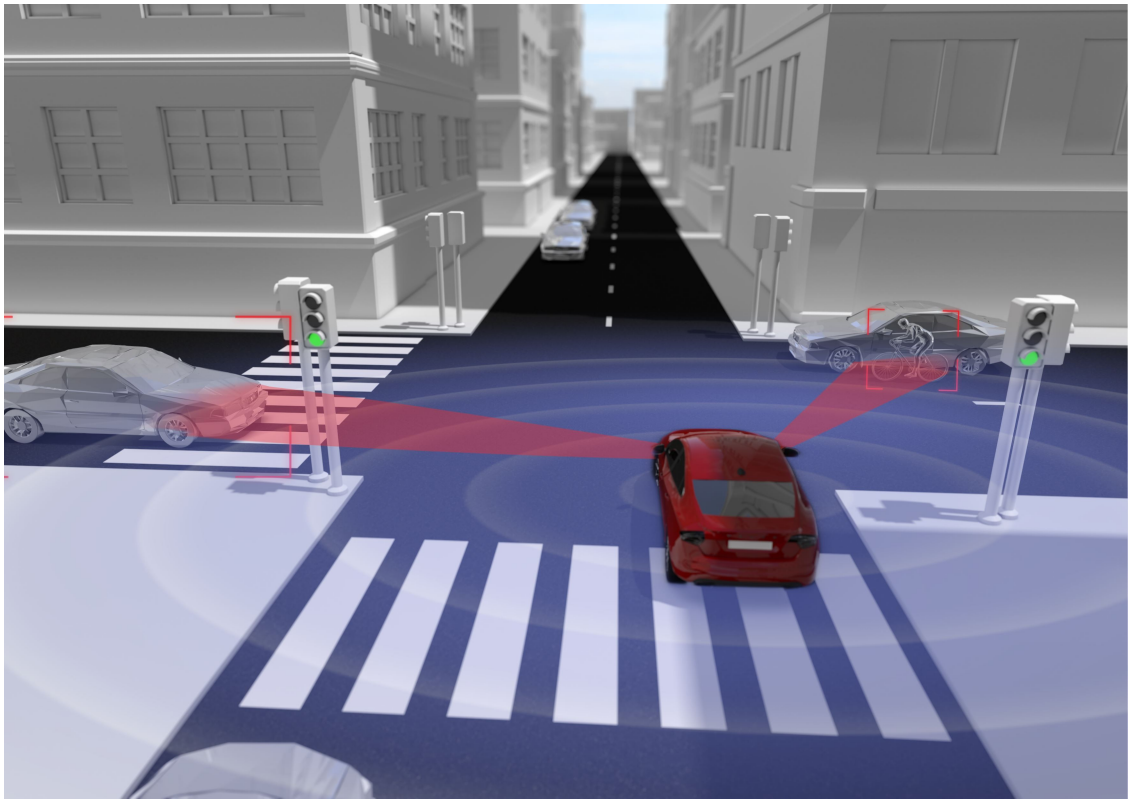
CHALMERS
UNIVERSITY OF TECHNOLOGY



# Track-to-track Fusion for Multi-target Tracking Using Asynchronous and Delayed Data

Master's thesis in Systems, Control and Mechatronics

ALEXANDER BERG
ANDREAS KÄLL

Department of Signals and Systems
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2017

Master's thesis EX022/2017

# Track-to-track Fusion for Multi-target Tracking Using Asynchronous and Delayed Data

ALEXANDER BERG
ANDREAS KÄLL


CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Signals and Systems
*Division of Signal Processing and Biomedical Engineering*
Chalmers University of Technology
Gothenburg, Sweden 2017

Track-to-track Fusion for Multi-target Tracking Using Asynchronous and Delayed Data
ALEXANDER BERG
ANDREAS KÄLL



| | | |
|---|---|---|
| Supervisor: | Ghazaleh Panahandeh, | Volvo Cars |
| Examiner: | Lars Hammarstrand, | Chalmers University of Technology |

Master's Thesis EX022/2017
Department of Signals and Systems
Division of Signal Processing and Biomedical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: A traffic scenario where object detections are shown. Source: Volvo Cars

Typeset in LaTeX
Gothenburg, Sweden 2017

Track-to-track Fusion for Multi-target Tracking Using Asynchronous and Delayed Data

ALEXANDER BERG
ANDREAS KÄLL
Department of Signals and Systems
Division of Signal Processing and Biomedical Engineering
Chalmers University of Technology

# Abstract

Self-driving and interconnected cars have recently become a big focal point in the automotive industry and many automotive OEMs are promising automated and self-driving cars within a few years. As the vehicles and surrounding infrastructure are becoming more sophisticated and able to deal with more complex situations, the involved hardware and software solutions become more intricate. These complex systems give rise to issues when verifying automated and cooperative functions which today's testing methods cannot handle. In this thesis, a proposed solution for improvement in accuracy of state estimates when combining data from multiple vehicles is developed which enables more accurate testing and verification of autonomous and cooperative vehicles. Furthermore, a vehicle can gain knowledge about objects which is located outside its sensors field of view by combining data from surrounding vehicles.

The proposed solution is a central fusion server to which the data from multiple vehicles is sent wirelessly and fused to produce a coherent and accurate description of all dynamic objects in the environment. The fusion algorithm utilises Bayesian Track-to-track fusion of out-of-sequence tracks. Filter predictions are performed using the cubature rule and unscented transform and the filter update is performed using Information De-correlation. Furthermore, a data association algorithm is developed using the Mahalanobis distance as a metric. The work has been carried out at Volvo Cars and the proposed algorithm has been implemented in a simulation platform developed by Volvo Cars.

The results show that the fusion algorithm improves the accuracy of the state estimates when data from several vehicles is combined rather than used individually. The frequency with which the vehicles send data and the transmission delay has a large impact on the fusion performance. If the transmission delay is too high or the sending frequency is too low it is no longer beneficial to fuse the data in terms of accuracy. This thesis shows that combining data from multiple vehicles can be beneficial and enable cooperative strategical or safety-related decision-making.

**Keywords:** Bayesian Filtering, Cubature Kalman Filter, Information De-correlation, Multi-object Tracking, Track-to-track Fusion, Unscented Kalman Filter

# Acknowledgements

We would like to thank our supervisor at Chalmers, Lars Hammarstrand, for input and guidance during the project. We would also like to thank our peers, Lars Sjöberg and Sangeetha Nagaraju for peer-reviewing our report and providing us with constructive feedback.

We would especially like to thank Volvo Cars for giving us the opportunity to perform our thesis with them and providing us with the tools and data needed to successfully complete the project. Special thanks to our supervisor at Volvo Cars, Ghazaleh Panahandeh, for providing us with her immense insight and knowledge. Thanks to her continuous involvement and support, the project could be steered in the right direction. We would also like to thank Christoffer Wedding at Volvo Cars for his insight in the platform which the proposed solution is implemented in.

Alexander Berg and Andreas Käll, Gothenburg, June 2017

# Contents

# List of Figures

# Acronyms

| | |
|---|---|
| **CA** | Constant Acceleration |
| **CKF** | Cubature Kalman Filter |
| **CT** | Coordinated Turn |
| **CV** | Constant Velocity |
| **ED** | Euclidean Distance |
| **EKF** | Extended Kalman Filter |
| **GDM** | Global Dynamic Map |
| **GPS** | Global Positioning System |
| **IDC** | Information De-correlation |
| **IMU** | Inertial Measurement Unit |
| **ITS** | Intelligent Transportation System |
| **KF** | Kalman Filter |
| **LDM** | Local Dynamic Map |
| **MD** | Mahalanobis Distance |
| **MOTA** | Multiple Object Tracking Accuracy |
| **MSE** | Mean Squared Error |
| **OOSM** | Out-of-sequence Measurement |
| **OOST** | Out-of-sequence Track |
| **PF** | Particle Filter |
| **S2TF** | Sensor-to-track Fusion |

| | |
|---|---|
| **SE** | Squared Error |
| **SPAS** | Simulation Platform for Active Safety |
| **T2TF** | Track-to-track Fusion |
| **UKF** | Unscented Kalman Filter |

# 1

# Introduction

In the realisation of sustainable mobility, self-driving vehicles and cooperative systems in an Intelligent Transportation System (ITS) [1] are of great importance. Current development is progressing at a fast pace and most vehicle manufacturers are preparing to launch automated and cooperative vehicles within a few years.

Modern day intelligent vehicles are equipped with advanced sensors which are used to observe the vehicles own state as well as the surrounding environment. Since all sensors are subject to measurement noise the observations of the objects suffer from uncertainty. By combining data from several sensors the uncertainty can be reduced. This combined view, hereafter referred to as a Local Dynamic Map (LDM), consists of the vehicles own state as well as the states of all the objects that the vehicle can detect and track which may include other cars, pedestrians, cyclists, etc. The LDM of a vehicle includes signals such as position, velocity, heading and yaw rate. The LDM also contains other properties of the host vehicle (the vehicle which produces the LDM) such as turn indicator status and brake light indicator. The LDM can then be used to make intelligent strategical or safety-related decisions. By combining this information with actuators in the vehicle such as electronic power steering and electronically controlled brakes, advanced control systems can be designed, for example autonomous braking and steering, lane keeping aid and autonomous drive.

In the upcoming era of Internet of Things [2], more and more devices are connected to each other and to the Internet in order to share information. This can also be applied to the automotive industry by connecting vehicles and its surrounding infrastructure to each other and create an ITS which enable cooperative strategical and safety-related decision-making. Path planning, slippery road warnings, queue warnings, fleet control, etc., can be enabled and improved by connecting vehicles to each other or to a central server.

Connecting vehicles also means that information in the LDMs can be shared and the vehicles knowledge about its surrounding can be extended beyond its sensors field of view. Two examples are illustrated in Fig. 1.1 where a complete picture of the entire traffic environment can be used to make better decisions.

**(a)** Traffic Overview **(b)** Red Light Violation

**Figure 1.1:** Two scenarios where ITS and shared data can be beneficial. Source: Volvo Cars.

Another benefit of sharing the LDMs is that the uncertainty in the state of an object detected by multiple vehicles can be decreased. In the scenario illustrated in Fig. 1.2 both of the vehicles observes an object next to the road, but due to measurement noise, the vehicles will not place the object in the same position in a global map. By combining the data from both of the vehicles a more accurate state estimate can be achieved. If this is done for all objects detected by multiple vehicles a combined view, hereafter referred to as a Global Dynamic Map (GDM), can be constructed. In the scenario in Fig. 1.2 the vehicles does not know their own states exactly due to errors in the Global Positioning System (GPS), Inertial Measurement Unit (IMU) and other internal sensors used to measure their own state. To cope with this the vehicles use filters to produce estimates (together with corresponding covariance matrices) of their own state which is represented by the dotted ellipses in Fig. 1.2. In this figure the GDM is not present and the blue estimates (stars) and uncertainties (dotted ellipses) together form the LDM produced by the blue vehicle. The same applies for the red vehicle.

In this project, the LDMs from the vehicles are sent wirelessly to a central server where the data is fused. As with any wireless communication, the transmission of data to the central server suffers from a varying time delay which in this case can also lead to packets arriving in the wrong order.

All dynamic objects in the environment such as vehicles, pedestrians and cyclists are referred to as *actors*. The actors who send data to the server are referred to as *active actors* and consequently the non-active actors are called *passive actors*. An active actor sends estimates of its own state and state estimates of other actors within its sensors field of view to the server, i.e. its LDM.

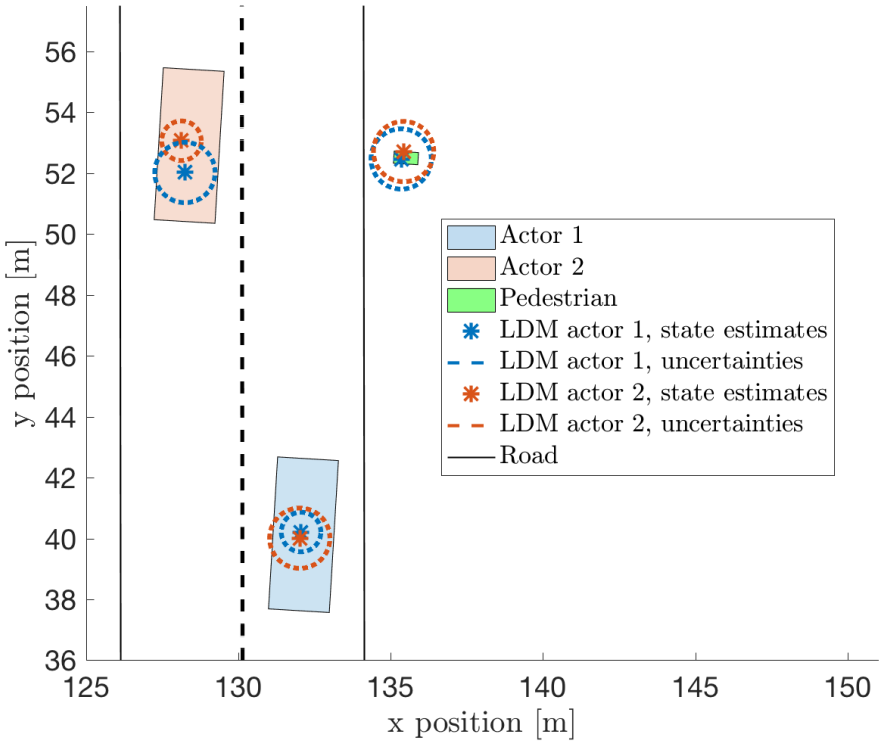



**Figure 1.2:** A traffic scenario involving two vehicles and one pedestrian. The local dynamic maps are represented as stars and dotted ellipses.

## 1.1 Thesis Objective

The aim of the thesis can be summarized with the following questions:

- What existing real-time fusion strategies can be utilised when fusing data from multiple vehicles in an urban environment?
- How can the effect of a varying time delay in the communication channel be minimized in terms of accuracy?
- How can tracks of the same object from multiple vehicles be associated with each other with high certainty?
- How can the algorithm detect if an object has left the field of view of all the vehicles and decide when to terminate the tracking of that object?
- Which algorithms can be utilized when performing fusion of already filtered data?
- Which motion models are most suitable for modeling a vehicle for tracking

purposes?

## 1.2 Related Work

The sub-problems in this thesis such as coping with communication delays, fusing probability densities (as opposed to fusing raw measurements) etc. are covered in the literature and different methods have been proposed to solve them. The complete problem of fusing data from multiple vehicles to get a coherent view of the environment has been treated in a master's thesis written by Johnny Ngu and Ronakorn Soponpunth [3]. In the solution proposed by the authors, an Information Kalman filter [4] was utilized which is formulated using inverse covariance matrices, i.e. information matrices. This filter was used together with a linear Constant Acceleration (CA) model and the inputs to the fusion server were assumed to consist of raw measurements. A comparison between the previous thesis and this one will be conducted in section 1.3 below.

### 1.2.1 Communication Delays

Out-of-sequence Measurement (OOSM) [5] and Out-of-sequence Track (OOST) are quite common when it comes to wireless data transfer. This means that packets containing measurements/data can be delayed such that they arrive at the central filtering unit in the wrong order. It is desirable to minimize the effect of the delays and OOSM/OOST when considering filter performance. Yaakov Bar-Shalom [6] and Keshu Zhang [7] discusses the OOSM problem where Bar-Shalom proposes a solution using "negative-time measurement update" where he assumes that the OOSM data is received within the last sampling interval. Subhash Challa et al. [8] extends the concept to also include OOST with data which is delayed more than one sample.

A trivial solution to this problem is to save all the data/LDMs and when an OOSM/OOST is received, the filter recalculates the estimates from the time the measurement/track was collected. This method is described by Mahendra Mallick et al. [9] and is called rollback. However, if there is a lot of OOSM/OOST, the rollback method requires a lot of computational power and might not be feasible in a real-time scenario involving a large number of actors.

### 1.2.2 Track-to-track Association

In cases where several actors detect the same object, the tracks from each actor needs to be fused. Before the fusion can be performed, a data association process needs to be done in order to detect if the observations are actually corresponding to a single hypothesized target. The topic of data association is discussed in [10]

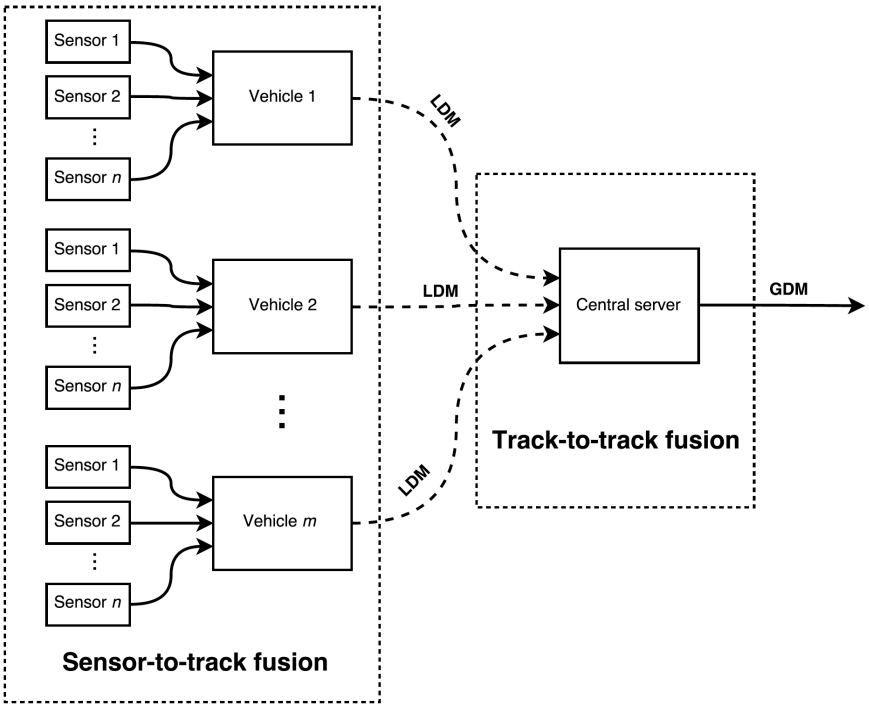



**Figure 1.3:** Flowchart of the information where each vehicle performs its local fusion to create an LDM which are then wirelessly communicated to the central server where the LDMs are fused into one GDM.

and [11] where the authors bring forth an association metric called Mahalanobis Distance (MD) which indicates how close two tracks are to each other.

### 1.2.3 Track-to-track Fusion

Each actor is equipped with sensors that locally fuses sensor measurements into tracks, this process is called Sensor-to-track Fusion (S2TF). These tracks are then communicated to a central server where they are fused again to create one global track list (GDM). This problem is called Track-to-track Fusion (T2TF) and is illustrated in Fig. 1.3. The T2TF problem is extensively covered in the literature where a wide variety of solutions are proposed [8], [12]–[15].

## 1.3 Major Contributions

The problem of fusing data from multiple vehicles to get a coherent view of the environment has been treated in a master's thesis project [3] done during 2016. In that thesis, a version of a Kalman Filter (KF) was used together with linear CA models. In this thesis, more accurate filters, the Unscented Kalman Filter (UKF) and Cubature Kalman Filter (CKF), are implemented and evaluated. In [3], the

tracks in each LDM was considered to be raw measurements. However, in reality they are already filtered tracks and hence regular filtering algorithms should not be used. In this thesis, the Information De-correlation (IDC) fusion algorithm is implemented and evaluated instead.

As a motion model, the Coordinated Turn (CT) model is used. This model is considered to be better than the CA model when it comes to describing the motion of a vehicle. However, since this model is non linear, a filter that can handle this non-linearity is needed. Hence the UKF and CKF is used which are commonly used in tracking problems. When considering the track association problem, the strategy in [3] was to use the Euclidean distance to determine if a measurement should be associated with a certain track. In this thesis another metric for data association, called the MD, is studied which incorporates the uncertainties of the tracks as well.

All of the major problems that is covered in this thesis has been studied before. However, there are no studies that covers the combination of all sub problems. That is T2TF, OOST, data association.

## 1.4 Demarcations

After the server has performed the fusion, the output (GDM) will in the future be used to control the involved actors by sending desired trajectories back to the vehicles from the server. This controller and the communication back to the actors is not considered in this thesis.

When considering the wireless communication channel, data packets may be lost during transmission. However, this is not treated in this thesis and it is instead assumed that all data packets will eventually be received by the server. Furthermore, the communication is assumed to be working properly and the implementation of the communication protocols is not considered in this thesis.

When the active actors send data to the central server, the data contains a timestamp to indicate when the data was collected. Each of the vehicles has its own clock which the time stamp is based on and when performing fusion on this data, these clocks needs to be synchronized. The process of synchronizing the clocks in all active actors is assumed to be done and is not considered in this thesis.

The traffic scenarios considered in this thesis is limited to a city area where the velocities of the actors are relatively low. Furthermore, the number of involved actors is less than five.

## 1.5 Thesis Outline

The remainder of this thesis is structured as follows. In Chapter 2 the theoretical background is introduced. The area of Bayesian estimation theory is introduced which in this case is focused on sigma point methods for state estimation. Furthermore, the theory regarding T2TF, data association and coordinate transformation is also covered.

In chapter 3 the proposed fusion algorithm is described in detail and pseudo-code for the different steps involved in the multi-target tracking process is presented.

Chapter 4 describes the scenario which the fusion algorithm is applied to and chapter 5 presents the results of the proposed algorithm.

In chapter 6, the designed fusion algorithm and results are discussed.

Finally, chapter 7 concludes the thesis, summarizes the results and brings forth possible further work that can be done within this area to further improve the algorithm.

# 2

# Theory

In this chapter, the basic theories used in the proposed fusion algorithm is presented. First, the basics in Bayesian estimation is presented where the UKF and CKF is described in more detail. Next, the topics of data association and coordinate transformation are introduced. The chapter finishes with a section regarding OOST to deal with the random delays in the communication channel.

## 2.1 Bayesian Estimation

A recursive method used for estimating parameters of interest is Bayesian estimation [16] which uses prior information about the parameters as well as input measurements/observations from one or more sensors to estimate the parameters. Typical sensors that are used for automotive applications are radar, camera, lidar and IMUs. The parameters of interest is collected into a state vector which is denoted $\mathbf{x}_k$ where common examples of parameters is position, velocity and heading of moving objects such as planes and vehicles. A more recent application of Bayesian estimation is to track surrounding objects of a vehicle for safety purposes which is well described by Lundgren in [17].

Bayesian estimation is based on some general assumptions where one of them is that the stochastic process of a systems state $\mathbf{x}_k$ always fulfills the Markov property. Here, $k$ denotes the time index. This property declares that future states only depend on the current state, not the previous states, i.e.

$$p(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{x}_{k-2}, \ldots, \mathbf{x}_0) = p(\mathbf{x}_k|\mathbf{x}_{k-1}). \tag{2.1}$$

Another important property that should be mentioned is that a measurement $\mathbf{z}_k$ taken at time $k$ only depends on the state at time $k$, i.e.

$$p(\mathbf{z}_k|\mathbf{x}_k, \mathbf{x}_{k-1}, \ldots, \mathbf{x}_0) = p(\mathbf{z}_k|\mathbf{x}_k). \tag{2.2}$$

The two basic properties explained is further illustrated in a Bayesian network in Fig. 2.1 where the relations between measurements and states over time is shown.

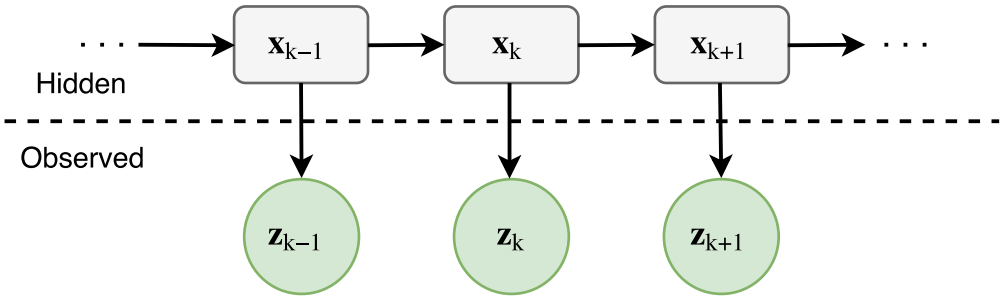



**Figure 2.1:** A Bayesian network where the conditional dependencies of the state evolution and between states and measurements are shown.

The goal with Bayesian filtering is to start with a prior state estimate $p(\mathbf{x}_{k-1}|\mathbf{z}_{1:k-1})$ where $\mathbf{z}_{1:k-1}$ denotes a collection of measurements from time index 1 to time index $k-1$ where $k$ is the current time step. The idea is to then use measurements to compute a posterior probability density function $p(\mathbf{x}_k|\mathbf{z}_{1:k})$ which summarizes all information of the state up to time $k$. When this density has been computed, the state estimate at time $k$ given information up to time $k$, denoted $\hat{\mathbf{x}}_{k|k}$ can be estimated based on an optimization criterion such as Mean Squared Error (MSE) or Maximum a Posteriori (MAP).

Bayesian estimation consists of two separable steps; prediction and measurement update. In the first step the predicted probability density function $p(\mathbf{x}_k|\mathbf{z}_{1:k-1})$ is calculated using the state transition density function, $p(\mathbf{x}_k|\mathbf{x}_{k-1})$ and the priori density function mentioned above. The equation used in this step is called the Chapman-Kolmogorov equation where the previously discussed Markov properties is used and is formulated as

$$\begin{aligned} p(\mathbf{x}_k|\mathbf{z}_{1:k-1}) &= \int p(\mathbf{x}_k, \mathbf{x}_{k-1}|\mathbf{z}_{1:k-1}) d\mathbf{x}_{k-1} \\ &= \int p(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{z}_{1:k-1}) p(\mathbf{x}_{k-1}|\mathbf{z}_{1:k-1}) d\mathbf{x}_{k-1} \\ &= \int p(\mathbf{x}_k|\mathbf{x}_{k-1}) p(\mathbf{x}_{k-1}|\mathbf{z}_{1:k-1}) d\mathbf{x}_{k-1} \end{aligned} \tag{2.3}$$

where $p(\mathbf{x}_k|\mathbf{x}_{k-1})$ is called the state transition. This transition is given by a mathematical motion/process model that describes how the states evolve over time. The state transition is defined as

$$\mathbf{x}_k = f_{k-1}(\mathbf{x}_{k-1}, \mathbf{v}_{k-1}) \tag{2.4}$$

where $f_{k-1}$ is the motion model and $\mathbf{v}_{k-1}$ is the process noise.

Some assumptions or simplifications are often done in Bayesian estimation. The process noise $\mathbf{v}_{k-1}$ is often assumed to be an additive, zero mean Gaussian random variable with covariance $\mathbf{Q}_{k-1}$, i.e.

$$\mathbf{v}_{k-1} \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_{k-1}).$$

Furthermore, the estimated probability density functions are also assumed to be Gaussian. The prior state estimate has the normal distribution defined as

$$p(\mathbf{x}_{k-1}|\mathbf{z}_{1:k-1}) = \mathcal{N}(\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{P}_{k-1|k-1}) \tag{2.5}$$

where $\hat{\mathbf{x}}_{k-1|k-1}$ is the state estimate from the previous time step (recursion), $\mathbf{P}_{k-1|k-1}$ is the corresponding covariance matrix and $\mathbf{z}_{1:k-1}$ is the measurements up to time $k-1$. The prior in Eq. 2.5 together with the motion model of the system is used to estimate the distribution at time $k$ given information up to time $k-1$ which is distributed as

$$p(\mathbf{x}_k|\mathbf{z}_{1:k-1}) = \mathcal{N}(\hat{\mathbf{x}}_{k|k-1}, \mathbf{P}_{k|k-1}). \tag{2.6}$$

The second part of Bayesian estimation, often called the measurement update step, is to include measurements into the state estimation. Here, the distribution of $\mathbf{x}_k$ is updated by using the information from the measurements $\mathbf{z}_k$ and the previously computed density function, $p(\mathbf{x}_k|\mathbf{z}_{1:k-1})$. This procedure is summarized by Bayes' rule as

$$\begin{aligned} p(\mathbf{x}_k|\mathbf{z}_{1:k}) &= p(\mathbf{x}_k|\mathbf{z}_k, \mathbf{z}_{1:k-1}) \\ &= \frac{p(\mathbf{z}_k|\mathbf{x}_k, \mathbf{z}_{1:k-1})p(\mathbf{x}_k|\mathbf{z}_{1:k-1})}{p(\mathbf{z}_k|\mathbf{z}_{1:k-1})} \\ &= \frac{p(\mathbf{z}_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{z}_{1:k-1})}{p(\mathbf{z}_k|\mathbf{z}_{1:k-1})} \end{aligned} \tag{2.7}$$

where $p(\mathbf{z}_k|\mathbf{x}_k)$ is called the likelihood and is modeled using a measurement model, denoted $h_k$. The denominator of Eq. 2.7, $p(\mathbf{z}_k|\mathbf{z}_{1:k-1})$, represents a normalization factor which guarantees that the density integrates to one. If this factor is ignored, the general equation becomes

$$\textit{posterior} \propto \textit{prior} \times \textit{likelihood}$$

which in words mean that by combining the information from the predicted prior density $p(\mathbf{x}_k|\mathbf{z}_{1:k-1})$ with additional information from the measurement model $p(\mathbf{z}_k|\mathbf{x}_k)$, the posterior density is obtained. The relationship between the states and measurements are generally described as

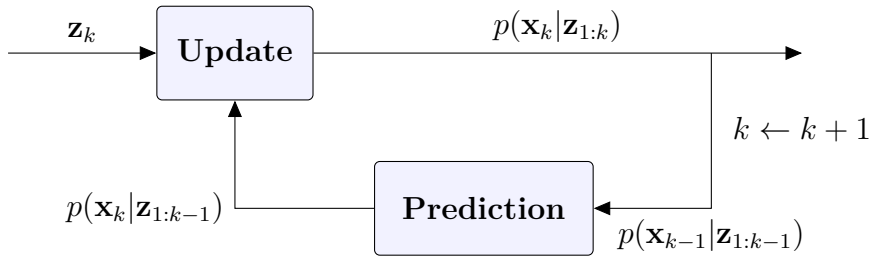



**Figure 2.2:** General Bayesian estimation scheme.

$$\mathbf{z}_k = h_k(\mathbf{x}_k, \mathbf{w}_k) \tag{2.8}$$

where $\mathbf{w}_k$ is the measurement noise which often is assumed to be additive and Gaussian distributed such that $\mathbf{w}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_k)$. The posterior density has now been computed and it is normally distributed with mean $\hat{\mathbf{x}}_{k|k}$ and covariance $\mathbf{P}_{k|k}$, i.e.

$$p(\mathbf{x}_k|\mathbf{z}_{1:k}) = \mathcal{N}(\hat{\mathbf{x}}_{k|k}, \mathbf{P}_{k|k}). \tag{2.9}$$

This general filtering procedure is illustrated in Fig. 2.2 and most filtering methods follow this scheme. However, they estimate the distributions differently. As can be seen, the procedure starts with a prior distribution $p(\mathbf{x}_{k-1}|\mathbf{z}_{1:k-1})$ that is sent to the prediction step which computes the predicted distribution $p(\mathbf{x}_k|\mathbf{z}_{1:k-1})$. Next a measurement is taken into account in the update step which computes $p(\mathbf{x}_k|\mathbf{z}_{1:k})$ and the procedure starts again.

There are several methods that estimates these distributions and the most common is the Kalman Filter (KF) [16]. The KF is considered as an optimal method for estimating parameters when all models included are linear and the measurement and process noise is additive and Gaussian distributed. However, since the mathematical model describing the motion of the states as well as the measurement models are often non-linear, several extensions of the KF have been developed to handle this. Some widely used examples are the Extended Kalman Filter (EKF), UKF and CKF. In this project the UKF and CKF will mainly be studied and they are further described in the following sections.

### 2.1.1 The Unscented Kalman Filter

The UKF [18] is a well known estimation method used when the models included are non-linear. It outperforms the EKF significantly in many applications when the models are highly non-linear without compromising on computational power. The EKF is based on linearization via Taylor series expansions of the non-linear motion

and measurement models. However, when linearizing, the behaviour of the non-linear models can not be captured accurately. The UKF uses something referred to as sigma-points ($\sigma$-points) instead. By introducing $\sigma$-points, the distributions can be estimated with higher accuracy. This is explained by the fact that multiple points that are all propagated through the non-linear motion model $f_{k-1}$ will capture the posterior mean and covariance better. As with all recursive Bayesian estimation, the UKF consists of two separate steps, one prediction step and one measurement update step.

### Prediction

In the first step, a set of $2n+1$ $\sigma$-points are calculated as

$$\boldsymbol{\chi}_{k-1}^{(0)} = \hat{\mathbf{x}}_{k-1|k-1} \tag{2.10a}$$

$$\boldsymbol{\chi}_{k-1}^{(i)} = \hat{\mathbf{x}}_{k-1|k-1} + \gamma(\mathbf{P}_{k-1|k-1}^{1/2})_i, \quad i = 1, 2, ..., n \tag{2.10b}$$

$$\boldsymbol{\chi}_{k-1}^{(i+n)} = \hat{\mathbf{x}}_{k-1|k-1} - \gamma(\mathbf{P}_{k-1|k-1}^{1/2})_i, \quad i = 1, 2, ..., n \tag{2.10c}$$

where $n$ is the number of states and the first $\sigma$-point is set to the state estimate $\hat{\mathbf{x}}_{k-1|k-1}$ from the previous filter recursion. The rest of the $\sigma$-points is spread around this estimate by a scaling parameter $\gamma$ and $(\mathbf{P}_{k-1|k-1}^{1/2})_i$ which is the $i$-th column of the lower triangular Cholesky decomposition of the covariance matrix $\mathbf{P}_{k-1|k-1}$ from the previous filter recursion.

Once the $\sigma$-points have been calculated, they are propagated through the non-linear motion model $f_{k-1}$ and the predicted state estimate is the weighted sum of the resulting sigma points. The predicted estimate and covariance matrix is calculated as

$$\hat{\mathbf{x}}_{k|k-1} \approx \sum_{i=0}^{2n} f_{k-1}(\boldsymbol{\chi}_{k-1}^{(i)}) W_i^m \tag{2.11}$$

$$\mathbf{P}_{k|k-1} \approx \mathbf{Q}_{k-1} + \sum_{i=0}^{2n} (f(\boldsymbol{\chi}_{k-1}^{(i)}) - \hat{\mathbf{x}}_{k|k-1})(\cdot)^T W_i^c \tag{2.12}$$

where $W_i^m$ and $W_i^c$ are weights associated with the $\sigma$-points and $\mathbf{Q}_{k-1}$ is the covariance matrix of the process noise.

There are several methods for selecting the weights in Eq. 2.11 and 2.12. One of the most widely used methods is based on three parameters that can be tuned to change the spread of the $\sigma$-points as in [19]. These parameters are $\alpha$, $\beta$ and $\kappa$ and

the weights are then defined as

$$W_0^m = \frac{\lambda}{\lambda + n} \tag{2.13a}$$

$$W_0^c = \frac{\lambda}{\lambda + n} + 1 - \alpha^2 + \beta \tag{2.13b}$$

$$W_i^m = \frac{1}{2(\lambda + n)}, \quad i = 1, 2, ..., 2n \tag{2.13c}$$

$$W_i^c = W_i^m, \quad i = 1, 2, ..., 2n \tag{2.13d}$$

where $\lambda = \alpha^2(n+\kappa) - n$. With this selection of weights, $\gamma$ is defined as $\gamma = \sqrt{n + \lambda}$. A normal selection is to set $\alpha$ small, that is $0 < \alpha \leq 1$, $\beta$ equal to 2 is optimal for Gaussian distributions and the scaling parameter $\kappa$ is usually set to zero.

A second approach of computing the weights suggested by Julier and Uhlmann in [18] is

$$W_0^m = W_0^c = 1 - \frac{n}{3} \tag{2.14a}$$

$$W_i^m = W_i^c = \frac{1}{6}, \quad i = 1, 2, ..., 2n \tag{2.14b}$$

where in this case, $\gamma = \sqrt{3}$.

### Measurement Update

Once the predicted moments $\hat{\mathbf{x}}_{k|k-1}$ and $\hat{\mathbf{P}}_{k|k-1}$ has been computed, measurements are used to update the state estimate. Similarly as in the prediction step, a set of $2n + 1$ $\sigma$-points are computed as

$$\boldsymbol{\chi}_k^{(0)} = \hat{\mathbf{x}}_{k|k-1} \tag{2.15a}$$

$$\boldsymbol{\chi}_k^{(i)} = \hat{\mathbf{x}}_{k|k-1} + \gamma(\mathbf{P}_{k|k-1}^{1/2})_i, \quad i = 1, 2, ..., n \tag{2.15b}$$

$$\boldsymbol{\chi}_k^{(i+n)} = \hat{\mathbf{x}}_{k|k-1} - \gamma(\mathbf{P}_{k|k-1}^{1/2})_i, \quad i = 1, 2, ..., n. \tag{2.15c}$$

Based on these $\sigma$-points, the predicted measurement $\hat{\mathbf{z}}_{k|k-1}$ and corresponding covariance matrix $\mathbf{P}_{zz}$ is calculated by a weighted summation of the $\sigma$-points propagated through the measurement model $h_k$ as

$$\hat{\mathbf{z}}_{k|k-1} \approx \sum_{i=0}^{2n} h_k(\boldsymbol{\chi}_k^{(i)}) W_i^m \tag{2.16}$$

$$\mathbf{P}_{zz} \approx \mathbf{R}_k + \sum_{i=0}^{2n} (h(\boldsymbol{\chi}_k^{(i)}) - \hat{\mathbf{z}}_{k|k-1})(\cdot)^T W_i^c. \tag{2.17}$$

The next step is to compute the cross-covariance between the state and measurements:

$$\mathbf{P}_{xz} \approx \sum_{i=0}^{2n} (\boldsymbol{\chi}_k^{(i)} - \hat{\mathbf{x}}_{k|k-1})(h(\boldsymbol{\chi}_k^{(i)}) - \hat{\mathbf{z}}_{k|k-1})^T W_i^c. \tag{2.18}$$

The cross-covariance is then used in order to compute the UKF Kalman gain, $\mathbf{K}_k$:

$$\mathbf{K}_k \approx \mathbf{P}_{xz}\mathbf{P}_{zz}^{-1}. \tag{2.19}$$

In the final part of the measurement update step, the measurement $\mathbf{z}_k$ is used to update the state estimate and covariance by using the previously computed moments from Eq. 2.16-2.18 and the Kalman gain in Eq. 2.19. The state and covariance estimate update is defined as

$$\hat{\mathbf{x}}_{k|k} \approx \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k(\mathbf{z}_k - \hat{\mathbf{z}}_{k|k-1}) \tag{2.20}$$

$$\mathbf{P}_{k|k} \approx \mathbf{P}_{k|k-1} + \mathbf{K}_k\mathbf{P}_{zz}\mathbf{K}_k^T. \tag{2.21}$$

When performing the measurement update, the weights, $W_i^m$ and $W_i^c$, and the scaling parameter, $\gamma$, is the same as in the prediction step.

### 2.1.2 The Cubature Kalman Filter

One of the drawbacks with the UKF is that in most cases when selecting the weights, the first weight $W_0$ ends up negative which could have a negative impact on the filter performance. When the first weight is negative, the computed covariance might not be positive semi-definite [20] and thereby not a true covariance. This will cause problems in the next time step of the filter due to the fact that the Cholesky decomposition requires that the input matrix is positive semi-definite. However, an alternative filter called CKF [20] can be utilized which does not suffer from this problem. This filter performs well for both high and low-dimensional systems and it is very similar to the UKF with the difference that the first $\sigma$-point is not included, hence it only uses $2n$ $\sigma$-points. This is equal to setting the first weight in a UKF equal to zero. The weights are selected such that all $\sigma$-points are weighted equally, i.e.

$$W_i^m = W_i^c = \frac{1}{2n}, \quad i = 0, 1, ..., 2n-1. \tag{2.22}$$

The scaling factor that is used to spread the sigma points around the mean is set as $\gamma = \sqrt{n}$ for the CKF as in [16]. As opposed to the UKF, there are no tuning parameters for the CKF.

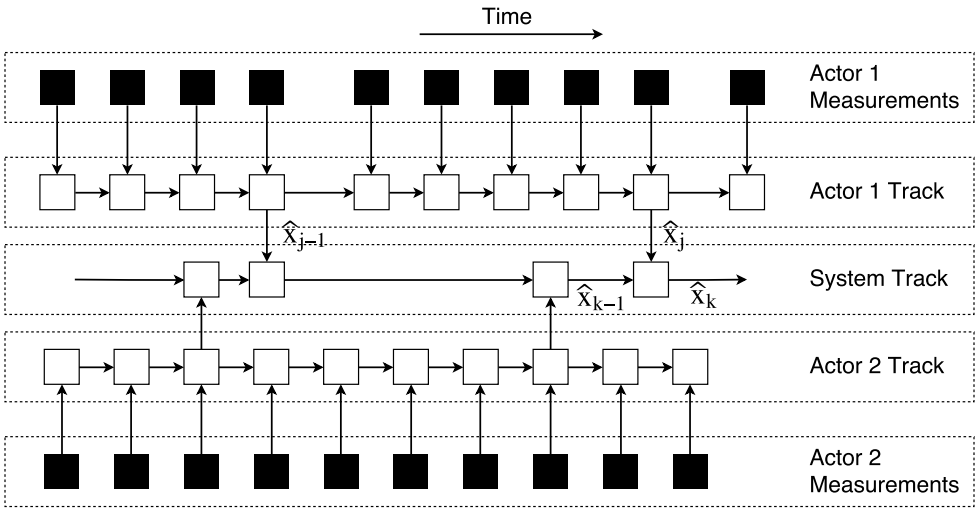



**Figure 2.3:** Illustration of the dependencies between tracks in hierarchical Track-to-track Fusion.

### 2.1.3 Track-to-track Fusion

The T2TF problem is quite different from regular sensor fusion where raw sensor measurements are used in the filter update. The problem here is to fuse already filtered data, that is to fuse tracks. This is illustrated in Fig. 2.3 where two actors perform their own local fusion to obtain so called local tracks. These tracks are then sent to the central system where they are fused to form system tracks. The system tracks contains state estimates as well as covariance matrices for all objects.

When performing T2TF it is important to be observant of the correlation problem that occurs. In regular fusion algorithms where the input is raw sensor measurements, the measurement errors are assumed to be uncorrelated Gaussian random variables. However, as discussed in [10], [21] and [22], this assumption does not hold for T2TF. Since the actors perform local fusion, the estimation error for a track produced at a certain time is correlated to the previously produced track estimation error. This phenomenon is illustrated in Fig. 2.3 where $\hat{\mathbf{x}}_j$ is correlated to and includes information from $\hat{\mathbf{x}}_{j-1}$. This implies that some data from actor 1 in Fig. 2.3 is correlated with the system track, e.g. $\hat{\mathbf{x}}_{k-1}$ is correlated with $\hat{\mathbf{x}}_j$ which both contains information from $\hat{\mathbf{x}}_{j-1}$. This means that a regular measurement update in a Bayesian estimator can not be performed since one assumption made is that the input measurement errors are uncorrelated over time. There exist alternative methods which deal with such problems, such as *Equivalent Measurement* or IDC described below.

### Equivalent Measurement

The method described in [10], called Equivalent Measurement, is an algorithm that is used to de-correlate the track to be fused from the global track. An equivalent measurement is computed based on the current as well as the previous track sent to the central system from the same actor. To compute the equivalent measurement, the previously sent track needs to be propagated/predicted in order to temporally align the tracks. The equivalent measurement $\mathbf{u}_k$ and the equivalent covariance matrix $\mathbf{U}_k$ are defined as

$$\mathbf{u}_k = \hat{\mathbf{x}}_{j-1} + \mathbf{P}_{j-1}(\mathbf{P}_{j-1} - \mathbf{P}_j)^{-1}(\hat{\mathbf{x}}_j - \hat{\mathbf{x}}_{j-1}) \tag{2.23}$$

$$\mathbf{U}_k = \mathbf{P}_{j-1}(\mathbf{P}_{j-1} - \mathbf{P}_j)^{-1}\mathbf{P}_j. \tag{2.24}$$

When the equivalent measurement has been computed, a regular KF update can be performed since the actor track now is uncorrelated with the system track. The equivalent measurement error is thereby uncorrelated with the system tracks estimation error and a regular KF update estimates the posterior distribution of the system track estimate as

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_{k-1} + \mathbf{P}_{k-1}(\mathbf{P}_{k-1} + \mathbf{U}_k)^{-1}(\mathbf{u}_k - \hat{\mathbf{x}}_{k-1}) \tag{2.25}$$

$$\mathbf{P}_k = \mathbf{P}_{k-1} - \mathbf{P}_{k-1}(\mathbf{P}_{k-1} + \mathbf{U}_k)^{-1}\mathbf{P}_{k-1}. \tag{2.26}$$

### Information De-correlation

Another method for fusing tracks which is described in [10] is called IDC (or sometimes also called information matrix fusion). As with the equivalent measurement, the idea with IDC is to only use the new information received from the local actor. The IDC uses information matrices instead of regular covariance matrices to identify the common information for the two tracks. An information matrix is simply the inverse of a covariance matrix and the IDC methods is now defined as

$$\hat{\mathbf{x}}_k = \mathbf{P}_k(\mathbf{P}_{k-1}^{-1}\hat{\mathbf{x}}_{k-1} + \mathbf{P}_j^{-1}\hat{\mathbf{x}}_j - \mathbf{P}_{j-1}^{-1}\hat{\mathbf{x}}_{j-1}) \tag{2.27}$$

$$\mathbf{P}_k = (\mathbf{P}_{k-1}^{-1} + \mathbf{P}_j^{-1} - \mathbf{P}_{j-1}^{-1})^{-1} \tag{2.28}$$

where $\hat{\mathbf{x}}_{j-1}$ and $\mathbf{P}_{j-1}$ is the previous state estimate and covariance from the local track that was sent to the central server (propagated to a common time).

The benefit with IDC is that the algorithm is very simple to implement. The algorithm only requires that the previous state and covariance is stored in memory. The method is considered to be almost optimal (optimal in the sense that it produces the same results as if the raw measurements where fused directly [10]) when the process noise is non-existing which is not the case in this problem. However, if the

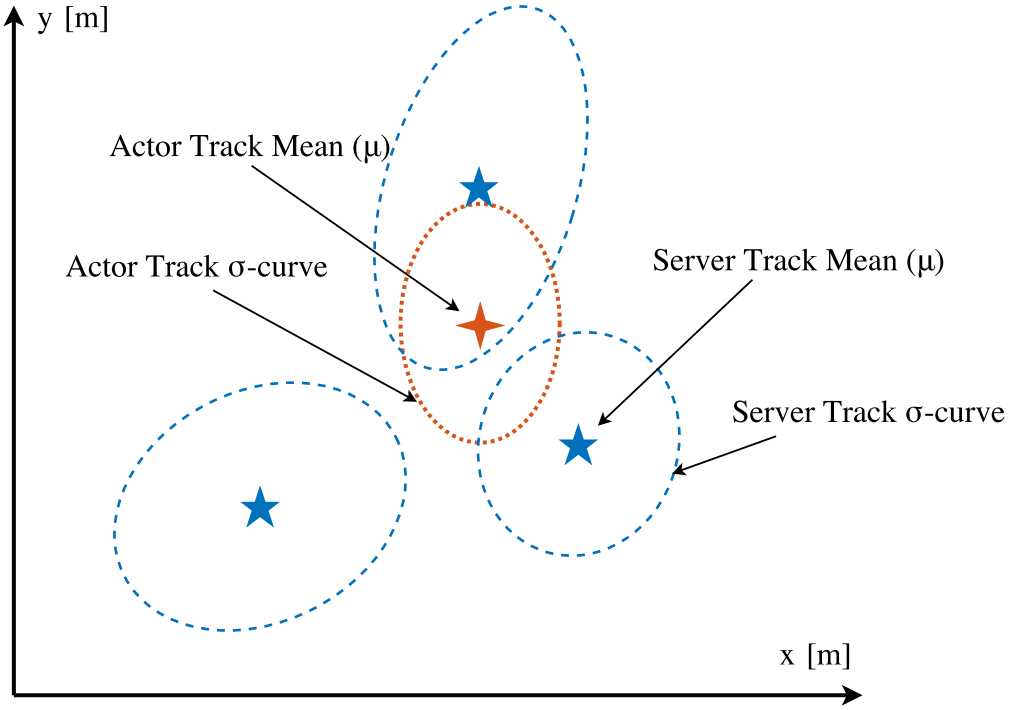



**Figure 2.4:** An illustration of the track association problem. This example includes one actor track (red) and three system tracks (blue).

process noise is small enough and the local tracks are received at a high rate, the algorithm has a high performance as shown in [10].

It should be mentioned that Eq. 2.27–2.28 is equivalent to the equations presented in Eq. 2.23–2.26. Thereby the equivalent measurement algorithm and the IDC algorithm should produce the exact same result. The advantages of IDC is that the method can be used when the distributions are non-Gaussian and it does not suffer from the numerical error that can occur when calculating equivalent measurements.

## 2.2 Data Association Metrics

The problem of associating one local track (a track from the local fusion done in each active actor) with a system track is sometimes referred to as multi-target track association [10], [11]. The problem is illustrated in Fig. 2.4 where a local track is to be associated with one of the server tracks. There are several association metrics that can be used to determine if two tracks represent the same object. Two of these metrics is described in more detail in this section.

### 2.2.1 Euclidean Distance

One simple association metric is described in [23] and is called the Euclidean Distance (ED). This metric is defined as

$$\gamma_{ED} = \sqrt{(\hat{\mathbf{x}}_k - \hat{\mathbf{x}}_j)^T(\hat{\mathbf{x}}_k - \hat{\mathbf{x}}_j)}, \tag{2.29}$$

where $\hat{\mathbf{x}}_k$ and $\hat{\mathbf{x}}_j$ is the state estimate of the two tracks. This metric only looks at the difference between the state estimates. This metric does not consider the covariance of the track estimates which is a drawback with this method. A common approach is to only use the lateral and longitudinal position in the state vector $\hat{\mathbf{x}}_k$ and $\hat{\mathbf{x}}_j$ when calculating the ED. This metric might be good enough for data association when the distance between two tracks is large but can perform poorly when the tracks have trajectories close to each other.

### 2.2.2 Mahalanobis Distance

Another association metric discussed in [10], [11] and [23] is the Mahalanobis Distance (MD) which is defined as

$$\gamma_{MD} = \sqrt{(\hat{\mathbf{x}}_k - \hat{\mathbf{x}}_j)^T(\mathbf{P}_k + \mathbf{P}_j)^{-1}(\hat{\mathbf{x}}_k - \hat{\mathbf{x}}_j)} \tag{2.30}$$

where $\hat{\mathbf{x}}_k$ and $\hat{\mathbf{x}}_j$ is the state estimate of the two tracks and $\mathbf{P}_k$ and $\mathbf{P}_j$ are the corresponding covariance matrices. This metric has the benefit that it normalizes the euclidean distance with the use of the covariances of each track estimate.

An example of the track association problem which can be solved using Mahalanobis distance is shown in Fig. 2.4. In this two-dimensional case, the server receives a track from an actor which is to be associated with one of the three existing system tracks. The dotted ellipses in the figure represents $\sigma$-level curves of the tracks, i.e. track uncertainties, which are taken into account when calculating the Mahalanobis distance.

To further illustrate how the performance of the ED and MD differs, a simple example is created. In this example, the only state included in the state vector is $x$ position to clearly illustrate the difference between the metrics. In Fig. 2.5, the distributions for two server tracks as well and one track from a LDM is illustrated. In the legend of the figure, the ED and MD is calculated as well. As can be seen the ED to the server tracks is the same for both distributions (0.5). However, when using the MD, the distance is shorter between the local track and the second server track since the distributions overlap more.

## 2.3 Coordinate Transformation

All active actors send their own state estimate as well as the state estimate of the objects it can detect to the central server, this is what is called an LDM. The ego states for the vehicles are expressed in the global coordinate frame $\{W\}$ illustrated in Fig. 2.6. Furthermore, the measured objects are expressed in a body-fixed coordinate frame $\{E\}$. At the fusion server, the output is expected to be in the global coordinate frame $\{W\}$, hence a coordinate change needs to be done to all tracked objects for each actor.

### 2.3.1 State Vectors

The coordinate change for the linear motion states (position and velocities) contains two steps: rotation and translation. A rotation matrix $\mathbf{R}$ is computed based on the heading angle $\theta_{a,W} = \arctan\left(\frac{v^y_{a,W}}{v^x_{a,W}}\right)$ of the actor and is defined as

$$\mathbf{R} = \begin{bmatrix} \sin(\theta_{a,W}) & \cos(\theta_{a,W}) \\ -\cos(\theta_{a,W}) & \sin(\theta_{a,W}) \end{bmatrix}. \tag{2.31}$$

A translation where the position of the actor is added to the states is also done in order to compute the object states in the global frame. The transformation is defined as

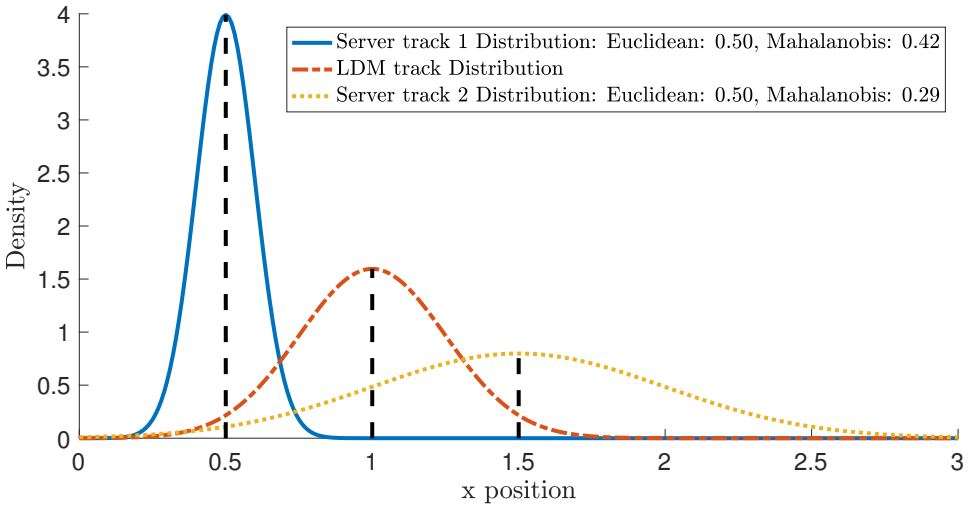



**Figure 2.5:** Simple comparison of the Euclidean and Mahalanobis distance for a one-dimensional case.

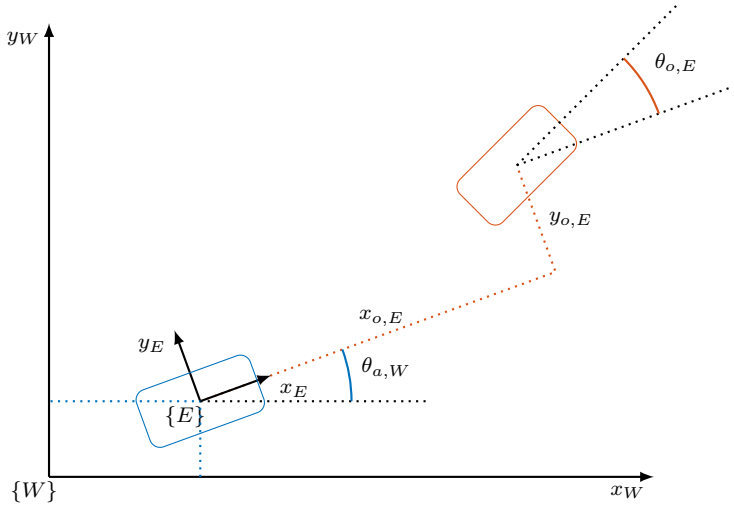



**Figure 2.6:** Definition of coordinate frames.

$$\begin{bmatrix} x_{o,W} \\ y_{o,W} \end{bmatrix} = \mathbf{R} \begin{bmatrix} x_{o,E} \\ y_{o,E} \end{bmatrix} + \begin{bmatrix} x_{a,W} \\ y_{a,W} \end{bmatrix} \tag{2.32}$$

where $x_{o,E}$ and $y_{o,E}$ is the states of the detected object relative to the body fixed coordinate system $\{E\}$ and $x_{a,W}$ and $y_{a,W}$ is the global coordinates for the actor.

The procedure of transforming the velocities are very similar to the transformation of the position and is performed as

$$\begin{bmatrix} v^x_{o,W} \\ v^y_{o,W} \end{bmatrix} = \mathbf{R} \begin{bmatrix} v^x_{o,E} \\ v^y_{o,E} \end{bmatrix} + \begin{bmatrix} v^x_{a,W} \\ v^y_{a,W} \end{bmatrix}. \tag{2.33}$$

The yaw rate $\omega_{o,W}$ of an object in the global coordinate frame is simply the relative yaw rate of the object, i.e.

$$\omega_{o,W} = \omega_{o,E}. \tag{2.34}$$

### 2.3.2 Covariance Matrices

In addition to the state estimates, the covariance matrices for the object tracks in an LDM needs to be expressed in the global coordinate system $\{W\}$. The covariance matrix of the relative positions $\mathbf{P}^{\text{pos}}_{o,E}$ can be formulated as

$$\mathbf{P}^{\text{pos}}_{o,E} = \begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix} \tag{2.35}$$

where $\sigma_x^2$ and $\sigma_y^2$ are the variances for $x$- and $y$-position of the object in the relative coordinate frame $\{E\}$ respectively. The covariance matrix describing the uncertain-

ties of the actors own state estimate is similarly defined as

$$\mathbf{P}_{a,W}^{\text{pos}} = \begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix} \tag{2.36}$$

where $\sigma_x^2$ and $\sigma_y^2$ are the variances for $x$- and $y$-position in the global coordinate frame $\{W\}$.The matrix in Eq. 2.35 is transformed to the global frame $\{W\}$ by using the rotation matrix $\mathbf{R}$ from Eq. 2.31 and the covariance matrix $\mathbf{P}_{a,W}^{\text{pos}}$ defined in Eq. 2.36 from the actor [24]:

$$\mathbf{P}_{o,W}^{\text{pos}} = \mathbf{R}\mathbf{P}_{o,E}^{\text{pos}}\mathbf{R}^T + \mathbf{P}_{a,W}^{\text{pos}}. \tag{2.37}$$

The velocity covariance for the objects is transformed similarly as with the position. The covariance matrix of the relative velocities $\mathbf{P}_{o,E}^{\text{vel}}$ is defined as

$$\mathbf{P}_{o,E}^{\text{vel}} = \begin{bmatrix} \sigma_{v^x}^2 & 0 \\ 0 & \sigma_{v^y}^2 \end{bmatrix} \tag{2.38}$$

where $\sigma_{v^x}^2$ and $\sigma_{v^y}^2$ is the variance in $x$- and $y$-velocity for the object in the local coordinate frame $\{E\}$. The covariance matrix describing the uncertainties in the ego state estimate for velocities is defined as

$$\mathbf{P}_{a,W}^{\text{vel}} = \begin{bmatrix} \sigma_{v^x}^2 & 0 \\ 0 & \sigma_{v^y}^2 \end{bmatrix} \tag{2.39}$$

where $\sigma_{v^x}^2$ and $\sigma_{v^y}^2$ are the variance in $x$- and $y$-velocity in the global coordinate frame $\{W\}$. Given this information, the velocity covariance for the objects in the global coordinate frame $\{W\}$ is calculated as

$$\mathbf{P}_{o,W}^{\text{vel}} = \mathbf{R}\mathbf{P}_{o,E}^{\text{vel}}\mathbf{R}^T + \mathbf{P}_{a,W}^{\text{vel}}. \tag{2.40}$$

Regarding the transformation of uncertainties in the rotational motion (yaw rate), the variances are simply added together as

$$\sigma_{\omega_{o,W}}^2 = \sigma_{\omega_{o,E}}^2 + \sigma_{\omega_{a,W}}^2. \tag{2.41}$$

## 2.4 Out-of-sequence Tracks

One of the main challenges with this project is that all data from the vehicles is transmitted wirelessly to the fusion server. As with all wirelsess communication, this introduces a delay. The delays are random which might lead to something

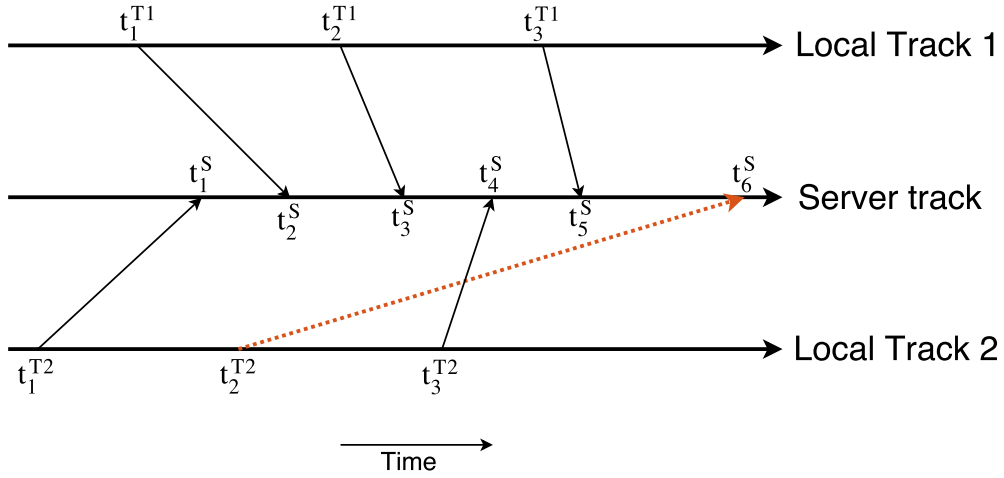



**Figure 2.7:** Data sent from two actors to the central server with varying time delay causing an Out-of-sequence Track.

referred to as OOST [22]. This means that two tracks from one actor might arrive at the fusion server in the wrong order which is illustrated in Fig. 2.7 where a data packet created at time $t_2^{T2}$ containing a LDM is delayed such that it arrives at the fusion server after the data collected at time $t_3^{T2}$ has been received.

When dealing with OOST, the optimal solution (meaning that all available information is used) is something referred to as rollback. Since the time stamp in the LDMs are assumed to be in absolute time, OOSTs can easily be detected and when this occurs, the filter goes back and recomputes all filter outputs for the server track from the time of the OOST up to current server time. By doing this, it is guaranteed that all available information is used at all times which produces the best state estimate for the server track.

However, as the number of OOST grows, so does the computational complexity since it involves a significant amount of re-computation. In addition to the computational complexity, previous track estimates from the LDMs that has arrived to the fusion server as well as previous server track estimates needs to be stored in memory.

# 3

# System Description

The designed fusion algorithm was constructed by smaller subsystems which perform specific tasks. These tasks are coordinate transformation, data association, data fusion, track management and output prediction. Together they form the fusion server which fuses several LDMs in order to create one GDM. The steps involved (including the wireless transmission) are illustrated in Fig. 3.1 where the dotted arrows correspond to the wireless communication between the actors and the fusion server.

The tracks in each LDM needs to be compared to the fusion server tracks to detect if the local and global tracks correspond to the same hypothesized target. If one of the local tracks can not be associated with any of the server tracks, a new global track is created for the new object. When all data has been associated (or new tracks has been created) it is fused together with the server tracks.

Before the association can be performed, the tracks need to be defined in the same global coordinate system. This is done by performing a coordinate transformation of the state estimates as well as the covariance matrices from local, body-fixed coordinate systems to a global coordinate frame.

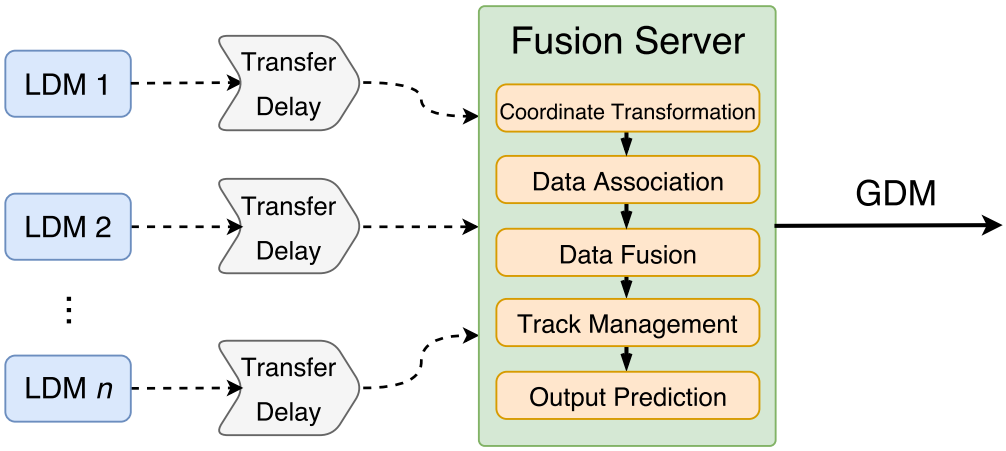



**Figure 3.1:** Block diagram of the proposed data fusion algorithm including the wireless communication.

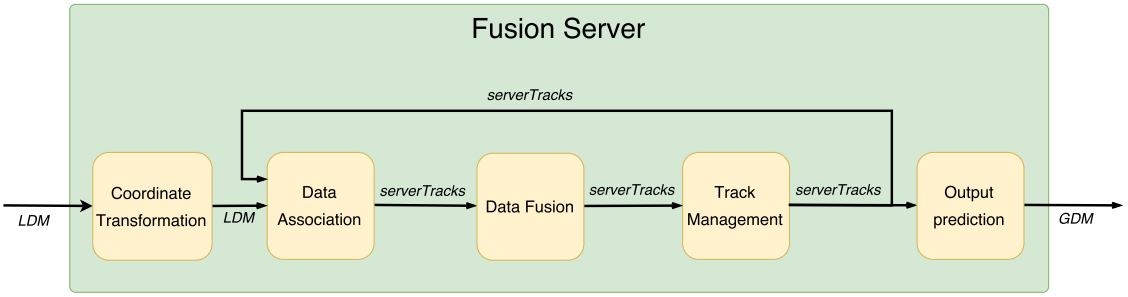



**Figure 3.2:** Fusion server implementation diagram.

Furthermore, a track management algorithm was implemented whose purpose is to terminate a server track if no association is done between the track and any of the local tracks for a certain amount of time. This occurs when an object has left the field of view of the active actors and no measurements can be made of that object. The server resources can thus be released to track other objects in the environment.

The output from the fusion server, a GDM containing the server tracks, will always lag in time since one server track has the same time stamp as the last local track which was associated to and fused with it. Therefore a final subsystem that makes a final prediction was implemented such that each server track contains a track estimate for the current time.

The algorithm designed in this thesis was implemented and evaluated in a Simulink [25] model created by Volvo Car Corporation called Simulation Platform for Active Safety (SPAS). As the name suggests, the model is used to simulate traffic scenarios and evaluate new active safety functions.

Fig. 3.2 further illustrates the steps performed in the fusion server. At each time step in the fusion server, a set of LDMs from the active actors arrives and after performing all involved steps a final GDM is produced. All steps involved are described in more detail below.

## 3.1 Motion Model

In several of the subsystems of the fusion server, both the local and server tracks needs to be aligned in time, i.e. the distributions need to be predicted. In order to accurately perform the prediction step of a Bayesian filtering algorithm, a good motion model is needed. Most systems can be described with a mathematical model which is an approximation of how the system behaves. A more accurate model will contribute to more accurate predictions. There exist several models to describe the motion of a vehicle, such as the Constant Velocity (CV), CA, CT [26], [27] and bicycle model [28]. In this thesis, the states that are of interest are global positions $x_k$, $y_k$, velocities $v_k^x$, $v_k^y$ as well as the yaw rate $\omega_k$ for the objects. Thus, the state vector used is defined as

$$\mathbf{x}_k = \begin{bmatrix} x_k & y_k & v_k^x & v_k^y & \omega_k \end{bmatrix}^T. \tag{3.1}$$

Previously mentioned motion models such as the CV and CA are great for many applications. They are linear, simple to implement and can in some cases model the movement of an object accurately. However, they are not suitable in this project since they do not represent the movement of a vehicle precisely. For example, the CV and CA models allow the vehicles to move sideways which contradicts the general behaviour of a vehicle (in normal driving conditions). Neither do they take the rotational motion (yaw rate) into consideration which means that these models are not precise when objects are turning. For this reason, the CT model was studied and when using Cartesian coordinates, the continuous time version of that model is defined as

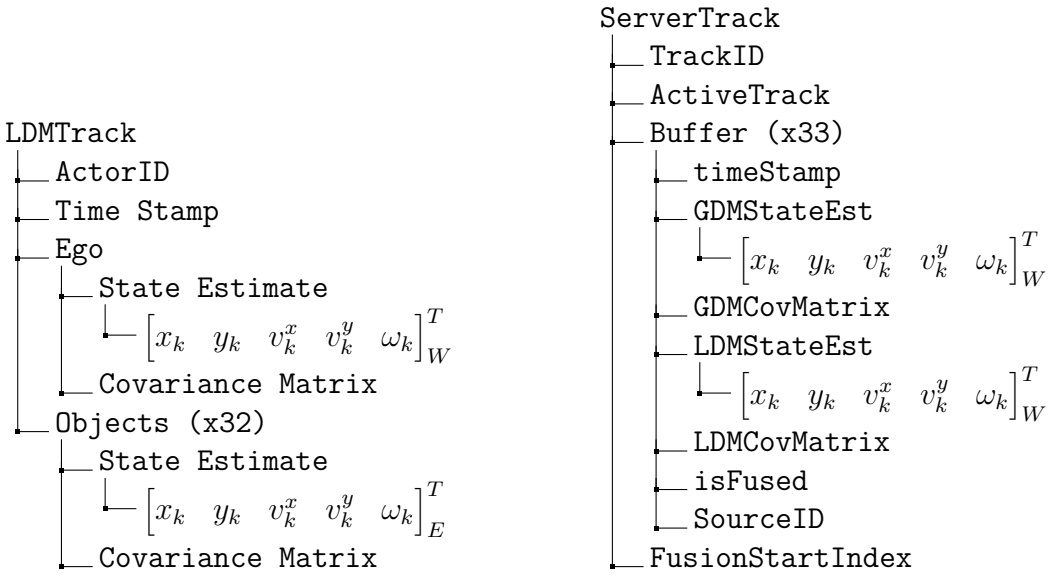
$$\begin{bmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{v^x}(t) \\ \dot{v^y}(t) \\ \dot{\omega}(t) \end{bmatrix} = \begin{bmatrix} v^x(t) \\ v^y(t) \\ -v^y(t)\omega(t) \\ v^x(t)\omega(t) \\ 0 \end{bmatrix} + \tilde{\mathbf{q}}(t) \tag{3.2}$$

where $\tilde{\mathbf{q}}(t)$ is the continuous time process noise. The discrete time version of Eq. 3.2 was obtained as in [27] which yields

$$\mathbf{x}_k = \begin{bmatrix} x_k \\ y_k \\ v_k^x \\ v_k^y \\ \omega_k \end{bmatrix} = \begin{bmatrix} x_{k-1} + \frac{v_{k-1}^x}{\omega_{k-1}} \sin(\omega_{k-1}T) - \frac{v_{k-1}^y}{\omega_{k-1}}(1 - \cos(\omega_{k-1}T)) \\ y_{k-1} + \frac{v_{k-1}^x}{\omega_{k-1}}(1 - \cos(\omega_{k-1}T)) + \frac{v_{k-1}^y}{\omega_{k-1}} \sin(\omega_{k-1}T) \\ v_{k-1}^x \cos(\omega_{k-1}T) - v_{k-1}^y \sin(\omega_{k-1}T) \\ v_{k-1}^x \sin(\omega_{k-1}T) + v_{k-1}^y \cos(\omega_{k-1}T) \\ \omega_{k-1} \end{bmatrix} + \mathbf{q}_{k-1} \tag{3.3}$$

where $T$ is the prediction time and $\mathbf{q}_{k-1}$ is the discretized process noise which is a zero mean, normally distributed random variable with covariance $\mathbf{Q}_{k-1}$. Assuming that the process noises is additive, Gaussian as well as uncorrelated with each other, the covariance matrix $\mathbf{Q}_{k-1}$ is defined as in [27] where $\sigma_{V_x}^2$, $\sigma_{V_x}^2$ and $\sigma_\omega^2$ is the variance in the velocities and yaw rate respectively:

$$\mathbf{Q}_{k-1} = \mathbf{G} \begin{bmatrix} \sigma_{V_x}^2 & 0 & 0 \\ 0 & \sigma_{V_y}^2 & 0 \\ 0 & 0 & \sigma_\omega^2 \end{bmatrix} \mathbf{G}^T, \qquad \text{where} \quad \mathbf{G} = \begin{bmatrix} \frac{T^2}{2} & 0 & 0 \\ 0 & \frac{T^2}{2} & 0 \\ T & 0 & 0 \\ 0 & T & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{3.4}$$




**(a)** LDM **(b)** serverTrack

**Figure 3.3:** Data definitions.

## 3.2 Data Definitions

The output from the first three subsystems in Fig. 3.2 is a collection of a data structure referred to as a *ServerTrack* which contains the data shown in Fig. 3.3b where the subscript of the state vectors indicates which coordinate frame the states are defined in. This data structure is internal in the fusion server and works as an internal buffer which stores previous input data and previous server state estimates with corresponding time stamps. Furthermore, the data association need information about the currently active server tracks in order to associate the local tracks from the LDMs with the server tracks. Therefore the *ServerTracks* are fed back to the data association at all times. In SPAS, the maximum number of actors is 32, thereby the maximum size of *ServerTrack* was set to the same size. Each track gets an individual ID number and if the track is active, a flag is set to one to indicate this and zero otherwise.

The *ServerTrack* contains a buffer that stores all associated tracks from the LDMs (ordered in time). When new associated data is inserted into the buffer, a variable that stores where in the buffer the fusion process should start is changed. The buffer size was set to the same size as the maximum number of actors plus one, i.e. 33. This ensures that if data is received from all actors within one sample time, the buffer is large enough to store all new data. Once the data has been associated, it is fused with the server track and the buffer is updated. As soon as an LDM track has been fused, a flag in the buffer corresponding to this data is set to one such that the filter knows that this data has been used.

The final variable contained in the buffer of each *ServerTrack* is a source ID that is

used to denote which active actor the local track came from. This variable is needed in order to perform the IDC algorithm in which the previous data from the same actor is used.

In addition to the *ServerTracks*, there is a second data structure that is used in the fusion server. This structure is denoted *LDMTrack* which is the input to the first two subsystems in the fusion server illustrated in Fig. 3.2. The structure of *LDMTrack* which is the data from the actors is shown in Fig. 3.3a. As mentioned in previous chapters, the LDM, i.e. *LDMTrack*, contains state estimates and covariance matrices for the actors' ego state as well as for all dynamic objects that it can detect and track. Furthermore, this structure contains a time stamp as well as an actor ID where the latter is individual for each active actor.

## 3.3 Coordinate Transformation

Since all local tracks in each LDM are to be associated with a server track, they need to be expressed in the same global coordinate system. Thereby the equations described in Section. 2.3 is used to transform all state estimates and corresponding covariances to global coordinates for all the objects in each *LDMTrack*. The transformation for each object is done based on the estimate of the actors' ego state. After the transformation, all of the objects in each LDM are expressed in a global coordinate frame and hence can be associated and fused with the server tracks.

## 3.4 Data Association

This subsystem seek to determine if a local track and a server track (contained in an LDM and the GDM respectively) corresponds to the same hypothesized target. A pseudo code for the data association process is listed in Algorithm 1 and is done by calculating the MD, defined in chapter 2 of this thesis, between the local track and all other server tracks. The local track is associated with the server track to which the distance is the smallest. However, since the cross-covariance between the local and global tracks is unknown, the simplified version of the MD in Eq. 2.30 is used in the association.

If a track from an LDM can not be associated with a server track, which means that the MD is too large, the track is assumed to be a new object which has entered the environment. The MD is compared to a threshold drawn from a $\chi^2$-distribution with 95 percent confidence interval. If the distance is below this threshold, an association is made between the local and server track. If the distance is above the threshold, the two tracks are considered to be too far apart and a new server track is created. In this case, the track is initialized with the local track estimate and covariance found in the LDM.

Before the association can be done, the local and server track needs to be temporally aligned so the tracks represent the object state in the same time instance. To do this, the data in the server track buffer with the time stamp closest to the local tracks time stamp is extracted. The track with the smallest time stamp is then predicted to the other tracks time. This is done using the prediction steps described in section 2.1.1 and 2.1.2.

For one local track in an LDM, the algorithm simply iterates over all server tracks and saves the ID and distance to the server track to which the distance is the smallest. Once associated, the local track is inserted into the buffer of the associated server track. As illustrated in Fig. 3.4, where each box represents a local track with time stamp $t_i$, the new data (the newly associated local track from the LDM) is inserted such that the buffer is sorted in time (the time at which the local track was created).

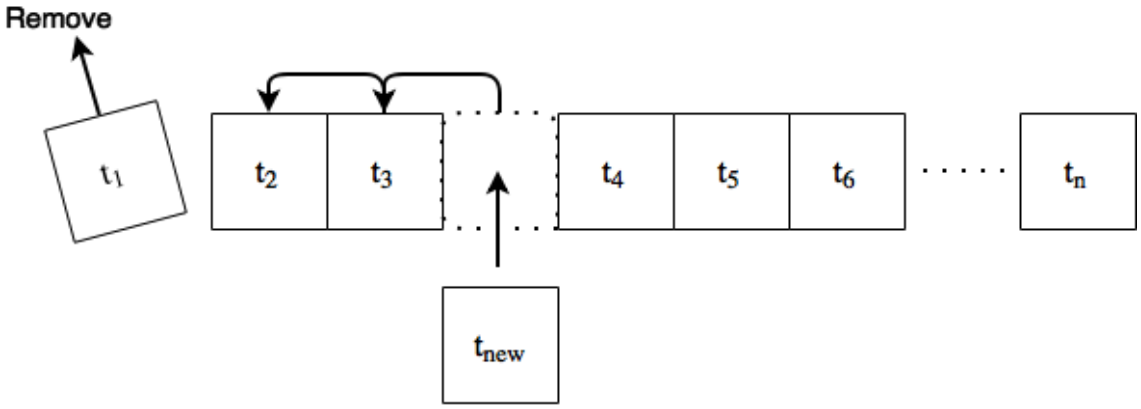



**Figure 3.4:** Illustration of how data is inserted in the buffer for each server track ($t_3 < t_{new} < t_4$).

**Algorithm 1:** Data Association

**input :** ServerTracks, LDMTracks
**output:** ServerTracks

```
// Loop through all new LDMs received in the server
for i ← 1 to length(LDMTracks) do
    // Merge ego and object tracks
    allTracks = [LDMTracks(i).Ego  LDMTracks(i).Objects];
    // Loop through all tracks
    for j ← 1 to length(allTracks) do
        // Extract track estimate, covariance and time stamp from
           local track
        [x_j, P_j, t_j] ← extractData(allTracks(j));
        // Initialize distance metric to a large number
        distance ← 10000;
        // Loop through each server track
        for k ← 1 to length(ServerTracks) do
            // Extract track estimate, covariance and time stamp from
               server track which is closes in time to the local
               track
            [x_k, P_k, t_k] ← extractDataClosestInTime(ServerTracks(k), t_j);
            // Align local and server tracks in time
            [x_k, P_k, x_j, P_j] ← alignTracksInTime(x_k, P_k, t_k, x_j, P_j, t_j);
            dist ← Mahalanobis(x_j, P_j, x_k, P_k);
            // Associate local track with server track that has the
               smallest Mahalanobis distance
            if dist < distance then
                distance ← dist;
                trackInd ← k;
            end
        end
        // Extract local track
        [x_j, P_j, t_j] ← extractData(allTracks(j));
        // Compare distance with threshold from χ²-distribution
        if distance < chi2inv(0.95, 5) then
            // Local track associated with a server track, put in
               buffer to be fused
            InsertTrackInBuffer(x_j, P_j, t_j, ServerTracks(trackInd));
        else
            // Local track not associated to any server track, create
               new server track, initialize with local track
            CreateNewTrack(x_j, P_j, t_j);
        end
    end
end
```

## 3.5 Data Fusion

When the data in each new LDM received at the server has been associated with the server tracks, fusion can be performed. In the data association, all tracks from the LDMs that were associated with a server track was put in the corresponding buffer for the associated server track (sorted in time). All data in the buffer is fused with the server track by using the IDC algorithm presented in section 2.1.3. For the IDC algorithm to work, the previous local track from the same actor needs to be stored in memory. Therefore, the size of the buffer in the fusion server was defined to be big enough to store the previous local tracks from all active actors, i.e. larger than 32.

The fusion algorithm iterates over all server tracks. For each track the algorithm iterates over the server track buffer, starting one index to the left of the recently received local track (put there in the data association process). This is done because this is where the most recent server track estimate and covariance which does not need to be updated is saved. The fusion algorithm predicts the chosen server track estimate to the time stamp of the local track estimate closest in time (positive time). The IDC algorithm is then used to produce a server track estimate for that particular time stamp. The algorithm then picks the resulting server track estimate and perform the prediction and update steps all over again until it reaches the end of the buffer (the most recent local track). The procedure is summarized in Algorithm 2.

**Algorithm 2:** Data Fusion Algorithm

**input :** ServerTracks
**output:** ServerTracks

```
// Loop through all server tracks
for m ← 1 to length(ServerTracks) do
    // Loop through all slots in the track buffer, starting at the
       index stored in FusionStartIndex
    for n ← FusionStartIndex to length(ServerTracks(m).Buffer)-1 do
        // Extract server state estimate, covariance and time stamp
        [x_k, P_k, t_k] ← extractData(ServerTracks(m).Buffer(n));
        // Extract local track from buffer (to which the server will
           be predicted)
        [x_j, P_j, t_j] ← extractData(ServerTracks(m).Buffer(n + 1));
        // Extract previous local track from buffer
        [x_{j-1}, P_{j-1}, t_{j-1}] ←
         findPrevLocalTrack(ServerTracks(m).Buffer, n);
        // Predict server track to next local tracks time
        [x_k, P_k] ← timeUpdate(x_j, P_j, t_j − t_k);
        // Predict previous local track from the same actor
        [x_{j-1}, P_{j-1}] ← timeUpdate(x_{j-1}, P_{j-1}, t_j − t_{j-1});
        // Perform update step using IDC fusion
        [x_k, P_k] ← IDC(x_k, P_k, x_j, P_j, x_{j-1}, P_{j-1});
        // Update server track
        inputDataInServerBuffer(x_k, P_k, ServerTracks(m).Buffer(n + 1));
    end
end
```

## 3.6 Track Management

The fusion server produces tracks for all objects in the environment that the active actors can detect and track. A problem which arose was how to detect when an object has left the environment and is no longer needed to track, i.e. a track management algorithm was needed. This algorithm keeps track of how long time has passed since the last track from an LDM was associated/fused with a certain server track. If the time since the last data was received/fused for a certain server track is above a certain threshold, the track is terminated and the resources are released.

When a new track has been created in the server, an extra requirement was added. If less than 5 tracks from the LDMs have been associated/fused with the server track, the threshold is set to a smaller value. These tracks are called tentative tracks and are used to be able to quickly remove tracks that do not correspond to a physical target (false detections). This guarantees that a new tentative track gets terminated quickly if no tracks are associated with it. The track management algorithm can be

seen in Algorithm 3 and the value for the threshold was set to 1 second for regular server tracks and 0.3 seconds for tentative tracks. After the track management has been performed, the output is fed back to the data association in the next time step as displayed in Fig. 3.2.

**Algorithm 3:** Track Management Algorithm

**input :** ServerTracks, currentTime
**output:** ServerTracks

```
// Loop through all server tracks
for i ← 1 to length(ServerTracks) do
    // Set variable timeOut
    timeOut ← 1;
    // Check if track is tentative
    if tentative then
        // Set low timeOut
        timeOut ← 0.3
    end
    // Check if no new data has been received for "timeOut"
       seconds
    if currentTime ≥ ServerTracks(i).Buffer(end).timeStamp + timeOut
     then
        // Terminate this track
        ServerTracks(i) ← [];
    end
end
```

## 3.7 Output Prediction

Since the time stamp for each fusion server track is the same as the time stamp for the last track from an LDM that was fused with it, a final prediction step is needed to get the server track estimates to the current time. To do the final prediction, the CKF and UKF time update equations described in section 2.1.2 and 2.1.1 are used and pseudo code for the output prediction is listed in Algorithm 4. After the final prediction, all server tracks are updated to the current time and represents the

GDM containing all dynamic objects that can be seen by the active actors.

**Algorithm 4:** Output Prediction Algorithm

**input :** ServerTracks, currentTime
**output:** ServerTracks

```
// Loop through all server tracks
for i ← 1 to length(ServerTracks) do
    // Extract server data
    [x_k, P_k, t_k] ← extractData(ServerTracks(i));
    // Predict state and covariance
    [x_k, P_k] ← TimeUpdate(x_k, P_k, currentTime − t_k);
    // Update ServerTracks
    ServerTracks(i) ← insertData(x_k, P_k, currentTime);
end
```

# 4

# System Evaluation

To test the performance of the fusion algorithm in the server, a simulation scenario was created. The scenario was designed to represent real-world driving situations and involves both active actors (vehicles) as well as passive actors (a pedestrian). The scenario was created manually in the simulation program SPAS where the desired trajectories for each object is predefined before the simulation starts.

To evaluate the performance of the fusion filter, evaluation metrics were needed. Since the ground truth was known for the simulated data, the Mean Squared Error (MSE) and Multiple Object Tracking Accuracy (MOTA) [29] was used as metrics. In this chapter, the test scenario used to test the fusion algorithm is described in detail. Furthermore, the evaluation metrics that were used are also presented.

.

## 4.1 Modeling Transmission Delay

All active actors send data to the fusion server with a fixed frequency during a test run. This frequency was changed between runs in order to show the effects it had on the fusion performance. As previously discussed, the wireless data transmission introduced delays of the data packets and to evaluate the impact the delay had, different values for the delay were tested.

The transmission delay was defined using a minimum and maximum value where the delay was modeled as a random variable between these limits. The minimum delay $d_{min}$ was defined to be 40 ms which was used for all data packets sent to the server. An additional random delay was added to each data packet to simulate the randomness of the wireless communication. The total delay $d$ was computed as

$$d = \begin{cases} |X|, & if \ \ |X| \leq 2\sigma \\ d_{max}, & if \ \ |X| > 2\sigma \end{cases} \tag{4.1}$$

where $X$ is a sample drawn from a normal distribution with mean $d_{min}$ and standard

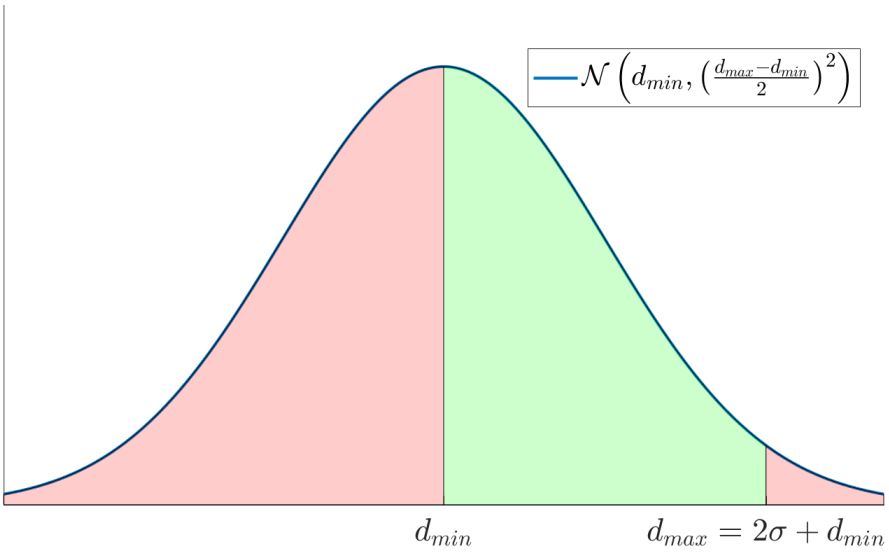



**Figure 4.1:** Illustration of how the delay for each data packet was generated. It was drawn from a normal distribution with mean $d_{min}$ and standard deviation $\sigma$. $d_{max}$ is the maximum delay that was applied to a packet defined as $d_{max} = d_{min} + 2\sigma$. The delay is selected such that it lies within the green area, that is between $d_{min}$ and $d_{max}$, and with a higher probability that the delay is closer to $d_{min}$. The blue curve represents the probability density function for the delay.

deviation $(d_{max} - d_{min})/2$ which is illustrated in Fig. 4.1 where $d_{max}$ is the maximum transmission delay. The total delay $d$ was thus saturated at the minimum value $d_{min}$ and maximum value $d_{max}$ where the delay was more likely to take values close to $d_{min}$.

## 4.2 Simulation Scenario

The algorithm was tested on a simulated scenario which involved three active actors who transmitted data to the fusion server as well as one passive actor (in this case a pedestrian) that was standing still next to the road. The scenario is illustrated in Fig. 4.2 and includes a straight road which leads to a roundabout where all vehicles turned left and continued driving forward. All of the vehicles drove with different velocities during the simulation and to further resemble a realistic scenario, car number two in Fig. 4.2 performed a manoeuvre and overtook car number one during the simulation. This simulation scenario lasts approximately 20 s. Fig. 4.3 further illustrates the true states for each actor during the simulation.

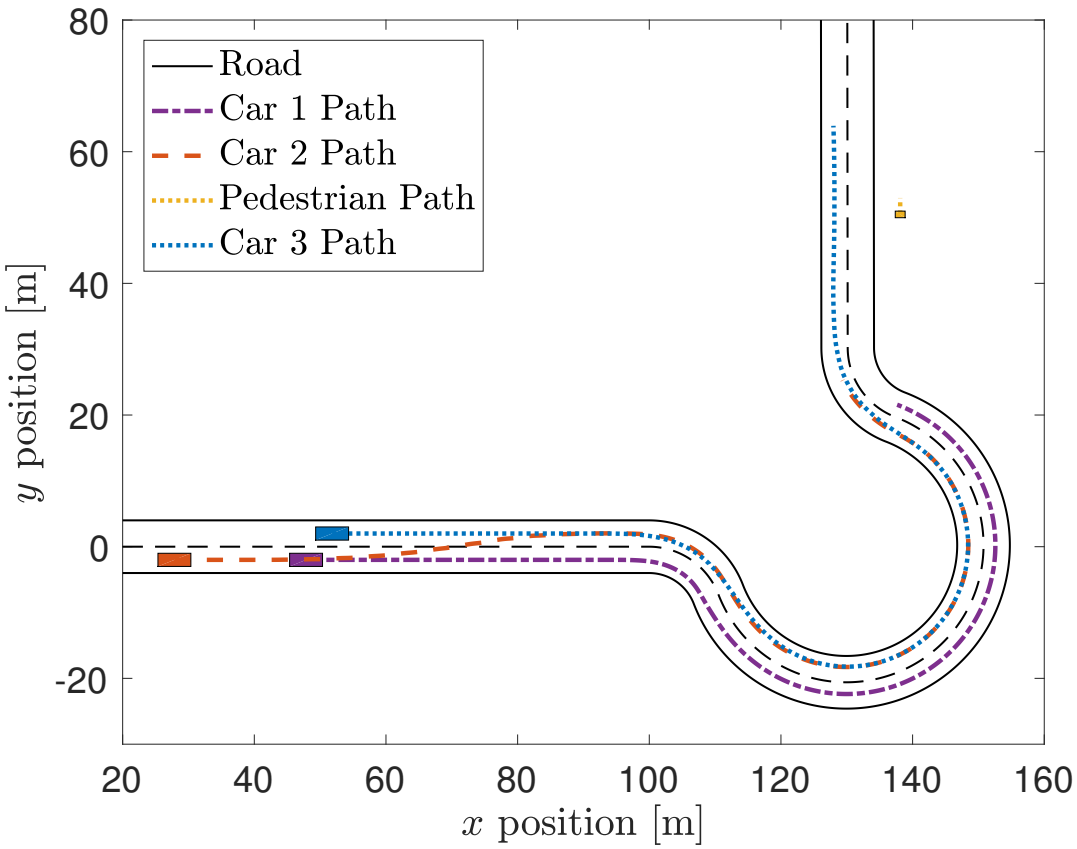
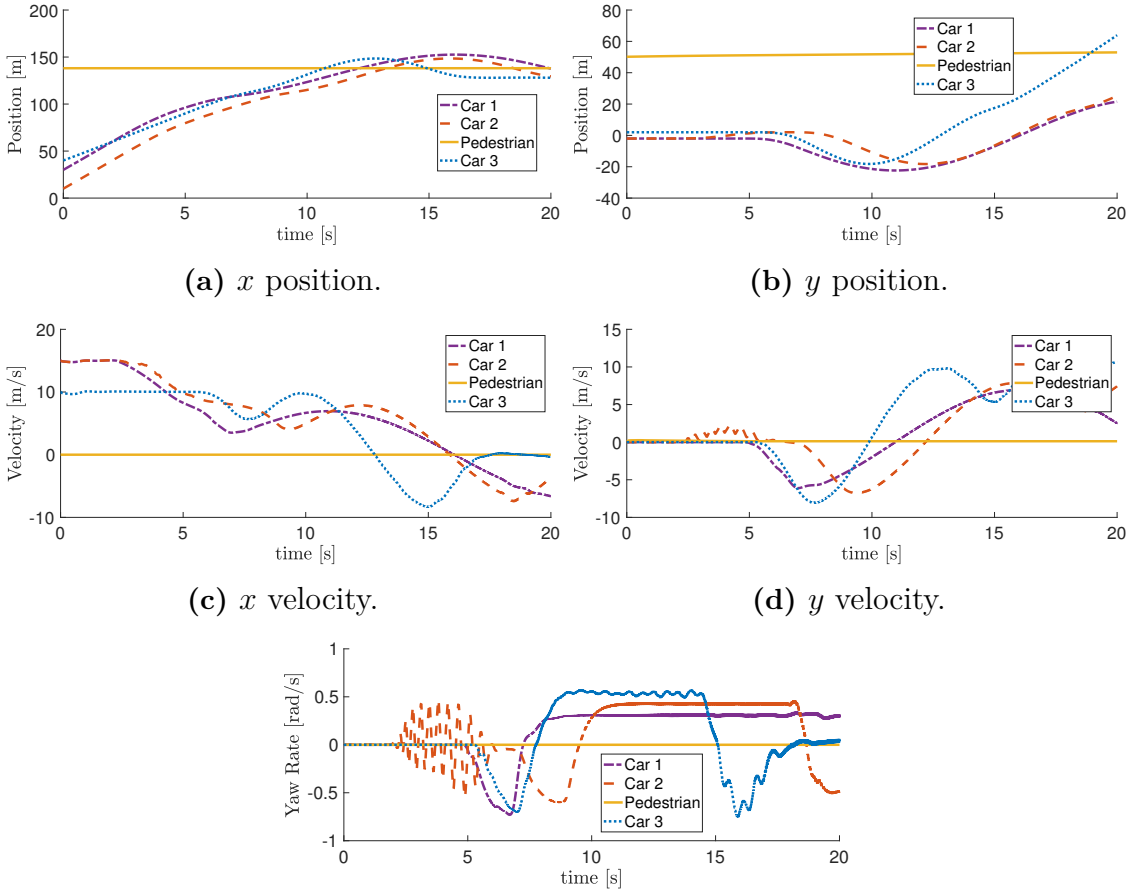



**Figure 4.2:** Path of the actors in the simulated scenario. The boxes represents the starting position of each actor and the dotted lines are the corresponding paths.




**(a)** $x$ position.

**(b)** $y$ position.

**(c)** $x$ velocity.

**(d)** $y$ velocity.

**(e)** Yaw rate.

**Figure 4.3:** The true states for each of the actors involved in the scenario.

The fact that there was an overtake included in the scenario tested the performance of the data association algorithm in the fusion server since the vehicles, and hence the measurements, were closer together. During the overtaking, track estimates from two actors could easily be associated with the wrong system track if not using a good association metric.

The scenarios tested is presented in Table 4.1 where the delays, sending frequency, filter type and the use of rollback was altered in each simulation. In the first scenario (A in Table 4.1) the communication delay was fixed to the minimum delay (40 ms), but for the other scenarios (B-D) in Table 4.1 the delay was random within a range. Furthermore, a different transmission rate was simulated in scenario C and finally in scenario D rollback was not used. That is, when tracks that arrived to the fusion server was older than the last fused track, they where ignored in scenario D. Commonly for scenario A-D is that in the fusion algorithm, CKF prediction was used. In the final scenario (E), the unscented transform (prediction step of UKF) was used instead.

**Table 4.1:** Definition of settings in the simulated scenario.

| Scenario | Delay | Sending Freq. | Filter | Rollback |
|---|---|---|---|---|
| A | 40 ms (fixed) | 10 Hz | CKF/IDC | Yes |
| B | 40–300 ms | 10 Hz | CKF/IDC | Yes |
| C | 40–300 ms | 3 Hz | CKF/IDC | Yes |
| D | 40–300 ms | 10 Hz | CKF/IDC | No |
| E | 40–300 ms | 10 Hz | UKF/IDC | Yes |

## 4.3 Evaluation Metrics

In single-object tracking, it is rather straight forward to evaluate the tracking performance in terms of accuracy. A widely used metric in these cases is the MSE described below which enables several tracking algorithms to be compared to each other. In multi-object tracking, the performance of the tracker is more difficult to assess since there is more logic involved such as the data association and track management. It is for example desirable to assess how often the tracker misses an object, i.e. does not maintain a track for an object. It is also desirable to examine how often false positives occur, i.e. how often a track is maintained for an object that does not exist. When assessing the proposed fusion server, in addition to the MSE metric, the MOTA was used to evaluate the performance of the fusion server which is described below.

### 4.3.1 Mean Squared Error

The first metric that was used when evaluating the fusion filter is the MSE which is defined as

$$MSE = \frac{1}{K}\sum_{i=1}^{K}(x_i - \hat{x}_i)^2. \tag{4.2}$$

Here $x_i$ is the ground truth at each time step $i$, $\hat{x}_i$ is the corresponding filter output and $K$ is the simulation length in number of samples. The MSE metric is always larger than or equal to zero and a smaller value indicates a better performance in terms of accuracy. One of the objectives with the fusion server is that the error of the filter output should be as small as possible and to test this, the MSE is a well-suited metric. This metric focuses on one track and does not consider e.g. the data association performance or the number of false positives of a multi-target tracker, but the MSE metric is good for evaluation of the accuracy of a single track.

### 4.3.2 Multiple Object Tracking Accuracy

Evaluation of multi-object tracking algorithm requires more thorough evaluation than single-object tracking. For this reason a metric, MOTA, was used in addition to the MSE mentioned above. The MOTA metric discussed in [29] evaluates the tracking algorithm based on object configuration errors, i.e. if the system traces an object trajectory correctly and only produces one track per object. The metric also incorporates the number of false positives, i.e if the system is tracking an object that does not exist as well as misses and mismatches done during the entire scenario. The MOTA metric is defined as

$$MOTA = 1 - \frac{\sum_{i=1}^{K}(m_i + f_i + s_i)}{\sum_{i=1}^{K} g_i} \tag{4.3}$$

where $m_i$ is the number of misses, $f_i$ the number of false positives, $s_i$ the number of mismatches and $g_i$ is the number of objects present at each time instance $i$. Thereby the metric is composed of three different error ratios. By summation of the three error ratios, the total error rate $E$ is obtained. Finally the MOTA is computed as $1 - E$. As before, $K$ is the total number of samples in the simulation.

In the proposed fusion algorithm, no effort has been made to minimize the number of mismatches that occurs, therefore the MOTA metric was modified such that mismatches was not penalized, i.e.

$$MOTA = 1 - \frac{\sum_{i=1}^{K}(m_i + f_i)}{\sum_{i=1}^{K} g_i}. \tag{4.4}$$

# 5

# Results

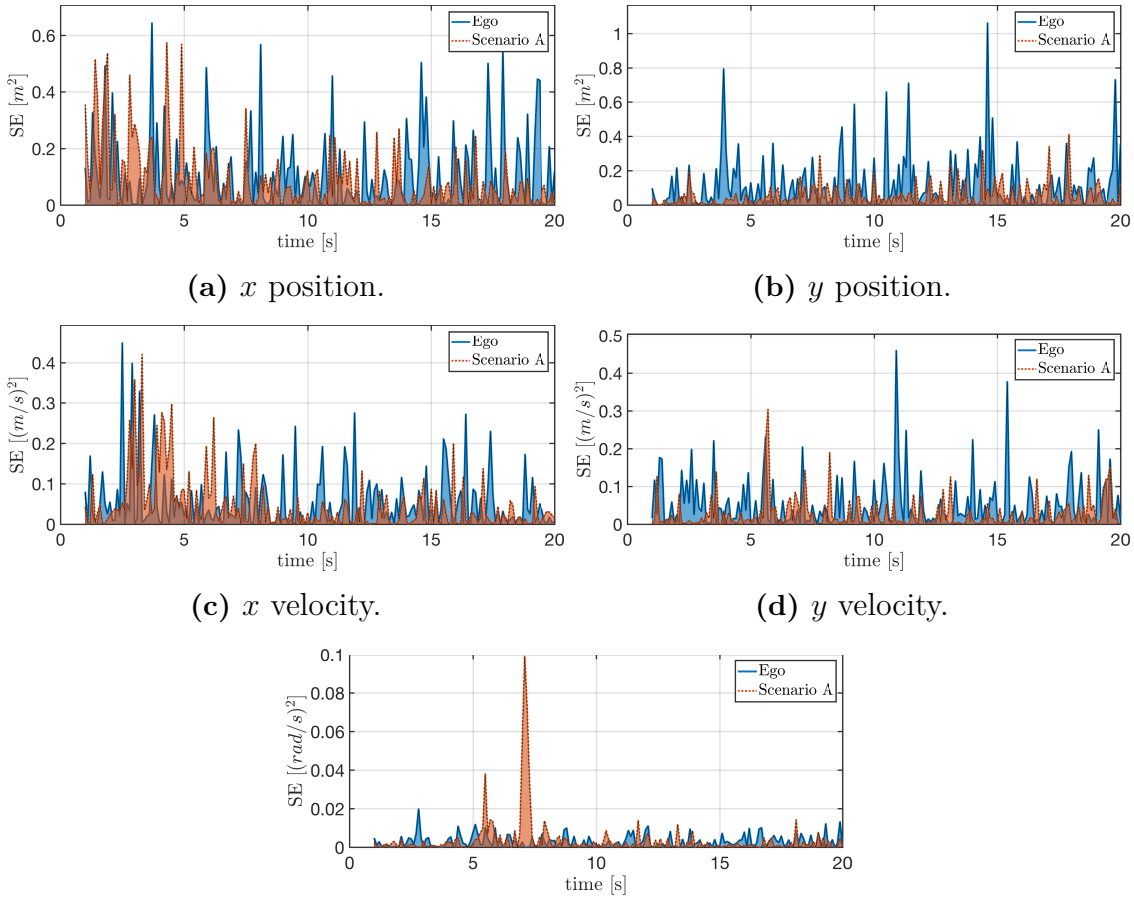
In this chapter, the performance of the fusion server is presented where the results are based on the scenario described in chapter 4. The filter output from the fusion server is compared with ground truth which was easily obtained in simulation. The performance metrics that are used are the previously discussed MSE and MOTA. Furthermore, plots of the Squared Error (SE) over time are studied to further observe the performance. The effects from changing the transmission delay; the frequency with which the actors send data as well as whether rollback is used or not is investigated for the scenario defined in the previous chapter. The performance of the fusion server is also compared to the individual state estimate contained in the LDMs.

The MSE as well as the MOTA from the simulation scenario is presented in Table 5.1 where MSE is calculated for each of the estimated states for one of the actors (car number one in Fig. 4.2). The results were similar for the other active actors as well and to be able to compare different setups, the results for the same actor is used. The MSE of the ego estimates is presented in the first line in Table 5.1. As can be seen, by fusing data from multiple sources the accuracy in the state estimates is clearly improved even if a small transmission delay is introduced. This is seen by comparing the results from the MSE from the ego estimates with the results from scenario A where a delay of 40 $ms$ is introduced.

Regarding the MOTA, Table 5.1 indicates that the computed MOTA value is high for all sub scenarios. However, extra delays and lower sending frequency decreases the MOTA slightly just as expected. Since the difference in the MOTA metric is small, the multi-target-tracker is thereby considered as robust.

**Table 5.1:** $MSE$ and $MOTA$ for the simulated scenario.

| **Scenario** | **MSE**$_x$ | **MSE**$_y$ | **MSE**$_{v_x}$ | **MSE**$_{v_y}$ | **MSE**$_\omega$ | **MOTA** |
|---|---|---|---|---|---|---|
| Ego | 0.1054 | 0.1132 | 0.0533 | 0.0501 | 0.0028 | - |
| A | 0.0866 | 0.0525 | 0.0470 | 0.0273 | 0.0036 | 0.996 |
| B | 0.1102 | 0.0563 | 0.0787 | 0.0442 | 0.0069 | 0.993 |
| C | 0.0870 | 0.0684 | 0.1129 | 0.0579 | 0.0096 | 0.979 |
| D | 0.1180 | 0.0593 | 0.0879 | 0.0524 | 0.0086 | 0.996 |
| E | 0.1101 | 0.0563 | 0.0787 | 0.0442 | 0.0069 | 0.993 |




**(a)** $x$ position.

**(b)** $y$ position.

**(c)** $x$ velocity.

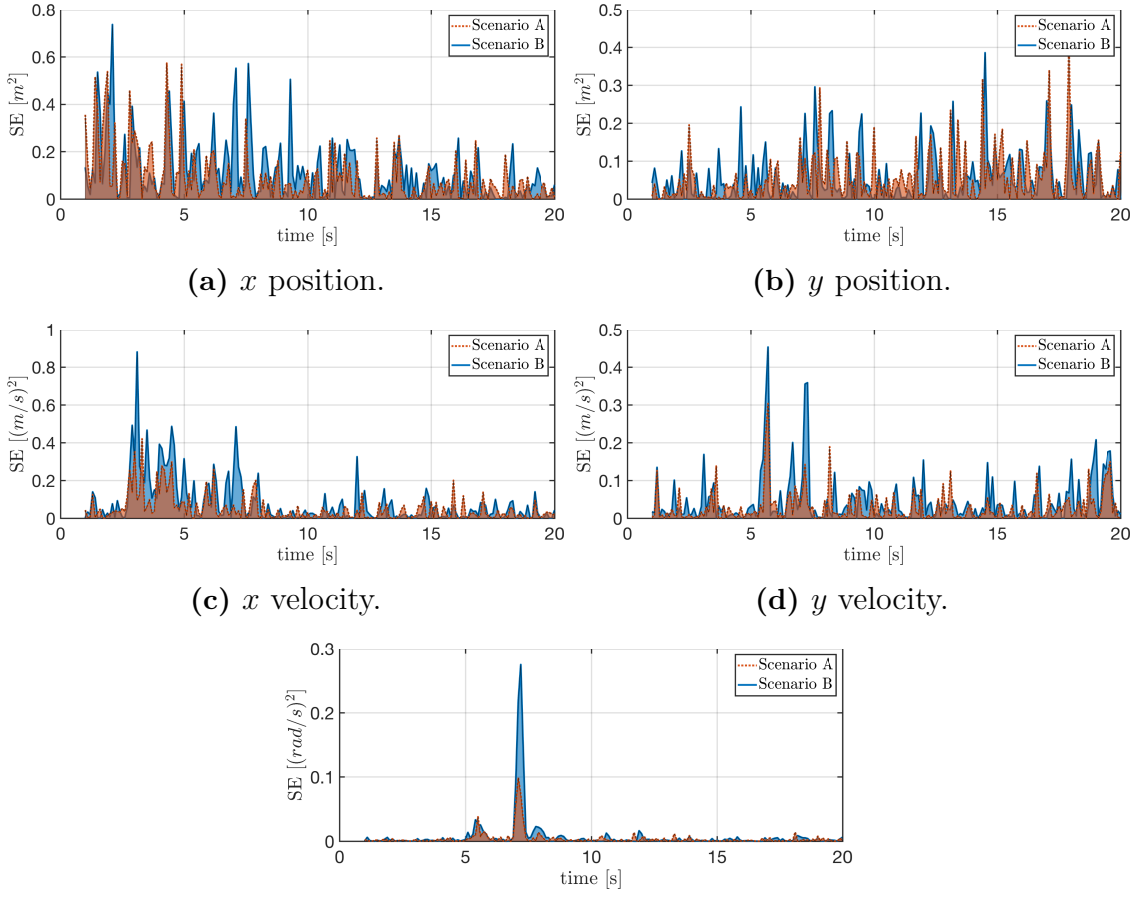
**(d)** $y$ velocity.

**(e)** Yaw rate.

**Figure 5.1:** Comparison of the Squared Error for the ego estimate and server filter output from scenario A.

## 5.1 Transmission Delay

The wireless communication from the actors to the central server suffers from random delays. Different values for this delay were tested, either it was constantly 40 ms for all data packets or it was modeled as described in Section 4.1 where the maximum delay was set to 300 ms. Scenario A represents an ideal scenario with the minimum possible transmission delay while B is a more realistic scenario. The effect the additional delays has on the fusion performance can be seen by comparing the MSE values from scenario A and B presented in Table 4.1. The SE for the two scenarios is also shown in Fig. 5.2 where the SE for the state estimates are shown over time. As expected, the accuracy of the state estimates decreases as the delays of the data packets increases.




**(a)** $x$ position.

**(b)** $y$ position.

**(c)** $x$ velocity.

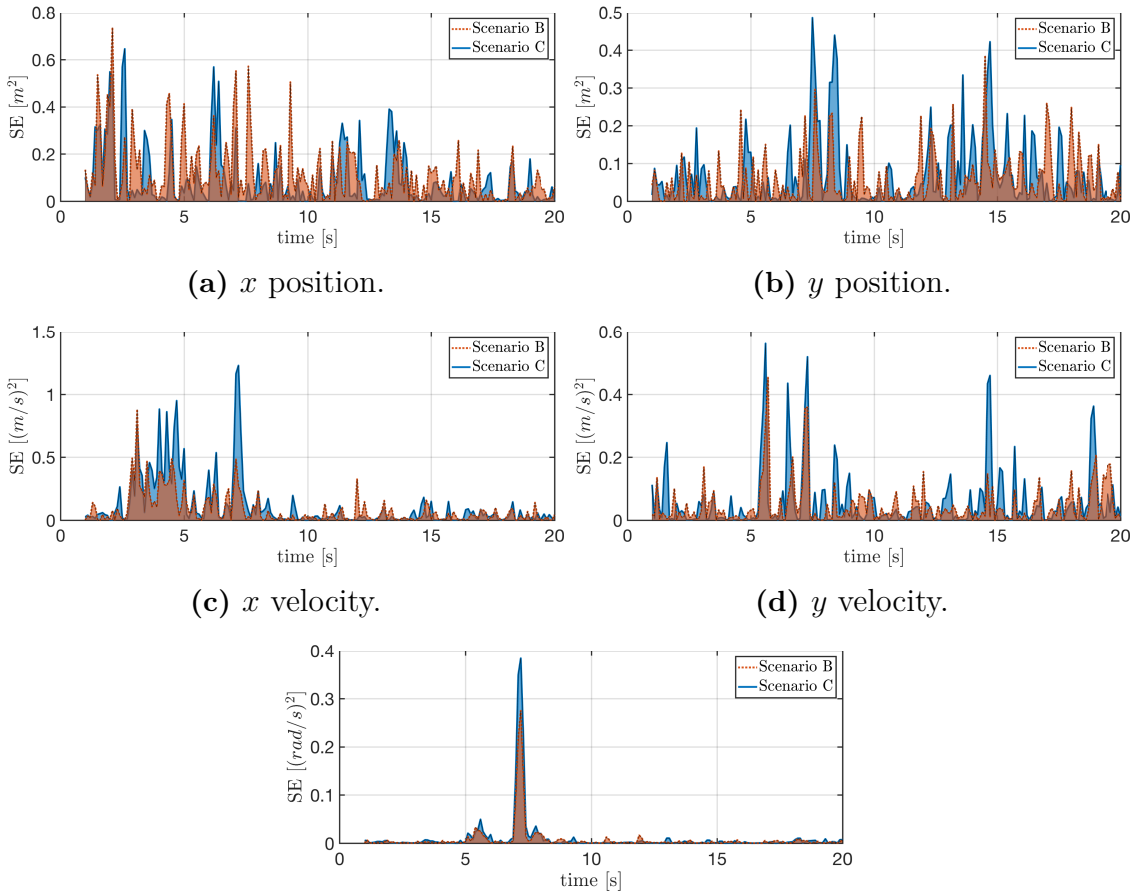
**(d)** $y$ velocity.

**(e)** Yaw rate.

**Figure 5.2:** Comparison of the Squared Error of the server filter output in scenario A and B.

## 5.2 Sending Frequency

Table 5.1 shows that the sending frequency of the vehicles has an impact on the accuracy of the state estimates. The MSE is generally increased as the sending rate is decreased from 10 Hz in scenario B to 3 Hz in scenario C. This behaviour is further illustrated in Fig. 5.3 where the evolution of the SE of the state estimates in scenario B and C is displayed.




**(a)** $x$ position.

**(b)** $y$ position.

**(c)** $x$ velocity.

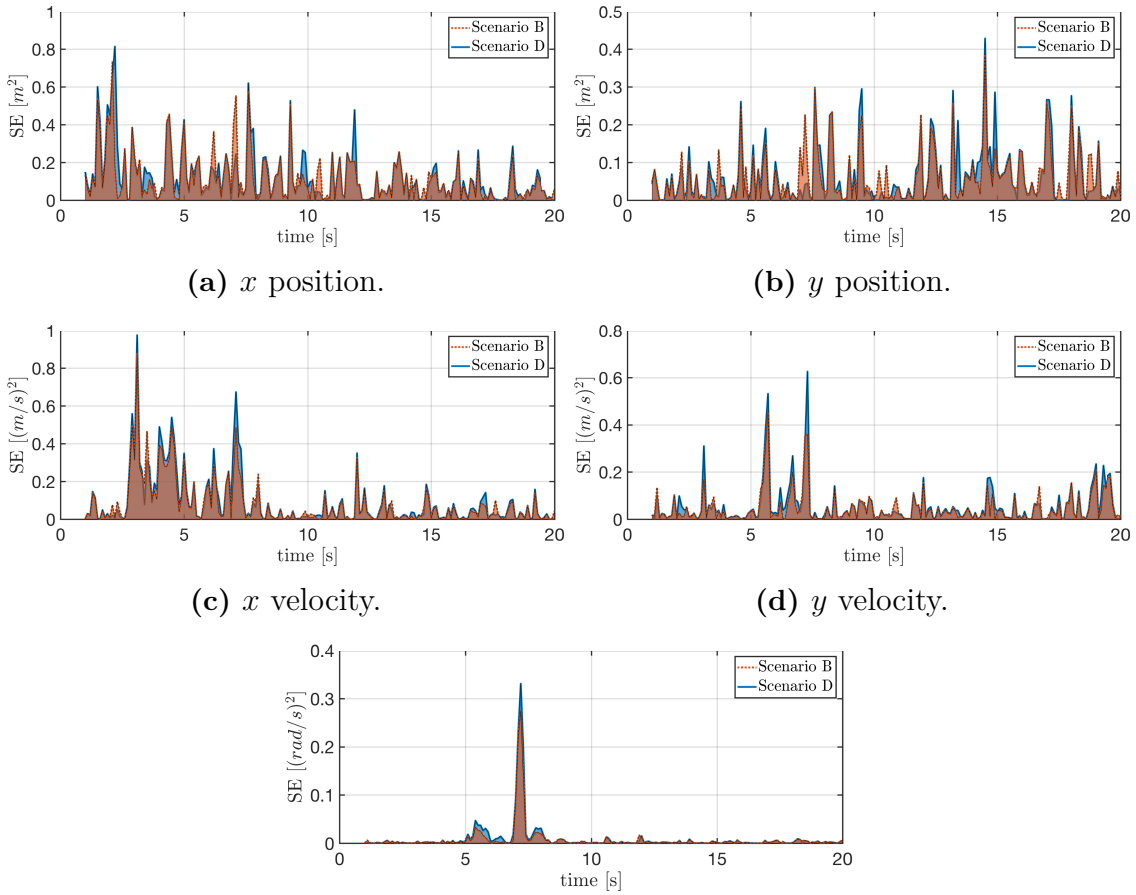
**(d)** $y$ velocity.

**(e)** Yaw rate.

**Figure 5.3:** Comparison of the Squared Error of the server filter output in scenario B and C.

## 5.3 Rollback

Table 5.1 also shows that if rollback is used, the accuracy of the state estimates is increased slightly in this scenario. This is seen by comparing the results from scenario B with D in Table 5.1. This is further illustrated in Fig. 5.4 where it can be seen that a little gain in accuracy is obtained when using rollback compared to ignoring the OOST.




**(a)** $x$ position.

**(b)** $y$ position.

**(c)** $x$ velocity.

**(d)** $y$ velocity.

**(e)** Yaw rate.

**Figure 5.4:** Comparison of the Squared Error of the server filter output in scenario B and D.

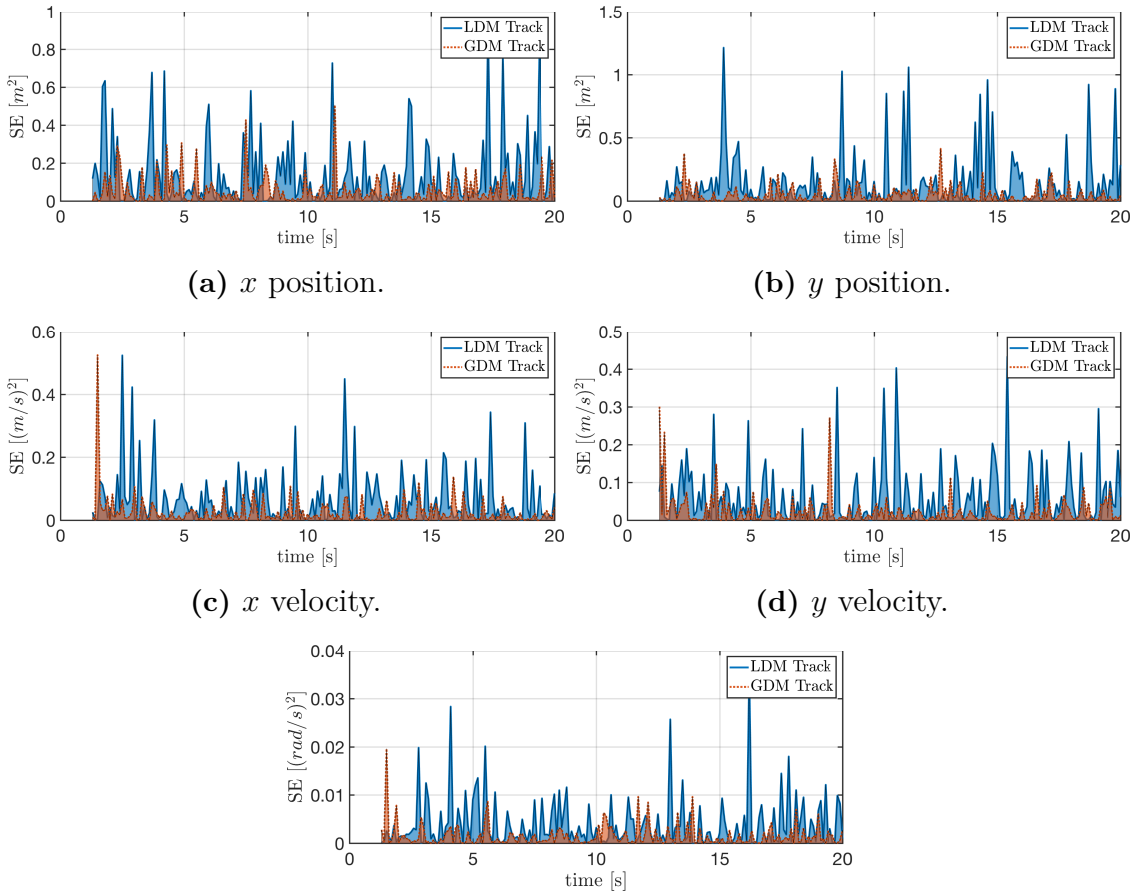
## 5.4 CKF vs. UKF

In Table 5.1, the MSE and MOTA results when using a UKF prediction is presented as well. By comparing the results from scenario B with E, it can be seen that both filters produces similar state estimates. The filter estimates and hence the squared error is close to identical and thereby it is not worth illustrating the difference with

plots of the SE. When using the UKF, the tuning parameters was set as in Eq. 2.14. When using the tuning parameters as in Eq. 2.13, the covariance matrix was in many cases not positive semi-definite.

## 5.5 Passive Actors

In the simulated scenario, there was one pedestrian included that stands still next to the road. By combining data from multiple sources, the state estimates for the pedestrian are improved in terms of accuracy which is illustrated in Fig. 5.5 where the SE is computed from the LDMs from one actor and compared with the fusion filter output (GDM) which combines the data from all active actors.




**(a)** $x$ position.

**(b)** $y$ position.

**(c)** $x$ velocity.

**(d)** $y$ velocity.

**(e)** Yaw rate.

**Figure 5.5:** Comparison of the Squared Error for state estimate of the pedestrian from one actor compared to the fusion server estimate.

# 6

# Discussion

The results obtained in simulation clearly indicates that it can be beneficial to fuse data from multiple vehicles. This was illustrated in Table 5.1 as well as in Fig. 5.1 where the accuracy of the state estimates can be improved compared to the ego estimate. In this setup, the conditions were ideal, i.e., the communication was assumed to be ideal and all packets had a constant delay of 40 ms. Since wireless communication is seldom ideal, a different setup was tested as well where each packet had an extra additional delay which potentially could lead to OOST. By introducing the larger delays, the accuracy of the state estimates was decreased just as expected, but was still comparable to the ego estimates displayed in Table 4.1 and Fig. 5.2.

Further tests were conducted to test the filter performance when the sending rate of the active actors was decreased. The fusion accuracy is clearly affected by the sending rate, as illustrated in Table 5.1 and Fig. 5.3. This is easily explained by the fact that the fusion server receives much less data to fuse. In the ideal case, we would like to increase the sending rate of the vehicles such that more data is used in the fusion algorithm which thereby would produce more accurate state estimates. However, due to limitations in the communication, 10 Hz is a reasonable sending frequency and we would not recommend to lower this limit. Especially if the data is to be used for controlling the vehicles.

In Fig. 5.5, the state estimates for the pedestrian from one of the actors (LDM) is compared with the fusion output (GDM). As can be seen, it can be extremely useful to fuse data from multiple sources for passive actors in terms of accuracy in the state estimate. The improvement of the accuracy is explained by the fact that more data is used. In this case, the uncertainties in the LDM tracks that is fused is approximately equally big. If instead the data from one vehicle had high accuracy and was fused with less accurate data from other vehicles, the increase in accuracy would not be as prominent.

Regarding the server estimate for an active actor, the state estimate from the ego state is fused with LDM tracks from other actors which have larger errors and uncertainties in the global coordinate frame. Thereby the improvement of the state estimates for active actors is not as big as for passive actors.

One of the purposes with the fusion server is to centrally steer actors to avoid

potential collisions. To be able to do this, an accurate GDM is needed. This project shows that this can be achieved if the communication is working well and e.g rollback is used. This algorithm can use the GDM that the fusion algorithm in this thesis produces. Another idea with the fusion server is that the GDM can be re-transmitted back to the actors in the test environment. This is not covered in this project, but it should be mentioned that the re-transmission will introduce extra errors in the state estimates due to communication delays.

This project did not focus on the computational complexity but rather on producing an accurate GDM. If the fusion algorithm is to be used on an actual server, the extra computational complexity that rollback introduces needs to be further studied. Ideally, if the server has high performance, rollback should definitely be used since the accuracy of the state estimates is increased. However, if cheaper and slower hardware is used, rollback can perhaps be ignored. Another idea is to set a limit on the rollback. For instance, if there are 5 or more newer tracks in the buffer as a new track arrives to the server, this track could be ignored. The analysis of the computational complexity of the fusion server is thereby left as future work.

By looking at the results from scenarios 1B and 1E in Table 4.1, it is clear that the performance of the fusion algorithm when using the CKF and UKF is nearly identical, both in terms of accuracy in the state estimates as well as the MOTA metric. Since the UKF uses one more sigma point, the computational time is increased. Since the CKF is more stable and does not suffer from the problem with non-positive semi-definite covariance matrices, the CKF is preferable for this application.

Regarding future work for the fusion server, the first thing that could be done is to implement the algorithm on a real server and get the communication between the vehicles and the server working. Once this has been done, the computational complexity can be investigated and a trade off between using rollback or not can be done. In SPAS it has been shown that the GDM can be used to centrally steer cars to avoid possible collisions. This part needs to be evaluated in the server as well.

In this project, $\sigma$-point methods were investigated when predicting the state evolution over time. Another possibility for future work is to investigate the performance when other filters such as the Particle Filter (PF) is used.

# 7

# Conclusion

The main purpose with this thesis was to develop an algorithm that produces an accurate state estimate of all dynamic objects in an environment based on data from multiple actors. This has successfully been achieved and has shown to increase the accuracy when a moderate transmission delay is added to the wirelessly transmitted data packets. Furthermore, it is shown that the sending frequency of the actors plays an important role on the fusion performance. Not surprisingly, a smaller delay and a larger sending frequency is desirable in order to accurately estimate the state of the actors.

The fusion algorithm was divided into subsystems that each performs a specific task. These were coordinate transformation, data association, data fusion, track management and final prediction respectively. This modular approach enables partial replacement of these subsystems if needed, e.g. if a different motion model is desired. The fusion algorithm was developed and analysed using MATLAB/SIMULINK and the algorithm was implemented in the previously mentioned SPAS model. The performance of the fusion server was evaluated in terms of the commonly used MSE and MOTA metrics.

This work has made it possible to create an accurate global map of all dynamic objects in an environment. This data can be used by other algorithms to steer vehicles such that e.g. potential collisions can be identified and avoided. Another implication with this work is that it enables vehicles to potentially extend their field of view by help of other vehicles. This can be done if the data is transmitted back to each vehicle.

# Bibliography


[1] G. Dimitrakopoulos and P. Demestichas, 'Intelligent transportation systems: systems based on cognitive networking principles and management functionality', *IEEE Veh. Technol. Mag.*, vol. 5, no. 1, pp. 77–84, 2010, ISSN: 15566072. DOI: 10.1109/MVT.2009.935537.

[2] O. Vermesan and Peter Friess, *Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems*. Aalborg: Rajeev Ranjan Prasad, 2013, p. 363, ISBN: 9788792982735. DOI: 10.2139/ssrn.2324902. arXiv: arXiv:1011.1669v3.

[3] J. Ngu and R. Soponpunth, 'Fusing data from multiple vehicles into a common picture', Master thesis, Chalmers University of technology, 2016.

[4] D. Fraser and J. Potter, 'The optimum linear smoother as a combination of two optimum linear filters', *IEEE Trans. Automat. Contr.*, vol. 14, no. 4, pp. 387–390, 1969, ISSN: 0018-9286. DOI: 10.1109/TAC.1969.1099196.

[5] H. Zhou, Z. Deng, Y. Xia and M. Fu, 'Update with out-of-sequence measurements', *Neurocomputing*, vol. 214, pp. 121–125, 2016, ISSN: 09252312. DOI: 10.1016/j.neucom.2016.05.078.

[6] Y. Bar-Shalom, 'Update with out-of-sequence measurements in tracking: exact solution', *IEEE Trans. Aerosp. Electron. Syst.*, vol. 38, no. 3, pp. 769–778, 2002, ISSN: 00189251. DOI: 10.1109/TAES.2002.1039398.

[7] K. Zhang, X. R. Li and Y. Zhu, 'Optimal update with out-of-sequence measurements for distributed filtering', *Proc. 5th Int. Conf. Inf. Fusion, FUSION 2002*, vol. 2, no. 6, pp. 1519–1526, 2002, ISSN: 1053587X. DOI: 10.1109/ICIF.2002.1020997.

[8] S. Challa, J. A. Legg and X. Wang, 'Track-to-track fusion of out-of-sequence tracks', *Proc. 5th Int. Conf. Inf. Fusion, FUSION 2002*, vol. 2, pp. 919–926, 2002. DOI: 10.1109/ICIF.2002.1020910.

[9] M. Mallick, S. Coraluppi and Y. Bar-shalom, 'Comparison of out-of-sequence measurement algorithms in multi-platform target tracking', *Proc. 4th Annu. Conf. Inf. Fusion*, vol. 2, 2012.

[10] C.-Y. Chong, K.-C. Chang, S. Mori and W. H. Barker, 'Architectures and algorithms for track association and fusion', *IEEE Aerosp. Electron. Syst. Mag.*, vol. 15, no. 1, pp. 5–13, 2000.

[11] X. Liu, H. Yin, H.-Y. Liu and Z.-M. Wu, 'Performance of track-to-track association algorithms based on mahalanobis distance', *Sensors & Transducers*, vol. 22, no. June, pp. 58–65, 2013.

[12] K. C. Chang, R. K. Saha and Y. Bar-Shalom, 'On optimal track-to-track fusion', *IEEE Trans. Aerosp. Electron. Syst.*, vol. 33, no. 4, pp. 1271–1276, 1997, ISSN: 00189251. DOI: 10.1109/7.625124.

[13] H. Li, F. Nashashibi, B. Lefaudeux and E. Pollard, 'Track-to-track fusion using split covariance intersection filter-information matrix filter (scif-imf) for vehicle surrounding environment perception', *IEEE Conf. Intell. Transp. Syst. Proceedings, ITSC*, pp. 1430–1435, 2013. DOI: 10.1109/ITSC.2013.6728431.

[14] Y. Bar-Shalom, 'On hierarchical tracking for the real world', *IEEE Trans. Aerosp. Electron. Syst.*, vol. 42, no. 3, pp. 846–849, 2006, ISSN: 00189251. DOI: 10.1109/TAES.2006.248192.

[15] M. Aeberhard, S. Schlichtharle, N. Kaempchen and T. Bertram, 'Track-to-track fusion with asynchronous sensors using information matrix fusion for surround environment perception', *IEEE Trans. Intell. Transp. Syst.*, vol. 13, no. 4, pp. 1717–1726, 2012, ISSN: 15249050. DOI: 10.1109/TITS.2012.2202229.

[16] S. Sarkka, 'Bayesian filtering and smoothing', *Cambridge Univ. Press*, p. 254, 2013. DOI: 10.1017/CBO9781139344203.

[17] M. Lundgren, 'Bayesian filtering for automotive applications', PhD thesis, Chalmers University of Tecknology, 2015, p. 48, ISBN: 9789175971759.

[18] S. J. Julier and J. K. Uhlmann, 'New extension of the kalman filter to nonlinear systems', *Proc. SPIE*, vol. 3068, no. 3, pp. 182–193, 1997, ISSN: 0277786X. DOI: 10.1117/12.280797. arXiv: arXiv:1011.1669v3.

[19] R. Kandepu, B. Foss and L. Imsland, 'Applying the unscented kalman filter for nonlinear state estimation', *J. Process Control*, vol. 18, no. 7-8, pp. 753–768, 2008. DOI: 10.1016/j.jprocont.2007.11.004.

[20] S. Haykin and I. Arasaratnam, 'Cubature kalman filters', *IEEE Trans. Automat. Contr.*, vol. 54, no. 6, pp. 1254–1269, 2009. DOI: 10.1109/TAC.2009.2019800.

[21] Y. Bar-Shalom, 'On the track-to-track correlation problem', *IEEE Trans. Automat. Contr.*, vol. 26, no. 2, pp. 571–572, 1981, ISSN: 15582523. DOI: 10.1109/TAC.1981.1102635.

[22] K. C. Chang, T. Zhi and R. K. Saha, 'Performance evaluation of track fusion with information matrix filter', *IEEE Trans. Aerosp. Electron. Syst.*, vol. 38, no. 2, pp. 455–466, 2002, ISSN: 00189251. DOI: 10.1109/TAES.2002.1008979.

[23] R. De Maesschalck, D. Jouan-Rimbaud and D. L. Massart, 'The mahalanobis distance', *Chemom. Intell. Lab. Syst.*, vol. 50, no. 1, pp. 1–18, 2000, ISSN: 01697439. DOI: 10.1016/S0169-7439(99)00047-7.

[24] T. Soler and M. Chin, 'On transformation of covariance matrices between local cartesian coordinate systems and commutative diagrams', *Tech. Pap. 45th Annu. Meet. ACSM*, pp. 393–406, 1985, ISSN: 07483244.

[25] MathWorks, *Simulation and model-based design*. [Online]. Available: https://se.mathworks.com/products/simulink.html (visited on 8th May 2017).

[26] X. Yuan, F. Lian and C. Han, 'Models and algorithms for tracking target with coordinated turn motion', *Math. Probl. Eng.*, vol. 2014, 2014. DOI: 10.1155/2014/649276.

[27] M. Roth, G. Hendeby and F. Gustafsson, 'Ekf / ukf maneuvering target tracking using coordinated turn models with polar / cartesian velocity', *17th Int. Conf. Inf. Fusion*, pp. 1–8, 2014.

[28] C. M. Lewandowski, N. Co-investigator and C. M. Lewandowski, *Vehicle Dynamics and Control*. Springer Science, 2015, vol. 1, pp. 1689–1699, ISBN: 9788578110796. DOI: 10.1017/CBO9781107415324.004. arXiv: arXiv:1011.1669v3.

[29] K. Bernardin and R. Stiefelhagen, 'Evaluating multiple object tracking performance: the clear mot metrics', *Eurasip J. Image Video Process.*, vol. 2008, 2008. DOI: 10.1155/2008/246309.