CHALMERS
UNIVERSITY OF TECHNOLOGY

Fusing data from multiple vehicles into a common picture

Master's thesis in
Computer Science - Algorithms, Languages, and Logic
Engineering Mathematics and Computational Science

Johnny Ngu
Ronakorn Soponpunth

Department of Computer Science and Engineering, and Mathematical Sciences
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2016

Fusing data from multiple vehicles into a common picture

JOHNNY NGU
RONAKORN SOPONPUNTH

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Department of Mathematical Sciences
Chalmers University of Technology
Gothenburg, Sweden 2016

Fusing data from multiple vehicles into a common picture
JOHNNY NGU
RONAKORN SOPONPUNTH

Supervisors:	Josef Nilsson,	Volvo Car Corporation
	Thomas Petig,	Chalmers University of Technology
	Mats Rudemo,	Chalmers University of Technology
	Elad Schiller,	Chalmers University of Technology
	Aila Särkää,	Chalmers University of Technology
	Yury Tarakanov,	Volvo Car Corporation

Examiners:	Mats Rudemo,	Chalmers University of Technology
	Mary Sheeran,	Chalmers University of Technology

Department of Computer Science and Engineering
Department of Mathematical Sciences
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden
Telephone +46 31 772 1000

Gothenburg, Sweden 2016

Fusing data from multiple vehicles into a common picture
JOHNNY NGU
RONAKORN SOPONPUNTH
Department of Computer Science and Engineering
Department of Mathematical Sciences
Chalmers University of Technology

Abstract

Modern day intelligent vehicles carry advanced sensors that gather data and fuse this into a map with dynamic objects. This overview of the car's immediate surroundings is then utilized by an autonomous-drive pilot to steer the vehicle. There have been lots of research where the data collection and the data fusion virtually have been at the same location, on a single vehicle. However, less is known when the data providers and the consumer are separated, and especially when the providers are spatially spread. Through this contrasting approach, we envision increased awareness and safety for all involved actors.

The separation introduces a communication problem, where the transmission time of the data becomes significant. In safety-critical systems, it is integral to receive new information as fast as possible. Thus, in order to study the dynamics of this model, we propose a novel test-track environment for Intelligent Traffic Systems (ITS) involving multiple vehicles where the data are centrally fused. The study is dissected into three areas: accuracy, network delay, and speed. Through this prototype system, we seek to illuminate difficulties and identify the possibilities of an on-line fusion server along with performance and scalability.

Our tracking algorithm is based on Bayesian Tracking Theory and tested with data provided by Volvo Car Corporation. Using this setup, our simulations show significant improvements in positional accuracy when the vehicles cooperate. We also find that the transmission delay is crucial in terms of both speed and precision of the algorithm. In addition, the bottlenecks were primarily located to algebraic operations, which put a boundary of the speed of our tracking algorithm. Due to the small scale of the system and the matrices, the possibility of optimizations was done by multithreading. The study made on our fusion server suggest that an on-line fusion server might support a few hundred cars. But what dictates the scalability is the trade-off between the speed and precision of the fusion server - which is extremely difficult, if not impossible to balance optimally.

Keywords: data processing, data fusion, algorithms, Bayesian filtering, fusion architecture, transmission delay, asynchronous fusion.

Acknowledgements

We would like to thank our supervisors at Chalmers: Thomas Petig, Mats Rudemo, Elad Schiller and Aila Särkää. During difficult times, they have guided and pushed us towards a good direction. Their expertise in this topic, and time they have spent reading the report have helped us greatly.

Special thanks also go to our supervisors at Volvo Car Corporation, Josef Nilsson and Yury Tarakanov. For their input during the course of this project, making the project progress, and also providing us with tools and data.

The amount of people involved in this project tells the tale that this is a very interesting topic. We are humbled by the immense support we have gotten, but most importantly, that we got the opportunity to write this thesis.

Johnny Ngu and Ronakorn Soponpunth, Gothenburg, June 2016

Contents

List of Figures

Acronyms

EtM Extrapolating the Measurement

EKF Extended Kalman Filter

FIFO First In First Out

GDM Global Dynamic Map

GNN Global Nearest Neighbor

GPS Global Positioning System

IF Information Filter

ITS Intelligent Traffic System

JPDA Joint Probabilistic Data Association

LDM Local Dynamic Map

MHT Multi Hypothesis Tracking

MOTA Multiple Object Tracking Accuracy

MOTP Multiple Object Tracking Precision

NN Nearest Neighbor

PDA Probabilistic Data Association

PF Particle Filter

TCP Transmission Control Protocol

UDP User Datagram Protocol

1
Introduction

In existing intelligent vehicles, every vehicle has a set of sensors whose data are merged into their own Local Dynamic Map (LDM). In the action of autonomous driving, the vehicles will make use of the LDM in order to drive in a safe manner. The concept of this project is that the information of all actors is sent to a remote central server, where this data are transformed into a detailed and accurate Global Dynamic Map (GDM). The possibilities, but also the problems lay in the physical separation, that the data are collected from multiple sources, but also that the data are processed remotely, in a high-end ITS. However, the physical separation induces transmission delay in the model, which reduces the overall performance. Through simulations, we show the improvements of cooperating cars, but also the magnitude of the deterioration that the delay poses.

The benefit of cooperating cars can be seen in Figure 1.1. For example, the LDM of Vehicle 1 is the red area, whereas its GDM is the union of all colored circles. Note that vehicle 1 is unaware of any pedestrians in its LDM, but gains knowledge of three pedestrians through its GDM. Thus, through information sharing, the accuracy and awareness can be enhanced through data fusion, rather than upgrading the quality of the sensors.

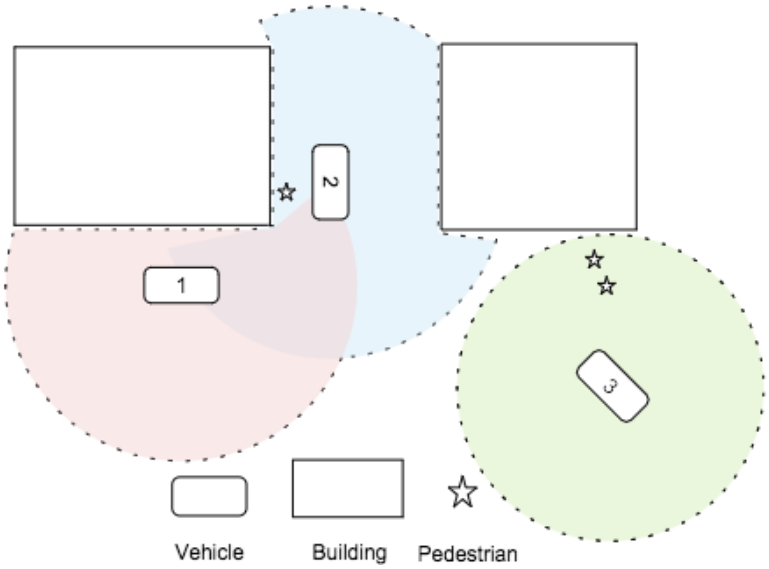

Figure 1.1: A traffic scenario with three vehicles, two buildings and three pedestrians. The vehicle's local dynamic map is plotted in a colored circle around itself, and its global dynamic map consists of the area within the dashed lines.

1.1 Related Work

Ever since one of the most notable contributions within Sensor Fusion was made by Kalman [23] in 1960, the topic has been applied to multiple areas. The area of this thesis, data fusion within the Automotive industry, has been covered in numerous papers. A common approach is taken by, e.g., Villysson [34], is to implement a Kalman Filter that merges the data from the vehicle's own sensors. Vehicles that use this fusion algorithm run their algorithm independently of the others, meaning that there is no communication between the vehicles. In a similar problem, Aeberhard et al. [3] uses Information Matrix Fusion, but instead focuses on the problem when the sensors operate at different frequencies. Blanc et al. [12] implements and compares the Extended Kalman Filter (EKF) and the Particle Filter (PF), and finds that the EKF is more suitable for a real-time system due to the computational burden of a PF. Bar-Shalom and Campo [7] presented a technique for fusing the state vectors that each individual sensor produces, which we extended in our work to comprise each individual vehicle instead. Chong et al. [15] implemented an optimal fusion algorithm under an arbitrary communication pattern, but without a transmission delay and assumption of no process noise.

In many approaches to data fusion within Automotive, the main challenges have been to maximize the accuracy of the fused output. Additionally, most approaches have only considered data fusion on a *single* vehicle, meaning that information sharing between vehicles is rare. Consequently, multiple systems have been operating *offline*, which means that the communication delay has not been an issue. However, there are some examples such as the surveillance system created by Okello and Challa [26], that provide on-line communication in order to fuse data from multiple sources. The communication between actors within Automotive has not generally concerned Sensor Fusion, but typically Collision Warning systems, such as the ones found in [10] and [36].

1.2 Challenges

Our work spans over a large area and obliges to combine contributions from multiple studies into a single model. Essentially, we want to provide on-line fusion on a remote central server. Thus, the main challenges are condensed into the following areas:

Accuracy: the main assumption is that global data fusion will provide better position estimates than using local data fusion. The goal is to show that collaborative data fusion significantly improves position estimates for the involved actors.

Communication: since the data fusion is performed on-line, there is a need to analyze how the network delay affects the server's performance. More specifically, the network delay can be decomposed to its mean and variance, and we will investigate how they affect the speed and accuracy of the tracking algorithm.

Performance: an important factor in a fusion server is its speed, and we want to provide a pilot implementation with real-time execution. We design a fusion architecture, locate bottlenecks in this pilot, and investigate the scalability of the fusion server.

1.3 Validation

To evaluate the challenges proposed, we first look into the precision of the output of the fusion server. To get an average positional error, we compare the estimated position with the ground truth. Then, we study the precision when the vehicles cooperate with each other in a global fusion, and when they keep their information to themselves.

In order to investigate the effect of the transmission delay, we separate the transmission delay into its mean and variance. We test how the precision is affected with increasing the transmission delay, while keeping the variance fixed. The effect of the variance is tested through fixing the mean transmission delay, and calculating the average positional error when the delay span is increased.

The speed of the fusion server is initially located through counting the average time the process spends in each part of the server throughout the simulation. When the bottlenecks are found, optimization will be conducted before simulating the scalability of the fusion server. It maintains real-time execution if the total processing time in the fusion server is lesser than the total simulation time. This shows that the data packets are processed faster than the newer data packets arrive.

All simulations are done on two scenarios, one with monotone driving, and one with more turns and interactions between the vehicles. Ideally, the results from both scenarios should show the same pattern in terms of accuracy and speed.

1.4 Our Contributions

Our proposition has been to provide an on-line system, consisting of an unknown amount of sources sending data to a remote central server, where the data are fused. The unique approach has mainly been the physical separation of the data providers and the consumer in a track fusion problem. Thus, our fusion architecture is presented to demonstrate possibilities of an on-line fusion along with performance and scalability, and also how it is affected by the network delay. But more importantly, our fusion server also indicates bottlenecks of the fusion architecture, which reflect what possibly could be optimized further.

The results showed that fused output was considerably better than when the cars are not collaborating. When we introduced a network delay to the system, we saw

a clear correlation between the network delay and the accuracy. However, a transmission delay is not only a fixed length, but a random entity within a span. The problem that this causes is that the data packets might arrive at the fusion server in a non-chronological order. Two methods were implemented to handle this, but the superior one was a method referred to as Rollback. The study shows that the delay variance degrades the accuracy more seriously than the mean delay, and causes in significant speed issues.

1.5 Scope

Track fusion: data fusion may refer to two domains: Sensor Fusion and Track Fusion. This division originates from the fact that sensors are all spatially placed at different locations on the vehicle, and in addition all vehicles are spread geographically. Sensor Fusion refers to local fusion at every vehicle, whereas Track Fusion is when the data are fused again, in a higher hierarchy. This project will only cover Track Fusion and not Sensor Fusion.

Server feedback: since the simulated paths are done ahead of time, the actors are unable to adjust their tracks according to the feedback. Thus, the server will not be able to communicate back to the actors.

Fault tolerance: we consider a system in which the transport layer is reliable, i.e., no packet loss and assume a packet delivery of every packet within a bounded period. We do however consider out of order packet delivery. Moreover, the server is always providing service at all times, i.e., the server does not crash.

Congestion: when the fusion server is unable to maintain real-time execution, it gets flooded with data packets that are accumulated into a queue. This deteriorates the performance, and might also lead to lost data packets. This project will not cover the problems that congestion pose.

Network security: an adversary is an anonymous client who aims to either modify or corrupt the services from the server. We assume a system without any adversaries.

2
Background

Our system consists of two significant structures: a client to simulate independent cars, and a fusion server to perform the data fusion. There is a clear separation between the client simulator and the fusion server, the modularity is to enable different input streams to be tested on the fusion server.

The car data were generated beforehand and consisted of the car's ground truth data. Hence, there was a need to invigorate the data to resemble independent cars sending data through the network. The cars broadcast the data to the fusion server, where the data are processed through four modules and finally fused together.

The sketch of the server architecture is shown in Figure 2.1, and further elaborated in the following sections. As explained in Section 1.5, the fusion algorithm on the client side will not be considered.

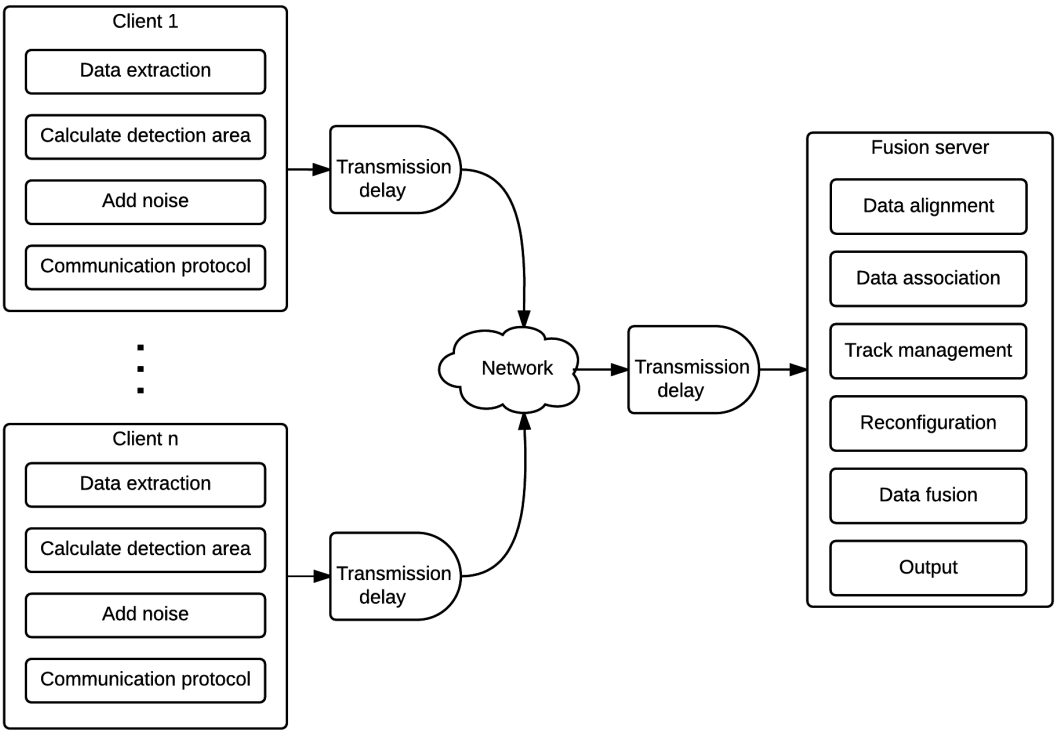

Figure 2.1: The system architecture

2.1 Client Simulation

The fusion server is driven by the arrival of data packets, which is considered as its input. In order to provide the inputs to the fusion server, we need to simulate them manually, and therefore, client simulation is needed to imitate real-world data in a real-time scenario.

We were provided data from Volvo Car Corporation, which were generated from their internal simulation environment. But it lacks some traits of an actual traffic scenario, such as the transmission delay between the clients and the fusion server. Additionally, the client input only contained the ground truth data, but sensor measurements in reality are tainted by noise. Therefore, noise was added to resemble realistic data produced by noisy sensors. Furthermore, since all cars supposedly are independent, we also need to provide them independent and asynchronous communication to the fusion server.

Moreover, the client input only contained ground truth state variables, where the state of a car could be the coordinates, velocity, acceleration, etc. Therefore, the client simulation also needs to determine what the cars will detect. To figuratively mimic car sensors, a car detects another car if it is within a certain radius and angle from itself. Hence, the data broadcast from each car is composed of ego-data and possibly data about other objects.

Since we sent the state variables to the fusion server and not raw sensor data, it implied that the data already had been fused one time. Thus, we figuratively had a local on-board fusion on every vehicle that sent the tracks to a remote central server, where they were fused again. This fusion architecture is also known as Hierarchical Fusion Architecture [25]. It bears the benefit of reducing the amount of data that needs to be transmitted, keeping the communication to a minimum. This would supposedly prevent, or at least alleviate a potential bottleneck.

2.2 Fusion Server

Data fusion is a process of an integration of multiple data from various sources into a holistic and coherent representation. In this case, the data are figuratively generated at the vehicles, by multiple devices such as GPS, radars, cameras, etc. Merging the data from these sensors are also known as Sensor Fusion [19]. When two sensors detect the same object, they will most likely not position this object at the exact same global coordinates due to noise. The aim of the Sensor Fusion is to predict the most likely state, using overlapping data from the sensors. Additionally, it is possible to merge these local estimates in a remote central server to retrieve a global estimate. However, the tracking algorithm in the fusion server does not only perform data fusion, but consists of several steps to process the data before the actual data fusion.

There are many ways to compose a tracking algorithm, approaches that may not necessarily be better than any other. For example, the fusion could solely be done at a local level, or only in a remote central server. Additionally, we need to decide which partners are allowed to communicate with each other, and if that should be a two-way communication. Other choices and options regarding the implementations will be discussed in Chapter 3, and will demonstrate the myriad of paths when designing a tracking algorithm.

When client input arrives at the fusion server, it starts a fusion process in the tracking algorithm. Since the process primarily follows a sequential schema, the tracking algorithm may be divided into several modules. Our fusion server consists of the following modules:

Data alignment: to manipulate the data so they are aligned in time, making them comparable.

Data association: to associate measurements to the correct tracks in the fusion server.

Track management: to validate tracks and to adjust the number of tracks in the server.

Reconfiguration: to adjust parameters in the data fusion according to the type of sensor setup.

Data fusion: to merge observations of objects together, in order to estimate their position as accurate as possible.

2.3 Communication

A crucial component to the server architecture is the network communication between clients and the server. That is, *when* the data are transmitted, and *how* the data are transferred between end-to-end. Ultimately, these choices affect the performance of the fusion server, and a deciding factor of how the algorithms are designed.

2.3.1 Communication Strategy

Safety-critical systems, especially within traffic scenarios are oftentimes reliant on getting information from every actor in a periodic and frequent manner, coupled with a low latency. One approach is to fuse the data as new information is received [16]. In the case of Figure 2.2, the Fusion server awaits data from the clients. Upon receiving one data packet from one client, the server starts the first fusion F_1. The listening and fusion continue in a similar manner for all other data packets. This fusion scheme is referred to as “asynchronous data fusion”, and is the strategy that we chose to implement.

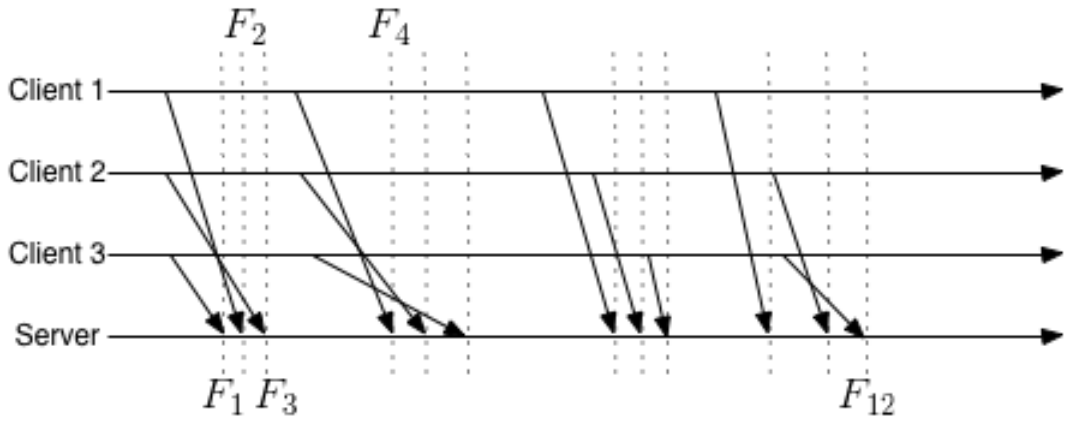

Figure 2.2: Asynchronous data fusion

However, for a remote central server, problems arise when the number of cars that the ITS handle gets too high. Since the actors send the data to the fusion server, the rate at which the server needs to fuse can get critically high.

Synchronous data fusion is another strategy which requires a fewer number of fusions. Instead of fusing when a data packet arrives, the data packets are synchronized to be sent to the fusion server at the same time. After all packets have arrived, they are collectively fused one time. By using more data in the data fusion, the fused output has a better accuracy. The main problem is that the server needs to know how many packets it should receive before starting the fusion. This means that the objects in the server need to be fixed or known at all times. Synchronous data fusion is promising in terms of speed and accuracy, but the inert property is an undesirable characteristic in a dynamic system, and was the main reason why this communication strategy was not adopted.

2.3.2 Communication Protocol

A communication protocol is a network mechanism that provides host-to-host communications. The communications may rely on a local network or remote networks connected by multiple routers. The most commonly known protocols are the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP) [35].

The UDP was chosen in this project to minimize network latency, since we want to control the total transmission delay. The term transmission delay should be considered as the total delay between when the data is collected to when it is collected in the fusion server. For example, this includes on-board fusion, delay in communication protocols, and delay in transportation.

Additionally, we assume that we have reliable non-FIFO communication channels with no packet loss. Also, the delivery time is bounded by a constant, since it is controlled by the client simulation.

2.4 Parallelization

The speed of a real-time system is paramount. Processing data with a slow pace degrades its precision, since it becomes outdated. Additionally, processing data at a fast pace naturally increases the scalability of the fusion server, which is a key factor. As the server performs a lot of repetitive tasks, a possibility would be to optimize it through parallelization. Parallel computing is an idea to compute a large scale of data by making use of multiple hardware-processing units simultaneously [31], but all parts of a system are seldom feasible to parallelize. Nonetheless, parallel computing always introduces an additional cost such as memory transfer overhead, thread synchronization, and thread spawning. However, these overhead time are considered as negligible when the parallelization significantly improves computation time with a large set of data.

In this project, the tracking algorithm consists of small but many matrices. This means that GPU computation may not be suitable due to the memory transfer overhead, in which the data are transferred between the shared memories in the CPU and the GPU. Instead, multithreading may be an alternative optimization step for the fusion server. Multithreading uses multiple threads of the CPU, and may perform tasks in parallel. Note that the parallelization using multithreading is therefore limited to the number of threads in the CPU. For example, if the CPU have 8 threads, and there are a parallelizable block with 64 tasks, the gain from multithreading is at most 8 times faster than sequential computing.

3
Tracking Algorithm

To effectively and accurately track objects in a system has been an issue within areas such as surveillance, automotive, and robotic industry for a long time [16, 18, 25, 30, 34, 37]. These systems may have multiple measurements of each object which necessarily do not completely match. The errors originate from an inaccuracy of the sensors, which seldom produce data that match the ground truth. The problem occurs when merging this data, and then predicting the position of the dynamic objects in a probabilistic manner.

3.1 Tracking Model

Several techniques have been developed to approach the track fusion problem, such as Maximum Likelihood and Maximum Posterior, Particle filters, Covariance consistency methods, and Bayesian tracking models. The Bayesian tracking theory is common in the track fusion area [16, 27, 34], and will be the model of choice in this project.

The concept of Bayesian tracking theory is to gather and combine the information from the multiple actors in order to predict the past, current and future true "states" of the involved objects. A state contains kinematic information such as position, velocity, and acceleration. The true state of all objects at time t is denoted as $\mathbf{X}_t$, that is:

$$\mathbf{X}_t = [(\mathbf{X}_t^1)^T, (\mathbf{X}_t^2)^T, ..., (\mathbf{X}_t^{n(t)})^T]^T$$

where $n(t)$ is the number of tracked objects in the system at time t, and ^{T} denotes the matrix transpose. $\mathbf{X}_t$ is also referred to as the "ground truth", and is the data that need to be estimated as good as possible, using only noisy sensor measurements. These measurements are provided to the tracking algorithm at every time instance and are denoted $\mathbf{y}_t \in \mathbb{R}^{n_y(t)}$, where $n_y(t)$ is the number of measurements at time t. Furthermore, all measurements up until time t are defined as:

$$\mathbf{Y}_{1:t} = [(\mathbf{y}_1)^T, (\mathbf{y}_2)^T, ..., (\mathbf{y}_t)^T]^T$$

Thus, if $\mathbf{X}_t$ is estimated with the use of the measurement data up to time k, it is denoted as $\hat{\mathbf{X}}_{t|k}$. Simply put, the goal of a tracking algorithm is to reproduce the estimate $\hat{\mathbf{X}}_{t|k}$ as close to $\mathbf{X}_t$ as possible.

The tracking model in this project concerns preventive safety systems and autonomous-drive pilots, and therefore, the parameters that need to be tracked are the information about the kinematics of each object i:

$$\mathbf{X}_t^i = [x_t^i, y_t^i, \dot{x}_t^i, \dot{y}_t^i, \ddot{x}_t^i, \ddot{y}_t^i]$$

where $\mathbf{X}_t^i$ is the state vector for object i, and $\dot{x}_t^i$ is the first time derivative of x_t^i (velocity) and $\ddot{x}_t^i$ is the second time derivative (acceleration).

As described in Section 2.3.1, a fusion process will be initialized as soon as a new data packet arrives. But irrespective of *when* to fuse, there are some sequential steps in the tracking algorithm that needs to take place before the actual data fusion. These steps can be divided into several modules, and a schematic overview of these and the resulting data flow can be seen in Figure 3.1:

Figure 3.1: How the data is processed when a fusion process is initialized.

The aim of the tracking algorithm is to get as good estimate as possible of $\mathbf{X}_t$. The estimate $\hat{\mathbf{X}}_t$ is calculated by getting the expected value of $\mathbf{X}_t$ given $\mathbf{Y}_{1:t}$. However, $\mathbf{X}_t$ is a difficult quantity to estimate, but one way to enhance the performance is to make assumptions of the underlying processes that govern the model. More precisely, assumptions are made on the motion of the objects, but also the sensor characteristics. In Bayesian tracking theory, these processes are oftentimes modeled as:

$$\mathbf{X}_t = \mathbf{f}_{t-1}(\mathbf{X}_{t-1}, \mathbf{q}_{t-1}) \tag{3.1}$$
$$\mathbf{y}_t = \mathbf{h}_t(\mathbf{X}_t, \mathbf{r}_t) \tag{3.2}$$

That is, $\mathbf{X}_t$ in Eq. (3.1) is a function of both its previous state and $\mathbf{q}_{t-1}$, which is a discrete time zero-mean Gaussian white noise process. This noise process is also referred to as "process noise", and stems from the fact that the actors are mainly controlled by humans which in general do not drive smoothly. The measurement model in Eq. (3.2) assumes that the measurements $\mathbf{y}_t$ are related to the true state $\mathbf{X}_t$ and the stochastic noise process $\mathbf{r}_t$. This process is also a discrete time zero-mean Gaussian white noise process and describes measurement noise as well as sensor modelling uncertainty [16, 29, 34].

3.2 Data Alignment

When a data packet arrives at the remote central server, the first thing is to align the data [14, 25]. Even though the packet only contains information about a few

actors, all tracks will be processed. The data will be outdated for all tracks, and needs to be aligned to the correct time. Looking at the scenario in Figure 3.2, Client 1 sends a data packet at time B that arrives in the server at the time C. The most recent fusion was induced by Client 2 at time A, and thus, we need to align the tracks from time A to time B. Note that in this case the fusion process is started at time C, which is the current time, but the fusion is performed for time step B. A temporary prediction to time step C is performed in the end, to get a fused output of the current time.

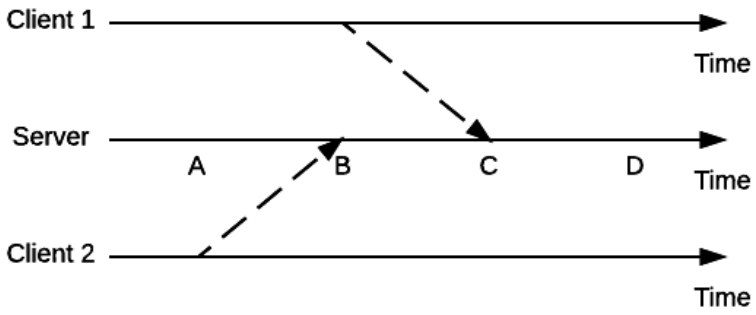

Figure 3.2: Clients communicate with the server, and the data packets arrive in chronological order.

However, an intricate problem arises when the delivery of some measurements are delayed, making the data packets arrive in a non-chronological order. That is, when a measurement is taken at a time A, but arrives much later, at the time D. This implies that there may be other clients whose data packets are transmitted and received in between this time period. A scenario can be seen in Figure 3.3:

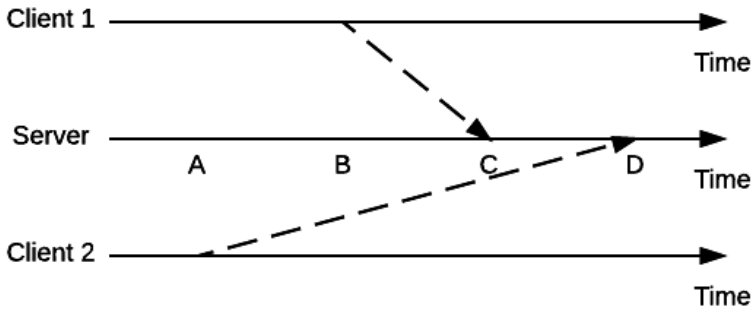

Figure 3.3: Client 2 sends its data at time A, but it arrives at time D. During this time, Client 1 has delivered one data packet between time A and D

There are a few approaches to this problem, and the first suggestion is to perform a rollback of the calculations to the delayed measurement's chronological place, and then recalculating the filters again. This method is simply referred to as Rollback, and it has an optimal solution, which means that the data provided are fully used [24]. However, this method has a high computational burden, especially if there is an actor that persistently delivers delayed measurements. Another approach is taken by Alexander [4], where it suffices to calculate a correction term when the delayed measurement arrives. This method is both computationally cheaper and optimal. However, it requires the measurement sensitivity matrix to be available at the current time, but this is not always the case [24]. Larsen et al. [24] implemented a method, referred to as Extrapolating the measurement (EtM), where the delayed measurement is extrapolated into the current time step, and then fused normally. This method has a low computational cost but produces suboptimal results.

3.3 Data Association

For every data packet, the sender knows its own identification number, its own ego-motion, and perhaps has some measurements of some other actors. However, the cars can not make confident estimates of which object it sees, but only that it sees *an* object. However, the remote central server keeps data for all the tracks, and can match the unidentified measurements with existing tracks.

There are some strategies for associating the measurements with the tracks, and the simplest way would be to calculate the Euclidean distance between each track and every measurement. Then, the track and measurement with the shortest distance are matched together, and this is sequentially done until no more matches can be made. This strategy is called Nearest Neighbor algorithm (NN) and is a greedy algorithm [14, 16]. A similar approach would be to consider all the distances between all tracks and measurements at once, then match as many measurements to tracks as possible, and find the combination that minimizes its total distance. This branch of methods is called Global Nearest Neighbor algorithms (GNN), and Blackman and Popoli [11] suggest that the Auction algorithm is a suitable candidate for a track fusion problem.

A probabilistic approach proposed by Bar-Shalom and Tse [8] is called Probabilistic Data Association (PDA). This algorithm associates a probability to all possible hypotheses. An extension to PDA is the Joint Probabilistic Data Association (JPDA), and differs from PDA by considering all of the measurements and all of the tracks at once [20]. When assigning a probability to a hypothesis, it considers the fact that this measurement can not be generated by other tracks. Lastly, Multi-Hypothesis Tracking (MHT) will estimate all possible hypotheses and keep all hypotheses each iteration. Then, MHT will utilize all its previous hypotheses in order to make an association [14, 28, 34].

Gating

One way of making the data association more effective, and remove unlikely track-measurement associations is to introduce a validation gate [11, 16, 20, 25]. The basic idea is to create a gate around a track, and only allow measurements that lie inside the gate to be associated with that track. The gate is an approximation of the uncertainty of how far the track could have traveled, and thus position, speed, and acceleration are taken into account when calculating the gate size.

An alternative is to use an elliptic gate, which makes assumptions of the physical restrictions of the vehicles. Since the vehicles have limited turning radius, it assumes more uncertainty in the vehicle's heading. Additionally, the results of the elliptic gate are used in the Auction algorithm. However, as the elliptic gate and Auction algorithm synergies with each other, they fit badly with the Information filter. In fact, both methods are created to be efficient with the Kalman filter.

Therefore, we opted for an easier approach, which would be to implement a circular gate. However, this implies that no assumptions are made on the motion of the track, that it is equiprobable to move in any direction. The benefit is that it is a computationally fast way to exclude improbable track-measurement pairs from the very beginning. In Figure 3.4, two measurements lie inside the gate, whereas two measurements lie outside the gate. The ones that are inside the gate are allowed to be matched to the track.

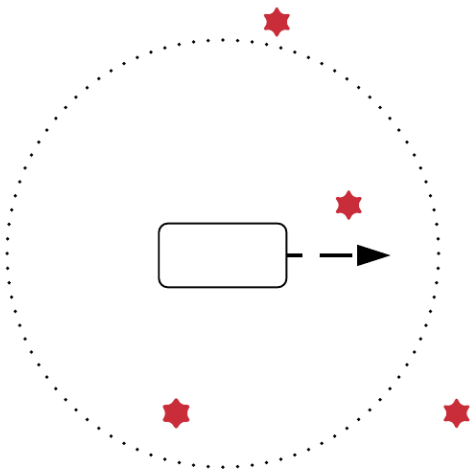

Figure 3.4: The track is visualized as a rectangle, and the red stars are the measurements. All measurements that lie inside the gate can be matched with the track.

3.4 Track Management

In the case where a measurement is unable to get matched with a track in the data association step, it is assumed to be a measurement of a new object. Thus, a new

track is created with the measurement data, and is initially a "tentative track". A new track will be initialized *after* Data Fusion, since it needs historic data to be updated from. The first measurement is taken as the truth since no better estimation is available.

A tentative track needs to be detected several times within a short period of time, or will otherwise be considered as a false detection and be removed. The other track type is called a "validated track", whose existence is more certain, and does not need to be detected within a short period of time again.

Since a tentative track has been tracked for a short period of time, its motion and existence is more uncertain. Therefore, it may have a larger validation gate than a validated track in the data association process. However, if a tentative track is associated a number of times within a short period of time, it gets upgraded to a validated track.

The last task of the track management is to make sure that tracks that are irrelevant get deleted from the server. This means that tracks that have left the operating area of the remote central server will be removed. This is necessary since the performance reduces when filters accumulate and memory is unnecessarily used.

3.5 Reconfiguration

Different kind of sensors has a different kind of properties. Some sensors are suited for detecting range, whereas some are better at detecting the relative angle to the ego-vehicle, or new sensor models may simply be more accurate than older versions. Depending on how much the sensors accurateness is trusted, the reconfiguration module should alter the weights of the noise matrix in the data fusion.

Additionally, the supplier of the sensors should specify how the noise of the measurement data is distributed. In Bayesian tracking theory, the measurement noise is assumed to be Gaussian [29], but the reconfiguration module could customize the sensor model for each individual sensor setup.

3.6 Data Fusion

The Data fusion in Bayesian tracking theory can be divided into the following steps: a filtering step to get the posterior distribution $p(\mathbf{X}_t|\mathbf{Y}_{1:t})$, and a prediction step to get the estimate $p(\mathbf{X}_{t+1}|\mathbf{Y}_{1:t})$ [16, 29, 34]. These two steps will take place on every track, that is, there will be an individual filter for every track. General theory regarding Kalman Filtering can be found in [29], and in particular [16].

3.6.1 Filtering

Filtering consists of a predictive step to get $p(\mathbf{X}_t|\mathbf{Y}_{1:t-1})$, and then an update step to get the posterior distribution $p(\mathbf{X}_t|\mathbf{Y}_{1:t})$.

In the prediction step, it is assumed that the density $p(\mathbf{X}_{t-1}|\mathbf{Y}_{1:t-1})$ is known from the previous recursion, and in the very first iteration an *a priori* estimate of $p(\mathbf{X}_0|\mathbf{Y}_0)$ is carefully chosen. Then, by using the motion model in Eq. (3.1) and the Chapman-Kolmogorov equation, the prediction, $p(\mathbf{X}_t|\mathbf{Y}_{1:t-1})$, becomes:

$$\begin{aligned} p(\mathbf{X}_t|\mathbf{Y}_{1:t-1}) &= \int p(\mathbf{X}_t|\mathbf{X}_{t-1}, \mathbf{Y}_{1:t-1}) p(\mathbf{X}_{t-1}|\mathbf{Y}_{1:t-1}) \, \mathrm{d}\mathbf{X}_{t-1} \\ &= \int p(\mathbf{X}_t|\mathbf{X}_{t-1}) p(\mathbf{X}_{t-1}|\mathbf{Y}_{1:t-1}) \, \mathrm{d}\mathbf{X}_{t-1} \end{aligned} \tag{3.3}$$

Note that the relation $p(\mathbf{X}_t|\mathbf{X}_{t-1}, \mathbf{Y}_{1:t-1}) = p(\mathbf{X}_t|\mathbf{X}_{t-1})$ holds, since the motion model (Eq. (3.1)) that governs $\mathbf{X}_t$, has the Markov property. Then, when a new measurement $\mathbf{y}_t$ at time step t arrives, the predicted density in Eq. (3.3) is updated. Since the measurements $\mathbf{y}_t$ are independent given $\mathbf{X}_t$, Bayes rule can be used to retrieve the posterior distribution $p(\mathbf{X}_t|\mathbf{Y}_{1:t})$,

$$\begin{aligned} p(\mathbf{X}_t|\mathbf{Y}_{1:t}) = p(\mathbf{X}_t|\mathbf{y}_t, \mathbf{Y}_{1:t-1}) &= \frac{p(\mathbf{y}_t|\mathbf{X}_t, \mathbf{Y}_{1:t-1}) p(\mathbf{X}_t|\mathbf{Y}_{1:t-1})}{p(\mathbf{y}_t|\mathbf{Y}_{1:t-1})} \\ &= \frac{p(\mathbf{y}_t|\mathbf{X}_t) p(\mathbf{X}_t|\mathbf{Y}_{1:t-1})}{p(\mathbf{y}_t|\mathbf{Y}_{1:t-1})} \end{aligned} \tag{3.4}$$

Analytical solutions for Eq. (3.3) and Eq. (3.4) are in general not retrievable unless some assumptions are made about the underlying processes. In this project, the assumption is that the motion and measurement models are linear with additive Gaussian noise, which is known as the Kalman filter.

3.6.2 Prediction

In the Kalman filter the assumption is that the motion model Eq. (3.1) and the measurement model Eq. (3.2) can be expressed as the linear system:

$$\mathbf{X}_t = \mathbf{F}_{t-1}\mathbf{X}_{t-1} + \mathbf{q}_{t-1} \tag{3.5}$$

$$\mathbf{y}_t = \mathbf{H}_t\mathbf{X}_t + \mathbf{r}_t \tag{3.6}$$

where $\mathbf{q}_{t-1} \sim \mathcal{N}(0, \mathbf{Q}_{t-1})$ and $\mathbf{r}_t \sim \mathcal{N}(0, \mathbf{R}_t)$. In the Kalman filter, it is assumed that the prior distribution $p(\mathbf{X}_0|\mathbf{y}_0)$ is Gaussian. By using the properties of linear Gaussian systems, the densities of Eq. (3.3) and Eq. (3.4) will also be Gaussian [16], that is,

$$p(\mathbf{X}_{t-1}|\mathbf{Y}_{1:t-1}) = \mathcal{N}(\hat{\mathbf{X}}_{t-1|t-1}, \mathbf{P}_{t-1|t-1}) \tag{3.7}$$

$$p(\mathbf{X}_t|\mathbf{Y}_{1:t-1}) = \mathcal{N}(\hat{\mathbf{X}}_{t|t-1}, \mathbf{P}_{t|t-1}) \tag{3.8}$$

$$p(\mathbf{X}_t|\mathbf{Y}_{1:t}) = \mathcal{N}(\hat{\mathbf{X}}_{t|t}, \mathbf{P}_{t|t}) \tag{3.9}$$

Where $\mathbf{P}_{\cdot|\cdot}$ are the covariance matrices of the Kalman filter, and $\hat{\mathbf{X}}_{\cdot|\cdot}$ are the estimates of the true states. Thus, the only parameters that needs to be tracked and calculated are the mean and variance of the densities.

Now, according to [16], by using Eq. (3.5) and Eq. (3.7), the prediction becomes:

$$\mathbf{P}_{t|t-1} = Cov(\mathbf{X}_t|\mathbf{Y}_{1:t-1}) = \mathbf{F}_{t-1}\,\mathbf{P}_{t-1|t-1}\,\mathbf{F}_{t-1}^T + \mathbf{Q}_{t-1} \tag{3.10}$$

$$\hat{\mathbf{X}}_{t|t-1} = \mathbb{E}[\mathbf{X}_t|\mathbf{Y}_{1:t-1}] = \mathbf{F}_{t-1}\,\hat{\mathbf{X}}_{t-1|t-1} \tag{3.11}$$

Then, to get to the update step, a measure called innovation needs to be introduced. The innovation $\tilde{\mathbf{y}}_t$ describes the new information in $\mathbf{y}_t$ that is uncorrelated with all the previous measurements $\mathbf{Y}_{1:t-1}$, that is:

$$\tilde{\mathbf{y}}_t = \mathbf{y}_t - \mathbb{E}[\mathbf{y}_t|\mathbf{Y}_{1:t-1}] \tag{3.12}$$

which is $\tilde{\mathbf{y}}_t \sim \mathcal{N}(0, \mathbf{S}_t)$, and where the innovation covariance $\mathbf{S}_t$ is:

$$\mathbf{S}_t = Cov\{\tilde{\mathbf{y}}_t|\mathbf{Y}_{1:t-1}\} = \mathbf{H}_t\mathbf{P}_{t|t-1}\mathbf{H}_t^T + \mathbf{R}_t \tag{3.13}$$

and hence the final update step can be formulated as:

$$\mathbf{P}_{t|t} = (\mathbf{I} - \mathbf{K}_t\mathbf{H}_t)\mathbf{P}_{t|t-1} \tag{3.14}$$

$$\hat{\mathbf{X}}_{t|t} = \hat{\mathbf{X}}_{t|t-1} + \mathbf{K}_t(\mathbf{y}_t - \mathbf{H}_t\hat{\mathbf{X}}_{t|t-1}) \tag{3.15}$$

where $\mathbf{K}_t$ is known as the Kalman gain and defined as:

$$\begin{aligned}\mathbf{K}_t &= \mathbf{P}_{t|t-1}\mathbf{H}_t^T\mathbf{S}_t^{-1} \\ &= \mathbf{P}_{t|t-1}\mathbf{H}_t^T(\mathbf{H}_t\mathbf{P}_{t|t-1}\mathbf{H}_t^T + \mathbf{R}_t)^{-1}\end{aligned} \tag{3.16}$$

The Kalman filter is said to operate in the "covariance domain", and filter that operates in the inverse covariance domain, or the "information domain" is the Information filter [29]. The Information filter is closely related to the Kalman filter [21], but is manipulated using linear algebra to operate with inverse covariance matrices [6, 13]. The benefit from using the Information Filter is that the matrices become computationally easier to work with, and is in general more sparse.

3.6.3 The Information Filter

To arrive at the Information filter, there are a few parts in the Kalman filter that needs to be changed: the computation of the error covariance (Eq. (3.14)), update of the state estimates (Eq. (3.15)), prediction of the error covariance (Eq. (3.10)), and lastly the prediction of the state estimate (Eq. (3.11)).

Error Covariance Computation

The inverse error covariance is retrieved by manipulating the error covariance Eq. (3.14) as follows:

$$\begin{aligned}\mathbf{P}_{t|t} &= (\mathbf{I} - \mathbf{K}_t\mathbf{H}_t)\mathbf{P}_{t|t-1} \\ \mathbf{K}_t\mathbf{H}_t\mathbf{P}_{t|t-1} &= \mathbf{P}_{t|t-1} - \mathbf{P}_{t|t}\end{aligned} \tag{3.17}$$

then reworking the Kalman gain that is given in Eq. (3.16):

$$\begin{aligned} \mathbf{K}_t &= \mathbf{P}_{t|t-1}\mathbf{H}_t^T(\mathbf{H}_t\mathbf{P}_{t|t-1}\mathbf{H}_t^T + \mathbf{R}_t)^{-1} \\ \mathbf{K}_t(\mathbf{H}_t\mathbf{P}_{t|t-1}\mathbf{H}_t^T + \mathbf{R}_t) &= \mathbf{P}_{t|t-1}\mathbf{H}_t^T \\ \mathbf{K}_t\mathbf{H}_t\mathbf{P}_{t|t-1}\mathbf{H}_t^T &= \mathbf{P}_{t|t-1}\mathbf{H}_t^T - \mathbf{K}_t\mathbf{R}_t \end{aligned} \tag{3.18}$$

Eq. (3.17) can then be inserted into Eq. (3.18):

$$\begin{aligned} (\mathbf{P}_{t|t-1} - \mathbf{P}_{t|t})\mathbf{H}_t^T &= \mathbf{P}_{t|t-1}\mathbf{H}_t^T - \mathbf{K}_t\mathbf{R}_t \\ \mathbf{K}_t\mathbf{R}_t &= \mathbf{P}_{t|t}\mathbf{H}_t^T \\ \mathbf{K}_t &= \mathbf{P}_{t|t}\mathbf{H}_t^T(\mathbf{R}_t)^{-1} \end{aligned} \tag{3.19}$$

Now, Eq. (3.19) is substituted into the error covariance update (Eq. (3.14)):

$$\begin{aligned} \mathbf{P}_{t|t} &= (\mathbf{I} - \mathbf{K}_t\mathbf{H}_t)\mathbf{P}_{t|t-1} \\ \mathbf{P}_{t|t}(\mathbf{P}_{t|t-1})^{-1} &= \mathbf{I} - \mathbf{K}_t\mathbf{H}_t \\ \mathbf{P}_{t|t}(\mathbf{P}_{t|t-1})^{-1} &= \mathbf{I} - \mathbf{P}_{t|t}\mathbf{H}_t^T(\mathbf{R}_t)^{-1}\mathbf{H}_t \\ \mathbf{I} &= \mathbf{P}_{t|t}\Big((\mathbf{P}_{t|t-1})^{-1} + \mathbf{H}_t^T(\mathbf{R}_t)^{-1}\mathbf{H}_t\Big) \end{aligned} \tag{3.20}$$

and by multiplying by $(\mathbf{P}_{t|t})^{-1}$, the inverse error covariance becomes:

$$(\mathbf{P}_{t|t})^{-1} = (\mathbf{P}_{t|t-1})^{-1} + \mathbf{H}_t^T(\mathbf{R}_t)^{-1}\mathbf{H}_t \tag{3.21}$$

State Estimate Update

Next quantity to rewrite is the State Estimate Update, which needs to be expressed in terms of $(\mathbf{P}_{t|t})^{-1}$. Hence, Eq. (3.15) is firstly multiplied by $(\mathbf{P}_{t|t})^{-1}$:

$$\begin{aligned} \hat{\mathbf{X}}_{t|t} &= \hat{\mathbf{X}}_{t|t-1} + \mathbf{K}_t(\mathbf{y}_t - \mathbf{H}_t\hat{\mathbf{X}}_{t|t-1}) \\ (\mathbf{P}_{t|t})^{-1}\hat{\mathbf{X}}_{t|t} &= (\mathbf{P}_{t|t})^{-1}\hat{\mathbf{X}}_{t|t-1} + (\mathbf{P}_{t|t})^{-1}\mathbf{K}_t(\mathbf{y}_t - \mathbf{H}_t\hat{\mathbf{X}}_{t|t-1}) \end{aligned} \tag{3.22}$$

Thereafter, we use the Kalman gain in Eq. (3.19) and insert this into Eq. (3.22):

$$\begin{aligned} (\mathbf{P}_{t|t})^{-1}\hat{\mathbf{X}}_{t|t} &= (\mathbf{P}_{t|t})^{-1}\hat{\mathbf{X}}_{t|t-1} + \mathbf{H}_t^T(\mathbf{R}_t)^{-1}(\mathbf{y}_t - \mathbf{H}_t\hat{\mathbf{X}}_{t|t-1}) \\ &= (\mathbf{P}_{t|t})^{-1}\hat{\mathbf{X}}_{t|t-1} + \mathbf{H}_t^T(\mathbf{R}_t)^{-1}\mathbf{y}_t - \mathbf{H}_t^T(\mathbf{R}_t)^{-1}\mathbf{H}_t\hat{\mathbf{X}}_{t|t-1} \\ &= \Big[(\mathbf{P}_{t|t})^{-1} - \mathbf{H}_t^T(\mathbf{R}_t)^{-1}\mathbf{H}_t\Big]\hat{\mathbf{X}}_{t|t-1} + \mathbf{H}_t^T(\mathbf{R}_t)^{-1}\mathbf{y}_t \end{aligned} \tag{3.23}$$

Lastly, Eq. (3.21) is inserted into Eq. (3.23) to get the new state estimate update:

$$(\mathbf{P}_{t|t})^{-1}\hat{\mathbf{X}}_{t|t} = (\mathbf{P}_{t|t-1})^{-1}\hat{\mathbf{X}}_{t|t-1} + \mathbf{H}_t^T(\mathbf{R}_t)^{-1}\mathbf{y}_t \tag{3.24}$$

Prediction of the Error Covariance

The prediction of the error covariance in Eq. (3.10) is changed by first taking its inverse:

$$\begin{aligned} \mathbf{P}_{t|t-1} &= \mathbf{F}_{t-1}\,\mathbf{P}_{t-1|t-1}\,\mathbf{F}_{t-1}^T + \mathbf{Q}_{t-1} \\ (\mathbf{P}_{t|t-1})^{-1} &= (\mathbf{F}_{t-1}\,\mathbf{P}_{t-1|t-1}\,\mathbf{F}_{t-1}^T + \mathbf{Q}_{t-1})^{-1} \end{aligned} \tag{3.25}$$

Then, Woodbury's matrix identity in Appendix A is used:

$$(\mathbf{S}+\mathbf{UTV})^{-1} = (\mathbf{S})^{-1} - (\mathbf{S})^{-1}\mathbf{U}((\mathbf{T})^{-1} + \mathbf{V}(\mathbf{S})^{-1}\mathbf{U})^{-1}\mathbf{V}(\mathbf{S})^{-1} \tag{3.26}$$

where

$$\mathbf{S} = \mathbf{F}_{t-1}\mathbf{P}_{t-1|t-1}\mathbf{F}_{t-1}^{T},\ \mathbf{U} = \mathbf{Q}_{t-1} \text{ and } \mathbf{T} = \mathbf{V} = \mathbf{I},$$

and then, by defining $\mathbf{M}_{t-1} = (\mathbf{F}_{t-1}\mathbf{P}_{t-1|t-1}\mathbf{F}_{t-1}^{T})^{-1}$, Eq. (3.25) becomes:

$$(\mathbf{P}_{t|t-1})^{-1} = \mathbf{M}_{t-1} - \mathbf{M}_{t-1}\mathbf{Q}_{t-1}(\mathbf{I}+\mathbf{M}_{t-1}\mathbf{Q}_{t-1})^{-1}\mathbf{M}_{t-1} \tag{3.27}$$

Prediction of the State Estimates

The last equation that needs to be changed is the prediction of the state estimates, and is reworked by multiplying Eq. (3.11) with $(\mathbf{P}_{t|t-1})^{-1}$:

$$\begin{aligned} \hat{\mathbf{X}}_{t|t-1} &= \mathbf{F}_{t-1}\,\hat{\mathbf{X}}_{t-1|t-1} \\ (\mathbf{P}_{t|t-1})^{-1}\hat{\mathbf{X}}_{t|t-1} &= (\mathbf{P}_{t|t-1})^{-1}\mathbf{F}_{t-1}\,\hat{\mathbf{X}}_{t-1|t-1} \end{aligned} \tag{3.28}$$

Then, $(\mathbf{P}_{t|t-1})^{-1}$ on the right hand side is substituted by Eq. (3.27), and if $(\mathbf{Q}_{t-1})^{-1}$ exists, Eq. (3.28) becomes:

$$\begin{aligned} (\mathbf{P}_{t|t-1})^{-1}\hat{\mathbf{X}}_{t|t-1} &= \Big[\mathbf{M}_{t-1} - \mathbf{M}_{t-1}\mathbf{Q}_{t-1}(\mathbf{I}+\mathbf{M}_{t-1}\mathbf{Q}_{t-1})^{-1}\mathbf{M}_{t-1}\Big]\mathbf{F}_{t-1}\,\hat{\mathbf{X}}_{t-1|t-1} \\ &= \Big[\mathbf{M}_{t-1} - \mathbf{M}_{t-1}(\mathbf{Q}_{t-1}^{-1}+\mathbf{M}_{t-1})^{-1}\mathbf{M}_{t-1}\Big]\mathbf{F}_{t-1}\,\hat{\mathbf{X}}_{t-1|t-1} \\ &= \Big[\mathbf{I} - \mathbf{M}_{t-1}(\mathbf{Q}_{t-1}^{-1}+\mathbf{M}_{t-1})^{-1}\Big]\mathbf{M}_{t-1}\mathbf{F}_{t-1}\,\hat{\mathbf{X}}_{t-1|t-1} \end{aligned} \tag{3.29}$$

and by substituting back $\mathbf{M}_{t-1} = (\mathbf{F}_{t-1}\mathbf{P}_{t-1|t-1}\mathbf{F}_{t-1}^{T})^{-1}$, the final form is retrieved:

$$(\mathbf{P}_{t|t-1})^{-1}\hat{\mathbf{X}}_{t|t-1} = \Big[\mathbf{I} - \mathbf{M}_{t-1}(\mathbf{Q}_{t-1}^{-1}+\mathbf{M}_{t-1})^{-1}\Big](\mathbf{F}_{t-1}^{T})^{-1}(\mathbf{P}_{t-1|t-1})^{-1}\hat{\mathbf{X}}_{t-1|t-1} \tag{3.30}$$

These four new relations are used in the Information filter algorithm, the schematic idea of how the algorithm operates can be seen in Figure 3.5:

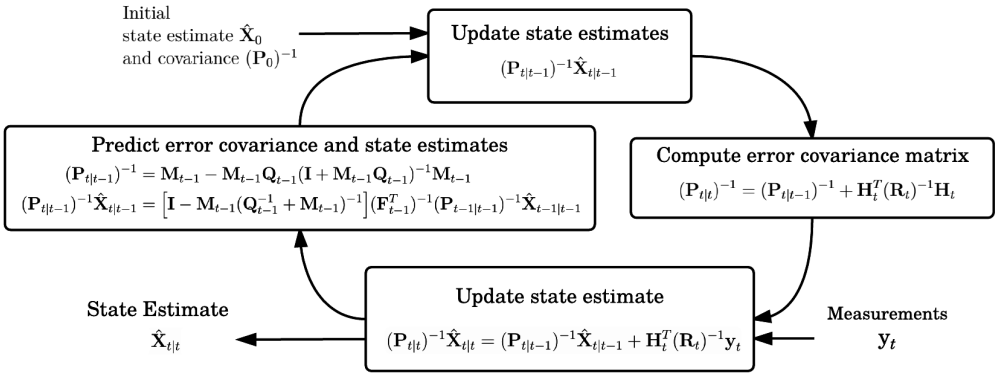

Figure 3.5: The Information Filter.

4
Implementation

Our architecture consists of two important parts: the client simulator and the fusion server. The client simulation is necessary, but secondary to our project, and thus we will only briefly explain its implementation. The focal point of our thesis is on the Fusion Server, and will be more intensively reviewed.

Both the client simulation and fusion server were written in C++, to be more easily integrated with an existing software at Volvo Car Corporation. Moreover, an integral part of a fusion server is its speed. C++ is a fast programming language since it provides efficient memory usage, low-level optimization and is closer to machine code [33].

4.1 Client Simulation

Since the client input is static data files that contain the ground truth state vectors of all cars, the client simulator needs to process the data to replicate a real traffic scenario with independent cars. Technically, each thread represents a car, and each thread performs the following operations:

- Extract ego data from the client input
- Extract data from other objects within sensor radius
- Add noise to the measurements
- Add data to a UDP packet
- Transmit the UDP packet with a specified transmission delay

More specifically, the UDP packet can be divided into segments according to Figure 4.1. Apart from the ego-position, Segment 1 will contain a unique identifier and the time stamp of the transmission. The following segments contain information of other car's state, that is, the position, velocity, and acceleration.

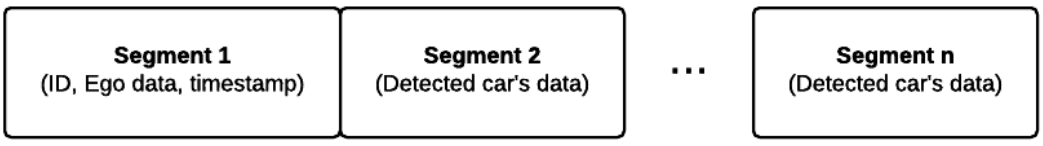

Figure 4.1: Packet segmentation

The client generates one thread for every car, and will repeatedly transmit data to the fusion server until the end of the simulation. Thus, the algorithm of the client is constructed as shown in Algorithm 1. We initially decide how often every vehicle will transmit data to the fusion server with the DT-value. Line 1 spawns the number of independent cars in the server. Every thread then initializes their own simulate_client function. Line 8 signals starting time of the simulation, and since the simulation is run in real-time, this is the starting reference for the mat_instance. The lines 10 to 21 gathers all necessary data into a data packet. Lastly, the data is

Algorithm 1: Client simulator
Input: traffic scenario data
Output: transmitted UDP packets to Fusion server

Simulation of the cars, which feeds data packets to the fusion server.

Constant: DT, α, β

```
1  threads ← new threads equal to number of cars
2  forall threads do
3  |  simulate_client(unique car_id for each thread)
4  end
5  Function simulate_client(car_id)
6  |  mat_instance ← parse client input with specific car_id to array instance
7  |  total_timestamp ← length of mat_instance
8  |  current_timestamp ← now()
9  |  while index < total_timestamp do
10 |  |  packet ← create empty UDP packet to store data
11 |  |  ego_data ← read from mat_instance at current_timestamp
12 |  |  noise ← generate normally distributed random variables
   |  |  // Segment 1
13 |  |  packet.push( unique identifier ) // add unique identifier to packet
14 |  |  packet.push( ego_data + noise )
15 |  |  packet.push( current_timestamp )
   |  |  // Segment 2...n
16 |  |  forall other cars do
17 |  |  |  other_ego_data ← read from mat_instance at current_timestamp
18 |  |  |  if other car inside my sensor radius then
19 |  |  |  |  packet.push( other_ego_data + noise)
20 |  |  |  end
21 |  |  end
22 |  |  delay ← generate uniform(α, β) distributed delay
23 |  |  new_thread ← spawn a new thread
24 |  |  new_thread.sleep(delay)
25 |  |  new_thread.send_packet(packet)
26 |  |  sleep(DT) // sleep this thread depending on sending rate DT
27 |  end
28 end
```

sent to a new thread to simulate transmission delay, while the car continues to send data packets. The transmission delay is distributed as uniform(α, β), and is predefined.

Through this setup, we can regulate the data and its flow. For example, the noise, the transmission delay, how often the cars transmit the data, a number of cars in the simulation, and what objects a car detects. Thus, this client simulator enables investigations to be made under a controlled environment.

4.2 Fusion Server

The fusion server has five important modules: Data Alignment, Data Association, Track Management, Reconfiguration and Data Fusion. All of whom consists of algorithms that are explained in the following sections.

4.2.1 Data Alignment

The Data Alignment primary duty is to align the filters to the correct time. This means that the filters need to be moved from the time of the previous fusion to the time of the current fusion. Also, the Data Alignment manages data packets that arrive in a non-chronological order.

The underlying assumption is that the actors move with constant acceleration, which is known as the *Constant acceleration model*. This means that the *acceleration increments* are assumed to be zero-mean Gaussian white noise process, which suggests that Eq. (3.5) takes the form:

$$\mathbf{X}_t = \mathbf{F}_{t-1}\,\mathbf{X}_{t-1} + \mathbf{G}\,\mathbf{q}_{t-1} \tag{4.1}$$

where

$$\mathbf{F}_{t-1} = \begin{bmatrix} 1 & 0 & T_s & 0 & \frac{T_s^2}{2} & 0 \\ 0 & 1 & 0 & T_s & 0 & \frac{T_s^2}{2} \\ 0 & 0 & 1 & 0 & T_s & 0 \\ 0 & 0 & 0 & 1 & 0 & T_s \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \text{ and } \quad \mathbf{G} = \begin{bmatrix} \frac{T_s^2}{2} & 0 \\ 0 & \frac{T_s^2}{2} \\ T_s & 0 \\ 0 & T_s \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

and where T_s is the time since the last update of the ego-vehicle, and is a variable that varies throughout the simulation. The variable $\mathbf{Q}_{t-1}$, or the covariance matrix of the noise term in Eq. (4.1) was replaced with the following diagonal matrix:

$$\mathbf{Q}_{t-1} = Cov\{\mathbf{G}\,\mathbf{q}_{t-1}\} = \begin{bmatrix} \frac{\sigma^2 T_s^4}{4} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{\sigma^2 T_s^4}{4} & 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma^2 T_s^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & \sigma^2 T_s^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sigma^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & \sigma^2 \end{bmatrix} \tag{4.2}$$

The inverse of these matrices will be used by the functions generateFInverse and generateQInverse in Algorithm 2 below.

Algorithm 2 shows the prediction procedure, which aligns the filter to the current fusion time. The filter is moved forward in time specified by the variable DT, and the measurement uncertainty is modeled by sigma. The class variable PInv is the inverse covariance matrix of the track, and PInv$\hat{X}$ is the inverse covariance matrix times the state estimates collected into one variable.

The prediction function is divided into three segments: (A) Generate basic matrices, (B) Setup intermediate matrices, and (C) Perform prediction step.

Algorithm 2: Prediction function
Input: amount of time to move the filters, the sensor noise term
Output: aligned filters

The procedure of moving one filter to the time of the current fusion. Filters are moved by DT time, and the measurement noise will be modelled by sigma.

Class variables: PInv, PInv$\hat{X}$

1 **Function** *prediction (DT, sigma)*
  `/* Segment (A)` `*/`
2 FInv $\leftarrow$ generateFInverse(DT)
3 QInv $\leftarrow$ generateQInverse(DT, sigma)
  `/* Segment (B)` `*/`
4 M $\leftarrow$ FInv^T $\times$ PInv $\times$ FInv
5 C $\leftarrow$ M $\times$ (M + QInv)$^{-1}$
  `/* Segment (C)` `*/`
6 PInv $\leftarrow$ M $-$ C $\times$ M
7 PInv$\hat{X}$ $\leftarrow$ (I $-$ C) $\times$ FInv^T$\times$ PInv$\hat{X}$
8 **end**

To the issue of data packets arriving in a non-chronological order, the choices were to rollback the filters, Alexander's method [4] and EtM, no method was clearly more beneficial than the others. But a rollback of the calculations and then recalculating the filters is chosen as a primary method since both accuracy and consistency are preserved.

In Figure 4.2, the rollback procedure is shown. If the data packet is in chronological order, the process will continue as normal to the Data association. Otherwise, a rollback is performed to the point where it is in chronological order, and then the subsequent steps are re-fused again. This is an expensive procedure, but good in terms of accuracy.

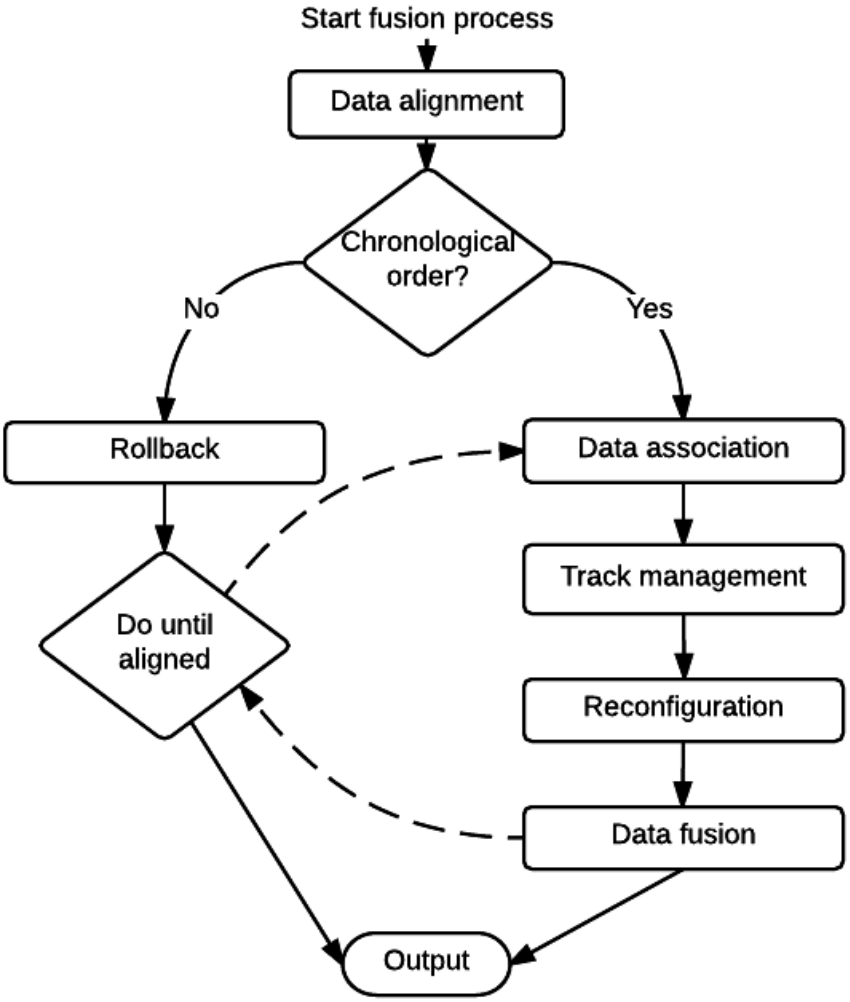

Figure 4.2: An illustration of the procedure of rollback.

4.2.2 Data Association

A global nearest-neighbor algorithm was chosen for associating the measurements with the tracks. Algorithm 3 below shows the entirety of the association algorithm. It takes the measurements and all the tracks as input, and returns two lists: all associated tracks and measurements (associated_PQ), and the measurements that did not get an association (tentative_tracks).

The algorithm begins in with gating the measurements to the tracks to exclude improbable pairs (line 2-9). The pass_gate function is simply a function that calculates how far the car might have traveled, and accounts for a small uncertainty.

The lines 10-21 describes how the measurements are chosen, with the best pairs first while considering that only one measurement can only be paired with one track.

This consideration is done on the lines 13-18, and removes the entries containing already matched measurements and tracks.

The line 22 saves all measurements that have not been associated with a track.

Algorithm 3: Data association
Input: all current tracks, all measurements
Output: list with associated measurements, list with tentative tracks

The association module seeks to group the measurements and tracks in a probabilistic manner. If a measurement is unable to be paired, it is put in tentative_tracks.

```
1  Function data_association (tracks, measurements)
       /* Gating phase                                              */
2      for measurements i do
3          for tracks j do
4              d ← Euclidean distance between measurement i and track j
5              if pass_gate(track i, measurement j, d) then
6                  priority_queue ← {d, j, i}
7              end
8          end
9      end
       /* Association phase                                         */
10     while !priority_queue.empty() do
11         [d, T1, M1] ← priority_queue.dequeue()
12         associated_PQ ← { d, T1, M1 }
13         Initialize new priority queue tmp_PQ
14         while !priority_queue.empty() do
15             [d, T2, M2] ← priority_queue.dequeue()
16             if T1 != T2 and M1 != M2 then
17                 tmp_PQ ← { d, T2, M2 }
18             end
19         end
20         priority_queue ← tmp_PQ
21     end
22     tentative_tracks ← all measurements with no association
23 end
```

4.2.3 Track Management

The track management module has primarily three tasks: to initialize new tracks, to upgrade tentative tracks to validated tracks, and to delete tracks. The entirety of the algorithm can be found below in Algorithm 4.

Tentative and validated tracks are needed due to uncertainty when tracking new objects, both in motion and existence. A tentative track can be upgraded if it is

consistently detected, which means that it needs to be detected several times within a specific amount of time, and is done between line 15 and 19.

In the deletion step, between line 5 and 14, both tentative and validated tracks are deleted if they are not detected within a certain period of time.

As mentioned in Section 3.4, the unmatched measurements from the data association process will be initialized as a tentative track. This is done lastly, on the lines 20 to 22. Note that between the lines 19 and 20, the filter will perform a filter update since there needs to be a historic measurement to be updated from. Thus,

Algorithm 4: Track management
Input: associated tracks, measurements with no association
Output: update of last detected, upgrade tracks, create new tentative tracks

This module manages all the tracks in the server. The lists from the Data Association is processed here. It upgrades a tentative track to a validated track if it has been detected multiple times, and also initializes new tracks.

```
1  Function track_management ( associated_tracks, new_tracks )
2      for associated_tracks do
3          Update when track was last detected
4      end
       /* Deletion                                                    */
5      for All validated tracks do
6          if Last update ≥ large time constant then
7              Delete the validated track
8          end
9      end
10     for All tentative tracks do
11         if Last update ≥ small time constant then
12             Delete the tentative track
13         end
14     end
       /* Validation                                                  */
15     for All tentative tracks do
16         if Detected multiple times within a short period of time then
17             Upgrade tentative track to a validated track
18         end
19     end
       /* === Reconfiguration ===                                     */
       /* === Filter update ===                                       */
       /* Initialization                                              */
20     for new_tracks i do
21         Initialize new tentative track for measurement i
22     end
23 end
```

the first measurement is taken as the best estimate of the track, and hence, new tracks tend to display volatile behavior.

4.2.4 Reconfiguration

Just before the data fusion process, the reconfiguration module may acquire information regarding the quality of the sensor setups of the ego-vehicle. A low-quality sensor with lower accuracy will be modelled with a larger noise $\mathbf{r}_t$ in Eq. (3.6). On the other hand, an accurate sensor will be modeled with less noise, since it is accurate and reliable.

There is no current support for determining the sensor setup in the simulation environment, thus real implementation of reconfiguration is not possible. However, the fusion architecture allows reconfiguration, although only one configuration is used.

4.2.5 Data Fusion

The remaining function to describe is the update steps of the Information filter. From the simulation environment, the actor has the possibility to not only measure another object's position, but also its velocity and acceleration. Therefore the measurement model will include this information in the filtering, and thus Eq. (3.6) becomes:

$$\mathbf{y}_t = \mathbf{H}_t\mathbf{X}_t + \mathbf{r}_t = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \mathbf{X}_t + \mathbf{r}_t \tag{4.3}$$

The covariance of the noise term $\mathbf{r}_t$ was tested with different values, and the following was the final matrix:

$$\mathbf{R}_t = Cov\{r_t\} = \begin{bmatrix} \sigma^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{\sigma^2}{10} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{\sigma^2}{10} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{\sigma^2}{10} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{\sigma^2}{10} \end{bmatrix} \tag{4.4}$$

with the parameter σ set to $\sigma = 3$. The values representing the velocity and acceleration ($\frac{\sigma^2}{10}$), implies better accuracy of the velocity and acceleration, compared to the GPS data. The noise of the data will be further elaborated in Section 5.3.

Lastly, we assume that there is no priori information, and hence the inverse covariance matrix $\mathbf{P}_0$ is initialized as a zero matrix. The update steps are then executed with the new measurements $\mathbf{y}_t$ together with the above matrices as shown in Eq. (3.21) and Eq. (3.24).

4.2.6 Fusion Server Overview

The fusion server is elaborate and consists of multiple items where timing sometimes is essential. Generally, the fusion is done for the time when the data packets are transmitted, and the fused outputs are always the current time. Figure 4.3 gathers all of the fusion server's items, and combines this with a timeline.

A fusion process is initialized when the data packet arrives at the server at T_3, and commences the Data Alignment. It begins by checking if the data packet arrived in a chronological order, and starts a rollback procedure if necessary. Otherwise, it aligns all tracks in the server from the last fusion T_1 to the new fusion time T_2. Then, the Data Association module tries to match the measurements in the data packet with existing tracks in the server and returns two lists: one with all the matches and one with measurements that did not get matched. These lists are used by the Track Management, and deletes tracks that have not been observed for a period of time, but also upgrades tentative tracks to a validated one. If the measurement arrived with some information of its sensor accurateness, the reconfiguration module adjusts the covariance of the noise matrix in Eq. (4.4) accordingly. Thereafter, the data fusion takes place on the matches made from the Data Association module. Before the output, new tracks are initialized with the measurements that did not get matched with a track, and the first value is taken as the truth of that track. Now, the fusion server is fully updated to time T_2, but due to delays, the current time of the server is later, at time T_3. Thus, a temporary prediction from time T_2 to time T_3 is made to get the most up-to-date output.

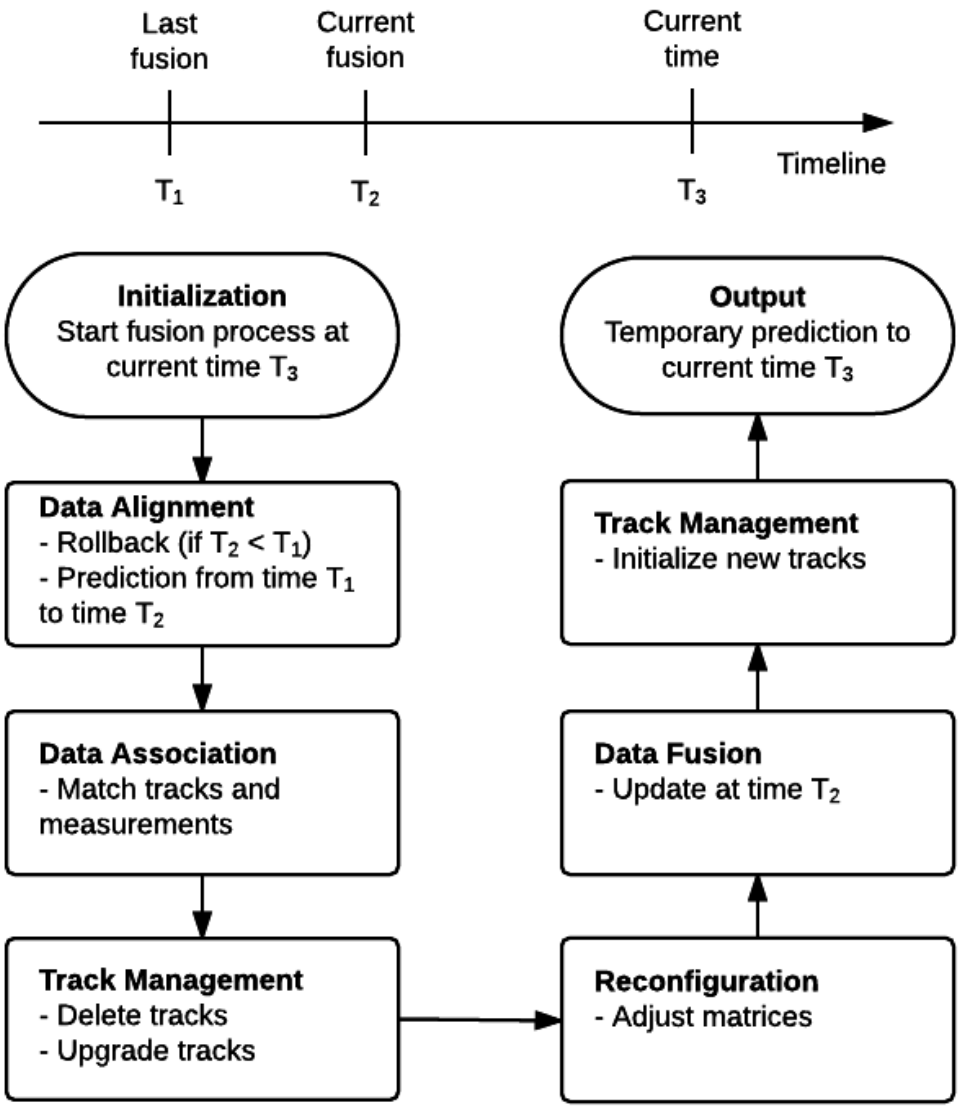

Figure 4.3: The fusion server together with all of its functionalities together with a timeline.

5
Data Sets

Here, we present the data sets that were used to test our fusion server. The data sets were provided by Volvo Car Corporation, and were generated by their internal simulation environment. The data consisted of ground truth data of the vehicles. If another vehicle was in a range of one vehicle's sensors, it got information of the other vehicle's state vector. This simulation environment had a millisecond precision, but the client simulator was run in real-time. Thus, it would transmit the measurement of the current time, and if the transmission frequency was set to 10 Hz, it would skip the 99 other measurements between the transmissions.

5.1 Scenario 1

The first traffic scenario is visualized in Figure 5.1 below. Car 1 starts driving from the coordinates $(10, -2)$ and Car 2 from $(50, -2)$. Both vehicles drive horizontally to the right, towards a still vehicle at coordinates $(400, -2)$. When approaching this still car, car 2 changes the lane whereas car 1 stops behind it. The total simulation time was 40 seconds and consisted of 40,000 frames.

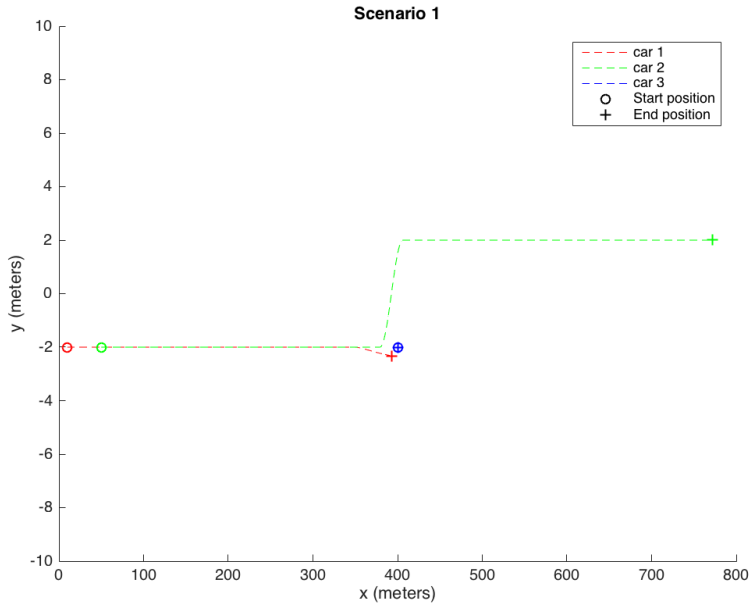

Figure 5.1: Two cars drive towards a still vehicle, one stops behind it, whereas one changes lane and continues driving.

5.2 Scenario 2

The second scenario in Figure 5.2, shows an interaction in a roundabout between three cars. Two cars approach from the left side, whereas another car comes in from the right side. All three cars ended up taking different exits. The total simulation time is 30 seconds, with a total of 30,000 frames.

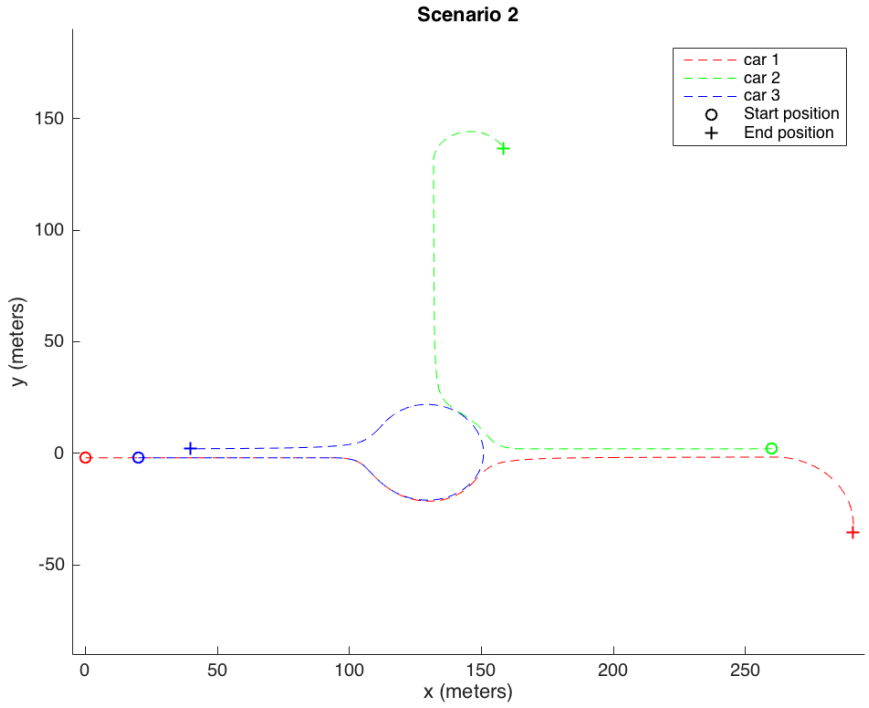

Figure 5.2: Three cars enter and exit a roundabout.

5.3 Noise

The data sets provided to us represented the ground truth. Thus, in order to get relevant results, the data needs to resemble real-world data. According to DoD [17], GPS receivers have been achieving horizontal accuracy of 3 meters or better 95% of the time. Additionally, the Velodyne LIDAR sensor HDL-64E achieves an accuracy of 2 cm and has a range of around 100 m [1]. The velocity and acceleration of the vehicles are generally very accurate with very small deviations.

Finally, we decided to add zero mean Gaussian noise to the true measurements with a standard deviation of 3 meters to the GPS data, and 1 meter to the sensor data. This setup would represent normal sensors, and not high-end sensors.

6
Results

Apart from displaying the benefit of fusing data globally, efforts have been placed upon revealing the problems that network delay presents, and investigating the speed and scalability of our fusion server.

When comparing the output with fusion (cars collaborate) and without fusion (no collaboration), we see significant improvements when the cars are sharing information and cooperating. Furthermore, our results suggest that the precision deteriorates with increasing mean latency. It was also found that data packets that arrived in a non-chronological order posed significant issues, both in terms of precision and computational speed.

The speed issues could mainly be traced down to two functions: one to align the data packets, and one to align the filters. The study ends with attempts to optimize the found bottlenecks, and investigating the scalability of the fusion server.

6.1 Setup

The data used were tainted by artificial noise, of which considerably deteriorated the performance of the fusion server. In order to generate different sequences of noises, seeds for the pseudo-random number generator varied in every run. Since the precision of the fusion server is closely linked with the client input, we mitigate the affect of the randomness to some degree through making 100 simulations of each test.

The performance of the fusion server is connected to the rate of which the vehicles broadcast, and in general we set the broadcasting-rate to 10 Hz. Another factor is the rollback size, which is a parameter that controls the number of allowable rollbacks for a data packet. The rollback size needs to be limited since the amount of historic data that needs to be stored could get infinite. Additionally, it makes little sense to rollback very far back in time, and if it exceeds the rollback size it is simply ignored. Nonetheless, a larger rollback size would in theory lead to better accuracy at the cost of more computational burden. Thus, these parameters need to be balanced, and in most experiments the rollback size is set to 5.

6.2 Precision

The tracking algorithm produces estimates, or hypotheses, of how the objects move in the system. This output can be analyzed in several ways, and one aspect is to consider the algorithm's precision to reproduce the ground truth. Bernardin and Stiefelhagen [9] introduce a metric to quantify the precision of the tracking algorithm: the Multiple Object Tracking Precision (MOTP). The MOTP estimate is retrieved by calculating the average positional error between the estimates $(\hat{x}_t, \hat{y}_t)$ and the ground truth (x_t, y_t). The Euclidean distance for track j and frame t is:

$$d_t^j = \sqrt{(\hat{x}_t^j - x_t^j)^2 + (\hat{y}_t^j - y_t^j)^2} \tag{6.1}$$

This is then averaged over all cars, and for all frames to get the MOTP:

$$MOTP = \frac{\sum_t \sum_j d_t^j}{\sum_t c_t} \tag{6.2}$$

Where c_t describes the number of tracks at frame t.

We start with the most basic simulation where the transmission delay is neglected (0 ms) in order to test the accuracy of the fusion server. To understand the fused output, recall from Figure 5.1, and especially the path of Car 2. In order to highlight the inaccuracies, a small piece of the path of Car 2 is displayed in Figure 6.1.

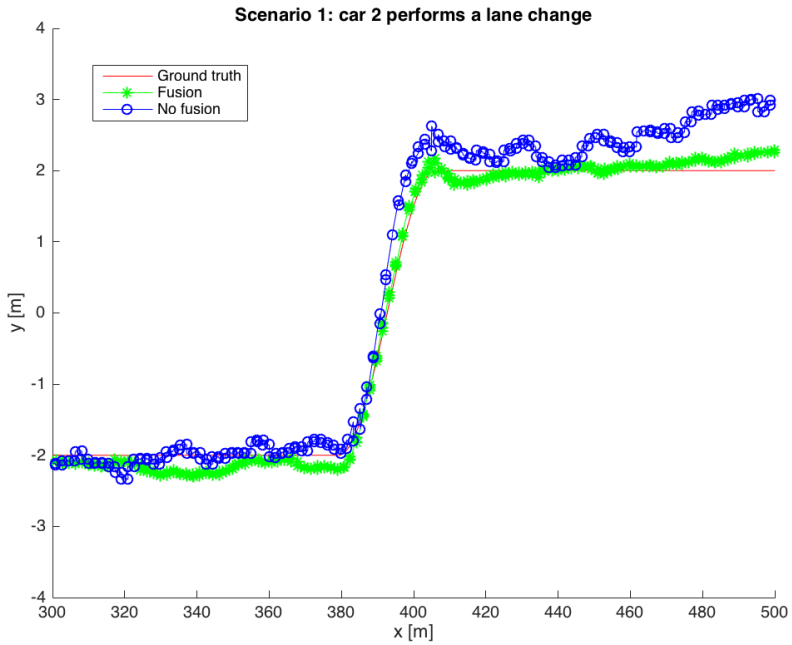

Figure 6.1: The path of Car 2 from Fig 5.1. The ground truth, path with fusion, and path without fusion. [Simulations: 1, Sending rate: 10 Hz, Rollback size: 5]

In a similar way, Car 3 is extracted from Scenario 2 in Figure 5.2. It drives around the whole roundabout, but only a small excerpt is shown in Figure 6.2, where it merely enters the roundabout.

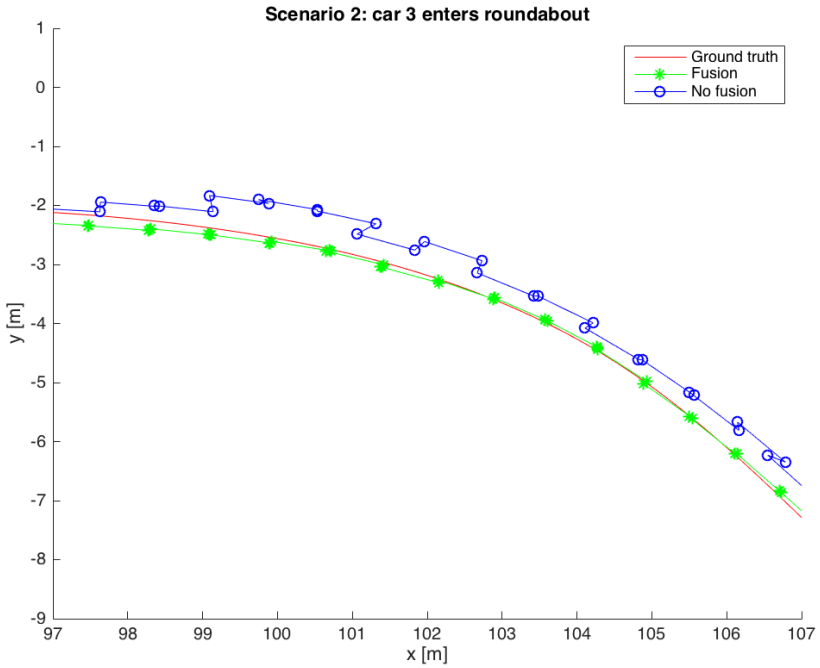

Figure 6.2: The path of Car 3 from Fig 5.2. The ground truth, path with fusion, and path without fusion. [Simulations: 1, Sending rate: 10 Hz, Rollback size: 5]

Then, we use the MOTP to quantify the accuracy of the fused output. In Figure 6.3 we see the MOTP of both scenarios, with and without fusion. The maximal and minimal MOTP found, together with the standard error of the mean.

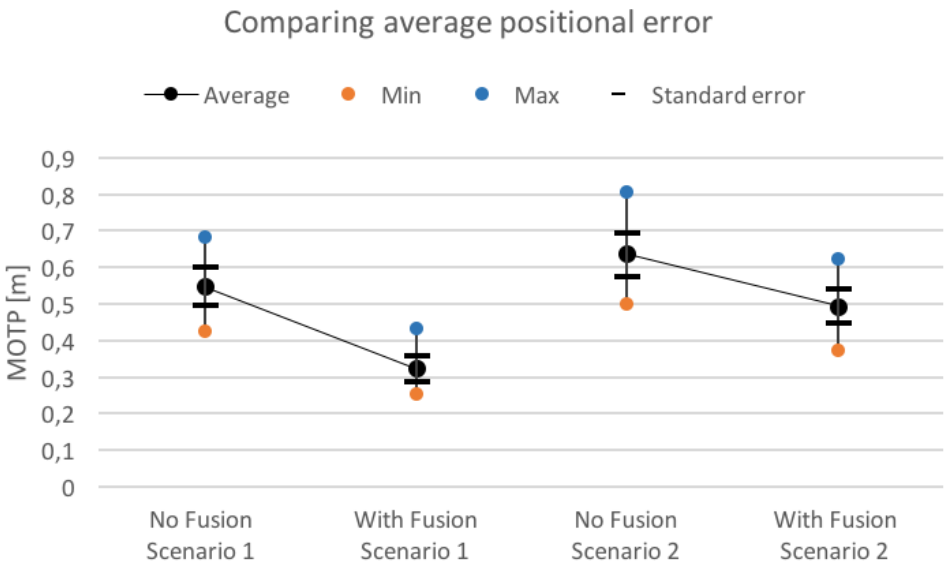

Figure 6.3: The average positional error when the cars do not collaborate (no fusion), and when they do (with fusion). [Simulations: 100, Sending rate: 10 Hz, Rollback size: 5]

6.3 Transmission Delay Testing

The transmission delay testing is done with regards to two aspects: the length of the delay and its spread. We start without spread and test the performance for different values of the delay length, and can be found in Figure 6.4. In this case, all measurements will arrive in a chronological order, and will never cause any rollbacks.

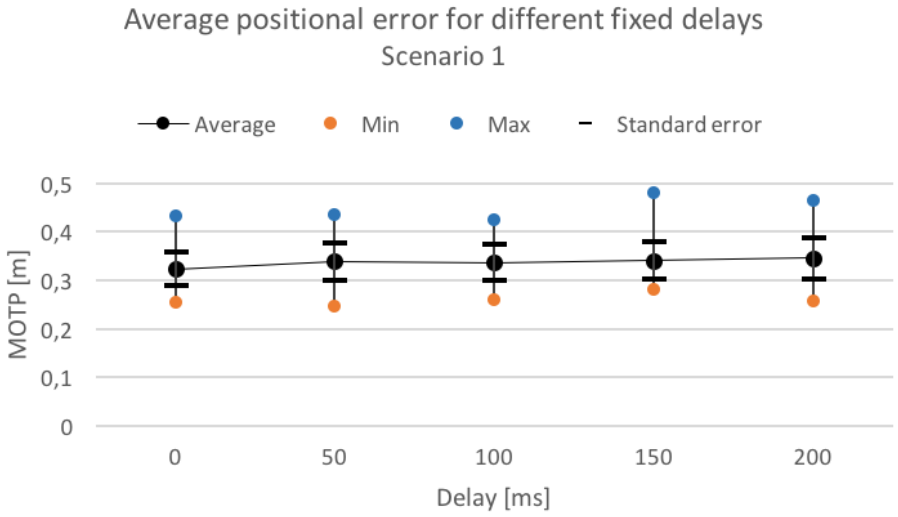

Figure 6.4: The average positional error and its statistical bounds for different fixed delays. [Simulations: 100, Sending rate: 10 Hz, Rollback size: 5]

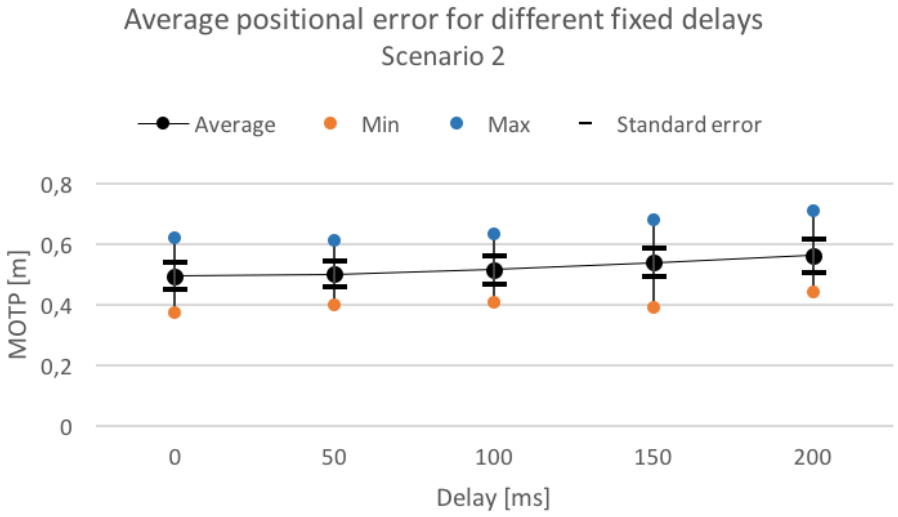

Figure 6.5: The average positional error and its statistical bounds for different fixed delays. [Simulations: 100, Sending rate: 10 Hz, Rollback size: 5]

Then, we tested the accuracy for different sizes of the spread of the transmission

delay. In Figure 6.6 and 6.7 all simulations had a mean transmission delay of 100 ms, and were tested with an increasing uniform transmission delay span.

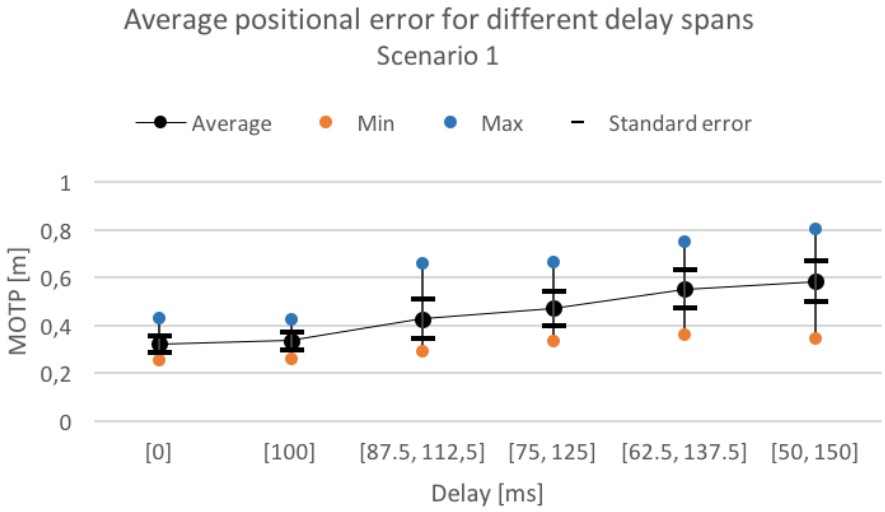

Figure 6.6: The average positional error and its statistical bounds for a given transmission delay span. [Simulations: 100, Sending rate: 10 Hz, Rollback size: 5]

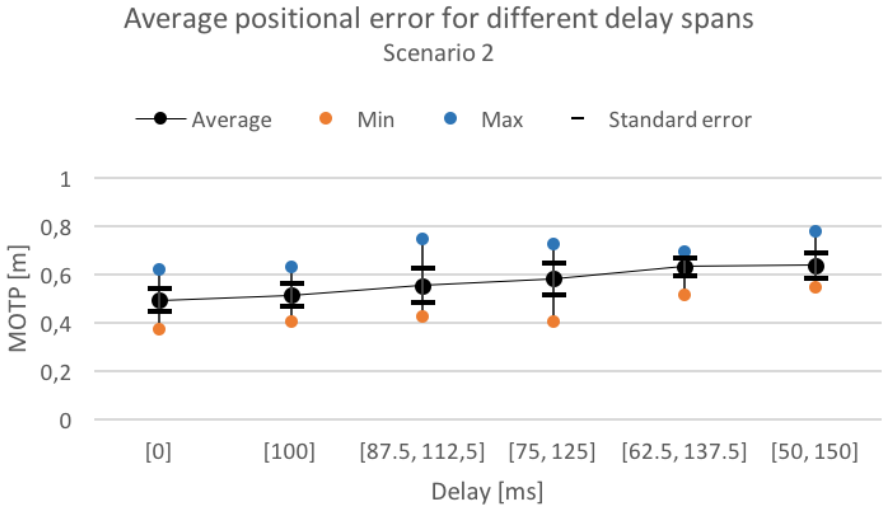

Figure 6.7: The average positional error and its statistical bounds for a given transmission delay span. [Simulations: 100, Sending rate: 10 Hz, Rollback size: 5]

6.4 Bottleneck Localization

This section seeks to evaluate the speed of the fusion server, how fast the internal modules are, and locate the bottlenecks. The case with a fixed delay of 100 ms is shown to the left in Figure 6.8. In this case, no rollbacks will take place, but

the fusion server will however account for the possibility of a rollback. Thus, the task of the rollback in this case is to store historic data, and will never perform any rollbacks. However, in the right figure rollbacks are possible, since the delay varies between [50, 150] ms.

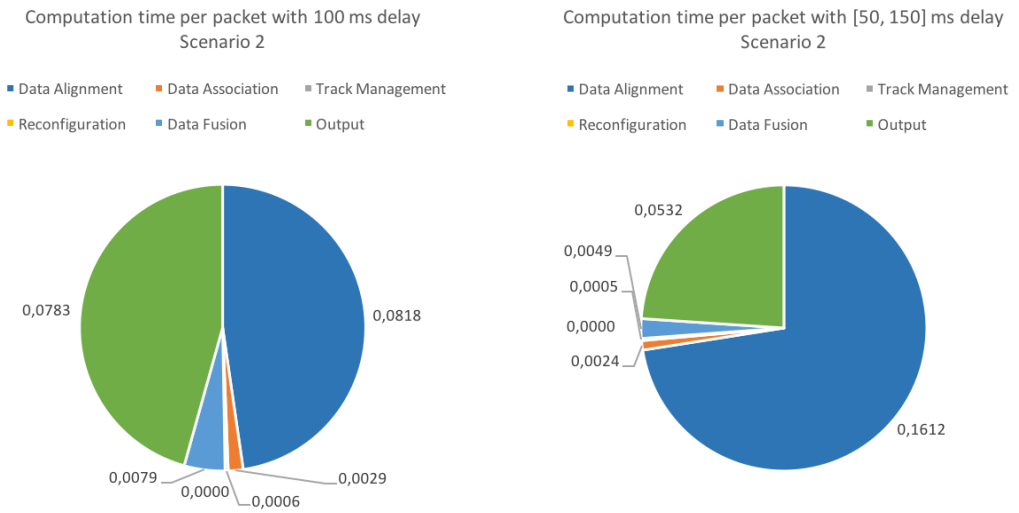

Figure 6.8: The average time spent in every module per packet with the delay 100 ms (left chart) and [50, 100] ms (right chart). [Simulations: 100, Sending rate: 10 Hz, Rollback size: 5]

Since the Data Alignment is a significant part of the total computation time, we decompose this module further. The Data Alignment have two tasks: to predict, and to perform rollbacks if necessary. Figure 6.9 divides the items into Prediction, Rollback, Output and Other, which is the remaining modules.

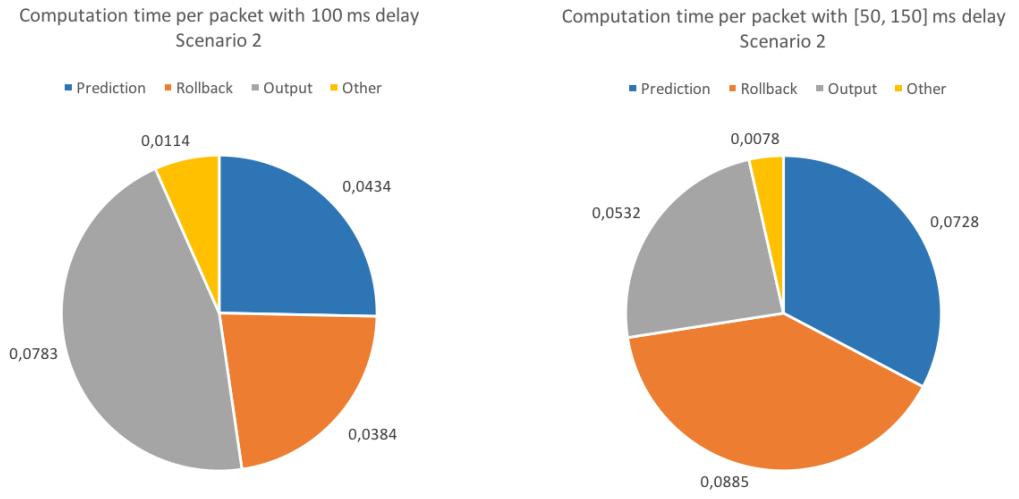

Figure 6.9: The left figure shows the average computation time when no rollbacks are present, whereas rollbacks are frequent in the right one. [Simulations: 100, Sending rate: 10 Hz, Rollback size: 5]

6.4.1 Extrapolating the Measurement

An alternative method to rolling back the filter and re-fusing it again was introduced in Section 3.2, which was called Extrapolating the Measurements (EtM). This method simply extrapolated old measurements to the same time as the most recent fusion, which would adjust the measurements into an artificial chronological order. The precision for different delay spans for Scenario 1 and 2, can be seen in Figure 6.10 and Figure 6.11 respectively. These graphs could be compared to the Figures 6.6 and 6.7.

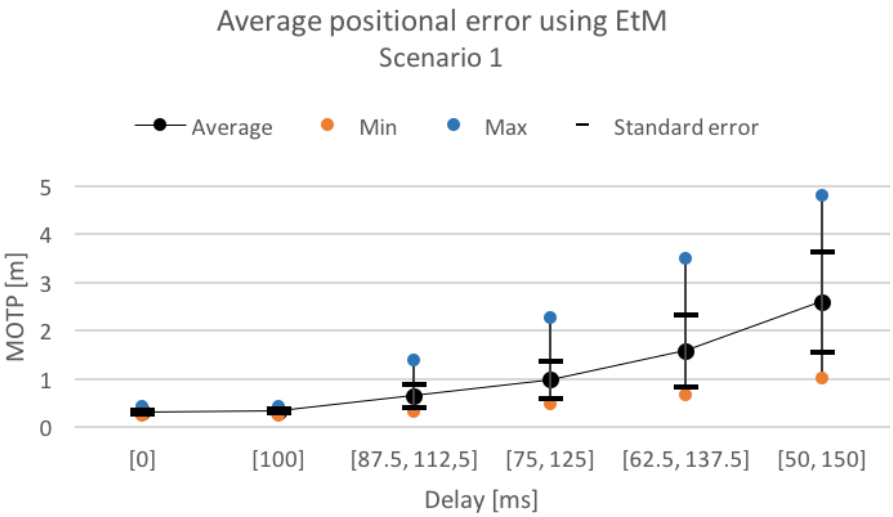

Figure 6.10: The MOTP for different spans of the delay. [Simulations: 100, Sending rate: 10 Hz, Rollback size: 5]

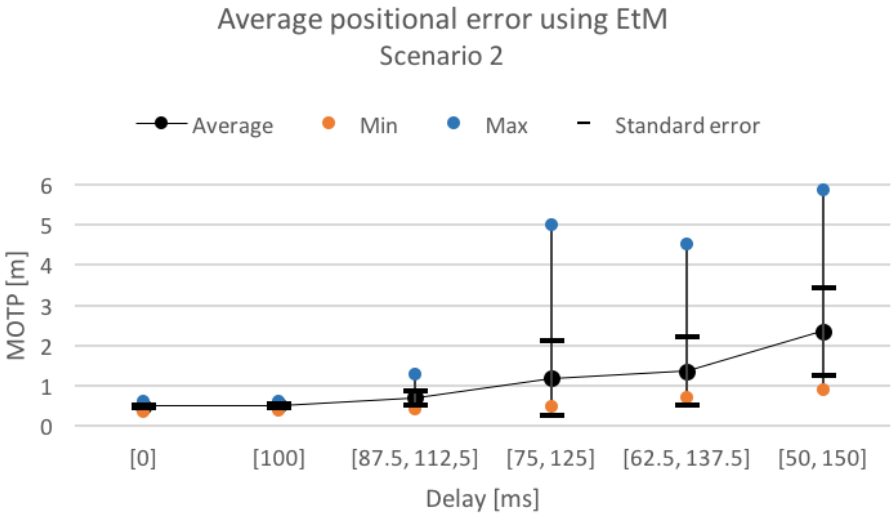

Figure 6.11: The MOTP for different spans of the delay. [Simulations: 100, Sending rate: 10 Hz, Rollback size: 5]

Precision aside, we also investigate the speed of using rollback and EtM. Table 6.1 shows the total computation time and the relative improvements for these methods.

Table 6.1: The average computation time for a data packet in the fusion server using rollback, compared to a fusion server using EtM. [Simulations: 100, Sending rate: 10 Hz, Rollback size: 5]

	Scenario 1		Scenario 2	
[87.5 - 112.5]	**EtM**	**Rollback**	**EtM**	**Rollback**
Average time [ms]	0.1872	0.253	0.1925	0.2547
Min time [ms]	0.1378	0.2161	0.1822	0.1797
Max time [ms]	0.2164	0.2808	0.2029	0.2799
Standard error [ms]	0.0287	0.0242	0.0064	0.0314
[50 - 150]				
Average time [ms]	0.2181	0.3413	0.222	0.3344
Min time [ms]	0.1751	0.3069	0.2052	0.3085
Max time [ms]	0.2652	0.3981	0.2327	0.3553
Standard error [ms]	0.0275	0.0318	0.0092	0.0144

Which shows that EtM always was faster than rollback, and reduced the computation time of the Data Alignment. To understand how much this improved the Data Alignment, Figure 6.12 displays the average computation time in each module for one data packet.

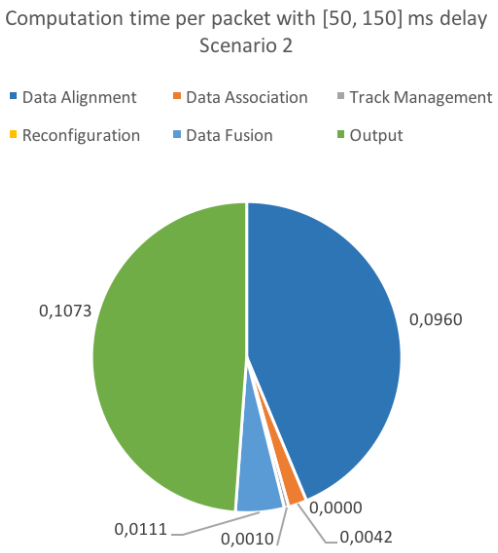

Figure 6.12: The average time spent in every module using EtM, with uniform delay span [50, 150] ms. [Simulations: 100, Sending rate: 10 Hz, Rollback size: 5]

However, as EtM resolves some speed issues, some significant ones still exists. The Data Alignment and output step have some similarities, and will be further scrutinized in the following section.

6.4.2 Prediction Analysis

Both the Data Alignment and output step performs a procedure called prediction, and moves all the filters forward to a specific time. In the case of data alignment, this time is the time of the current fusion, whereas the output step moves the filters to the current time. The prediction function is explained in detail by Algorithm 2 in Section 4.2.1, and consists of the following steps:

(A) Generate basic matrices
(B) Setup intermediate matrices
(C) Perform prediction step

In Figure 6.13 the items (A) - (C) are timed, but only the prediction function in the Data Alignment, and not the prediction function in the output step.

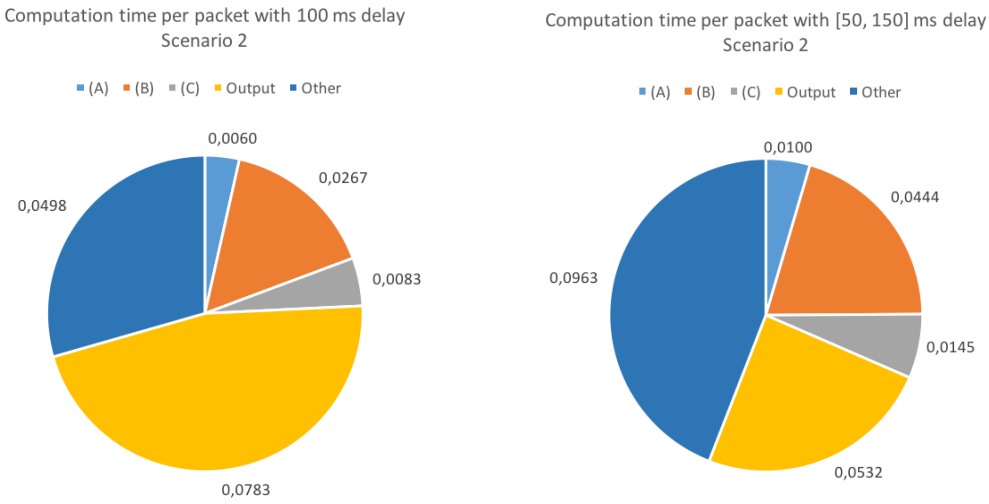

Figure 6.13: The average computation time per packet. The prediction function in Data Alignment is separated into (A), (B) and (C). Note that rollback is put under the item "Other". [Simulations: 100, Sending rate: 10 Hz, Rollback size: 5]

6.5 Performance

As explained in Section 2.4, since the Information filter use many but small matrices, multithreading is a suitable optimization technique. However, only the prediction function in the Data Alignment and in the output step can be done in parallel, whereas data association and track management need to be executed sequentially. However, the Data Fusion module is not a clear candidate for parallelization, since a car needs to detect a lot of other objects to be beneficial. Hence, this limits the gain of parallelization and might actually slow down the tracking algorithm.

Thus, the algorithm would in principle work the same as before, but the prediction in Data Alignment and the output step are done in parallel. The parallelized

modules are highlighted in red in Figure 6.14. Note that the threads need to be synchronized before proceeding to the next step.

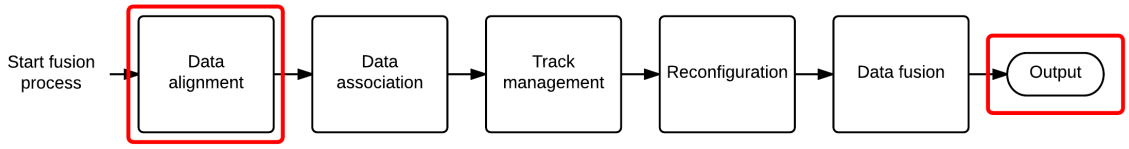

Figure 6.14: The fusion server and the parallelized modules highlighted in red color.

In order to guarantee that the results are compared with the same criteria, the pseudo-random number generator for delay and noise are fixed with the same seed for every observation. Additionally, the performance is tested on the same computer with the following specifications:

- CPU: Intel Core i7 (4 cores, 8 threads), 2.00 GHz
- Memory: 8 GB, 1333 MHz DDR3
- Operating system: OS X version 10.9.5
- Compiler: LLVM version 6.0 (clang)

Figure 6.15 shows the improvements from multithreading when the delay varies uniformly in the span [87.5, 112.5] ms, for different amount of cars. Similarily Figure 6.16 displays the improvements when having a uniform delay in the span [50, 150] ms.

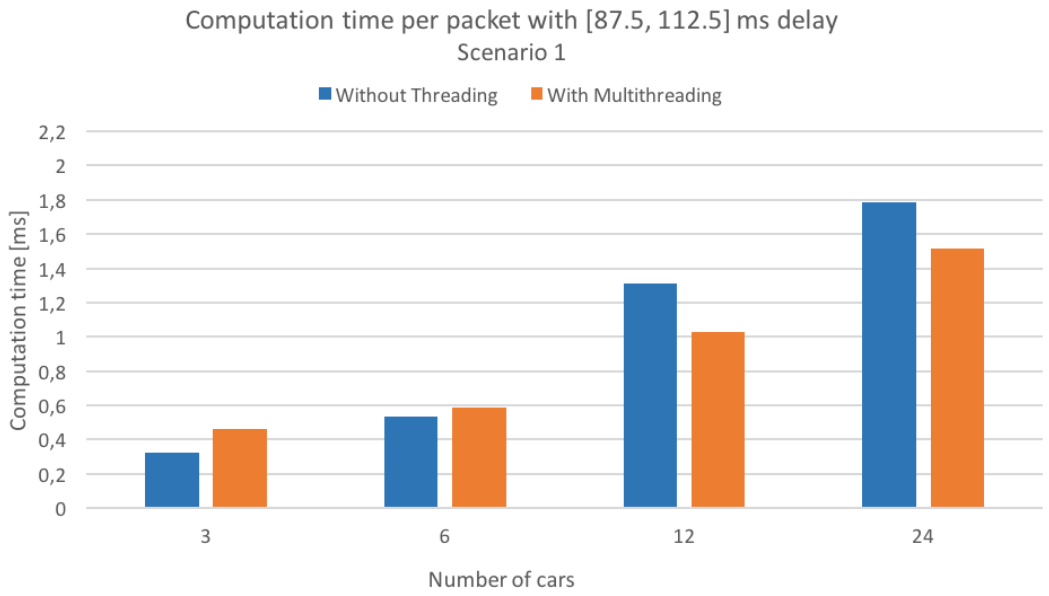

Figure 6.15: The average computation time per packet with delay span [87.5, 112.5] ms. [Simulations: 100, Sending rate: 10 Hz, Rollback size: 5]

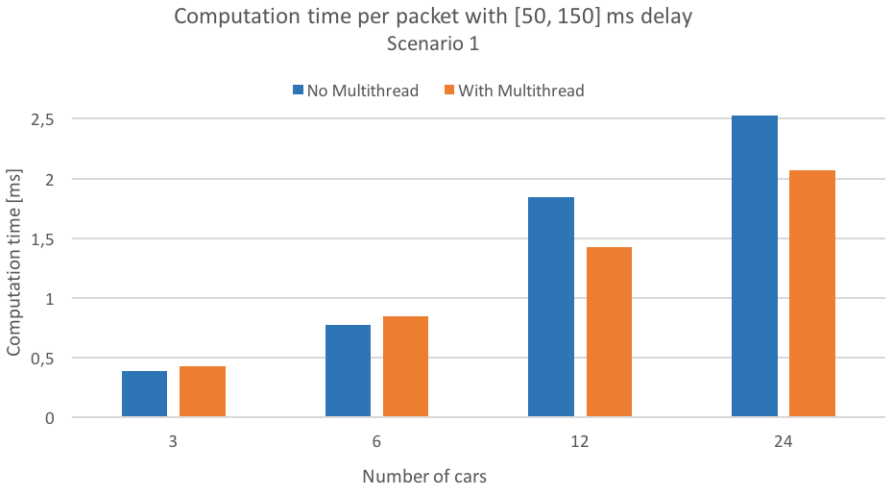

Figure 6.16: The average computation time per packet with delay span [50, 150] ms. [Simulations: 100, Sending rate: 10 Hz, Rollback size: 5]

The results with multithreading are slower with 6 or fewer cars. However, we see that the fusion server got improved by multithreading algorithm when we have more than 6 cars. According to this experiment, the multithreaded version performs more than 20 % faster with 12 cars, and almost 20 % faster for 24 cars for both spans.

Moreover, when the size of the delay span is increased from [87.5, 112.5] ms to [50, 150] ms, the computation time also increase relatively. For example, the computation time with 24 cars increased by approximately 30 %. This, due to the fact that larger delay spans increase the probability of data packets arriving non-chronologically. To further understand the improvements, we display the improvements for each module in Figure 6.17.

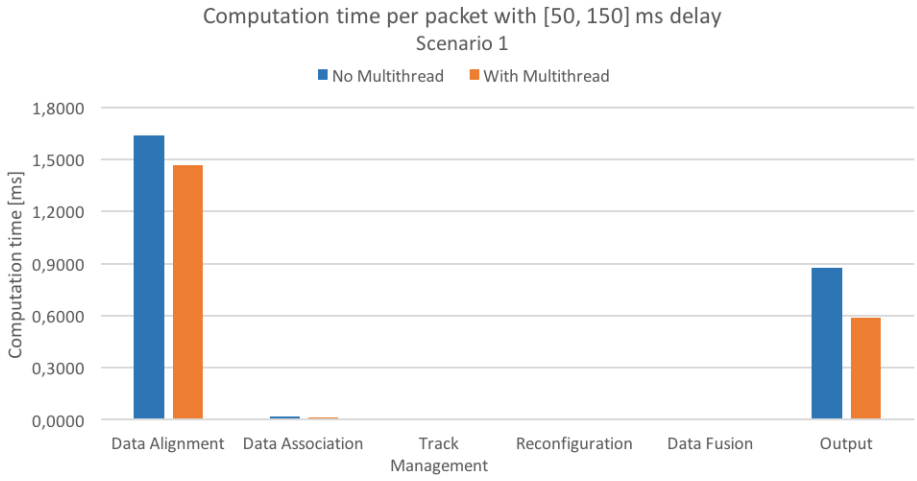

Figure 6.17: The average computation time with 24 cars for each module. [Simulations: 100, Sending rate: 10 Hz, Rollback size: 5]

The result shows that multithreaded algorithm for Data Alignment and the output step does improve the computation time noticeably. But they still constitute of the majority of the total computation time.

6.6 Scalability

Theory dictates that the broadcasting frequency should affect the scalability of the server, due to the different amount of data packets delivered to the system per vehicle. Sending with a high frequency is desirable, but simply not feasible in a large-scaled system. In Figure 6.18, we illustrate the precision when cars broadcast at different frequencies. The test is performed with 3 cars in the server, and when the transmission delay varies uniformly between 50 and 150 ms.

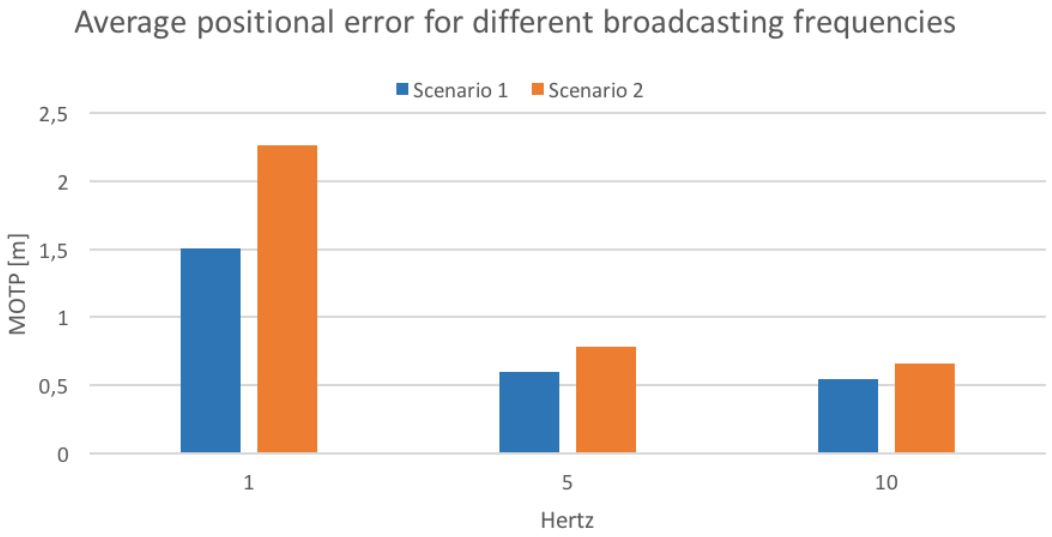

Figure 6.18: The MOTP for different sending rates with delay span [50, 150] ms. [Simulations: 100, Rollback size: 5]

When the cars send data at 1 Hz, the average positional error is very high. But the experiment suggests that the average positional error exponentially decays with increasing sending frequency.

6.6.1 Different Sending Rates

Although the server provides more accuracy with increasing number of inputs, it also comes with higher computational costs. Therefore, we conducted another experiment to show how the fusion server scale up when there are more than 3 cars in the server by duplicating the same set of data. In order to guarantee that the results are compared with the same criteria, the pseudo-random number generator for delay and noise are fixed with the same seed for every observation. The experiments are set up with a delay between 50 and 150 ms, and with increasing numbers of cars until real-time execution no longer is maintained. Since Scenario 1 spans over

30 seconds, the real-time execution criterion is broken when the total computation time exceeds the total simulation time. Figure 6.19 and 6.20 shows the scalability when the vehicles communicate with the server at 10 Hz, which is the frequency most previous experiments have been conducted with.

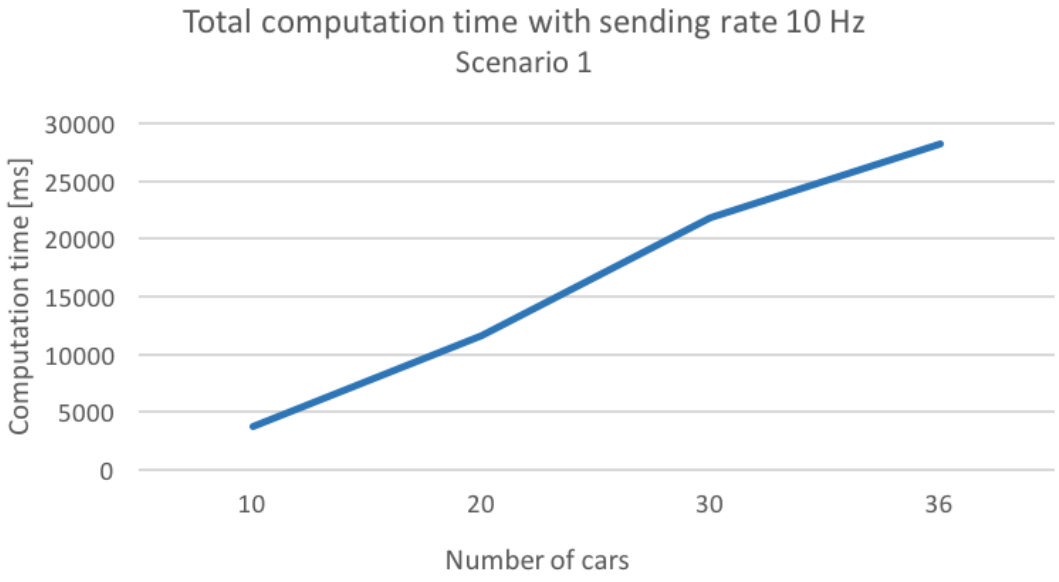

Figure 6.19: The total computation time with delay span [50, 150] ms. [Simulations: 100, Sending rate: 10 Hz, Rollback size: 5]

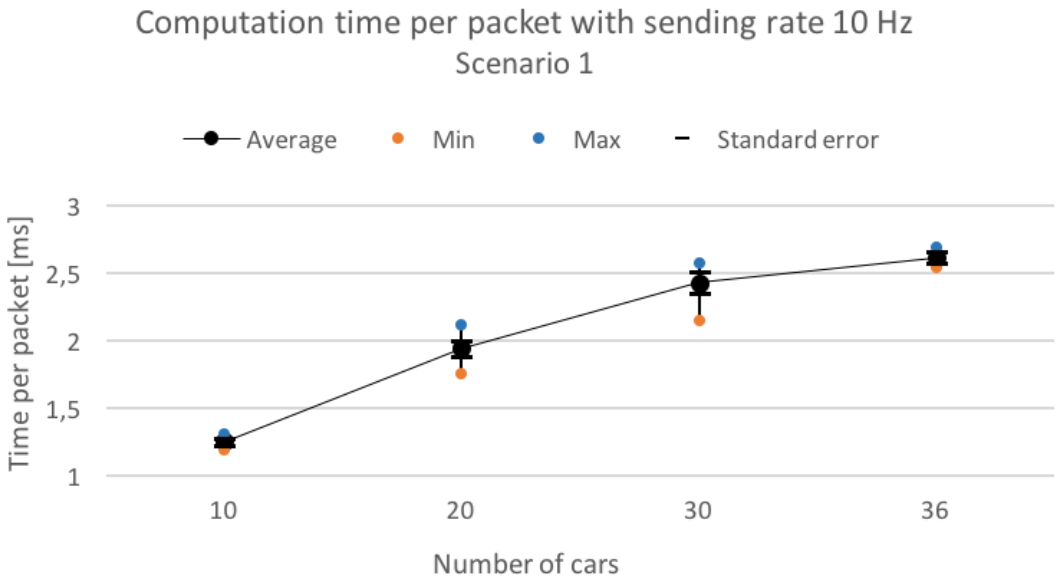

Figure 6.20: The computation time per packet with delay span [50, 150] ms. [Simulations: 100, Sending rate: 10 Hz, Rollback size: 5]

We find that the fusion server can support up to 36 cars with the specified settings, which can be considered as a small system. In order to increase the number of cars the server can support, the car's sending frequency is lowered. Figure 6.21 and 6.22 shows the scalability when the transmission frequency is set to 1 Hz.

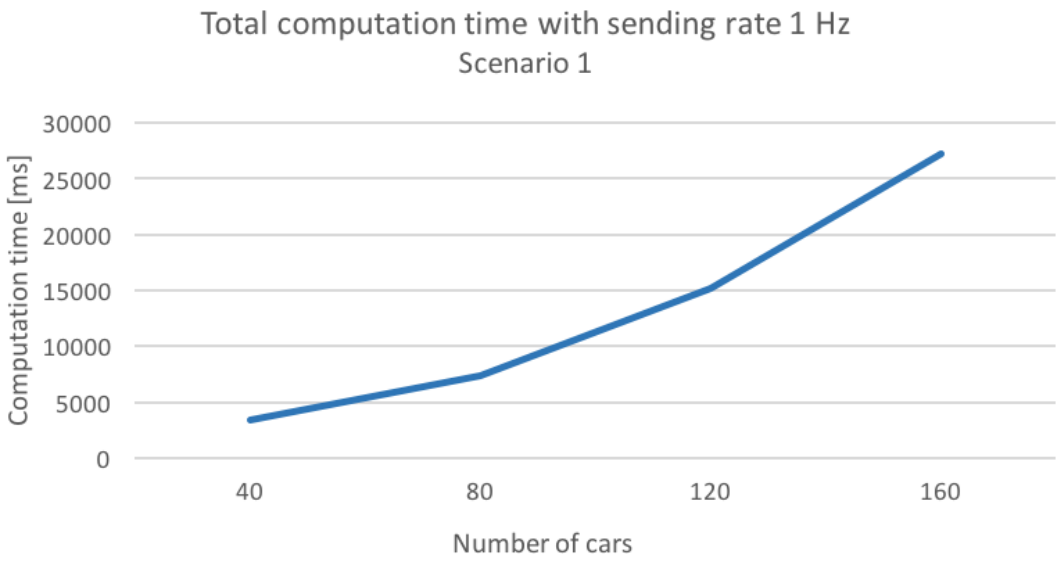

Figure 6.21: The total computation time with delay span [50, 150] ms. [Simulations: 100, Sending rate: 1 Hz, Rollback size: 5]

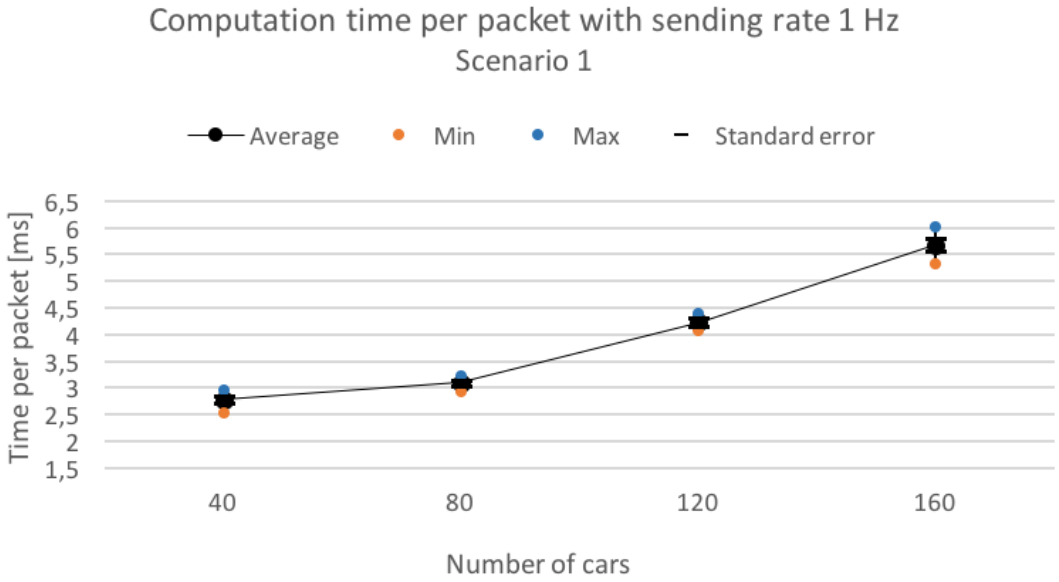

Figure 6.22: The computation time per packet delay span [50, 150] ms. [Simulations: 100, Sending rate: 1 Hz, Rollback size: 5]

6.6.2 Rollback Allowance

Another important factor for scalability is rollback size. Previously, the experiments were tested with the rollback size of 5. This section will demonstrate the result when rollback size is increased from 5 to 10, since allowing more rollbacks should affect the scalability of the fusion server significantly.

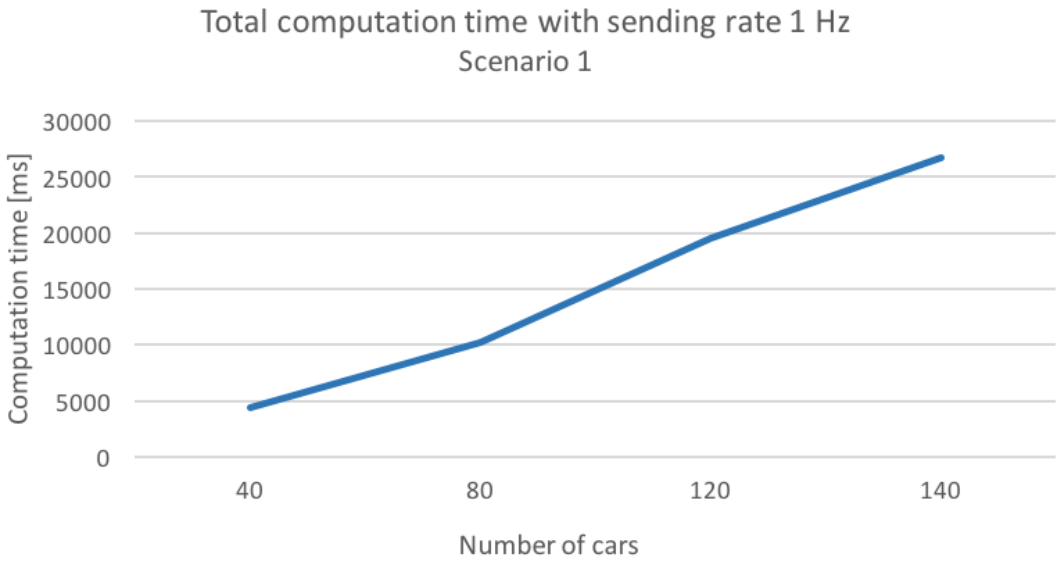

Figure 6.23: The computation time per packet with delay span [50, 150] ms. [Simulations: 100, Sending rate: 1 Hz, Rollback size: 10]

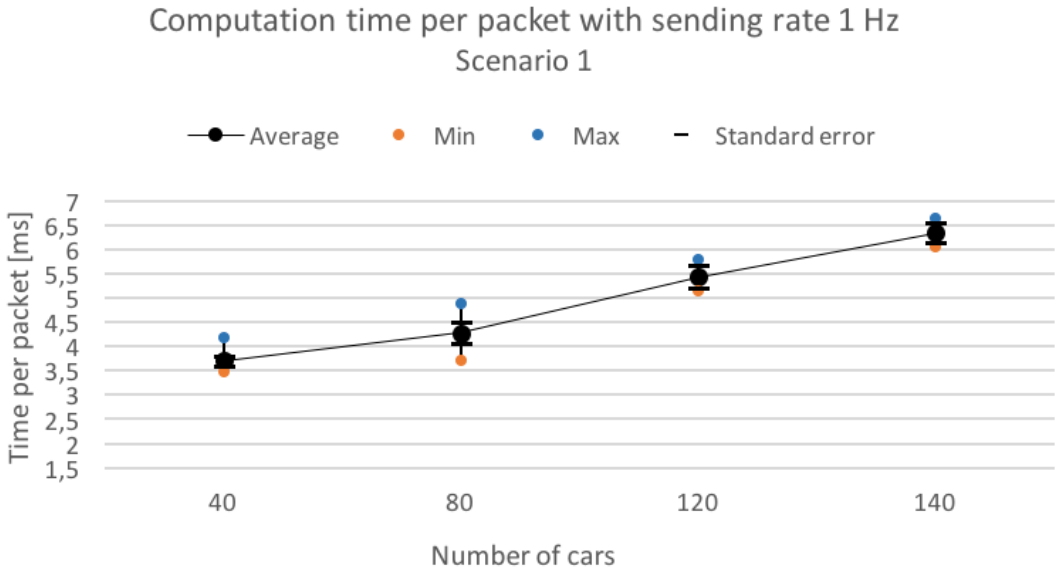

Figure 6.24: The computation time per packet with delay span [50, 150] ms on Scenario 1. [Simulations: 100, Sending rate: 1 Hz, Rollback size: 10]

As seen in Figure 6.22 and 6.24, the fusion server capacity decreases from 160 cars to 140 cars. Moreover, computation time per packet also increases noticeably, for instance, the average computation with 40 cars is increased from 2.78 ms to 3.70 ms, which is around 33 %.

7
Discussion

Due to the open-ended nature of the track fusion problem in general, and our approach in particular, estimating the reliability of the results and the degree of success has been a difficult endeavor. There exist a myriad of approaches to the track fusion problem, but the predicament is that very few overlap sufficiently to our study. Additionally, the absence of real-world testing was a weakness in our study. Our simplified prototype system may have overlooked critical aspects of a real-world scenario. These factors make it difficult to draw any definitive conclusions from our results. Hence, this pilot project should merely be seen as a guideline to the dynamics of an on-line fusion server.

7.1 Accuracy

The results from comparing cars using local fusion and global fusion are demonstrated in Figure 6.3. We find a significant difference between the output without fusion and with fusion and shows that a lot of accuracy is attained when the cars are collaborating. Apart from the improved accuracy, the maximum and minimum MOTP was more concentrated around the mean. The decrease in standard error signified less volatility in the results, and a more stable tracking algorithm.

Comparing the results to different work are however more troublesome, and does not do any work justice. But to get a grasp of the magnitude of accuracy, we provide the results made by other tracking algorithms. Numerous studies have used the KITTI Vision Benchmark Suite [22], which uses a Velodyne LIDAR HDL-64E [1] with a precision of 2 cm and a OXTS RT 3003 GPS [2] with an accuracy of 1 cm under perfect conditions. Spinello et al. [32] got a MOTP of 0.16 m, whereas Yves [38] achieved an average positional error of 0.14 m. Our noise replicated a fairly cheap sensor setup and obtained a MOTP of approximately 0.32 m for Scenario 1, and around 0.49 m for Scenario 2.

7.2 Transmission Delay

For different lengths of the latency, we see from Figure 6.4 and 6.5 a slight correlation between the length of the delay and the MOTP. From the best case (0 ms) and the worst case (200 ms), the MOTP increases around 0.02 m and 0.065 m for Scenario 1 and 2 respectively. The degradation from longer transmission delay was expected, since it is impossible for the tracking algorithm to foresee turns and velocity changes.

The fusion algorithm thrives when the cars drive in a constant and predictable manner, but is however noticeably affected by the latency. The increased MOTP for Scenario 2 compared to Scenario 1 is the consequence of having more turns and velocity changes. This is further demonstrated by Figure 7.1, which shows the MOTP frame by frame through the simulation of Scenario 2. The MOTP increases considerably when the cars begin to enter the roundabout and the simulation with 200 ms delay have more difficulty predicting the position of the cars. To get a message 200 ms late will naturally cause bad precision of the tracking algorithm, especially when travelling at high speeds.

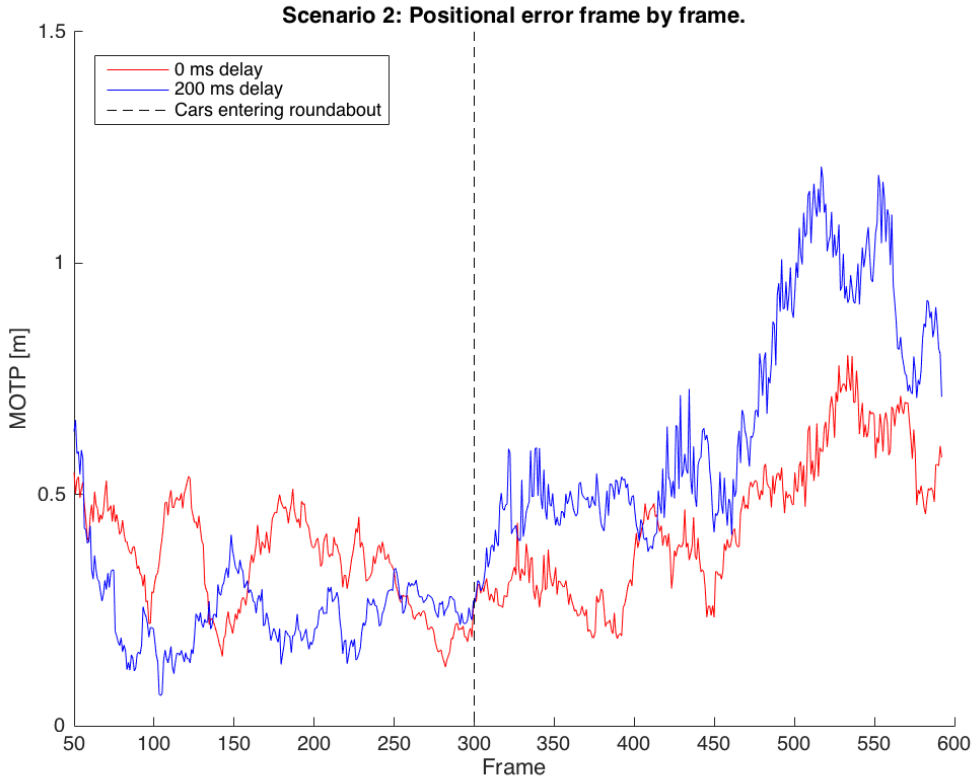

Figure 7.1: The average positional error frame by frame for Scenario 2. Note that the total simulation spans between the frames 0 and 900. [Simulations: 1, Sending rate: 10, Rollback size: 5]

The affect of the transmission delay spread is seen in Figure 6.6 and 6.7, and despite using an optimal strategy to put the measurements in a chronological order, we find that the delay span distinctively affects the precision. When comparing the span of the delay to the length of the delay, we interestingly find that the variance of the delay causes more troubles than the mean delay.

7.3 Performance

It is evident that the bottleneck in our fusion server lies within Data Alignment and the output step. One bottleneck could be localized to the rollback function, and in order to circumvent this costly feature, EtM was implemented. But as seen in Figure 6.10 and Figure 6.11, the precision exponentially deteriorates when the delay span increases. However, this method's upside can be observed in Table 6.1, where the fusion server that uses EtM is considerably faster. Despite the improvements from this new method, we see from Figure 6.12 that the Data Alignment module and output step still had the highest computational burden. Both contained the prediction function, and was further analyzed in Section 6.4.2. The largest bottlenecks were localized to algebraic operations, and specifically matrix inversions and matrix multiplications.

Thus, we instead optimized parts of the fusion server using multithreading. The results demonstrated that with multithreading, the fusion server gets approximately 20 % faster compared to the non-optimized version. However, we also found that it becomes slower when there were around six cars or fewer in the server, due to the fact that there is additional time for spawning and joining new threads. In addition to that, the fusion server is unable to benefit from parallel computing in small systems, since the size of the parallelizable blocks is small. Furthermore, parallelization covers only a small part of the fusion server, while other parts still run sequentially. "Amdahl's law" is often used in parallel explanation, "the theoretical speedup is always limited by the part of the task that cannot benefit from the improvement" [5].

Additionally, the result in Figure 6.17 shows that the average computation time for Data Alignment and the output step are suspiciously high. While other modules take a small portion of the computation time. The limited gain from multithreading could be explained by the limited amount of threads in the CPU, combined with a high overhead time.

7.4 Scalability

Since all cars transmit its data to the fusion server, the number of cars significantly affects the fusion server. Our tests suggest that the average computation time per data packet grows linearly with every car, and the total computation time tend to be exponential with every car. But this result should not be fully trusted. Due to the limited data sets, we had to artificially spawn the same scenario multiple times. These were isolated simulations, which meant that the cars were unable to detect each other in-between the simulations. If they were able to detect each other, we would have gained improved accuracy, but at the cost of more computational power.

From Figure 6.20, we see that the fusion server can support up to 36 cars sending data at 10 Hz. When sending at 1 Hz, the fusion server can support 160 cars when the rollback size is 5, and 140 cars when the rollback size is 10. The difficulty

lay in determining the best ratio between all parameters that affect the precision and speed of the fusion server. For example, we see from Figure 6.18 that the precision of the fusion server degrades dramatically when cars transmit at 1 Hz. Lower sending frequencies might be fine in a vehicle dense area, but might not be feasible otherwise. Furthermore, by comparing Figure 6.22 and Figure 6.24, we see that the average computation time per packet raised by almost 28% when the rollback size was increased from 5 to 10. A larger rollback size yields a better precision, but result in a slower performance. The trade-off between speed and performance has been the general theme when determining the scalability, and does not have an unambiguous answer.

7.5 Extensions

The fusion server consists of many modules and features that further can be tuned and extended. A lot of interesting features becomes possible if the client and fusion server are allowed to communicate to each other. The primary problems found in this thesis have been towards the network stability, and doing rollbacks. A way to lower the randomness by the network and the risk for rollback, is to make the vehicles broadcast with as far interval to the other vehicles as possible. The bigger the interval between two transmissions, the lower the probability that the data packets will arrive in a non-chronological order. Figure 7.2 shows a scenario when two vehicles send their data from an arbitrary starting position, which happens to be very close to each other. In this case, the transmission delay on Client 2 needs only to be at least 11 ms longer than Client 1 to trigger a rollback. Or, the transmission delay of Client 1 needs to exceed Client 2's by at least 91 ms to trigger a rollback. In the case of Broadcast spreading shown in Figure 7.2, the clients need to exceed the other by at least 51 ms to trigger a rollback. Since the transmission delay should in average be very close to each other, Broadcast spreading should conceivably lead to fewer rollbacks.

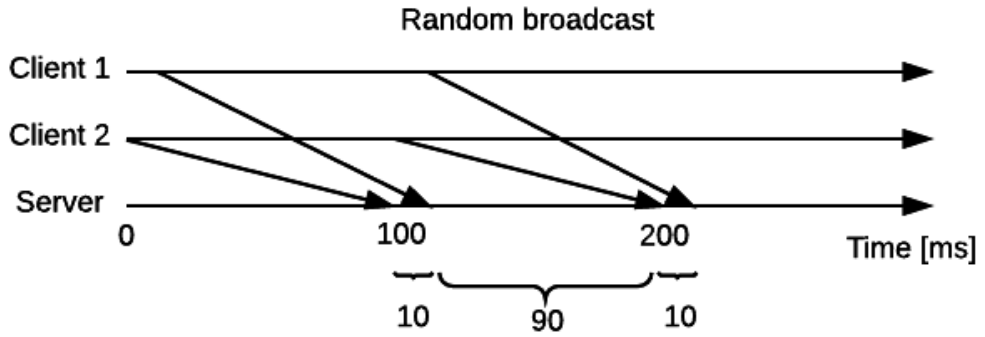

Figure 7.2: Random broadcasting time for two clients.

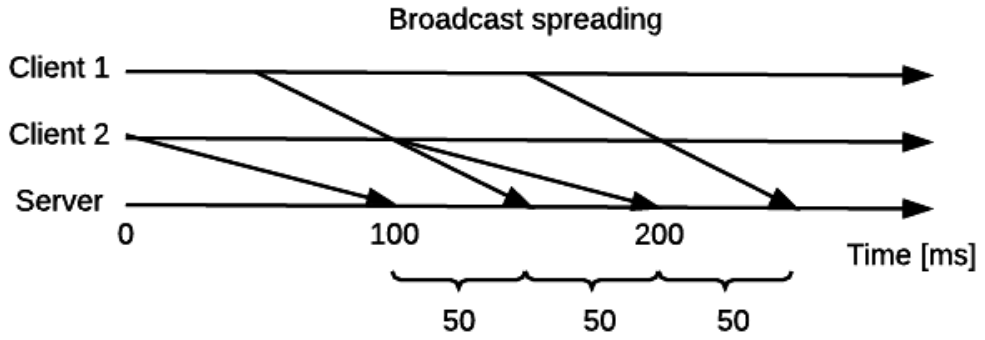

Figure 7.3: Two vehicles spreading their broadcasting times to be as far away of each other as possible.

On the same theme, if the server would be able to communicate with the cars, one possibility to increase the scalability of the server would be adjusted the cars broadcasting frequency depending on the number of cars in the area. Cars within a small area would detect each other more frequently, rendering it possible to lower the number of updates to the fusion server. The problem lays however in creating such an algorithm that is able to optimize the relation between the number of cars and accuracy.

Furthermore, the Kalman Filter and the Information Filter handles linear motion well but are both basic approaches. Therefore, other implementations such as the Extended Kalman Filter, or the Particle Filter should also be evaluated, since they handle non-linear motion better. In addition to only test one fusion algorithm at a time, an advanced system could possibly integrate multiple filters, depending on the traffic situation or the driver. For example, either driving in urban or countryside environment, and either having a human or autonomous driver. A human driver in an urban environment could pose a highly non-linear behaviour, whereas an autonomous driver in a countryside environment might be very predictable and linear in motion.

Apart from preciseness, performance is also an important issue for the fusion server. Parallel computing was considered as an alternative to enhance a computation capacity. On the other hand, distributed computing may be suitable for the fusion server. Ideally, when a single fusion server is not enough to support a large scale of computations, more fusion servers are built and connected together to a network. The tasks are communicated by message passing, in which workloads can be shared among connected servers. However, an additional time between server-to-server communication must be taken into account in order to maintain the quality of the fusion service.

8

Conclusion

We have created a client simulator to resemble the characteristics of independent cars in a real world scenario. A fusion server was created to fuse the incoming data into a holistic picture in real-time. We studied the affects when the sources and the consumer are at different locations. The physical separation causes the transportation time of the data to become significant, and deteriorate the performance.

The results from our fusion server have been exceedingly difficult to compare to other work. However, we saw significant improvements when using global data fusion instead of local data fusion. We also found that the length of the network delay affected the precision of the tracking algorithm. The delay span did not only significantly impact the precision, but also afflicted the speed of the tracking algorithm. In particular, the Data Alignment and the output step were by far the largest bottleneck. The rollback feature produced satisfying precision, but had a considerable computational burden, since its computation time is proportional to the other modules. Another bottleneck was found to be the prediction function in Data Alignment and the output step. Attempts were made to optimize this function through parallelization, but with limited success. The size of the matrices and the scale of the system were simply too small, and could not be fully benefited. Altogether, with a fusion server that still have not been fully optimized, our study indicate that an on-line fusion server with our approach might support hundreds of cars, rather than thousands. However, the scalability is linked to numerous parameters, such as the sending rate and rollback size.

To create a commercial infrastructure that fuses data from multiple vehicles into a common picture, one has to at the least address two important challenges. The first one is the speed of the tracking algorithm. A fast-tracking algorithm has more flexibility, and can choose to be fast, or to be more precise. The other challenge is the network stability. A lot of precision is lost by a network with high latency. It helps the tracking algorithm immensely to get a measurement that early indicates a change in the driving pattern. Stability in the network also relates to a tight delay span, that the packets oftentimes arrive in chronological order, which in turn lead to less computation time in the tracking algorithm. The excess computational power could instead intelligently be spent by the Intelligent Traffic System to adjust the relation between speed and precision. Thus, a stable network forges positive feedback, and solves a lot of issues by nature. By granting the ITS large margins, it increases its probability to solve unpredictable and complex traffic scenarios, creating a safer environment for all actors.

Bibliography

[1] Velodyne LIDAR HDL-64E. URL http://velodynelidar.com/hdl-64e.html. Accessed 2016-5-13.

[2] OXTS RT 3003. URL http://www.oxts.com/products/rt3000-family/. Accessed 2016-5-19.

[3] Michael Aeberhard, Stefan Schlichthärle, Nico Kaempchen, and Torsten Bertram. Track-to-track fusion with asynchronous sensors using information matrix fusion for surround environment perception. *Intelligent Transportation Systems, IEEE Transactions on*, 13(4):1717–1726, 2012.

[4] Harold L Alexander. State estimation for distributed systems with sensing delay. In *Orlando'91, Orlando, FL*, pages 103–111. International Society for Optics and Photonics, 1991.

[5] Gene M Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, pages 483–485. ACM, 1967.

[6] Brian D. O Anderson and John B Moore. *Optimal filtering*. Prentice-Hall, 1979.

[7] Yaakov Bar-Shalom and Lorenzo Campo. The effect of the common process noise on the two-sensor fused-track covariance. *Aerospace and Electronic Systems, IEEE Transactions on*, (6):803–805, 1986.

[8] Yaakov Bar-Shalom and Edison Tse. Tracking in a cluttered environment with probabilistic data association. *Automatica*, 11(5):451–460, 1975.

[9] Keni Bernardin and Rainer Stiefelhagen. Evaluating multiple object tracking performance: the clear mot metrics. *Journal on Image and Video Processing*, 2008:1, 2008.

[10] Subir Biswas, Raymond Tatchikou, and Francois Dion. Vehicle-to-vehicle wireless communication protocols for enhancing highway traffic safety. *Communications Magazine, IEEE*, 44(1):74–82, 2006.

[11] Samuel Blackman and Robert Popoli. Design and analysis of modern tracking systems. 1999. *Artech House, Norwood, MA*.

[12] Christophe Blanc, Laurent Trassoudaine, and Jean Gallice. Ekf and particle filter track-to-track fusion: a quantitative comparison from radar/lidar obstacle

tracks. In *Information Fusion, 2005 8th International Conference on*, volume 2, pages 7–pp. IEEE, 2005.

[13] Robert Grover Brown and Patrick Y.C Hwang. *Introduction to random signals and applied Kalman filtering*. J. Wiley, 1992.

[14] Federico Castanedo. A review of data fusion techniques. *The Scientific World Journal*, 2013, 2013.

[15] Chee-Yee Chong, Kuo-Chu Chang, and Shozo Mori. Distributed tracking in distributed sensor networks. In *American Control Conference, 1986*, pages 1863–1868. IEEE, 1986.

[16] Lars Danielsson. *Tracking and radar sensor modelling for automotive safety systems*. PhD Thesis. Department of Signals and Systems, Chalmers University of Technology, 2010.

[17] US DoD. Global positioning system standard positioning service performance standard. *Assistant secretary of defense for command, control, communications, and intelligence*, 2008.

[18] Louis Drolet, François Michaud, and Jean Côté. Adaptable sensor fusion using multiple kalman filters. In *IROS*, pages 1434–1439, 2000.

[19] Oliver E Drummond. Hybrid sensor fusion algorithm architecture and tracklets. In *Optical Science, Engineering and Instrumentation'97*, pages 485–502. International Society for Optics and Photonics, 1997.

[20] Thomas E Fortmann, Yaakov Bar-Shalom, and Molly Scheffe. Sonar tracking of multiple targets using joint probabilistic data association. *Oceanic Engineering, IEEE Journal of*, 8(3):173–184, 1983.

[21] Qiaoqiang Gan and Chris J Harris. Comparison of two measurement fusion methods for kalman-filter-based multisensor data fusion. *Aerospace and Electronic Systems, IEEE Transactions on*, 37(1):273–279, 2001.

[22] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3354–3361. IEEE, 2012.

[23] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960.

[24] Thomas Dall Larsen, Nils A Andersen, Ole Ravn, and Niels Kjøjlstad Poulsen. Incorporation of time delayed measurements in a discrete-time kalman filter. In *Decision and Control, 1998. Proceedings of the 37th IEEE Conference on*, volume 4, pages 3972–3977. IEEE, 1998.

[25] Martin E Liggins, Chee-Yee Chong, Ivan Kadar, Mark G Alford, Vincent Vannicola, Stelios Thomopoulos, et al. Distributed fusion architectures and algorithms for target tracking. *Proceedings of the IEEE*, 85(1):95–107, 1997.

[26] Nickens N Okello and Subhash Challa. Joint sensor registration and track-to-track fusion for distributed trackers. *Aerospace and Electronic Systems, IEEE Transactions on*, 40(3):808–823, 2004.

[27] Anna Petrovskaya and Sebastian Thrun. Model based vehicle tracking in urban environments. In *IEEE International Conference on Robotics and Automation, Workshop on Safe Navigation*, volume 1, pages 1–8, 2009.

[28] Donald B Reid. An algorithm for tracking multiple targets. *Automatic Control, IEEE Transactions on*, 24(6):843–854, 1979.

[29] Simo Särkkä. *Bayesian filtering and smoothing*. Cambridge University Press, 2013.

[30] Toshio Shimada, Kenji Toda, and Kenji Nishida. Real-time parallel architecture for sensor fusion. *Journal of Parallel and Distributed Computing*, 15(2):143–152, 1992.

[31] SM-D. Cuda programming. In *Network Security 2013.1*, 2013.

[32] Luciano Spinello, Matthias Luber, and Kai O Arras. Tracking people in 3d using a bottom-up top-down detector. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1304–1310. IEEE, 2011.

[33] Bjarne Stroustrup. *The C++ programming language*. Pearson Education India, 1986.

[34] Johan Villysson. Robust tracking of dynamic objects in lidar point clouds. Master's thesis. Department of Mathematical Sciences, Chalmers University of Technology, 2014.

[35] Gary R Wright and W Richard Stevens. *TCP/IP Illustrated*, volume 2. Addison-Wesley Professional, 1995.

[36] Xue Yang, Jie Liu, Nitin H Vaidya, and Feng Zhao. A vehicle-to-vehicle communication protocol for cooperative collision warning. In *Mobile and Ubiquitous Systems: Networking and Services, 2004. MOBIQUITOUS 2004. The First Annual International Conference on*, pages 114–123. IEEE, 2004.

[37] Murali Yeddanapudi, Yaakov Bar-Shalom, and Krishna R Pattipati. Imm estimation for multitarget-multisensor air traffic surveillance. *Proceedings of the IEEE*, 85(1):80–96, 1997.

[38] Pilat Yves. Dynamic Objects Tracker in 3D. Master's thesis, Swiss Federal Institute of Technology Zurich, 2011. Supervisor: Kastner Ralf and Maye Jerome.

A

Appendix 1

Lemma (Woodbury matrix identity):

$$(\mathbf{S}+\mathbf{UTV})^{-1} = (\mathbf{S})^{-1} - (\mathbf{S})^{-1}\mathbf{U}(\mathbf{T}^{-1} + \mathbf{V}(\mathbf{S})^{-1}\mathbf{U})^{-1}\mathbf{V}(\mathbf{S})^{-1} \tag{A.1}$$

Proof: The Woodbury identity matrix is proven by checking that the Woodbury identity ($\mathbf{S}+\mathbf{UTV}$) times its inversion on the right side forms the identity matrix:

$$\begin{aligned}
&\Big(\mathbf{S}+\mathbf{UTV}\Big)^{-1}\Big[(\mathbf{S})^{-1} - (\mathbf{S})^{-1}\mathbf{U}\Big((\mathbf{T})^{-1} + \mathbf{V}(\mathbf{S})^{-1}\mathbf{U}\Big)^{-1}\mathbf{V}(\mathbf{S})^{-1}\Big] \\
&\quad = \mathbf{I} + \mathbf{UTV}(\mathbf{S})^{-1} - \Big(\mathbf{U} + \mathbf{UTV}(\mathbf{S})^{-1}\mathbf{U}\Big)\Big((\mathbf{T})^{-1} + \mathbf{V}(\mathbf{S})^{-1}\mathbf{U})^{-1}\mathbf{V}(\mathbf{S}\Big)^{-1} \\
&\quad = \mathbf{I} + \mathbf{UTV}(\mathbf{S})^{-1} - \mathbf{UTU}\Big((\mathbf{T})^{-1} + \mathbf{V}(\mathbf{S})^{-1}\mathbf{U}\Big)\Big((\mathbf{T})^{-1} + \mathbf{V}(\mathbf{S})^{-1}\mathbf{U}\Big)^{-1}\mathbf{V}(\mathbf{S})^{-1} \\
&\quad = \mathbf{I} + \mathbf{UTV}(\mathbf{S})^{-1} - \mathbf{UTV}(\mathbf{S})^{-1} \\
&\quad = \mathbf{I}
\end{aligned}$$

Where the matrices $\mathbf{S}$, $\mathbf{T}$, $\mathbf{U}$ and $\mathbf{V}$ are of the correct conformable sizes.