Sensor Data Fusion for Multiple Configurations

Fredrik Sandblom and Joakim Sörstedt

Abstract—This paper presents a fusion architecture designed for vehicle manufacturers that use multiple sensor systems to realize several active safety applications, ranging from standard systems to autonomous systems. Advantages and disadvantages of design choices are discussed and the methods we have chosen to implement in two demonstrator vehicles are used as examples. The main contributions are the separation of the architecture into three categories, the state vector parametrization, and the use of multiple output-lists. Together, these choices make it possible to re-use verified core functionality while filters can be independently tuned to meet application-specific requirements.

I. INTRODUCTION

The objective of a sensor data fusion system is to combine information from multiple sources, e.g., sensor data or prior knowledge, to produce an accurate representation of some physical quantity that is of interest [1], [2]. Here, focus is on a fusion system to support Advanced Driver Assistance Systems (ADAS). In such systems, sensors are used to monitor the environment surrounding the ego vehicle, and the system uses this information to decide on an appropriate action such as issue a warning or braking intervention. The task of the fusion system is to describe the surrounding environment with a sufficient level of detail and accuracy to meet the requirements of the applications. The problems at hand are in many ways similar to those in an object tracking system [3], but our objective is not to design a system which is optimal in that its estimates meets any particular optimality criteria, such as the minimum mean square error criteria. Rather, the focus is to design a system flexible [4] towards new sensor configurations and new estimation components, while meeting the requirements from various ADAS applications.

Today, most ADAS applications on the market can be supported using one or a couple of sensors since they all require approximately the same description of the surrounding environment. The number of sensor configurations is therefore relatively small. This will change in the future for two reasons. First, the vehicle manufacturers are developing more complicated systems, ranging from avoiding accidents in intersections to completely autonomous vehicles. These systems require a much more detailed description of the surroundings than the current sensor systems provide. Second, while only the top-performing vehicles will be equipped with sensors to enable such advanced functionality, the vast majority of all vehicles will be equipped with basic ADAS applications, such as automatic emergency braking systems.

This development is driven by legislation [5] and ratings from different New Car Assessment Programs (NCAP), e.g., [6], [7], [8]. Consequently, different sensor set-ups within the same vehicle program must be supported. For each of these sensor configurations it is desirable to re-use as much of the implementation as possible, so that it does not have to be verified multiple times.

This article is organized as follows: A problem formulation is given in Section II and the outline of the proposed architecture is described in Section III. Specific details of the architecture are provided in Sections IV–VI together with suitable design choices. Finally, conclusions are summarized in Section VII.

II. PROBLEM FORMULATION

We consider a fusion system that uses $I$ sensor signals, $z^i$, valid at times $t_{z^i}$ (for $i = 1, \ldots, I$) respectively, to produce a set of $J$ object lists $S_k = \{S_k^1, \ldots, S_k^J\}$ as shown in Fig. 1. The integer $k$ denotes the output time, $t_k$, where $t_k = kT$ and $T$ is the constant time between each fusion cycle.

Each object list $S_k^j$ contains a number of different objects, each with the following information: $\{x^j, P_x^j, \theta^j, P_\theta^j\}$, where $x$ is the mean value of the dynamic state of the object, $P_x$ the corresponding covariance matrix, $\theta$ is an attribute vector containing static properties of an object such as type and $P_\theta$ is the associated covariance matrix to $\theta$. The objective is to at times $t_k$, $t_{k+1}, \ldots$ compute:

$$S_k = f \left( z^1, \ldots, z^I, \ldots, z^{k-1}, \xi_{t_{z^j}=t_{z^i}}, t_k \right),$$

where $\xi_{t_{z^j}=t_{z^i}}$ denotes the ego vehicle trajectory traveled during the most recent $t_k - t_{z^i}$ seconds, i.e., since $z^i$ was valid. The content of $z^i$ can range from uncorrelated measurements, $y^i$, to a filtered state estimate, $x^i$, with corresponding covariance matrix, $P_x^i$.

![Fig. 1. The sensor data, $z^i$, can describe, e.g., other road users, lane markings, and electronic map data.](image)

The question at hand is how to design the fusion system such that components are (a) re-usable, (b) verifiable, and (c) high-performing. The scope is limited to designing a system supporting current state of the art ADAS using sensor technologies available on the market.
III. PROPOSED ARCHITECTURE

The proposed architecture is presented in Fig. 2. Noticeable design features include the partitioning of operations into services, based on how frequently the operation must be executed, and the decision to use multiple output object lists, each created by a unique path through the fusion system. On a high level, the fusion system can be described as providing three types of services:

1) Time alignment: to compensate for the individual time-delays from several asynchronous sensors and the motion of both the targets and the ego vehicle during these delays.
2) Object state fusion: the combination of several single-sensor estimates or measurements into a single estimate.
3) Virtual sensors: the creation of specific estimates that either are not available from a single sensor or can not be measured at a reasonable cost.

These three services are discussed in Sections IV–VI. The remainder of this section is directed towards explaining requirements which influence the architecture.

First, it is important to understand why multiple sensors are added to a vehicle in the first place. Three important reasons are:

1) To extend the joint field-of-view.
2) To independently confirm target existence, e.g., to meet functional safety [9] requirements.
3) To realize applications that demand estimates which a single sensor cannot provide.

It is clear that the first two reasons do not necessarily require more than a basic fusion scheme. The challenge is to facilitate using components required to realize more advanced functionality — the third reason — in the same system that hosts also standard functions.

Second, as a consequence of the last two reasons, sensors are typically chosen to complement each other as much as possible. This means that their measurement principles are independent of each other and that they provide accurate estimates related to different state vector elements, e.g., radial distance and velocity from a radar and lateral distance and object classification from a camera.

Third, for a vehicle manufacturer, the ability to change sensor configuration as new technologies are available is a key feature of the fusion system. Maintaining a high level interface, i.e., state vector parameterization, facilitates such changes and the system is therefore designed with the assumption that at least one sensor is providing filtered tracks using that interface.

IV. TIME ALIGNMENT

In its simplest form, the sensor fusion system only consists of time alignment, i.e., it predicts delayed and asynchronously sampled state vectors from the sensors and presents them in a common reference frame. In order to do so, the following needs to be determined:

Parameterization: how should the information in to and out of the sensor fusion system be represented?
Coordinate system: which reference frame should be used for internal representation and in the output interface? Coordinate systems used today include local or global origin and Cartesian, polar or road aligned coordinates.
Alignment procedure: using prediction models to time align object tracks.

These design choices partly depend on each other, i.e., the choice of parameterization influences which coordinate system that is appropriate, which in turn affects the alignment methods. The design will be a trade off between the application requirements and the sensing principle.
1) **Parameterization:** For dynamic objects such as vehicles and pedestrians, two fundamentally different types of accident scenarios is targeted, involving turning and non-turning objects. Non-turning scenarios such as rear-end collisions can be predicted with sufficient accuracy using the constant acceleration (CA) model [10]. The CA state vector is:

\[
x_{ct}^a = [x_t \ y_t \ \dot{x}_t \ \dot{y}_t] \quad (2),
\]

where \((x, y)\) are Cartesian coordinates and \(t\) is the time for which \(x_{ct}^a\) is computed.

Turning scenarios are more complicated in that they require a different parameterization than the non-turning scenarios to be accurately predicted. The coordinate turn constant acceleration (CT) model [10] describes such maneuvers and is parameterized:

\[
x_{ct}^t = [x_t \ y_t \ v_t \ \psi_t \ a_t \ c_t] \quad (3),
\]

where \(v_t\) and \(a_t\) are the absolute velocity and acceleration magnitude in the direction of \(\psi_t\), and \(c_t\) is the local curvature. It is shown in [11] that this parameterization (3) can be used to predict and avoid accidents involving turning vehicles.

The predictive models for both parameterizations (2)–(3) are included in Appendix A for the convenience of the reader.

Not all sensors can supply estimates of vehicle states in the CT parameterization but, on the contrary, most automotive sensors have embedded trackers that can estimate the CA states, or a subset of them. Since there exists bidirectional transformations between the CA (2) and CT (3) parameterizations (see appendix B), we propose that the sensor fusion interface should contain the union of the two:

\[
x_t = [x_t \ y_t \ \dot{x}_t \ \dot{y}_t \ \psi_t \ v_t \ a_t \ c_t] \quad (4).
\]

If a sensor provides a CA interface to the fusion system, the equations in appendix B are used to compute the missing CT states, and vice versa. As a result, the applications always receive the information needed even if it is not produced by the sensor, and the fusion system can ensure that the transformed estimates are consistent over time. It has been shown in [12] how CA estimates can be transformed and filtered using the CT dynamic model with only minor delays in the transformed data. The CA and CT parameterizations are illustrated in Fig. 3.

![Fig. 3. The proposed state representation of a target vehicle, see Eq. (4), in a (local) coordinate system centered in the host vehicle.](image)

Static objects fixed to the ground, such as lane markings, road edges, barriers or drivable areas are better described by a state vector comprising position and extension, although for certain objects, e.g. traffic signs, a point in either local coordinates \((x, y)\) or global coordinates \((G_x, G_y)\) is sufficient. More detailed shape parameters are needed for larger objects such as the road. A simple solution is to represent these objects using a list of coordinates, but often parametric equations such as the clothoid model [13] are used instead.

As we will see in the next two sections, it is generally wise to choose a parameterization for which the described curve or shape is invariant to affine transformations, for example B-splines [14].

2) **Coordinate system:** It is desirable to describe the surrounding environment in as few different coordinate systems as possible. Mainly because implemented routines such as the alignment methods can be reused for different sensor setups, and partly because it simplifies the interpretation of the output interface from the sensor fusion system.

The local Cartesian coordinate system shown in Fig. 3 is used for dynamic objects. Most radar and camera sensor suppliers support the interface and it is suitable for the proposed parameterization (4). The origin is placed at the point around which the ego vehicle is rotating. The center of the rear axle is a sufficiently good approximation in most normal driving scenarios for a car.

There is a strong argument to use a global coordinate system rather than a local one when considering stationary objects, since no state prediction is required when time alignment of sensor data is performed. However, since most sensors provide data in local coordinates, which also is preferred by most applications, we suggest using the same local Cartesian coordinate system for both stationary and dynamic objects. It should be pointed out though that this choice might change if new sensor technologies are used, e.g. vehicle-to-vehicle communication sensors which provide measurements in global coordinates.

3) **Alignment procedure:** To fuse information from asynchronous sensors with different delays, the object tracks must first be expressed in a common reference frame. For example, let \(\dot{x}_t\) be an object track expressed in the coordinate system \((\bar{x}, \bar{y})\) at time \(t\) and let \(\bar{P}_{x_t}\) be the covariance matrix associated to \(\dot{x}_t\). The task is to express this track and the corresponding covariance matrix in the coordinate system \((x, y)\), which is situated at the ego vehicle rear axle at a later time \(t + \Delta t\), shown in Fig. 4. During this time, the coordinate system has moved and is pointing in a different direction, which can be expressed as an affine transformation that maps \(\dot{x}_t\) and \(\bar{P}_{x_t}\) to \(x_t\) and \(P_{x_t}\):

\[
x_t = R\dot{x}_t + T \quad (5)
\]

\[
P_{x_t} = R\bar{P}_{x_t}R^T, \quad (6)
\]

![Fig. 4. Time alignment example: The ego vehicle motion results in a coordinate transformation and the target motion, indicated by the dashed line connecting the covariance contours, requires a dynamic state prediction.](image)
where $\mathbf{R}$ is a rotation matrix and $\mathbf{T}$ a translation vector. After compensating for the movement of the coordinate system, a prediction model is used to calculate how the object has moved during $\Delta t$:

$$x_{t+\Delta t} = g(x_t, v_t),$$

(7)

where $v_t$ is a Gaussian zero mean noise process with covariance matrix $\mathbf{Q}_v$. Both these steps are visualized in Fig. 4.

For the parameterization in (4) the coordinate transformation matrices in equations (5) – (6) are:

$$\mathbf{R} = \begin{bmatrix} R_{\Delta x} & 0 & 0 & 0 \\ 0 & R_{\Delta y} & 0 & 0 \\ 0 & 0 & R_{\Delta \psi} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{T} = \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta \psi \\ 0_{3 \times 1} \end{bmatrix},$$

(8)

where

$$0 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \mathbf{I} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},$$

(9)

and $R_{\Delta \psi}$ is a clockwise rotation matrix:

$$R_{\Delta \psi} = \begin{bmatrix} \cos(\Delta \psi) & \sin(\Delta \psi) \\ -\sin(\Delta \psi) & \cos(\Delta \psi) \end{bmatrix}. \quad (10)$$

The translation vector, $[\Delta x \ \Delta y]^T$, and the rotation angle, $\Delta \psi$, are calculated using the ego vehicle history, $\xi_{t:t+\Delta t}$:

$$\begin{bmatrix} \Delta x \\ \Delta y \\ \Delta \psi \end{bmatrix} = \begin{bmatrix} x_{t+\Delta t} \\ y_{t+\Delta t} \\ \psi_{t+\Delta t} \end{bmatrix} - \begin{bmatrix} x_t \\ y_t \\ \psi_t \end{bmatrix}. \quad (11)$$

The prediction model (7) for the CA states is linear:

$$x_{t+\Delta t}^{\text{ca}} = \mathbf{A}_{ca} x_t^{\text{ca}} + \mathbf{B}_{ca} v_t^{\text{ca}},$$

(12)

whereas the additional CT states are given by a non-linear model:

$$x_{t+\Delta t}^{\text{ct}} = g(x_t^{\text{ct}}, v_t^{\text{ct}}).$$

(13)

Details regarding the CA and CT prediction models are given in appendix A.

It is clear from equations (5) – (6) that any points described using Cartesian coordinates can be aligned similarly, including stationary objects. Parametric models, of e.g. lane markings, can pose a challenge if not invariant to rotation, in which case a useful approximation is to sample points using the model, time-align these points, and then estimate the parameters of the aligned model from the samples.

V. OBJECT STATE FUSION

Following our argumentation in Section III, at least one sensor is assumed to provide filtered estimates of some states, $x^i$, contained in the sensor data $z^i$. Therefore, when fusing $z^i$ with data from another sensor, common approximations to the recursive estimation solution such as the Kalman filter [15] are not applicable without violating underlying filtering assumptions, since the measurements are not independent.

By object state fusion, we mean to combine $x^i$ with information from other sensors, $z^j$, as illustrated in Fig. 5. The fusion system is executed periodically at a higher rate than the object sensors, using the most recent estimates from each sensor in each fusion step. A dashed arrow indicates an optional connection. For example, the fused result from the previous iteration can be used to facilitate data association and to remember attributes such as object type, which otherwise would be lost when a track is no longer observed by the classifying sensor. Yet, the old estimate is not necessarily used in the state update.

Each object state fusion iteration comprises several steps: Data association, Track management, Priority selection and State update, shown in Fig. 2. The first three steps prepare for the state update and are only briefly discussed in this article. They are well described in the tracking literature, such as [3], and well-known methods can be applied in the proposed architecture. For completeness we list the methods we are using in the following section.

A. Preparing for state update

Data association deals with the problem of deciding if one or several objects in the sensor object lists are the same and thereby can be fused. We use the Global Nearest Neighbor (GNN) method implemented using the auction algorithm [16] and a distance metric based on position and velocity residuals.

The purpose of a track management (TM) scheme in a tracking system is to (a) start new tracks from unassociated measurements and (b) delete tracks that are no longer observed or doesn’t meet some minimum level of quality, see e.g. [3]. In our case the sensors themselves have an internal TM-scheme and the main purpose of a second track management in the fusion system is to maintain tracks internally even if they are removed by the sensors. This is useful, e.g., to remember attributes when an object is moving in and out of sight of a particular sensor, or when an internal estimation algorithm is employed for a particular purpose (see Section VI).

Priority selection is a simple but important component in the presented architecture. Multiple object lists are created in the output to enable application specific interfaces and these lists cannot contain too many objects. The priority selection is a set of rules that selects objects to be fused and be part

Fig. 5. Sensor data is delayed due to pre-processing in the sensors and communication delays. The left and right borders of a sensor box indicate when a measurement is made, and is available, respectively. Although the fusion only uses the most recent data from each sensor, we have included counters ($m, n$) in this figure to correctly depict how the content changes over time.

Each object state fusion iteration comprises several steps: Data association, Track management, Priority selection and State update, shown in Fig. 2. The first three steps prepare for the state update and are only briefly discussed in this article. They are well described in the tracking literature, such as [3], and well-known methods can be applied in the proposed architecture. For completeness we list the methods we are using in the following section.

A. Preparing for state update

Data association deals with the problem of deciding if one or several objects in the sensor object lists are the same and thereby can be fused. We use the Global Nearest Neighbor (GNN) method implemented using the auction algorithm [16] and a distance metric based on position and velocity residuals.

The purpose of a track management (TM) scheme in a tracking system is to (a) start new tracks from unassociated measurements and (b) delete tracks that are no longer observed or doesn’t meet some minimum level of quality, see e.g. [3]. In our case the sensors themselves have an internal TM-scheme and the main purpose of a second track management in the fusion system is to maintain tracks internally even if they are removed by the sensors. This is useful, e.g., to remember attributes when an object is moving in and out of sight of a particular sensor, or when an internal estimation algorithm is employed for a particular purpose (see Section VI).

Priority selection is a simple but important component in the presented architecture. Multiple object lists are created in the output to enable application specific interfaces and these lists cannot contain too many objects. The priority selection is a set of rules that selects objects to be fused and be part
of the output. These rules depend on the application and the goal is to reduce the number of target candidates as much as possible without actually missing any relevant targets. The fusion system is better equipped to do so than a single sensor since it can position tracks on the road using fused estimates, and the scheme can be developed and verified using previously logged data.

B. State update

There are numerous ways to combine sensor tracks. Focus here is on three conceptually different types of fusion, characterized in how similar the state vectors are prior to fusion. Understanding these differences have a larger impact on fusion design than the choice of method for combining the estimates.

Fig. 6 shows two state vectors that have some states in common (redundant information) and some that are unique for each state vector (complementary information). These are fused into a state vector where some parts originate from a single sensor and other parts is a function of both estimates.

\[
\begin{bmatrix}
  x_1^i & x_2^i & x_1^j & x_2^j \\
  x_3^i & x_4^i & x_3^j & x_4^j
\end{bmatrix}
= 
\begin{bmatrix}
  x_1^i & x_2^i & \theta_1^i & \theta_2^i \\
  x_1^j & x_2^j & \theta_1^j & \theta_2^j
\end{bmatrix}^T
\]

![Fig. 6. Fusion of two state vectors, \( x_1^i = [x_1^i, x_2^i, x_3^i, x_4^i]^T \) and \( x_1^j = [x_1^j, x_2^j, \theta_1^j, \theta_2^j]^T \). Redundant states are combined whereas complementary states or properties are inherited unchanged.](image)

Fusing complementary information is clearly straightforward given a successful data association, whereas redundant states require specific care to avoid data incest in the fused output. We differentiate between complementary fusion, track-to-track (T2T) fusion and track-to-measurement fusion.

1) Complementary fusion: Assuming there are no shared states in \( x_1^i \) and \( x_1^j \), the augmented fused state is

\[
x_t = \begin{bmatrix} x_t^i \\ x_t^j \end{bmatrix}.
\]

![Fig. 7. State fusion example: A sensor is added to the scenario depicted in Fig. 4, resulting in a classification of the target and an increased estimation accuracy. The circular covariance contour represents the uncertainty of the fused estimate.](image)

Naturally, some states must be shared so that objects in different lists can be associated as the same object, but they need not necessarily affect the fused output. An automotive safety system that can make use of such a fusion is an auto-brake system that reacts to radar targets if they are also classified as a vehicle by a camera.

2) Track-to-track fusion: Tracks defined in the same state-space can be directly compared and combined to improve the estimation accuracy, which is denoted T2T fusion. Several methods are known in literature and this architecture is designed for such methods to be used interchangeably. An introduction to T2T-fusion can be found in [17] and different methods are compared in [18]. The covariance intersection (CI) fusion method [19] is straightforward to apply and is employed in implementations of the proposed architecture. The fused estimate, \( x_c \), of estimates \( x_1^i \) and \( x_1^j \) is:

\[
x_t = P_{x_t}^{-1} \left( \omega P_{x_t}^{-1} x_1^i + (1 - \omega) P_{x_t}^{-1} x_1^j \right) \quad (15)
\]

\[
P_{x_t}^{-1} = \omega P_{x_t}^{-1} + (1 - \omega) P_{x_t}^{-1} \quad (16)
\]

where \( \omega \) is a weight used for tuning.

Better state estimates give more accurate predictions and, consequently, a better performing automotive safety system. The concept is illustrated in Fig. 7.

A more challenging situation arises if \( x_1^i \) and \( x_1^j \) do not share states, yet contains essentially the same information. For example, \( x_1^{ct} \) and \( x_1^{ns} \) are related by an unambiguous mapping, but only two of the six states are identical. A method for fusing transformed states is presented in [12], which can be employed when the standard CI-fusion cannot be directly applied.

3) Track-to-measurement fusion: Assume that sensor data \( z^i \) contains the tracked states \( x^i \), which are to be fused with non-filtered measurements \( z^j \) contained in \( z^j \). Intuitively, such track-to-measurement filtering is straightforward — predict the sensor-track and update it with the measurement. Unfortunately this is possible only when using the most recent measurement. When an updated sensor-track, \( x_1 \), is available, any fused track comprising old estimates from the same sensor should discard of the old estimates before using \( x_1 \) to avoid data incest. However, it is difficult to do so without also losing information from any past fused measurements, \( y^j \). To avoid the problem, the sensors must be designed to transmit tracked data such that an optimal de-centralized scheme can be constructed, see e.g.,[20], or implement a method for data incest removal [21].

The presented fusion architecture is designed to use two approximations which in practice have shown to perform sufficiently. The first approximation is to ignore old measurements, i.e.:

\[
p(x_t | z_{1:k}, z_{1:k}^j) \approx p(x_t | z_{1:k}^j, z_{1:k}^j), \quad (17)
\]

and the second is to track the unfiltered measurements independently of the fusion, then fusing both tracks using standard T2T methods. The former approximation is useful when the measurement is relatively accurate and mainly affects predictions, such as a transmitted yaw-rate signal. The latter method has benefits when the measured signal is noisy, yet has a useful dynamic model. One such example is short-range distance estimates from ultrasonic sensors, which can be tracked using a standard Kalman filter and is useful if the radar reflection point fluctuate longitudinally on the target vehicle.
VI. VIRTUAL SENSORS

Some applications require information that either is not available from a single sensor or would be too costly to measure. Modules dedicated to providing this information are categorized as Virtual sensor modules in the proposed architecture. These modules can be very simple, such as Snail trails, which computes the trajectory of an object’s previous positions — presented in the current coordinate system,

$$\mathbf{x}_{traj} = \begin{bmatrix} x_{t_k} & \cdots & x_{t_{k-N}} \\ y_{t_k} & \cdots & y_{t_{k-N}} \end{bmatrix},$$  \hspace{1cm} (18)

or Traffic flow, which estimates the traffic intensity in the different lanes of the road. Other modules can be much more complicated, for instance Road geometry fusion, see Fig. 8, which estimates the shape of the road ahead by fusing camera observations of lane markings with radar observations of other road users and static objects next to the road [22]. More examples that would fit into this category are ego vehicle positioning, route prediction, and the examples presented in [23].

These modules are all tightly coupled to the applications. Some are even developed to support a certain application with specific tasks, such as target selection for an adaptive cruise control application, which offer a great flexibility towards tailoring the environmental description according to the requirements of the applications. It is therefore important to develop these methods in-house, and to do so as independently as possible from a particular sensor set-up. In practice, an internal sensor list, $S^I_{k,t}$, can be tuned exclusively for such a module.

VII. CONCLUSIONS

A vehicle manufacturer should avoid using raw measurements in the sensor data fusion system in order to be flexible towards sensor changes and facilitate functional safety management. Instead, focus should be on the correct treatment of data and a systematic approach towards verifying and re-using code components and complete functions. If the required performance cannot be met using a high-level algorithm (e.g., T2T fusion), a sensor system employing low-level fusion can be employed separately and the output treated as a single sensor in the architecture. Nevertheless, the time alignment described in Section IV is always needed.

A fusion system based on the proposed architecture is implemented in a collaborative project [24] that comprises two demonstrator vehicles — a truck and a car — with different electrical architectures, equipped with radars, cameras and other sensors from multiple suppliers. These vehicles have demonstrated applications ranging from emergency braking to multi-target steering maneuvers. The proposed fusion system has been successfully employed on both demonstrator vehicles, sharing all components but sensor specific tuning parameters (e.g. data association thresholds), indicating that the design is appropriate for multiple sensor and vehicle configurations.

APPENDIX

This section clarifies the relation between the proposed state vectors and their respective prediction models.

A. Prediction models

The matrices in the prediction model (12) for the CA parameterization are given by

$$\mathbf{A}_{ca} = \begin{bmatrix} 1 & 0 & \Delta t & 0 & \Delta t^2/2 & 0 \\ 0 & 1 & 0 & \Delta t & 0 & \Delta t^2 \\ 0 & 0 & 1 & 0 & \Delta t & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ \Delta t^2/2 & 0 & \Delta t^2 \\ 0 & \Delta t \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$  \hspace{1cm} (19)

and the nonlinear prediction model for the CT parameterization (3) is:

$$x_{t+\Delta t} = x_t + \cos \psi_t \left( v_t \Delta t + \frac{a_t \Delta t^2}{2} + \frac{\dot{a}_t \Delta t^3}{2} \right) - \cdots \sin \psi_t \left( \frac{v_t \beta_t \Delta t^2}{2} + (v_t \beta_t + 2a_t \beta_t) \Delta t^3 + \cdots \right)$$

$$y_{t+\Delta t} = y_t + \sin \psi_t \left( v_t \Delta t + \frac{a_t \Delta t^2}{2} + \frac{\dot{a}_t \Delta t^3}{2} \right) + \cdots \cos \psi_t \left( \frac{v_t \beta_t \Delta t^2}{2} + (v_t \beta_t + 2a_t \beta_t) \Delta t^3 + \cdots \right)$$

$$\psi_{t+\Delta t} = \psi_t + v_t \Delta t + \frac{a_t \Delta t^2}{2} + \frac{\dot{a}_t \Delta t^3}{2} + \cdots$$

$$v_{t+\Delta t} = v_t + a_t \Delta t + \frac{\dot{a}_t \Delta t^2}{2}$$

$$c_{t+\Delta t} = c_t + \dot{c}_t \Delta t$$

$$a_{t+\Delta t} = a_t + \dot{a}_t \Delta t,$$  \hspace{1cm} (21)

where $\dot{c}_t$ is a curvature change rate noise term, $\dot{a}_t$ is a jerk noise term and $\beta_t, \dot{\beta}_t$ are defined as

$$\beta_t = v_t c_t$$

$$\dot{\beta}_t = a_t c_t + v_t \dot{c}_t + a_t \dot{c}_t + \frac{\dot{a}_t c_t \Delta t}{2} + \frac{\ddot{a}_t c_t \Delta t^2}{2}.$$
B. State vector transformations

The relation between the CA and CT state vectors is given by \( f_1 : x_{ca} \mapsto x_{ct} \) and \( f_2 : x_{ct} \mapsto x_{ca} \):

\[
f_2(x_{ct}) = \begin{bmatrix} x \\ y \\ \sqrt{x^2 + y^2} \\ \alpha \sin \psi \\ \alpha \cos \psi - cv^2 \cos \psi \\ \alpha \sin \psi + cv^2 \sin \psi \end{bmatrix}
\]

\[
f_1(x_{ca}) = \begin{bmatrix} x \\ y \\ \sqrt{x^2 + y^2} \\ \alpha \sin \alpha + \frac{\dot{y} \cos \alpha - \dot{x} \sin \alpha}{\sqrt{\dot{x}^2 + \dot{y}^2}} \\ \dot{y} \sin \alpha + \dot{x} \cos \alpha \end{bmatrix}
\]

where \( \alpha = \arctan \left( \frac{\dot{y}}{\dot{x}} \right) \). These transformations are sufficient to transform a given point in one state-space to the other and back again.

ACKNOWLEDGMENT

This work has been supported by the Strategic Vehicle Research and Innovation Program (FFI), which is funded by the Swedish Governmental Agency for Innovation Systems (VINNOVA).

REFERENCES