Proceedings of the IEEE ITSC 2006
2006 IEEE Intelligent Transportation Systems Conference
Toronto, Canada, September 17-20, 2006

A Real-Time, Multi-Sensor Architecture for fusion of delayed observations: Application to Vehicle Localization

C. Tessier, C. Cariou, C. Debain
CEMAGREF
24, avenue des Landais
BP 50085, 63172 Aubière, France
e-mail: cedric.tessier@cemagref.fr

F. Chausse, R. Chapuis
LASMEA - UMR 6602
24, avenue des Landais
63177 Aubière, France
e-mail: chausse@lasmea.univ-bpclermont.fr

C. Rousset
ECA
Rue des Frères Lumière BP 24
83078 Toulon Cedex 09, France
e-mail: cro@eca.fr

Abstract— **This paper presents a software framework called AROCCAM that was developed to design and implement data fusion applications. This architecture permits to build applications in a very short time unburdening the user of sensor communication. Moreover, it manages unsynchronized sensors and delayed observations in an elegant manner that permits to the user to fuse those information easily, taking into account the environment perception date.**

In this paper, a fusion methodology for delayed observations is first presented in order to point the problem of latency periods in a fusion system. These latency periods are then taking into account within our embedded architecture needing only a little effort from user. Finally, benefits of AROCCAM architecture are demonstrated via a real-time vehicle localization experiment carried out with an outdoor robot.

I. INTRODUCTION

Nowadays, more and more vehicles are equipped with intelligent systems. These systems have to realize particular tasks such as: vehicle localization, automatic guidance, obstacle avoidance, pedestrian detection,.... To accomplish their tasks, they process sensor data. However, it is necessary to write piece of softwares to communicate with sensors. This task is tedious and most of all a lack of time. A solution to this problem is to use an embedded architecture. Such architectures permit to facilitate the development of algorithms by managing the communication between those algorithms and sensors. For instance, they can collect sensor data and control sensors without blocking algorithm mechanism.

The architecture proposed here has to respect several requirements:

- management of unsynchronized data and sensors latency,
- recording and replaying of sensor data in real-time,
- engineering requirements: re-usability, integration, maintenance, processing efficiency
- user requirements: easy of use, programming error detection.

When a system uses a single sensor, it can only sense a partial and incomplete part of the environment. By contrast, a multi-sensor approach is a way to improve environment perception. It's necessary to notify that a sensor provides an observation of the environment at a particular time. Another sensor provides a similar information at another time. The difficulty of a multi-sensor system is to fuse sensor data with different dates.

- In some works [1], [2], [3], sensor data are fused without taking into account the fact that information are unsynchronized.
- In other works [4], the data sampling frequency is increased in order to consider that all information are synchronized each other. Unfortunately, this assumption is false since each sensor has a latency time, which can not be taken into account by this way.

Data fusion systems become more and more used, which motivated us to design an architecture to answer the problem of unsynchronized sensor data taking also into account sensor latency.

This paper is organized as follows. Section II depicts the works related to embedded architecture, emphasising assets and drawbacks of each approach. In section III, our architecture is described by presenting each module and their objectives. Functionalities that ease the development of embedded software are also enumerated. Section IV enumerates all the latency periods that can appear in an observation and details the consequences of fusion of delayed observations. A solution is suggested to deal with such observations in a data fusion system. Section V describes the mechanism of AROCCAM to process delayed observations. Finally, a typical application involving our solution is given in the last section: real-time vehicle localization.

II. RELATED WORKS

Since several decades, different embedded architectures have been developed, each answering to requirements of a particular field of applications. However, all these architectures are agree with the necessity to be divided in several components.

For instance, the LAAS architecture [5] has three hierarchical levels, having different temporal constraints and manipulating different data representations. The main asset of such a decomposition is the realization of prototype software applications in a very short time. In the same manner, SCOOT-R [6], [7] the acronym for "Server and Client Object Oriented for the Real-Time", offers a framework for distributing tasks on multi-processing unit architecture. This software is in charge of communication between each processing unit. This permits to realize time-consuming applications by distributing tasks on several computers.

Another feature pointed up in the SCOOT-R architecture is the real-time aspect. It consists in giving the sensor data to the algorithm as soon as it's available. In the case of this architecture, a communication between each components must be implemented, since it is a distributed application. In order to solve the problem of communication time in a real-time software, a real-time network is used. Moreover, the real-time aspect of the mechanism of SCOOT-R components obliges each component to work in a very short time not to be declared as defective. Unfortunately, the determination of the processing time of sensor data is sometimes not possible. The upper limit of the processing time can always be evaluated. In general, vision algorithms like road detection [8] can take more than $100ms$ per images. This reduce the utilisation of this architecture to mighty processing unit and to avoid time-consuming algorithms. In the same manner as SCOOT-R, RT-MAPS [9], [10], is divided into several components and date sensor data with an accurate clock as well. In that particular case, the aim of the dating is to use RT-MAPS like a numeric videotape recorder. Note that this date is the reception date of the sensor data. In a first time, it can record all data in a synchronized dated database. Then, it's possible to replay all these data later. The asset of this function is that it permits to improve an algorithm by testing it on the same databank or to compare several algorithms. Now, having compared several architectures for embedded applications, let's analyse the main functions of our architecture before discussing latency problem.

III. AROCCAM

In our architecture called AROCCAM, the acronym for "Architecture d'ordonnancement de capteurs pour la création d'algorithmes modulaires", we pointed up the modular and simple aspect. As we want an easy to use architecture, AROCCAM is only divided into three components (Figure 1).

- **A driver module** is responsible for the communication with external entities like sensors, softwares, computers,... There is a particular driver module for a particular communication bus (IEEE, CAN network, Ethernet network, Serial ports,...).
- **A brik module** is an application algorithm. Thanks to a subscription to driver modules, a brik module receives directly sensor data without having to know the communication protocol. In general, the user has to write piece of software only in this area.
- **The heart module** is the final component. This component has not to be modified or adapted by the user. It is responsible for the communication between driver modules and brik modules, the threads creation, memory management, ...

Moreover, like RT-MAPS, our architecture dates accurately each sensor data gathered. This permits to the user to replay all the recorded sensor data at the desired speed. In order to replay exactly the sensor data in the same way

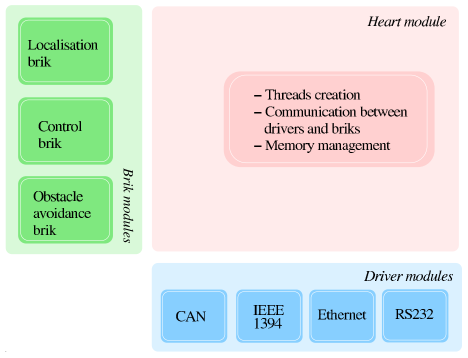


Fig. 1. AROCCAM Software Architecture.

that they were recorded (order of sensor data reception and interval time between two sensor data), the date of each sensor data corresponds to the reception date of the data by AROCCAM during recording.

After having described components involved in our architecture and their objectives, let's now analyse in details the problem of delayed observations fusion.

IV. A FUSION METHODOLOGY FOR DELAYED OBSERVATIONS

A. Presentation

The real-time aspect of embedded architectures is a feature pointed up. This feature consists in giving the sensor data to the algorithm as soon as it is available. However, as it was suggested by [11], most of the sensors have a latency time. Such architectures remain real-time sensor data collecting softwares but are not real-time environment perception softwares.

As sensor latency time can't be suppressed, it's therefore impossible to realize the real-time architecture last proposed. It means that the user must take these latenesses into account. In [12], a solution is suggested to the user. In this paper, we propose an elegant manner to manage this problem without making the development of an application algorithm more complex.

In the next section, is described in details the kind of latency periods that can appear.

B. Observation and latency period

The aim of the software that must be implemented in our architecture is to realize a particular task. In this section, we take the example of an accurate real-time positioning of a car on a digital map [13]. In this case, the vehicle is equipped with a video camera, oriented in the direction of the road, aiming at detecting the road. Then this detection permits to locate the vehicle thanks to a digital map listing road configurations.

An information that permits to participate to the achievement of the software task is called here **an observation**. It can be a sensor data or the processing result of this data. These observations are fusioned in the software. Let's analyse in details the process to obtain an observation (Figure 2) in our example.

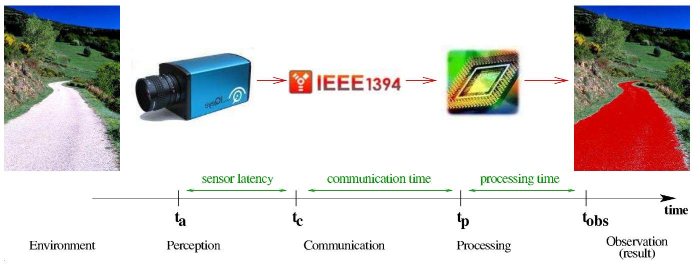


Fig. 2. The latency periods for obtaining an observation.

In general, the process to obtain an observation can be divided into three steps, each of them requiring a process time:

- *perception*. The sensor captures at time t_a a perception of the environment. Modern systems use smart sensors, i.e. an analog sensor linked to a local controller. The local controller gets the analog information, performs the analog to digital conversion, and sends the results through a digital bus like RS232 or CAN or Firewire. In the following of this paper, the word sensor will refer to a smart sensor. The time required by the local controller to prepare the result corresponds to the sensor latency.
- *communication*. As explained just above, it consists in sending the result through a digital bus at time t_c.
- *processing*. At time t_p, the embedded architecture, like AROCCAM, receives the sensor data. As our architecture works in real-time, we consider that the sensor data is dated as soon it's available by the computer. However, this information is not directly useable for the algorithm task. This last one has to process these data to extract a worth observation. The time required to treat the sensor data is the processing time.

In all embedded architectures, dating sensor data accurately, consists in timestamping those data with the date t_p like we do. However, the application algorithm can only fuse observations with the same date: the perception date. Sometimes, the data sheet of the sensor or directly the sensor during the experiment provides the sensor latency. When there is no information, a temporal calibration of sensors has to be done. Unfortunately, it's not always possible to estimate accurately all these parameters. We suggest to timestamp the observation with the date t_a and include with this date the dating precision.

C. Consequences of delayed observations fusion

To illustrate the fusion of delayed observations, let's keep our example of vehicle localization. For vehicle localization task, the estimated position is valuable only at a specific time. The robotic community calls this characteristic, the spatio-temporal localization. It means that the estimated vehicle position is function of time. For each result produced by those systems, a time imprecision induced a spatial imprecision. A spatial time induced error of $10cm$ is given by an imprecision of $1.5ms$ at $250km/h$ or $15ms$ at $25km/h$. To build an accurate localization system, it's necessary to keep in mind the observations latency times.

D. Fusion of delayed observations

The aim of the system is to compute a result: the state vector, and this vector is function of time. In the previous section, the latency periods for obtaining an observation have been detailed. The problem here, is the fusion of a delayed observation with a particular state vector. It means that an observation received by system has to be used to update a particular state vector: the one that describes the system at observation's date. To realize this task, past state vectors are stored as well as all observations permitting those estimations.

The fusion principle is the following:

- Each new observation received at t_{obs} is stored and sorted chronologically with this perception date t_a by substracting all the latency times from date t_{obs}. (Fig 3),

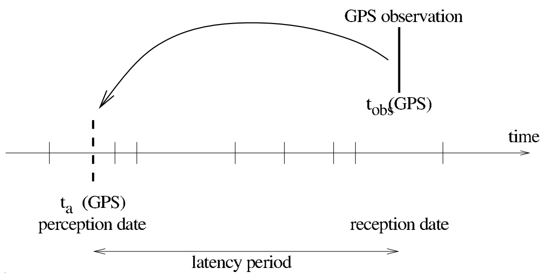


Fig. 3. Observations sorting.

- Then the state vector previous to this new observation (before date $t_a(GPS)$, i.e. $t_a(camera)$) is used to predict the state vector at the inserted observation date (i.e. $t_a(GPS)$) (Fig 4),

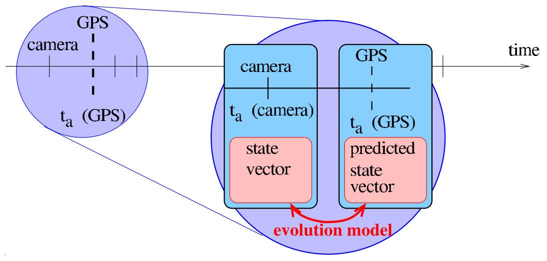


Fig. 4. State vector prediction.

- The inserted observation is used to update the predicted state vector (Fig 5),

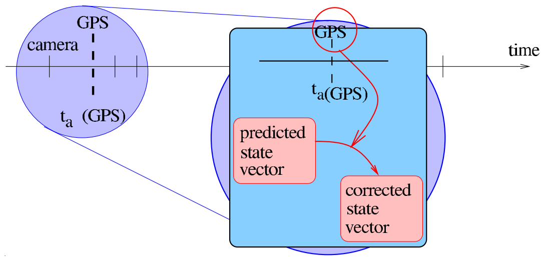


Fig. 5. State vector predicted updating.

- This updating method must be applied from observation date until the end of the list (Fig 6).

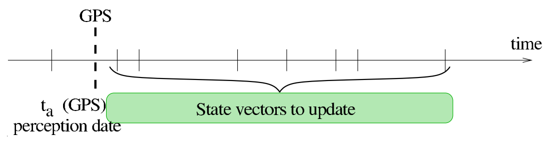


Fig. 6. State vectors to update.

The objective of the system is to offer an optimal estimation of the state vector at any time.

V. Implementation

In this section, we explain how to create a brik, emphasising the fusion mechanism of AROCCAM.

A. From sensors to observations

1) The subscription procedure: The first step in the design of a brik is to choose sensors to process in the algorithm. AROCCAM proposes to the user a subscription procedure to simplify this step. It consists in specifying useable sensors and the appropriate module for gathering sensor data. During this subscription procedure, the user must also indicate the processing method to apply to each sensor. These processing methods are used to produce observations.

2) The buffer structure: The observations, produced by the processing methods, don't constitute the solution of the algorithm. The aim of the buffer structure presented here is to collect the result of the algorithm (the state vector) at a particular moment. This structure can also be used to stock temporary results in order to compute the next buffer structure at the next observation reception.

3) Mechanism: Figure 7 summarizes the observation creation process. First, a driver module gathers a raw sensor data at time t_p and send it to the appropriate function of the brik. Then, an observation is created and timestamped with date t_a by subtracting all the latency periods to the date t_p.

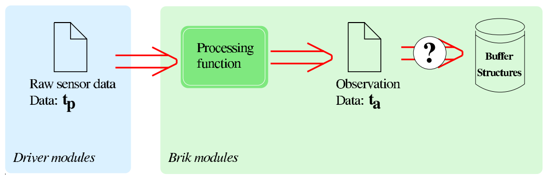
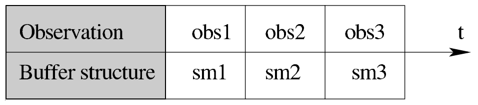


Fig. 7. Synopsis of the creation of an observation.

The next step consists in producing the result, it means use the new created observation to produce or update a buffer structure.

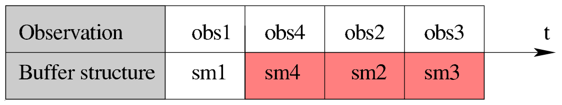
B. From observations to buffer structures

1) The chronological list: AROCCAM makes available a chronological list to the user. This list identifies all the observations ordered by date t_a. A buffer structure is associated to each observation (Figure 8(a)). As mentioned before, a buffer structure stores the application's result at a given date. The aim of this list is to keep each buffer structure updated. Unfortunately, an observation does not always insert at the end of the list because of latency periods. It means that the insertion of an observation in the middle of the list has to generate the updating of all the buffer structures stored from the date of the observation (Figure 8(b)).



Observation	obs1	obs2	obs3
Buffer structure	sm1	sm2	sm3

(a) The chronological list with three observations and their associated buffer structure



Observation	obs1	obs4	obs2	obs3
Buffer structure	sm1	🟥 sm4	🟥 sm2	🟥 sm3

(b) The list with the new inserted observation 4. Three buffer structures become outdated.

Fig. 8. The chronological list.

The updating of the list is a fastidious task for the user. AROCCAM proposes not only a chronological list but also an automatic updating method.

2) The subscription procedure: A brik has to make a subscription to produce observations. It has also to make a subscription to enable the automatic updating of the list. It consists in specifying for each kind of observation, the appropriate method to be used to update the associated buffer structure. In the following of this paper, those functions will be call updating functions.

3) The automatic list updating: When an observation has to be inserted in the list, AROCCAM manages its introduction at the good location. It means at the date of the observation. Then it calls the updating function of the inserted observation defined in the subscription procedure. This last function takes 3 parameters: the inserted observation, the buffer structure to update and the previous buffer structure (chronologically):

- updating_function (obs4, **sm4**, sm1)

In addition, it is also necessary to update all the buffer structures whose dates are more recent than the last inserted one. This is done by calling successively the updating functions of other observations:

- updating_function (obs2, **sm2**, sm4)
- updating_function (obs3, **sm3**, sm2)

The main assets of this process is that it permits to the user to concentrate on processing functions and updating functions without worrying about latency periods. This task is devoted to AROCCAM. The most recent state matrix contains the optimal result of the application.

VI. Experimentation and Results

The following section describes experimentations that were carried out to validate our approach. Thus, an outdoor localization system was chosen.

A. Description

A terrestrial robot was used for the experiments. This research platform, of the Research Federation TIMS (Technologie de l'information, de la mobilité et de la sûreté), (Figure 9), was initially equipped with an on-board PC running Linux RTAI. Several sensors was added to it for the realization of the system.

The objectives of this system is to locate the vehicle with $2m$ accuracy using only low-cost sensors.

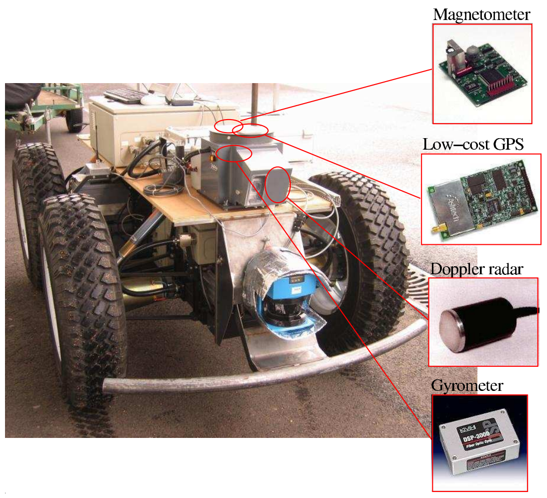


Fig. 9. The "RobucarTT" with its sensors

1) Sensors: Sensors used in this system are presented in Table I. The GPS system and the magnetometer permits to initialize the system: vehicle position and orientation. Then, two proprioceptive sensors: a Doppler radar and a gyrometer, and the GPS are used to locate the vehicle. A particle filter is employed to estimate the vehicle position.

TABLE I
PROBABILITY VALUES FOR PERCEPTIVE TRIPLETS

sensor	latency period	acquisition frequency
magnetometer	$5ms$	$16Hz$
low-cost GPS	$6-100ms^{\star}$	$10Hz$
Doppler radar	-	$10-77Hz$
gyrometer	-	$20Hz$

*: supplied directly by the sensor.

Table I lists not only sensors but also their latency period and their acquisition frequency. These latency periods take into account the latency period of the sensor and also the communication time on the bus that links the sensor to the computer. In [14], the author suggests a method to estimate accurately this communication time and in [15], a temporal calibration of a ultrasound sensor is proposed. These methods can be used to made a temporal calibration of sensors used by AROCCAM. However, the latency period of a sensor still remains an estimation. To take into account the timestamping error of the observation, we convert this error into data inaccuracy like it is proposed in [16]. We can also notice that all these sensors are not synchronized since they are affected by different latency period and different acquisition frequency. The use of AROCCAM seems to be necessary.

2) Vehicle progress model: In this part, we focus on small displacements. Assuming the flatness of the environment where the robot is running, position and attitude of the vehicle are resumed to the position of the vehicle in the 2D plane (Oxy) defined by $x(t)$ and $y(t)$ and orientation of the car with respect to the (Ox) axis given by $\theta(t)$. Measurements from the vehicle are the average speed $V(t)$ given by the Doppler radar and angular speed $\omega(t)$ given by the gyrometer.

Figure 10 shows the vehicle state (position and orientation) at two particular moments: t_k and t_{k+1}.

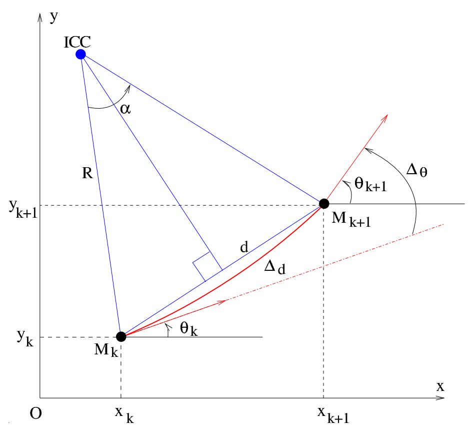


Fig. 10. Small displacement between two successive positions.

Relation between vehicle displacement Δ_d and average speed V is given by: $\Delta_d \approx d = Te \cdot V$ where Te is the sampling period. In the same way, relation between vehicle rotation Δ_θ and angular speed ω is given by: $\Delta_\theta = Te \cdot \omega$.

Thus, the vehicle progress model is:

$$\left\{\begin{array}{lcl} x_{k+1} & = & x_k + \Delta_d \cdot cos(\theta_k + \Delta_\theta/2) \\ y_{k+1} & = & y_k + \Delta_d \cdot sin(\theta_k + \Delta_\theta/2) \\ \theta_{k+1} & = & \theta_k + \Delta_\theta \end{array}\right. \quad (1)$$

3) Why use AROCCAM: It's necessary to use the AROCCAM architecture in this system since:

- **sensors are unsynchronized:** equation (1) supposed that the two proprioceptive sensors supply synchronized observations.
- **sensors have a latency period.**

An elegant solution to solve these difficulties is to use the architecture suggested in this paper.

B. Experimentation results

The scenario used to validate our architecture is presented in figure 11. As we can see, the trajectory is complex and contains several curves. This path have been obtained by a manual run. During the experiment, successive absolute positions giving by a GPS RTK (THALES Navigation) with $2cm$ accuracy, are recorded. The position estimated by our method is compared each time with the GPS-RTK reference trajectory by computing the difference (error) between them.

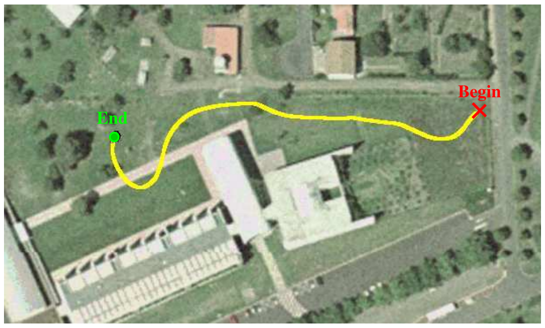


Fig. 11. Trajectory realized in the experiment.

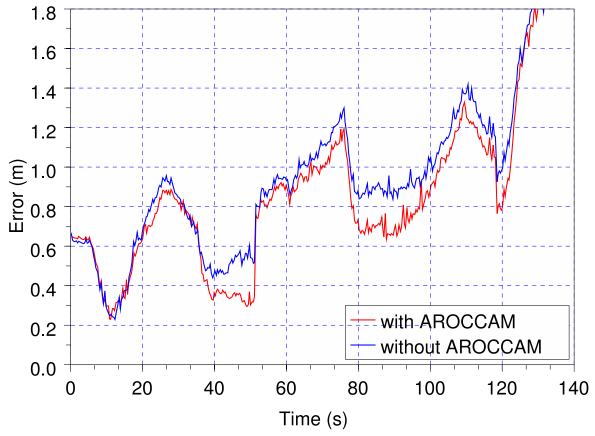


Fig. 12. Localization error with two architectures.

On the following figure (figure 12), the localization error is depicted for two experimentations:

- In red line: localization error with the use of AROCCAM, taking into account the latency periods.
- In blue line: localization error with a classical embedded architecture.

We can notice that, as expected, latency periods affect the localization system results (for instance, the fuse of a GPS position referring a past vehicle postion with the actual vehicle position). In our example, a maximal error of $20cm$ is added to the system when a classical architecture is used. These results permit to show the benefit given by our approach.

VII. Conclusion

This paper has proposed an embedded architecture for the fusion of delayed observations. A fusion methodology has first been designed. First a description of the latency periods from the sensor to the data fused in the algorithm has been presented. The consequences of delayed observations fusion have been explained though the example of the vehicle localization and a method has been suggested. Then the AROCCAM architecture has been presented focusing on the features that permit to fuse delayed observations. Finally, a vehicle localization application has been proposed to illustrate our architecture.

Even though the AROCCAM architecture allows to reduce errors in fusion of unsynchronized information, our aim here is to draw conclusions not from the demonstration but from the process of building embedded applications. We think that the AROCCAM architecture offers an elegant and easy manner to build fusion algorithm. The benefit was really substantial: during all the programming and debugging process, we never had problem with thread communication, real-time multithread management, transmission of data between different system,. . . All these aspects were dealt with by our software architecture.

Moreover several other applications were developed under AROCCAM with no difficulties and good results.

References

[1] Gianluca Ippoliti, Leopoldo Jetto, Alessia La Manna, and Sauro Longhi. Improving the robustness properties of robot localization procedures with respect to environment features uncertainties. In *International Conference on Robotics and Automation (ICRA)*, Barcelona, Spain, April 2005.

[2] C. Kwok, D. Fox, and M. Meila. Real-time particle filters. *IEEE, Sequential State Estimation*, 92(2), 2004.

[3] David Filliat. *Cartographie et estimation globale de la position pour un robot mobile autonome*. PhD thesis, LIP6/AnimatLab, Université Pierre et Marie Curie, Paris, France, December 2001. Spécialité Informatique.

[4] Marc-Michael Meinecke and Marian-Andrzej Obojski. Potentials and limitations of pre-crash systems for pedestrian protection. In *International Workshop on Intelligent Transportation*, Hamburg/Germany, March 15-16 2005.

[5] Rachid Alami, Raja Chatila, Sara Fleury, Matthieu Herrb, Felix Ingrand, Maher Khatib, Benoit Morisset, Philippe Moutarlier, and Thierry Siméon. Around the lab in 40 days... In *IEEE International Conference on Robotics and Automation*, San Francisco, USA, 2000.

[6] Khaled Chaaban, Paul Crubillé, and Mohamed Shawky. *Computer Science*, chapter Real-Time Framework for Distributed Embedded Systems, pages 96–107. Springer-Verlag GmbH, 2004.

[7] Khaled Chaaban, Paul Crubillé, and Mohamed Shawky. Real-time embedded architecture for intelligents vehicles. In *Proceeding of the 5th Real-time Linux workshop*, Valencia, Spain, November 2003.

[8] P. Jeong and S. Nedevschi. Efficient and robust classification method using combined feature vector for lane detection. *Ieee transactions on circuits and systems for video technology*, 15(4):528–537, 2005.

[9] Iyad Abuhadrous, Fawzi Nashashibi, and Claude Laurgeau. Multi-sensor fusion (gps, imu, odometers) for precise land vehicle localisation using rtmaps. In *11th International Conference on Advanced Robotics ICAR*, 2003.

[10] Fawzi Nashashibi. Rtm@ps: a framework for prototyping automatic multisensor applications. In *IEEE Intelligent Vehicles Symposium*, October 3-5 2000.

[11] Mikael Kais, Laurent Bouraoui, Steeve Morin, Arnaud Porterie, and Michel Parent. A collaborative perception framework for intelligent transportation system applications. In *Intelligent Transportation Systems Conference ITSC*, Vienna, Austria, September 13-16 2005.

[12] Iyad Abuhadrous. *Systéme embarqué temps réel de localisation et de modélisation 3D par fusion multi-capteur*. PhD thesis, Ecole des Mines de Paris, january 2005.

[13] Jean Laneurit, Roland Chapuis, and Frédéric Chausse. Accurate vehicle positioning on a numerical map. *International Journal of Control, Automation, and Systems*, 3(1):15–31, March 2005.

[14] Olivier Bezet. *Etude de la qualité temporelle des données dans un système distribué pour la fusion multi-capteurs*. PhD thesis, Université de Technologie Compiègne, november 2005.

[15] Mark J. Gooding, Stephen H. Kennedy, and J. Alison Noble. Temporal calibration of freehand three-dimensional ultrasound using image alignment. *Ultrasound in Medicine & Biology*, 31(7):919–927, July 2005.

[16] Olivier Bezet and Véronique Cherfaoui. Influence of timestamping error on data inaccuracy. In *Proceedings of the Eighth International Conference of Information Fusion*, Philadelphia, PA, USA, July 2005.