether a method is likely to be able to perform in *real-time* at a certain frequency.

In this case *real-time* is defined as giving a guarantee that the algorithm has a processing time of no more than a limited time. For instance a real-time algorithm at 10 Hz will mean that the processing time of the algorithm is never longer than 0.1 s, even in the worst case scenario. This definition is very important with an eye on the safety of a tracking system in a traffic environment.

Measuring only the processing time is not sufficient, because an algorithm can become faster when it drops tracks. In order to verify the accuracy of the tracker, the error between the ground truth object location and the track will be recorded and given as an indicator as well. This is obviously only possible in simulated scenarios, where the actual ground truth is known. With real sensor data it is very difficult to know the exact location of the objects with respect to the sensor, especially in automotive applications, where the range can be as much as 200 m.

Besides the error between the ground truth and tracked objects, the number of missed objects and false positive tracks will be counted. A missed object is a situation where there is an actual object in

the ground truth data, but the tracker fails to detect it. This is a dangerous scenario when translated to a traffic situation, because the algorithm (car) will not know there is an object and thus will not try to avoid it if necessary.

On the opposite side of the scale is the false positive track. In this case the tracker will indicate the presence of an object, where there is no object in the ground truth data. Although less of a safety issue in traffic situations, this may lead to unwanted actions of the vehicle. It may for instance brake to avoid a collision, while no object is there. This number is likely not zero for any of the algorithms, since any measurement, including noise, that is not matched to a track will be initiated as its own track for the next frame.

To prevent the vehicle from making rash decisions based on measurement noise, it may be useful to have two lists of tracks: confirmed and unconfirmed tracks. With the counter introduced in Section 2.2.3, this could be as simple as indicating a track as confirmed whenever its counter is larger than a certain number.

To summarize; the performance of the algorithms in this thesis will be compared using:

- The average cycle time
- The maximum cycle time
- The RMS error between the ground truth and tracks

The top two measures will indicate the speed of the algorithms and performance can be said to be better when the times decrease. The last indicator is to verify the accuracy against the JPDA algorithm without clustering. If these values are similar for the new methods, it can be said that the performance is comparable. The number of tracks is also counted and compared, to ensure tracker consistency.

3 Methods to construct regions

Since the JPDA tracking and association algorithm gets exponentially slower with an increasing number of objects ([7-9] and Section 2.4), splitting the surroundings into several areas will lead to a faster overall process. It is assumed that a tracker will be able to process up to fifteen objects simultaneously at a frequency of 10 Hz. Several methods already exist to generate a region of interest (ROI) around an autonomous vehicle. In this chapter some of these methods, as well as a completely new approach are discussed.

The methods will be compared based on several properties. The number of regions is important for two reasons: less regions means less tracking algorithms running in parallel, but also less boundaries between regions. The boundaries are important, because it is very possible and likely that some objects will move from one region to the next. This can create issues with tracks and measurements, so a certain amount of overlap is needed between the regions. How much overlap is required is discussed in Chapter 4, in this chapter the assumption is made that less regions is beneficial, because there will be less boundaries and thus less areas with overlap.

A more complex description of the region boundaries will mean more computation is required to construct them. Less complexity is therefore beneficial, however the adaptability of the regions will reduce whenever the complexity decreases. These two properties should be balanced carefully. The complexity of the regions can also depend on the type of data that is used to construct the boundaries. Some data is readily available from the sensors, other data needs to be processed first. This is another property to take into consideration.

Continuity of the regions' shape is also important, because quickly changing shapes can cause issues with the tracks being initialized in the wrong areas. A non-dynamical shape is very continuous, but a shape that smoothly follows a contour can also be considered continuous. Lastly, an assumption is made on the efficiency of the trackers for the regions. Whenever there are very few objects in a region, the association task will run fast, but the overhead (of computations) is larger compared to more densely populated regions.

3.1 Static grid - Simple rectangles

The first method is the simplest method to divide an area into several pieces; cut it up into smaller sized regions. Such a philosophy for the ROI works fine for short ranges or on a highway. The only questions remaining for this method is how large should the regions be and how many of these regions are appropriate.

The desired layout of the regions around an autonomous vehicle is in the form of shells in which the first shell encloses the whole vehicle and the direct surroundings in *all* directions. A suitable way to do this would be to create a rectangle and place the host vehicle somewhere inside this rectangle. For the width of the rectangle the average size of three lanes is used. This way, there is always one lane inside the area on either side of the host vehicle. The standard width of a lane on the Dutch highway is 3.5 meters, so the width of the first rectangle will be 10.5 meter; 5.25 m to either side of the host vehicle.

To determine the length of the rectangle the requirement of a maximum of 15 objects inside the area is used in combination with the traffic density. If this density is known, the maximum length of the rectangle can be determined. Traffic statistics [40] show that a maximum of 6000 vehicles per hour per lane is reached, before the traffic gets too busy to remain at the desired speed. Any more vehicles and a traffic jam will occur. Table 7 shows some more numbers on the traffic density in certain situations.

	Speed (km/h)	**Lane capacity (vehicles/h)**
Normal	100	2000-2500
Maximum	100	6000
Traffic jam	0-50	-

Table 7. Traffic statistics [40] for a highway.

The rectangles will be designed for the worst case scenario, so that the algorithms remain reliable in all traffic situations. Considering the situation with three lanes all filled up to the maximum capacity before the traffic stalls, the length of the ROI is approximately 83 m.

$$
\begin{aligned}
&l = \frac{n_v}{n_l \cdot \frac{q_l}{v}} = \frac{15}{3 \cdot \frac{6000}{100}} \cong 0.083\ km = \ 83\ m \\
&where: \\
&n_v = maximum\ number\ of\ vehicles \\
&n_l = number\ of\ lanes \\
&q_l = capacity\ per\ lane\ (vehicles/h) \\
&v = velocity\ in\ km/h
\end{aligned}
\tag{9}
$$

The location of the host vehicle inside the rectangle can now be determined. The front view is more important than the rear view, because in most use-cases the host vehicle will move forward, with the rest of the traffic going in the same, or similar direction. It is more important to see an object suddenly stopping in front of you, than it is to see a similar situation behind you. For this reason the host vehicle will be placed at about 1/3 along the length of the rectangle, as shown in Figure 30.

Figure 30. The location of the first rectangular region around an autonomous vehicle. Not to scale.

Now that the first region is set, the rest of the rectangles can be laid around it in a grid. Assuming a sensor range of 200 m forward and backward and 25 m to either side, the complete picture of grids will look like Figure 31. To create a grid that covers the previously described sensor ranges, a total of 25 regions must be defined. However, since occlusion will inevitably occur at larger distances, some of the regions may be combined into single, larger regions. For example; three regions to the far left side of the vehicle may be combined into one region (marked with X).

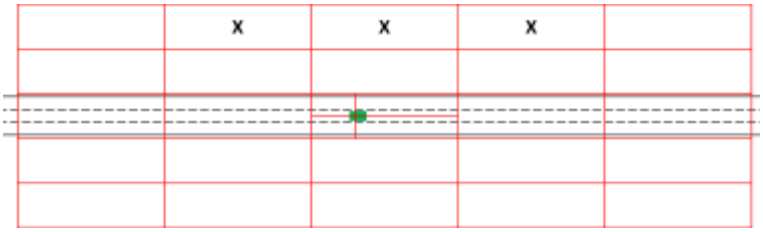

Figure 31. All regions for the entire sensor range. Note that some regions may be combined to limit the number of regions/trackers. Not to scale.

The downside of this method is not so much the number of regions, nor the question of how to combine regions in the most efficient fashion. Rather the fact that this grid will always remain stationary in the vehicle frame and does not take into account any movement of the vehicle or the surrounding environment. Imagine a situation where the road bends, or at a T-junction, some of the regions will only cover the roadside which may be completely unimportant to the traffic situation. Whereas the range to the sides may not be enough to cover the parts of the road that *are* important.

In Figure 32 a situation at a T-junction is visualized; many of the regions do not cover any part of the infrastructure at all, But more importantly, the roads from the left and right are only viewed for the first 25 m. This may not be enough when a car is approaching from either side while the host vehicle intends to cross the path of that oncoming car.

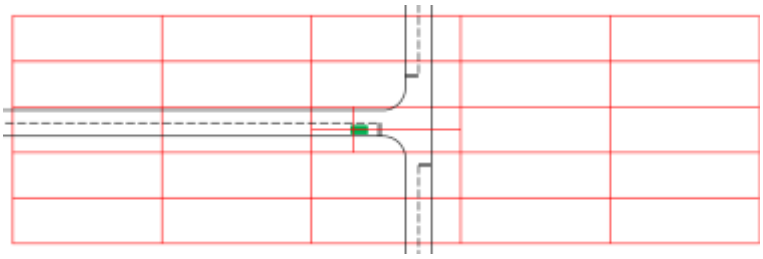

Figure 32. A fixed grid region approach at a T-junction. The front regions cover unimportant roadsides, while the limited range of 25 m to the sides is not enough to ensure safe entry onto the road. Not to scale.

A similar situation occurs in a continuous radius corner, as displayed in Figure 33. The road is again not in the center rectangles and the end of the corner is not covered by any of the regions.

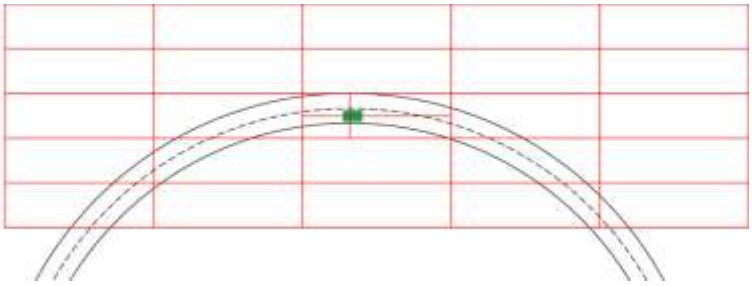

Figure 33. A fixed grid region approach in a constant radius corner. The important infrastructure is condensed into only a couple of regions, whereas many regions cover unimportant roadside. not to scale.

Many more traffic situations can be thought of where this method does not suffice. The main problem is the lack of flexibility and adaptability of the regions to the current situation.

Benefits

- Very simple method of splitting

Drawbacks

- Not dynamic
- Limited field of view
- Many trackers, some not being used efficiently

3.2 Shells - Ellipses

A static approach to dividing the sensor area into smaller regions seems to be limiting. Researchers have tried to find elegant solutions to make the regions change shape dynamically. One idea is to use vehicle parameters to change the size and shape. The simplest option is to define shells around the vehicle using the velocity to scale these regions. As discussed before, the lateral size of the region is of less importance than the longitudinal size.

This method works by dividing the surroundings into several elliptical shells around the vehicle, whose size is dependent on the vehicle velocity. A minimum size is defined to cope with situations where the host vehicle is driving slowly or at a complete stop. The longitudinal size is determined based on the traffic density information in Table 7. At 100 km/h the length of a region should be about 80 m. This space is traveled in around three seconds, which means that setting the length of the first ellipse to three times the vehicle speed (in m/s) is a proper choice.

$$l_{ellipse} = 3 \cdot v_{host} \tag{10}$$

The width of the ellipse can be less than the length, because traffic is assumed to go in approximately the same direction as host vehicle. To make sure that at lower speeds the region still covers approximately one lane on either side of the vehicle, the width is set to one quarter of the length of the region.

$$w_{ellipse} = \frac{3}{4} \cdot v_{host} \tag{11}$$

Subsequent regions will have a length and width that is a multiple of these values, until the final region covers all extremities of the sensor range (Figure 34). The car will be centered in the ellipses. It is important to note that the ellipses will become impractically small whenever the vehicle speed is very low. A lower size limit should be used to cope with slow movement and standstill. This limit can for instance be set at the minimum expected velocity in normal traffic. Assuming that this speed is about 30 km/h in an urban environment, the minimum length and width will be 25 and 6.25 m.

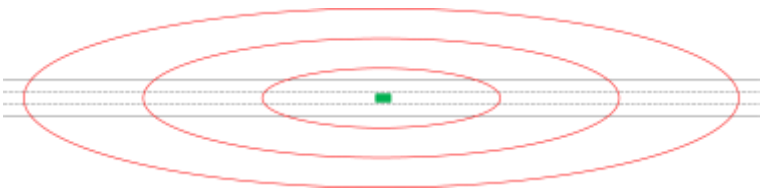

Figure 34. The basic lay-out of the 'shells' approach. consecutively larger ellipses envelop the host vehicle, the size being variable with the host vehicle speed. Not to scale.

Shell-like approaches have been proposed before in literature [41] and have a great benefit to a static grid. The number of required regions is much less, leading to less required trackers and thus a faster algorithm. However, similar problems arise when more realistic roads are put to the test. Figure 35 and Figure 36 show the two standard situation also used in the static grid discussion; a constant radius corner and a T-junction. In both situations similar problems occur as with the static grid approach: Coverage off unimportant areas and no coverage in important areas.

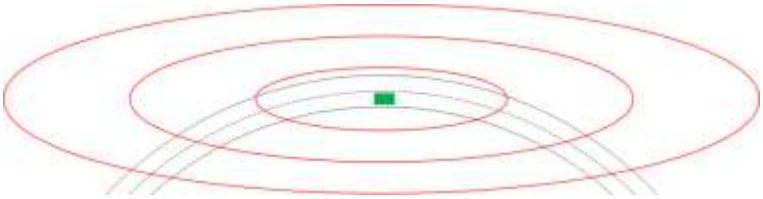

Figure 35. The shells approach in a constant radius corner. Since the coverage is similar to the static grid approach, similar problems arise. Not to scale.

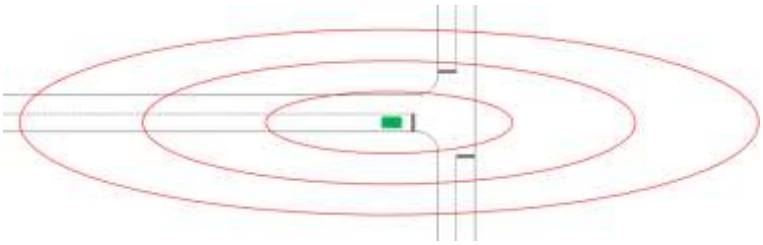

Figure 36. The shells approach at a T-junction. The view into the roads from either side is limited. Not to scale.

Benefits

- Less regions required
- Size scales with vehicle speed

Drawbacks

- Similar problems occur in corners / at junctions

3.3 VYR – Velocity-Yaw-Rate

An even more dynamical way of defining regions around an autonomous vehicle is using more vehicle parameters. Velocity can be used to define the length of the regions, while the yaw-rate can be used to estimate the future path of the host vehicle. The choice for these two parameters is easy, since both are already available within other systems of the car: Velocity is needed for cruise control, while the yaw-rate is required for systems like ESP.

The length of the regions can be determined in a similar way to the previous method, as described in Section 3.1 and 3.2. The data collected in Table 7 is valid at 100 km/h, but the length should scale with the velocity. At 100 km/h, the approximately 80 m will be traversed in about 2.9 seconds. Therefore it seems appropriate to make the length of the region around three times the current velocity (in m/s):

$$l_{VYR} = 3 \cdot v_{host} \tag{12}$$

Making the length of the region dependent on the velocity has limitations. At slow speeds or standstill, the region will become so short that it is impossible to contain any objects in it. A lower size limit should be set. During driving in normal traffic, the speed is expected to remain above 30 km/h. The length of the region belonging to this speed will be used as a lower size limit.

$$l_{min} = 3 \cdot \left(\frac{30}{3.6}\right) = 25\ m \tag{13}$$

The sides of the region are determined by the yaw-rate of the vehicle, as described in [42] and Equation (14). By dividing the vehicle speed by the yaw-rate, the instantaneous center of rotation (ICR) can be determined. This is the radius of the path of the vehicle, so to get the left and right borders of the desired region, half of the desired width of the region should be added or subtracted. The same reasoning as before is used to determine the width of the region; one lane on either side, which leads to 10.5 m. Note that this method assumes no slip and does not predict the vehicle path, but rather estimates it using the current vehicle parameters.

$$\begin{aligned}
&y_{ICR} = \frac{v}{\dot{\psi}} \\
&\mathrm{R_{ICR}} = \frac{\mathrm{v}}{|\dot{\psi}|} \\
&R_{left} = R_{ICR} - sign(\dot{\psi}) \cdot \frac{w_{des}}{2} \\
&R_{right} = R_{ICR} + sign(\dot{\psi}) \cdot \frac{w_{des}}{2} \\
&with\ w_{des} = desired\ width\ of\ the\ region
\end{aligned} \tag{14}$$

When the vehicle is driving on a straight road, R_{ICR} is equal to infinity, which may lead to problems in the programming of the vehicle systems. It is advisable to set an upper limit to the radius, to make sure this does not cause any system malfunctions. Note also that a negative yaw-rate will give a negative radius for both side boundaries, which indicates that the center of rotation is to the right of the vehicle, i.e. has a negative y-value in the vehicle frame.

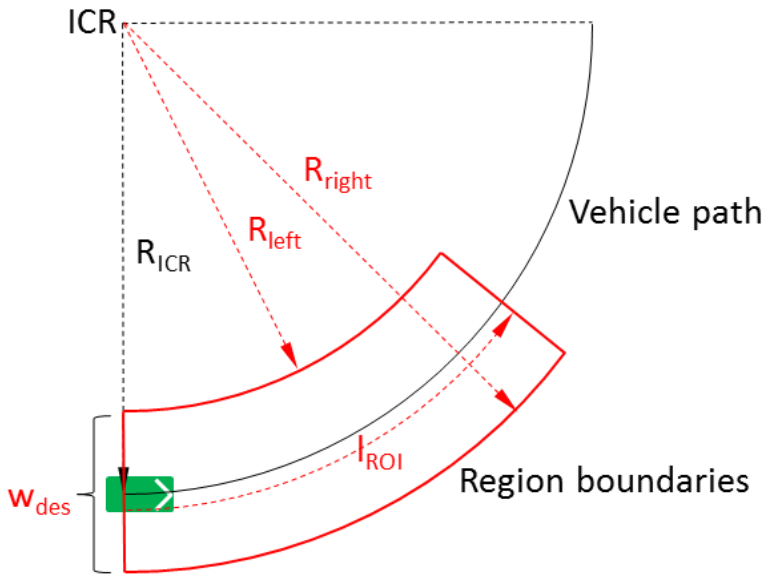

Figure 37. The side boundaries of the VYR approach are determined based on the vehicle yaw rate and the Instantaneous Center of Rotation (ICR).

Figure 38 shows the situation in a constant radius curve, the regions follow the vehicle path perfectly. In this situation it is possible to have less regions, especially to the sides, because the regions will shape to cover the important infrastructure.

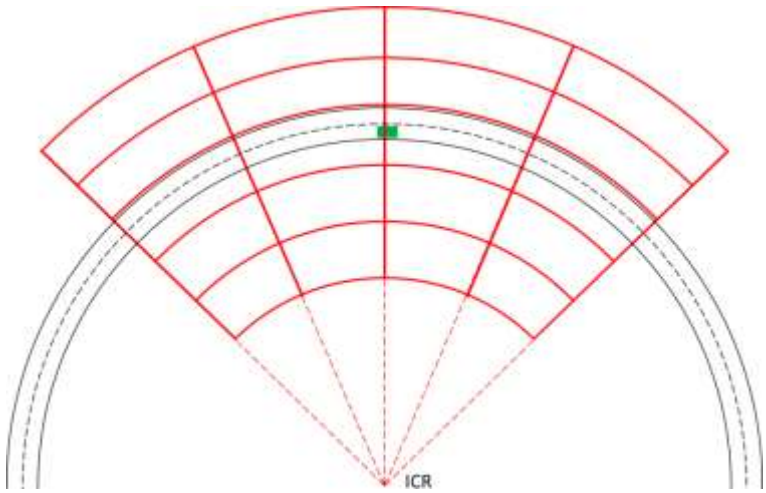

Figure 38. VYR regions in a constant radius curve. The center regions cover the road. Not to scale.

Of course the above example is not a good representation of the real world, and plenty of scenarios can be thought of where the VYR approach does not work as desired. In an S-bend, for example, the regions will cover the expected path of the vehicle as it was in the first part of the bend, But this will be completely the wrong way for the second part of the bend (Figure 39).

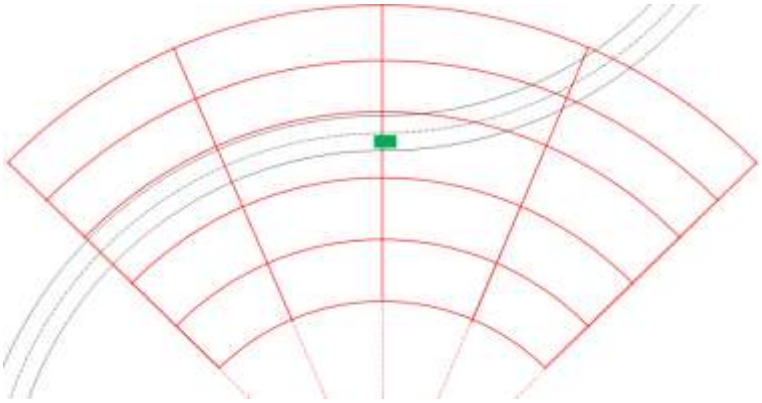

Figure 39. The VYR approach in an S-bend, the regions now do not cover the road as desired. Not to scale.

While driving up to a T-junction or intersection the VYR approach will create regions with a similar shape as the static grid approach, because the yaw-rate is close to zero. The same problems will arise in this situation; many regions cover unimportant roadside and important parts of the road are missed because of the limited view to the sides when the yaw-rate is close to zero. This situation can be seen in Figure 40.

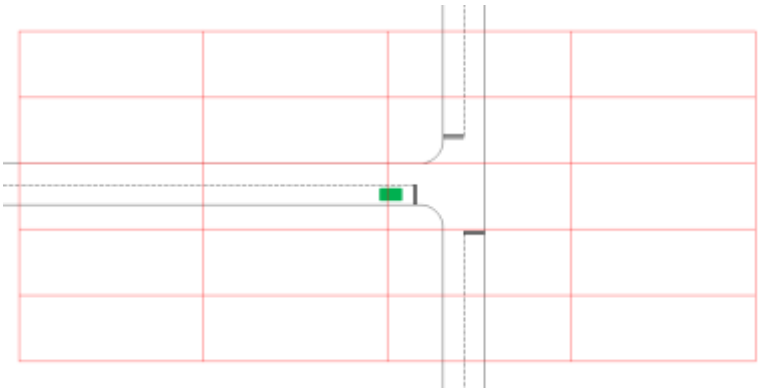

Figure 40. VYR approach while driving up to a T-junction. Similar problems occur as with the static grid approach. Not to scale.

As this method estimates the vehicle path using the yaw-rate of the vehicle, the number of regions can be decreased. The regions will cover more of the likely path of the vehicle than with a static grid approach, certainly at a short range. Regions to the side may be combined into fewer, but larger regions, because the number of expected objects is less. However, the number of regions may not be the main problem of a tracking algorithm, the coverage is much more important. If a certain area is not covered by any region, the autonomous vehicle cannot make a decision with full confidence of maximum safety.

Benefits

- Dynamic shape works well on short range
- Less regions required

Drawbacks

- Does not work in all real-life situations
- Region follows *instantaneous* estimation (cannot 'look' ahead)

3.4 Path-planner

So far methods using vehicle information have not come up with satisfactory results, but since this project is not about a normal car, but rather an autonomous vehicle, another very powerful piece of information is available: the future path of the vehicle. This path is known, because the path-planner in the vehicle will determine which control actions lead to the optimum situation in the near future, taking into account other traffic, safety, comfort and potentially many more parameters.

Figure 41. Path-planner approach. The region will shape with the future path of the host vehicle, while the length is dependent on the vehicle speed. In this image a lane change to the left is planned (green dashed arrow). Not to scale.

Figure 41 shows what the regions might look like when a lane change is made on a straight highway. Note that the width of the regions is again chosen as three lanes (one lane on either side of the host vehicle) and that the length of the regions can again be dependent on the host vehicle predicted speed. To shape the regions behind the vehicle, historical data can be used.

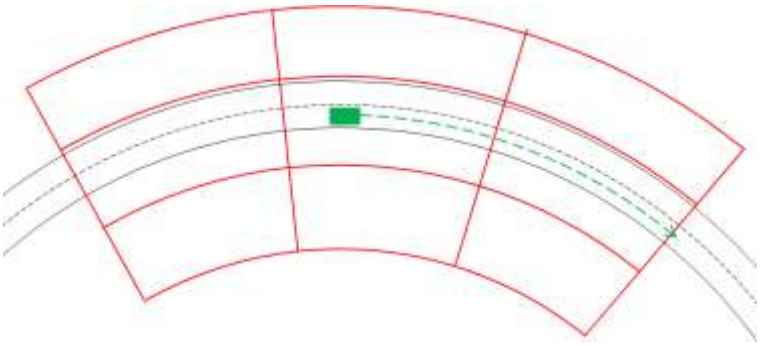

Figure 42. The path planning approach in a constant radius corner. The performance is similar to the VYR approach. Not to scale.

When the vehicle is in a constant radius corner (Figure 42), the situation looks very similar to that of the VYR approach. The future path of the vehicle follows the road, as does the previously traveled path. Contrary to the VYR method, the path-planning approach also works in corners with a non-constant radius or even S-bends (Figure 43).

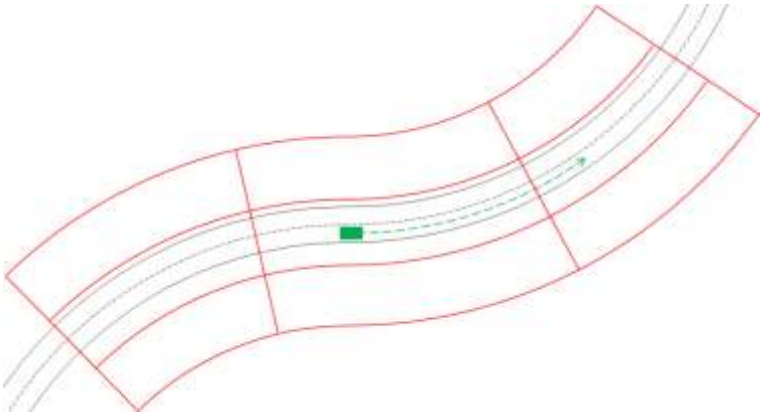

Figure 43. In an S-bend, the path planner approach performs better than the VYR method, because the future path is known. Not to scale.

At an intersection some problems are solved as well: The road is well observed in the direction of future travel, but in other directions there is still a lack of coverage as can be seen in Figure 44. The traffic coming from the left cannot be observed at a large enough range to make a safe decision for executing the turn.

Note that Figure 44 is not to scale and the regions actually cover only about 15-20 m of the road to the left of the host vehicle. If this were a junction at a rural road, the speed limit may be 80 km/h (~22 m/s), leading to little time between detection and intersecting paths of the oncoming vehicle and the host vehicle. Furthermore, a gap of about 7 seconds is perceived as acceptable for such a situation [5]. A gap of less than a second is therefore not sufficient by any means.

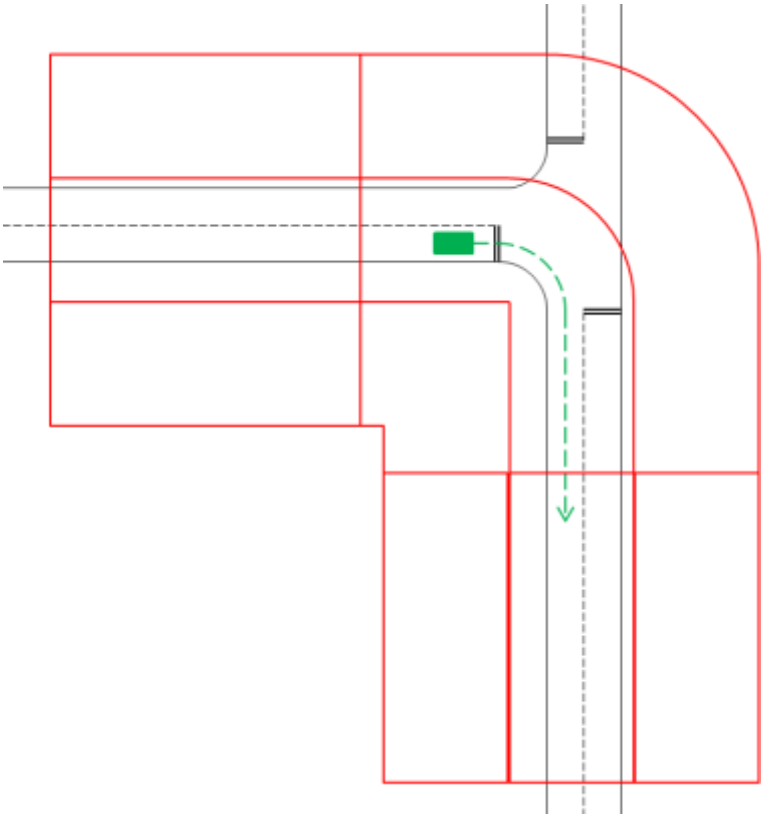

Figure 44. The path planning approach at a T-junction. The future path and the oncoming lane in that direction are covered, but the lane from the left is not. Not to scale.

A completely new issue that needs to be addressed with this method is that of a change in decision of the future path. For example when the current driving lane on a highway is blocked for some reason (obstacle or slower traffic), the autonomous vehicle will have several options and may change its plans from one time-step to the next when new information is available. In Figure 45 the vehicle may first decide to take the first path (red, dashed), but may reconsider when there is another car approaching from behind in the left lane. Option 2 (blue dash-dotted, lane change right) or 3 (black dotted, slow down/stop) may turn out to be more desirable, leading to a change in the planned path.

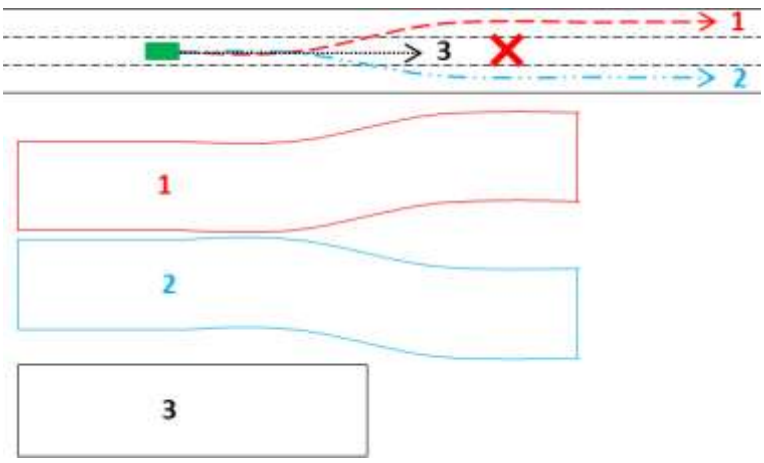

Figure 45. A path planner always has multiple options, how should the tracking be done in these situations? (option 1: lane change left, 2: lane change right, 3: reduce speed / stop). Not to scale.

If only the true future path is used to determine the tracking regions around the car, these regions will suddenly change in shape (bottom part of Figure 45). This in turn may lead to mismatches in the association process and cause the tracker to perform less than optimal or even miss objects completely.

One solution to this issue would be to track not only the current planned path, but also create regions using less likely, but still plausible paths that are available in the path-planner. Since all regions have their own association/tracker, mismatching is less likely to occur. It will however mean that some areas are tracked more than once, because there is a lot of overlap between the regions created using different paths. It will thus require more computation power to do the job, while in fact some of the work is carried out multiple times.

Benefits

- Very likely to contain important objects
- Few regions required

Drawbacks

- Requires previous position data (of host vehicle)
- Some scenarios will remain a problem (junctions)
- Dilemma when the path changes

3.5 eHorizon – Extended maps

A final method to determine the tracking regions discussed in this report is a novel method that uses information from detailed maps to create regions that shape with the road contours. This extended map information is available in the autonomous vehicle through the eHorizon [11;43] system, based on the ADASIS [44] protocol. Details such as road curvature, number of lanes, speed limits, slope of the road etc. are all available when the current GPS position is entered into the system.

Using prior knowledge of the road ahead is not a new concept; several researches have been carried out, mostly about using the upcoming road slope to adapt the behavior of a truck cruise control system [45]. For example: shifting to a lower gear before driving up a slope, or reducing throttle input when a downward slope is upcoming. However, this information has not previously been used to divide the surroundings of an autonomous vehicle into smaller regions to aid in tracking.

The information can be used to shape regions to the road edges of the future path, including the infrastructure that may cross that path in the near future. If, like before, the (maximum) width of three lanes is retained and a length that scales with velocity, nothing much changes compared to the path-planning approach in a highway scenario. Whenever a lane change is performed, the region will shape in a similar fashion (Figure 41 applies).

In a constant radius corner, or an S-bend this also applies; the regions will look very similar to Figure 42 and Figure 43. At a T-junction the region is shaped very different, taking into account not only the future path, but also the part of the infrastructure that may contain crossing traffic (Figure 46A). In addition to the first region, there are several regions around that to make sure the entire sensor range is covered without having too many objects in every region (Figure 46B).

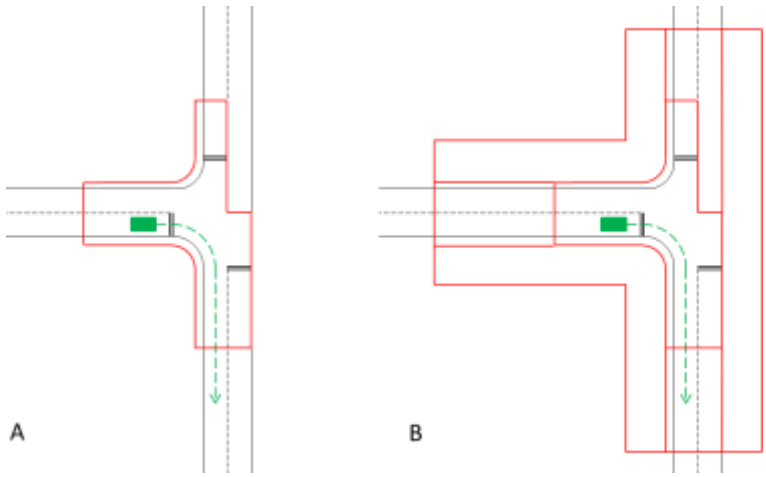

Figure 46. The road-contour method follows the infrastructure boundaries as given by the eHorizon system. A) The first region, closest to the vehicle also covers part of the road from the left. B) A 'shell' of regions around the first region provide a buffer, to ensure all objects within the sensor range are tracked. Not to scale.

An added benefit of this method is that the regions can be reduced in width if there is no interesting infrastructure there. This can lead to excluding objects from the tracker, which may lead to regions that can be slightly longer (larger in driving direction), because the limit of maximum objects within the region is not reached by the narrower shape.

As an example, consider a dual-lane highway, which is Standard in many European countries. In all of the approaches described before, the region closest to the host vehicle is wider than the road because of the fixed width of three lanes (10.5 m). With the eHorizon data, it is known how wide the road is and the region width can be adopted to it, leading to a situation as in Figure 47, where the regions' shape envelops only the two lanes.

Since a maximum traffic density of the lanes does not change (6000 vehicles / hour) but the number of lanes reduces from three to two, the length of the region can be increased before the limit of 15 objects is reached. Equation 9 from Section 3.1 can be used again, but now with $n_l = 2$, leading to a length of 125 instead of 83 m. Note that there are two regions on either side, to cover any unexpected objects that may be of interest, such as debris rolling onto the road.

$$l = \frac{n_v}{n_l \cdot \frac{q_l}{v}} = \frac{15}{2 \cdot \frac{6000}{100}} \cong 0.125\ km = 125\ m \tag{15}$$

Figure 47. On a dual-lane highway, the number of regions can be limited. The width and length can be adopted to the situation as well. Not to scale.

This method has the potential to work in any traffic situation possible, provided it is properly applied. Figure 48 shows how a standard roundabout is approached; the first region (Figure 48A) follows the direction of travel of the host vehicle, as well as part of the roundabout to the left, where other traffic may intersect the host vehicle path. As before multiple regions around this first region are used to cover the entire infrastructure within the sensor range and some part of the roadside to cover any unexpected objects (Figure 48B).

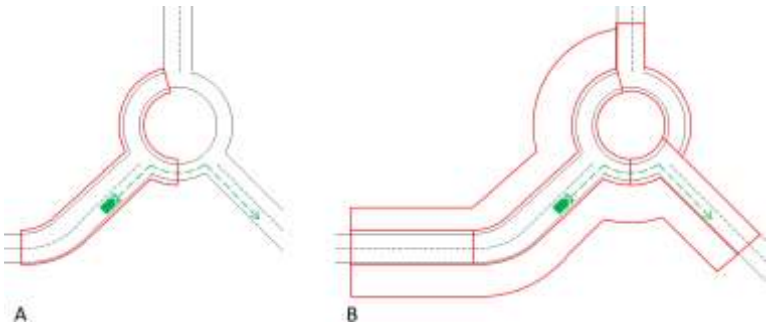

Figure 48. Extended map approach on a roundabout. A) the first region is shaped along the direction of travel and along the infrastructure crossing the future path. B) More regions are shaped around the first region to cover all objects of interest within the sensor range. Not to scale.

The benefit of this method over the path-planning approach is not only the coverage of paths of potentially crossing traffic, but also the fact that these regions do not suddenly change shape. The regions will slide along with the host vehicle, adopting to the infrastructure contours, but this is a smooth process.

It also means that objects will only be tracked once, since only one planned path is used. If a situation occurs where a lane is suddenly blocked on a highway (described before, Figure 45), the

first region will not suddenly switch to the new planned path, but smoothly morph into its new shape.

Benefits

- few regions required
- Crossing traffic is covered
- Should work in all traffic situations

Drawbacks

- Complex shape of regions
- Requires accurate (up-to-date) map data

3.6 Concept discussion

Five methods of generating regions around an autonomous vehicle were discussed in this chapter, each with their own benefits and drawbacks. In this section the methods are compared using six properties and a recommendation is made upon which methods should be subjected to an in-depth analyses.

3.6.1 Number of regions

The number of regions is equivalent to the number of tracking/association algorithms that should run in parallel, because every region will have its own algorithm. In general fewer regions lead to better performance, because less algorithms and computations are needed. Furthermore, less regions mean less boundaries and so less of an overlap area is needed. The static grid approach requires the most regions to cover the entire sensor range, because it does not adapt to the vehicle parameters or the environment at all, it will thus get the lowest score of 1/5.

The VYR method can use less regions, but still needs a fair amount, because the method needs to cover the situations where the instantaneous estimation of the vehicle path is not accurate (S-bend etc.): 2/5. The Shells approach performs another step better, because the shells cover larger areas than the VYR regions; it will get 3 out of 5 points. Since the path-planner method and the eHorizon approach both consider the real future route of the host vehicle, they are able to adapt in the best way to the situation. Both require a roughly equal number of regions to cover the sensor area, so both are awarded 5 points out of 5.

3.6.2 Complexity of regions

Although the computation of the shapes is not too extensive, compared to an association algorithm, it can be important to have simpler shaped regions. Less complexity can also lead to less contradictions or errors during operation.

The simplest method is of course the static grid approach, which requires no external data to generate the regions, this is indicated by giving the highest possible score of 5/5. Close behind (complexity-wise) are the shells approach and VYR method. Both methods have regions that can fairly easily be constructed using geometrical rules (both 4/5).

The last two methods use much more complex shapes for regions, with a slight advantage for the path-planning approach. The vehicle path is already known in detail, the boundaries of the path-planning regions are basically just offsets from that path (score 2/5). The eHorizon regions are shaped with the edges of the infrastructure, making it more complex than ellipses, rectangles and even smooth paths. It is awarded the worst score: 1/5.

3.6.3 Adaptability of regions

This property indicates to what degree the regions can adapt to changing conditions, be it vehicle parameters or environmental.

A special case is made for the static grid approach; since it does not change shape or size regardless of any circumstances, it gets a score of zero. The shells method gets only a slightly higher score (1/5) because it only uses the velocity. In the middle of the scale is the VYR approach, which uses two parameters to adapt to the estimated future vehicle path (3/5).

The path planner method adapt perfectly to the future vehicle path, but only gets 4/5 points, because it does not adapt to the environment. The highest score is reserved for the eHorizon approach, which should shape to any situation.

3.6.4 Required data to construct region

The amount of data required to construct the regions is another indication of how heavy the algorithm is. This time the scale is inverted; whenever a method needs less data, this is beneficial for the processing speed. Less data is thus related to a higher score.

The highest score goes to the static grid method, which does not require any sensor data (5/5). Since the shells approach requires the vehicle velocity and the VYR method needs the yaw-rate in addition to that, they are scored with 4 and 3 points respectively.

The extended maps method is regarded as the second-worst data consumer, because the eHorizon data is readily available from the system, whereas the path planning method requires the future path of the vehicle, which is not supplied by a sensor, rather computed by the host vehicle itself. To construct the future path much more data is needed.

3.6.5 Continuity of region

With continuity of the regions, the transition of the shapes from one frame to the next is indicated. If a system follows a contour, it is regarded continuous. Whenever the shape of a region changes shape drastically from one frame to the next, it is considered discontinuous.

Only one method is very different to the others in this category: the path planning approach. In most cases this method will work fine and the regions are shaped continuously, but in some cases a discontinuity can occur when the planner suddenly decides to take another path. Since this will most likely occur in emergency situations or other (relatively) high-risk scenarios, this can be considered as a substantial problem.

All methods are rewarded a score of 5 points, with the exception of the path planning approach, which scores only 3/5 points.

3.6.6 Expected efficiency of trackers

When a region is expected to not have many objects most of the time, but it is required to ensure coverage in specific scenarios, the efficiency can be said to be low for those regions. As an example, think of the highway scenario with the static grid approach. If the highway is straight, many of the regions to the left and right will not contain any objects of interest. These regions will however have a tracking and association algorithm running. Even though there are no objects in this region, there is a certain amount of computations that needs to be performed, making the processing time required per object relatively high.

From this example it follows that the efficiency of the static grid method is not very high. In fact, it will score only one point. A slightly better performance is expected for the shells and VYR methods, which can have fewer regions (and algorithms) for the same coverage, making them slightly more efficient (both score: 2/5).

The path planning and map approaches have a very high efficiency, since the regions cover much more of the important area around the host vehicle. Although the path-planner method does not

cover all the important space, it can be said that the efficiency of the regions that are created is just as high as with the eHorizon method.

Method	# of regions	Complexity	Adaptability	Required data	Continuity	Efficiency	Total score
Static grid	1	5	0	5	5	1	17
Shells	3	4	1	4	5	2	19
VYR	2	4	3	3	5	2	19
Pathplanner	5	2	4	1	3	5	21
eHorizon	5	1	5	2	5	5	23

Table 8. Summary of the expected performance of the described method to generate tracking regions. Scores are on a 1-5 scale, with 5 being the best score. Reasoning behind the scores can be found in the text above.

Adding up all the scores in Table 8 for every method leads to a total score of 23 for the eHorizon approach, two more than the path-planner method's score. Provided that the method can deliver the claims that are discussed, this method is the best way to divide the surroundings of an autonomous vehicle into smaller regions, to ensure that the association and tracking tasks can be performed sufficiently fast. This method will be worked out in greater detail in the next chapter.

It is however important to keep in mind that it is possible for the GPS signal to fail, in which case the position of the vehicle on the map is no longer known accurately. A fallback method is desired to ensure the safety of the vehicle at all times. Choosing the second best method as a failsafe would be a logical choice, even more so because this method also uses the future vehicle path to construct the regions. For this research the assumption is made that the GPS position and maps are always available and the region tracking algorithm can perform its task at any time.

4 The proposed algorithm in detail

The extended map approach is the best way to go for region splitting in an autonomous vehicle according to the trade-off in Section 3.6, but Chapter 3 only described the general methods. In this chapter the details of the extended map method are described, as well as the expected challenges for combining the regions. The main focus will be on implementing a system that uses infrastructure knowledge to construct the regions in MATLAB.

Section 4.1 will go into detail about overlap between the regions and how to track the objects that are close to the border of a region. The questions about what data can be used and how to use this will be answered in Section 4.2, while Section 4.3 will evaluate whether it is necessary to have different algorithm parameters for different regions. Section 4.4 discusses how the method is implemented in a MATLAB function and finally a description of the implementation of the existing methods to compare performance with is given in Section 4.5.

4.1 How to handle objects close to region boundaries

Whenever the tracking area is divided into smaller regions, there will be boundaries between them. A mechanism should be in place for the situation with an object crossing the boundary, or even just moving close to this edge. Figure 49 shows a situation where tracked objects (red circles) are traveling close to a region boundary (red line). The measurements (blue crosses) that may be associated with this object are not all within the same region.

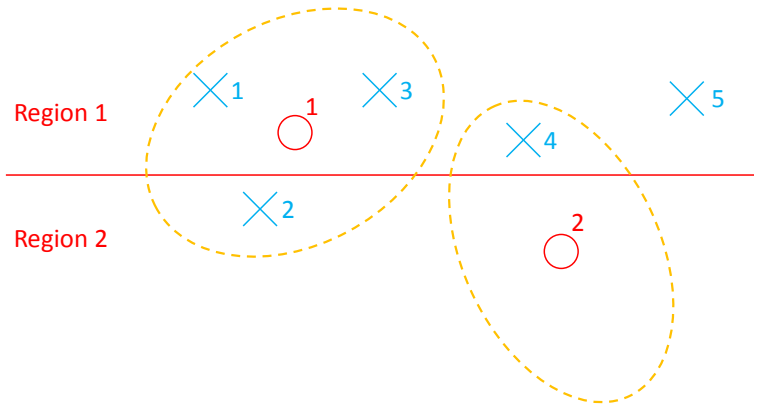

Figure 49. Example of objects (red circles) close to a boundary (red line), with measurements (blue crosses) around it. Confidence ellipses are shown in orange dashed lines.

Assume the situation for the tracker of the top region of Figure 49; since the JPDA filter will try to find the most likely measurement to the track, only the top four measurements are taken into account. Two errors may occur: Firstly, measurement 2 is not taken into account for the estimation of object 1, Secondly, measurement 4 does not belong to any track and may wrongfully be initiated as a new track.

This example shows that it is necessary for all measurements within the confidence ellipse of a tracked object to also be within the same region. This can be achieved by moving the boundary until the complete confidence ellipse fits inside the region. However; moving the boundary may cause another object, or track to get into a similar situation.

A better solution would be to maintain the strict pre-defined boundaries for the tracks, or the predicted position of the objects, but move the boundaries for the measurements, to include all measurements that are within the confidence ellipses of the objects. Figure 50 shows how this may work in the previous example, note that the new measurement boundaries always extend *outwards* from the region the tracker belongs to.

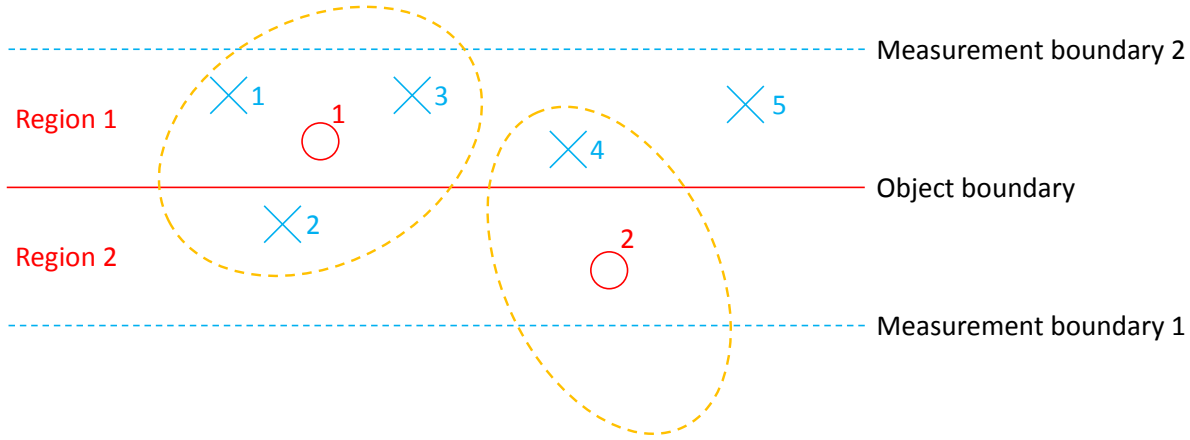

Figure 50. Secondary boundary for measurements, used to include the measurements that are outside the main region, but may belong to the track close to the boundary. Note that the measurement boundaries (blue dashed lines) extend outwards *from the main regions.*

Including more measurements into the region may lead to those measurements being marked as unassociated to any track (measurement 4 for region 1). In this case, a normal tracking algorithm assumes those measurements to be new tracks for the next time-step. However, the unassociated measurements may belong to an object/track that was just outside the original region (outside the object boundary, object 2). It would be wrong to initiate a track in this case.

This issue can be solved by having an extra variable along with every measurement: a *used*-flag. This variable will be set to *false* for every measurement before any tracking commences. Whenever a measurement is used, i.e. it was inside a confidence ellipse of a track in any region, it will be set to *true*. If a measurement is within two confidence ellipses *and* within two regions, it may be set to *true* twice, but the end result is still that the flag is set to *true*. After all the regions have performed their tracking, a check is made and only the measurements that still have a *false* used-flag (measurement 5) will be initiated as new tracks for the next time step.

In the unlikely event that a predicted position of a tracked object is exactly on a boundary, a choice has to be made to include the object in only one region. This choice can either be made to have the object in the closest region to the host vehicle, or in the region with the lowest index (programmatically). The choice is not really important, because all measurements within the confidence ellipse are included in the same region. The choice on how to solve this matter is discussed in Section 4.4.

4.1.1 Example

As an example using the simple scenario specified before, the algorithm steps are given. Assume a frame that looks like Figure 51. With two objects and five measurements close to a boundary between two regions.

First the tracking algorithm for the top region is called; there is one track visible in this region (object 1). The measurements to take into account are all measurements that are above the lower blue dashed line; All five measurements will be given as an input to the first tracking algorithm. Since only the first three measurements are within the confidence ellipse of the track, the JPDA algorithm will only use these to update the state of the track, and thus only the *used*-flags of these three measurements are set to *true*. Measurements 4 and 5 are unused measurements, their flag remains false. Note that no new tracks are initiated yet, this is done after all regions have performed their tracking.

After this process, the JPDA algorithm runs for the second region: In this case there is one track (object 2) and again all measurements are within the *measurement* boundary of the region (top blue dashed line). The JPDA will only use measurement 4, because this is the only one within the confidence ellipse. The flag of measurement 4 is changed to *true*, the rest of the flags will remain as they were.

Now that the tracking is completed for all regions, the track initiation function is called, but a new track is only created if a measurement is still flagged as *false*. In this example only measurement number 5 still has this value, so only this measurement is initiated as a new track for the next time step.

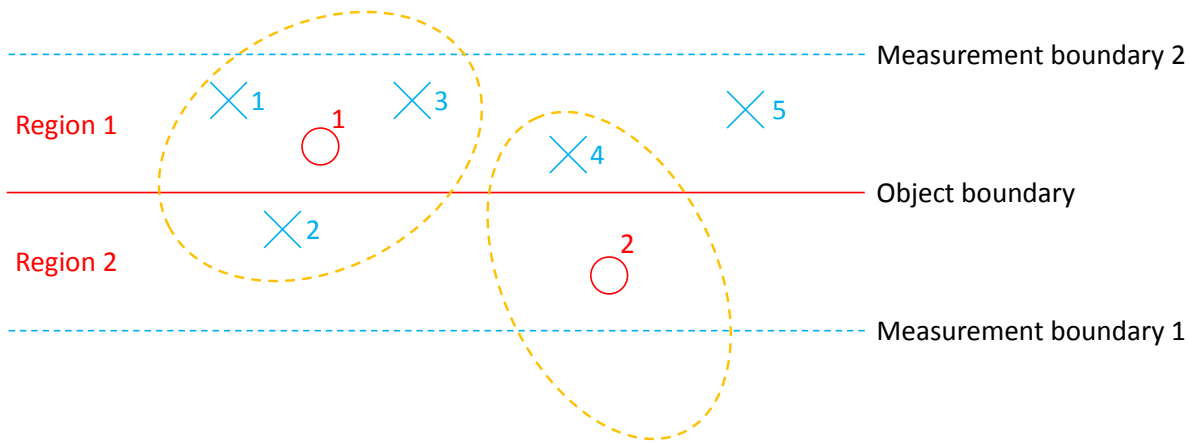

Figure 51. Example scenario to discuss the algorithm steps. circles are tracks, crosses measurements. Red solid line: region boundary, blue dotted lines: measurement boundaries, green dash-dotted ellipses: confidence ellipses.

A benefit of separating the boundaries like this and initiate new tracks only after all tracking is done is that the distance between the measurement boundary and the region boundary can be chosen beforehand. As long as this distance is equal or larger than the size of the largest confidence ellipse, the system will work properly. So instead of checking all confidence ellipse sizes and taking the largest one as the distance, a fixed value can be assumed as a parameter for the algorithm.

4.1.2 Real-life measurement boundary

As an example of how an extended border would look in a more realistic scenario, Figure 48A is copied and the additional *measurement boundary* is drawn into the image (Figure 52). In this image the red solid line represents the object boundary of the first region and the blue dashed line is the measurement boundary of the same region. Note the green dashed arrow that is the future route of the vehicle.

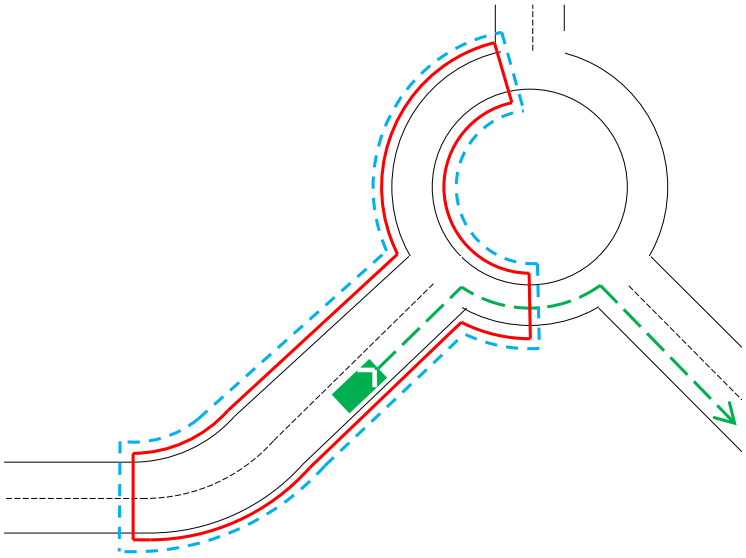

Figure 52. Closest region to the vehicle (red solid line), with the measurement boundaries (blue dashed) in the roundabout scenario. The green dashed arrow represents the future route of the vehicle Not to scale.

To indicate how the first region will look in a range of situations, a piece of virtual road is generated and the region is constructed as the vehicle drives in this scenario. The scenario consists of straight roads, corners, intersections and a roundabout, as can be seen in Figure 53. For the sake of testing the region generation algorithm, the scenario does not have any real-life sizes, but the relative sizes of the sections are realistic. The infrastructure is reconstructed using nodes (black dots) which are connected via sections (grey lines). This format is consistent with the output of the eHorizon system available in the car.

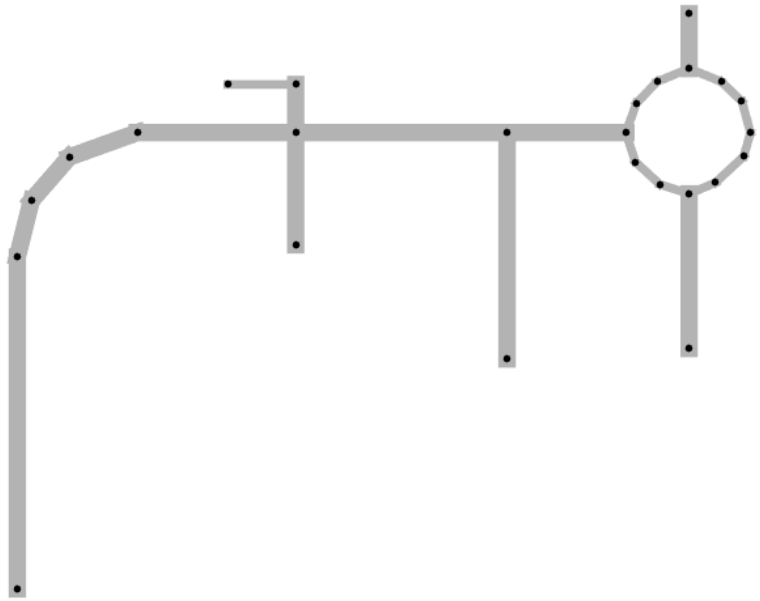

Figure 53. Sample scenario to test the region construction. Black dots are nodes*, Grey lines indicate the* sections *of infrastructure between the nodes.*

In the scenario the vehicle will start in the lower left corner and follow the road, always going straight on the intersections. Then drive three-quarters of the roundabout before stopping in the top-right corner of the image. The region that is constructed is twice as large in the forward direction, compared to the rear of the vehicle.

The first point of interest is the corner, where there are two (or more) sections that are within range of the vehicle. The region generation algorithm will construct a polygon-shaped region, which is created by merging rectangle boxes that enclose the infrastructure of every section. The full process of the merging will be explained in Section 4.2.5. For now it is assumed that this process is performed.

In Figure 54a the constructed region can be seen on the straight, where it is visible that the region extends twice as far to the front as it does to the rear. When the first section of the corner is in view, Figure 54b applies; the two rectangles are merged together to form one polygon, encompassing both sections. Even further into the bend (Figure 54c) up to four sections are combined.

Note that the whole process of combining the rectangles into a polygon is actually taking place twice; once for the object region and again for the measurement region.

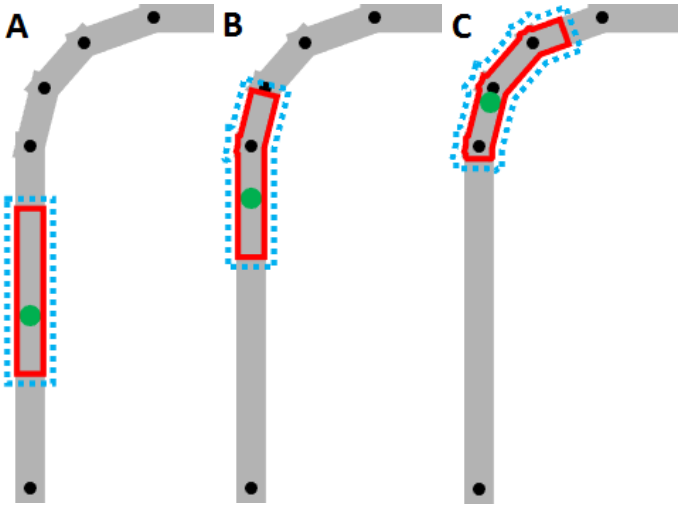

Figure 54. First region as constructed by the algorithm in the simulated scenario, driving towards the first bend. Green dot: vehicle, red solid line: object region, blue dotted line: measurement region.

Next up the road is an intersection, which splits the road ahead into three separate roads. The region constructer will determine all the possible paths within the viewing distance that is implemented and merge all the rectangles that are within range. In Figure 55a and Figure 55b the vehicle has not passed the intersection. The length of the region in all three parts is the same and can even consist of more than one section after the intersection node, as can be seen in the path going to the vehicle's left in Figure 55b.

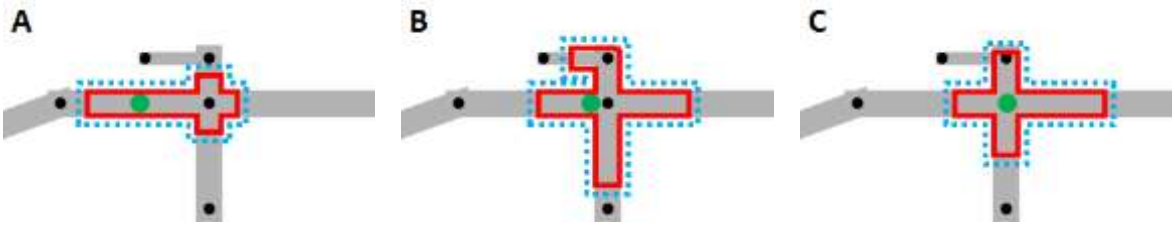

Figure 55. First region as constructed by the algorithm in the simulated scenario, driving along the first intersection.

The situation in Figure 55c is different; the vehicle has just passed the intersection node, meaning the intersection is effectively now behind it. This will make the view-distance shorter, as was indicated in Figure 54a. After the intersection the region will again be shaped as a rectangle, since the vehicle is on a straight road once more.

A final point of interest in this simulated scenario is the roundabout (Figure 56). In the first image (Figure 56a) it is visible that the region is equally long both on the left and right sides. When on the roundabout, complex shapes are possible, such as in Figure 56b. In Figure 56c it is once more clear that the rearward view is less than the forward view.[1]

[1] A video of the scenario described in this section is available on request by e-mail: rstakelbeek@gmail.com.

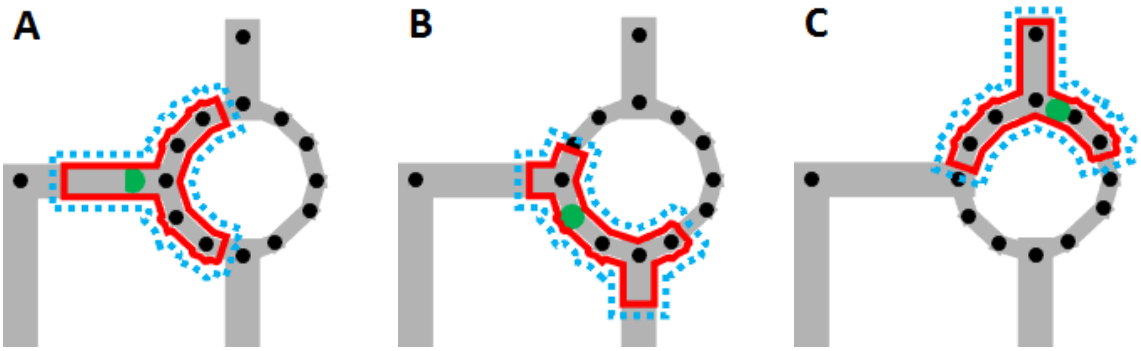

Figure 56. First region as constructed by the algorithm in the simulated scenario, coming up to and driving on the roundabout.

4.2 From data to region definition

Before diving into the details of how to construct the regions around the vehicle using the extended map data, first it is required to find out what type of data is available. Section 4.2.1 will describe what data is given by the eHorizon system and how this can be used by the algorithm, while Section 4.2.2 and 4.2.3 explain the location of the object and measurement boundaries. The next two sections are used to describe the process of generating the region boundaries, with Section 4.2.4 explaining how the possible paths on the infrastructure are found and Section 4.2.5 discussing the process of merging the rectangles of the sections into the complete region boundary polygon.

4.2.1 What data does eHorizon provide?

The eHorizon system outputs its data according to the ADASIS protocol [46]. In this protocol the path of the infrastructure is described using *stubs*. Every stub is a section of road with similar parameters such as maximum speed or number of lanes. Whenever a parameter of the road changes, a new stub is given. In case of corners or bends in the infrastructure, the system divides the road into smaller parts that can be processed as straight sections of road. The protocol specifies that the road trajectory can be obtained by linear interpolation between the given stub locations (See Appendix C for more information).

The given stub locations define the center line of the road. From other messages such as; lane width, and number of lanes, the required information to reconstruct the infrastructure is gathered. The system thus provides everything to reconstruct the roads. Aside from the infrastructure information, the system also provides the relative location of the ego vehicle on the stubs. The offset from the starting point on the current stub is given, as well as the lateral position on the road section. This places the vehicle on an exact point relative to the infrastructure. Table 9 shows all the data that is required by the algorithm to construct the regions.

Part	Variable	Description
Section (stub)	ID	Identifier for sections
	Start node	Starting point of sections (node ID)
	End node	Ending point of sections (node ID)
	Length	Length of section
	Number of lanes	-
	Lane width	-
	Heading	Global heading of section
	(Road width)	Can be calculated
Node (connections)	ID	Identifier for nodes
	X-position	Global position (GPS converted to Cartesian coordinates)
	Y-position	Global position (GPS converted to Cartesian coordinates)
Ego vehicle	Current section ID	-
	Longitudinal distance	From section start
	Lateral distance	From centerline

Table 9. Parameters from the eHorizon system required by the algorithm to form the regions around the infrastructure.

Some of the information in Table 9 is redundant and can be calculated from other variables. For example; it is possible to obtain the length of a section when the position of the start- and end-nodes is known. If the information is not readily available from the system it can be obtained by

some simple calculations. The ADASIS-protocol states that all the above parameters should be available for use.

4.2.2 Region object boundary

The regions are constructed in the algorithm as polygons. For every section of road a rectangle will be defined that has the size of the infrastructure plus a margin of error. To indicate the boundaries, refer to Figure 57, where all the boundaries are shown. This section will discuss the distance to the object boundary d_O, whereas the measurement boundary (d_M) is explained in Section 4.2.3.

The distance d_O provides room for objects that are close to the boundary of the road, but not on it. For example in an urban environment, a person might walk from a park (outside of the infrastructure) onto the road, to cross it. The margin of error should be large enough to detect and confirm the existence of the person, to be able to act upon it if required.

The size of the margin is a parameter for the region construction algorithm so it can be changed easily, possibly even *on-the-fly* depending on the environment the vehicle is in. In this research the margin is chosen as a fixed number for all scenarios. The size is based on the *Cramer-Rao lower bound* (CRLB) [47] which gives an optimal value for the tracker covariance, and thus the required extra space between the infrastructure boundary and the object boundary.

However, the CRLB gives a value for the tracker (combined with the measurements) accuracy only (c in Figure 57), but this assumes that the location of the sensors is exactly known. In this project, this location is given by the GPS locator system, which uses *differential GPS* to determine the location of the vehicle in the World (a in Figure 57). A further error can occur if the map information of the eHorizon system is not exactly as the real World (b in Figure 57). There will always be small differences between the real infrastructure and the maps that have been programmed into the computer systems. All these inaccuracies have to be summed up in order to find a suitable value for the distance between the infrastructure boundary and the object boundary.

$$d_O = acc_{tracker} + acc_{loc} + acc_{eHor} \tag{16}$$

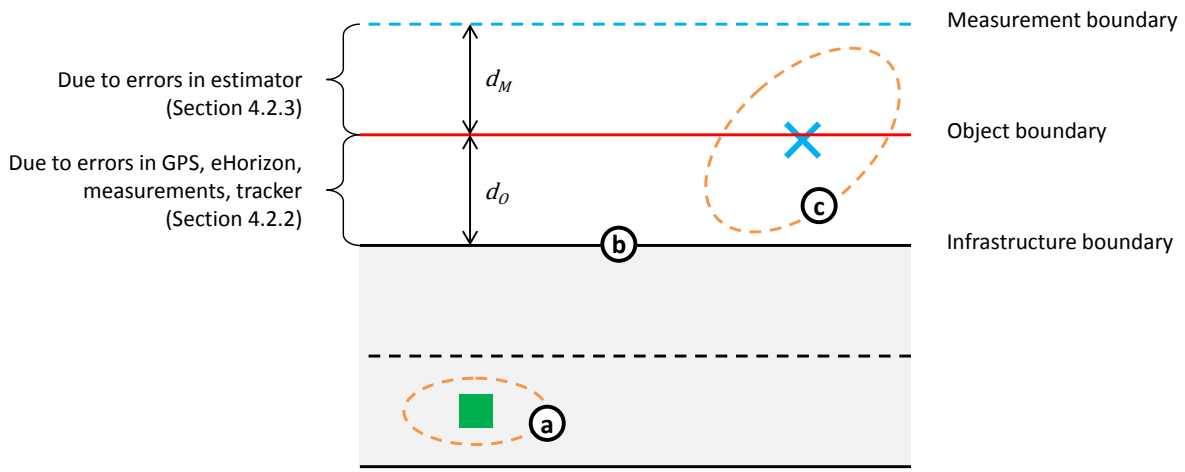

Figure 57. Indication of the different boundaries of the regions (not to scale). The infrastructure boundary is what is desired to be the region boundary, but several measurement errors and accuracies make it necessary to extend the object boundary outwards. Similarly for the measurement boundary (Section 4.2.3).

The CRLB in Equation (16) is the worst case scenario for the tracker while the noise levels are consistent with the noise levels in the specification of the radar sensors [48]. The worst case scenario is found by trying all the ranges and angles within the survey area of the radar sensor and computing the covariance after three steps of the filter. The three steps represent the time required to confirm an object as a track in the algorithm. The obtained covariance matrix gives information about the tracker accuracy in both x- and y-directions. To make sure that the whole infrastructure is within the object boundary, the maximum of those values is taken.

While looping over all the ranges and angles within the sensor survey area, the first fact to note is that the tracker is performing its computations in a Cartesian frame, whereas the sensor specifications are given in a polar coordinate frame, with range and bearing. The first step is to convert the sensor specifications to something useful for the tracker

The estimate for the first covariance matrix can now be obtained by using the diagonal elements of the measurement noise. This will of course be an estimation of the true covariance, since it is not possible to transform the exact shape of the error ellipse from polar to Cartesian coordinates, with only parameters in the x- and y-direction.

The Cramer-Rao lower bound (CRLB) [7;47;50] describes the lower bound of the variance of the tracker, which gives a good indication of the performance of a tracker that has converged to its optimal operation. The CRLB does not take into account the false alarm measurements and model inaccuracies, so the method has to be slightly changed to suit the needs for this project. In [7], the parameter q_2 is introduced as an *information reduction factor* (IRF) to correct for false alarms and missed measurements.

In [50;51] a method for finding the *posterior* CRLB is presented that does not involve many matrix inversions, rather it iteratively determines the bound. This process is not unlike running the filtering algorithm for a number of cycles and find the covariance matrix. In the case of a linear motion model and assuming Gaussian zero-mean white noise, the information matrix can be determined as shown in Equation (17). To make sure the filter is settled, the equation is performed 10 times.

$$J = (Q + A \cdot P_k \cdot A')^{-1} + \mathrm{q}_2 \cdot H \cdot R^{-1} \cdot H' \tag{17}$$

The first part of the formula in Equation (17) is the filter updating the covariance matrix as it would during normal operation, adding the process noise to the outcome. The second part is based on the sensor accuracy; it takes the measurement noise matrix and determines the estimated measurements, and finally multiplied with the IRF, to take care of false alarms and missed measurements. The tracker is tested for every resolution-cell within the range of the sensor and the worst case outcome is taken as a basis.

The resulting information matrix has to be inverted to obtain the estimated lower bound of the covariance matrix. From this matrix the expected *tracker* accuracy can be computed by taking the maximum eigenvalue. Inputting all the sensor specifications and tracker parameters, and then taking the eigenvalue of the resulting covariance matrix, gives the variance expected for the tracker operation. It is required to have all the objects within the object region boundary, so the value for 2σ is taken as the accuracy for the tracking algorithm (Equation (18)).

$$\begin{aligned} &acc_{tracker} = 2 \cdot \sigma = 2 \cdot \sqrt{Var} = 2 \cdot \sqrt{0.14} \cong 0.75\ m \\ &with \quad Var = \max(\lambda_{cov}) \\ &and \quad \lambda_{cov} \quad the\ eigenvalues\ of\ the\ covariance\ matrix \end{aligned} \tag{18}$$

The second error that will lead to a larger required d_O is the accuracy of the localization system used to localize the host vehicle. In this project the *Advanced Navigation Spatial Dual* system is used [52], with access to *Satellite-Based Augmentation Systems* (SBAS) and corrections using the vehicle odometry. In future iterations of the project RTK corrected GNSS will be used, leading to a sub-centimeter accuracy. The current implementation however, gives an estimated accuracy of 0.5 m.

The total distance d_O also takes into account the accuracy of the created maps. Continental is supplying the eHorizon system and in their specification a partnership with HERE is mentioned [53] for the maps that are used. In their own published text, HERE describe their aim of having maps that are up to 10-20 cm accurate, to allow smart vehicle systems to operate properly [54]. These texts lead to the assumption that the accuracy of the map data used in the eHorizon system is 0.2 m.

The total margin of error can be calculated by simply adding all the parts together, leading to the parameter d_O for the object region:

$$d_O = acc_{tracker} + acc_{loc} + acc_{eHor} = 0.75 + 0.5 + 0.2 = 1.45\ m \tag{19}$$

4.2.3 Measurement boundary extension

The value to be used for the distance of the measurement boundary from the object boundary (d_M, Figure 57) is evaluated in this section. For this purpose an experiment is conducted with parameters that are similar to the real-life sensor data.

It is assumed that most of the position data is available from the radar sensors that are on the vehicle. In the specifications of the Continental ARS-300 series [48] the expected accuracy is 1.5% of the range and 1^{O} in the angle. In Section 4.2.2, the posterior CRLB led to a covariance matrix for the tracker during nominal operation. This covariance matrix is taken as a basis for an experiment to determine the required size of d_M.

Using this covariance matrix, the Kalman filter inside the tracker is called, but without supplying measurements. This resembles the scenario where there are no measurements available for the object, but the track is not deleted, due to its previously verified status, i.e. the counter of the track has not reached zero (See Section 2.2.3). The covariance of the object will obviously grow in this moment, because there are no corrections that can be made based upon measurements (Figure 58).

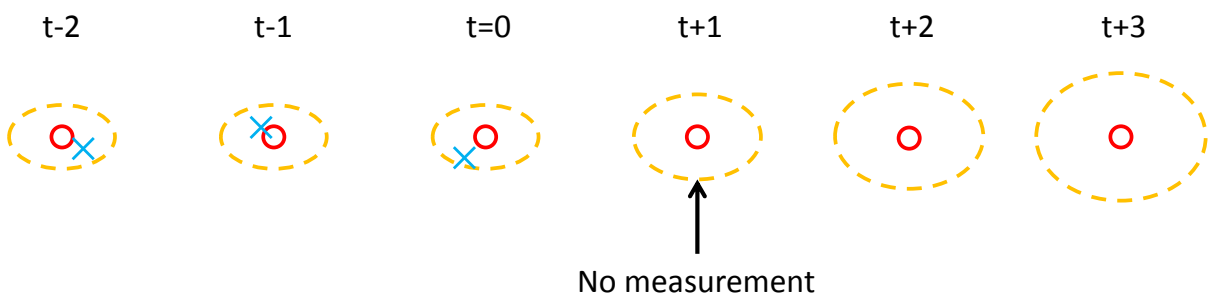

Figure 58. Indication of the growth of the confidence ellipse (covariance matrix) when no measurements are available. Image is not to scale. Red circles: estimated object location, blue crosses: measurements, orange dashed lines: confidence ellipses.

The covariance matrix after the tenth step without measurements is taken as a guide to determine the size of d_M. The reason for taking ten steps is that this resembles a track with the worst accuracy, before it gets deleted due to lack of updates. The maximum eigenvalue of the covariance matrix is approximately 0.18, which equates to the variance in position. The standard deviation can be computed by taking the square root of the variance.

Since a normal-distribution of the noise levels are assumed in a Kalman filter, the predicted location can also be described by a Gaussian function. The object is expected to be at the predicted location with a standard deviation as calculated. This means there is a 95% likelihood that the object will be within two times the standard deviation of the predicted location. Calculating the value for the measurement boundary extension can thus be done as shown in Equation (20).

$$d_M = 2 \cdot \sigma = 2 \cdot \sqrt{Var} = 2 \cdot \sqrt{0.18} \cong 0.85\ m \tag{20}$$

This means the measurement boundary should be approximately 0.85 m on the *outside* of the object boundary. This parameter will be used throughout the experiments as a fixed number.

4.2.4 Process to find all sections within region range

Before merging is possible, all the sections within range of the region should be identified. An iterative process is used to determine this. The section in which the vehicle is located is known, as well as the position within this section. The distance to the end-node of this section is then also known. Whenever this distance is less than the *view-distance* used, the algorithm will search for all sections that are connected to this node.

Excluding the current section, this leaves all the possible next sections. The process now continues with all these sections; if the end-node of the section is within range of the region, a search is performed on all sections sharing that node. If a section's end-node is further away than the required distance, the section is cut at the required length of the region. This means that instead of a rectangle that contains the entire section, the rectangle will be shorter than the section in the longitudinal direction. This cutting-off is visible in nearly all of the examples from Figure 54 through Figure 56. When this process is finished for all paths, the list of sections within range is complete. A flow chart of the process is displayed in Figure 59.

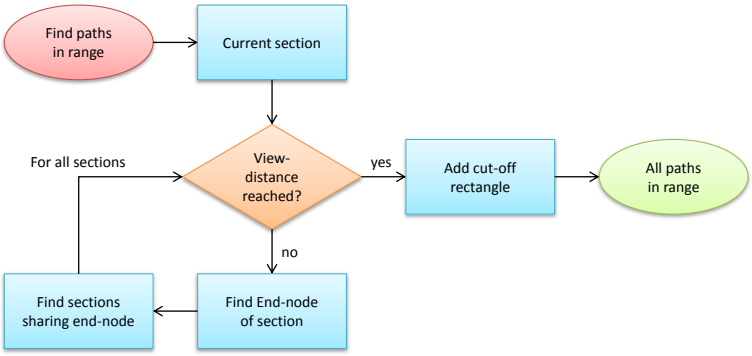

Figure 59. Flow chart of the process of finding all the paths within the view-distance. Start at the red circle (left) and follow the tree. A blue rectangle specifies an action, an orange diamond asks a question. Continue until the end is reached (green circle).

Of course this iterative process has to be performed both to the front of the vehicle and to the rear, taking into account that there are possible different viewing distances for each direction. A more sophisticated method could be implemented that also changes the view-distance into side-roads accordingly. This would however require more computations and an interconnection with the path-planning algorithm. At this time the simple fixed-distance method is used.

4.2.5 Merging section-polygons

While determining which sections are located within the region, the rectangles are immediately added to the region polygon. Doing the merging immediately after identifying the sections to add requires less programming statements and is thus faster than separating the two processes. This also means that the merging always takes place between one polygon and one rectangle.

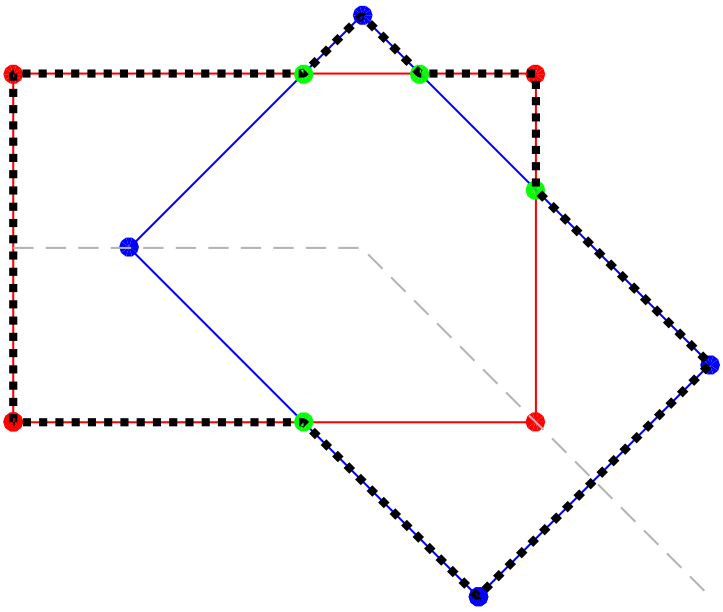

Figure 60. The red polygon and blue rectangle are merged together to form one polygon (black dotted line). At every intersection of polygon vertices (green dots) the resulting polygon shifts from one input shape to the other. Details in text.

In Figure 60 an example is given for two rectangles, red and blue, that are merged into one polygon black dotted. In the paragraphs below the process of this merging is explained step by step.

Before the true merging starts, several checks have to be made on the input. First of all, the algorithm assumes that all polygons (or rectangles) are defined in a clockwise fashion. This means that the points that are defined as corners, are ordered. In case of the red rectangle: the first point is the lower-left, the second the upper-left, then the upper-right and the lower-right.

The function used in this thesis does *not* perform the check for a clockwise definition of the polygons. It is assumed that the rectangles are always constructed in a clockwise manner, leading to a clockwise defined output. By not performing this check, some valuable processing time is saved. It does however make the merging function less generally applicable.

If the check for a clockwise definition is to be made, the process is as follows: First find the lower-left-most corner of the polygon by performing a simple search for the minimum coordinates. Then take this point and its two neighbors from de input polygon. Define the line between the first point and the second as u and the line between the second and third points as v. Next perform the simple calculations from Equation (21). If the value of a is smaller than b, the polygon is defined clockwise, else it is counter-clockwise.

$$\begin{aligned} ux &= x(2) - x(1) \\ uy &= y(2) - y(1) \\ vx &= x(3) - x(2) \\ vy &= y(3) - y(2) \\ a &= ux \cdot vy \\ b &= uy \cdot vx \\ cw \quad & if\ a < b \end{aligned} \tag{21}$$

Another input check that *is* inside the function is to verify whether the first point of polygon 1 (red) is outside of polygon 2. This *must* be true in order for the merging to work properly. The assumption is therefore that the polygons are actually intersecting: They are not two polygons far apart and one polygon does not enclose the entirety of the other.

The check is performed by a function [55] based on the MATLAB-function `inpoly` which returns a Boolean vector of the same size as the number of inputs. If the first point of the first polygon is *inside* the second polygon, the corner-numbering is changed until the new first point of polygon 1 is *outside* of polygon 2. The same check is now performed on the second polygon, to make sure that its first corner is also outside of polygon 1.

The method for this check is to determine how many times a line into a specified direction, coming from the point of interest is intersecting a vertex of the polygon. If this is an odd number, the point lies within the polygon, else it is outside. The algorithm for this check has been described as early as 1962 as *algorithm 112* [56].

The real process of merging can now start, since the input is in the correct format. The first step is to find all the intersections of the two polygons (green dots in Figure 60). The intersections can be found by checking for all the line pairs whether they cross or not, by solving a system of equations. Let L_1 and L_2 contain the coordinates of both endpoints of two line segments:

$$\begin{aligned} L_1&: \ x_1(1), y_1(1) \ \ and \ \ x_1(2), y_1(2) \\ L_2&: \ x_2(1), y_2(1) \ \ and \ \ x_2(2), y_2(2) \end{aligned} \tag{22}$$

A variable $t1$ can be defined as the relative position of an intersection along the line L_1. t_1 will be 0 if the intersection is at the first end-point of L_1 and 1 if it is at the second end-point. Any value in between means the intersection is between the two end-points and thus is a real intersection of the line segments. A similar variable is defined for the second line (t_2). The coordinates of the intersection will be designated as x_0 and y_0. A system of four equations can be created with this knowledge, with four unknowns.

$$\begin{aligned}(x_1(2) - x_1(1)) \cdot t_1 &= x_0 - x_1(1)\\ (x_2(2) - x_2(1)) \cdot t_2 &= x_0 - x_2(1)\\ (y_1(2) - y_1(1)) \cdot t_1 &= y_0 - y_1(1)\\ (y_2(2) - y_2(1)) \cdot t_2 &= y_0 - y_2(1)\end{aligned} \tag{23}$$

Or written in matrix notation:

$$\begin{bmatrix} x_1(2) - x_1(1) & 0 & -1 & 0 \\ 0 & x_2(2) - x_2(1) & -1 & 0 \\ y_1(2) - y_1(1) & 0 & 0 & -1 \\ 0 & y_2(2) - y_2(1) & 0 & -1 \end{bmatrix} \cdot \begin{bmatrix} t_1 \\ t_2 \\ x_0 \\ y_0 \end{bmatrix} = \begin{bmatrix} -x_1(1) \\ -x_2(1) \\ -y_1(1) \\ -y_2(1) \end{bmatrix} \tag{24}$$

Once solved, the intersection will be added to the list of intersections if the conditions for t_1 and t_2 hold:

$$intersection\ \ if\colon\ \ 0 \leq t_1 < 1\ \ and\ \ 0 \leq t_2 < 1 \tag{25}$$

A derivative of the MATLAB-function `polyxpoly` is used [57] which gives as an output the coordinates of the intersection point and also the vertex-index on which the intersections is, for both polygons. The vertex-index is essentially the identifier for the sides of the polygon; vertex 1 is the line between corner 1 and corner 2 of the polygon, vertex 2 is between corner 2 and 3, and so on. These indices are very useful in the programming of this function.

All of the corner-points of the merged polygon are now known, but the correct order is not, since they are contained in three structures: one for the red polygon, one for the blue polygon, and one for the intersections. Getting all points into one output and in the correct order takes only a *walk* around the polygons.

Start at the first corner of the first polygon and check whether there is an intersection on the path to take, which is the first vertex. If there is *no* intersection, proceed to the next corner, add it to the list, and repeat the process on the next vertex. If there *is* an intersection on this vertex, check how many intersections there are on this vertex. If there is only one, add this intersection, then also add the next point on the next polygon.

Adding the next point of the other polygon after the intersection is possible, because the polygons were defined clockwise and the added polygon is not a complex shape, but a simple rectangle. If there is more than one intersection, find the one closest to the starting point of the vertex. Add the

intersection and the next point on the next polygon, as before. The process of adding corners and intersection points needs to continue until the first point of the first polygon is reached again.

To make the process of merging the polygons a little more clear, a flowchart is presented in Figure 61. Start at the left red circle (objective) and follow the tree until the end point is reached at the right green circle (solution). Blue rectangles are actions to take, while orange diamonds are questions that need to be answered.

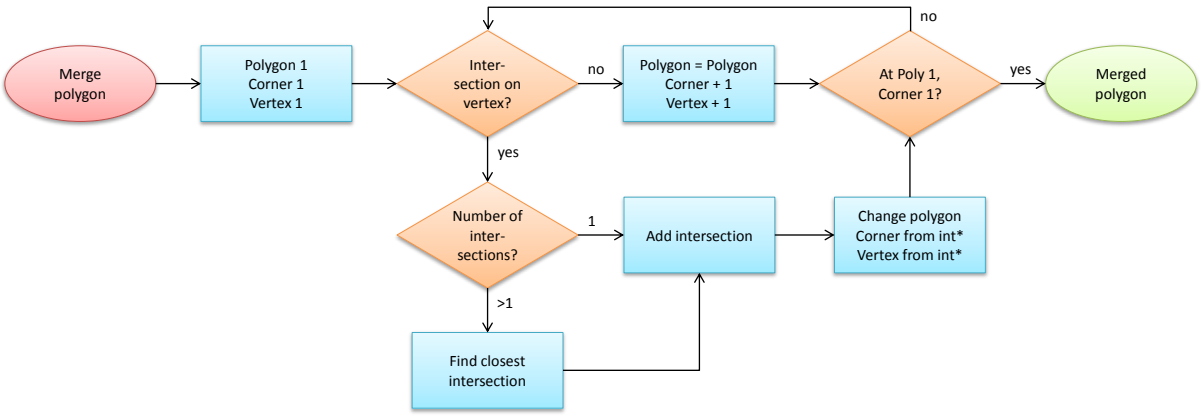

*Figure 61. Flowchart of the process of merging two polygons together. Start at the red circle (left) and follow the tree. A blue rectangle specifies an action, an orange diamond asks a question. Continue until the end is reached (green circle). *The vertex index and corner number can be obtained from the intersection-data.*

Once the merging of the polygons is done, the output of the algorithm will be a polygon describing the boundary of a region. This can then be used the next part of the algorithm to determine which points should be tracked.

4.3 Region tracker in this research

Although the final goal is to generate many regions, as explained in Section 3.5, for the initial experiment the choice has been made to generate only one region close to the vehicle. This region will contain all the infrastructure for a certain distance in front of the vehicle and a certain (other) distance behind the car.

Splitting the environment into only two regions may not sound as a big potential improvement over a single region (JPDA on the entire survey area), but aside from the region shape, that should exclude less important objects already, another method to improve speed is proposed.

The accuracy of tracking gets less important whenever objects are at a greater distance. For example: On a highway it is not essential to know whether a vehicle is 150 or 151 meters away. The driving decisions will not dramatically change with only one meter between the two scenarios. In an urban environment however, at closer range, the difference of one meter may be very important to know. It could mean the difference between assuming a pedestrian is on the sidewalk, or on the road ahead.

Ultimately, the tracker is responsible for the accuracy of the object locations. Of course the sensor accuracy plays a big role, but by combining information, a tracker can estimate a position of an object more accurate than a single measurement could do.

For the current experiment it is proposed to use the accurate, but computationally heavy JPDA algorithm for the region closest to the car, while everything outside of that region is filtered using a simple and fast Kalman filter with nearest neighbor association. This will speed up the tracker for the outer region immensely, while maintaining a higher accuracy closer to the vehicle.

It is also possible to use a clustering JPDA algorithm inside the region, but it is expected that clustering will have only a minor effect, due to the low number of objects that are within the region. The whole point of the region is to keep the number of objects low, so an extensive algorithm can be used inside.

4.4 Implementation of region-tracker algorithm

To test the proposed method, a scenario is developed in PreScan, a software package that can simulate a traffic environment of choice and generate data comparable with real sensor data. This data is output as a MATLAB structure and thus the decision is made to implement the algorithms also in MATLAB code.

In the real vehicle the code will be compiled and is expected to run much faster than on a desktop computer using MATLAB functions, but for this research it is more important to show the relative performance difference between the existing and proposed methods. In any case the actual processing time will depend greatly on the used hardware, be it in a desktop computer, or in the actual vehicle.

In Sections 4.1-4.3 the construction of the regions was explained, but this is not the end of the algorithm. To make sure the objects within this region are tracked correctly, it is necessary to determine which tracks and which measurements belong to it. This paragraph will discuss the general outline of the algorithm, for one complete step in time.

Similar to the other tracking algorithms used in this research (JPDA and clustering), the input for the region-tracker is quite simple. First of all the measurements from the (radar) sensors have to be known and the second input is also straight-forward: The known tracks, given in the same structure as with the other algorithms. Thirdly the parameters for the algorithm are given, these include the filtering parameters, motion model, etc. But should now also include the parameters specific to the region constructor, such as the size of the measurement boundary and the view-distances. Finally an extra input is required that describes the environment and infrastructure. This data is of course the data taken from the eHorizon system.

In normal operation the first step of the region-tracker is to add IDs to the measurement, so that whatever the order, it is still known exactly which measurement it originally was. This is implemented by simply extending the matrix containing the measurements with one additional column, containing the original row-number of the measurement.

The next step is constructing the regions following the process described in Section 4.2. Note that it is necessary to generate a boundary twice: Once for the object boundary, and once for the measurement boundary. With the boundary definitions known, the next task is to determine which tracks are within the object boundary, and which measurements are within the measurement boundary. The information is stored in a separate data-structure for every region.

The actual tracking algorithms then take over: The full JPDA for the region close to the car, and the nearest neighbor Kalman filter (NNKF) for the other region. In future implementations a JPDA tracker can be used for all of the regions, with the possibility to run the trackers for every region in parallel to speed up this part of the algorithm. Keep in mind that the trackers in both regions only associate and filter, the track maintenance takes place *after* all the trackers have performed their tasks. The only thing the trackers will need to do is set the correct *used*-flag for the measurements.

The final step of the region-tracker as a whole will thus be the maintenance of the tracks; Tracks that were not updated for a certain amount of time/frames will be deleted, while measurements that

remain with their used-flags as *false* will be initiated as new tracks for the next time step. A diagram showing the structure of the function and sub-functions is shown in Figure 62.

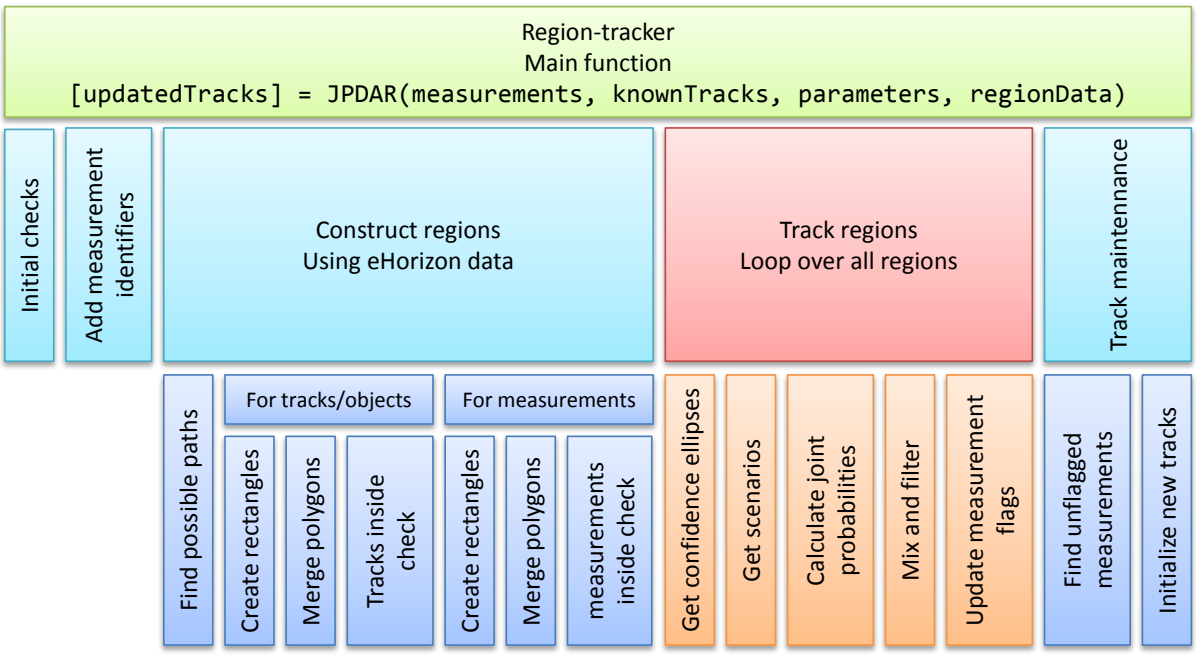

Figure 62. Visual representation of the region-tracker code and sub-functions.

From this diagram it is also clear that a large part of the algorithm is not directly the tracking of objects, rather the preparation to perform the tracking properly. The actual association and filtering sub-functions are indicated in red and orange colors in Figure 62. Although this diagram is not to scale concerning processing time, it gives an indication to how much work has to be done before the actual tracking takes place. The benefit of the proposed method (time-wise) should be enough to justify the extra overhead calculations.

4.5 Implementation of existing methods

The method of region splitting using the extended maps of the eHorizon system has been explained (Sections 3.5 and 4.1-4.3). To see how this method performs, a comparison is made with two other algorithms; a JPDA filter [7] without clustering and with the clustering method from Dezert and Bar-Shalom [38]. In this section the implementation of these methods will be discussed.

4.5.1 JPDA filter without clustering

The JPDA filter without clustering will act as a base-line for all other algorithms. The JPDA algorithm is a proven method [7] and is accurate enough to be used in a safe manner in an autonomous vehicle. However the processing time is a big problem for this method, especially if many sensors are used and the environment get more complex, such as in an urban scenarios. Section 2.2 already explained the basics of the method, but a more in-depth description of this implementation is given in this section.

As with all the tracking functions, the JPDA algorithm is implemented in MATLAB code and requires the same simple inputs: The new measurements, the known tracks and a data-structure containing the parameters for the algorithm, such as the motion model and noise estimation matrices. The output will be the estimations of the tracks for the next time step. A diagram of the function-structure is shown in Figure 63.

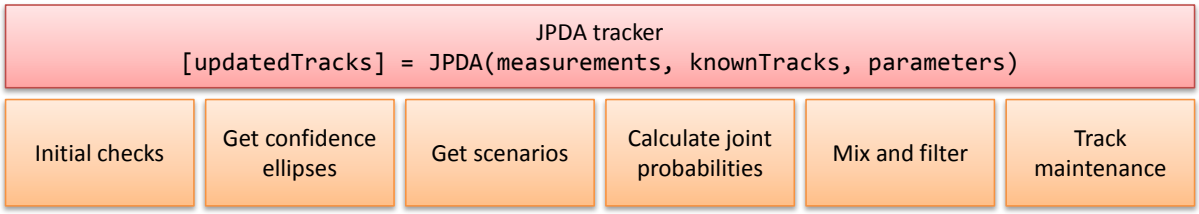

Figure 63. Visual representation of the JPDA tracker code and sub-functions.

The function starts by checking whether the input conditions are correct and if they are, continues with generating the confidence ellipses for all the tracks. Once the confidence ellipses are created, the validation matrix (Ω) is constructed and used to determine the possible scenarios. Then for every scenario the probability of it being the correct one has to be calculated, before the mixing and filtering stage can commence. The last part in the function is about the track maintenance; deleting tracks that were not updated for a number of frames and generating new tracks for measurements that were not used. The mathematics of this process have already been discussed in Section 2.2.

A perceptive reader may have noticed that the JPDA trackers is very similar to the red/orange shaded part in Figure 62 of the region tracker. In fact they are the same, except for the first and last orange shaded blocks. The initial checks in the JPDA algorithm take place at the start of the region tracker-function, so they are not shaded orange in Figure 62. The same is true for the track maintenance block that is moved to the end of the region tracker function. A last difference is that the measurement used-flags do not occur in the JPDA algorithm. All block in between are the same and both algorithms actually call the exact same sub-functions to perform these tasks.

4.5.2 Clustering

The clustering method [38] was introduced in Section 2.5 and is implemented in such a way that it has the same inputs and outputs as the JPDA algorithm without clustering. The generation of clusters is performed *inside* the function, so the cycle time will include both the cluster generation *and* the association and filtering algorithms.

As can be seen in Figure 64, the shape of the *clustering-tracker* is roughly the same as that of the region tracker, shown in Figure 62. Some initial checks are performed before the clusters are defined and constructed. Tracking takes place for every cluster (red and orange shaded in the figure) and afterwards the track maintenance is executed. The difference is of course that in this method there are clusters instead of regions.

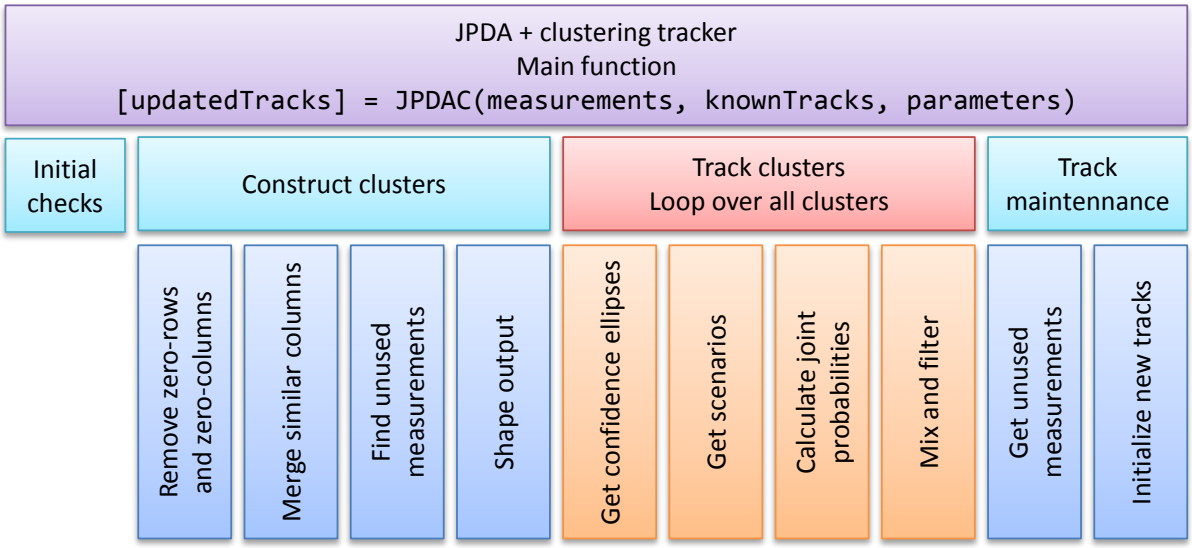

Figure 64. Visual representation of the clustering tracker *code and sub-functions.*

Again it can be noticed that the actual tracking part (red and orange shaded), which will run separately for every cluster, is very similar to the JPDA algorithm. In fact most of the sub-functions are shared by both methods. In the current implementation the clusters are processed sequentially, one after the other, but in a modern computer system it may be possible to track several clusters in parallel at once, reducing the resulting processing time further.

5 Experiment

The proposed method has to be tested against the existing methods, to determine the performance difference. For that purpose an experiment is devised which includes simulated objects using the software PreScan (Section 2.3 and 0) and recorded data from the eHorizon system. Section 5.1 will go into detail about the experimental setup and assumptions made, while the results of the experiment are presented in Section 5.2.

5.1 Description of scenarios

The simulation software PreScan does not currently support the output of an extended map-system, such as eHorizon used in the vehicle in this project. This complicates the experiment, because simulating a realistic output of the eHorizon system is a very extensive job. To prevent any errors during this experiment, the decision was made to record the output of the eHorizon system, while driving on the public roads. Since the data gathered is the actual output, this part of the experiment is not a simulation, rather a play-back of real-life data.

5.1.1 Sensors

Although it would have been possible to also record the data from the other sensors on the vehicle, this would not be realistic for the experiment. Since the performance of the tracker is *the* parameter to test, it is required to have a ground-truth for all the measured objects. It would very hard to obtain this ground-truth in a non-coordinated scenario. As a reminder; the objects around the vehicle would be normal traffic and objects, and therefore are potentially very unpredictable in their behavior.

Instead of using real sensor data, the objects are simulated in PreScan, where a ground-truth is available to test the accuracies of the trackers against. The infrastructure of the recorded data was imported into the software, as was the GPS-track of the vehicle during the recording. This means that the trajectory of the vehicle, the speed-profile and the eHorizon data are exactly as they were during the road-test.

With the PreScan software it is now possible to create objects at specific locations and perform experiments in a very controlled way. The objects are *measured* using a sensor that has the same specifications as the Continental radar [10;48] used in the vehicle. This means the maximum range is set to 60 meters while the field of view (FOV) is 56 degrees (28 degrees to either side). The resolution and accuracies are also taken from the specification sheet; the given *accuracy* is assumed to be 2σ.

The ground-truth data is also gathered with a *sensor* in PreScan; according to the software manual the *AIR-sensor* (Actor Information Receiver) can be used for such a task. This sensor gives the true location of all the objects in the scenario, as well as an identifier for each object. This identifier is also *measured* by the radar sensor and in this way the error between a measurement and the ground-truth can be found.

This AIR-sensor has three options for giving the location of an object: bounding-box edge (BBE), bounding-box center (BBC), and center of gravity of the object (COG). Using the BBE as ground-truth

is not a good idea, since the perceived location of the object will change, with the direction in which the object is observed.

Figure 65 shows a PreScan model of a 20-year old ash tree that is used in the scenario. The bounding box of this object is shown in red, with its center indicated by the blue lines and dot. The COG is shown as a set of yellow arrows, which are extended into gray lines. When this object is detected by an AIR-sensor in BBE-mode, located to the left of the picture, the return will be a point at the number 1. When looking from below the picture, the dot at 2 will be given as a result and when looking from the right; point 3. In fact, the entire red line is the location of the return in BBE-mode, depending on the direction the object is viewed from.

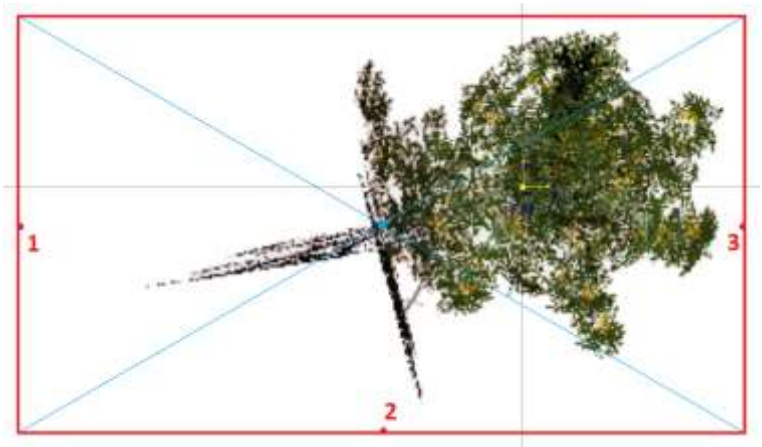

Figure 65. Example of a PreScan model of a tree (Ash 20y) with its bounding box (red line). The center of the bounding box is indicated with a blue dot. The center of gravity is shown as a set of yellow arrows, extended with gray lines.

Taking the center of the bounding box as the ground truth (BBC-mode) may lead other problems. From the PreScan help-file it can be read that the radar sensor used for the measurements (TIS-sensor) generates returns based on the geometry of the object. In case of the tree in Figure 65 most of the sensor returns will therefore be in the right half of the bounding box, depending on the direction of viewing.

The BBC-mode is thus likely to give an error between measurements and ground-truth that is not centered around zero, but has an offset. Taking the COG as the ground-truth leads to the least problems, but one should keep in mind that even with this ground-truth, the error may be quite large, because the radar measures the edges of the actual 3D-model, while the ground-truth is based on the COG of the model.

This is once again showed in Figure 66 with a PreScan model of a car. Due to the size of the object, the difference between the TIS (radar) measurement and the ground truth can be about 2.75 m off if measured from behind (rear bumper). The offset between measurements cannot be prevented, because the current sensors available in PreScan do not offer the options.

For this experiment it means that the value of the error should not directly be interpreted as a tracker performance indicator. Rather it can be used to compare one tracking algorithm with another. If both trackers operate with a similar accuracy, the values of the error between track and ground truth should be the same.

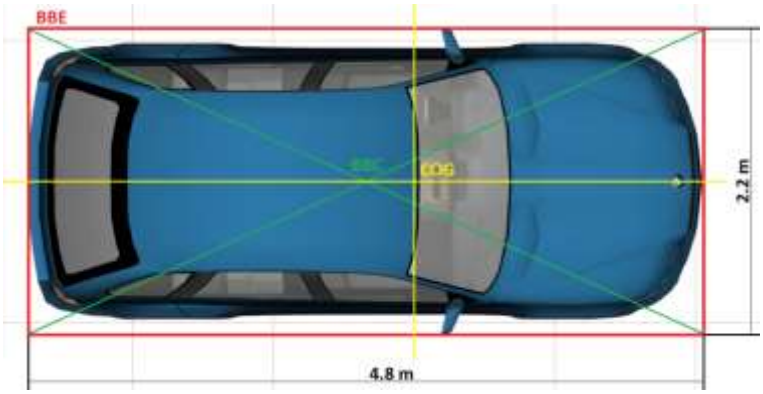

Figure 66. Example of a PreScan model of a car (BMW X5) with its bounding box (red line). The center of the bounding box is indicated with green lines. The center of gravity is shown as a set of yellow lines.

Both the *radar* data and the ground-truth data are converted from range/angle into World coordinates, because the tracker operates on these coordinates as well, as discussed in Section 2.2.4. To achieve this the position and heading of the ego-vehicle are used. In the real vehicle this data would come from the GPS and inertial correction unit, which have a limited accuracy, but in this research the data is taken from PreScan and no noise is added to these values.

In the experiment a recorded GPS trace is used to create a trajectory in the PreScan environment. This recorded data off course has the same accuracy as the GPS-receiver on the car, but in the experiment the GPS trace is assumed to be the ground truth ego vehicle motion and no noise is added.

All sensors in the scenario run at a frequency of 20 Hz, which therefore requires an algorithm processing time of less than 0.05 seconds for it to be applicable without any loss of frames.

5.1.2 Environment

The recording of the data took place in an urban area in Delft, The Netherlands and is shown in Figure 67. The red line indicates the route the test vehicle has taken (left to right in the figure) and the yellow dots represent the roads that are also generated in the PreScan environment. From this top-view some challenges can already be seen; there are sharp corners, smooth bends and lots of trees next to the road.

Figure 67. Map of the scenario used to evaluate the tracker performances. Red line: route followed, yellow dots: other roads in simulation. Map taken from Google Maps [58].

All the stationary objects are also generated in the PreScan environment, their placement based on the map (Figure 67) and recorded video data. The objects are placed in such a way that they represent the real-life situation as good as possible. Slight variations between the simulated position and true position can occur, but both the radar measurements and ground truth are simulated inside the PreScan scenario.

In the PreScan environment only a limited number of buildings is available and since they are only for cosmetic purposes, no custom buildings were imported. The buildings are not the same as those in the real World, but whenever a 2-store house was present in real-life, a 2-store house is present in the PreScan scenario. The resemblance between the real World and the PreScan environment is visible in Figure 68 and Figure 69.

The scenario starts out on a main urban road, going around a shopping center, with trees on either side of the road (Figure 68). Between the trees on the outside of the corner a side-road is visible and several bikes are simulated driving on this road. On the road in front several cars are simulated to represent a normal traffic flow. The speed at this first section around the long left-hand bend is between 20 and 30 km/h.

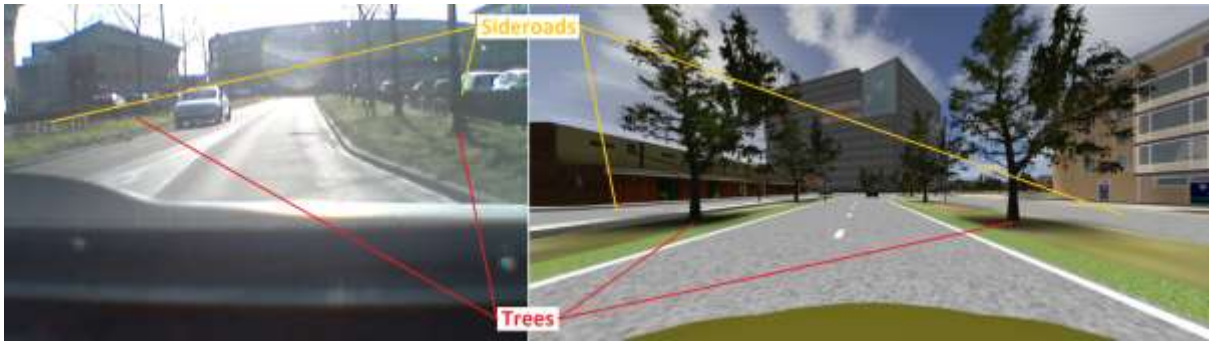

Figure 68. Comparison of real-life video footage (left) and the PreScan simulated environment (right) in the first part of the scenario. Trees and side-roads are visible in both screen-captures, as well as some traffic.

In the latter part of the scenario the scene changes to a more domestic area, with houses close to the road, parked cars, and sidewalks (Figure 69). The speed is lower; around 15-20 km/h and the trees are not as close to the road as in the previous part. At the very bottom of Figure 67 there is a small forest about 10 meters from the road. This forest is within the FOV of the radar, so many measurements are expected in this area that are not part of the traffic and pose no threat.

In this situation it is expected that the proposed region-tracker will perform well, since the forest is outside of the infrastructure-region and will thus be tracked with a faster nearest neighbor tracker. Both the JPDA algorithm with and without clustering are expected to have many hypotheses and therefore may run significantly slower.

Figure 69. Comparison of real-life video footage (left) and the PreScan simulated environment (right) in the latter part of the scenario. A more domesticated environment with parked cars and grouped trees on the right side at the end of the street.

The scenario is taken from a longer dataset, making the first point in the radar-data not at $t = 0$, but at $t = 168.2\ s$. The recording of radar data is 95 seconds long and ends at $t = 263.2\ s$. This has no consequences on the performance of any algorithm, it is just an explanation on why the simulation time does not start at zero.[2]

[2] A video of the scenario is available on request by email: rstakelbeek@gmail.com

5.1.3 Tracker parameters

All three trackers that are used in the experiments share a Kalman filter at their basis. The parameters for this filter are the same regardless of the tracker. In this experiment a constant acceleration motion model is used and only the position is measured. As stated before, the tracking takes place in a Cartesian World-frame, in this case the PreScan coordinate system. In the actual vehicle the UTM coordinate frame may be used, which shares the Cartesian properties while it is well defined for the entire Earth.

The measurement noise matrix for the filter is based on the specifications of the Continental ARS-300 radar [48]. Since the specifications state the accuracy in range and angle, a conversion to Cartesian coordinates is required. For simplicity, the accuracy for both the x- and y-direction is set to the largest sigma coming from the conversion. The noise level estimations therefore represent the worst-case scenario and the measurements are expected to be slightly more accurate than the estimation.

Modeling the process noise is based on the expected maximum acceleration as described in [59]. The assumption was made that an average driver will generate up to a maximum of 0.3 g of cornering acceleration in an urban area. Since this experiment is conducted in a Cartesian frame, this acceleration is also expected to be the maximum acceleration along either coordinate-axis.

$$x_{k+1} = A \cdot x_k + \Gamma \cdot \mathrm{v_k}$$

$$A = \begin{bmatrix} 1 & 0 & dt & 0 & 1/2 \cdot dt^2 & 0 \\ 0 & 1 & 0 & dt & 0 & 1/2 \cdot dt^2 \\ 0 & 0 & 1 & 0 & dt & 0 \\ 0 & 0 & 0 & 1 & 0 & dt \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{26}$$

$$\Gamma = [1/2 \cdot dt^2 \quad 1/2 \cdot dt^2 \quad dt \quad dt \quad 1 \quad 1]'$$

The state equation used is as shown in Equation (26), where the last term can be exchanged for the process noise matrix Q. The process is demonstrated in [59] and Q can be computed using the formula in Equation (27). As a guideline the estimation of $\Delta\sigma_a$ is said [59] to be between 0.5 and 1 times the maximum expected acceleration increment per time step (Δa_M). With an estimated acceleration of 0.3 g, $\Delta\sigma_a$ becomes approximately 0.15 m/s^2.

$$Q = \Gamma \cdot \sigma_a^2 \cdot \Gamma'$$
$$0.5 \cdot \Delta a_M \leq \sigma_a \leq \Delta a_M \tag{27}$$

Aside from the noise on the process and the measurements, several false measurements are created which could be radar reflections, or other anomalies that lead to measurements that do not represent real objects. A random number of these false measurements between 0 and 5 is generated for every frame. All false measurements have a random range and bearing and are processed in the same way as normal measurements. For the tracking algorithms they are indistinguishable from the other measurements.

In theory a false measurement will be initialized as a track at the end of the frame, but since no new measurement is expected near the false alarm in the next time-step, the track will degrade and be deleted shortly after it has been created. The track will not be confirmed and is thus of no consequence to any decision-making algorithm further up the controller-chain.

The size of the confidence ellipses in the JPDA algorithms has been set to 4σ. Though this is a large number, initial tests indicated this gave better results than a more common 3σ gate size. A reason may be unaccounted for measurement errors within the sensor models in the PreScan software.

For the proposed region-tracker, the *view-distances* have been set to 50 m forward and 30 m rearwards. A longer forward range would make little sense, since the range of the used radar sensor is 60 m. The range is close to this limit, because dangerous situations should be detected at an early stage. Assuming vehicles driving 8-10 m/s, the maximum closing speed between two vehicles in opposing direction is 20 m/s. With a 50 m forward view, this leave 2.5 s to detect an object, evaluate its path and respond to it accordingly.

The rearward view can be less, since the closing speed will be less as well. Normal traffic will drive in the same direction as the ego vehicle, or it will have a trajectory away from it. When the ego vehicle is stopped, there are 3 seconds to evaluate an object entering the region at 10 m/s and respond to its behavior.

One should notice that only one radar is simulated at the front bumper of the vehicle. The rearward region will thus always be void of measurements. However; the region is constructed in this way to make sure a realistic measure of the overhead of the algorithm can be estimated, since the algorithm is intended for vehicles with a 360° FOV.

5.1.4 Testing the limit

The scenario described in Section 5.1.2 is a good indicator for an average urban scenario. To signify a situation where the region-tracker should have an even greater effect on computation times, a second scenario is constructed. The scenario is based on a real-life observation of radar data at an intersection. Figure 70 shows a T-shaped intersection with trees, bushes and tall grass within the FOV of the radar while driving up to the intersection.

Figure 70. T-shaped intersection with tall grass and bushes in the FOV of the radar while driving towards the intersection. This scenarios gives lots of cluttered measurements. Image from Google Streetview [60].

The radar sensor picks up a lot of clutter from the tall grass, with many measurements very close together. The internal pre-filtering of the radar does not recognize this as false measurements, because there are so many returns close together.

It is expected that the region-tracker will perform well in this situation, because all the measurements outside of the infrastructure are not tracked with a slow and thorough JPDA algorithms, but with a simple, much faster nearest neighbor Kalman filter. A JPDA algorithm without clustering on the complete surroundings of the car is expected to run out of memory very quickly and the clustering-tracker is likely to return a large cluster with many objects, leading to slow processing times as well.

Since the scenario is especially suited to the region-tracker, the performance gain is also expected to be rather large. It is important to keep in mind that this is a situation that is possible in real-life, but not a scenario that will occur all the time. The expected performance gains should be seen in this perspective as well.

Unfortunately the intersection shown in Figure 70 is not available in the current eHorizon dataset, so a similar scenario is generated which does not have an intersection, rather a 90° sharp turn with simulated trees and bushes along the outside (Figure 71). To demonstrate similar situation, a parking lot with walking people is added on the right side of the road, before entering the corner.

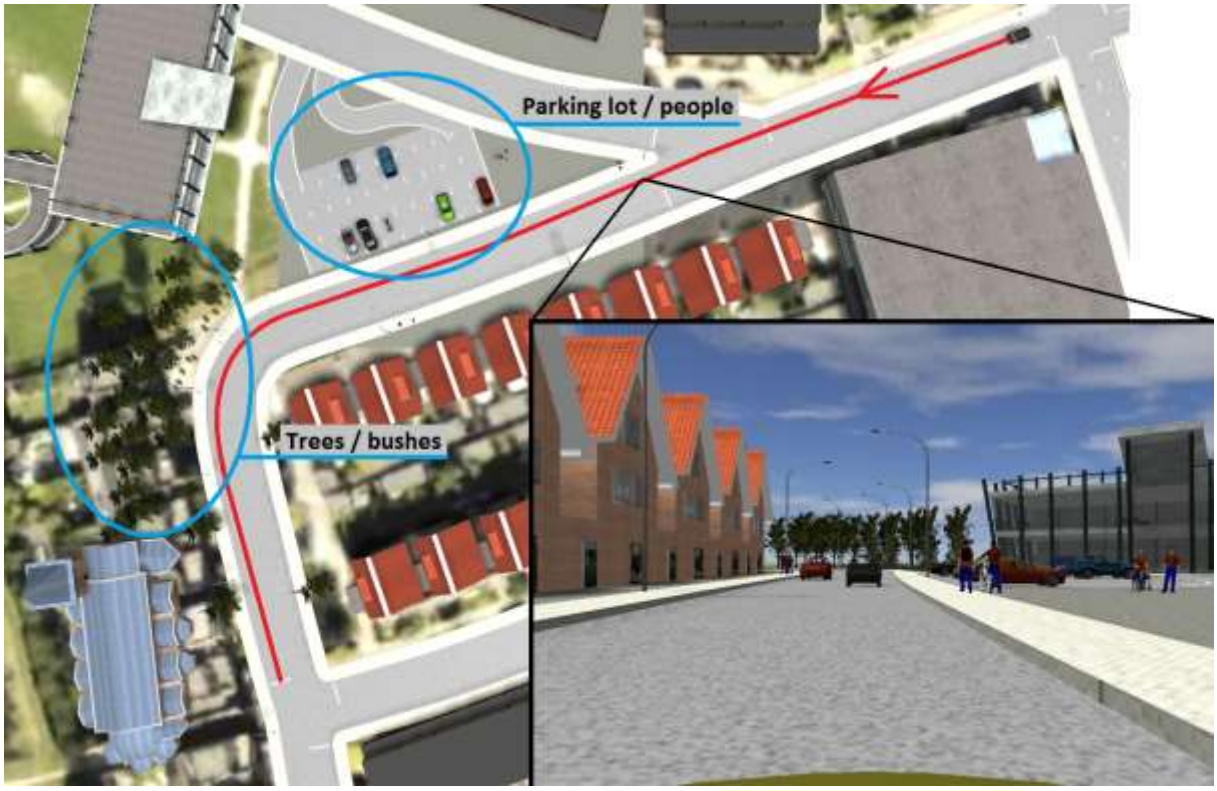

Figure 71. Top view and forward view (inset) of the scenario to test the limits of the algorithms. The red line is the route taken in the simulation, a parking lot and the trees and bushes are circled in blue.

The scenario starts with all the nature elements outside of the FOV of the radar and ends when the vehicle has passed the turn, totaling approximately 30 seconds of simulation time. The parameters for the trackers are kept the same as with the previous scenario, so the only change is the input data (measurements and eHorizon).[3]

5.1.5 Border crossing

A third scenario is created to show the performance of the region-tracker when an object continuously goes in and out of the region. For this purpose the vehicle is put at a fixed location on the road and objects are generated that move along a sinusoidal path along the edge of the region. For the experiment several pedestrian are simulated following a path with a constant velocity. One pedestrian is generated using the PreScan software, while several others are simulated with MATLAB. The pedestrians occur on either side of the region and travel in both forward and backwards direction. The amplitude and period of the sinusoidal paths are different for all the objects.

[3] A video of the scenario is available on request by email: rstakelbeek@gmail.com

Figure 72. Top view of the third scenario with the paths of the pedestrians. the region boundary is visible in red.

In Figure 72 a top view of the scenario is shown, with the paths of the pedestrians in several colors. The region boundary is visible in red and the FOV of the radar is indicated with a yellow line. Table 10 specifies the properties of the paths taken by the pedestrians, including the color in the figure. The experiment runs for 24 seconds, since most objects are out of range by that time.

Object	Color	Amplitude	Period	Speed	Direction
PreScan 1	Magenta	Custom	Custom	3 m/s	Towards
MATLAB 1	Green	2 m	15 m	3 m/s	Towards
MATLAB 2	Black	0.75 m	8 m	2 m/s	Towards
MATLAB 3	Blue	1 m	10 m	2 m/s	Away from
MATLAB 4	Orange	0.8 m	8 m	2 m/s	Away from

Table 10. Parameters for the pedestrian movement.

5.2 Simulation results

In this section the results of the scenarios are presented and discussed, but before going into the specific results (Sections 5.2.2 - 5.2.4), some general remarks are made on the results in Section 5.2.1. All the scenarios are run on the same computer, a Dell OptiPlex 790 with an Intel® Core™ i5-2400 CPU running at 3.10 GHz and 8 GB of RAM. The simulations are done within MATLAB R2014b on a Microsoft Windows 7 Enterprise 64-bit operating system.

5.2.1 Remark on eHorizon accuracy

While reviewing the first results of the region-construction algorithm it became clear that in some areas the regions does not follow the infrastructure as displayed in the PreScan environment. The first thought was that the simulated environment is not representing the true World accurate enough, but it turns out to be a problem in the maps of the eHorizon system, as shall be explained in this section and in detail in Appendix C.

There are two main parts of real recorded data that are used in this research: the recorded GPS-trace of the vehicle and the corresponding output of the eHorizon system, according to the ADASIS protocol. Although the accuracy of the GPS-trace recorded is not yet on the scale of centimeters or better, one can see that the inaccuracies are at least consistent. In other words; there may be some offset between the real position and the reported one, but a recorded GPS-trace is a smooth line, following the road. The GPS-trace does not show a ragged behavior with a positive error in one time-step and a negative error in the next. This leads to the assumption that the GPS-track shows only an offset error, not an independent error over time.

The recorded GPS-trace is imported into the PreScan software, just as the road-network is. The road-network is based on (navigation-) maps provided by *Open Street Maps* (OSM) [61] and are slightly modified to cope with OSM/PreScan-inconsistencies. The bulk placement of the roads is based on the maps provided by OSM. When looking at the GPS-trace and the imported road-network, it is visible that most of the GPS-trace is placed inside the visible road boundaries. Sometimes, especially in sharp corners, the trace *cuts* the corner a little, but overall it follows the lanes quite well.

Based on this observation, an assumption can be made that the accuracies of the GPS-trace and of the OSM map are good enough (close enough to each other) to ensure that regions can be constructed. However, looking at one frame of the scenario with the region as constructed by the algorithm plotted on top (Figure 73), one can see great discrepancies between the visible map (PreScan/OSM) and the region borders (red line).

Note that the error is not only in the offset, which is visible in the left image in Figure 73, but also in the heading of the road sections. Compare the blue and red lines in the right image, which represent the center of the roads as defined by the OSM map and the eHorizon system respectively. The heading can only be wrong if some of the definition-points of the map are significantly different between the two maps.

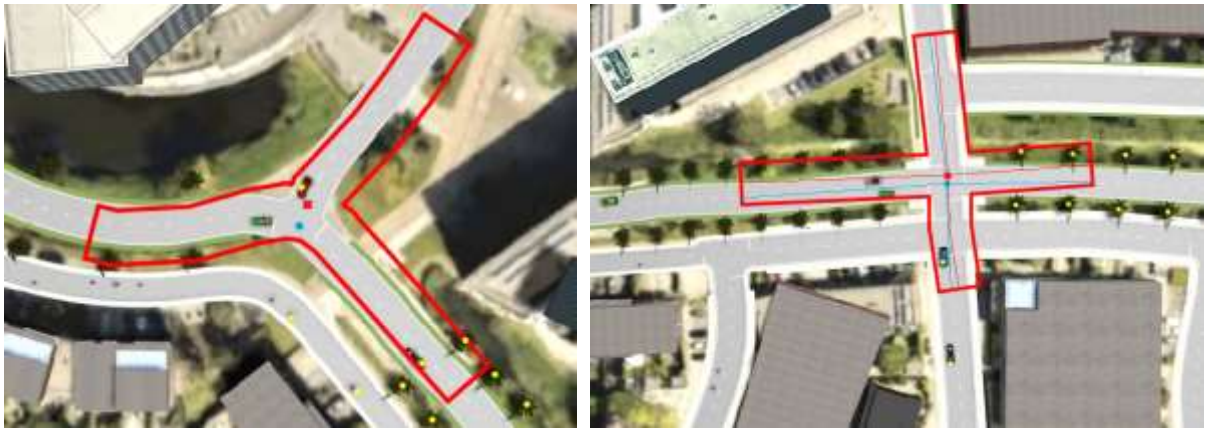

Figure 73. Discrepancies between OSM/PreScan maps (underlay) and the data recorded from the eHorizon system that is used to generate the red borders. Especially in the right image it is clear that there is not only an offset error, but also an error in road heading (compare red and blue lines).

The procedure described above leads to the conclusion that the main part of this error is within the eHorizon system. To understand where exactly the error comes from, the eHorizon system must be examined a bit closer. The input of the eHorizon system is the GPS-position and heading of the vehicle. In this experiment, this is the exact same data as the recorded GPS-trace that is imported into the PreScan environment.

The output of the eHorizon system is a list of messages on the CAN-bus of the vehicle. These messages are carefully processed and reformatted by some algorithms. What is important is the structure of the output data: The road is described by sections that are in turn defined by points in *GPS-coordinates*. This means that, as far as the road definitions go, the output of the eHorizon system is fixed in *space*. The only time-variant element is: the current position from the GPS-input is used to limit the output to the near surroundings only.

Completing the reasoning, based on all the information above, it can be assumed that the map used inside the eHorizon system has some large discrepancies with the real World. The definition of some of the points on the road is far from the true location. In other words; the assumption made in Section 4.2.2 about the eHorizon mapping accuracy (20 cm accuracy) is at the very least too optimistic.

Although the accuracy is not nearly enough to ensure perfect workings of the region-tracking algorithm, no changes are made to the parameters of the region constructor yet. It is expected to still see a gain in performance, even with a less than desirable map accuracy. The true potential of the proposed region-tracker should be shown in the second scenario (Section 5.2.3).

After contact with the manufacturer of the eHorizon system used in the vehicle it became clear that the assumption made about the stubs, or nodes is wrong. These points are *not* always on the center line of the infrastructure, but are sometimes used as support points for a more complex description of the road geometry. For this research the data is used as it is and interpreted as if all points represent the center of the roads.

5.2.2 Scenario 1 – Results

The main metric to compare the methods is the processing time of the algorithms; if an algorithm takes longer than the time between measurements to calculate a frame, it cannot keep up and will eventually fail. In the experiment the measuring frequency is 20 Hz, although the Continental radar actually works at about 15 Hz. A processing time of 0.05 s would mean the algorithm can work in real-time in this scenario and a cycle-time of 0.067 s means it can keep up with the real radar measurements.

Note that in this section the first algorithm is called the *standard JPDA* algorithm. In the context of this thesis it means a normal JPDA algorithm is used, with no extensions or enhancements added to it. In many applications some form of clustering is used to make sure the algorithm can run sufficiently fast. This is the reason that the clustering JPDA (JPDA + CDA, section 2.5.1) is added to the comparison.

In Figure 74 the processing times of the three algorithms of the first experiment are shown. First not that a logarithmic scale is used in the y-direction, to make sure all the data fits in a reasonable sized graph. Secondly; the data, especially the JPDA (red) is rather *spiky*. To make the graphs in the results more easily interpretable, it was decided to use a 5-sample moving average filter. This will keep the most important information from the data, while the graph looks much smoother and is easier to interpret. The result of the filtered data for the processing times is shown in Figure 75. In the results section many of the graphs will be generated with smoothened data; if this is the case, it will always be displayed in the top-left corner of the figure and in the caption of the figure.

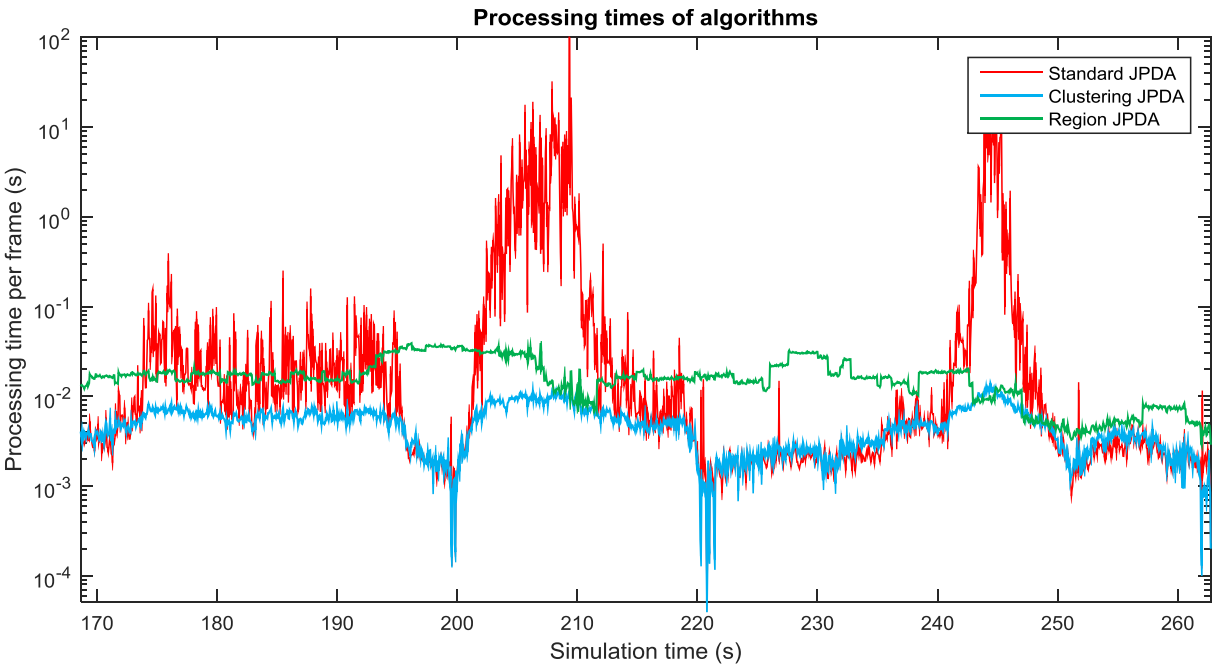

Figure 74. Processing times of the three tested algorithms. Note that the y-axis has a logarithmic scale.

Figure 75 shows the processing times during the entire scenario for all three algorithms. Note that the scale on the y-axis is logarithmic. The graph displayed below is generated by using a moving average over 5 samples. The maximum processing time of the JPDA algorithm without clustering is 1083.55 s at a simulation time of 209.35 s. For the JPDA processing time there are 250 data-points out of the 1904 that exceed 0.1 s. So even if the algorithm were to run at only 10 Hz, approximately

13% of the time the algorithm would give delayed results, which could lead to dangerous situations on the road.

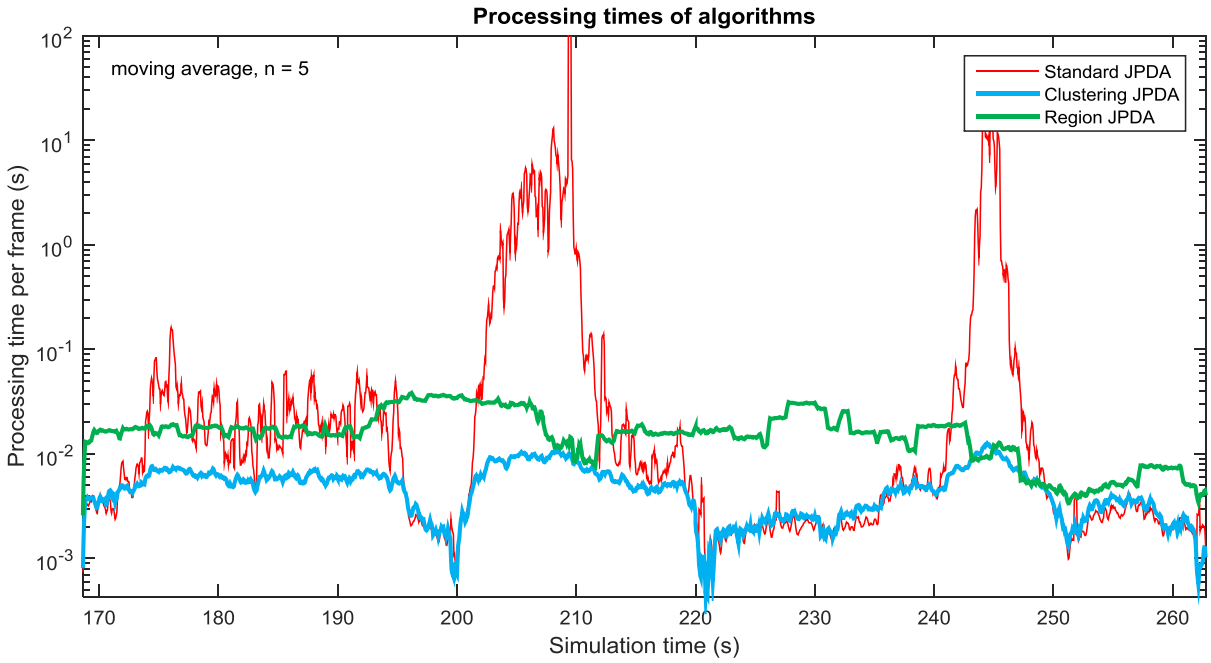

Figure 75. Processing times of the three tested algorithms, a moving average of 5 frames is used to smoothen the data to make it more presentable. Note that the y-axis has a logarithmic scale.

It can also be observed that the region tracking algorithm is generally slower than the clustering JPDA, which is due to the higher overhead of the former algorithm because of the computations of the regions. In Figure 76 this is demonstrated by plotting the total processing time (green) and the time it takes to generate the regions (black) in one graph. The black line always is close to the green one, indicating that the major part of processing time is spent on constructing the regions.

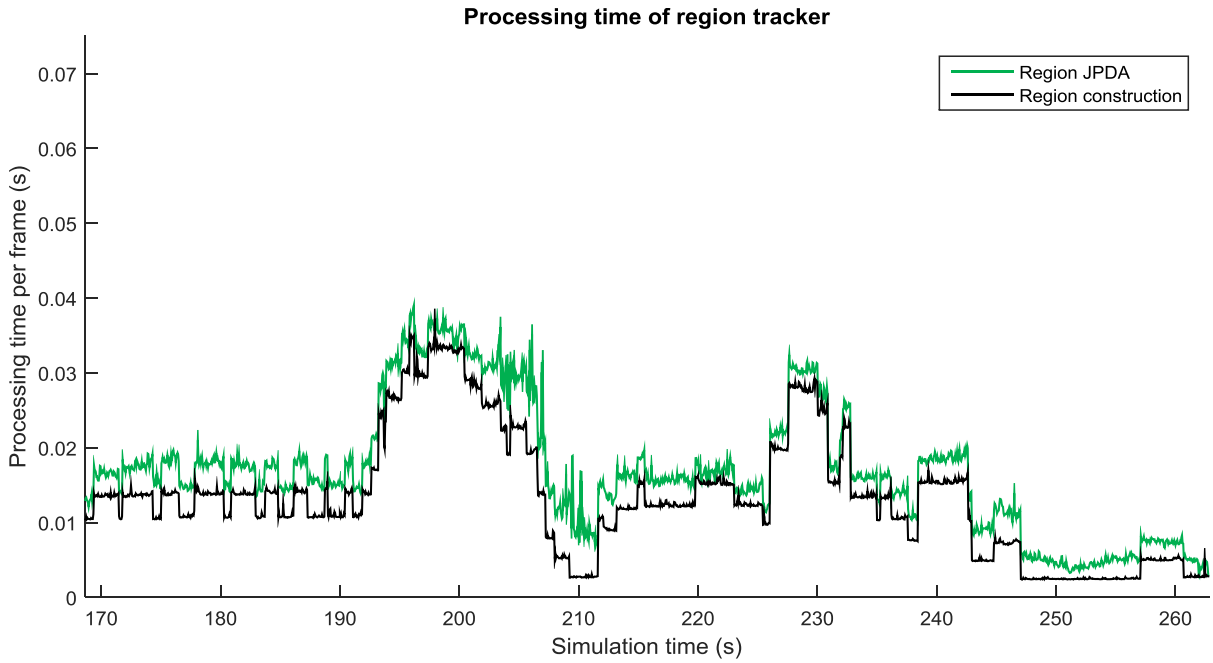

Figure 76. Processing times of the region tracker (green) and the time taken for region construction (black). A large portion of the processing time is due to the region construction part of the algorithm.

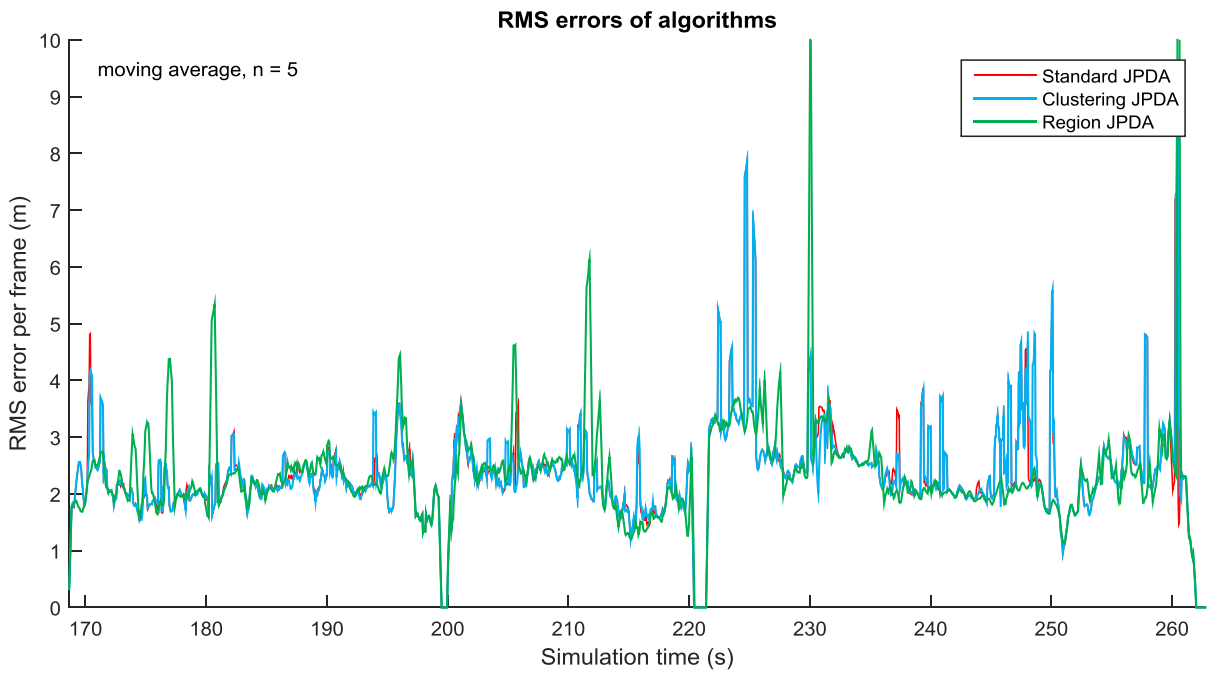

Figure 77. RMS-errors of the three tested algorithms, a moving average of 5 frames is used to smoothen the data to make it more presentable. All three algorithms show a very similar error.

The graph in Figure 77 shows the RMS-error for all the algorithms. For every track the error between the position given by the track is compared against the ground truth of the object. The RMS value of all errors in one time-step is taken, leading to a single value for every algorithm in every time-step. The errors for all the algorithms are very similar during the entire experiment, although some peaks occur.

In Section 2.4 it was shown that the number of hypotheses is an important indicator of the processing time of the algorithms. To this end, the number of hypotheses per time-step is given for all algorithms in Figure 78. The JPDA algorithm has to evaluate up to 737.280 hypotheses at $t = 244.05\ s$.

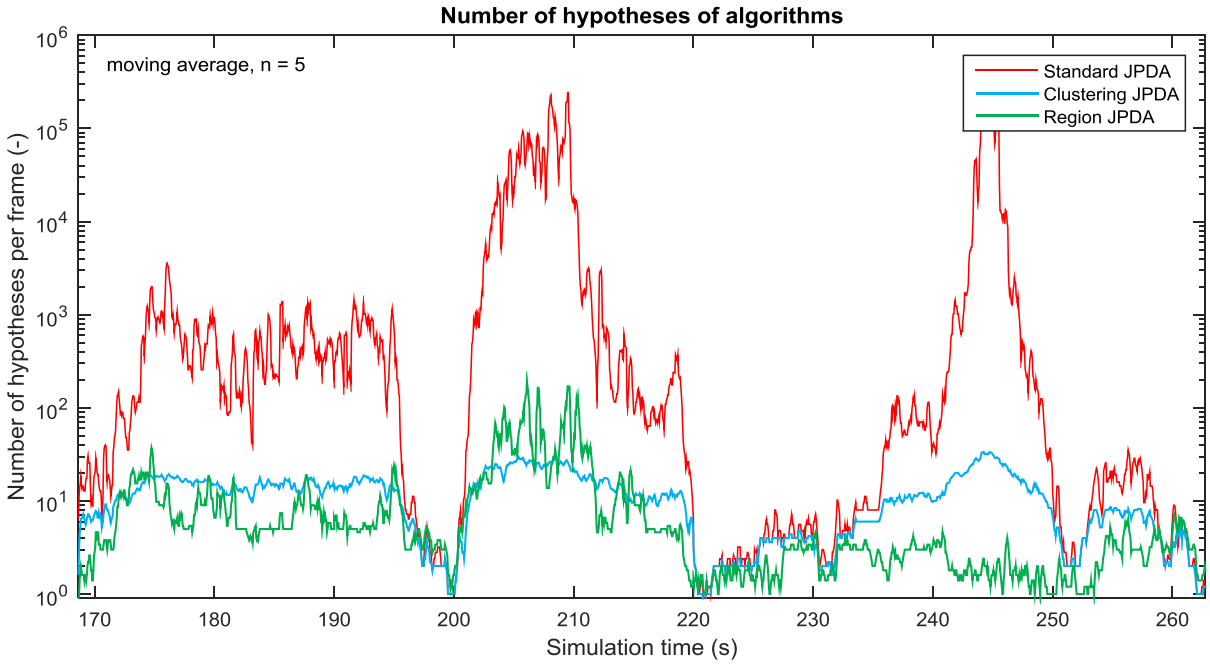

Figure 78. Number of hypotheses for every time-step and every algorithm, a moving average of 5 frames is used to smoothen the data to make it more presentable. Note that the y-axis has a logarithmic scale.

The previously displayed graphs give detailed information about the performance of the algorithms at every time step of the experiment, but Table 11 takes the data and compresses into a single value by taking either the average, or the maximum values during the experiment. This can give a more recognizable indicator of the performance of the algorithms

Metric	Unit	JPDA	Clustering JPDA	Region JPDA
Average processing time	s	1.022	0.00490	0.0160
Maximum processing time	s	1083	0.0139	0.0380
Average RMS error	m	2.313	2.318	2.315
Average # hypotheses	-	8179	11.4	5.7
Maximum # hypotheses	-	737280	38	129

Table 11. Tracker performance compressed into single values. This gives a comprehensible indication of the general performance.

From Table 11 it is clear that the JPDA algorithm without clustering takes much longer than both the clustering, and the region tracker. At the points where many objects are within the radar range, the maximum processing time peaks at as much as 1083 s. Both the clustering and region trackers are able to process the data much faster, with the clustering tracker slightly faster than the region tracker.

The RMS errors are very similar, which was already demonstrated in the graph of Figure 77. The average RMS error differs only by half a centimeter among the three algorithms. However, when looking at the number of hypotheses, there is a similar distinction as seen with the processing times.

The JPDA without clustering has a high maximum and average and the clustering tracker shows a slightly lower number of hypotheses than the region tracker.

5.2.3 Scenario 2 – Results

In this scenario many objects are just outside the infrastructure boundaries, but close enough to be within the radar range. Especially from $t = 247\ s$ onwards, the number of measurements is high, because of the many trees and bushes just outside of the road.

Although the difference is not as clear as in the previous scenario with the JPDA algorithm without clustering, Figure 79 shows that the clustering tracker takes much more time than the region tracker whenever the number of measurements gets high. The processing time of the region tracking algorithm is much more stable.

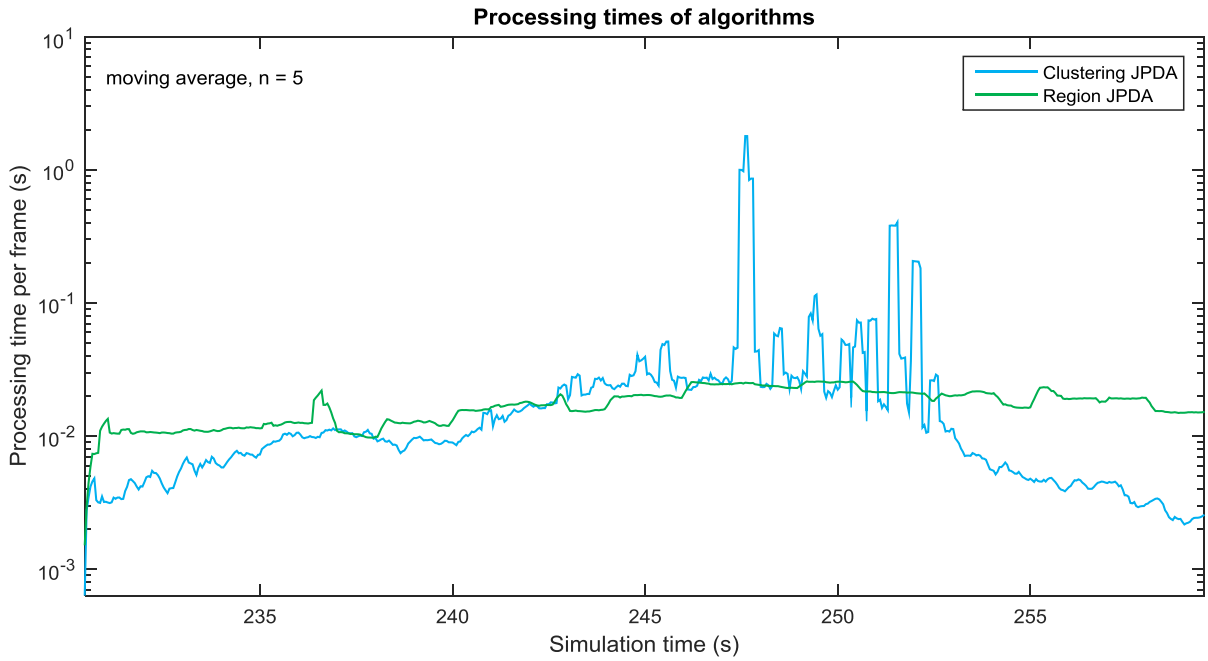

Figure 79. Processing times of the two tested algorithms, a moving average of 5 frames is used to smoothen the data to make it more presentable. Note that the y-axis has a logarithmic scale.

The reason for this stable behavior is not because there are less objects tracked, as can be seen in Figure 80: Both algorithms have a similar number of tracks in their memory at any point in time during the experiment.

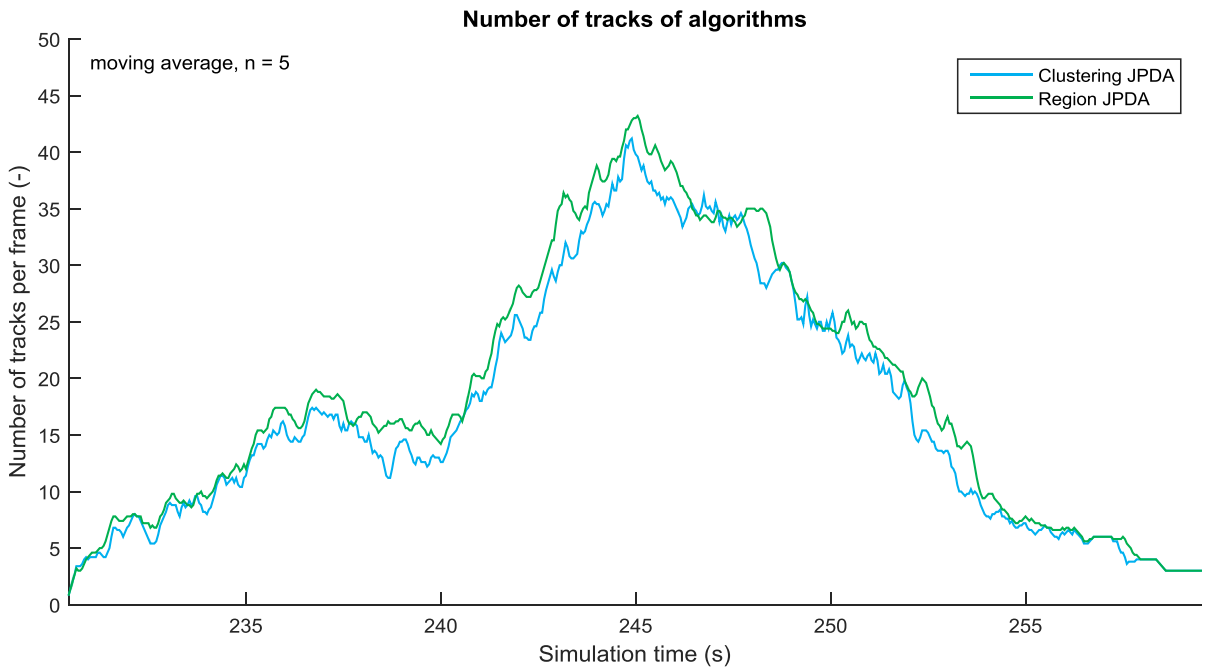

Figure 80. Number of tracks of the two tested algorithms, a moving average of 5 frames is used to smoothen the data to make it more presentable.

However, the number of hypotheses is much higher for the clustering algorithm than it is for the region tracker, especially just before the corner, when the trees are within radar range. The low number of hypotheses is due to the nearest neighbor tracker, used in the outer region, which results in a single hypothesis for all the tracks in that region.

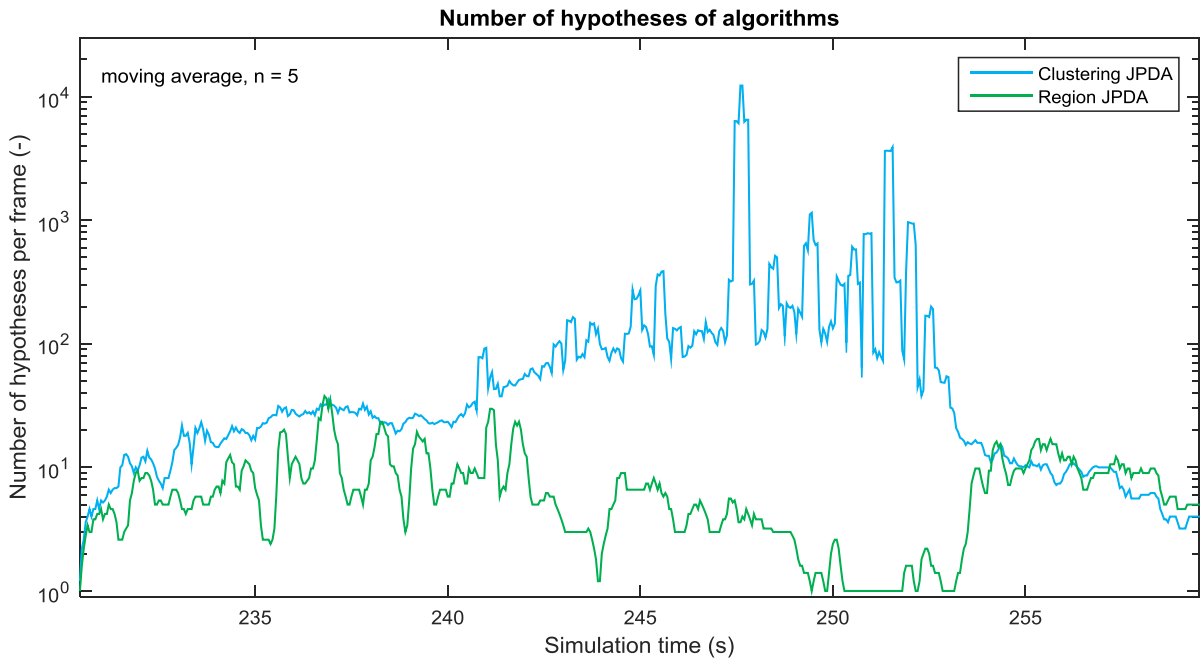

Figure 81. Number of hypotheses of the two tested algorithms, a moving average of 5 frames is used to smoothen the data to make it more presentable. Note that the y-axis has a logarithmic scale.

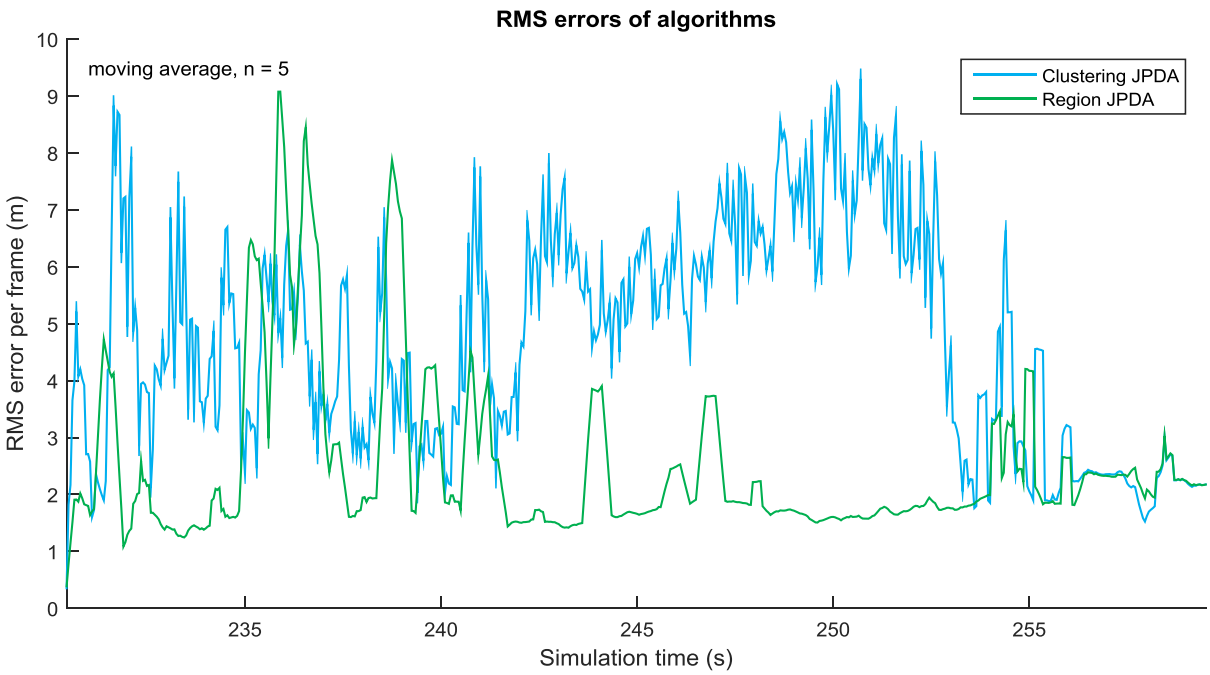

Figure 82. RMS-error with the ground truth of the two tested algorithms, a moving average of 5 frames is used to smoothen the data to make it more presentable.

When looking at the RMS-errors during the experiment, it becomes clear that the clustering algorithm loses track of objects in the cluttered part of the simulation. Errors of around six meters would not occur in a less cluttered environment, as was demonstrated in the first experiment (Section 5.2.2).

As with the first experiment, the results of the second scenario are also presented as compressed numbers in Table 12. For all the metrics presented, the clustering algorithm performs less than the region tracking algorithm; There are longer processing times, more hypotheses and the RMS-error is higher.

Metric	Unit	Clustering JPDA	Region JPDA
Average processing time	s	0.0359	0.0173
Maximum processing time	s	4.78	0.0422
Average RMS error	m	4.67	2.51
Average # hypotheses	-	220	7.1
Maximum # hypotheses	-	31032	73

Table 12. Tracker performance compressed into single values. This gives a comprehensible indication of the general performance.

5.2.4 Scenario 3 – Results

In the final experiment the vehicle is stationary and several pedestrians move in sinusoidal paths along the boundaries of the region. This experiment is conducted to show the performance of the tracking algorithm while objects cross the border of the region. Remember that the objects inside the region are tracked with a JPDA algorithm, while those outside are filtered with a nearest neighbor algorithm.

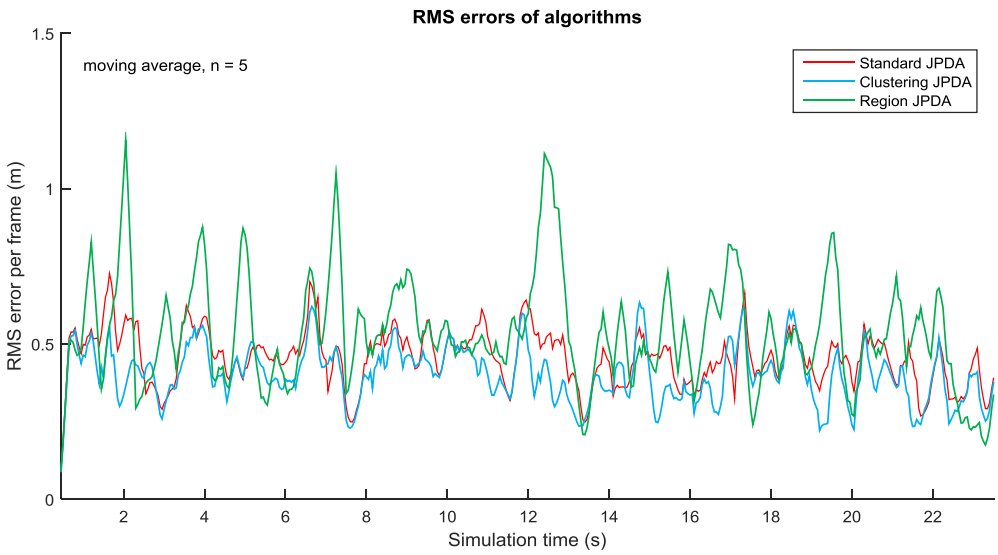

Figure 83. RMS-error with the ground truth of the three tested algorithms, a moving average of 5 frames is used to smoothen the data to make it more presentable.

The graph in Figure 83 shows the RMS error for every frame between the tracks and the ground truth position of the objects. Although there are some small differences in the spikes, one can see that the errors are very similar among the algorithms. An interesting point to take from this figure is the size of the error, which is much smaller than in the previous two scenarios.

Whereas in the previous experiments many of the objects such as trees and cars are rather large, in this experiment the PreScan object is a pedestrian, with a much smaller bounding box. The MATLAB generated objects do not even have a size at all, since only the location is simulated. The use of the PreScan software leads to measurements that are closer to the center of gravity when objects are smaller. What is important is that all the trackers perform similarly when the same data is provided.

Metric	Unit	JPDA	Clustering JPDA	Region JPDA
Average processing time	ms	2.17	3.34	4.73
Maximum processing time	ms	3.64	4.59	5.84
Average RMS error	m	0.45	0.40	0.52

Table 13. Tracker performance compressed into single values. This gives a comprehensible indication of the general performance.

All of the algorithms perform well within the limits of the requirements, as the scenario is not intended to push the trackers to their extremes.

6 Conclusions and discussion

In this section the results of the three experiments will be discussed and conclusions about the performance will be formed. Section 6.1 contains the discussion of the experiment results while Section 6.2 summarizes the conclusions that are drawn from the discussion. Finally, Section 6.3 will go into detail about the lessons learned during this research and recommend areas for further investigation, and parts that show potential for improvements.

6.1 Discussion

In this section the results will be discussed and interesting events are highlighted.

6.1.1 Processing time

The first scenario of the experiments was created to simulate a drive through an urban environment with a realistic number of objects and a normal traffic density. The results are therefore similar to what is expected in the majority time spent in urban scenarios. During the experiment the JPDA algorithm without clustering fails to deliver results on time, with a maximum processing time of over 1000 seconds (Figure 75 and Table 11). The number of objects, especially trees, lead to so many sensor returns that the number of hypotheses grows beyond the capabilities of the algorithm.

Since the data is recorded and simulated at 20 Hz, the cycle time should be no more than 0.05 s to ensure that the tracker can keep up with the data. The JPDA algorithm without clustering fails to deliver here, but both the clustering and region trackers *are* able to process the data fast enough. The clustering algorithm has a cycle time that is faster than the region tracker, which is mostly due to the generation of the regions (Figure 76).

In a normal urban situation the clustering tracker performs approximately 3-4 times faster than the region algorithm, so at first glance this seems to be a better choice. If only the processing time was important this would be true, but the regions that are generated offer much more information than the clusters of the clustering algorithm do. The region can be used to filter out object (as shown in the second experiment) or it can be used by other systems in the autonomous vehicle to define a driving corridor. These purposes should be kept in mind, but are not the topic of this research.

Filtering out objects using the region *is* the topic of this research and is exactly what was demonstrated in the second experiment. In this simulation a large number of objects is generated just outside the infrastructure boundaries; the details are discussed in Section 5.1.4. Since the JPDA algorithm without clustering already failed to perform fast enough in the first scenario, this tracker is left out of the second experiment.

Due to the large number of objects that are close together, the clustering algorithm defines large clusters with many measurements and tracks that are processed using a JPDA algorithm. The region tracker only uses the JPDA algorithm inside the region, so most of the objects are tracked with the very simple nearest neighbor Kalman filter. The large clusters make the clustering algorithm slow, as can be seen from Figure 79 and Table 12. The region tracker manages to stay below 0.05 s with its maximum processing time and is thus able to run all the data before the next batch comes in.

A final test was carried out while processing the data of the first experiment to see the effect of using a clustering JPDA algorithm inside the region instead of the JPDA tracker without clustering. The results of this are not presented in the corresponding section, because no significant difference was detected. The difference in processing times showed the clustering + region tracker to be between 23 % slower and 16 % faster per frame during the course of the experiment. On average the clustering tracker inside the region performs only a fraction of a millisecond better than the JPDA with no clustering in the region.

The main reason to use regions is to limit the number of objects to be tracked inside the region. The low number of objects leads to a relatively fast JPDA algorithm. Although the tracking inside the clustering algorithm is likely to be slightly faster, the overhead for constructing the clusters neglects that benefit when few objects are present. For the current size of the region introducing clustering within the region gives no benefit to the total performance of the tracker.

6.1.2 Performance is more than processing times

Processing times on their own do not show the complete performance. As discussed in Section 2.6; the accuracy of the trackers should be similar, because at the bases all the algorithms are the same. With an average RMS error that is within 1 cm between the three algorithms in the first experiment, this seems to be the case. The size of the RMS error seems to be rather big, but can be explained by the used data (See also Section 5.1.1).

The ground truth data is generated using the center of gravity of the objects, whereas the TIS sensor that simulates the radar calculates the actual return of the object. Especially when objects have an extensive size, such as trees and cars, this can give an offset between the measured data and the ground truth. This is however not a measurement error, rather a discrepancy between the measurement and the definition of the ground truth.

This difference is due to the software package that is used. The AIR sensor for the ground truth only has the options to return the bounding box (edge or center) of the object, or the center of gravity. The choice was made to use the COG, because this is the one closest to the truth and also gives a consistent result (See Section 5.1.1). Another option would have been to use a second TIS sensor and read the data from this sensor without any noise added to it. The problem with that approach is that the TIS is sensitive for occlusion and can therefore not detect any objects that are behind other objects.

Whereas the error differs only slightly in the first experiment, the second shows a greater difference. In this case only the clustering and region trackers are used. The region tracker performs very similarly to the first simulation with an error that is 2.5 m, but the clustering algorithm seems to lose track of objects regularly and with 4.7 m the average error is almost twice as large.

The large error for the clustering tracker shows that there are certain situations where even an accurate algorithm has to concede to its limits. This is the reason to use a region of interest, in this case the proposed method to use the infrastructure boundaries to create this region. Filtering out the less important objects and using a simple but fast algorithm to keep track of their rough movement has a great benefit in this situation.

6.1.3 Boundary crossing

The third experiment was conducted to indicate the performance of the region tracker while objects continuously cross the boundary between two regions. The scenario was run on all the trackers and all of them show very similar results, as was to be expected. The processing times are lowest for the JPDA without clustering, the clustering algorithm has a little overhead (generating the clusters) and comes second, while the region tracker has the most overhead for calculating the regions, but still performs much faster than the 0.05 s required for the 20 Hz data. However, this is the simulation where no problems were expected regarding processing times at all.

The reason to do this experiment was to show the error of the algorithm does not change when an object changes from one region to another. At first glance this does not appear to be the case. The average RMS error for the JPDA algorithms with and without clustering are close together (0.45 and 0.40 m respectively), but the region tracker has an average error of 0.52 m.

To see where this increase in error is coming from, the results from the experiment are studied in greater detail: The error is calculated for objects that are within the region, and outside of the region separately. Since different trackers are used for each region (JPDA inside, NNKF outside), there may also be differences in the errors. Table 14 shows that this is indeed the case; the objects inside the region are tracked with a very similar position error as with the standard algorithms. The objects have a larger error when the nearest neighbor approach is used in the outer region.

	JPDA	**Clustering JPDA**	**Region JPDA** *combined*	**Region JPDA** *inside region*	**Region JPDA** *outside region*
Error (m)	0.45	0.40	0.52	0.39	0.61

Table 14. RMS-errors calculated separately per region for the region tracking algorithm. Other errors are for reference.

6.2 Conclusions

After the discussion of the results in the previous section, this section will summarize the conclusions that are drawn from the experiments. All conclusions are written down as a clear quote, the paragraph following will explain why this conclusion is drawn.

> *The region tracker is faster than a JPDA algorithm without clustering in normal urban traffic*

From the first experiment it is clear that the JPDA algorithm without clustering takes much too long to process the measurements and cannot realistically be implemented on an autonomous vehicle without any filtering of the input data. The region tracker is shown to perform well enough in this simulation to be able to process the data before the next batch of measurements is available.

> *Using clustering JPDA inside the region tracker shows no improvement in this setting*

As briefly discussed in Section 6.1.1, the low number of objects inside the region (by design) means that there is little to no benefit to use a clustering algorithm inside the region instead of the JPDA without clustering. This applies to the current parameters for the algorithm; if the region is chosen to be larger and contains more objects, this may change the outcome.

> *The clustering JPDA fails to perform with large clusters, which can occur at large distance or outside the road boundaries, the region tracker can prevent those from being tracked with a detailed algorithm*

The second experiment is used to demonstrate this fact. Based on a real-life situation, many objects outside of the infrastructure, but close to each other lead to large clusters and consequently high processing times for the clustering algorithm. The region tracker *filters out* these objects and only tracks them with a fast nearest neighbor Kalman filter, leading to only slightly slower processing times than with the standard traffic scenario.

Large clusters can also occur at greater distances from the vehicle, because measurements may become less accurate. Less accurate measurements lead to larger gates and a greater likelihood of tracks and measurements forming a cluster. In this situation the range limit of the region will also disregard the clusters and instead track those objects with a nearest neighbor Kalman filter.

> *Using regions to divide the tracking has no influence on the tracker RMS error*

In all scenarios the average RMS errors for the region tracking algorithm are in line with those of the JPDA algorithm. In the final scenario the error is slightly larger, but it is demonstrated that this is only the case for the (less important) objects that are outside of the closest region. Any objects tracked within the region are tracked with the same accuracy as a normal JPDA algorithm tracking the entire survey area.

> *Improvements can be made in the construction of the regions by implementing the provided data more accurately*

As discussed in Appendix C the interpretation of the eHorizon data was wrong. The assumption was made that the stubs given by the system were on the middle of the road and could be joined by straight lines. After discussing this with the manufacturer of the system, it turns out to be a wrong assumption. The nodes are *not* always on the center of the infrastructure, and they cannot always be

joined by straight lines. Implementing a better road-reconstructor should lead to a better approximation and therefore more accurate region definitions.

> *Clustering JPDA is faster than the region tracker, but constructed regions may be useful elsewhere in the vehicle systems*

In the first and third experiment realistic situations are simulated and in both instances the clustering algorithm is faster than the region tracker; 3-4 times faster in the normal traffic scenario, approximately 20-30 % faster in the third experiment. The overhead to calculate the regions is larger than the extra processing time needed to compute the clusters. The constructed regions can however be used for other task or decision making by the autonomous vehicle, where the clusters are not useful for other tasks. Such task could include generating a driving corridor for the path planning algorithm.

The conclusions drawn so far will help in answering the main research questions in this thesis: "*How can a novel method using extended maps be implemented for region splitting in an autonomous vehicle?*". The theory to this was explained in Chapter 4 and the resulting algorithm was tested in three experiments. The results of these experiments show that the implementation of a region constructor based on the eHorizon system, using the assumption of linear road sections and polygons to describe the region boundaries is feasible and better than using a JPDA algorithm without clustering. When large cluster occur, such as at large distances or with many objects close together, but outside the infrastructure, the region tracker performs better than a clustering JPDA algorithm.

> *The proposed region tracker using an extended map system performs faster compared to a JPDA algorithm without clustering and shows no performance drop in a densely cluttered environment, where clustering JPDA algorithm fails due to large clusters.*

The proposed region tracking algorithms showed no signs of performance drop in any of the scenarios tested. In most normal traffic situations, the clustering JPDA will perform faster, but pushing the limits on this algorithm made it fail, where the region tracker kept up its processing speed. Using regions shaped along the infrastructure boundaries is a good way to filter out important objects from the less important ones that are further away and not located on the road.

6.3 Recommendations on future work

The conclusions from Section 6.2 indicate a substantial gain in performance is obtained by using regions to split the tracking into several sub-processes. However, there is always room for improvements which will be discussed in this section. Some other interesting additions to a tracking algorithm for autonomous vehicle will be briefly discussed. These methods may improve performance or speed of the tracker in general. For this thesis there was no time to implement all of the features, but future research could be conducted on one of the following topics.

6.3.1 More elaborate tests

Although the three experiments in this research show that the region tracking algorithm works well in many situations, further testing should be performed to ensure that there are no situation where the tracker will fail. The simulations are performed in such a way to encompass as many urban traffic situations as possible, but specific scenarios should be tested. This testing can be done in the same way as with this research; recording the eHorizon data and using PreScan to generate object, but preferably the algorithms should be compiled to run on an actual vehicle driving in the real World.

At first the algorithm could run on a vehicle that is driven by a person, to verify the performance with respect to computation times. This would require the algorithm to be compiled into the computers inside the vehicle. Compiling the code to a more low level language, such as C++, will most likely result in a significant increase in performance.

6.3.2 Using the eHorizon data road curvature

Briefly mentioned in the conclusions and more detailed in Appendix C, the eHorizon system was not used in its full potential. Due to incomplete documentation the assumption was made that the infrastructure consisted of linear sections between stubs. In a later stage it was realized that the roads can be described by higher order functions such as clothoids or polynomials. Using this extra information will very likely result in a better description of the road and therefore also a more accurate description of the region boundaries.

While using the road curvature provided by the eHorizon system will very likely lead to a more accurate region description, it will also mean a more complex function to construct this region. In this research rectangular sections are added together to form a polygon describing the region boundaries. Using more complex functions as road descriptions, will likely mean a more complex definition for the regions as well. Further research should be performed to find out whether this added complexity is within the possibilities of computation power.

6.3.3 Keeping region boundaries in memory

Although using more complex description can make the region construction process more demanding on computing power, it may be possible to reduce the load by keeping track of the region boundaries. In the current implementation the regions are constructed independently for every frame. This ensures that the region is always up to date, but also means that a lot of processing is done in consecutive frames that is very similar.

It may be possible to use the region information from one frame and use that to *update* the region boundaries as opposed to constructing everything starting from zero. In this case there will inevitably be more data that needs to be communicated from one frame to the next, but it may save greatly on processing times. In the first experiment (normal driving through an urban area) it was

found that the majority of processing time is spent constructing the regions. During the experiment about 13 ms (74% on average) of the time was spent on region construction (min. 3 ms, max. 37 ms), this can also be seen from the graph in Figure 76.

Although the absolute time taken to construct the region is not very much in this scenario, it may increase significantly if larger, or more regions are constructed. Optimizing the method of region creation could result in an algorithm that is even faster than the presented region tracker.

Another possibility to reduce the processing times is to construct the region only every n^{th} frame. Since the infrastructure does not change, this is possible if the vehicle speed is low enough to stay inside the region. If this option is used and every tenth frame the region is constructed anew, this could reduce the region construction time to one tenth of its original. Assuming the 13 ms average time it takes to construct regions, the entire region tracking algorithm could be as much as 10ms faster on average per frame than shown in this research.

The details of this method should be investigated; both the tracking of region boundaries and keeping the same regions for several frames can potentially reduce the total processing times significantly.

6.3.4 Using more than one region

In this research only one region was constructed and objects were tracked with a JPDA algorithm whenever they were within this region, and with a nearest neighbor Kalman filter when outside. This method worked well in the experiments, but there may be situations where it is beneficial to look further ahead, or to the sides of the road. In this situation it is possible to add regions and have the objects inside every region be tracked by their own algorithm. This will lead to a greater area that is covered by an accurate algorithm, while keeping the number of objects low enough for it to run at the required speed.

The theory for the construction of the regions was briefly discussed in Section 3.5, but was not implemented in this research. It would be interesting to see whether adding more regions will lead to a faster tracker or an algorithm that can accurately track more objects than the current implementation.

6.3.5 More complex algorithms can be used

When adding more regions to the algorithm, it also becomes possible to make the regions smaller. Smaller regions will generally have less objects in them, which paves the way for even more complex tracking algorithms. In the literature survey conducted before starting this research several methods of tracking were disregarded because of their high computational requirements. These algorithms include *Multi-Hypotheses tracking* (MHT) and *particle filtering*, which have shown good results at the cost of great processing requirements.

By using the region constructor from this research it may be possible to limit the objects in a region to a manageable number. The algorithms should be able to give even more accurate information about the objects, than the JPDA algorithm does.

Instead of using more complex algorithms on smaller areas, it may be useful to increase the size of the areas and use a clustering algorithm inside the region. This possibility was briefly tested in this research, but showed no improvement over using a JPDA algorithm without clustering inside the

region due to the low number of objects. Changing the size of the region may also change the difference in processing times.

6.3.6 Using infrastructure trajectories within the filtering

Around the turn of the century there were already some researchers looking into the possibility to use known information of the infrastructure to aid in tracker performance [7]. Although these researches used an airborne platform to survey an area on the ground, the idea may be feasible to use in ground-bound autonomous vehicles as well.

Kirubarajan et al. [62;63] is one of the first researches done on this topic and builds on the general idea to model the movement of a vehicle that is on a section of road as a one-dimensional problem. In most cases this is a valid assumption, since a vehicle that is on a road, will most likely stay on it. In practice they use an IMM filter (Section 2.1.3) which has a specific motion model for any possible trajectory on the road ahead.

Whenever there is an intersection of roads, a model is made for every possible direction of travel. The IMM method will be used to determine which model (direction) the vehicle is actually traveling. One extra model is added that is two-dimensional and does not take into account the infrastructure data. This model is created to cope with objects that are not on the road, or objects that move out of the road.

Streller [64] extended on this research in 2008 by solving some of the minor problems with this method, while Mertens et al. [65] extended the method to work with three-dimensional infrastructure descriptors. Before that all movement took place in a plane, but by adding the third dimension (height) the predictions can be made more accurate.

Using infrastructure data in this way is very different than generating a region of interest as was proposed in this thesis. It may however be possible to use the available data using multiple methods to gain a performance advantage. To work well in an environment that is suitable for an autonomous vehicle it is necessary to track with greater accuracy than proposed in the aforementioned papers. It is very important to know at which lane a vehicle is, to be able to decide on proper actions. It may be required to have a model for every lane on the road for this approach to work properly. Further research could indicate whether this is a viable method for an autonomous vehicle.

Appendix A Implementation of the tracker functions

The algorithms used to track the objects are very similar regarding the inputs and outputs, to make sure that they are interchangeable without too much effort in rewriting the code. As a guideline the JPDA-tracker is used to show the inputs and outputs. The extra *modules* inside the clustering and region-trackers are described afterwards.

The tracker-functions are at the moment implemented in MATLAB code, but it should pose no problem to recode it into a more basic language, such as C++. The syntax of the call to the JPDA-tracker is shown in Code 1, with the required inputs and outputs shown in bold letters. The non-bold inputs and outputs are purely for evaluation purposes and do not have a significant effect on the functions in processing time.

```
[newTracks,nassoc,nhyp] = trackerJPDA(measurements,tracks,parameters,varargin)
```

Code 1. Function call to the JPDA tracking algorithm as implemented in MATLAB. Bold variables are required.

A.1 Inputs

The first input to the algorithm is the matrix containing the measurements of the sensors. These should already be transformed into the Cartesian coordinate system that is used in the tracking algorithms. Every row of the matrix represents the measurements of one object, while the columns represent the parameters of the object. In this research only the position is measured, so the measurement matrix has the form as shown below.

Object ID 1	x-position 1	y-position 1
Object ID 2	x-position 2	y-position 2
…	…	…
Object ID *m*	x-position *m*	y-position *m*

Note that the first column is not actually measured data, but an ID for the object. In real life situations, this ID is not available, but in this research it is used to check the position accuracy of the tracking algorithms. If more properties of the objects are measured and used in the filter of the trackers, columns can be added. For example; if the speed in both directions is also measured, the matrix would be $m \times 5$ in size.

The input *tracks* is a data-structure containing all information about all tracks. It will be initialized as empty, since there are no tracks when the algorithm runs for the first time. Once one track (or more) is initialized, the structure will contain all information about the track(s). Every track can be accessed by filling in an index right after the *track* variable, then after a dot (.) the requested data should be stated. Table 15 shows which information is available and what type it is.

The state and covariance should require no explanation; they are the usual outputs of any filtering algorithm based on the Kalman filter. In this research a state of six variables is used, because a constant acceleration model is implemented for the prediction of the motion of objects.

Field	Size	Description
tracks(n).state	6×1	State vector $[x\ y\ v_x\ v_y\ a_x\ a_y]'$
tracks(n).covariance	6×6	Covariance matrix
tracks(n).counter	Integer	For track maintenance
tracks(n).region	Integer	*Not used*
tracks(n).objID	Integer	ID of the object in the track (for evaluation only)
tracks(n).error	Double	Absolute error with ground truth (in m)

Table 15. Fields of the tracks *input variable.*

The counter indicates the track certainty, as explained in Section 2.2.3. It essentially is a counter of frames since the track was initialized, with a maximum value implemented. It increases when a track is associated, and decreases if no measurement is found for the track in a certain moment in time.

Initially the plan was to implement multiple regions, but as a first step the algorithm described in this thesis only has one region: objects can be either inside, or outside of it and are tracked according to this. In future implementations, the *tracks(n).region*-variable can be used to identify in which region the track was found.

Finally; the variables *objID* and *error* are only used to evaluate the performance of the algorithms. The *objID* is the same ID as used by PreScan for the measurements and ground-truth. Using this ID, the *error* can be calculated with respect to the exact known position of the object.

The last required input is the data-structure containing the parameters for the trackers. The meaning of the parameters is explained in the description in Table 16 below. *ClusterGate* is only used for the clustering algorithm. *AddObj*, *addMeas* and *viewDist* are only used by the region-tracker.

Field	Size	Description
parameters.motionModel	6×6	Motion model used in the filter
parameters.measurementModel	2×6	Measurement model used in the filter
parameters.processNoise	6×6	Expected process noise (for filter)
parameters.measurementNoise	2×2	Expected measurement noise
parameters.gateSize	Double	Size of the gates in σ
parameters.Pdetect	[0 - 1]	Probability of detection
parameters.counterMax	Integer	Maximum of the counter
parameters.counterAdd	Integer	Counter is increased by this number when track is associated
parameters.counterSub	Integer	Counter is decreased by this number when track is not associated
parameters.mergeDist	Double	If tracks are closer than this distance (in m) they are merged into one track
parameters.outputStateSize	Integer	Length of state vector (calculated)
parameters.clusterGate	Double	Size of the clustering gates in σ
parameters.addObj	Double	Object region is extended this far (in m) outside the infrastructure
parameters.addMeas	Double	Measurement region is extended this far (in m) outside the region boundary
parameters.viewDist	1×2	Distance along infrastructure of region extension [front , rear] in m

Table 16. Fields of the parameters *input variable.*

To make the algorithm interchangeable with the region-tracker, it is possible to input more variables into the function. All extra inputs will however be disregarded by the JPDA algorithms with and without clustering.

The last input for the region-tracker is the eHorizon data which is transformed into a more readable format for this research. The input *regiondata* is again a data-structure that contains all the eHorizon data that is required for the tracking algorithm. Table 17 below shows the structure of the input, as well as a short description.

Field	**Size**	**Description**
regiondata.sect	$s \times 8$	List of sections within eHorizon
regiondata.node	$p \times 3$	List of nodes within eHorizon
regiondata.egoSec	Integer	ID of current section
regiondata.egoLon	Double	Distance from start of current section (in m)

Table 17. Fields of the regiondata *input (region-tracker only).*

The list of sections contains all the information about the infrastructure properties between any two connected nodes. Every row in this matrix represents one section, which is a straight line between two nodes. The image below shows which elements are present in the sections list.

Section 1 Section ID	Section 1 Startnode ID	Section 1 Endnode ID	Section 1 Length	Section 1 # of lanes	Section 1 Lanewidth	Section 1 Heading	Section 1 Road width*
Section 2 Section ID	Section 2 Startnode ID	Section 2 Endnode ID	Section 2 Length	Section 2 # of lanes	Section 2 Lanewidth	Section 2 Heading	Section 2 Road width*
…	…	…	…	…	…	…	…
Section *s* Section ID	Section *s* Startnode ID	Section *s* Endnode ID	Section *s* Length	Section *s* # of lanes	Section *s* Lanewidth	Section *s* Heading	Section *s* Road width*

From this list of sections, all information is available, except the location on the map. The location information is inherited through the start- and end-node IDs in the second and third columns. Using a node ID instead of a fixed location (coordinates) gives a better insight into multiple connections to a node, which occur at intersections. The shape of the node-list is shown below and contains only the position of the nodes and off course the node ID.

Node 1 ID	Node 1 x-position	Node 1 y-position
Node 2 ID	Node 2 x-position	Node 2 y-position
…	…	…
Node *p* ID	Node *p* x-position	Node *p* y-position

The position given in the node list is the PreScan x- and y-coordinates, but in the actual vehicle this can be any Cartesian coordinate, for example; UTM. The parameter *egoSec* defines the current section ID of the vehicle within the sections of the eHorizon data and the *egoLon* gives the distance (in m) from the start of the current section to the vehicle. This information is used to calculate which sections are within range of the region view distance.

A.2 Outputs

There is only one output to all the functions that is necessary: all the information of the tracked objects. *newTracks* is a data-structure of the same form as the input *tracks*. It contains the state and covariance matrix for every track, as required by any filtering algorithm based on the Kalman filter. The counter for track maintenance is updated, as are the object ID and the absolute error. The region is currently not in use, as in this research there is only one region at most (see Table 18).

Field	Size	Description
newTracks(n).state	6×1	State vector $[x\ y\ v_x\ v_y\ a_x\ a_y]'$
newTracks(n).covariance	6×6	Covariance matrix
newTracks(n).counter	Integer	For track maintenance
newTracks(n).region	Integer	*Not used*
newTracks(n).objID	Integer	ID of the object in the track (for evaluation only)
newTracks(n).error	Double	Absolute error with ground truth (in m)

Table 18. Fields of the newTracks *output variable.*

Aside from the required output, there are several outputs that are used to evaluate the performance of the trackers and to be able to generate images and figures. Table 19 below gives a short summary of these outputs and a description.

Output	Used in	Description
nassoc	All	Number of association made by the algorithm
nhyp	All	Number of hypotheses processed
nclusters	Cluster	Number of clusters created
conf	Cluster	Used to evaluate the confidence ellipses
polygons	Region	Information to draw the region in a figure

Table 19. Other outputs that are only used for evaluation of the algorithms.

A.3 Function trees

To make the code more readable, some of the sub-processes are put in their own functions and sub-functions. The tables in this section can be used to see which function is required and how it is used. The names used are the same as the filenames. A short description is given in the last column of the tables.

JPDA algorithm	Description
getPARAMETERS	Get the tracker parameters
trackerJPDA	Main function for JPDA
└ subInitTracks	To initialize new tracks
└ subFilterOnly	Filter tracks, no association
└ subGetOmega	Get the validation matrix Ω
└ subGetScenarios	Get the possible scenarios from Ω
└ subGetProbabilities	Calculate the probabilities
└ subMixAndFilter	Mix measurements and filter results
└ subTrackMaintenance	Initialize and delete tracks
| └ subInitTracks	To initialize new tracks
└ subDeleteDuplicates	Delete duplicate tracks

Table 20. Function tree of the JPDA algorithm.

Clustering JPDA algorithm	Description
getPARAMETERS	Get the tracker parameters
trackerJPDAC	Main function for clustering JPDA
└ subInitTracks	To initialize new tracks
└ subFilterOnly	Filter tracks, no association
└ subGetOmegaCluster	Get the validation matrix Ω
└ subGetClusters	Get the clusters
└ subClusterFilter	Process the clusters
| └ subGetOmega	Get the validation matrix Ω
| └ subTrackMaintenance	Initialize and delete tracks in Cluster
| | └ subInitTracks	To initialize new tracks
└ subTrackMaintenanceCluster	Initialize and delete tracks
└ subDeleteDuplicates	Delete duplicate tracks

Table 21. Function tree of the clustering JPDA algorithm.

Region JPDA algorithm	Description
getPARAMETERES	Get the tracker parameters
trackerJPDAR	Main function for region JPDA
└ subInitTracks	To initialize new tracks
└ subGetPolygons	Calculate the region boundaries
| └ subMergePoly	Merge two polygons
| └ subInPoly	Check for points inside polygon
| └ subPolyIntersections	Check for intersections between polygons
└ subFilterOnly	Filter tracks, no association
└ subGetPolygons	Calculate the region boundaries
| └ subMergePoly	Merge two polygons
| └ subInPoly	Check for points inside polygon
| └ subPolyIntersections	Check for intersections between polygons
└ subGetRegions	Calculate the region boundaries
| └ subMergePoly	Merge two polygons
| | └ subInPoly	Check for points inside polygon
| | └ subPolyIntersections	Check for intersections between polygons
| └ subInPoly	Check for objects inside regions
└ subTrackRegion	Track objects inside region
| └ subFilterOnly	Filter tracks, no association
| └ subGetOmega	Get the validation matrix Ω
| └ subGetScenarios	Get the possible scenarios from Ω
| └ subGetProbabilities	Calculate the probabilities
| └ subMixAndFilter	Mix measurements and filter results
└ subNNKF	Nearest neighbor KF for outer region
└ subInitTracks	Initialize new tracks (maintenance)
└ subDeleteDuplicates	Delete duplicate tracks

Table 22. Function tree of the region JPDA algorithm.

Appendix B The PreScan software

The data used to evaluated the performance of the algorithms discussed in this report is generated in part with use of the software PreScan, by TASS International. This software package was originally created to test advanced driver assistance systems (ADAS), such as automatic cruise control and lane keeping assist systems. The possibility of adding multiple sensors to the host vehicle and simulating traffic on realistic infrastructure elements makes it a great tool to test autonomous vehicle systems as well.

In this research an environment close to Delft, The Netherlands, is imported in the software and traffic is simulated. The returns from the simulated radar sensor are combined with recorded data from a vehicle driving the same roads in real life. The results of this are then used to evaluate a novel object tracking algorithm. In this appendix the main scenario of this thesis will be used to explain how PreScan was used to obtain the results presented in Section 5.2.

B.1 Environment

The first step in using PreScan is generating the environment to test in. For this research a real-life scenario was recreated, to facilitate the usage of recorded data in combination with simulated data. A map was downloaded from OpenStreetMaps.org [61] that contains the area of interest. The map can be imported into PreScan, which then converts it to a road-network inside the PreScan environment.

Importing this OSM-file is very easy, but getting everything to connect perfectly requires some manual adjustments. Especially at intersections some problems can occur regarding connections and number of lanes. These issues are not hard to correct, but do require some time to work on.

What really helped at this stage were the *underlays* that are imported. Google Maps [58] was used to generate an aerial image of the surroundings. The image is then used as an underlay for the PreScan scenario, meaning that the image can be used to locate the lanes and position objects in the PreScan environment. The map underlays also give a more realistic appearance to the scenario, since it looks like the real environment with green patches where there is grass etc.

Another piece of data that is imported into PreScan is the GPS-trace of the real data-recording. The position is recorded every second and PreScan interpolates this into a trajectory and speed-profile at the scenario frame-rate of 20 Hz.

B.2 Sensors

Although the vehicle used in this project will eventually be equipped with a 360° radar coverage and several camera sensors to add to that, for this experiment a single radar sensor is simulated. The radar is facing forward and has the same specifications as the short-range beam of the Continental ARS-300 radar that is used within the project [48]. This means a FOV of 56° with a resolution of 4° and a maximum range of 60 m. All noise levels for this sensor are also entered as specified in the documentation of the radar.

The radar is simulated using the *technology independent sensor* or TIS, which returns a position of an object in a similar way as any active sensor would. Examples are radar and LIDAR sensors, which fire

a beam and interpret the reflection of this beam. The output of the TIS is a range and a bearing and possibly a range-rate (Doppler velocity) as well. In this research only the range and bearing are used as measurements. One other return of the sensor is used to evaluate the performance of the algorithms: the target ID. This ID is impossible to measure with a real sensor and thus only exists in the PreScan environment. The ID corresponds with the object in the scenario; every object, be it a car, tree, bike, or building, has a unique identifier. By storing the ID of the measured object, a comparison can be made with the true position at any time during the simulation.

The true position, or ground-truth position, is *measured* by another sensor in PreScan: the *actor information receiver* or AIR-sensor. This sensor returns information of all the objects within range, exactly as it is, so without noise or other artifacts. It is also insensitive to occlusions of objects: a bike behind a car *will* be detected with the AIR-sensor, whereas the TIS (radar) will not give a return if the object is not in direct line-of-sight of the sensor. By setting the FOV of the AIR-sensor to 180° and the maximum range to 100 m, it is ensured that objects that are tracked always have a corresponding ground-truth position available.

The AIR-sensor returns the position of objects just like the radar/TIS does, with a range and bearing and also gives information about the ID of the object. Other information includes the ego-velocity and heading of the object, although this information is not used in this research.

B.3 Traffic and obstacles

Having a car driving around in a simulated environment is nice, but it only gets useful to collect data when objects are positioned in the scenario. A start was made with road-side objects, in this case mostly trees. As stated before, the underlay is used to position trees at realistic locations. Care is taken to try and match the real World trees as close as possible. Of course there will be slight variations, because it still is a simulation, but the general location and number of objects is as close to reality as possible as was demonstrated in Section B.1.

Other stationary objects that are simulated include the buildings along the route. PreScan provides a range of standard buildings, such as houses and offices, but none of the buildings are exactly as they are in the real World when it comes to appearance. However, the location of buildings is again as close as possible to the truth, using the map underlay to determine the positions. Furthermore the type of buildings is close to the reality as well; where-ever there is a two-store house, this will also be simulated in the PreScan simulation.

For this research buildings are purely inserted to make the scenario look acceptable, they do not show up on any of the sensors. The reasoning behind this is that the used TIS-sensor to simulate the radar does not have a wave-propagation model to simulate the measurements and thus may return unexpected results when a large object such as a building is within sensor range.

Aside from the stationary objects an attempt is made to include moving objects in such a way that they represent normal traffic. To this end 12 bicycles and 25 cars are simulated following several different routes in the scenario. Since in real life there is a dedicated bicycle path, the bicycles are not simulated on the same road as the host vehicle, but they are within range of the radar sensor while driving on one of the side-roads.

Most of the cars that are simulated represent traffic in the opposing direction of the ego vehicle, while one vehicle drives in front, within radar range for most of the scenario. Several cars will make maneuvers at intersections, just as one would expect real traffic to do. All vehicles, including bicycles have a constant speed, although this speed is not the same for all vehicles.

B.4 Data gathering

The GPS-trace of the experiment on the real road was imported into PreScan, but this was not the only piece of real data that was used. During the experiment the CAN-messages from the eHorizon system were recorded every second. This data was later reconstructed into a more usable format for MATLAB and made into a function that has the traveled distance as input and delivers the output of the real eHorizon system at that location.

To use this function, PreScan is used to compute the traveled distance along the taken path (GPS-trace). Using this function it is now possible to gather the information that eHorizon gives at any time during the simulation.

It is at this point that a connection between PreScan and MATLAB/Simulink is made. PreScan generates a Simulink model that includes all the actors and objects in the scenario. The model can be used to implement a controller to the ego-vehicle, but in this research it is only used to gather the measurement data from the sensors in the simulation.

The data output is synchronized in simulation time and all data is gathered at 20 Hz. The data measured in the simulation is added to the processed eHorizon data and everything is stored in a single data-struct that can be accessed for every time-step in the simulation. The data-struct contains all the information, including sensor measurements, ground-truth data and all parameters of the scenario.

The data-struct is then used to run the tracking algorithms on the measurements. The trackers run in MATLAB and at this point the connection with PreScan is no longer required, because all the measurements are already stored. To run the trackers, a loop is created that accesses the data for every time-step in the simulation, feeds it to the algorithms and then stores the results. The tracks are kept in memory for the next time-step, where they are combined with the new measurement data.

Appendix C Remark on the eHorizon system

This appendix is written to explain the issues encountered with the eHorizon system during the MSc research; "Object tracking for autonomous vehicles with use of region splitting" by Randy Stakelbeek, for the Delft University of Technology, within the DAVI project. A short summary of the research will be given first, so that the reasoning behind the issues can be understood better.

C.1 Summary of research topic

Object tracking is a major concern for autonomous vehicles, since tracked objects form the basis of all the decisions that will be made by the vehicle. From previous research [4], it is known that the *joint probabilistic data association* (JPDA) method is very capable of tracking objects in a cluttered environment, even if those objects are maneuvering of crossing each other's paths. A clear disadvantage of using JPDA is that it is computationally heavy, especially when many objects are concerned.

The JPDA algorithm gets exponentially slower with the number of objects and with the number of measurements within the survey area. The proposal in this research is to split the survey area into regions to ensure that the objects can be tracked sufficiently fast. For this research the region closest to the vehicle will be shaped along the infrastructure boundaries. Most of the objects outside of the infrastructure are of little influence to the immediate decisions of the autonomous vehicle. The objects will be tracked with a less accurate, but faster, nearest neighbor Kalman filter (NNKF).

C.2 Initial thoughts about the eHorizon system

The eHorizon system was designed to give extended information about the road ahead of a vehicle, in order to improve *advanced driver assistance systems* (ADAS) and is based on the standardized ADASIS protocol. The extended protocol specification was available to the DAVI project because of a collaboration with the manufacturer [5] of the eHorizon system used. The latest version of this document available was v2.0.3.0 from December 2013 [6]. The system uses the latest version of the protocol with additional messages that are defined by the manufacturer.

The assumption was made that the infrastructure definitions of the eHorizon system would be accurate enough to be able to generate regions with boundaries that can follow the boundaries of the infrastructure. To this end the PROFILE LONG messages were used to obtain longitude and latitude data of the defined points in the road segments, as specified in the protocol: "*The absolute road geometry, expressed in terms of longitude/latitude coordinates has no explicit place in the ADASIS v2 protocol. We assume that in ADAS applications this information has very limited usability. Nevertheless, absolute road geometry can easily be published by ADASIS v2 Horizon Provider as two PROFILEs: LONGITUDE profile and LATITUDE profile.*" [7].

[4] Y. Bar-Shalom and P. K. Willett, Tracking and Data Fusion, A handbook of algorithms YBS Publishing, 2011.
[5] RoadScape™, by MACOM
[6] 200v2.0.3-D2.2-ADASIS_v2_Specification.0
[7] 200v2.0.3-D2.2-ADASIS_v2_Specification.0 , Section 6.6, page 118.

Since no detailed information about the location of these points with respect to the infrastructure is given, it was assumed that all points were located on the center of the road. This method is backed by the statement made in the protocol document: *"Linear interpolation of LONGITUDE and LATITUDE PROFILEs will in effect result in classical, poly-line based road geometry."* [8]. This assumption was made at the start of the project.

C.3 Initial results of research w.r.t. the eHorizon system

The results obtained creating the region were not as good as expected, so an investigation into why this is the case was started. In the image in Figure 84 one frame of the result is shown. The red lines indicate the constructed region, using the eHorizon data, the blue lines indicate the (visually) correct geometry of the roads. The vehicle fitted with the eHorizon system is in the center of the image (green dot) and is driving to the right in this image.

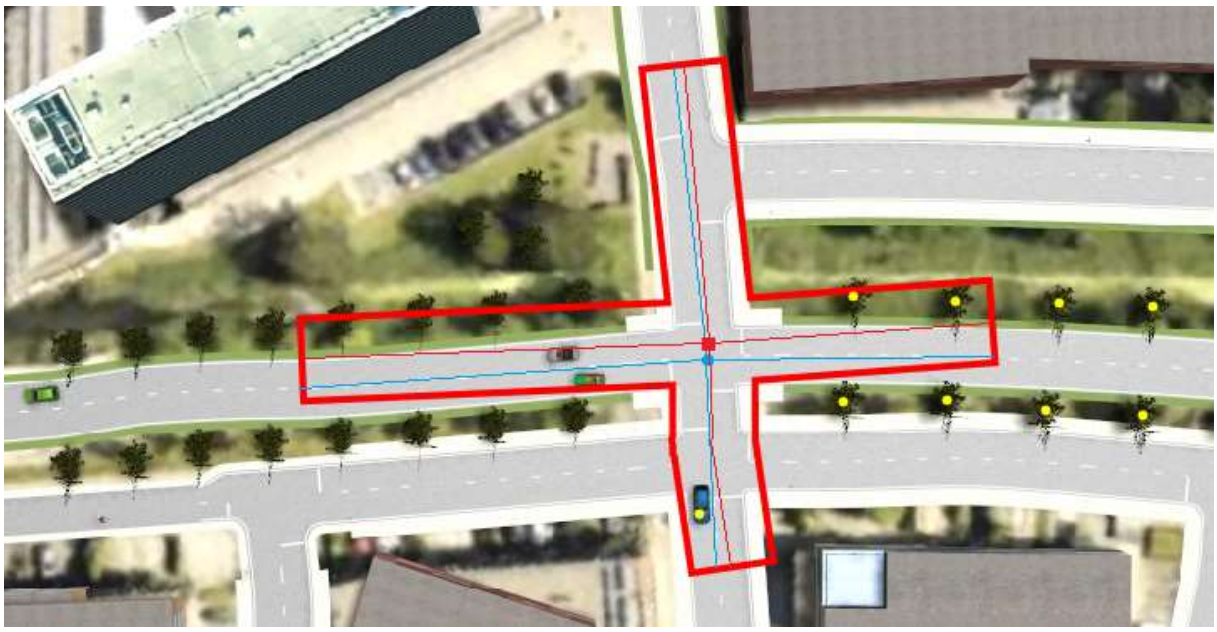

Figure 84. One frame of the results of region construction. Red lines indicate the region, constructed using the eHorizon data. Blue lines indicate the (visually) correct geometry. The underlay is taken from Google Maps, the software used is PreScan and MATLAB.

The eHorizon data was recorded, as well as the GPS-trace of the vehicle. The GPS-trace is imported into the software, along with a map of the infrastructure (OpenStreetMaps.org (OSM)) and the underlay (Google Maps).

At first sight the problem seemed to be just a small issue with offset between the maps, however, at this intersection it is clear that there is not only a difference in offset between the maps, but the points used to define the infrastructure in eHorizon seem to give a different geometry than the GPS-trace and the other maps used. The angle between the blue and red line at the left of the intersection is in the opposite direction of the angle on the right side.

Taking into consideration all the accuracies of the systems and data used, this is still a large error. Since the GPS-trace is real data and the OSM map *and* Google Maps underlay seem to correspond

[8] 200v2.0.3-D2.2-ADASIS_v2_Specification.0 , Section 6.6, page 118.

well with this trace, it is assumed that there is either a large error in the map used in the eHorizon system, or the definition of the points is different than assumed.

C.4 Problems using the eHorizon system

From this analysis and further discussions with the manufacturer is follows that not all the points given in the PROFILE LONG messages are situated in the center of the road. Reading from the ADASIS v2 protocol document this observation is supported: "*Profiles are defined by specifying* profile support points *and by specifying the* interpolation *method that is to be used to calculate the profile between support points.*" [9].

However there seems to be a problem with the provided data; the used method of interpolation is not always given as an output, so reconstructing the infrastructure geometry is at the very least inconsistent. At moments when the supporting data *is* available, such as the curvature, the definition on how to use this data is mostly not given. The protocol document indicates that almost any method of interpolation is possible, but the output of the system does not provide this at this moment in time.

The road could be described as a polynomial (up to an order of 5), clothoid, or even other methods. In any case, there does not seem to be an indication as to whether a given point is on the center line of the road, or whether it is a support point that is only used as information for an interpolation method.

At this moment the assumption is made that the number of *profile messages* also define the interpolation method that should be used. However, it remains unclear whether this is the right way to use the information available. Furthermore, the exact usage of the parameters given also remains unclear.

C.4.1 Conclusions

Right now it is hard to reconstruct the road geometry because the data is provided inconsistently. The assumption that the number of *profile messages* can be used to define the interpolation method does not always lead to satisfactory results.

Whenever there is data available that is intended to be used in interpolating the road geometry, the method of interpolation is not clear. This means that the data cannot be used correctly.

C.4.2 Recommendation

Contrary to the assumption made in the protocol document and quoted before (Section C.2), knowing the exact infrastructure geometry can be of great use to an autonomous vehicle. Including the definition of the road geometry in the protocol in such a way that it is easily and *consistently* reconstruct-able would make understanding the environment around the vehicle much easier and more reliable.

[9] 200v2.0.3-D2.2-ADASIS_v2_Specification.0 , Section 5.7.1, page 100.

Bibliography

[1] L. da Vinci, "Codex atlanticus," *Originally written between 1478 and 1519*, 2006.

[2] M. E. Rosheim, "Leonardo's Programmable Automaton and Lion," in *Leonardo's Lost Robots* Springer Berlin Heidelberg, 2006, pp. 21-68.

[3] America's Independent Electric Light And Power Companies and Boys' Life, "Because electricity may be the driver!," 1956, p. 7.

[4] J. &. J. E. Carlson, "Auto Tech Report - Emerging Tech - Autonomous Car," *IHS Technology*, 2013.

[5] K. J. Bussemaker, "Sensing requirements for an automated vehicle for high-way and rural environments," *Delft University of Technology repository*, 2014.

[6] R. A. R. Stakelbeek, "Tracking objects surrounding an autonomous vehicle - A review of methods and algorithms," 2015.

[7] Y. Bar-Shalom and P. K. Willett, *Tracking and Data Fusion, A handbook of algorithms* YBS Publishing, 2011.

[8] L. Y. Pao, "Centralized multisensor fusion algorithms for tracking applications," *Control Engineering Practice*, vol. 2, no. 5, pp. 875-887, 1994.

[9] L. Y. Pao, "Distributed multisensor fusion," *Am. Inst. of Aeronautics and Astronautics*, 1994.

[10] Continental, "Continental ARS-300 datasheet," Continental, 2012.

[11] K. Becker, "eHorizon - Products in the market and future," *unknown*, 2013.

[12] D. L. Hall, "An introduction to multisensor data fusion," *Proceedings of the IEEE*, vol. 85, no. 1, pp. 6-23, 1997.

[13] E. Coelingh and A. Eidehall, "Collision Warning with Full Auto Brake and Pedestrian Detection - a practical example of Automatic Emergency Braking," in *Intelligent Transportation Systems (ITSC), 2010 13th International IEEE Conference on Intelligent Transportation Systems* 2010, pp. 155-160.

[14] A. Amditis, A. Polychronopoulos, I. Karaseitanidis, and G. Katsoulis, "Multiple sensor collision avoidance system for automotive applications using an IMM approach for obstacle tracking," in *Information Fusion, 2002. Proceedings of the Fifth International Conference on Information Fusion,* 2 ed 2002, pp. 812-817.

[15] A. Polychronopoulos and U. &. T. F. Scheunert, "Centralized data fusion for obstacle and road borders tracking in a collision warning system," in *International Conference on Information Fusion* 2004.

[16] B. S. Y. Rao and H. F. Durrant-Whyte, "A Fully Decentralized Multi-Sensor System For Tracking and Surveillance," *The International Journal of Robotics Research*, vol. 12, no. 1, pp. 20-44, 1993.

[17] C. Y. Chong, S. Mori, and W. H. Barker, "Architectures and algorithms for track association and fusion," *Aerospace and Electronic Systems Magazine, IEEE*, vol. 15, no. 1, pp. 5-13, 2000.

[18] N. Xiong, "Multi-sensor management for information fusion: issues and approaches," *Information Fusion*, vol. 3, no. 2, pp. 163-186, 2002.

[19] A. Bacha, C. Bauman, R. Faruque, M. Fleming, C. Terwelp, C. Reinholtz, D. Hong, A. Wicks, T. Alberi, and D. &. o. Anderson, "Odin: Team victortango's entry in the darpa urban challenge," *Journal of Field Robotics*, vol. 25, no. 8, pp. 467-492, 2008.

[20] M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, D. Dolgov, S. Ettinger, D. Haehnel, T. Hilden, G. Hoffmann, and B. &. o. Huhnke, "Junior: The stanford entry in the urban challenge," *Journal of Field Robotics*, vol. 25, no. 9, pp. 569-597, 2008.

[21] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. N. Clark, J. Dolan, D. Duggins, T. Galatali, and C. &. o. Geyer, "Autonomous driving in urban environments: Boss and the urban challenge," *Journal of Field Robotics*, vol. 25, no. 8, pp. 425-466, 2008.

[22] B. Friedland, "Optimum Steady-State Position and Velocity Estimation Using Noisy Sampled Position Data," *Aerospace and Electronic Systems, IEEE Transactions on*, vol. AES-9, no. 6, pp. 906-911, 1973.

[23] J. H. Painter and D. Kerstetter, "Reconciling steady-state Kalman and alpha-beta filter design," *Aerospace and Electronic Systems, IEEE Transactions on*, vol. 26, no. 6, pp. 986-991, 1990.

[24] S. R. Rogers, "Instability of a decoupled Kalman tracking filter," *Automatic Control, IEEE Transactions on*, vol. 34, no. 4, pp. 469-471, 1989.

[25] H. Weiss and J. B. Moore, "Improved extended Kalman filter design for passive tracking," *Automatic Control, IEEE Transactions on*, vol. 25, no. 4, pp. 807-811, Aug.1980.

[26] H. A. &. B. E. Blom, "Combining IMM and JPDA for tracking multiple maneuvering targets in clutter," in *Proceedings of the 5th International Conference on Information Fusion,* 1 ed 2002, pp. 705-712.

[27] R. Rastgoufard, "The Interacting Multiple Models Algorithm with State-Dependent Value Assignment." Tulane University, 2012.

[28] Y. Bar-Shalom, "Tracking methods in a multitarget environment," *Automatic Control, IEEE Transactions on*, vol. 23, no. 4, pp. 618-626, 1978.

[29] T. E. Fortmann and Y. Bar-Shalom, "Multi-target tracking using joint probabilistic data association," in *Decision and Control including the Symposium on Adaptive Processes, 1980 19th IEEE Conference on Decision and Control,* 19 ed 1980, pp. 807-812.

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

[31] M. S. Lee, "An efficient multitarget tracking algorithm for car applications," *Industrial Electronics, IEEE Transactions on*, vol. 50, no. 2, pp. 397-399, 2003.

[32] Y.-H. Kim, "Automotive radar tracking of multi-target for vehicle CW/CA systems," *Mechatronics*, vol. 14, no. 1, pp. 143-151, 2004.

[33] R. J. Fitzgerald, "Development of Practical PDA Logic for Multitarget Tracking by Microprocessor," in *American Control Conference, 1986* 1986, pp. 889-898.

[34] Z. Hu and H. &. B. M. Leung, "Analysis of probabilistic data association filters in a real multitarget radar tracking environment," in *SPIE's 1995 Symposium on OE/Aerospace Sensing and Dual Use Photonics* 1995, pp. 342-353.

[35] W. Qin and F. &. Q. C. Hu, "A new fast algorithm for multitarget tracking in dense clutter," *Journal of Control Theory and Applications*, vol. 3, no. 4, pp. 383-386, 2005.

[36] H. A. Quevedo, S. S. Blackman, T. Nichols, and R. J. Dempster, "Reducing MHT computational requirements through use of cheap JPDA methods," in *International Symposium on Optical Science and Technology* 2001, pp. 246-257.

[37] G. W. Pulford, "Taxonomy of multiple target tracking methods," *Radar, Sonar and Navigation, IEE Proceedings*, vol. 152, no. 5, pp. 291-304, Oct.2005.

[38] J. Dezert and Y. Bar-Shalom, "Joint probabilistic data association for autonomous navigation," *Aerospace and Electronic Systems, IEEE Transactions on*, vol. 29, no. 4, pp. 1275-1286, Oct.1993.

[39] T. Fortmann, Y. Bar-Shalom, and M. Scheffe, "Detection thresholds for tracking in clutter--A connection between estimation and signal processing," *Automatic Control, IEEE Transactions on*, vol. 30, no. 3, pp. 221-229, 1985.

[40] T. Bellemans, "Traffic control on motorways," -, 2003.

[41] N. Bourbakis and M. Findler, "Smart cars as autonomous intelligent agents," 2001, pp. 25-32.

[42] C. Nunny, A. Kummert, and S. Muller-Schneiders, "A novel region of interest selection approach for traffic sign recognition based on 3D modelling," 2008, pp. 654-659.

[43] Continental, "Clever Map Utilization with eHorizon: Continental Reduces Fuel Consumption with Intelligent Use of Navigation Data," Continental,2008.

[44] ADASIS, "ADASIS Forum Terms of Reference," ADASIS, 2011.

[45] E. Hellstrom, "Explicit use of road topography for model predictive cruise control in heavy trucks," *Student thesis*, 2005.

[46] C. Ress, D. Balzer, A. Bracht, and S. Durekovic, "Adasis protocol for advanced in-vehicle applications," in *15th World Congress on Intelligent Transport Systems* 2008.

[47] J. Taylor, "The Cramer-Rao estimation error lower bound computation for deterministic nonlinear systems," *IEEE Transactions on Automatic Control*, vol. 24, no. 2, pp. 343-344, Apr.1979.

[48] Continental, "Continental ARS 30X /-2 /-2C/-2T/-21 Long Range Radar datasheet," 2012.

[49] S. J. Julier and J. K. Uhlmann, "Unscented filtering and nonlinear estimation," *Proceedings of the IEEE*, vol. 92, no. 3, pp. 401-422, Mar.2004.

[50] P. Tichavsky, C. H. Muravchik, and A. Nehorai, "Posterior Cramer-Rao bounds for discrete-time nonlinear filtering," *IEEE Transactions on Signal Processing*, vol. 46, no. 5, pp. 1386-1396, May1998.

[51] N. Gordon, B. Ristic, and S. Arulampalam, "Beyond the kalman filter: Particle filters for tracking applications," *Artech House, London*, 2004.

[52] Advanced Navigation, "Spatial Dual Reference Manual," 2015.

[53] Continental, "Predictive Assistance: Continental Presents Adaptive Cruise Control with eHorizon," 2015.

[54] HERE maps and S. Schuman, "Why we're mapping down to 20cm accuracy on roads," 2014.

[55] D. Engwirda, "Fast points-in-polygon test," MATLAB file exchange: 2007.

[56] M. Shimrat, "Algorithm 112: position of point relative to polygon," *Communications of the ACM*, vol. 5, no. 8, p. 434, 1962.

[57] D. M. Schwarz, "Polyintersections," MATLAB file exchange: 2008.

[58] Google Maps, "Map of scenario," 2016.

[59] Y. Bar-Shalom, X. R. Li, and T. Kirubarajan, *Estimation with applications to tracking and navigation: theory algorithms and software* John Wiley & Sons, 2004.

[60] Google Streetview, "T-Shaped intersection," 2014.

[61] OpenStreetMap.org, "Map of Delft, ©OpenStreetMap contributors, availble under the Open Database License," 2016.

[62] T. Kirubarajan, Y. Bar-Shalom, K. R. Pattipati, and I. Kadar, "Ground target tracking with variable structure IMM estimator," *Aerospace and Electronic Systems, IEEE Transactions on*, vol. 36, no. 1, pp. 26-46, Jan.2000.

[63] T. Kirubarajan, Y. Bar-Shalom, K. R. Pattipati, and I. Kadar, "Ground-target tracking with topography-based variable-structure IMM estimator,", 3373 ed 1998, pp. 222-233.

[64] D. Streller, "Road map assisted ground target tracking," 2008, pp. 1-7.

[65] M. Mertens and M. Ulmke, "Ground Moving Target Tracking with context information and a refined sensor model," 2008, pp. 1-8.

About the author

Rudolf Anthony Randy Stakelbeek was born on May 21, 1985 in Meppel, The Netherlands. He received his BSc. degree in Mechanical Engineering in 2012 from the Delft University of Technology and his MSc. degree in 2016 from the same university by completing this thesis.

His interests have always been within the automotive sector, which is why he has conducted research in the field of autonomous vehicles since 2013 within the DAVI project, initiated by the Delft university of Technology.