2012 Intelligent Vehicles Symposium
Alcalá de Henares, Spain, June 3-7, 2012

A Track-To-Track Association Method for Automotive Perception Systems

Adam Houenou*+, Philippe Bonnifait*, Veronique Cherfaoui*, Jean-François Boissou+

***Abstract*—Recent and future driver assistance systems use more and more sensors, that have individual tracking modules. For target tracking, it becomes necessary to find techniques to manage as simply as possible the use of a great number of independent and heterogeneous sensors, at the different stages of the process. This paper presents a modular high-level track-fusion architecture for a multisensor environment. This architecture allows the variation of the number and the types of the used sensors with no major change in the tracking algorithm. The paper also tackles the multisensor track-to-track association issue with a new algorithm based on a particular track-to-track distance computation. An example of target tracking method is shown to make use of the proposed architecture and the track-to-track association algorithm.**

***Index Terms*—Multisensor track-to-track association, track level fusion, target tracking, processing architecture.**

I. Introduction

Target tracking is a basic task for a number of Advanced Driver Assistance Systems (ADAS). It allows the system to get a representation of the vehicle's environment, containing potential obstacles with their kinematic parameters. This dynamic representation can then be used by different decision modules, depending on the considered functionality. Recent ADAS use more and more sensors (Fig 1) in order to increase confidence in detections, to enlarge their field of view (f.o.v.) and to get more complementary information. Many sensors are now equipped with an individual processing module that performs a local tracking task and reports a list of targets identified through time. Target tracking for a system that uses such sensors consists in fusing the received sensor-tracks in order to get upper level tracks, called *system-tracks* in this paper. It is a track-level data fusion. A discussion of the advantages and disadvantages of this architecture is presented in chapter 9 of [5]. The sensors work independently and the high level of abstraction of the data they provide allow the implementation of a tracking process that is not affected by the use of heterogeneous sensors.

In a monosensor target tracking process, the *data association* step is usually the first one; It consists in matching the new reported observations to the current tracks. In a multisensor environment, one must match the sensor reports that correspond to the same target in order to estimate the target's state by merging the different sensor reports. There

*Heudiasyc UMR CNRS 7253 - Université de Technologie de Compiègne, *adam.houenou@hds.utc.fr*
+PSA Peugeot Citroën

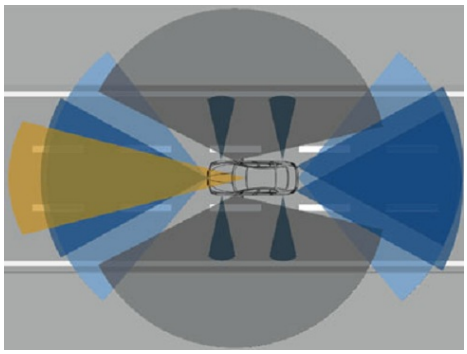

Figure 1. Vehicles are equipped with more and more sensors

are many data association algorithms such as *Nearest Neighbor* (NN), *Probabilistic Data Association*, *Multi-Hypothesis Tracking* (MHT) and their variants (see [5]) but most of them are only known for the two-data-classe (current tracks and reported observations) association cases. The cases with more data classes considered simultaneously are less frequently encountered. Data association errors can generate a number of problems such as ghost track, target loss and track ID permutation. Errors usually occur when the targets are close to each other (causing ambiguities) or because of sensors' lack of accuracy.

This paper aims at presenting a track-to-track association algorithm that can be used for as many sensors as wanted. This algorithm is used in a multisensor tracking system presented thereafter. The processing architecture that has been used makes it possible to vary the number and the types of the sensors with little modification in the structure of the tracking system. In section II, we present the tracking architecture and the internal tasks of the tracking system. In section III, we discuss the data synchronization issue in a multisensor environment. In section IV, the data association algorithm is described in details and in section V, we show some simulation results.

II. Global architecture of the tracking system

This system is intended to be used in vehicles that may have different types and different number of sensor devices (lidars, radars, cameras). So it has been designed to be as sensor-independent as possible. The global processing architecture (Fig. 2) is inspired from the one used in [6] for the vehicle that won the 2007 Darpa Urban Challenge. It separates the tasks into two main layers, a *Sensor Layer* (SL) and a *Fusion Layer* (FL). The common idea is to

978-1-4673-2118-1/$31.00 ©2012 IEEE

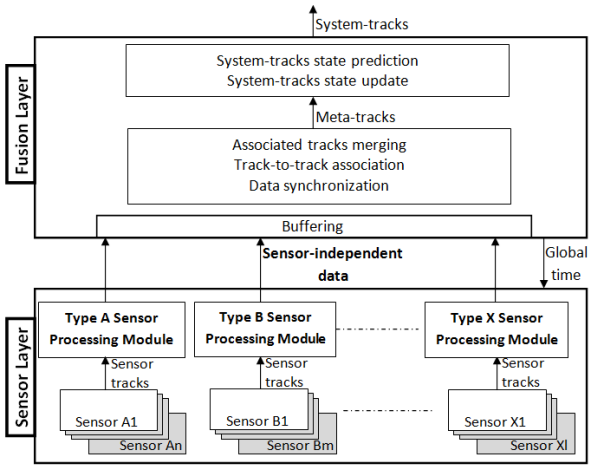

Figure 2. Processing architecture

perform the sensor-specific tasks in the SL and to provide the FL with sensor-independent data so that there would be no modification in the tracking algorithm even if the number and the types of the sensors were modified (because of sensor failure, addition or removal). In our case, there are less interactions between the layers and the internal tasks of the layers are different.

The SL has the role of collecting sensor-specific data from each physical sensor and converting them into a common sensor-independent data type. These data are then sent to the FL that performs the target tracking.

A. Sensor Layer

The perception of the environment is done by means of several sensors that work independently and provide their reports at different rates. Since each sensor has its own internal tracking module the reports are lists of target tracks identified through time. We assume that a track is defined by a timestamp, an identification number, a target state estimate and its error covariance matrix. The target state is composed of different parameters depending on the sensor specifications. In this paper, we will focus only on the tracking of the position and velocity which are usually the minimal reported parameters. A sensor report is relative to the sensor's individual mobile coordinate system and time base. In order to make a global tracking on the reunion of the individual f.o.v., it is necessary to convert the collected sensor data to a common mobile coordinate system and a common time base. These conversions are respectively the *spatial alignment* and the *temporal alignment*. Sensors that have the same data types are managed by the same *Sensor Processing Module* that can be run in parallel instances as new sensor reports are being received (see Fig2). If a sensor with a new data type is added, it will be necessary to add a new Sensor Processing Module. The Sensor Processing Modules perform the following tasks:

- *Feature extraction:* Sensors reports are usually received in many packages that first need be reassembled in order to form utilizable data structure. The reassembling rule is specific to each sensor. One must check the sensor's documentation.
- *Spatial alignment:* For automotive applications, it is generally sufficient to have a 2D positioning since one can assume that the road is locally plane all around the vehicle and that there is no flying or underground object. Therfore, the sensors' measures are supposed to be made in a plane considered to be the road plane. The common coordinate system is also defined in that plane. A rigid planar transformation can then be used to convert measures from an individual sensor coordinate system to the common coordinate system. Indeed, the relative position of the common coordinate system's origin and orientation in each sensor coordinate system is supposed to be known (sensor configuration and physical mounting point).
- *Temporal alignment:* The timestamp given to a report by a given sensor originates from the sensor's internal clock. In practice, the sensor clocks have different times (due to unknown sensor booting time) but it is possible to estimate the relative offset between them by using the global time (coming from the FL's clock). The individual timestamps can then be converted into global time. Because of clock drifts, the temporal alignment is periodically repeated. .
- *Data validation:* Invalid data are not transmitted to the FL. The validity can be checked by known sensor properties (validity flag or check-sum value).
- *Sensor meta-data:* In complement of the sensors estimates, other information are useful at the FL level. For instance, the identification of the sensors is used for data association and data synchronization, the f.o.v. are used to check overlapping areas for data association.

Here is an example of sensor-independent data for a given track: *{Timestamp, Sensor meta-data, Track ID, State estimate (position and velocity), Estimation error covariance matrix}*.

B. Fusion Layer

The target tracking task itself is performed in the FL. The sensor tracks are asynchronously received from the SL and fused to form system tracks. The fusion process works as follows: First, a software synchronization of the received data is implemented and a fusion rate is defined so that at each fusion time, the system has a report from each sensor. We discuss this choice in section III. The fusion operation consists in associating the reported tracks so that each cluster is composed of tracks originating from the same target. For that, we use a novel Track-To-Track Association (TTTA) algorithm. Often, data association algorithms are used to match track state predictions with sensor observations. In [1] a multisensor TTTA technique based on the computation of degrees of membership was described but the number of clusters (equivalent to the number of targets) has to be known in advance. The TTTA algorithm here, is based on a specific Track-To-Track Distance (TTTD) computation,

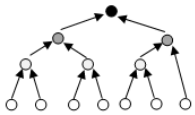

Figure 3. Track merging architecture.

using buffered past track states. We present it in details in section IV. After the TTTA stage, each cluster stands for one detected target. Depending on the overlapping of the f.o.v. of the sensors, and on their detections, there is one or more reported tracks per cluster. The tracks in each cluster are simultaneously merged to form a *meta-track*. The merging is done by forming pairs of tracks that are hierarchically merged as explained in Figure 3. Two tracks (X_1, P_1) and (X_2, P_2) are merged using equations (1) and (2).

$$X = P_2 \cdot (P_1 + P_2)^{-1} \cdot X_1 + P_1 \cdot (P_1 + P_2)^{-1} \cdot X_2 \quad (1)$$

$$P = P_2 \cdot (P_1 + P_2)^{-1} P_1 \quad (2)$$

This formula (derived from the Kalman filter gain in [10]) is optimal at minimizing the trace of the resulting covariance, under the assumption of uncorrelated measurement noise. Since it is also a commutative and associative fusion operator, fusion can be done in any order. As explained by Bar-Shalom in [2], there may be an inter-sensor correlation due to the temporal correlation of the filters. In our problem, it is difficult to determine this inter-sensor correlation because in practice the characteristics of the filtering performed by each sensor is generally unknown. In [4] a formula was proposed to approximate it, but only in the case of Kalman filters.

The meta-tracks are then associated to the current system-tracks state predictions using the Global Nearest Neighbor algorithm [8]. The prediction is done by using a linear and constant velocity prediction model (described in [3]). The non-associated meta-tracks are used to initialize new system-tracks. The non-associated system-tracks are predicted at each iteration until they match with a meta-track or until they are deleted after a lap of time where the correspondent target is supposed to be definitely lost. The system-tracks that match with meta-tracks are updated by the meta-track's state with no extra filtering.

III. Data synchronization and Buffer management

In order to perform a correct data fusion, it is crucial not only to keep coherence on the data chronology but also to fuse data only if they are relative to the exact same instant. Incoherence on data chronology is usually caused by sensors' latency; indeed, because of internal sensor processing time and data transfer time, it may happen that a data is received by the system after others that are more recent yet. This case is called *out-of-sequence*. Many solutions have been suggested to answer the out-of-sequence issue ([5], [7], [11]). Most of them recommend the use of buffers. The one suggested in [11] consists in fusing the data as they are

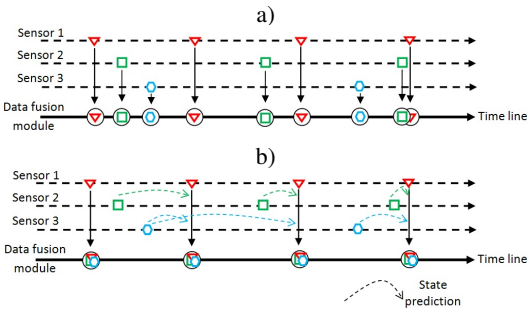

Figure 4. Data synchronization. a) Asynchronous fusion. b) Software synchronization

being received (regardless of the fusion technique used) and keeping them in a buffer for a limited lap of time. When an out-of-sequence case is detected, the buffer is used to go back into the past and redo the fusion, with the right chronology, up to present time.

The fusion of many data at a given instant is non sense if these data are not relative to the same time. Thus, in a multisensor environment where data are received at different rates, there are two possibilities: either an update of the system-tracks is performed every time a new sensor data is received (Fig 4.a) either, a fusion rate is defined, then, at each fusion time an estimate of each sensor's data is computed and a system update is performed with all the estimates (Fig 4.b). In the first case, the quality of the update (amount of information gained) depends on the accuracy and the type of information provided by the sensor whose data has just been used. To explain this, let's say that one of the sensors used does not make an estimate of the lateral velocity of targets; after the fusion of its reports, there won't be an update of the tracks lateral velocity. We opted for the second case, where we have steadier updates at steady time. Our solution works as follows: each sensor has a dedicated *Sensor Buffer* where only its last report is saved, meaning that a new report overwrites the previous one. The fusion module works at the same rate as the fastest sensor, meaning that a fusion task is performed every time a report from this sensor is received. When the fastest sensor sends a new report, a state prediction is made for the other sensors' tracks, using their (buffered) last report. The prediction is performed assuming that the targets have a linear trajectory with a constant velocity between two sensor reports. So, we obtain an apparent synchronization of the sensors. The state predictions and the fastest sensor report are saved in a *History Buffer* at each fusion time. When an out-of-sequence report is detected, the *History Buffer* is used to insert that report at "the right moment in the past". Then successive fusion operations are made to catch up the present time, using the following saved reports in the buffer. In order not to use an excessive amount of memory, the *History Buffer*'s size is limited.

IV. Track-To-Track Association

TTTA consists in clustering all the currents sensor tracks so that the tracks of a same cluster should be state estimates

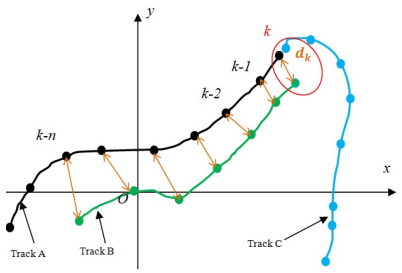

Figure 5. The use of track history for a better TTTA

of the same target. For our multisensor TTTA algorithm, two important and logic assumptions have to be made:

1) in each cluster there is at most one track of each sensor
2) a given track can not be in two different clusters.

This means that we assume a sensor won't generate more than one report per target and that two or more targets won't be reported as a single one; which is the ideal case. In practice, that may not be true, due to filtering errors but this algorithm will precisely allow the detection of ghost reports. Indeed, in areas covered by more than one sensor, the ghost reports will be isolated in their cluster. The algorithm is a generalization of the Nearest-Neighbor algorithm to more than two data classes. However it is based on computations of two-by-two distances between tracks.

A. Track-To-Track Distance computation

A track corresponds to the state estimate of a single target identified through time. We can then introduce the notion of *track history*. Figure 5 is the representation of three tracks from instant $k-n$ to instant k. The state estimate here is limited to the Cartesian position (x, y) (for the sake of representation). At instant k Track C is closer to Track A than Track B. But considering all the known history (past states) of the three tracks, it is clear that Track B is more similar to Track A than Track C. The idea is to use the track histories in order to make better association. We use a similar formula (Eq 3) as the one shown in [9] to compute the "distance" between two tracks, using their past estimates.

$$D_k^{(a,b)} = \frac{1}{n}\sum_{i=0}^{n-1} d_{k-i}^{(a,b)} \tag{3}$$

with n being the history size (number of past reports considered) and

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

where a and b are two tracks and X_k and P_k are respectively the state estimate and the error covariance matrix of the corresponding state estimate at instant k. The value assigned to n is equivalent to a lap of time that must not be greater than the more recent track's age.

With this formula, the resulting distance between Track C and Track A will be bigger than the distance between Track B and Track A and the association error will be avoided at instant k.

Algorithm 1 Multisensor Track-To-Track Association

1) Collect the tracks of all the sensors
2) Assign a number from 1 to N to each track, N being the total number of tracks.
3) Create a $N \times N$ array for the TTTDs between the tracks
   a) Set cells over the diagonal to a defined maximal value ($MaxVal$) in order not to compute twice the distance between two same tracks.
   b) Set cells corresponding to two tracks of the same sensor to $MaxVal$, in order not to associate two tracks of a same sensor.
   c) Set the remaining cells to the distance between the corresponding two tracks.
   d) Set cells where the distance is greater than a defined threshold to $MaxVal$. The threshold symbolizes a gate out of which we assume that the two tracks cannot originate from the same target.
4) Loop: Determine the minimal value ($MinVal$) of the array and its position (lin, col) in the array. While $MinVal$ is smaller than $MaxVal$
   a) If none of the corresponding two tracks has not been inserted in a cluster yet, then put both of them in a new cluster.
   b) If only one has already been inserted in a cluster, then add the second to that cluster.
   c) If both have already been inserted in a cluster (the same or not), then do nothing.
   d) Set to $MaxVal$ cells in the line lin and the ones in the column col, that correspond to tracks reported by the two concerned sensors.
5) Each track that has not been inserted in a cluster forms a new cluster (singletons).

B. Multisensor Track-To-Track Association algorithm

The multisensor TTTA process is described in algorithm 1.

Iteratively finding the minimal distance guaranties that every new association made is the best possible at each iteration and that the first associations are better than the followings. That is why, at stage 4, if both tracks already have a cluster, nothing is changed. The value given to the threshold (stage 3.e) depends on the sensors' accuracy. When the sensors are accurate, their reports are close to each other because they are close to the actual target state. In that case, the threshold can be defined as small as possible. If they are inaccurate, reports about a same target may be far from each other, so the threshold needs to be enlarge. The size of the history in the Track-To-Track Distance (TTTD) formula (Eq. 3) is the second parameter of the algorithm. When it is defined big, there are less chances for association errors because more past track states are taken into account but the computational requirement becomes bigger.

If at stage 4 there is a multiple minimum value (which should be rare), each hypothesis, in accordance with the first considered cell, must be evaluated; then if the results are

	Setting 1			Setting 2			Setting 3			Setting 4		
	S1	S2	S3	S1	S2	S3	S1	S2	S3	S1	S2	S3
x	2	6	2	2	15	13	2	25	25	10	15	2
y	2	5	3	2	15	16	2	23	24	10	10	2
v_x	3	4	5	3	10	12	3	19	21	2	8	8
v_y	3	4	5	3	10	10	3	19	21	2	8	8

Table I
SENSOR ACCURACY SETTINGS USED DURING THE SIMULATION

different, the chosen hypothesis is the one where the sum of the TTTD of tracks in the same clusters is the smallest.

An example of execution of the algorithm is shown in Appendix.

V. SIMULATION RESULTS

In order to evaluate the performance of this tracking method in various situations, we designed a simulator. Real automotive scenarios can be simulated and we obtain the position and the velocity of the vehicles (targets and host) during the simulation, with a chosen sample rate, in a fixed global Cartesian coordinate system. These are considered as reference data. Sensor measurements are also elaborated. It is possible to simulate several kinds of sensors by adding to the reference data a uniform white noise with a variable amplitude w.r.t. the sensor's basic accuracy and the target's range. Target visibility is simulated by checking sensors' f.o.v. and alignment of objects w.r.t. the sensor's position. Then, the tracking module of each sensor is simulated, by performing a Kalman filter with a linear and constant velocity prediction model (see [3]). A virtual sensor's report at time t is a timestamped list of its tracks. For simplification purpose, the global tracking is done in a fixed Cartesian coordinate system.

We created an urban driving context with four target cars, containing straight road parts, a roundabout and an intersection. The host vehicle is equipped with three sensors, $S1$, $S2$ and $S3$, all in front view sensing. The sensors' f.o.v. are made quite redundant in order to evaluate the performance of the TTTA algorithm in various situations and to compare the global tracking to the individual sensor trackings. The simulation has been run several times with different sensor accuracies as reported in table I. The values in the table are the maximal measurement error rates, expressed in percentage, for each state parameter, for a target at 100m range. When the target is closer, the maximal measurement error rate is smaller. The actual measurement error is a random value in the correspondent interval. One can see that from setting 1 to setting 3 sensor $S1$ does not change while sensors $S2$ and $S3$ are more and more degraded. These settings are used to evaluate the TTTA algorithm. Setting 4 is used to evaluate the result of the global tracking.

A. TTTA Algorithm

The algorithm has two parameters that need to be tuned: the gating threshold (see stage 3.e of the algorithm) and the size of the history considered for the TTTD computation (see eq. 3). We study here the impact of each parameter on the result in terms of erroneous associations. The simulation has been run 5 times per sensor setting per parameter and per chosen parameter value. For a complete execution of the scenario, the TTTA algorithm is run about 400 times (at each sample time). The association errors were of two types:

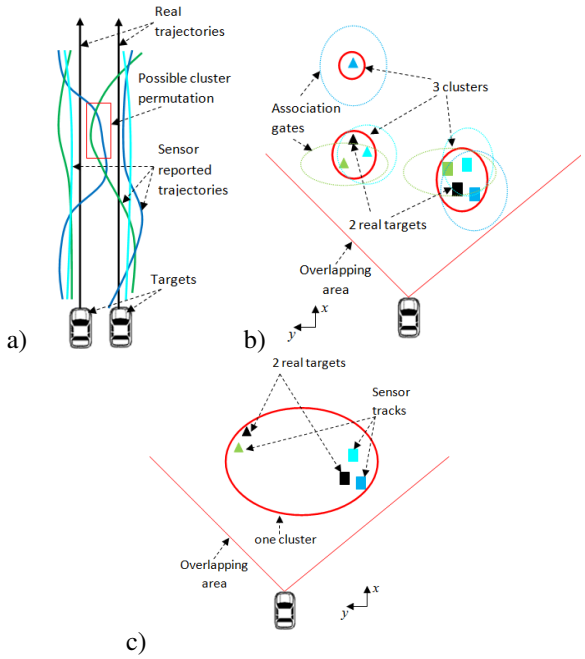

Figure 6. Association errors. Targets are represented in black and sensor reports in color. a) Cluster permutation b) More clusters than targets c) Less clusters than targets

1) Some tracks permuted their clusters: This can happen when the correspondent targets evolve very closely from each other for a lap of time greater than the defined history size (see Fig. 6a). The advantage of TTTD is to help distinguishing between tracks that are close only for a moment and the ones that have always been quite close because they probably originate from the same target. If the track history is not long enough, target that move closely for a long time may be mistaken due to measurement noise. A solution is to enlarge the history size but a trade-off has to be made between this issue and the computational requirement. Indeed adding one unit to the history size (equivalent to the simulation period) increases the execution time by approximately 1% in our case (three sensors and four targets). It gets worse if the numbers of sensors and targets are greater.
2) There were more clusters than actual targets: it happens when a sensor reports ghost target or when we have erroneous target state estimates that fall out of the association gate (see Fig. 6b). The solution is to enlarge the threshold but that increases the computational requirement and can lead to associating tracks that are not similar at all.

A third possible type of errors, that did not occur during the tests is to have less clusters than real targets. If the gating threshold is defined too large and in case of miss detections, reports originating from 2 different targets may be clustered

a)

	Threshold value				
	10	12	15	20	30
Setting 1	0.8	0	0	0	0
Setting 2	83.3	31.2	1	0	0
Setting 3	87.1	42.3	5	4.2	0

b)

	History size		
	5	10	15
Setting 1	0.8	0	0
Setting 2	83.3	31.2	0
Setting 3	87.1	42.3	0

Table II
PERCENTAGE OF ERRONEOUS ASSOCIATIONS W.R.T. SENSOR ACCURACIES AND A) GATING THRESHOLD B)HISTORY SIZE

together (see Fig. 6c). However, this case should be rare because it needs each sensor to have a miss detection; indeed if at least one does not miss a detection, the number of cluster will be at least equal to the number of targets.

Table II.a reports the percentage of erroneous results w.r.t. the threshold value. The history size is fixed to 10 here. One can notice that the more inaccurate sensors are (as shown by settings 2 and 3) the larger the threshold should be chosen to have a good result. An expression of the relationship between the sensor accuracies and the required threshold cannot be clearly defined because the statistical distribution followed by the TTTD (Eq. 3) is not known. The right threshold should be chosen experimentally.

Table II.b reports the percentage of erroneous results as a function of the history size. The gating threshold is fixed to 30. One can notice that taking into account more past sensor reports helps to solve ambiguous situations caused by temporary closeness of some targets and leads to less erroneous associations.

Whatever the sensor accuracies, it is possible to parametrize the algorithm so that it performs well.

B. Fusion result

The efficiency of the global tracking has also been evaluated. Here we use setting 4 described in Table I, for the sensor accuracies. $S1$ is accurate for target velocity estimation, $S3$ for target localization and $S2$ is less accurate for both. The gating threshold is set to 30 and the the history size equals 10. Figure 7 shows the longitudinal position and speed errors for the tracking of a chosen target. The colored curves are the result for each sensor tracking and the black curve is for the global tracking. One can notice that for each state parameter the global tracking result is always close to the estimate provided by the best sensor even if it is not always better than it. This is because in the merging formula (equations 1 and 2), sensors with a large error covariance matrix have a low (but not null) weight. Nevertheless, globally speaking (for all state parameters), the global tracking is better than each sensor's tracking.

VI. CONCLUSION AND FUTURE WORK

In this article, track-level multisensor target tracking has been studied for automotive applications. We have proposed a processing architecture that is robust to the addition, removal or failure of sensors. The use of many sensors has a lot of advantages but one can take benefit on it only if correct associations between the reported sensor tracks is made. For this we have proposed a multisensor TTTA method that uses track histories. This approach has several advantages like

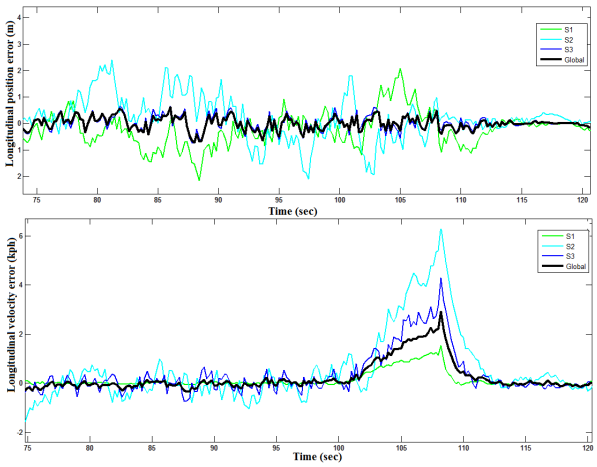

Figure 7. Global tracking errors versus sensor tracking errors

the ability to reduce ambiguities and to partially cover the lack of accuracy of sensors. We have also reminded a buffer management that solves the out-of-sequence and the data synchronization issues.

We focused only on the position and the velocity tracking but future work include other attributes such as object dimensions and classification. We are also working on global track existence probability management.

REFERENCES

[1] A. M. Aziz. Fuzzy track-to-track association and track fusion approach in distributed multisensor multitarget multiple-attribute environment. *Electrical Eng. Dpt., Military Technical College, Cairo, Egypt*, 2006.

[2] Y. Bar-Shalom. On the track-to-track correlation problem. *IEEE Trans. on Automatic Control*, 26:571 – 572, 1981.

[3] Y. Bar-Shalom, M. K. Kalandros, L. Trailovuc, and L. Y. Pao. Tutorial on multisensor management and fusion algorithms for target tracking. *Proc. of the 2004 American Control*, pages 4737–4751, 2004.

[4] Y. Bar-Shalom and X. R. Li. Multitarget multisensor tracking: principles and techniques. *IEEE AES Systems Magazine*, pages 41 – 44, Feb. 1996.

[5] S. S. Blackman and R. Popoli. *Design and analysis of modern tracking systems*. ArtechHouse, Incorporated, 1999.

[6] M. S. Darms, P. E. Rybski, C. Baker, and C. Urmson. Obstacle detection and tracking for the urban challenge. *IEEE Trans. on ITS*, 10, Sep 2009.

[7] N. Kaempchen and K. Dietmayer. Data synchronization strategies for multi-sensor fusion. *Proc. of the 10th World Congress on ITS and Services*, 2003.

[8] P. Konstantinova, A. Udvarev, and T. Semerdjiev. A study of target tracking algorithm using global nearest neighbor approach. *International conference on computer systems and technologies - CompSys Tech'*, 2003.

[9] D. Müller, J. Pauli, M. Meuter, L. Ghosh, and S. Müller-Schneiders. A generic video and radar data fusion system for improved target selection. *IEEE Intelligent Vehicles Symposiun, Germany*, Jun 2011.

[10] R. C. Smith and P. Cheeseman. On the representation and estimation of spatial uncertainty. *The International Journal of Robotics Research*, 5(4):56–68, 1986.

[11] C. Tessier. Système de localisation basé sur une strategie de perception cognitive appliqué a la navigation autonome d'un robot mobile. Master's thesis, Universite Blaise Pascal - Clermont II, 2007.

APPENDIX

Let's consider a host vehicle with four sensors, tracking four target vehicles in their overlapped f.o.v. Figure 8 depicts an example of sensor reports where T_{ij} are the positions

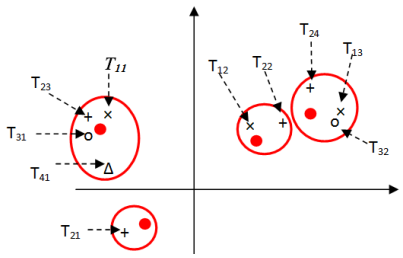

Figure 8. Example of TTTA. T_{ij} are sensor reported positions. Red dots are the actual positions of the targets. Red ellipses are correct track clusters

of a reported track, with i being the number of the sensor and j the number of the track. The red dots represent the actual positions of the targets and the red ellipses are clusters containing targets that originate from the same target. Hereafter we depict the evolution of the TTTD array and the obtained clusters from stage 3 to the end of the algorithm.

Stage 3

	T_{11}	T_{12}	T_{13}	T_{21}	T_{22}	T_{23}	T_{24}	T_{31}	T_{32}	T_{41}
T_{11}	∞	×	×	×	×	×	×	×	×	×
T_{12}	∞	∞	×	×	×	×	×	×	×	×
T_{13}	∞	∞	∞	×	×	×	×	×	×	×
T_{21}	13.5	20	22	∞	×	×	×	×	×	×
T_{22}	19	3	6	∞	∞	×	×	×	×	×
T_{23}	1	16	20.5	∞	∞	∞	×	×	×	×
T_{24}	20.5	7	2.5	∞	∞	∞	∞	×	×	×
T_{31}	2	15.5	20	12.5	16.5	1.5	17.5	∞	×	×
T_{32}	21	7.5	0.5	17	5	18.5	4.5	∞	∞	×
T_{41}	5.5	10	18	11	15	6.5	16	4	14	∞

	T_{11}	T_{12}	T_{13}	T_{21}	T_{22}	T_{23}	T_{24}	T_{31}	T_{32}	T_{41}
T_{11}	∞	×	×	×	×	×	×	×	×	×
T_{12}	∞	∞	×	×	×	×	×	×	×	×
T_{13}	∞	∞	∞	×	×	×	×	×	×	×
T_{21}	∞	∞	∞	∞	×	×	×	×	×	×
T_{22}	∞	3	6	∞	∞	×	×	×	×	×
T_{23}	1	∞	∞	∞	∞	∞	×	×	×	×
T_{24}	∞	7	2.5	∞	∞	∞	∞	×	×	×
T_{31}	2	∞	∞	∞	∞	1.5	∞	∞	×	×
T_{32}	∞	7.5	0.5	∞	5	∞	4.5	∞	∞	×
T_{41}	5.5	10	∞	∞	∞	6.5	∞	4	∞	∞

Stage 4: Loop

	T_{11}	T_{12}	T_{13}	T_{21}	T_{22}	T_{23}	T_{24}	T_{31}	T_{32}	T_{41}
T_{11}	∞	×	×	×	×	×	×	×	×	×
T_{12}	∞	∞	×	×	×	×	×	×	×	×
T_{13}	∞	∞	∞	×	×	×	×	×	×	×
T_{21}	∞	∞	∞	∞	×	×	×	×	×	×
T_{22}	∞	3	6	∞	∞	×	×	×	×	×
T_{23}	1	∞	∞	∞	∞	∞	×	×	×	×
T_{24}	∞	7	2.5	∞	∞	∞	∞	×	×	×
T_{31}	2	∞	∞	∞	∞	1.5	∞	∞	×	×
T_{32}	∞	7.5	0.5	∞	5	∞	4.5	∞	∞	×
T_{41}	5.5	10	∞	∞	∞	6.5	∞	4	∞	∞

Clusters
$\{T_{13}, T_{32}\}$

	T_{11}	T_{12}	T_{13}	T_{21}	T_{22}	T_{23}	T_{24}	T_{31}	T_{32}	T_{41}
T_{11}	∞	×	×	×	×	×	×	×	×	×
T_{12}	∞	∞	×	×	×	×	×	×	×	×
T_{13}	∞	∞	∞	×	×	×	×	×	×	×
T_{21}	∞	∞	∞	∞	×	×	×	×	×	×
T_{22}	∞	3	6	∞	∞	×	×	×	×	×
T_{23}	1	∞	∞	∞	∞	∞	×	×	×	×
T_{24}	∞	7	2.5	∞	∞	∞	∞	×	×	×
T_{31}	2	∞	∞	∞	∞	1.5	∞	∞	×	×
T_{32}	∞	∞	∞	∞	5	∞	4.5	∞	∞	×
T_{41}	5.5	10	∞	∞	∞	6.5	∞	4	∞	∞

Clusters
$\{T_{13}, T_{32}\}$
$\{T_{11}, T_{23}\}$

	T_{11}	T_{12}	T_{13}	T_{21}	T_{22}	T_{23}	T_{24}	T_{31}	T_{32}	T_{41}
T_{11}	∞	×	×	×	×	×	×	×	×	×
T_{12}	∞	∞	×	×	×	×	×	×	×	×
T_{13}	∞	∞	∞	×	×	×	×	×	×	×
T_{21}	∞	∞	∞	∞	×	×	×	×	×	×
T_{22}	∞	3	6	∞	∞	×	×	×	×	×
T_{23}	∞	∞	∞	∞	∞	∞	×	×	×	×
T_{24}	∞	7	2.5	∞	∞	∞	∞	×	×	×
T_{31}	2	∞	∞	∞	∞	1.5	∞	∞	×	×
T_{32}	∞	∞	∞	∞	5	∞	4.5	∞	∞	×
T_{41}	5.5	10	∞	∞	∞	6.5	∞	4	∞	∞

Clusters
$\{T_{13}, T_{32}\}$
$\{T_{11}, T_{23}, T_{31}\}$

	T_{11}	T_{12}	T_{13}	T_{21}	T_{22}	T_{23}	T_{24}	T_{31}	T_{32}	T_{41}
T_{11}	∞	×	×	×	×	×	×	×	×	×
T_{12}	∞	∞	×	×	×	×	×	×	×	×
T_{13}	∞	∞	∞	×	×	×	×	×	×	×
T_{21}	∞	∞	∞	∞	×	×	×	×	×	×
T_{22}	∞	3	6	∞	∞	×	×	×	×	×
T_{23}	∞	∞	∞	∞	∞	∞	×	×	×	×
T_{24}	∞	7	2.5	∞	∞	∞	∞	×	×	×
T_{31}	∞	∞	∞	∞	∞	∞	∞	∞	×	×
T_{32}	∞	∞	∞	∞	5	∞	4.5	∞	∞	×
T_{41}	5.5	10	∞	∞	∞	6.5	∞	4	∞	∞

Clusters
$\{T_{13}, T_{32}, T_{24}\}$
$\{T_{11}, T_{23}, T_{31}\}$

	T_{11}	T_{12}	T_{13}	T_{21}	T_{22}	T_{23}	T_{24}	T_{31}	T_{32}	T_{41}
T_{11}	∞	×	×	×	×	×	×	×	×	×
T_{12}	∞	∞	×	×	×	×	×	×	×	×
T_{13}	∞	∞	∞	×	×	×	×	×	×	×
T_{21}	∞	∞	∞	∞	×	×	×	×	×	×
T_{22}	∞	3	6	∞	∞	×	×	×	×	×
T_{23}	∞	∞	∞	∞	∞	∞	×	×	×	×
T_{24}	∞	7	2.5	∞	∞	∞	∞	×	×	×
T_{31}	∞	∞	∞	∞	∞	∞	∞	∞	×	×
T_{32}	∞	∞	∞	∞	5	∞	4.5	∞	∞	×
T_{41}	5.5	10	∞	∞	∞	6.5	∞	4	∞	∞

Clusters
$\{T_{13}, T_{32}, T_{24}\}$
$\{T_{11}, T_{23}, T_{31}\}$
$\{T_{12}, T_{22}\}$

	T_{11}	T_{12}	T_{13}	T_{21}	T_{22}	T_{23}	T_{24}	T_{31}	T_{32}	T_{41}
T_{11}	∞	×	×	×	×	×	×	×	×	×
T_{12}	∞	∞	×	×	×	×	×	×	×	×
T_{13}	∞	∞	∞	×	×	×	×	×	×	×
T_{21}	∞	∞	∞	∞	×	×	×	×	×	×
T_{22}	∞	∞	∞	∞	∞	×	×	×	×	×
T_{23}	∞	∞	∞	∞	∞	∞	×	×	×	×
T_{24}	∞	∞	∞	∞	∞	∞	∞	×	×	×
T_{31}	∞	∞	∞	∞	∞	∞	∞	∞	×	×
T_{32}	∞	∞	∞	∞	5	∞	4.5	∞	∞	×
T_{41}	5.5	10	∞	∞	∞	6.5	∞	4	∞	∞

Clusters
$\{T_{13}, T_{32}, T_{24}\}$
$\{T_{11}, T_{23}, T_{31}, T_{41}\}$
$\{T_{12}, T_{22}\}$

	T_{11}	T_{12}	T_{13}	T_{21}	T_{22}	T_{23}	T_{24}	T_{31}	T_{32}	T_{41}
T_{11}	∞	×	×	×	×	×	×	×	×	×
T_{12}	∞	∞	×	×	×	×	×	×	×	×
T_{13}	∞	∞	∞	×	×	×	×	×	×	×
T_{21}	∞	∞	∞	∞	×	×	×	×	×	×
T_{22}	∞	∞	∞	∞	∞	×	×	×	×	×
T_{23}	∞	∞	∞	∞	∞	∞	×	×	×	×
T_{24}	∞	∞	∞	∞	∞	∞	∞	×	×	×
T_{31}	∞	∞	∞	∞	∞	∞	∞	∞	×	×
T_{32}	∞	∞	∞	∞	∞	∞	∞	∞	∞	×
T_{41}	∞	∞	∞	∞	∞	6.5	∞	∞	∞	∞

Clusters
$\{T_{13}, T_{32}, T_{24}\}$
$\{T_{11}, T_{23}, T_{31}, T_{41}\}$
$\{T_{12}, T_{22}\}$

Stage 5

Clusters: $\{T_{13}, T_{32}, T_{24}\}$, $\{T_{11}, T_{23}, T_{31}, T_{41}\}$, $\{T_{12}, T_{22}\}$, $\{T_{21}\}$

This is the expected result.