GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN

ISSN 1612-6793

Bachelor's Thesis

submitted in partial fulfilment of the
requirements for the course "Applied Computer Science"

Track-to-Track Association and Fusion for Cooperative Automotive Environment Perception

Simon Steuernagel

Institute of Computer Science

Bachelor's and Master's Theses
of the Center for Computational Sciences
at the Georg-August-Universität Göttingen

11. February 2019

Georg-August-Universität Göttingen
Institute of Computer Science

Goldschmidtstraße 7
37077 Göttingen
Germany

☎ +49 (551) 39-172000
FAX +49 (551) 39-14403
✉ office@informatik.uni-goettingen.de
🌍 www.informatik.uni-goettingen.de

First Supervisor:	Jun.-Prof. Dr.-Ing. Marcus Baum
Second Supervisor:	Dr.-Ing. Andreas Leich

[illegible]

I hereby declare that I have written this thesis independently without any help from others and without the use of documents or aids other than those stated. I have mentioned all used sources and cited them correctly according to established academic citation rules.

Göttingen, 11. February 2019

Abstract

Most modern cars use some form of environment perception systems to detect nearby objects and assist the driver based on this. Multiple vehicles can exchange this information to improve the accuracy of their measurements. To this end, received information needs to be fused with the locally available information, which as a prerequisite requires the association of tracked objects from different sources with each other. Such a cooperative approach to perception can lead to significantly improved position estimates and can be used to pass on information that is not available to communication partners.

This work aims at creating a system that can handle information from multiple sources by associating and fusing sensor tracking results. Incoming information is processed to synchronize data sources and fit the format that the algorithms require. Afterwards, tracked objects are matched with each other to allow for information fusion. For this purpose, an association algorithm that is based on the positional distance between objects is used. Two variations of a fusion algorithm are implemented and compared.

The system is applied to real-world traffic data gathered by multiple cars in a city scenario. Different traffic situations are analyzed alongside a simple simulation setup that provides measurable result statistics.

Contents

1 Introduction

Advanced Driver Assistance Systems (ADAS) are installed in almost every modern car. They provide the driver with valuable information and additionally can perform maneuvers such as braking in emergency situations where a human driver would not be able to react fast enough. Some ADAS applications are local to the vehicle, such as Electronic Stability Control (ESC), which is a standard system that improves a cars stability by detecting loss of traction. Other applications involve other road users as well: For example, Adaptive Cruise control holds a cars speed at a set value, but automatically brakes if leading vehicles slow down to maintain a safety distance to them.

ADAS rely heavily on object tracking. For this, different types of sensors are used, including LIDAR and RADAR sensors. These sensors provide the vehicle with distance measurements which can be processed and used for object tracking. The resulting track is the fundamental input data for the assistance application. As more and more cars are equipped with sensors and means of inter-vehicle communication, the topic of cooperative perception for ADAS is becoming more and more important. Messages received from other cars and road-side units can be used to significantly improve the local track of nearby objects, including error correction and extension of the local map past the Field-of-View (FOV) of equipped sensors.

Cooperative Perception refers to the concept that multiple cars exchange measurements between each other. This can happen on different scales: A car might only send a Cooperative Awareness Message (CAM) [1] to broadcast its own position to other road users. A more sophisticated approach could include sending a Cooperative Perception Message (CPM) which informs nearby receivers of the position of all objects in the FOV of the sender. This type of message was introduced by the Ko-PER project [2]. A receiver could then fuse this information with tracks from its own sensors, to improve and extend the list of tracked objects.

Closely related to this is the concept of Car-to-Car (C2C) and Car-to-Infrastructure (C2I) communication, as well as the combination of both, called Car-to-X (C2X) communication. This means that cars communicate with each other (only C2C) or on top of that also with road side infrastructure (C2X). Depending on the form of the messages exchanged between communication partners, different kinds of information can be send. CAMs and CPMs are the implementation of C2X communication that this work is based on.

Using received perception messages comes with several advantages over local-only perception: For example, messages about a dangerous road section can be propagated from car to car, giving following vehicles a longer reaction time and therefore preventing accidents. Another example for a dangerous situation that can be improved by cooperative perception are intersections with low visibility. A car approaching such a junction can broadcast its position and perceived objects to inform nearby road users about objects that are visible from a different angle. Furthermore the received information can be used to improve the accuracy of object position estimates.

Of course, this technology comes with challenges as well. Local sensor measurements and incoming messages are not arriving at the same time. Therefore, time-synchronization of input tracks is necessary before any further steps can be performed. Additionally, the sheer mass of information that can be received requires highly efficient algorithms and strong computation power in the fusion center, i.e. the local vehicle, since real-time processing is necessary for online use scenarios.

This work is based on track-level fusion, which means that the input data for all operations are pre-processed tracks of (multiple) objects. Ideally, this allows for sensor independence of the implementation: Various types of sensors can be dynamically fused together, since all kinds of data get processed into the same format representing a track of objects. On top of that, the overall speed performance of the algorithm is not hindered by object classification, which would need to happen multiple times if all involved cars communicated raw sensor measurements, but used the same data as input for their own tracking algorithm as well. However, this also comes with the downside that certain sensor information is lost during the tracking. For example, if multiple sensors spread across multiple vehicles all produce similar unprocessed output, then a lower-level fusion might achieve better results than a track-level fusion, because it it does not involve any assumptions made based on the tracking algorithm.
Additionally, the performance of the cooperative environment perception relies heavily on the performance of the tracking systems used across all sensors. Overall the advantages of track-level fusion outweigh the disadvantages in this scenario, which is why this approach was chosen. This is often referred to as Track-to-Track (T2T) fusion.

The main goal of this work was to develop a T2T association and fusion algorithm for real-world data gathered in a city environment by multiple cars equipped with sensors and object tracking systems.

This involves synchronizing received data and pre-processing it as necessary to create suitable input for multi-object algorithms with a dynamic number of sensors as possible input sources. Furthermore, the data from different sensors needs to be associated, i.e. objects in different measurements need to be matched to each other. Afterwards, this resulting data can be fused in different ways, for example to extend the FOV of a car or to improve the accuracy of the positional measurements of tracked objects.

This work was done in direct cooperation with the DLR Institute of Transportation Systems.

2 Related Work

Similar problems and approaches have been examined in previous works:

Houenou et. al. presented a multi-sensor data association approach in [3]. Their work is not based on cooperative perception, but on multiple sensors that are all used by a single vehicle. This problem has many parallels to the association problem in a cooperative environment, since in both cases data from multiple sources that might be received in an asynchronous fashion needs to be processed. They identify the main difference between such a situation and a mono-sensor situation: in the latter, measured data needs to be associated with previously generated tracks from the past, but in this case different measurements need to be associated with each other to form such a track. Due to the possibly changing number of inputs and the fact that the number of inputs is not limited to 2 (old track and new measurement), this requires new algorithms. For this, they present a multi-sensor Track-to-Track algorithm and a corresponding distance measurement for objects. Their work is on an object-level scale, and uses only positional and velocity data along with covariance values.

In [4] a slightly different approach to the association is discussed. Here, the goal was to integrate non-kinematic information such as target width or target classification with classes such as e.g. "car, truck, pedestrian" into the algorithm. If this information is accurately given, performance of the association was able to be drastically improved by incorporating it into the process. The approach was not designed with cooperative perception in mind.

An example of an application for ADAS is active blind spot detection, which warns the driver if objects are close to the blind spot on the side of the car. This can be combined with information gathered from data transferred via C2X communication, resulting in a cooperative blind spot detection system [5]. It is important to note that this approach only extends a given blind spot detection system. If the system would heavily rely on received data, it could not be applied generally, since so far only few vehicles use the required technology. The underlying concept of the work is to extend the measured map of objects with information received from other vehicles, therefore expanding the range in which cars in the blind spot can be detected by a significant margin. This proved to be a very successful extension of the basic blind spot detection system, and therefore shows how rather simple fusion techniques that only rely on extending the range in which objects can be used for higher-level functions can be successfully involved in building cooperative ADAS.

If a more classic approach to Track-to-Track fusion is pursued, in which multiple object tracks should be fused together to improve the accuracy of the estimated position of each object, different algorithms need to be employed. Seeliger and Dietmayer [6] tested multiple variations of the so called Covariance Intersection algorithm, which is described in detail in Section 4.2, in a multi-vehicle, multi-object scenario where a full suite of processing and fusion algorithms was applied. Additionally, other algorithms requiring more input data were tested as well. Their approach involved four steps:

1. Temporal Alignment
2. Spatial Transformation
3. Association
4. Fusion

The focus of their work was the information fusion, where the algorithms were compared with regard to measurement errors. They determined a best choice of the variants of the Covariance Intersection and applied this to real-world data acquired from multiple vehicles.

These works show different approaches to the same or similar problems, even though not necessarily based on a cooperative approach. Yet as pointed above, a general multi-sensor situation has many parallels to a cooperative situation.

3 Problem analysis

The presented problem can be broken down into three parts: First, the data needs to be prepared to enable further processing. Afterwards, object tracks in the processed data need to be associated with each other, which in turn is the input for the final information fusion step. In the following section, the content of the data and the problems of association and fusion are analyzed.

3.1 Data Review

The DLR Institute of Transportation Systems recorded traffic in a city scenario using multiple vehicles equipped with LIDAR sensors. The resulting data was provided to serve as the basis of this work.

The available data was split into three parts. The first two batches of data included tracking results based on LIDAR data from the ego-vehicle, as well as a record of incoming CAMs that contained positional information from a single car with the necessary equipment, that was driving near the recording car. The third batch was formatted differently: It consisted of two files that each contained tracked objects from a LIDAR scanner. These were recorded simultaneously and could hence be used to simulate received CPMs for one of the objects.

The first record was 17 seconds long. It consisted of a single situation: The car with C2X equipment passed the ego-vehicle while continuously sending CAMs. After driving ahead of the ego-vehicle for a few seconds, the recording was stopped.

The second file was significantly longer, with a total length of 228 seconds. Multiple road scenarios were captured in this recording, including turns, overtaking maneuvers as well as "normal" straight driving. Just like the first file, it included a LIDAR scan with object tracking generated by the ego vehicle and CAMs from a second vehicle.

The third batch was recorded while two cars were repeatedly going through two roundabouts with a short straight road between them, therefore passing each other multiple times. Here, no data was exchanged between the two vehicles, but the LIDAR scans of both cars were available.

They were used to simulate a CPM from one car to the other, since this message format includes exactly this kind of data.

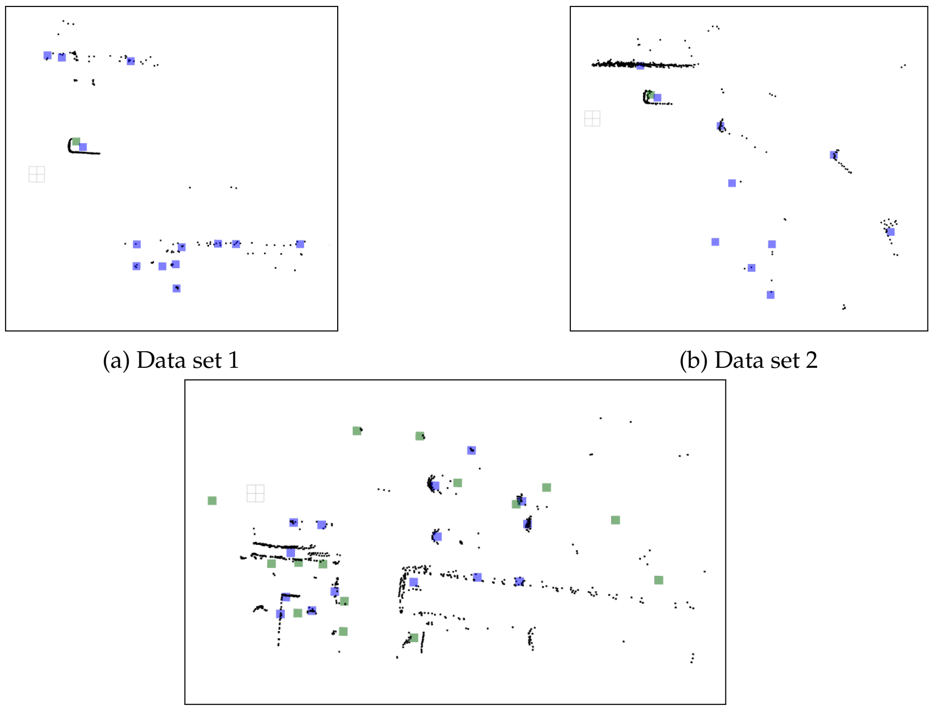
Content and Quality of the Data

All data included positional data of all tracked objects and ego-positions in CAMs where applicable. This is the most important part of the data, since it is the basis of all algorithms. The positions of objects did not jump around unexpectedly, which would have indicated measurement errors. Velocity data was also included for all objects. However, since this is more difficult to acquire, its accuracy was lower than the accuracy of the position estimates. For example, static objects on the side of the road did not get assigned a velocity of zer, but instead had a velocity value that changed over time, which impedes differentiation from actual vehicles that do have a real velocity harder. LIDAR scan results also included the size of a tracking box, however this data was changing a lot over time as well. Since most used algorithms are not made for this kind of more elaborate object information, it was discarded for this application. Additionally, CAMs only included a static value for box length and width, and therefore made comparisons for association or fusion obsolete. Rotation angle data was provided as well, but since it was not used in any step of the process, its quality was assessed only shortly. Objects that moved in a single direction had a changing angle, so the measurement was not perfectly accurate. However, it is possible that it could still be incorporated into the state space. Since this estimate never had covariance data attached, it could not be easily used along side the positional or velocity measurements in advanced algorithms.

In the first two datasets covariance data was only present for the LIDAR scans, but not for the CAMs. This made more elaborate approaches to association and fusion more difficult , since most state-of-the-art approaches are heavily based on covariance information. The third dataset did not include any covariance data.

In general, the objects that were included in the tracks of the LIDAR scanners had no classification attached, making it hard to distinguish between vehicles and objects on the side of the road. This was an issue in the third dataset, where too many objects were close to each other, making it impossible to even visually tell apart vehicles from other tracking results if no filtering was performed on the raw tracking data.

Due to their definition, CAMs arrived at a significantly lower frequency than measurements from the local LIDAR-based tracking system. The information on local tracking of cars in the third dataset was not sent with an intentionally low frequency as it was manually merged afterwards and could therefore be synchronized easier with the local data.


(a) Data set 1

(b) Data set 2

(c) Data set 3

Figure 3.1: Visualization of point clouds and tracking results in all datasets. Squares represent object tracks, and different colors indicate different sensor origin. Black dots represent point cloud data.

Visualization of the Data

Figure 3.1 shows a visualization of the core information included in the datasets. The presented data has not been filtered in any way. All data sets included the original point cloud measured by the LIDAR-scanner, which is represented by black dots that are spread across most of the images. The ego position of the recording vehicle is shown by a white two-by-two tile. Since the task was based on track level, this data was only used to analyze output and evaluate results.

On top of the point cloud, the tracking results are displayed. Objects are internally represented by position, angle, velocity and bounding box size. The visualization is merely based on the estimated position, and other variables are only used as algorithm input as explained above. Blue squares represent an object tracked based on the LIDAR data in all three images. Green squares represent information received from other vehicles, or in the case of the third dataset the tracking result of the LIDAR scanner of the second vehicle.

In Figure 3.1a it is easily possible to tell apart cars from other objects. While a lot of road side objects are tracked as well, these are grouped at the side of the road and are therefore unlikely to interfere with association or fusion algorithms. In parts of the second dataset, shown in Figure 3.1b, this is slightly more problematic: multiple vehicles spread across a curve are tracked. Yet several other objects are included in the tracking as well, some of which are far enough away to not interfere, while others are in positions where are car might potentially be. Without the point cloud as a visual reference, telling apart cars from road side objects would be significantly harder in this situation.

The third dataset suffers from the same problem. As shown in Figure 3.1c, the tracks from both vehicles include a significant amount of clutter. Even with the point cloud as an overlay, the unfiltered data is near impossible to analyze.

This shows why pre-processing, association and fusion algorithms are necessary to evaluate information gathered by sensors.

3.2 T2T Association

The association is the first necessary step to fuse data from different sources, and can itself already provide valuable results. Without this step, no fusion is possible, since all fusion algorithms work on associated objects, but different sensors will provide only their own measurement.

Track-to-Track Association refers to the problem of finding matching objects in a given list of tracks. In practice, every sensor produces a different set of tracked objects, each with its own unique ID. However, an object will most likely have different IDs assigned in different sensor tracks. The goal of the association step is therefore to create clusters of IDs, where each entry is the ID of a different sensor track referring to the same object. For example, if an object has been assigned ID 501 by sensor X and ID 11 by sensor Y, the goal of the association algorithm would be to create a list that matches the ids X.501 and Y.11 with each other.

For the association, different kinds of data can be used. The most basic operation involves only positional data of all objects, and using that creates matches based on measured physical distance. However, more elaborate approaches might involve using a bigger state space. Other entries that can be used for comparison of objects could for example be: velocity, object size or even object classification, depending on which of these are available. Additionally, if covariance data is included in the measurements, this can be used to weigh entries of the state space against each other, for example by reducing the impact of uncertain measurements, which would come with higher covariance entries.

The distance between two objects in the general case is not the physical euclidean distance, but rather an abstract metric of similarity between two objects. Ideally, an object should have a distance of 0 between its measurements in the tracks, however, due to measurement errors and sensor differences, this is highly unlikely to be the case in a real scenario.

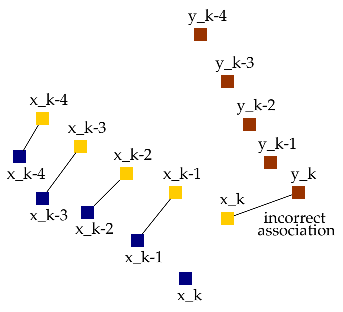


Figure 3.2: Problematic situation for association due to path intersection of multiple vehicles. Every square represent a measurement for an object at a certain time step. Different colors indicate different sensors and the labeling indicates the object (either x or y) and the respective time step (k-4 to k)

The association algorithm has to fulfill two conditions with regard to the handling of data points, according to [3]:

1. One object may not be associated with different objects in the same time step.
2. Association should be performed between tracks, so no object from a track should be associated with an object from the same track.

Another possible issue that can arise during the association is visualized in Figure 3.2. Here, the different colored boxes represent position estimates from different sensors. The yellow and blue boxes belong to the same object x, while the red box belongs to object y. In time steps $k-4$ through $k-1$, the correct association will be made. However, due to slightly increased measurement error in time step k, the estimate for y is now closer to the yellow-marked estimate than the blue one. Therefore, if only the most recent estimate is looked at, a mismatch will be caused by this situation. This can happen when two tracks of the same object from different sensors are slightly apart, and another object crosses the objects paths. Then the association will likely be incorrect for the time steps where the new object is in close vicinity to the trajectory of the first vehicle.

The issue was also discussed by Houenou et. al. [3], where a *track history* was introduced as a possible solution. This means that during the association, not only the most recent measurement is used to estimate the distance between the objects, but instead past measurements are compared to each other as well. This solves the issue described above, since then objects that were close to each other for a longer time will take priority over objects that just got very close for one or two measurements.

However, the implementation of a track history is no universal solution to the problem. The concept is based on a close to perfect in-track association, i.e. the fact that one real-world object never changes its ID within a single track. Yet due to measurement errors, loss of sight and other problems, loss of in-track association is a possible and indeed frequent issue. In that case, the history for the track is reset, and the concept of track to track history provides no benefit, until a new history has been built. A second problem is computation time: the association algorithm needs to be as fast as possible, and having to compare more tracks in every step can cause performance issues, that in a real application might cause the algorithm to lag behind the live time. Consequently, the size of the history that is compared in every association step needs to balanced between a large value for better prediction and a smaller value for better runtime.

This issue of performance becomes even more pressing if the incoming data is not perfect synchronized. In that case, the past steps need to be compared to each other, and for every measurement a matching measurement with respect to their time stamps from the other sensor needs to be chosen.

Another option for dealing with similar scenarios without relying on a track history is using velocity and orientation information. If this data is available in high enough quality, it can be used to successfully distinguish objects that only converge for a short span of time, since they will likely be moving at different speeds (overtaking) or in different direction (crossing paths). To rely on such information requires well-known data quality though, which is especially hard in the field of cooperative perception, where one must assume that communication partners might use outdated sensors or tracking techniques.

3.3 T2T Fusion

After objects were successfully associated with each other, the resulting output was used to fuse the data of different sensors together. For this, different goals are possible:

- Field-of-View extension
- Improvement of measurement accuracy
- Error Detection and Correction

Field-of-View extension refers to increasing the range in which the ego-system is aware of other objects. Usually, the FOV is based on local sensors, such as LIDAR/RADAR-scanners or cameras. However, in that case every wall will obstruct the sight of the car. Positional data from other vehicles can then be used to extend the map past such obstacles. Depending on the received data, this can be achieved on different levels: If only CAM data is received, the position of CAM-senders can be added to the local map, unless these positions were associated with objects that were detected already in the previous association step. This prevents duplicate objects from being added to the map. If CPM data is received, then the maps can be merged, and all objects that were not associated with an object on the local map can be added. While this can potentially be a lot more powerful than only merging positions of the sending objects, it comes with the important assumption that senders classify objects in the same way as the local system. If the local system attempts to only track road users, but another system also includes static road side object tracks, then the local system needs to be able to differentiate between these types.

The classic approach to cooperative data fusion is the improvement of measurement accuracy. Given the state vectors of two or more tracks of the same object but from different sources, all these measurements should be fused together to create a single, more accurate state estimate of the object. For this, a plethora of well-tested algorithms is available. However, most of these algorithms are based on using the covariance of measurements alongside the actual measurement to accomplish this. While there have been a few works regarding fusion of data from so-called legacy sensors which do not provide covariance data, for example by H. Chen and Y. Bar-Shalom [7], this has not been a focus in recent developments. When seen in the context of cooperative fusion, another issue of this goal becomes apparent: If only few of the tracked objects send additional data, performing fusion for only these single tracks will potentially cause disparity between the tracking of objects with C2X equipment and those without it.

Another possible target of track-to-track fusion that can be explored is error detection and correction. If a local system loses track of certain object due to measurement errors, this can be detected by combining the local information with additional information from different sources. For this, one can imagine lots of different scenarios and possible errors. For example, if a received C2X message indicates an object in front of a detected object, it is likely that some kind of measurement error is happening, since usually this object would block the line of sight to the detected object and should itself be tracked already. However, this comes with the inherent issue of trusting received information over locally measured data, at least if error correction should be performed. If detection of errors is the only goal, a situation as described would provoke some form of warning, but no decision on which data should be trusted needs to be made.

For both association and fusion, different algorithms can be used. The details of selected algorithms tackling the problems presented above will be presented in the following chapter.

4 Algorithms

Two types of algorithms are necessary to achieve cooperative perception: Those for association and those for fusion. These are discussed independently in the following chapter.

4.1 Algorithms for Association

The association is mostly based on a single algorithm as presented by Houenou et. al. in [3]. They describe the algorithm as "a generalization of the Nearest-Neighbor algorithm to more than two data classes" [3, page 707].

The basic idea of their algorithm is to group all measurements of all sensors into a quadratic matrix, and iteratively select best matches from this matrix. These matches then form clusters, that each represent a group of measurements belonging to the same real-world object. By manipulating the matrix entries in a specific way, certain errors, such as matching measurements that originally belonged to the same sensor, are prevented.

As a first step, a metric for the distance between two objects needs to be established. To this end, the authors define $d_k^{(a,b)}$, the distance between tracks a and b at time step k. A track for this purpose includes the following information: X_k^a is the state estimate of track a at time k. The state estimate usually consists of at least the x- and y-Position of the object, but can also include more information, such as velocity. P_k^a is the covariance estimation that belongs to track for X_k^a. Therefore, if the state space consists of n Dimensions (e.g. $n = 2$ for pure positional data), X_k^a will be an n dimensional vector, while P_k^a will be a $n \times n$ matrix.
Equation 4.1 is the definition of $d_k^{(a,b)}$ according to [3]:

$$d_k^{(a,b)} = (X_k^a - X_k^b)^T \cdot (P_k^a + P_k^b)^{-1} \cdot (X_k^a - X_k^b) + ln(|P_k^a + P_k^b|) \tag{4.1}$$

This is very similar to the Mahalanobis distance, which is defined as follows for two vectors x and

y with their covariance matrix P:

$$d_{mahalanobis}(x, y) = \sqrt{(x - y)^T \cdot P \cdot (x - y)} \tag{4.2}$$

The Mahalanobis distance is commonly used in similar scenarios, for example in [8]. The differences between the two distance definitions are the square root in the Mahalanobis definition, and the addition of the term based only on covariance in the definition of Houenou et. al. The lack of the square root is easy to explain: all distance values are only used to be compared with each other, and adding the square root into the formula would not change the outcome of any comparison as it is a strictly monotonically increasing function. Therefore, it would be wasted computation time. The authors give no explanation for the offset introduced by $ln(|P_k^a + P_k^b|)$. One possible explanation would be to prevent tracks with high covariance, which implies high measurement uncertainty, from being associated with each other, since the distance between them might only be small because of measurement errors.

In Section 3.2 the notion of a *tracking history* was introduced. When such a system is used, the final distance calculation does not only depend on the current measurement from time k, but also on previous measurements $k - 1$, $k - 2$, ..., $k - n$. Here, n is the size of the tracking history, in numerical discrete time steps. For $n = 0$, no history is considered and $D_k^{(a,b)} = d_k^{(a,b)}$.
For $n > 0$, Equation 4.3 defines $D_k^{(a,b)}$:

$$D_k^{(a,b)} = \frac{1}{n} \sum_{i=0}^{n-1} d_{k-i}^{(a,b)} \tag{4.3}$$

Note that the tracking history size n is defined as a universal hyperparameter. However, since different tracks have different length, n needs to be chosen carefully in every step. Let n_{target} be the global selection of the maximum tracking history size, and $|a|$ be the current length of track a. Then, for the comparison of tracks a and b the parameter n needs to be selected as follows to fit it to the global parameter and the length of both tracks:

$$n = \min(n_{target}, |a|, |b|) \tag{4.4}$$

For this work, the state estimate used was only based on positional distance. However, if more elaborate variables are included in the state estimate, this simple numerical comparison can potentially cause issues. For example, if a numerical representation of the object class was included, this distance calculation would not make effective use of it.
Another potential issue arises with data that is scaled differently. If velocity data would be included, the same difference in velocity and position would be weighted equally. However, depending on the used units, the difference in velocity could be significantly more impactful than the positional one. For example, if two measurements were being compared with both the difference in position

and velocity being approximately three. Then, if the distance is given in metres, and the velocity in meter per second, a difference of three for the position could simply be caused by measuring different ends of the same vehicle, but a difference of $3m/s = 10.8km/h$ can be huge in a city scenario. Still, these differences would have the same impact on distance estimation, unless their weighting is included in the covariance matrix in some way.

The algorithm for T2T association is based on a matrix that stores the distance between all tracks. For this, all tracks of all sensors are assigned a number between $1..N$. The distance matrix M is of size $N \times N$. For the initialization of the matrix, the following steps need to be performed:

- First, initialize the following variables and functions:
  - Introduce a value called $MaxVal$. This value should be greater than any potential measurement, and represents that the two tracks that are connected by a cell with this value may not be associated.
  - Let G be the *gating threshold*, which is the highest possible distance value two measurements may have, that still allows for association of the tracks.
  - Define the function $sensor(i)$, which returns a numerical representation of the sensor that produced track $i \in [1, N]$.
- Then, the distance matrix M needs to be filled. Every cell of M holds the distance between the two measurements $i, j \in [1, N]$ or $MaxVal$, according to the following definition:

$$M_{i,j} = \begin{cases} MaxVal & \text{if } i = j \\ MaxVal & \text{if } sensor(i) = sensor(j) \\ MaxVal & \text{if } D_k^{(i,j)} > G \\ D_k^{(i,j)} & \text{otherwise} \end{cases} \tag{4.5}$$

  Equation 4.5 applies three constraints to the matrix: the first prevents tracks from being associated with themselves, the second prevents tracks from being associated with tracks produced by the same sensor and the last constraint prevents tracks that exceed the previously defined gating threshold from being associated with each other.
- The goal of the next step is to create clusters of tracks that represent the association. A cluster will then contain all tracks that belong to the same real-world object. Here the value and the position of the minimum value of the matrix is selected. This operation will be looped, and in every execution the following needs to be done, depending on how many of the two tracks that correspond to the cell with the minimum distance are already in a cluster.

Number of Tracks in Cluster	Operation
0	Add both tracks to a new cluster
1	Add the track that is not in a cluster to the cluster of the other track
2	Do nothing

After this, all cells in the same line and column as the selected cell that belong to the respective same sensor are set to $MaxVal$. This prevents multiple tracks from one sensor from being matched with the same track of another sensor.

- Once the minimum value in the matrix is found to be $MaxVal$, the loop is stopped, and all tracks that have not been assigned to a cluster form their own clusters, that therefore only have one entry and do not represent an association. These clusters are called *singletons*.

This concludes the execution of the association algorithm for a single time step. All tracks have been assigned into clusters. Clusters that have more than one entry represent an association of multiple tracks of the same object.

The algorithm is comprised of three parts that are important for a runtime analysis: The creation of the distance matrix, the loop that creates clusters and the final step in which the remaining tracks are added to singleton clusters. In the following, let N be the total number of tracks, as above.

For the initialization of the matrix, every cell needs to be analyzed using Equation 4.5. Since the matrix has size $N \times N$, this step requires N^2 operations.
Every step in the main loop to create clusters is dominated by the 'minimum' operation to find the next cell that will be used for cluster generation. The matrix entries potentially change in every step, since some are replaced by $MaxVal$, hence it is not possible to simply create a cache to reduce complexity of this in subsequent iterations. Due to that, all cells of the matrix need to be checked in every step, which means N^2 operations are necessary to find the minimum. This step needs to be performed at most once for each measurement, so an upper bound for the number of times the loop is run is N, and therefore at most $N \cdot N^2 = N^3$ operations need to be executed.
However, certain distance values in the matrix are replaced with $MaxVal$ in every step, and the breaking condition of the loop is $MinVal < MaxVal$. Therefore, the total number of repetitions is not exactly N, but $C = f \cdot N, \text{where} f \in (0,1)$. A better upper bound is hence defined by $C \cdot N^2$, yet $C = f \cdot N$ holds, so $C \cdot N^2 = f \cdot N^3$, which is in $\mathcal{O}(N^3)$.
The final step of the algorithm is to group all remaining measurements into their own singleton clusters, which takes $\mathcal{O}(N)$ operations.

All operations inside the loop and the matrix creation can be executed in constant time, except for the 'minimum' function as outlined above. Thus overall, the algorithm requires $\approx N^2 + C \cdot N^2 + N$ operations, which again is in $\mathcal{O}(N^3)$.
Therefore it is crucial to the runtime of the association algorithm to keep the total number of sensor measurements as low as possible, by removing all unnecessary measurements from the input.

This means pre-processing of the input to remove errors and tracks that are not relevant to the association, for example static road-side objects, is an important step towards a reduced runtime.

4.2 Algorithms for Fusion

After the tracks have been associated, the result can be used as input for an information fusion algorithm, that combines the measurements from different sensors into a single, improved measurement. Seeliger presents different fusion algorithms in [8]. The presented algorithms are split in two groups: Methods that involve information beyond state estimate and covariance, and those that do not. For this work, only those that do not require additional information are relevant. This leaves two possible algorithms, the Simple Convex Combination (SCC) and the Covariance Intersection (CI). The SCC algorithm is described as inconsistent and error-prone, so it was discarded for this work.

The basic formulation of the CI algorithm is the same for the "normal" variant and all variations. It uses the following variables:

P_i is the covariance matrix of track i

$\hat{x}_i$ is the state estimate of track i

ω_i is used as weighting, that belongs to track i

$\hat{x}$ is the final state estimate

P is the final covariance estimate

N is the total number of tracks to be fused

The fusion formulas are defined as follows, according to Seeliger [8, page 54]:

$$P^{-1} = \sum_{i=1}^{N} \omega_i P_i^{-1} \tag{4.6}$$

$$\hat{x} = P \sum_{i=1}^{N} \omega_i P_i^{-1} \hat{x}_i \tag{4.7}$$

$$\sum_{i=1}^{N} \omega_i = 1, \text{ with } \omega_i \in [0, 1] \tag{4.8}$$

With these equations, a final state estimate can be created from a set of track estimates with corresponding covariance. Additionally, constraints for ω are given. However, the acquisition of real values for ω_i is still undefined.

This is where the variants of the algorithm differ: in the original CI, ω_i is determined by solving an optimization problem depending on the determinant of covariance matrices. Due to the computational load that this can cause, different approximate solutions were presented.

If only two tracks should be fused, Equations 4.6, 4.7 and 4.8 can be simplified. Since only ω_1 and ω_2 will remain, one can reduce this to using a single ω, and use $1-\omega$ for the other weighting, due to the constraints imposed in Equation 4.8.

Two options of calculating this ω (that also can be extended back to the original, general case) are the Fast Covariance Intersection (FCI) and the Improved Fast Covariance Intersection(I-FCI). The approximations of ω by $\tilde{\omega}$ that these two algorithms present are as follows [8, page 56-58]:

$$\tilde{\omega}_{fast} = \frac{\det(P_2)}{\det(P_1)+\det(P_2)} \tag{4.9}$$

$$\tilde{\omega}_{improved} = \frac{\det(P_1^{-1}+P_2^{-1}) - \det(P_2^{-1}) + \det(P_1^{-1})}{2\det(P_1^{-1}+P_2^{-1})} \tag{4.10}$$

The I-FCI algorithm can be more stable than the FCI algorithm, which produces suboptimal results under certain conditions [8, page 56-58]. However, it is slightly more complex, so under heavy time constraints, usage of the FCI algorithm might be favorable. Other variants of the Covariance Intersection algorithm exist, but both the I-FCI and the FCI were found to be fast and suitable solutions, so no further methods for estimating the weighting factor are presented here.

If no covariance information is given, the CI algorithm simplifies into an averaging of the input. This can be easily shown for the case of $N=2$:
Since no covariance is given, assume $P_1 = P_2$. Therefore Equation 4.11 follows from the fusion formulas:

$$\tilde{\omega}_{fast} = \frac{d}{d+d} = 0.5 \text{ , with: } d = \det(P_1) = \det(P_2)$$

$$\tilde{\omega}_{improved} = \frac{2d-d+d}{2\cdot 2d} = \frac{2d}{4d} = 0.5 \text{ , with: } d = \det(P_1^{-1}) = \det(P_2^{-1}) \tag{4.11}$$

Because the weighting of the state estimates in this case is ω and $1-\omega$, both state estimates will be weighted with a factor of 0.5, which is equivalent to the average across all dimensions of the state space.

Using these algorithms, state estimates from multiple sources can be associated and fused with each other, if they are given in the correct format and are synchronized. The following chapter discusses the implementation of these algorithms using the previously presented data.

5 Implementation

The implementation of the program can be split into different parts that depend on each other. In the following, after an introduction of the tools that were used, these parts are presented one by one.

5.1 Tools

The Robot Operating System(ROS) [9] is used as the central framework for the program. ROS is used widely for different robotic applications. The framework is based on a set of so-called nodes, which are independent processes that communicate via messages on topics. Any node can publish to any topic, or subscribe to it to receive information about messages published by other nodes. Yet every node is independent of all other nodes in the system. This is a very useful characteristic of the framework, since it allows plug-and-play of different applications by using standardized message formats. For example, ROS includes an executable node called "rviz" which can visualize a wide variety of input, ranging from simple points to mark objects to full point clouds. This can be directly connected to the implementation from this work to visualize original input or program results.

For the implementation the free programming language python was chosen. It offers support for a wide variety of packages for different purposes and allows for quick implementation, but does not provide perfect runtime optimization. Furthermore, ROS offers native support for python.

5.2 Simulation of test data

As described in Section 3.1, the main work was based on real traffic data recorded using two vehicles equipped with different sensors. In addition to this data a simulation setup that mainly served the purpose of algorithm testing was created. Having simulated data available allows for significantly easier testing of algorithms than purely relying on real data, since such data does not come with a ground truth that can be used as a basis for comparison and is considerably more

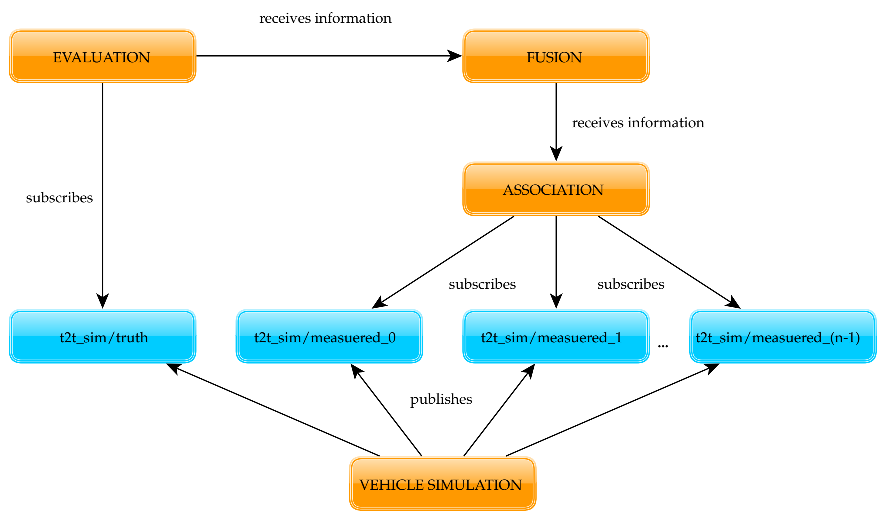


Figure 5.1: ROS Simulation Setup

noisy than desired for a simple testing scenario. A simulation with predictable outcome helps overcoming these issues.

The simulation setup was based on the same standardized message formats that the real world data was in as well. A visual representation of the ROS setup can be seen in Figure 5.1. Blue boxes represent ROS topics that the simulation uses, and yellow boxes represent central program parts. The simulation of vehicles is performed in an independent node. This node publishes messages to multiple topics: The ground truth topic and the topics that represent the measurements of the n sensors. The association algorithm uses messages from the *t2t_sim/measured_X* topics as input, and passes the results on to the fusion algorithm. The evaluation is based on the ground truth, so this part subscribes to the *t2t_sim/truth* and additionally uses the output generated by the fusion function.

This setup can be used for a variety of traffic scenarios. Any number of involved sensors can be simulated by manipulating the number of measurement topics used. Multiple runs will yield different results, because the measurement noise is randomly drawn from a Gauss-distribution. By fixing the seed of the random number generator that is used to generate the normal distributed noise reproducible results can be achieved as well. Different measurements can have different variance, which means that multiple measurement can simulate sensors of different quality.

For testing purposes, eight vehicles were simulated on two roads opposite to each other, each with two lanes. Three cars per side were moving at a slow pace in the right lane, while the fourth car was passing them on the left. This creates multiple interesting time points: Whenever the faster cars pass a slower one, potential for error rises, since the cars will be closer to each other than usual. Additionally, the two passing cars will be close to each other at some point as well, but move in opposite directions, leaving a smaller time window for mismatching. Depending on the initialization, these two things can happen simultaneously. Overall, per execution seven moments can be identified in which errors are significantly more likely than during the rest of the simulation. The cars in the simulation were moving at constant speed, i.e. no acceleration was done during the simulation, and did not make any turns.

Measurements were simulated by adding Gaussian-distributed noise to the ground truth. A more sophisticated alternative would be to track all objects using a tracking system such as a Kalman Filter or a similar state-of-the-art algorithm. This would likely yield results that are closer to what real world tracking systems would produce. Since the simulation was only used to create a very simple testing environment for a subset of all used algorithms and not to measure performance, this would provide little to no advantage over creating noise using simple random distributions. Hence, the previously described way was chosen.

The main parameter of the simulation was therefore the standard deviation of the Gauss distribution used to add noise. This parameter was then linearly changed across all simulated sensors, so that different levels of quality were included in the resulting output.

The simulation was very useful to test whether association and fusion algorithms were working correctly and producing meaningful results. It was kept intentionally simple to make verification of this easier. Yet this also means that performance of different algorithms could not be compared on this basis. There are also some important issues with such a simplistic setup, especially when comparing it to a real world scenario.

First of all, the generation of meaningful covariance data is difficult. In practice a diagonal matrix containing only the variance was used, but this is not similar to the kind of covariance data a tracking algorithm would supply. Therefore, tests regarding the use of covariance were limited.

Furthermore, as described above, measurement errors were generated using a simple noise model. For a more realistic approach, using a more elaborate sensor measurement simulation might be favorable.

The third and likely most significant constraint was the perfect synchronicity of all steps: data from all sensors arrived at the same time, with all time stamps matching each other perfectly. Such a situation is unrealistic when preparing for a real world scenario where messages can be lost, have transmission delay and different sensors will in general produce data at a different frequency. For example, one sensor might only produce a single measurement while other sensors produce five in the same time frame. While this was intentionally disregarded for the simulation, it is a very important factor that makes real world applications more challenging and therefore can not be ignored completely.

5.3 Pre-Processing of real data

Before any association or fusion algorithms can use the raw input, several steps of pre-processing are necessary. Any data that should be processed by the association and fusion algorithms needs to be synchronized. If parts of the data arrive with a slight delay, a buffering system needs to be implemented. Should the data arrive with a significant delay of for example several hundred milliseconds, it can not be used for systems that rely on live data. In that case, a prediction based on some form of velocity model can be used to extrapolate an estimate for the current position of the object, which can then be used in further processing steps.

All C2X data that was received was stored. The data includes a time stamp, so whenever the next iteration of the algorithm should be performed, a C2X measurement that matches the LIDAR measurement could be selected. The resulting data was not removed from the buffer due to the difference in frequency between the two sources. Overall, the LIDAR tracker produced more data points than the C2X objects sent. So every received message needs to be used in conjunction with multiple scans.

An optional system was created to allow simulated C2X measurements to be inserted into the buffer every time a real CAM was received. These simulated messages had a slight delay in time and used the estimated velocity of the vehicle to manipulate its position in the real world. Using this method smoothed the difference between the two sensor sources, since the LIDAR measurements would continually move away from the C2X measurements until a new data point was received and added to the buffer.

The most important step of pre-processing was the transformation of the data into the same coordinate frame. Since both cars measure coordinates in their own frames, a direct overlay between the two is not possible. ROS provides the *tf* package with *transformer* objects to transform objects between different frames. The data sets included the necessary *tf* information. For this, the two important frames were *odom*, which is a global coordinate frame in the UTM coordinate system and the *ibeo_front_center* frame. The latter is the local coordinate frame for the ego-vehicle, which is centered around the middle of the front part of the car. In the first two data sets, the CAM data was received in the *odom* frame, and the LIDAR tracking results were already in the *ibeo_front_center* frame. Therefore, only a single transformation using a *transformer* was necessary.

The transformer object supplied by the *tf* package handles both rotation and translation of the input. Hence, transformation of velocity data is not possible with it: The velocity data is relative to the origin of the coordinate system and no translation is necessary for it. Instead, the rotation of velocity needs to be performed manually, by acquiring the transformation matrix from the *transformer* object, extracting the rotation matrix from it, and finally multiplying this matrix with the vector of velocity in both directions to acquire a transformed version of it.

For the third data set, the transformation was not as simple. Since both tracks were stored in different files that were simply played at the same time, all coordinate frames existed twice. Therefore, two *transformer* objects were necessary. The first one manipulated messages that were related to the ego-vehicle, while the second one exclusively dealt with messages from the second vehicle, that was used to simulate CPMs. This second *transformer* transformed all object measurements from the *ibeo_front_center* frame of its respective vehicle into the *odom* frame. This data was then passed on as a simulated CPM. The *transformer* for the ego-vehicle had access to information for the transformation from its own *odom* frame to its local target frame *ibeo_front_center*. Even though the received data was technically in an unknown coordinate frame (the *odom* frame of the secondary vehicle), the *transformer* could change it to the local target frame. This is only possible due to the global coordinate system UTM, which is used by both *odom* frames. Using these steps, the data could be transferred between the two local frames of both vehicles.

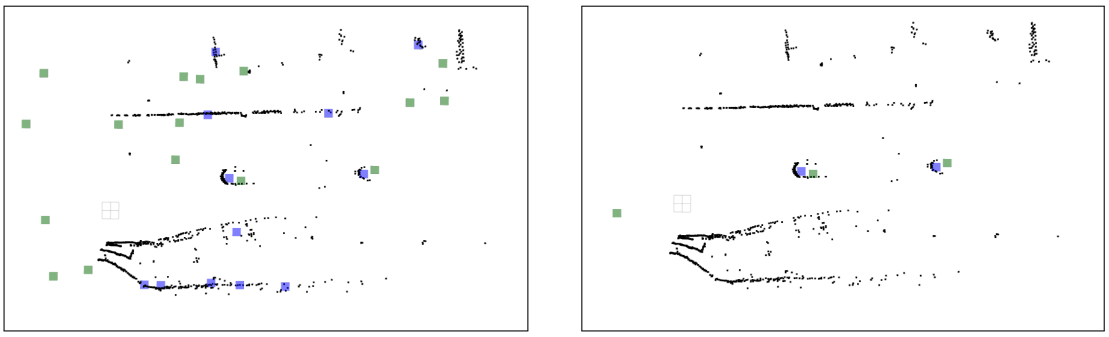

(a) Without filtering (b) With filtering

Figure 5.2: Effect of applying a velocity-based filter on the the third data set. Every square represents one tracked object and different colors indicate different sources.

The data in the third dataset was too cluttered with non-vehicle objects to be used in its raw form. Due to this, a filter needed to be employed. To this end, a velocity threshold was introduced, and all objects that had a velocity entry lower than this value were removed in the pre-processing. In an optimal case, the velocity of all stationary road-side objects should be zero, but due to measurement errors this is not the case. Therefore, the threshold needs to be selected higher, potentially causing issues with slow moving cars. Overall, such a threshold is far from optimal, since association and fusion algorithms usually should also consider stationary cars and not just moving ones. However, as seen in Figure 5.2a, the data is hard to use without a filter, and the velocity is the best parameter available.
Figure 5.2 shows the effect of the filter: on the left side, the unfiltered data can be seen. After the filter was applied, the total number of objects was reduced by a significant margin, as seen on the right side. Most of the removed tracks can easily be identified as road side objects or general tracking errors, however one object that is possibly a slow moving vehicle on the upper right side of the road is removed as well. Without the filter, this data could barely be processed at all, but after filtering the quality of the tracking is improved.

5.4 Association

The core of the final application is the association algorithm. The theoretical algorithm used has been presented in Section 4.1. In the following, it is assumed that only two sources of measurements (sensors) are present, which matches the number of sources in all data sets. However it is easily possible to generalize all necessary functions to a larger set of sensors. A central variable of this is the abstract metric of distance between tracks.

The distance between two objects from different time steps can be estimated using a simple constant velocity model. The following parameters need to be established: s is the numerical difference in time steps between $\hat{x}_1$ and $\hat{x}_2$, the position estimates of the sensors. m is a fixed multiplicative factor depending on the relation between velocity, position and measurement frequency: If two measurements per second are made, $\hat{v}$ is in m/s and $\hat{x}$ in m, then $m = 0.5$. The estimated velocity of the object in question is $\hat{v}_1$.

$$d_{constant\_velocity} = |(\hat{x}_1 + (s \cdot m \cdot \hat{v}_1)) - \hat{x}_2| \tag{5.1}$$

This is the euclidean distance between the two objects after moving the object that is behind in time steps to a new estimated position that should match the new position better.

As outlined above, the pre-processing included an option for adding simulated measurements into the buffer of object positions. The calculation performed for estimating this new position is the same as the first part of Equation 5.1. This allows normal distance estimates to work on the data, since the difference in time was reduced to a minimum by applying the simple movement model before any data got processed by the association algorithm. The distance D can then be calculated using Equation 4.3. One important issue remains: In the algorithm definition in [3], all time steps are considered to be synchronized and at the same frequency. However, the data used in this case does not fulfill these two requirements. The pre-processing was a big step towards the assumptions, but data may still arrive with slight delays and even out-of-order. So before the distance calculation can be performed, matching measurements need to be selected.

The selection of these is based on two lists of potentially different size. Each list holds objects with time stamps that represent measurements at a certain point in time. Lists can not simply be reduced in size, because the length of a list does not imply how much real time it spans. For example, if a list of length 10 and one of length 2000 should be matched, and the first list has one entry for every second while the second one has 100 per second, then taking the first 10 entries of the second list and matching those one-to-one on the entries of the first list would not produce meaningful results. Instead, the whole list needs to be searched.

Comparison used	History size in seconds	Seconds to process information
$keycomp_{straightforward}$	0.5	21.184532
	3	335.114723
$keycomp_{unordered}$	0.5	6.659661
	3	17.223231

Table 5.1: Absolute runtime of different history matching functions

The straightforward way of doing this is to iterate over objects in the smaller list, select the best match of the second list for the current object, and store this pair. Let l_1 be the size of the first list, and l_2 be the size of the second one. Then the total number of comparisons necessary is:

$$keycomp_{straightforward} = \min(l_1, l_2) \cdot \max(l_1, l_2) \tag{5.2}$$

For the sake of evaluation, assume that the lists are balanced in size, so that $l = l_1 = l_2$. Then, $keycomp_{straightforward} = l^2$. So this straightforward way requires a quadratic amount of time stamp comparisons.

One can not assume that the lists are already ordered, since sensor measurements may arrive out-of-order. For ordered lists, the number of key comparisons can be reduced to:

$$keycomp_{ordered} = \max(l_1, l_2) \tag{5.3}$$

by iterating over the smaller list once and storing a position bookmark in the second list, since then every element of the longer list only needs to be looked at once. This is not possible for unordered lists, since in that case every match for an entry in the smaller list needs to be looked at independently. When using such a bookmark, even in the worst case scenario where the last entry of the longer list matches the last entry of the smaller list, the number of necessary comparisons is still linear with regard to the size of the longer list. Sorting of lists can be easily done in $\mathcal{O}(n \log n)$. So even if both lists need to be sorted, the total number of comparisons can be reduced to:

$$keycomp_{unordered} \approx (l_1 \log l_1) + (l_2 \log l_2) + \max(l_1, l_2) \tag{5.4}$$

If the lists are balanced in size, this means $2(l \log l) + l$ comparisons are necessary, which is significantly less than what is necessary in the straightforward implementation.

Table 5.1 provides information on the actual runtime of the different implementations. For all testing, the same scenario was measured. The system used has 8 GB of RAM, and uses an Intel$^{®}$Core^{TM} i5-6600 CPU.
As expected, the approach based on sorting lists first achieves better results. Most notably, it scales significantly better with an increase in history size: Increasing the history size by a factor of six only increases the runtime by a factor of $r_{improved} \approx 2.59$. For the straightforward way, the factor

was $r_{straightforward} \approx 15.82$. A large history still contributes negatively to the overall runtime. If no history were to be considered, the overall runtime is reduced to 0.720123 seconds in the same scenario.

The association algorithm was implemented as presented in Section 4.1. This required managing the matrix of track distances, along with another array that stores which track was assigned to which cluster. A cluster is a set of sensor measurements that are all assumed to originate from the same real world object and therefore should be associated. To maintain order within the non-singleton clusters and allow matching of associated objects back to their original tracks without needing extensive object comparisons, the order of tracks in the cluster should be same as the order of tracks in the input for the association algorithm. Alternatively, it can be ensured that the object IDs of tracks are not just unique within the track, but also across all tracks.
The final result that the association algorithm produces is a list of clusters, including singletons. This can then be used as input for any further data processing steps.

5.5 Fusion

To fuse these results, the CI algorithm as discussed in Section 4.2 was used. Both the FCI and the I-FCI variants for approximating ω were implemented. In practice, the partial lack of covariance estimates in the real world data meant that the algorithm was tested on the simulated data, but was not applied to the data sets. For these, if fusion should be performed at all, a simple average method yields the same result as a CI algorithm would, as explained above.

Depending on the usage of the data, other forms of fusion are possible as well: For example, a simple map extension technique could be realized by extending the list of tracked vehicles by those from the external task that were in singleton clusters, i.e. not associated with any object of the local map.

A different possible application of T2T fusion is error detection and correction. For this, a wide variety of use-cases are possible, and depending on those the association results need to be processed differently. There were no obvious tracking errors found in the data sets, which means that data would need to be simulated to test any implemented methods for this goal. Due to this, no example of error correction or detection was implemented.

5.6 Visualization

After the fusion step, the results need to be visualized in some form to facilitate the evaluation of the algorithm results. For the association, a simple command line print of all non-singleton clusters proved to be a simple yet effective solution. After the association and fusion the visual representation of objects was extended by differently colored dots that represent the algorithm output. If only association but no fusion was performed, the color of markers that represent the associated objects was changed. If association and fusion were performed, new objects were placed at the positions that resulted from the fusion of association clusters.

All visual output had two possible formats. The data was either plotted into a separate window where only dots that represented tracked objects were visible, or it was published to a ROS topic as markers that could be visualized in rviz. This allows for overlaying of the tracking results on top of the point clouds that the LIDAR scanner produced.

Plotting the tracks in an external window means significantly less visual objects are displayed at the same time. This can be a great advantage when looking at the position of specific cars or results. Furthermore, it is easily possible to manipulate the display to the needs of the current situation. For example, the visualization can be changed from simple dots to rectangles representing the bounding boxes of the tracks. On top of the position, object IDs can be added as labeling, which is very helpful for recognizing cars in clutter of road-side objects, or for checking the consistency of ID labeling of vehicles.

When visualizing in rviz, markers are added as an overlay to the point cloud visuals. This means that matching between point cloud clusters and tracked objects is easier compared to the other approach, where a second window with the point cloud needs to be open in parallel. Due to that, the approach was in general more suitable for the analysis and evaluation of output results. The third dataset suffered from a high amount of non-vehicle objects in the tracking, and to remove them a velocity threshold needed to be defined. To test the impact of different values, the overlay visualization was invaluable.

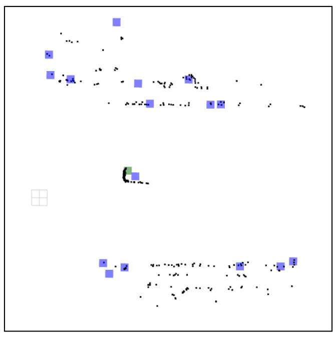
6 Evaluation

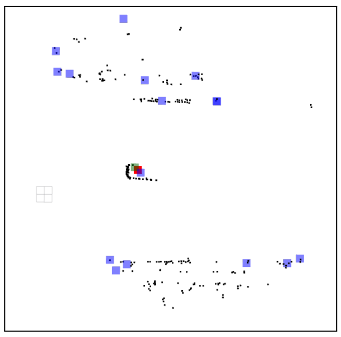
Overall, evaluation of real-world data is difficult since no ground truth is available for comparison. This means that association and fusion results from the three data sets can not be analyzed statistically. Instead, a visual interpretation of results was conducted.

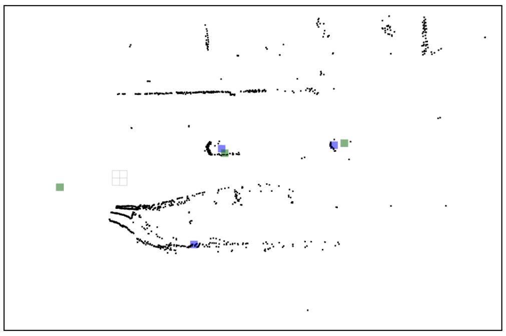
Sample output can be seen in Figure 6.1. Here, fusion output for two different scenarios is presented. The first is from data set 1, the second one from data set 3. On the left side, the normal data visualization can be seen. Fusion output was added to this on the right side. Results for the association are implied: since the fusion was simply an average operation between two data points, the red marker representing it will be placed in the middle between the two associated objects. Throughout the whole data set, very few association errors were made. Situations that were prone to errors usually included a loss of tracking of a car. If another tracked object was close enough to the real world car that was not represented in the track anymore, the association could switch to this object, causing incorrect results. This was not observed frequently though.

In general, a tracking history was rarely needed. This is to be expected though, since it has a specific purpose, as outlined above. A situation where it was necessary to prevent mismatching is shown in Figure 6.2. On the left side, the association and therefore also the fusion is executed correctly. The CAM visualized by the green marker is associated with the vehicle slightly below it. Right after this time step, the objects move, and the slight measurement error that causes the offset between the CAM and the tracking from the ego-vehicle is large enough to move a non-vehicle object closer to the green marker than its real match, as seen on the right side. Due to this, the CAM is temporarily associated with an incorrect object. This situation can be avoided by using a track history. In this case, incorporating measurements from the last two seconds is enough to correct the distance scores in such a way that the association is performed correctly.

Overall the T2T association algorithm implemented proved to be effective on the real world data. There were barely any errors in the association step. The fusion step was reduced to an average operation due to the partial lack of covariance information. Depending on the use-case, it might make sense to leave it out if no covariance is available. The use of a tracking history in general improved the accuracy of the results, however it negatively affects the runtime performance of the program. This was the main issue that remained after implementation of the algorithm: The


(a) Data set 1 without fusion marker


(b) Data set 1 with fusion marker


(c) Data set 3 without fusion markers

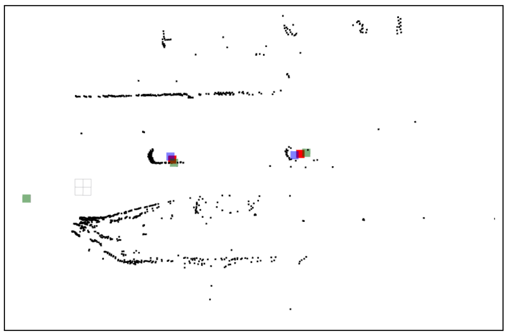
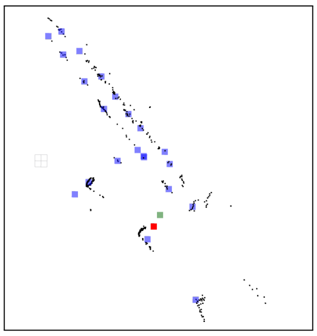

(d) Data set 3 with fusion markers

Figure 6.1: Visualization of fusion results in the real world data


(a) Correct association

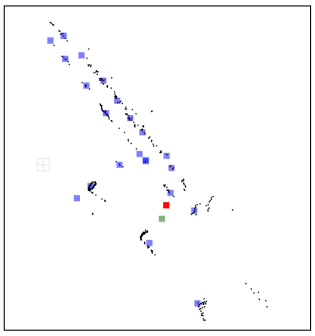

(b) Incorrect association in the same situation

Figure 6.2: Association error made without tracking history. In the figures blue markers represent the objects generated by the LIDAR tracker, green markers represent CAMs, red markers represent fusion estimates and the two-by-two square represents the position of the ego-vehicle. For graphics regarding data set 3, the green markers represent objects from the LIDAR tracker of the secondary vehicle.

program is not fast enough to work on live data, however when the collected data is played at a reduced speed, the program is able to follow.

CI Variation	σ_1	σ_2	$RMSE_1$	$RMSE_2$	$RMSE_{CI}$	Improvement(%)	$1 - \omega$	Variance
—	2	2	2.787	2.735	1.932	30.689	0.5	3.636
FCI	2	3	2.850	4.108	2.593	9.030	0.165	5.294
I-FCI	2	3	3.0166	4.096	2.433	19.333	0.452	17.451
FCI	6	7	8.468	9.514	6.641	21.569	0.351	4.850
I-FCI	6	7	8.629	9.450	6.288	27.132	0.481	12.453

Table 6.1: Average RMSE Values across ten Monte Carlo runs

The T2T fusion algorithms were evaluated using the simulation setup that was created as detailed in Section 5.2. Ten Monte Carlo runs were performed for different sensor and algorithm parameters. The average results of these runs are presented in Table 6.1. Five different parameter setups were tested. For the measurements, two parameters are important: σ_1 and σ_2. These are the standard deviations of the normal distributions that were used as noise for the measurement. The covariance matrix returned by the sensors was a diagonal matrix containing only the respective variance values, based on $\sigma_{1|2}$. Both the I-FCI and the FCI algorithm were tested with two different σ settings. Additionally, the special case of $\sigma_1 = \sigma_2$ was tested once. In that case, both algorithms result in a simple average operation between the two sensor estimates.

The measurement results are presented as Root Mean Square Error (RMSE) in an abstract unit of measurement distance that is roughly equivalent to metres. All RMSE values were compared to the ground truth, i.e. the real position of the car at the time. In all test cases, the first had better measurement accuracy ($\sigma_1 \leq \sigma_2$). All results of the fusion were therefore compared to the results of this sensor. The improvement in percent of the fusion over the single measurement is defined by: $\frac{(RMSE_1 - RMSE_{CI})}{RMSE_1} \cdot 100$.
Additionally, the table provides the weighting of the secondary sensor as $1 - \omega$. The ω value is calculated by the CI algorithm variant that was used. The last column of the table contains the variance of the improvement in percent across all ten Monte Carlo runs.

The simulation shows that the fusion provided an improvement in positional accuracy over direct use of the main sensor in all scenarios. The effect of the algorithm was noticeably better when higher values for the standard deviation were chosen, yet the fusion of two "good" measurements proved to be the most effective case. Furthermore, the table shows some differences between the FCI and the I-FCI algorithm. On average, the I-FCI achieved better results. This is due to the selection of ω, which is significantly closer to 0.5 when using the I-FCI. Therefore, the second sensor was weighted almost as high as the primary sensor. The FCI algorithm provides a more conservative estimate that is mostly based on the better measurement. In this scenario, this causes a worse performance. However, the variance of the improvements in accuracy for the I-FCI is significantly higher as well, most likely because noisy measurements are included with a too high weight that cause errors in the final estimates. Overall this points to the I-FCI as the superior variation of CI algorithms, especially since the FCI algorithm is unstable under certain conditions.

7 Conclusion

The goal that was set in the beginning was to create a program for T2T association and fusion of real world data. In general this was achieved: the association algorithm performed well on real world data, and while the fusion algorithm was not tested on the data sets, simulation results show it is working as expected. Pre-processing of the measurements was necessary to reshape the data to fit as input for the algorithms. It was implemented as a core part of the system ranging from time synchronization, which was necessary in most cases, to filtering of data and application of a constant velocity model to data that arrived at low frequency.
Since the program is based on the ROS framework and no proprietary software, it is easy to use with other systems, as the independence of ROS nodes allows fast plug-and-play with other applications.

The main flaw of the final program is that it is not fast enough to work on live data. Instead, input needs to be slightly slowed down compared to the original measurements. Depending on parameter selection, the association can either be faster or more accurate.

The first step in future work should be the improvement of the runtime performance of the algorithms. For use in an actual vehicle, real-time updates need to be processed without compromising accuracy. In its current state, the program can only be used to evaluate previously recorded data, but can not be integrated into cooperative ADAS.

Furthermore, several other modules can be added. An internal tracking system that creates estimated covariance matrices for all measured objects would allow T2T fusion algorithms to be used in any scenario, which is a fundamental requirement of different applications that might use software like this. There has also not been any testing regarding error detection and correction. This is however an interesting field of application, as it could be used to support all kinds of already installed assistance systems by improving estimation accuracy of the internal sensors.
All testing was conducted using only two sensor sources that provided similar data. If employed in a traffic scenario, the number of sensors that provide data at the same time would likely be varying. On top of that, the types of data sources used would also not be limited to LIDAR scanners, but for example might include cameras installed at traffic lights. In theory, all implemented algorithms work with an arbitrary number of input sources. However, effects on runtime or overall

performance were not analyzed so far. Additionally, as other sensors might send data delayed, a system that estimates the current position of objects included in slightly outdated tracks can be established.
If object classification is available, incorporating it into the association algorithm and the distance calculation would likely lead to improved results. Yet the association algorithm in its current state can not process qualitative data like that in a meaningful way.

Overall cooperative perception can be a promising extension of ADAS. This work created fundamentals of a working system that incorporates information gathered via communication with other vehicles into the local perception and points out targets for upcoming work.

Bibliography

[1] *Specification of Cooperative Awareness Basic Service*, ETSI, March 2011, v1.2.1:TS 102 637-2.

[2] Forschungsinitiative Ko-FAS. Ko-PER - Kooperative Perzeption. [Online]. Available: http://ko-fas.de/41-0-Ko-PER---Kooperative-Perzeption.html

[3] A. Houenou, P. Bonnifait, V. Cherfaoui, and J. Boissou, "A Track-To-Track Association Method for Automotive Perception Systems," in *2012 IEEE Intelligent Vehicles Symposium*, June 2012, pp. 704–710.

[4] B. Duraisamy, T. Schwarz, and C. Wöhler, "On Track-to-Track Data Association for Automotive Sensor Fusion," in *2015 18th International Conference on Information Fusion (Fusion)*, July 2015, pp. 1213–1222.

[5] O. Sawade, B. Schäufele, J. Buttgereit, and I. Radusch, "A Cooperative Active Blind Spot Assistant as Example for Next-gen Cooperative Driver Assistance Systems (CoDAS)," in *2014 IEEE Intelligent Vehicles Symposium Proceedings*, June 2014, pp. 76–81.

[6] F. Seeliger and K. Dietmayer, "Inter-Vehicle Information-Fusion with Shared Perception Information," in *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, Oct 2014, pp. 2087–2093.

[7] H. Chen and Y. Bar-Shalom, "Track Fusion with Legacy Track Sources," in *2006 9th International Conference on Information Fusion*, July 2006, pp. 1–8.

[8] F. Seelinger, "Fahrzeugübergreifende Informationsfusion für ein Kreuzungsassistenzsystem," Ph.D. dissertation, University of Ulm, 2017.

[9] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "ROS: an open-source Robot Operating System," in *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*, Kobe, Japan, May 2009.