A DESIGN ARCHITECTURE FOR SENSOR DATA FUSION SYSTEMS WITH APPLICATION TO AUTOMOTIVE SAFETY

Fredrik Bengtsson, Lars Danielsson

In this paper we present a modular sensor data fusion functional architecture, tailored for development of automotive active safety systems. The purpose of the fusion system is to provide active safety applications with accurate knowledge regarding the environment surrounding the vehicle. Our proposed functional architecture is designed in such way that the fusion system is easy to maintain, upgrade and re-use. These aspects are assessed by the use of a reference implementation which is evaluated in terms of tracking performance and scalability. Furthermore, the reference implementation demonstrates that a system can be implemented using rapid prototyping tools, from which we can automatically generate c-code.

INTRODUCTION

In order for automotive active safety applications to make decisions about when to warn or intervene in dangerous traffic situations, reliable information about the traffic environment surrounding the vehicle is needed. Measurements on the environment are typically supplied by sensors mounted on the vehicle, for example radar sensors and vision systems. In order to meet the challenging requirements posed by safety critical applications, it is increasingly common to use information from several on-vehicle sensors in a *data fusion* framework. The task of fusing sensor data is performed by a *tracking system* or *perception layer* (1).

Tracking systems have been researched extensively and there is a significant amount of results available regarding system design, e.g. (2, 3). In this paper we have regarded the task of system design from an automotive safety system research perspective, which in some aspects differ from many other tracking applications; sensors or hardware are subject to change and multiple applications pose different requirements. At the same time, hardware and software should to a high degree be shared components. The result is a functional architecture that allows for robust, versatile implementations using known tracking strategies.

The proposed architecture is used in a reference implementation to evaluate the design principles. Algorithms are implemented using Mathworks Embedded MATLAB, from which it is possible to generate c-code and demonstrate the system in real-time on prototype PC hardware. The work is supported by Swedish *Intelligent Vehicle Safety Systems* (IVSS) program and is a part of the *Sensor Fusion for Safety* (SEFS) project.

PROBLEM FORMULATION

The main objective for this paper is to describe a functional architecture that can be used when designing a fusion system, all the way from a research platform to the vehicle production system. In this section we discuss aspects that need to be considered in order to solve this task.

Fusion Architecture

Active safety system research is aided by a perception layer where sensors can be exchanged and different tasks can be developed and evaluated separately. It is also desirable to have an architecture that allows code to be re-used through different development steps, *i.e.* it should be straightforward to go from the research system to an in-vehicle system. Embodiments intended for in-vehicle programs need to be developed in parallel with safety applications, while as long as possible keeping the door open for introducing new sensors or achievements regarding fusion methods. For these reasons, an architecture suitable for both research and embedded implementations, as illustrated in Figure 1, would be very useful.

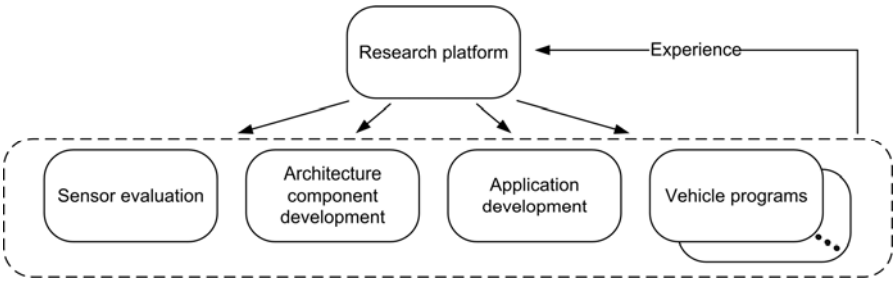

Figure 1: Fusion system components are used in multiple applications, ranging from sensor evaluation and research to embedded implementations in a vehicle program. A common architecture makes it possible for every application to make use of recent developments.

Therefore we aim to present a modular architecture that facilitates fusion system development jointly with sensor evaluation and in-vehicle studies.

Practical considerations

In the most natural multi-sensor tracking scenario all sensors are synchronized and the complete state vector is updated using all available information simultaneously. However, in practice sensor data is delayed due to internal signal processing algorithms or limited communication bandwidth and will arrive to the fusion in an asynchronous manner. If this is not considered, performance will suffer, e.g. as shown in Figure 2.

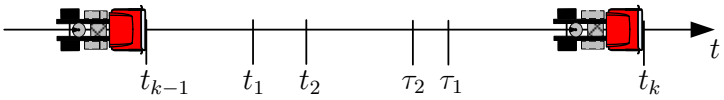

Figure 2: During t_{k-1} to t_k a truck moves from left to right in the figure. Sensor one reports the position at time t_1 and sensor two reports the position at t_2. However, the detections are delayed until τ_1 and τ_2 respectively. In this figure $\tau_1 > \tau_2$, which implies that the vehicle moved forward in the time interval $(t_{k-1}, \tau_2]$ and then backwards during $(\tau_2, \tau_1]$.

The arrival order of measurements is generally unknown in advance, and in a system with delays it may happen that we receive a so called out-of-sequence measurement. As shown in Figure 3, it is possible for a measurement to arrive after the state vector has been updated with information from a newer measurement, *i.e.* out-of-sequence, in practice a problem which requires special treatment.

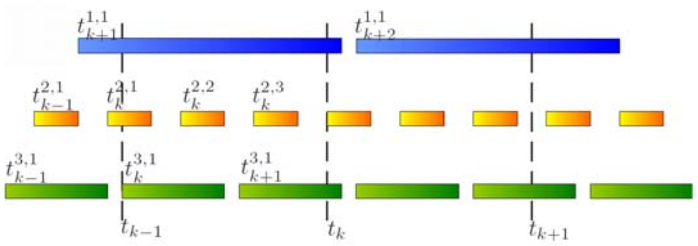

Figure 3: Sensor data measured at $t_k^{(i,j)}$, arrive to the fusion at time t_k, where i define the sensor and j is a measurement counter for that sensor. The fusion system receives data at times $(t_0,\ t_1,\ \ldots, t_k,\ \ldots)$ and must handle asynchronous measurements, some of which take longer time to reach the fusion system than others. One reason for such behaviour is that certain sensors transmit accumulated measurements in a burst, *i.e.* a list of measurements. Note that $t_k^{(i,j)}$ may be smaller than t_{k-1}.

Filter technique and data association

Algorithms typically make use of a Kalman filter (KF) framework, while sensor specific methods are used in pre-processing steps such as data association and track initialization. It is important that the architecture supports filtering techniques such as Extended Kalman filter, Unscented Kalman filter, multiple model frameworks and, to some extent, Monte Carlo methods. At the same time, sensor specific adaptations must be allowed.

Modularity

There is often an architectural conflict between a modular and re-usable system and the optimal signal processing algorithm, fully exploiting all the information in signals and models. To achieve the flexibility described above, processing should be performed in a certain order with limited information exchange between adjacent functional blocks. In other words it needs to be modular, which may lead to suboptimal processing.

TRACKING SYSTEM

The task of a tracking system is to, at a time t, describe the measured environment as good as possible given all available data, including uncertainties. Here we introduce the discrete-time state vector $\mathbf{x}_k$ to contain everything we want to know at time $t_k = kT_s(k)$, where $T_s(k)$ is the system sample time and $k \in \mathbb{N}$ a counter. Similarly, we introduce $\mathbf{y}_k$ to be the vector of measurements on the surroundings received in the time period $(t_{k-1}\ ,\ t_k]$. For a system with N sensors we form

$$\mathbf{y}_k = [(\mathbf{y}_k^1)^T\ (\mathbf{y}_k^2)^T\ \ldots\ (\mathbf{y}_k^N)^T]^T \tag{1}$$

The task can now be formulated as to calculate the posterior density $p(\mathbf{x}_k|\mathbf{y}_1, \ldots \mathbf{y}_k)$. In order to do this, two statistical models are needed, a *process model* and a *measurement model*. Textbooks which thoroughly explain estimation and modelling are *e.g. (4, 5).*

In an automotive context, a tracking system is responsible for refining the information supplied by onboard sensors to supply safety applications with information about the surrounding traffic situation. In Figure 4 this relation is depicted together with components

that are necessary in order to calculate $p(\mathbf{x}_k|\mathbf{y}_1, \ldots \mathbf{y}_k)$. This includes *gating and data association*, *track management*, *measurement update* and *state prediction*. More information about these different components can be found in *e.g.* (3, 6, 7).

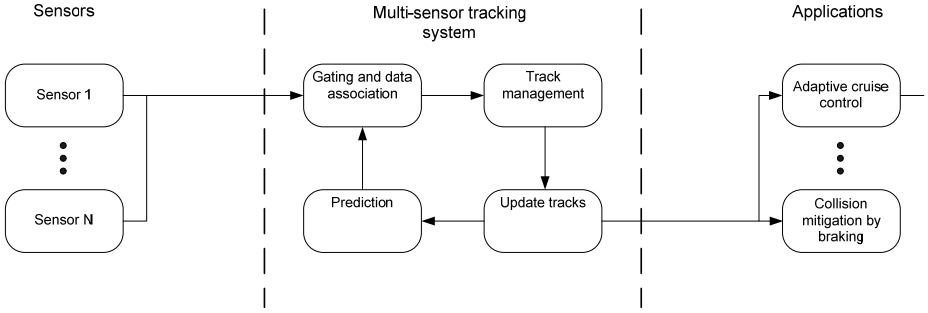

Figure 4: Schematic view of a tracking system and its components, in the context of typical automotive comfort or safety applications.

In a multi-sensor system, as in Figure 4, depending on the update rate of the tracking system, the measurement vector from each sensor, $\mathbf{y}_k^i$, may be composed of several measurements of different age. In this case $\mathbf{y}_k^i$ has the following structure

$$\mathbf{y}_k^i = [(\boldsymbol{\gamma}_k^{(i,1)})^T \; \tau_k^{(i,1)} \; (\boldsymbol{\gamma}_k^{(i,2)})^T \; \tau_k^{(i,2)} \; \ldots \; (\boldsymbol{\gamma}_k^{(i,M^i(k))})^T \; \tau_k^{(i,M^i(k))}]^T \tag{2}$$

where $M^i(k)$ are all measurement lists from sensor i received in the time period $(t_{k-1}, t_k]$, with corresponding time stamps $\tau_k^{(i,0:M^i(k))}$. That is to say, $\boldsymbol{\gamma}_k^{(i,j)}$ contains all measurements received at time $\tau_k^{(i,j)} \in [\tau_k^{i,j-1}, t_k]$. Note that $\tau_k^{i,0}$ is defined as the time sensor i last delivered data prior to t_{k-1} and that $M^i(k) \geq 0$, i.e. sensor i can deliver zero or multiple measurements during an update cycle of the fusion system. If the sensor delays are known, the time for the actual measurement $t_k^{(i,j)}$, can be derived from $\tau_k^{(i,j)}$. Thus it is possible for $t_k^{(i,j)}$ to be smaller than t_k. Figure 3 shows how measurements from three different sensors fall in different measurement vector slots, for example $\{\boldsymbol{\gamma}_k^{(2,1)}, \boldsymbol{\gamma}_k^{(2,2)}, \boldsymbol{\gamma}_k^{(2,3)}, \boldsymbol{\gamma}_k^{(3,1)}\} \in \mathbf{y}_k$.

FUSION ARCHITECTURE

A central requirement on the perception layer is a high degree of modularity, so that algorithm components can be continuously developed and sensors may be exchanged or added to the system. Further, it must be suitable for automotive safety systems regarding *e.g.* in- and output data, computational load and terms of robustness, etc. It is a challenge to design a system with these properties, but the work is aided by an appropriate high level functional design.

Architecture outline

Our goal is to treat perception layer functional tasks in a fashion that facilitate usage according to Figure 1. One important aspect is how to control the flow of information; the magnitude of data transfer easily grows during development. We suggest a one-way data flow where usage of internal variables is minimized and transferred variables are well

specified and available to modules in a predictable fashion. A well organised data flow that has grown during development can be slimmed at a later stage, depending on which methods are actually used. Functional tasks are grouped into modules, chosen to facilitate separate development and are shown in Figure 5, supporting a variety of state update methods to be used for each sensor individually. Models should be restricted to the class of models with the Markov property. However, usage of statistical linearization methods allow models to be readily exchanged – only the transformation function need to be defined – which further supports modularity.

Functional modules

A functional architecture supporting an asynchronous fusion scheme, as discussed in the problem formulation, is shown in Figure 5. The general idea is to have a common prediction and delay compensation block, run at each internal iteration in the fusion scheme, and to divide the subsequent processing modules by sensor data origin. A dedicated sensor processing module is used for each sensor, *e.g.* if internal sensor data is to be processed the filter internal sensors module is activated. After all data in a cycle has been processed, additional track handling strategies may be applied on a global level. It is recommended to treat all parameters needed in the fusion system as input and output signals, to avoid internal "global parameters".

The architecture is readily extended with new blocks, *e.g.* including data from an e-horizon system or implementing an estimator to be used for complex posterior distributions. Adaptations such as performing *e.g.* track-to-track fusion, can be assigned to existing modules.

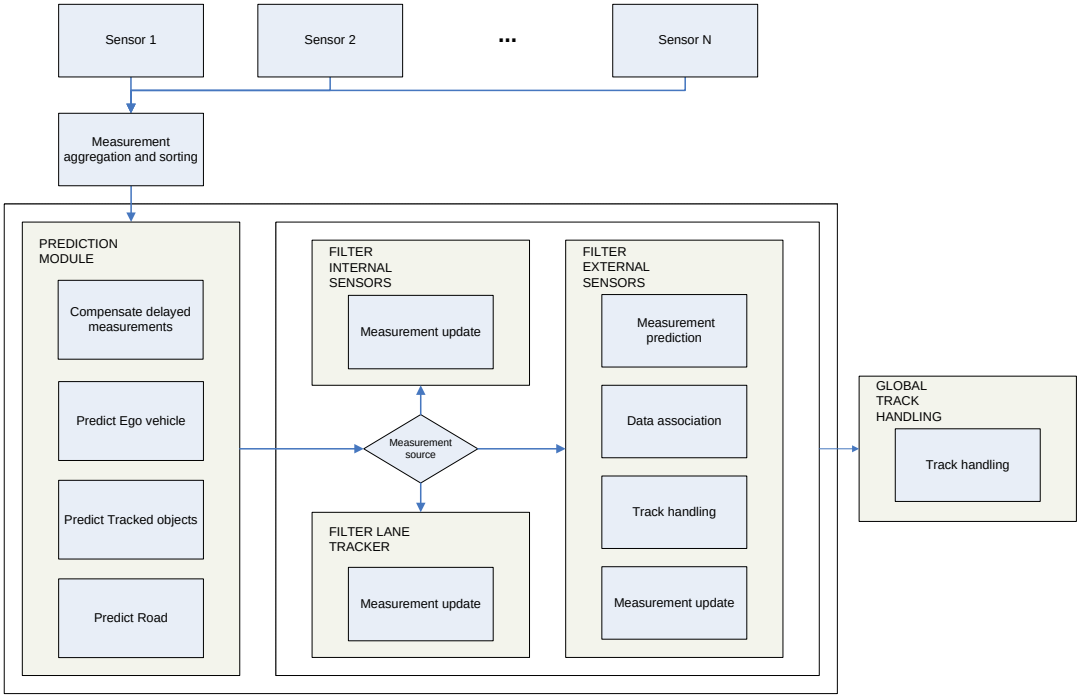

Figure 5: Fusion algorithm functional architecture. The large block is called once for each received measurement, executing necessary prediction and filter modules in the correct order.

Measurement aggregation and sorting

The information from each sensor arriving to the fusion system is stored until the beginning of the next system cycle, allowing *e.g.* platform specific conversions. The aggregated measurements are sent to the fusion system in a sorted manner; the measurement that occurred first is sent first, etc. Naturally, any knowledge regarding expected measurement delays should be incorporated in the sorting.

Prediction module

The prediction of the state is made in four basic components, *predict ego vehicle*, *predict tracked objects, predict road* and *compensate delayed measurements*. These components make sure that the measurements and state vector are aligned in time for the upcoming filter blocks. Typically this module is used to predict $\mathbf{x}_{k-1}$ to the time of the earliest measurement $t_{\kappa_1} \in (t_{k-1}, t_k]$ and subsequently with the next measurement at $t_{\kappa_2} \in [t_{\kappa_1}, t_k]$. The *compensate delayed measurements* block is called in the event of an out-of-sequence-measurement, and several strategies can be used to process such data. If models allow, a so called retrodiction step can be incorporated in the update procedure, explained *e.g.* in (8). An optimal retrodiction increases system complexity as it requires us to store previous states, a problem which can be avoided using sub-optimal alternatives. Nevertheless, blocks in the corresponding *filter external sensor module* must support retrodicted state updates which in itself add complexity. A procedure that does not affect following modules is to disregard measurements deemed to old, and to adapt the measurement distribution accordingly for measurements actually used.

Filter internal sensors

The ego vehicle is often modelled in more detail than observed vehicles and measurements not available from other vehicles are typically obtained at a high rate. For these reasons it is suitable to perform the filtering of measurements from internal sensors in a separate module.

Filter lane tracker

Lane tracker measurements are often pre-processed and should if possible be treated using knowledge regarding internal filtering models. Nevertheless, this does not prevent $\mathbf{r}_k$ from being updated with data from external sensors in another module.

Filter external sensors

Each sensor observing the environment outside the ego vehicle has a corresponding filter module. Sensors may observe the world quite differently and it is here detailed sensor models are used to perform *measurement prediction*, *data association* and *state updates*. Track management such as *track initialization* or calculating *track score* can be carried out, and in multiple model frameworks, model probabilities are updated. Note that the module input generally is an object list, which is required for efficient data association.

Global track management

Track deletion, validation and merging of tracks can be done on a global level to complement the limited set of track management tasks done locally for each sensor. Strategies for object classification, for example during track validation, can be implemented here.

IMPLEMENTATION

We have implemented a simple, but fully autonomous, tracking system to be evaluated before proceeding to develop components further. It is based on the proposed architecture and runs on a platform that can be used in real-time in vehicle demonstrators (9). Detailed information regarding this particular implementation can be found in (6, 7).

Parameterization

For a total of n_c tracked cars, the state vector $\mathbf{x}_k$ is partitioned as

$$\mathbf{x}_k = [(\mathrm{z}_k^{ego})^T \ (\mathrm{z}_k^1)^T \ (\mathrm{z}_k^2)^T \ \ldots \ (\mathrm{z}_k^{n_c})^T \ \mathrm{r}_k^T]^T \tag{3}$$

where $\mathbf{z}_k^i$ is the state vector for vehicle i and $\mathbf{r}_k$ describes the road. The choice of states in $\mathbf{z}_k^i$ is directly affected by the choice of vehicle motion model. Similarly, the parameterization of $\mathbf{r}_k$ is coupled with the choice of road process model.

Sensor setup

The evaluation sensor setup is limited to one radar, a lane tracker camera and internal sensors measuring ego vehicle states.

Radar

Delphi's ACC3.5, 77GHz Automotive Radar, with a detection range of approximately 150 meters and an opening angle of 16° is mounted in the front of the vehicle. Up to 20 unfiltered detections, each consisting of range (r [m]), range rate ($\dot{r}$ [$\frac{m}{s}$]) and azimuth (φ [rad]), grouped in an *object list* are reported every measurement cycle (100 ms). The j^{th} object list[1] is written

$$\boldsymbol{\gamma}^{(1,j)} = [r_1 \ \varphi_1 \ \dot{r}_1 \ r_2 \ \varphi_2 \ \dot{r}_2 \ \ldots \ r_{20} \ \varphi_{20} \ \dot{r}_{20}]^T, \tag{4}$$

letting index one denote the radar sensor.

Vision system

A camera based lane tracking system provides measurements on the vehicle heading relative the road Ψ_{rel} [rad], the distances to the left and right lane markings R_{off} and L_{off} [m] respectively, and the road curvature c_0 [m^{-1}]. The j^{th} measurement in a fusion cycle is

[1] for the j^{th} object list in the k^{th} iteration, but k is left out at this point for clarity reasons

$$\boldsymbol{\gamma}^{(2,j)} = [\Psi_{rel}\ R_{off}\ L_{off}\ c_0]^T, \tag{5}$$

if the vision sensor is indexed as sensor number two.

Internal sensors

Several internal sensors are available in modern vehicles. In this system the vehicle speed v $[\frac{m}{s}]$, acceleration a $[\frac{m}{s^2}]$ and yaw-rate $\dot{\Psi}$ $[\frac{\text{rad}}{s}]$ are used for ego vehicle tracking. Consequently,

$$\boldsymbol{\gamma}^{(3,j)} = [v\ a\ \dot{\Psi}]^T. \tag{6}$$

Filtering

A statistical linearization algorithm (10) is used to estimate effects of nonlinear transformations, resulting in the so called Unscented Kalman filter (UKF). Effects of linear transformations are calculated analytically. The global nearest neighbour data association method is implemented using the auction algorithm (11) and unlikely associations are ruled out using ellipsoidal gates. Unassociated measurements yield new tracks, confirmed when associated with measurements *n*-times out of *m* possible. Track deletion occurs using a similar scheme, or when uncertainties are larger than a threshold. Measurement delays are estimated and assumed known and, when using a single radar, there are no problems with out-of-sequence-measurements.

Vehicle motion model

The same parameterization and dynamic model is used for other vehicles as well as for the ego vehicle. We include the global position (ξ_x, ξ_y), heading Ψ, velocity v, yaw-rate $\dot{\Psi}$ and acceleration a in the vehicle state vector:

$$\mathbf{z}_k = [\xi_x\ \ \xi_y\ \ \psi\ \ v\ \ \dot{\psi}\ \ a\ ]^T. \tag{7}$$

The motion model is derived from the continuous-time model

$$\dot{\mathbf{z}}(t) = [v(t)cos(\psi(t))\ \ v(t)sin(\psi(t))\ \ \dot{\psi}(t)\ \ a(t)\ \ 0\ \ 0]^T + [0\ \ 0\ \ 0\ \ 0\ \ v_{\ddot{\psi}}(t)\ \ v_{\dot{a}}(t)]^T. \tag{8}$$

$v_{\ddot{\psi}}(t)$ and $v_{\dot{a}}(t)$ describe modelling errors and are continuous time Gaussian stochastic processes. Both are zero mean and white, with intensity $q_{\ddot{\psi}}$ and $q_{\dot{a}}$, respectively. A fixed step-length discrete model is used, a derivation of which is presented in (12).

Road process model

We follow suggestions from *e.g. (13)* and use a clothoid model to make a local approximation of the road curvature around the ego vehicle. The curvature c_0 changes as a linear function, so at distance η ahead of the ego vehicle, the curvature is written

$$c_0(\eta) = c_0(0) + \eta\, c_1, \tag{9}$$

The road state vector contains ego vehicle *heading angle relative the road, distances to the lane markings* and the *curvature parameters* (c_0, c_1). Apart from c_0 described in (9), all states are modelled as constants influenced by zero mean white Gaussian noise (WGN).

Measurement models

The radar sensor model is kept fairly simple, which is sufficient for this evaluation. It is assumed that the vehicle can be modelled as a point target, *i.e.* at most one measurement may originate from each vehicle. The measurement equation for one radar detection ($i = 1$), is

$$\gamma_k^{1,j} = \begin{bmatrix} r \\ \dot{r} \\ \varphi \end{bmatrix} + \mathbf{w}_k^{1,j}, \tag{10}$$

where, omitting the time dependency (k) and assuming the detection originates from vehicle l,

$$\begin{aligned} r &= \sqrt{(\xi_x^l - \xi_x^{\text{ego}} - \epsilon_x)^2 + (\xi_y^l - \xi_y^{\text{ego}} - \epsilon_y)^2} \\ \dot{r} &= v^l \cos(-(\psi^l - \psi^{\text{ego}}) + \varphi) - v^{\text{ego}} \cos(\varphi) \\ \varphi &= \tan^{-1}(\frac{\xi_y^l}{\xi_x^l}) - \psi^{\text{ego}} - \epsilon_\psi \end{aligned} \quad . \tag{11}$$

$(\epsilon_x, \epsilon_y, \epsilon_\psi)$ represent the sensor mounting position and orientation in the local ego vehicle coordinate system. The ego vehicle motion is considered a deterministic control signal and measurement noise, $\mathbf{w_k^{1,j}}$, is WGN with known covariance.

Measurement models for the ego vehicle sensors and the vision system are linear. The vision system sensor model ($i = 2$), using the identity matrix $\mathbf{I}$, becomes

$$\gamma_k^{2,j} = \begin{bmatrix} \mathbf{I}_{4\text{x}4} & 0 \end{bmatrix} \mathbf{r_k} + \mathbf{w_k^{2,j}}, \tag{12}$$

and the sensor model for internal sensors ($i = 3$) is written

$$\gamma_k^{3,j} = \begin{bmatrix} \mathbf{0}_{3\text{x}3} & \mathbf{I}_{3\text{x}3} \end{bmatrix} \mathbf{z_k^{\text{ego}}} + \mathbf{w_k^{3,j}}. \tag{13}$$

EVALUATION

It is hard to assess and quantify architecture performance. In this section we aim to demonstrate that the purposed fusion architecture is capable of producing a functional and processing efficient perception layer, using rapid prototyping tools. This is accomplished by evaluating the estimation accuracy of the reference implementation and by making predictions of the computational load when additional sensors are included.

Performance comparison and evaluation

We compare the estimates from the reference implementation with that of a sensor specific tracker used in active safety systems already on the market, relative ground truth

data. The aim is to show that our modular system is capable of performing on par with or better then a dedicated sensor tracker based on the same data, and that estimates are sensible relative ground truth data.

The comparison is limited to two scenarios. Scenario 1 involves the lead vehicle making a hard braking manoeuvre whereas in scenario 2, the host vehicle is accelerating towards the target vehicle. Sensor data from the described sensor setup is collected together with highly accurate *differential global positioning system* (DGPS) measurements of the position of both the host and the tracked vehicle. Using these DGPS position measurements we estimate the quantities needed to evaluate the tracking performance. Note that only the position is measured directly and that other states are calculated using the motion model derived from (8) and a UKF filter. Hence, the estimates of the other states can mainly serve as an indication of the magnitude of the true errors. The parameters in the implementation have been tuned to cover common scenarios, and the same setup is used for scenario 1 and scenario 2.

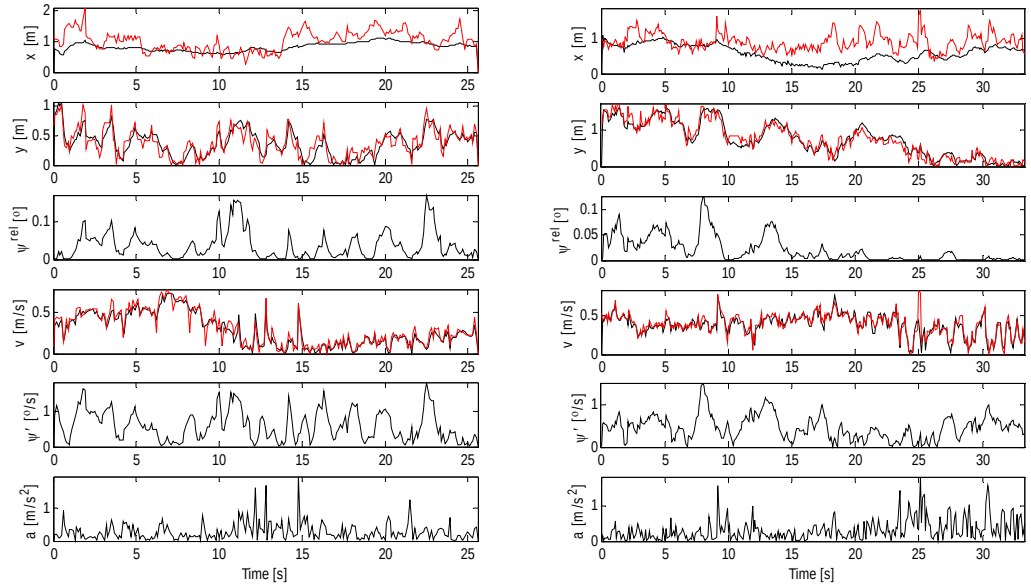

Figure 6: Evaluation of estimation accuracy for scenario 1 (left) and scenario 2 (right) using DGPS position measurements as reference. In the figures are the Euclidean error shown for all the states in the state vector for the target vehicle using the reference implementation (black) as well as for the sensor specific tracker (red), for applicable states.

Results, shown in Figure 6, indicate that a tracking system implemented using the proposed architecture is capable of delivering sensible estimates and that performance is comparable with that of a dedicated sensor tracker.

Processing time evaluation

The second aspect of our fusion architecture to be evaluated is it's scalability in terms of processing time for the multi-sensor system. As shown in Figure 5, adding a sensor does not affect the content of other modules. Hence, we can imitate a multi-sensor system by

independently duplicating both the prediction for the tracks and the filtering module for the external object sensor, i.e. simulating additional radar sensors. This way it is possible to simulate a larger number of simultaneous tracks, and multiple sensors.

In Table 1 the mean and maximum processing time for a system with 1 – 5 simulated external object sensors is shown. Simulated sensors deliver data simultaneously, which in corresponds to a "worst case" scenario. The processing times are measured running the algorithm in MATLAB on a laptop PC (Intel Pentium M 1,66 GHz). Initial tests indicate faster processing when running algorithms on a dedicated development hardware, i.e. xPC or dSPACE autobox (14, 15).

Table 1: Mean and maximum processing time for a system based on data from 1 - 5 sensors.

Processing time [s]	DF1	DF2	DF3	DF4	DF5
Mean	0,0049	0,008	0,0110	0,0141	0,0171
Maximum	0,0123	0,0191 (+55%)	0,0284 (+48%)	0,0367 (+29%)	0,0462 (+20%)

The measured processing times indicate that this implementation can run at least in 20 Hz incorporating data from 5 sensor similar to the forward looking radar.

CONCLUSIONS

We have proposed a fusion functional architecture that is modular in nature and support code and component re-use through different incarnations of the system. The architecture supports rapid prototyping tools, from which code suitable for production projects can be automatically generated. Sensor specific filter components with well defined input and output signals allow a high degree of adaptation while maintaining the general filtering framework. Several filtering techniques, such as the KF, UKF and EKF, are supported and can be used jointly in a single system. A reference implementation of the system based on standard methods run in real-time and performs on par with, or better than, a sensor specific tracker when using the same input data. We conclude that the proposed fusion architecture facilitates technology transfer and enables us to research and develop high-performing multi-sensor tracking systems for automotive safety systems in a structured, non-limiting, fashion.

REFERENCES

(1) A. Polychronopoulos, A. Amditis, U. Scheunert, and T. Tatschke, *Revisiting JDL model for automotive safety applications: the PF2 functional model,* in *The 9th International Conference on Information Fusion*. 2006: Florence.

(2) Y. Bar-Shalom and W. Dale Blair, *Multitarget-Multisensor Tracking Volume III: Applications and Advances*, Artech House, 2000.

(3) S. Blackman and R. Popoli, *Design and analysis of modern tracking systems*, Artech House, 1999.

(4) Y. Bar-Shalom, X. Rong Li, and T. Kirubarajan, *Estimation with applications to tracking and navigation*. 2001, John Wiley & Sons, Inc.: New York, NY.

(5) B. Ristic, S. Arulampalam, and N. Gordon, *Beyond the Kalman filter: particle filters for tracking applications*. 2004, Artech House: Norwood, MA.

(6) F. Bengtsson, *Models for tracking in automotive safety systems*. Institutionen för signaler och system, Chalmers tekniska högskola, 2008.

(7) L. Danielsson, *Tracking Theory for Preventive Safety Systems*. Institutionen för signaler och system, Signalbehandling, Chalmers tekniska högskola, 2008.

(8) Y. Bar-Shalom, "Update with out-of-sequence measurements in tracking: exact solution". *Aerospace and Electronic Systems, IEEE Transactions on*, 2002. **38**(3): p. 769-777.

(9) F. Bengtsson, L. Danielsson, and J. Gunnarsson, *D1.41 Definition of project demonstrators*. 2006: SEFS Deliverable D1.41.

(10) S.J. Julier and J.K. Uhlmann, "Unscented filtering and nonlinear estimation". *Proceedings of the IEEE*, 2004. **92**(3): p. 401 - 22.

(11) D.P. Bertsekas, "The auction algorithm for assignment and other network flow problems: a tutorial introduction ". *Interfaces*, 1990. **20**(4): p. 133 - 49.

(12) J. Gunnarsson, L. Svensson, L. Danielsson, and F. Bengtsson. *Tracking vehicles using radar detections*. in *Intelligent Vehicles Symposium, 2007 IEEE*. 2007.

(13) E.D. Dickmanns and A. Zapp, *A curvature-based scheme for improving road vehicle guidance by computer vision*, in *Proceedings of the SPIE Conference on Mobile Robots*. 1986.

(14) *dSPACE AutoBox*. Available from: http://www.dspace.com/ww/en/pub/home/products/hw/accessories/autobox.cfm.

(15) *xPC Target*. Available from: http://www.mathworks.com/products/xpctarget/.